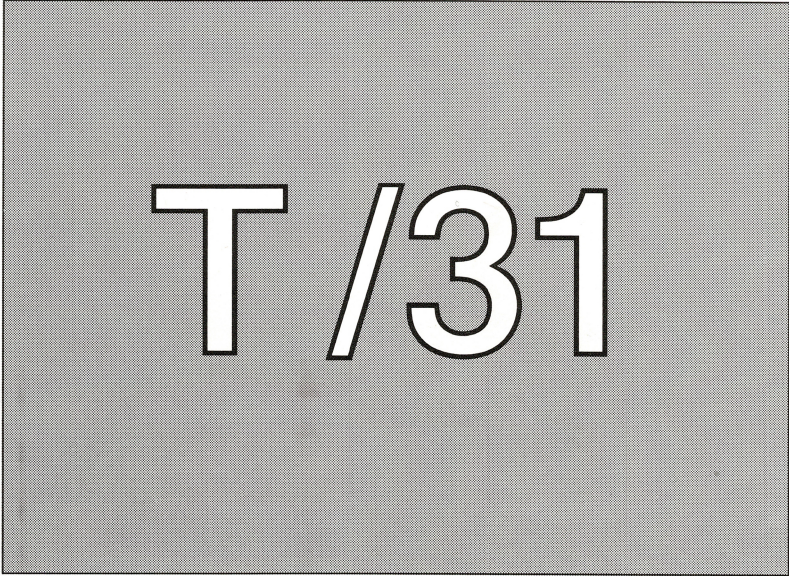


Targon® /31



T /31

User's Guide

NIXDORF
COMPUTER

© Nixdorf Computer AG 1989
Printed in W.-Germany 05.1990
Technische Änderungen vorbehalten

Systemliteratur

Targon® /31

Systemliteratur

Targon® /31

Systemliteratur

Targon® /31

Bitte abtrennen und in die Tasche im Handbchrücken einstecken.

Einführung in das Betriebssystem UNIX

Grundlagen für UNIX-Anwender

Benutzung des Dateisystems

Übersicht über die Anleitungen

Anleitungen zum Zeileneditor

Anleitung zum Bildschirmeditor (vi)

Anleitung zur Shell

Anleitung zur Kommunikation

1

2

3

4

5

6

7

8

Übersicht über das Dateisystem

Übersicht über UNIX-Systemkommandos

Kurzübersicht über ed-Kommandos

Kurzübersicht über Kommandos des Editors vi

Übersicht über die Shell-Kommandosprache

Konfiguration des Terminals

Glossar

A

B

C

D

E

F

G

Organisationsblatt

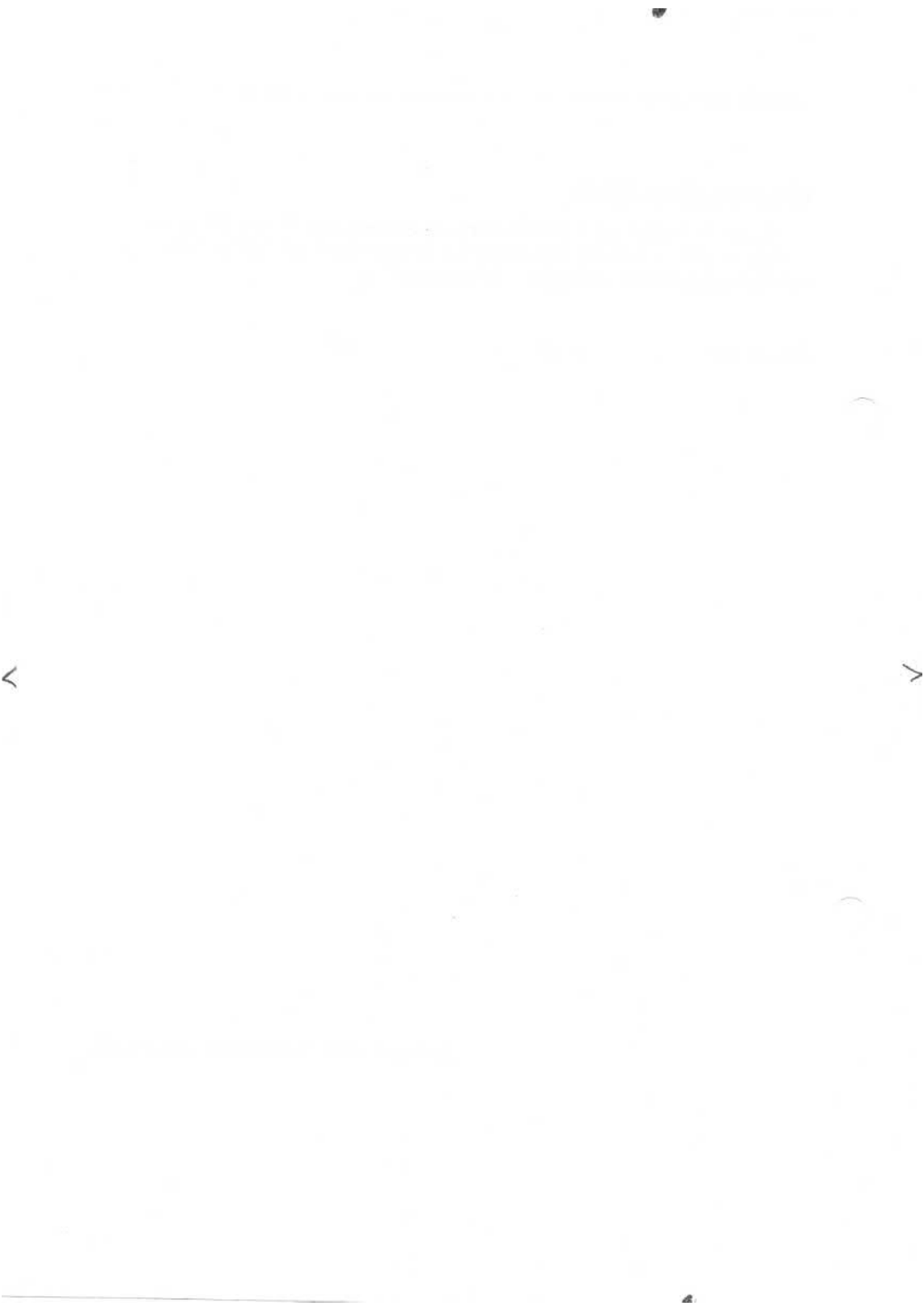
Dieses Blatt gibt eine Übersicht über alle Änderungen, die seit der ersten Auflage an diesem Handbuch durchgeführt wurden. Es wird bei jeder Änderungsmitteilung mitgeliefert und ist jeweils auszutauschen.

Erstauflage:

1.5.89

Rel. 4

Dieses Handbuch wurde mit troff erstellt.



Inhaltsverzeichnis

Vorwort ix

Teil 1: UNIX-Systemüberblick

Kapitel 1: Einführung in das Betriebssystem UNIX

Komponenten des Betriebssystems UNIX 1-1

Funktionen des Betriebssystems UNIX 1-3

Kapitel 2: Grundlagen für UNIX-Anwender

Einführung 2-1

Das Terminal 2-2

Zuweisung eines Benutzernamens 2-13

Eröffnen einer UNIX-Sitzung 2-14

Teil 2: UNIX-Systemanleitungen

Kapitel 3: Benutzung des Dateisystems

Einführung 3-1

Struktur des Dateisystems 3-2

Die Position des Benutzers im Dateisystem 3-4

Organisation eines Verzeichnisses 3-17

Zugriff auf Dateien und ihre Bearbeitung 3-32

Zusammenfassung	3-78
Kapitel 4: Übersicht über die Anleitungen	
Einführung	4-1
Textbearbeitung	4-2
Die Shell	4-7
Elektronische Kommunikation	4-12
Programmieren im System	4-14
Kapitel 5: Anleitungen zum Zeileneditor (ed)	
Einführung in den Zeileneditor	5-1
Hinweise zur Verwendung dieser Anleitung	5-2
Einführung	5-3
Übung 1	5-15
Allgemeines Format der Kommandos des Editors ed	5-16
Adressierung von Zeilen	5-17
Übung 2	5-32
Text einer Datei anzeigen lassen	5-33
Text erstellen	5-36
Übung 3	5-44
Text löschen	5-46
Text ersetzen	5-52
Übung 4	5-61
Sonderzeichen	5-63

Übung 5	5-74
Text verschieben/kopieren	5-76
Übung 6	5-85
Weitere nützliche Kommandos und Hinweise	5-86
Übung 7	5-96
Lösungen zu den Übungen	5-97
Kapitel 6: Anleitung zum Bildschirmeditor (vi)	
Einführung	6-1
Vorbereitungen	6-4
Anlegen einer Datei	6-7
Editieren von Text: Der Kommando-Modus	6-10
Verlassen von vi	6-18
Übung 1	6-21
Steuern des Cursors auf dem Bildschirm	6-22
Positionieren des Cursors in nicht angezeigtem Text	6-41
Übung 2	6-54
Eingeben von Text	6-56
Übung 3	6-61
Löschen von Text	6-62
Übung 4	6-69
Ändern von Text	6-70
Elektronisches Verschieben von Text	6-78

Übung 5	6-83
Spezielle Kommandos	6-84
Verwendung von Kommandos des Zeileneditors in vi	6-87
Verlassen von vi	6-94
Sonderoptionen für vi	6-97
Übung 6	6-100
Lösungen zu den Übungen	6-101
Kapitel 7: Anleitung zur Shell	
Einführung	7-1
Die Shell-Kommandosprache	7-2
Übungen zur Kommandosprache	7-39
Shell-Programmierung	7-40
Anpassen der Benutzerumgebung	7-97
Übungen zur Shell-Programmierung	7-103
Lösungen zu den Übungen	7-105
Kapitel 8: Anleitung zur Kommunikation	
Einführung	8-1
Austauschen von Nachrichten	8-2
mail	8-3
mailx	8-16
Übersicht über mailx	8-17
Optionen in der Kommandozeile	8-19

Senden von Nachrichten: die Tilde-Escape-Sequenzen	8-20
Verwaltung eingegangener Post	8-32
Die Datei .mailrc	8-42
Senden und Empfangen von Dateien	8-47
Vernetzung	8-68

Anhänge, Glossar

Anhang A: Übersicht über das Dateisystem

Anhang B: Übersicht über UNIX-System-
kommandos

Anhang C: Kurzübersicht über **ed**-Kommandos

Anhang D: Kurzübersicht über Kommandos des
Editors **vi**

Anhang E: Übersicht über die Shell-Kommando-
sprache

Anhang F: Konfiguration des Terminals

Glossar

Vorwort

Die Themen dieses Handbuchs sind in zwei Hauptgruppen unterteilt: ein Überblick über das UNIX-Betriebssystem und die Tutorials der wichtigsten Werkzeuge unter UNIX. Es folgt eine kurze Beschreibung dieser Kapitel. Der letzte Abschnitt dieses Vorworts, "Notationsvereinbarungen," beschreibt die typographische Notation, die in allen Kapiteln dieses Handbuchs benutzt wurde. Sie werden diesen Abschnitt von Zeit zu Zeit zu Rate ziehen, wenn Sie dieses Handbuch lesen.

System-Überblick

Dieser Teil des Handbuchs besteht aus 3 Abschnitten, die Sie in das zugrundeliegende Prinzip des UNIX-Betriebssystem einführen. Jedes Kapitel liefert Informationen, die in den nachfolgenden Kapiteln benötigt werden. Daher ist es wichtig, diese Kapitel in Reihenfolge zu lesen.

- Kapitel 1, "Einführung in das Betriebssystem UNIX", gibt einen Überblick über das Betriebssystem.
- Kapitel 2, "Grundlagen für Anwender von UNIX", beschreibt die allgemeinen Regeln und Richtlinien beim Umgang mit dem UNIX-System. Es enthält wichtige Informationen über den Gebrauch von Terminals, wie man eine Berechtigung für das System erhält und wie man auf ein UNIX-System Zugriff erhält.
- Kapitel 3, "Benutzung des Dateisystems", gibt einen Überblick über das Arbeiten mit dem Dateisystem. Hier werden Sie über Kommandos informiert, mit denen Sie sich Ihre eigene Verzeichnis-Struktur erstellen können, wie Sie auf Unterverzeichnisse zugreifen und diese manipulieren können, wie Sie Dateien darin organisieren können und wie Sie andere Verzeichnisse, auf die Sie Zugriff haben, inhaltlich untersuchen können.

Anleitungen zur Benutzung des UNIX-Systems

Der zweite Teil des Handbuchs enthält Tutorials mit folgendem Inhalt: die Texteditoren `ed` und `vi`, die Shell-Kommando- und Programmiersprache und die elektronischen Kommunikations-Werkzeuge. Zum tieferen Verständnis des Stoffs sollten Sie die Beispiele und Übungen durcharbeiten, wenn Sie die Tutorials lesen. In den Tutorials wird davon ausgegangen, daß Sie das in Kapitel 1-3 vorgestellte Konzept verstanden haben.

- Kapitel 4, "Übersicht über die Anleitungen", ist eine Einführung der folgenden vier Kapitel, den Tutorials im zweiten Teil des Handbuchs. Es hebt die Möglichkeiten des UNIX-Systems hervor, wie den Kommandomodus, das Editieren von Text, elektronische Kommunikation, die Programmierung und Hilfen zur Software-Entwicklung.
- Kapitel 5, "Anleitung zum Zeileneditor (ed)", unterrichtet Sie, wie Sie den Texteditor ed benutzen, um an einem Terminal oder einem Terminal-Drucker Textdateien zu erstellen oder zu modifizieren.
- Kapitel 6, "Anleitung zum Bildschirmeditor vi", beschreibt die Handhabung des Texteditors vi, mit dem Sie an einem Terminal Texte erstellen oder modifizieren können.

NOTE

Der Editor, vi, basiert auf der Software, die entwickelt wurde von der University of California, Berkeley, California; Computer Science Division, Department of Electrical Engineering and Computer Science. Alle Rechte und Lizenzen werden von der Regents of the University of California vergeben.

- Kapitel 7, "Anleitung zur Shell", unterrichtet Sie über den Gebrauch der Shell als Kommandointerpreter und als Programmiersprache, um Shell-Programme zu erstellen.
- Kapitel 8, "Anleitung zu Kommunikation", zeigt Ihnen, wie Sie Nachrichten und Dateien an Benutzer des eigenen UNIX-Systems und auch zu anderen UNIX-Systemen verschicken können.

Zusammenhängende Informationen

Sechs Anhänge und ein Glossar über die UNIX-System-Terminologie werden zusätzlich zur Übersicht bereitgestellt.

- Anhang A, "Übersicht über das Dateisystem", illustriert, wie Informationen im UNIX-Betriebssystem gespeichert werden.
- Anhang B, "Übersicht über UNIX-Systemkommandos", beschreibt, in alphabetischer Ordnung, alle UNIX-Systemkommandos, die im Handbuch benutzt wurden.

- Anhang C, "Kurzübersicht über ed-Kommandos", ist eine Kurzanweisung für den Zeileneditor `ed`. (Zur näheren Information sollten Sie das Kapitel 5, "Anleitung zum Zeileneditor `ed`" benutzen.) Die Kommandos werden in der Reihenfolge beschrieben, wie sie im Kapitel 5 behandelt wurden.
- Anhang D, "Kurzübersicht über vi-Kommandos", ist eine Übersicht über den Bildschirmeditor `vi`, der im Kapitel 6 "Anleitung zum Bildschirmeditor `vi`" besprochen wird. Die Kommandos werden in der Reihenfolge von Kapitel 6 besprochen.
- Anhang E, "Übersicht über die Shell-Kommandosprache", ist eine Zusammenfassung der Shell-Kommandosprache, Notation und Programm-Konstruktionen, wie sie im Kapitel 7, "Anleitung zur Shell", beschrieben wurden.
- Anhang F, "Konfiguration des Terminals", zeigt Ihnen, wie Sie Ihr Terminal konfigurieren müssen, um es am UNIX-System benutzen zu können.
- Das Glossar definiert die zum UNIX-System gehörende Terminologie, die in diesem Buch benutzt wurde.

Notationsvereinbarungen

Die folgenden Notationsvereinbarungen werden in diesem Handbuch benutzt.

fett	Benutzer-Eingaben, wie zum Beispiel Kommandos, Kommando-Optionen und -Argumente, Variablen und Namen von Verzeichnissen und Dateien werden fett geschrieben.
<i>kursiv</i>	Variablenamen, denen noch Werte zugewiesen werden müssen (wie zum Beispiel <i>Paßwort</i>) werden <i>kursiv</i> dargestellt.
konstante Abstände	Die Ausgaben von UNIX, wie zum Beispiel das Bereit-Zeichen und die Eingabe-Aufforderung zu einem Kommando werden mit konstanten Abständen dargestellt.
< >	Die Eingabe dieser Zeichen wird nicht auf dem Bildschirm dargestellt, wie zum Beispiel Paßworte, Tabulatoren oder das RETURN-Zeichen. Diese Eingaben werden in Kleiner-/Größer-Zeichen eingeschlossen.
<^Zeichen>	Kontrollzeichen werden zwischen Kleiner-/Größer-Zeichen dargestellt, da sie bei der Eingabe nicht angezeigt werden. Das Zirkumflex-Zeichen (^) stellt die Kontroll-Taste dar (sie ist normalerweise mit CTRL beschriftet). Um ein Kontroll-Zeichen einzugeben, drücken Sie die Kontroll-Taste und gleichzeitig das mit <i>Zeichen</i> angegebene Zeichen. Um zum Beispiel <^d> einzugeben, drücken Sie die Kontroll-Taste und gleichzeitig die Taste d; d erscheint dabei nicht auf dem Bildschirm.
[]	Optionale Kommando-Optionen und -Argumente, wie zum Beispiel [-msC], werden in eckigen Klammern angegeben.
	Das Pipe-Zeichen () trennt optionale Angaben, von denen eine angegeben werden muß. Zum Beispiel beim Kommando mit folgendem Format: <i>Kommando [arg1 arg2]</i>

	Wenn Sie das Kommando benutzen müssen Sie entweder <i>arg1</i> oder <i>arg2</i> mit dem Kommando angeben.
...	Auslassungszeichen nach einem Argument bedeuten, daß mehrere Argumente angegeben werden können.
	Pfeile am Bildschirm (wie in den Beispielen des Kapitel 6) stellen den Cursor dar.
<i>Kommando(Nummer)</i>	Ein Kommando, dem eine Nummer in runden Klammern folgt, deutet auf den Teil der UNIX-System Reference Manuals hin, in dem dieses Kommando beschrieben wird. (Es gibt drei Reference Manuals: das <i>User's Reference Manual</i> , <i>Programmer's Reference Manual</i> und das <i>System Administrator's Reference Manual</i> .) Zum Beispiel weist die Notation <i>cat(1)</i> auf den Abschnitt 1 (des <i>User's Reference Manual</i>) hin, in dem das <i>cat</i> -Kommando beschrieben wird.

In einfachen Kommandos wird das *\$*-Zeichen zur Darstellung des Shell-Bereit-Zeichens benutzt. Dies gilt nicht für alle Systeme. Welches Zeichen auch immer von Ihrem System benutzt wird, Sie müssen wissen, daß dieses Zeichen vom System ausgegeben wird; obwohl das Bereit-Zeichen manchmal am Anfang einer Kommando-Zeile abgedruckt ist, so wie es auch an Ihrem Bildschirm erscheint, so sollten Sie es doch nicht nochmals eingeben. (Das *\$*-Zeichen wird ebenfalls dazu benutzt, um auf den Wert von Positionsparametern und benannten Variablen zuzugreifen; näheres siehe Kapitel 7.)

In allen Kapiteln werden vollständige oder auch ausschnittsweise Bildschirme zur Darstellung von Beispielen benutzt, um zu zeigen, wie die Aus-/Eingabe auf dem Bildschirm aussieht, wenn Sie mit dem UNIX-System arbeiten. Die Beispiele zeigen, wie Sie den UNIX-System-Editor benutzen, wie Sie kleine Programme schreiben und Kommandos ausführen. Die Eingabe (Zeichen, die von Ihnen eingegeben werden) und Ausgabe (Zeichen, die vom UNIX-System ausgegeben werden) wird in den oben angegebenen Konventionen angezeigt. Die Beispiele sind unabhängig vom Terminal, das Sie benutzen.

Die Kommandos der Abschnitte eines Kapitels werden am Ende eines Abschnitts noch einmal angesprochen. Eine Zusammenfassung der vi-Kommandos, aufgelistet nach ihrer Wichtigkeit, finden Sie im Anhand D. Am Ende einiger Abschnitte werden Übungen bereitgestellt, damit Sie noch einmal mit den Kommandos üben können. Die Lösungen der Übungen in den Kapiteln werden am Ende des Kapitels angegeben.

NOTE

Der Text des *User's Guide* wurde mit dem UNIX-System-Texteditor *Rand* bearbeitet, der im Handbuch *Rand-Editor* beschrieben wird, und mit der DOCUMENTER'S WORKBENCH Software: **troff**, **tbl**, **pic** und den **mm**-Macros, formatiert.

Kapitel 1: Einführung in das Betriebssystem UNIX

Komponenten des Betriebssystems UNIX	1-1
Funktionen des Betriebssystems UNIX	1-3
Der Kern	1-4
Das Dateisystem	1-5
Normale Dateien	1-5
Verzeichnisse	1-6
Gerätedateien	1-6
Die Shell	1-9
Kommandos	1-9
Funktionen der Kommandos	1-10
Eingabe der Kommandos	1-11
Ausführung von Kommandos	1-13



Komponenten des Betriebssystems UNIX

Das Betriebssystem UNIX besteht aus einer Reihe von Programmen (bzw. Software), mit denen der Rechner gesteuert, die Verbindung zwischen dem Benutzer und dem Rechner hergestellt wird, und die Werkzeuge für die Arbeit des Benutzers darstellen. Es wurde im Hinblick darauf konzipiert, dem Benutzer eine unkomplizierte, effiziente und flexible Umgebung für die Datenverarbeitung zur Verfügung zu stellen. Das Betriebssystem UNIX bietet insbesondere folgende Vorzüge:

- Es ist ein System, das sich für die Ausführung einer großen Vielfalt von Jobs oder Anwendungen eignet.
- Es bietet eine Dialogumgebung, in der der Benutzer direkt mit dem Rechner kommunizieren kann und sofort Reaktionen auf seine Anforderungen und Eingaben erhält.
- Als Mehrplatzsystem bietet es die Möglichkeit, die Betriebsmittel des Rechners mit anderen Benutzern gemeinsam zu nutzen, ohne daß darunter die Produktivität leidet.

Dieses Verfahren wird als Timesharing bezeichnet. Das Betriebssystem verarbeitet die Anforderungen der Benutzer der Reihe nach, es wechselt jedoch so schnell von einem Benutzer zum anderen, daß es den Anschein hat, als ob es mit allen Benutzern gleichzeitig kommunizierte.

- Im Multi-Tasking-Betrieb können mehrere Programme gleichzeitig ausgeführt werden.

Das Betriebssystem UNIX gliedert sich in vier Hauptkomponenten:

Kern	Der Kern besteht aus einem Programm, das den tatsächlichen "Kern" des Betriebssystems darstellt; er koordiniert die internen Funktionen des Rechners (wie zum Beispiel die Zuordnung von Betriebsmitteln). Die Arbeit des Kerns bleibt für den Benutzer transparent; der Benutzer braucht nichts über ihn zu wissen, um seine Arbeit zu erledigen.
Dateisystem	Das Dateisystem dient zur Verwaltung und Nutzung der Datenbestände. Man kann damit auf einfache Art Informationen speichern und später wieder auf sie zugreifen.

Shell	Die Shell ist ein Programm, das als Kommandointerpreter dient. Sie stellt das Bindeglied zwischen dem Benutzer und dem Kern des Betriebssystems dar; sie interpretiert die Kommandos des Benutzers und führt sie aus. Man bezeichnet die Shell als interaktiv (bzw. dialogfähig), da sie Eingaben des Benutzers liest und Meldungen an ihn ausgibt.
Kommandos	Kommandos sind eigentlich Namen von Programmen, die man eingibt, damit der Rechner die Programme ausführt. Programmpakete werden Werkzeuge genannt. Das Betriebssystem UNIX enthält Werkzeuge für Aufgaben wie das Erstellen und Ändern von Texten, Schreiben von Programmen und Entwickeln von Softwarewerkzeugen sowie das Austauschen von Daten mit anderen Benutzern über den Rechner.

Funktionen des Betriebssystems UNIX

Abbildung 1-1 zeigt ein Modell des Betriebssystems UNIX. Jeder Kreis stellt dabei eine der Hauptkomponenten von UNIX dar: Kern, Shell und Benutzerprogramme bzw. Kommandos. Die Pfeile veranschaulichen die Rolle der Shell als Medium, durch das der Benutzer und der Kern miteinander kommunizieren. Im weiteren Verlauf dieses Kapitels werden die einzelnen Komponenten zusammen mit einer weiteren wichtigen Einrichtung des Betriebssystems UNIX, dem Dateisystem, beschrieben.

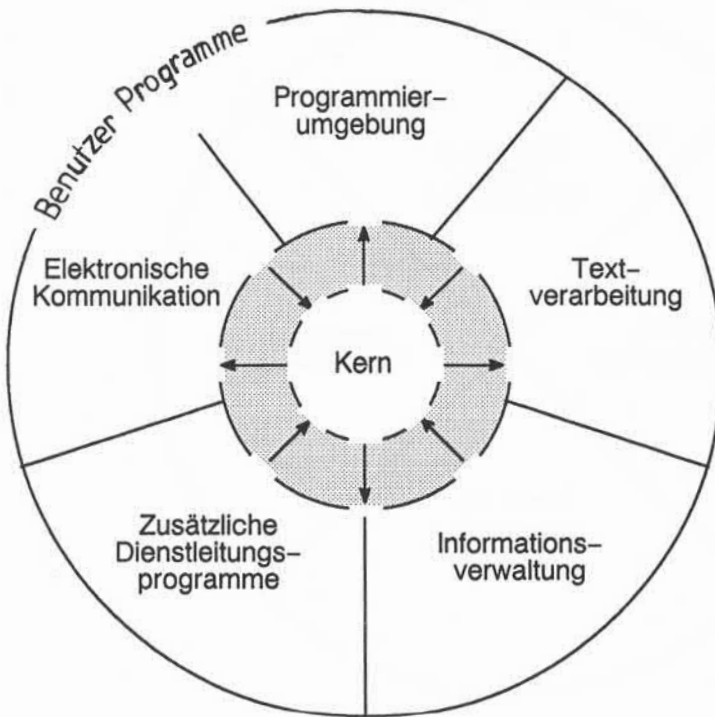


Abbildung 1-1: Modell des Betriebssystems UNIX

Der Kern

Der zentrale Bestandteil des Betriebssystems UNIX wird Kern (oder Kernel) genannt. Er steuert den Zugriff auf den Rechner, verwaltet den Arbeitsspeicher des Rechners und das Dateisystem und weist den Benutzern die Betriebsmittel des Rechners zu. Abbildung 1-2 veranschaulicht die Funktionen des Kerns.



Abbildung 1-2: Funktionen des Kerns

Das Dateisystem

Das Dateisystem bildet den Eckpfeiler des Betriebssystems UNIX. Mit ihm können Informationen in einer logischen Struktur organisiert, abgerufen und verwaltet werden. Das Dateisystem hat eine hierarchische Struktur; wenn es sichtbar wäre, würde es wie ein Struktogramm oder ein umgekehrter Baum aussehen (Abbildung 1-3).

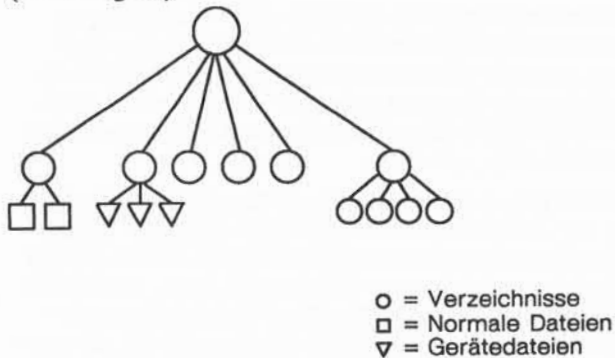


Abbildung 1-3: Hierarchische Struktur des Dateisystems

Die Grundeinheit des Betriebssystems UNIX ist eine Datei. Dabei gibt es drei verschiedene Typen: normale Dateien, Verzeichnisse ("Verzeichnisse") und Geräte Dateien (siehe Kapitel 3, "Benutzung des Dateisystems").

Normale Dateien

Eine normale Datei besteht aus einer Sammlung von Zeichen, die vom System als eine Einheit behandelt werden. Normale Dateien dienen zum Speichern von Informationen, die man aufbewahren will. Sie können aus Text für Briefe oder Berichte, aus Code für Programme oder aus Kommandos bestehen, mit denen man die eigenen Programme ausführt. Nachdem man eine Datei erstellt hat, kann man weitere Informationen in sie aufnehmen, Informationen aus ihr löschen, oder sie insgesamt löschen, wenn sie nicht mehr benötigt wird.

Verzeichnisse

Ein Verzeichnis ist eine Datei höherer Ordnung, in der eine Gruppe von zusammenhängenden Dateien zusammengefaßt wird. Beispielsweise können im Verzeichnis **Umsatz** die Dateien mit den monatlichen Umsatzdaten unter den Namen **jan, feb, mar** usw. gespeichert sein. Man kann jederzeit Verzeichnisse anlegen, Dateien in sie aufnehmen oder aus ihnen löschen oder die Verzeichnisse selbst löschen.

Alle von einem Benutzer angelegten Verzeichnisse, deren Eigentümer er auch ist, befinden sich in seinem Home-Verzeichnis. Dieses Verzeichnis wird dem Benutzer vom System zugeordnet, wenn er eine gültige Anmeldung erhält. Über dieses Verzeichnis hat nur der Benutzer selbst das Verfügungsrecht; ohne seine Erlaubnis kann darin niemand Dateien lesen oder beschreiben, und er legt auch die Struktur des Verzeichnisses selbst fest.

Im Betriebssystem UNIX gibt es außerdem mehrere Verzeichnisse, die das System selbst verwendet. Die Struktur dieser Verzeichnisse ist auf allen UNIX-Systemen sehr ähnlich. Zu diesen Verzeichnissen gehören **/unix** (der Kern) und einige weitere wichtige Systemverzeichnisse; sie befinden sich in der Dateisystemhierarchie direkt unter dem Root-Verzeichnis. Das Root-Verzeichnis (durch **/** gekennzeichnet) ist der Ausgangspunkt in der Dateistruktur von UNIX: alle anderen Verzeichnisse und alle Dateien sind hierarchisch unter ihm angeordnet.

Gerätedateien

Gerätedateien sind die ungewöhnlichsten Komponenten des UNIX-Dateisystems. Eine Gerätedatei steht für ein physikalisches Gerät, wie beispielsweise ein Terminal, ein Plattenlaufwerk, ein Bandlaufwerk oder eine Übertragungsverbindung. Das System liest aus Gerätedateien und schreibt in sie ebenso wie bei normalen Dateien. Im Unterschied dazu werden jedoch durch die Schreib- und Leseanforderungen die normalen Dateizugriffsmechanismen nicht aktiviert; stattdessen wird das Steuerprogramm für das Gerät aktiviert, das dieser Datei zugeordnet ist.

In manchen Betriebssystemen ist es erforderlich, den jeweils vorliegenden Dateityp anzugeben und eine Datei in einer bestimmten Form zu verwenden. In diesen Fällen muß jeweils berücksichtigt werden, wie die Dateien gespeichert sind, da es sich um sequentielle Dateien, Dateien mit wahlfreiem Zugriff oder Binärdateien handeln kann. Das Betriebssystem UNIX macht jedoch keine solchen Unterschiede, und daher ist das UNIX-Dateisystem einfach in der Handhabung. Es muß beispielsweise kein Speicherbedarf für die Dateien angegeben

werden, da das System diesen automatisch bestimmt. Muß ein Benutzer oder ein von ihm geschriebenes Programm auf ein bestimmtes Gerät zugreifen, wie z. B. einen Drucker, braucht er lediglich das Gerät wie eine Datei anzugeben. Im Betriebssystem UNIX gibt es nur eine Schnittstelle für alle Eingaben des Benutzers und die Ausgaben des Systems an ihn; dadurch wird die Arbeit des Benutzers mit dem System weitgehend vereinfacht.

Abbildung 1-4 ist ein Beispiel eines typischen Dateisystems. Das Root-Verzeichnis enthält dabei den Kern (`/unix`) und einige weitere wichtige Systemverzeichnisse.

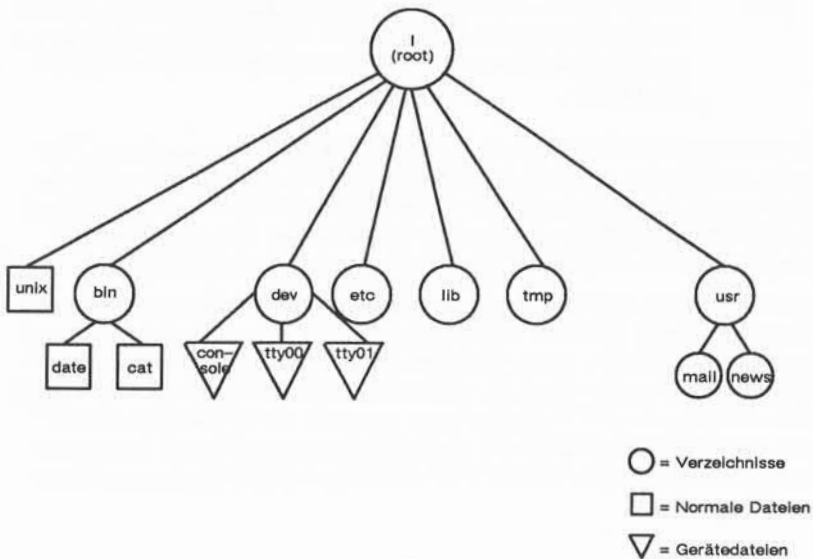


Abbildung 1-4: Beispiel eines Dateisystems

<code>/bin</code>	enthält viele ausführbare Programme und Dienstprogramme.
<code>/dev</code>	enthält Gerätedateien, die Peripheriegeräten wie der Konsole, dem Zeilendrucker, den Benutzerterminals und Plattenlaufwerken zugeordnet sind.
<code>/etc</code>	enthält Programme und Datendateien zur Systemverwaltung.
<code>/lib</code>	enthält Bibliotheken für Programme und Programmiersprachen.
<code>/tmp</code>	dient zur Aufnahme von Zwischendateien (temporären Dateien), die von jedem Benutzer angelegt werden können.
<code>/usr</code>	enthält weitere Verzeichnisse, darunter mail , in dem die Dateien zur Speicherung der elektronischen Post untergebracht sind, und news , in dem Dateien zur Speicherung von wichtigen Neuigkeiten stehen.

Zusammenfassend kann man sagen, daß die vom Benutzer angelegten Verzeichnisse und Dateien den Teil des Dateisystems bilden, über den er selbst das Verfügungsrecht hat. Andere Teile des Dateisystems werden vom Betriebssystem selbst angelegt und gepflegt, wie zum Beispiel `/bin`, `/dev`, `/etc`, `/lib`, `/tmp` und `/usr`; ihre Struktur ist auf allen UNIX-Systemen weitgehend identisch.

Das Dateisystem wird in den weiteren Kapiteln ausführlicher beschrieben. In Kapitel 3 wird gezeigt, wie eine Verzeichnisstruktur in einem Dateisystem organisiert wird, wie auf Dateien zugegriffen wird, und wie sie verwaltet werden. Kapitel 4 enthält eine Übersicht über die Fähigkeiten des Betriebssystems UNIX. Eine wirksame Nutzung dieser Fähigkeiten hängt davon ab, wie vertraut der Benutzer mit dem Dateisystem ist, und wie gut er auf die darin gespeicherten Informationen zugreifen kann. Kapitel 5 und 6 sind Anleitungen zum Erstellen und Editieren von Dateien.

Die Shell

Die Shell ist ein einzigartiger Kommandointerpreter, der es dem Benutzer ermöglicht, mit dem Betriebssystem zu kommunizieren. Die Shell liest die eingegebenen Kommandos und interpretiert sie als Aufträge zur Ausführung anderer Programme, zum Zugriff auf Dateien oder zur Ausgabe von Informationen. Die Shell ist gleichzeitig eine leistungsfähige Programmiersprache, die in vielem der Programmiersprache C ähnelt und über Programmierereinrichtungen wie bedingte Ausführung von Kommandos und Datenflußsteuerung verfügt.

Das Modell eines UNIX-Systems in Abbildung 1-1 veranschaulicht den Ablauf der Kommunikation zwischen dem Benutzer und dem Rechner in beiden Richtungen über die Shell.

In Kapitel 4 werden die Fähigkeiten der Shell beschrieben. Kapitel 7 ist eine Anleitung zum Schreiben von einfachen Shell-Programmen, die als Shell-Scripts bezeichnet werden, und die zum Anpassen der Umgebung an die eigenen Anforderungen dienen.

Kommandos

Ein Programm besteht aus einer Reihe von Anweisungen an den Rechner. Programme, die vom Rechner ohne vorherige Übersetzung ausgeführt werden können, werden ausführbare Programme oder Kommandos genannt. Dem Normalbenutzer des UNIX-Systems stehen viele standardmäßige Programme und Werkzeuge zur Verfügung. Dem Benutzer, der das UNIX-System zum Schreiben von Programmen und zum Entwickeln von Software verwendet, stehen auch Systemaufrufe, Unterprogramme und andere Werkzeuge zur Verfügung. Selbstverständlich können auch alle selbst geschriebenen Programme eingesetzt werden.

In diesem Handbuch wird der Benutzer in viele der UNIX-Systemprogramme und -werkzeuge eingeführt, die häufig benötigt werden. Weitere Informationen zu diesen oder anderen Standardprogrammen sind im *User's Reference Manual* zu finden. Informationen zu Werkzeugen und Routinen zur Programmierung und Softwareentwicklung sind im *Programmer's Reference Manual* enthalten.

Diese Nachschlagewerke stehen auch on-line zur Verfügung (On-Line-Dokumentationen sind im Dateisystem des Rechners gespeichert). Dabei können Seiten aus diesen On-Line-Handbüchern über das Kommando **man** (Abkürzung für "Manual" - Handbuch) abgerufen werden. Nähere Angaben zur Verwendung des Kommandos **man** sind unter **man(1)** im *User's Reference Manual* zu finden.

Funktionen der Kommandos

Im äußeren Kreis des Modells des Betriebssystems UNIX in Abbildung 1-1 werden die Systemprogramme und Werkzeuge in folgende funktionale Kategorien eingeteilt:

Textverarbeitung	Das System enthält Programme wie Zeilen- und Bildschirm-Editoren zum Erstellen und Ändern von Texten, ein Programm zur Rechtschreibprüfung und optionale Textformateinrichtungen, mit denen Papierausdrucke hoher Qualität erzeugt werden können, die sich zur Veröffentlichung eignen.
Datenverwaltung	Das System enthält viele Programme, mit denen Dateien und Verzeichnisse erstellt, strukturiert und gelöscht werden können.
Elektronische Kommunikation	Mehrere Programme, darunter mail , ermöglichen die Übermittlung von Informationen an andere Benutzer und andere UNIX-Systeme.
Softwareentwicklung	Verschiedene UNIX-Systemprogramme stellen eine benutzerfreundliche Programmierumgebung mit Schnittstellen von UNIX zur Programmiersprache und mit einer Vielzahl von Dienstprogrammen zur Verfügung.
Weitere Dienstprogramme	Das System bietet auch Rechenfunktionen und Funktionen zum Erstellen von Grafiken.

Eingabe der Kommandos

Damit das UNIX-System eingegebene Kommandos verstehen kann, müssen diese im richtigen Format, bzw. in der richtigen Kommandozeilensyntax eingegeben werden. Diese Syntax bestimmt die Reihenfolge, in der die einzelnen Bestandteile einer Kommandozeile einzugeben sind. Ebenso wie man in einem normalen deutschen Aussagesatz das Subjekt vor das Verb stellt, müssen die Bestandteile einer Kommandozeile in der durch die Kommandozeilensyntax vorgegebenen Reihenfolge angegeben werden, da die Shell des UNIX-Systems sonst die Anforderung nicht interpretieren kann. Die Syntax einer Kommandozeile des Betriebssystems UNIX kann beispielsweise so aussehen:

Kommando *Option(en)* *Argument(e)* <CR>

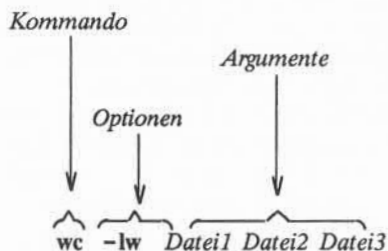
In jeder Kommandozeile von UNIX müssen mindestens zwei Dinge eingegeben werden: ein Kommandoname und die Eingabetaste RETURN (in diesem *Handbuch* wird stets die Notation <CR> verwendet, wenn die RETURN-Taste gedrückt werden muß). Eine Kommandozeile kann außerdem Optionen oder Argumente oder beides enthalten.

Die einzelnen Komponenten sind folgendermaßen definiert:

- Ein *Kommando* ist der Name des Programms, das ausgeführt werden soll.
- Eine *Option* ist eine Ergänzungsangabe zu einem Kommando und modifiziert die Form der Ausführung des Kommandos.
- Mit einem *Argument* werden Daten angegeben, auf die das Kommando angewendet werden soll (in der Regel ein Verzeichnis- oder Dateiname).

In Kommandozeilen, die Optionen und/oder Argumente enthalten, sind die einzelnen Bestandteile durch mindestens ein Leerzeichen voneinander zu trennen (Leertaste drücken). Enthält ein Argument ein Leerzeichen, ist dieser Name in doppelte Anführungszeichen zu setzen. Lautet das Argument für ein Kommando beispielsweise **muster 1**, ist es folgendermaßen einzugeben: "**muster 1**". Läßt man die doppelten Anführungszeichen weg, wird die Shell **muster** und **1** als zwei getrennte Argumente interpretieren.

Bei einigen Kommandos können in einer Kommandozeile mehrere Optionen und/oder Argumente angegeben werden, wie im folgenden Beispiel einer Kommandozeile:



In diesem Beispiel ist `wc` der Name des Kommandos; er wird durch zwei Optionen, `-l` und `-w`, modifiziert. Außerdem wurden drei Dateien (`Datei1`, `Datei2` und `Datei3`) als Argumente angegeben. Man kann die meisten Optionen zusammen verwenden, nicht jedoch Argumente.

Im folgenden Beispiel wird die richtige Reihenfolge und der richtige Abstand der einzelnen Komponenten einer Kommandozeile dargestellt:

Falsch	Richtig
<code>wcDatei</code>	<code>wc Datei</code>
<code>wc-lDatei</code>	<code>wc -l Datei</code>
<code>wc -l w Datei</code>	<code>wc -lw Datei</code>
<code>wc Datei1Datei2</code>	<code>wc Datei1 Datei2</code>

Dabei ist zu beachten, daß jede dieser Kommandozeilen durch Drücken der RETURN-Taste abgeschlossen werden muß.

Ausführung von Kommandos

Abbildung 1-5 zeigt den Steuerungsablauf bei der Ausführung eines Kommandos durch das Betriebssystem UNIX.

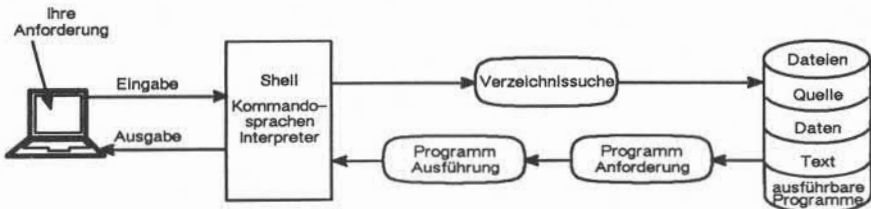


Abbildung 1-5: Ausführung eines Kommandos im Betriebssystem UNIX

Ein Kommando kann zur Ausführung eingegeben werden, wenn ein Bereit-Zeichen (wie zum Beispiel `␣`) auf dem Bildschirm erscheint. Die Shell betrachtet das Kommando als Eingabe, sucht in einem oder mehreren Verzeichnissen nach dem angegebenen Programm und übergibt diese Anforderung zusammen mit dem angeforderten Programm an den Kern des Betriebssystems. Der Kern befolgt dann die Anweisungen des Programms und führt damit das eingegebene Kommando aus. Nach dem Ende des Programms meldet die Shell mit dem Bereit-Zeichen, daß sie das nächste Kommando entgegennehmen kann.

In diesem Kapitel wurden einige Grundprinzipien des Betriebssystems UNIX beschrieben. In den folgenden Kapiteln wird die Anwendung dieser Prinzipien im Hinblick auf die Anforderungen von Anwendern zur Datenverarbeitung erläutert.

Kapitel 2: GRUNDLAGEN FÜR UNIX-ANWENDER

Einführung	2-1
Das Terminal	2-2
Terminalkonfiguration	2-3
Die Tastatur	2-4
Eingabekonventionen	2-6
Das Bereit-Zeichen	2-8
Korrektur von Tippfehlern	2-8
Verwendung von Sonderzeichen als Textzeichen	2-10
Eingabegeschwindigkeit	2-11
Unterbrechen eines Kommandos	2-11
Verwendung von Steuerzeichen	2-12
Zuweisung eines Benutzernamens	2-13
Eröffnen einer UNIX-Sitzung	2-14
Anmeldung	2-16
Paßwort	2-16
Probleme beim Anmelden	2-20
Einfache Kommandos	2-22
Abmeldung	2-23

Einführung

In diesem Kapitel werden die allgemeinen Regeln und Richtlinien für die Arbeit mit dem Betriebssystem UNIX erläutert. Insbesondere wird die erforderliche Konfiguration des Terminals beschrieben, und es wird erläutert, wie die Tastatur zu bedienen ist, wie ein Benutzername definiert wird, wie man sich am System an- und abmeldet und wie man einfache Kommandos eingibt.

Für die Arbeit mit dem UNIX-System benötigt man:

- ein Terminal
- einen Benutzernamen (ein Name, mit dem man sich dem UNIX-System gegenüber als berechtigter Benutzer identifiziert)
- ein Paßwort, mit dem die Identität des Benutzers geprüft wird
- Informationen zur Herstellung einer Wählverbindung und zum Zugriff auf ein UNIX-System, wenn das Terminal nicht direkt an den Rechner angeschlossen ist.

In diesem Kapitel werden die Notationskonventionen wie in den übrigen Teilen dieses *Handbuchs* verwendet; sie wurden im Vorwort erläutert.

Das Terminal

Ein Terminal ist ein Gerät zur Ein- und Ausgabe: man gibt damit Anforderungen in das UNIX-System ein, und das System gibt über das Terminal seine Antworten an den Benutzer aus. Es gibt zwei Haupttypen von Terminals: Bildschirmterminals und Druckerterminals. Ein Bildschirmterminal ist in Abbildung 2-1 dargestellt.

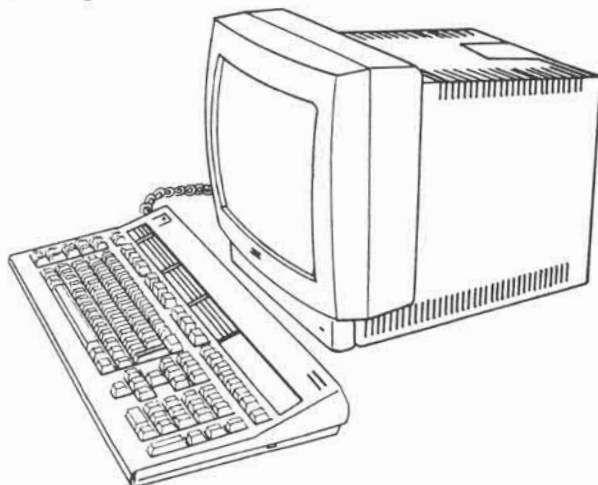


Abbildung 2-1: Bildschirmterminal BA80

Beim Bildschirmterminal werden die Ein- und Ausgaben auf einem Bildschirm angezeigt, bei einem Druckerterminal werden sie auf Endlospapier fortlaufend ausgedruckt. Die Bedienung und die Reaktionen des Systems bleiben dabei im wesentlichen dieselben. Die Informationen in diesem Handbuch, die sich auf ein Bildschirmterminal beziehen, gelten ebenso auch für den Ausdruck auf einem Druckerterminal, wenn es nicht ausdrücklich anders angegeben wird.

Weitere Informationen zu den Nixdorf-Terminals sind in den Beschreibungen der entsprechenden Terminals enthalten.

Terminalkonfiguration

Unabhängig vom verwendeten Terminaltyp muß das Terminal auf jeden Fall richtig konfiguriert werden, um mit dem UNIX-System kommunizieren zu können. Wer noch keine Erfahrung bei der Konfiguration eines Terminals hat, sollte jemanden mit Erfahrung zur Unterstützung heranziehen.

Die Konfiguration eines Terminals hängt vom verwendeten Terminaltyp ab. Manche Terminals werden über Schalter konfiguriert, andere direkt über die Tastatur mit Hilfe einer Reihe von Funktionstasten. Die entsprechenden Informationen sind im Benutzerhandbuch zum betreffenden Terminal zu finden.

Die folgende Liste von Konfigurationsprüfungen sollte mit jedem Terminal durchgeführt werden, bevor man versucht, sich am UNIX-System anzumelden.

1. Terminal einschalten.
2. Das Terminal auf ON-LINE oder REMOTE (Remote-Betrieb) einstellen. Dadurch wird das Terminal direkt vom Rechner kontrolliert.
3. Den Modus FULL DUPLEX einstellen. Damit kann die Kommunikation zwischen dem Benutzer und dem UNIX-System in beiden Richtungen (Eingabe/Ausgabe) erfolgen.
4. Ist das Terminal nicht direkt bzw. fest an den Rechner angeschlossen, ist sicherzustellen, daß der verwendete Akustikkoppler bzw. Modem auf den Modus FULL DUPLEX eingestellt ist.
5. Die Zeichengenerierung auf LOWER CASE einstellen. Kann ein Terminal nur Großbuchstaben erzeugen, paßt sich das UNIX-System dem an, indem es alles in Großbuchstaben ausgibt.
6. Das Terminal auf NO PARITY einstellen.
7. Die richtige Baudrate einstellen. Dies ist die Geschwindigkeit, mit der der Rechner mit dem Terminal kommuniziert; sie wird in Zeichen pro Sekunde gemessen (ein Terminal, bei dem die Baudrate auf 4800 eingestellt ist, sendet und empfängt 4800 Zeichen pro Sekunde). Je nach Rechner und Terminal stehen Baudraten zwischen 300 und 19200 zur Verfügung. Manche Rechner können Zeichen sogar mit noch höheren Geschwindigkeiten verarbeiten.

Die Tastatur

Das Layout von Terminaltastaturen ist nicht allgemein genormt. Alle Terminaltastaturen haben jedoch einen gemeinsamen Zeichensatz, der aus 128 Zeichen besteht und ASCII-Zeichensatz genannt wird (ASCII ist ein Kurzwort für American Standard Code for Information Interchange - Amerikanischer Standardcode zum Informationsaustausch). Die Tasten sind mit Zeichen beschriftet, die für den Benutzer verständlich sind (wie zum Beispiel den Buchstaben des Alphabets); gleichzeitig ist jeder Taste auch ein ASCII-Code zugeordnet, der vom Rechner verstanden wird.

Die Anordnung der Tasten bei einem typischen ASCII-Terminal entspricht weitgehend der einer Schreibmaschine. Zusätzlich stehen noch einige weitere Tasten für Funktionen wie beispielsweise das Unterbrechen von Verarbeitungsaufträgen zur Verfügung. In Abbildung 2-2 ist als Beispiel eine Tastatur an einem ASCII-Terminal dargestellt.

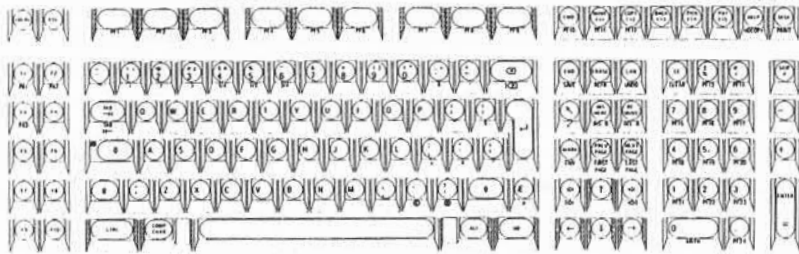


Abbildung 2-2: Layout der Tastatur CT08

Die Tasten sind folgendermaßen belegt:

- die Buchstaben des deutschen Alphabets (Groß- und Kleinbuchstaben)
- die Ziffern (0 bis 9)
- eine Anzahl von Sonderzeichen (! @ # \$ % ^ & () _ - + = ~ ' { } [] \ ; " ' < > , ? /)
- Wörter mit besonderer Bedeutung (wie RETURN - Eingabetaste und BREAK - Unterbrechung) und Abkürzungen (wie DEL für Löschtaste, CTRL für Steuertaste und ESC für Escape).

Tastaturen von Terminals und Schreibmaschinen haben die alphanumerischen Tasten gemeinsam, Terminaltastaturen haben jedoch außerdem Tasten, die für die Verwendung mit einem Rechner vorgesehen sind. Diese Tasten sind mit Zeichen oder Symbolen beschriftet, die auf ihre Funktionen hinweisen. Ihre Anordnung kann jedoch von Terminal zu Terminal unterschiedlich sein, da es kein genormtes Layout für Tastaturen gibt.

Eingabekonventionen

Damit der Benutzer effektiv mit dem UNIX-System arbeiten kann, sollte er mit den Konventionen zur Eingabe vertraut sein. Im UNIX-System müssen die Kommandos in Kleinbuchstaben eingegeben werden (es sei denn, das Kommando enthält einen Großbuchstaben). Weitere Konventionen sind so definiert, daß damit bestimmte Aufgaben ausgeführt werden; durch einfaches Drücken einer Taste oder Tastenkombination werden beispielsweise Buchstaben oder Zeilen gelöscht. Die Zeichen, mit denen diese Aufgaben ausgeführt werden, werden Sonderzeichen genannt. In Abbildung 2-3 sind die Konventionen zu den Sonderzeichen aufgeführt. Auf den nachfolgenden Seiten werden sie im Einzelnen erläutert.

Taste(n)	Bedeutung
␣	Bereit-Zeichen des Systems (Aufforderung, ein Kommando einzugeben)
#*	Ein Zeichen löschen (Erase)
@	Eine ganze Zeile löschen (Kill)
<BREAK>	Ausführung eines Programms bzw. Kommandos unterbrechen
	Aktuelle Kommandozeile löschen
<ESC>	In Verbindung mit einem anderen Zeichen wird damit eine besondere Funktion ausgeführt (wird dann Escape-Sequenz genannt) In einer Editiersitzung mit dem Editor vi wird damit der Texteingabemodus beendet, und es erfolgt die Rückkehr in den Kommandomodus.
<CR>	RETURN-Taste (Eingabetaste) drücken. Damit wird eine Eingabezeile beendet, und der Cursor wird in eine neue Zeile gesetzt.
<^d>†	Eingabe in das System beenden oder abmelden.
<^h>	Rücktaste zur Verwendung bei Terminals, die nicht über eine separate BACKSPACE-Taste verfügen.
<^i>	Horizontaltabulator für Terminals ohne Tabulatortaste
<^s>	Vorübergehend die Ausgabe auf dem Bildschirm anhalten
<^q>	Ausgabe auf dem Bildschirm weiterlaufen lassen, nachdem sie mit dem Kommando <^s> angehalten wurde.

* Nicht druckbare Zeichen erscheinen in spitzen Klammern (< >).

† Zeichen, denen ein Zirkumflex (^) vorangestellt ist, werden Steuerzeichen genannt (im allgemeinen Sprachgebrauch werden sie CONTROL-Buchstabe genannt). Diese Steuerzeichen werden eingegeben, indem man die CONTROL-Taste gedrückt hält und dann die betreffende Buchstabentaste drückt.

Abbildung 2-3: Eingabekonventionen im UNIX-System

Das Bereit-Zeichen

Das standardmäßige Bereit-Zeichen im UNIX-System ist das Dollarzeichen ($\$$). Wenn dieses Bereit-Zeichen auf dem Terminal erscheint, erwartet das UNIX-System eine Anweisung des Benutzers, d. h. die Eingabe eines Kommandos mit anschließendem Drücken der RETURN-Taste.

Das Dollarzeichen ($\$$) ist der Standardwert für das Bereit-Zeichen. In Kapitel 7 wird erklärt, wie man es ändern kann, wenn man ein anderes Zeichen oder eine Zeichenkette als Bereit-Zeichen definieren will.

Korrektur von Tippfehlern

Für die Korrektur von Tippfehlern stehen zwei verschiedene Tasten zur Verfügung; mit ihnen werden Textzeichen gelöscht. Mit der Taste @ wird die aktuelle Zeile gelöscht, mit der Taste # (Nummernzeichen) wird das zuletzt eingegebene Zeichen gelöscht. Diese Tasten sind standardmäßig mit diesen Funktionen belegt. Sollen jedoch andere Tasten dafür verwendet werden, können diese Funktionen umbelegt werden (nähere Angaben dazu sind unter "Umbelegen der Löschfunktionen" weiter unten in diesem Abschnitt und "Definition der Terminaloptionen" in Kapitel 7 zu finden).

Löschen der aktuellen Zeile: das Kill-Zeichen @

Mit der Taste @ wird die aktuelle Zeile gelöscht. Wenn man sie drückt, wird das Zeichen @ an das Ende der Zeile gesetzt, und der Cursor rückt in die nächste Zeile. Die fehlerhafte Zeile wird nicht vom Bildschirm gelöscht, sondern einfach ignoriert.

Die Taste @ wirkt jeweils nur in der aktuellen Zeile; daher muß sie gedrückt werden, bevor man die RETURN-Taste drückt, wenn man eine Zeile löschen will. Im folgenden Beispiel wird ein Kommando in einer Kommandozeile falsch geschrieben und über die Taste @ annulliert:

```
whooo@  
who<CR>
```


Löschen des zuletzt eingegebenen Zeichens: das Zeichen #

Mit der Taste # (Nummernzeichen) wird das zuletzt eingegebene Zeichen bzw. die zuletzt eingegebenen Zeichen in der aktuellen Zeile gelöscht. Gibt man das Zeichen # ein, wird der Cursor auf das letzte Zeichen gesetzt, so daß man es überschreiben und damit löschen kann. So können Tippfehler einfach korrigiert werden.

Es können beliebig viele Zeichen gelöscht werden, wenn man die entsprechende Anzahl von Nummernzeichen (#) eingibt. In der folgenden Kommandozeile werden beispielsweise zwei Zeichen durch die Eingabe von zwei Nummernzeichen gelöscht.

```
dattw##e <CR>
```

Dies wird dann vom Betriebssystem UNIX als Kommando `date` - richtig geschrieben - interpretiert.

Die BACKSPACE-Taste

Für das Löschen eines Zeichens wird häufig die BACKSPACE-Taste anstelle der Taste # verwendet. Drückt man die BACKSPACE-Taste, wird der Cursor um eine Stelle zurückgesetzt und löscht dabei das letzte Zeichen. Dabei wird im Gegensatz zur Taste # kein Zeichen auf dem Bildschirm zwischen Fehler und Korrektur gesetzt. Nach dem Korrigieren mit der BACKSPACE-Taste sieht die Textzeile so aus, als sei kein Fehler gemacht worden.

Das Erase-Zeichen # und die BACKSPACE-Taste löschen Zeichen gleichermaßen, mit der BACKSPACE-Taste hat man jedoch eine bessere Übersicht über die Eingabe.



Bestimmte Terminals erkennen die Taste # nicht als Löschzeichen.

Umbelegen der Löschkfunktionen

Wie bereits erwähnt, können die Tasten zum Löschen von Zeilen und Zeichen (Kill bzw. Erase) geändert werden. Sollen diese Zeichen nur für die Dauer einer Arbeitssitzung umbelegt werden, kann dies über ein Kommando an die Shell erreicht werden; dann werden die Löschkfunktionen wieder auf die Standardtasten (# und @) gelegt, sobald man sich abmeldet. Sollen auf Dauer andere Tasten dafür verwendet werden, muß die Umbelegung in einer Datei mit dem Namen `.profile` definiert werden. Nähere Hinweise zum vorübergehenden und dauerhaften Umbelegen von Tasten sowie eine Beschreibung der Datei `.profile` sind in Kapitel 7 enthalten.

Beim Umbelegen der LösCHFunktionen auf andere Tasten sind drei Dinge zu beachten. Erstens kann im UNIX-System jeweils nur eine Taste mit einer LösCHFunktion belegt werden. Wird eine Funktion auf eine andere Taste gelegt, wird sie gleichzeitig von der Standardtaste abgezogen. Belegt man beispielsweise die BACKSPACE-Taste mit der LösCHFunktion der Taste #, kann man mit der Taste # keine Zeichen mehr löschen. Damit stehen auch nicht mehr zwei Tasten für diese Funktion zur Verfügung.

Zweitens werden solche Umbelegungen von allen anderen UNIX-Systemprogrammen übernommen, die über die entsprechende Funktion verfügen. Der interaktive Texteditor `ed` (in Kapitel 5 beschrieben) ermöglicht es beispielsweise, Text mit derselben Taste zu löschen, die auch für die Korrektur von Fehlern in einer Kommandozeile der Shell (wie hier beschrieben) verwendet wird. Legt man daher die Erase-Funktion auf die BACKSPACE-Taste, muß die BACKSPACE-Taste auch im Editor `ed` zum Löschen von Zeichen verwendet werden. Die Taste # führt dann diese Funktion nicht mehr aus.

Schließlich ist darauf zu achten, daß alle in der Datei `.profile` definierten Umbelegungen erst nach der Anmeldung gültig werden. Macht man also bei der Eingabe des Benutzernamens oder Paßwortes einen Fehler, muß er mit der Taste # korrigiert werden.

Unabhängig davon, welche Tasten man verwendet, gelten sie jeweils nur in der aktuellen Zeile. Daher müssen Fehler korrigiert werden, noch bevor man die RETURN-Taste am Ende der Zeile drückt.

Verwendung von Sonderzeichen als Textzeichen

Man kann ein Sonderzeichen mit seiner eigentlichen Bedeutung als Textzeichen eingeben. Da das Betriebssystem UNIX normalerweise Sonderzeichen als Kommandos interpretiert, muß man das System anweisen, diese Sonderbedeutung zu ignorieren, wenn man es als Textzeichen verwenden will. Zu diesem Zweck verwendet man den umgekehrten Schrägstrich (`\`); er wird vor das Sonderzeichen gesetzt, das als normales Zeichen behandelt werden soll. Damit wird dem System mitgeteilt, die Sonderbedeutung zu ignorieren und es als normales Textzeichen zu behandeln.

Angenommen, man möchte folgenden Satz in eine Datei schreiben:

Auf diesem Notenblatt erscheint nur ein #.

Damit das UNIX-System das Zeichen # nicht als Anweisung interpretiert, ein Zeichen zu löschen, wird ein \ vor dem Zeichen # eingegeben. Unterläßt man dies, so wird das Leerzeichen nach dem Wort "ein" gelöscht, und der Satz wird folgendermaßen aussehen:

Auf diesem Notenblatt erscheint nur ein.

Daher ist der Satz folgendermaßen einzugeben:

Auf diesem Notenblatt erscheint nur ein \#.

Eingabegeschwindigkeit

Wenn das Bereit-Zeichen auf dem Bildschirm erscheint, kann man beliebig schnell Zeichen eingeben, selbst wenn das UNIX-System gerade ein Kommando ausführt oder auf ein Kommando reagiert. Da die Eingaben des Benutzers und die Ausgaben des Systems durcheinander auf dem Bildschirm erscheinen, ist dies für den Benutzer unübersichtlich. beeinträchtigt jedoch nicht die Funktionsfähigkeit des UNIX-Systems, da es über die Fähigkeit zum Vorauslesen (Read Ahead) verfügt. Dadurch kann das System Eingaben und Ausgaben separat behandeln. Es nimmt Eingaben (die nächste Anforderung des Benutzers) entgegen und speichert sie, während es Ausgaben (Antworten auf vorherige Anforderungen) auf dem Bildschirm ausgibt.

Unterbrechen eines Kommandos

Soll die Ausführung eines Kommandos unterbrochen werden, ist einfach die Taste BREAK oder DELETE zu drücken. Das UNIX-System unterbricht dann das Programm und gibt ein Bereit-Zeichen auf dem Bildschirm aus. Damit zeigt es an, daß das zuletzt ausgeführte Kommando in der Ausführung unterbrochen wurde, und daß es für die Eingabe des nächsten Kommandos bereit ist.

Verwendung von Steuerzeichen

Auf der Tastatur des Terminals befindet sich eine Steuertaste. Sie ist meist mit CONTROL, CTRL oder - bei einer deutschen Tastatur - mit STRG beschriftet und befindet sich in der Regel links von der Taste A oder unter der Taste Y. Die CONTROL-Taste dient in Verbindung mit anderen Zeichen dazu, Steuerfunktionen auf Eingabezeilen anzuwenden. Diese Form von Kommandos nennt man Steuerzeichen. Einige Steuerzeichen dienen dazu, allgemeine Aufgaben wie dem Zurücksetzen des Cursors mit gleichzeitigem Zeichenlöschen und Tabulatorvorschübe auszuführen, mit anderen werden Kommandos eingegeben, die nur innerhalb des UNIX-Systems definiert sind. So wird beispielsweise mit einem der Steuerzeichen (CONTROL-s) die Ausgabe auf einem Terminal vorübergehend angehalten.

Ein Steuerzeichen wird eingegeben, indem man die CONTROL-Taste gedrückt hält und dann die betreffende Buchstabentaste drückt. Die meisten Steuerzeichen erscheinen nicht auf dem Bildschirm, wenn sie eingegeben werden; sie werden daher hier zwischen spitzen Klammern dargestellt (siehe "Notationskonventionen" im Vorwort). Die CONTROL-Taste wird durch einen Zirkumflex (^) vor dem betreffenden Buchstaben dargestellt. Mit <^s> wird beispielsweise das Zeichen CONTROL-s dargestellt.

Am häufigsten verwendet werden Steuerzeichen zur Steuerung der Ausgabe auf dem Bildschirm und zum Abmelden vom System. Will man verhindern, daß Informationen bei der Ausgabe zu schnell über den Bildschirm laufen, gibt man <^s> ein, um die Anzeige anzuhalten. Wenn man danach die folgenden Informationen lesen will, gibt man <^q> ein, um die Ausgabe weiterlaufen zu lassen.

Zum Abmelden vom UNIX-System ist <^d> einzugeben (nähere Einzelheiten dazu siehe unter "Abmelden" weiter unten in diesem Kapitel).

Außerdem werden im UNIX-System Steuerzeichen dazu verwendet, Funktionen zu realisieren, die bei einigen Terminals nicht über Funktionstasten zur Verfügung stehen. Ist auf einer Tastatur beispielsweise keine BACKSPACE-Taste vorhanden, kann die Taste(nkombination) <^h> stattdessen verwendet werden. Ist keine Tabulatortaste vorhanden, können Tabulatoren mit <^i> gesetzt werden, sofern die Terminaloptionen entsprechend definiert sind (siehe dazu den Abschnitt "Probleme beim Anmelden").

Nachdem nun das Terminal richtig konfiguriert ist und die Belegung der Tastatur bekannt ist, bleibt noch ein Schritt zu tun, bevor man sich im UNIX-System anmelden kann: man muß sich einen Benutzernamen zuweisen lassen.

Zuweisung eines Benutzernamens

Ein Benutzername ist der Name, mit dem das UNIX-System überprüft, ob der Benutzer, der sich Zugang zum System verschaffen will, dazu berechtigt ist. Er ist jedesmal einzugeben, wenn sich ein Benutzer anmelden will.

Man erhält einen Benutzernamen, indem man durch den Systemverwalter ein Benutzerkonto einrichten läßt. Für die Auswahl eines Benutzernamens gibt es ein paar Regeln. Normalerweise hat der Name eine Länge von drei bis acht Zeichen. Er kann eine beliebige Kombination von alphanumerischen Zeichen (Kleinschreibung) enthalten, muß jedoch mit einem Buchstaben beginnen. Er darf keine Sonderzeichen enthalten.

Der Benutzername wird in der Regel nach der am betreffenden Standort geübten Praxis definiert. So können beispielsweise die Benutzer eines Systems jeweils ihre Initialen, ihre Nachnamen oder ihre Spitznamen als Benutzernamen verwenden. Beispiele für zulässige Benutzernamen sind **zugvogel**, **petra** und **jib**.

Eröffnen einer UNIX-Sitzung

In der Regel verwendet man entweder ein Terminal, das direkt mit dem Rechner verbunden ist, oder eines, das über eine Telefonleitung mit einem Rechner kommuniziert.

NOTE

In diesem Abschnitt wird eine typische Anmeldeprozedur beschrieben; sie ist jedoch vom jeweiligen System abhängig. Es gibt viele Möglichkeiten, sich über eine Telefonleitung an einem UNIX-System anzumelden. Es ist möglich, daß aus Sicherheitsgründen dafür eine spezielle Telefonnummer oder ein anderer Sicherheitscode erforderlich ist. Informationen über die Anmeldung an einem UNIX-System von außerhalb des Standortes sind beim Systemverwalter einzuholen.

Das Terminal einschalten. Ist es direkt angeschlossen, erscheint sofort `login:` in der linken oberen Ecke des Bildschirms.

Erfolgt die Kommunikation mit dem Rechner über eine Telefonleitung, muß nun eine Verbindung hergestellt werden. Im folgenden wird dieses Verfahren anhand eines Beispiels beschrieben (im konkreten Fall ist das Verfahren beim Systemverwalter zu erfragen).

1. Die Telefonnummer des UNIX-Systems wählen. Man wird dann eines der folgenden Signale hören:
 - Belegtzeichen. Entweder ist der Rechner oder die Leitung belegt. Auflegen und neu wählen.
 - Anhaltender Wählton, aber keine Antwort. Dies bedeutet in der Regel, daß die Telefonleitung nicht in Ordnung oder das System aufgrund eines mechanischen Defektes oder einer elektronischen Störung nicht betriebsbereit ist. Auflegen und später neu wählen.
 - Hochfrequenter Ton. Damit wird signalisiert, daß das System zugänglich ist.
2. Ertönt der hochfrequente Ton, den Telefonhörer in den Akustikkoppler einlegen bzw. kurz den entsprechenden Knopf des Modems drücken (siehe Bedienungsanleitung des entsprechenden Gerätes) und dann den Hörer wieder auflegen (siehe Abbildung 2-4).
3. Nach ein paar Sekunden erscheint die Eingabeaufforderung `login:` in der linken oberen Ecke des Bildschirms.

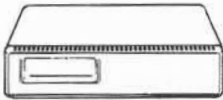
4. Nun kann eine Reihe sinnloser Zeichen auf dem Bildschirm erscheinen. Dies bedeutet, daß auf der verwendeten Wählleitung mehrere Baudraten möglich sind und das UNIX-System versucht, mit dem Terminal Verbindung aufzunehmen, dabei jedoch die falsche Übertragungsgeschwindigkeit verwendet. Die Taste BREAK oder RETURN drücken; damit wird das System angewiesen, eine andere Geschwindigkeit zu verwenden. Erscheint nach ein paar Sekunden noch immer nicht die Eingabeaufforderung `login:`, ist die Taste BREAK oder RETURN nochmals zu drücken.



Datentelefon



Akustikkoppler



Modem

Abbildung 2-4: Datentelefon, Modem und Akustikkoppler

Anmeldung

Wenn die Eingabeaufforderung `login:` erscheint, den Benutzernamen eingeben, dann die RETURN-Taste drücken. Lautet der Benutzername beispielsweise `zugvogel`, sieht die Anmeldezeile folgendermaßen aus:

```
login: zugvogel <CR>
```



Es ist zu beachten, daß die Eingabe in Kleinbuchstaben erfolgen muß. Verwendet man Großbuchstaben bei der Anmeldung, erwartet das UNIX-System bis zur nächsten Anmeldung nur Großbuchstaben und antwortet auch nur mit diesen. In diesem Fall können zwar viele Kommandos in Großschreibung eingegeben und ausgeführt werden, es können jedoch keine Dateien editiert werden.

Paßwort

Danach fordert das System zur Eingabe des Paßwortes auf. Das Paßwort eingeben und die RETURN-Taste drücken. Aus Sicherheitsgründen wird das Paßwort nicht auf dem Bildschirm angezeigt.

Sind sowohl Benutzername als auch Paßwort für das UNIX-System gültig, gibt das System die Tagesmeldung und/oder aktuelle Nachrichten - sofern vorhanden - und danach das Standard-Bereit-Zeichen (\square) aus. In der Tagesmeldung können beispielsweise vorgesehene Systemwartungsmaßnahmen stehen, in den aktuellen Nachrichten die Ankündigung eines neuen Systemwerkzeugs. Nach der Anmeldung sieht der Bildschirm etwa so aus:


```
login: zugvogel<CR>
password:
UNIX system news
$
```

Macht man bei der Anmeldung einen Tippfehler, gibt das UNIX-System die Meldung `login incorrect` auf dem Bildschirm aus. Dann erhält der Benutzer ein zweites Mal die Möglichkeit, sich anzumelden; dies wird durch eine neue Eingabeaufforderung `login:` angezeigt.

```
login: tugvogel<CR>
password:
login incorrect
login:
```

Die Anmeldeprozedur kann auch fehlschlagen, wenn die Übertragungsverbindung zwischen dem Terminal und dem UNIX-System unterbrochen wurde. Dann muß die Verbindung mit dem Rechner erneut hergestellt werden (genauer gesagt, die Verbindung mit der Vermittlungseinrichtung, über die das Terminal an den Rechner angeschlossen ist) bevor man erneut versucht, sich anzumelden. Da dieses Verfahren von Standort zu Standort unterschiedlich ist, sollte man sich an den Systemverwalter wenden, um Genaueres über eine Verbindung mit der Vermittlungseinrichtung zu erfahren.

Hat man sich bisher noch nicht am UNIX-System angemeldet, unterscheidet sich das Verfahren unter Umständen von dem hier beschriebenen. Dies ist der Fall, wenn der Systemverwalter die wahlweise einsetzbare Sicherheitseinrichtung aktiviert hat, mit der neuen Benutzern vorläufige Kennwörter zugewiesen werden, wenn sie ihre Benutzerkonten einrichten. Hat man ein vorläufiges Paßwort, verlangt das System, daß man ein neues Paßwort wählt, bevor man sich am System anmelden kann.

Durch dieses obligatorische Auswählen eines ausschließlich für diesen Benutzer gültigen Paßwortes wird die Systemsicherheit erhöht. Der Schutz der Betriebsmittel des Systems und der persönlichen Dateien hängt davon ab, daß man sein Paßwort geheimhält.

Das tatsächliche Verfahren bei der ersten Anmeldung richtet sich nach der geübten Praxis der Systemverwaltung am betreffenden Standort. Es wird jedoch dem folgenden Beispiel sehr ähnlich sein.

1. Die Verbindung wird hergestellt, und das UNIX-System zeigt die Eingabeaufforderung `login:` an. Den Benutzernamen eingeben und die RETURN-Taste drücken.
2. Das UNIX-System gibt die Eingabeaufforderung für das Paßwort aus. Das vorläufige Paßwort eingeben und die RETURN-Taste drücken.
3. Das System meldet, daß das vorläufige Paßwort nicht mehr gültig ist, und daß ein neues definiert werden muß.
4. Das System fordert dazu auf, das alte Paßwort nochmals einzugeben. Das vorläufige Paßwort eingeben.
5. Das System fordert zur Eingabe des neuen Paßworts auf. Das gewählte Paßwort eingeben.

Kennwörter müssen die folgenden Voraussetzungen erfüllen:

- Ein Paßwort muß aus mindestens sechs Zeichen bestehen. Bei längeren Kennwörtern werden nur die ersten acht Zeichen geprüft.
- Ein Paßwort muß mindestens zwei alphabetische Zeichen und mindestens ein numerisches oder ein Sonderzeichen enthalten. Die alphabetischen Zeichen können Klein- oder Großbuchstaben sein.
- Ein Paßwort darf weder identisch mit dem Benutzernamen sein noch eine Umkehrung oder Umstellung desselben darstellen. Wenn das System die Einhaltung dieser Regel prüft, werden Groß- und Kleinbuchstaben als gleichwertig interpretiert.

- Ein neues Paßwort muß sich durch mindestens drei Zeichen vom alten Paßwort unterscheiden. Auch für diesen Vergleich werden Groß- und Kleinbuchstaben als gleichwertig interpretiert.

Beispiele gültiger Kennwörter sind: **apr84il**, **Johann0s** und **BRAV3S**.

NOTE

Bei bestimmten UNIX-Systemen können für die Auswahl eines Paßwortes andere Anforderungen bestehen. Genaue Informationen sind beim Systemverwalter zu erfragen.

6. Zur Kontrolle fordert das System den Benutzer auf, das neue Paßwort zu wiederholen. Das neue Paßwort nochmals eingeben.
7. Wird das neue Paßwort bei der Wiederholung nicht genauso wie beim ersten Mal angegeben, meldet das System, daß die Kennwörter nicht übereinstimmen und fordert dazu auf, die Eingabe nochmals zu wiederholen. Bei manchen Systemen kann jedoch die Übertragungsverbindung abgebrochen werden, wenn das Paßwort nicht genauso wie beim ersten Mal eingegeben wird. In diesem Fall muß das Anmeldeverfahren wieder von vorne (Schritt 1) begonnen werden. Stimmen die Kennwörter überein, erscheint das Bereit-Zeichen.

Auf dem folgenden Beispielbildschirm wird das Verfahren (Schritt 1 bis 6) für Erstbenutzer des UNIX-Systems zusammengefaßt.

```
login: zugvogel <CR>
password: <CR>
Your password has expired.
Choose a new one.
Old password: <CR>
New password: <CR>
Re-enter new password: <CR>
$
```

Probleme beim Anmelden

Ein Terminal wird in der Regel wie erwartet reagieren, sofern es richtig konfiguriert wurde. Unter Umständen können jedoch unerwartete Reaktionen auftreten, wie zum Beispiel, daß der Zeilenvorschub (CR) nicht richtig arbeitet.

Einige Probleme können einfach durch Abmelden und erneutes Anmelden behoben werden. Besteht ein Problem nach dem zweiten Anmeldevorgang weiter, sollten folgende Einstellungen überprüft werden, bevor man sich ein drittes Mal anmeldet:

Datentelefon	Ist das Terminal über Telefonleitung an den Rechner
bzw. Modem	angeschlossen, ist zu prüfen, ob die Baudrate und der Duplex-Modus richtig angegeben sind.
Schalter	Bei bestimmten Terminals müssen einige Schalter so eingestellt sein, daß sie mit dem UNIX-System kompatibel sind. Arbeitet man mit solch einem Terminal, sind die Einstellungen zu überprüfen.

Informationen zur Überprüfung der Terminalkonfiguration sind im Abschnitt "Terminalkonfiguration" in diesem Kapitel enthalten. Weitere Informationen über Tastatur, Terminal, Datentelefon und Modem sind in den Handbüchern zu den entsprechenden Geräten zu finden.

In Abbildung 2-5 werden einige Verfahren dargestellt, mit denen man bestimmte Probleme bei der Anmeldung erkennen, interpretieren und beheben kann. Reicht dies nicht aus, ist der Systemverwalter zu Rate zu ziehen.

Problem†	Mögliche Ursache	Maßnahme
Sinnlose Zeichen	Falsche Baudrate am UNIX-System RETURN- oder BREAK-Taste drücken	
Eingabe/Ausgabe erscheint in GROSSBUCHSTABEN	Terminalkonfiguration auf GROSSBUCHSTABEN eingestellt	Abmelden, dann Zeichengenerierung auf Kleinbuchstaben setzen
Eingabe erscheint in GROSS-, Ausgabe in Kleinschreibung	CAPS- (bzw. CAPS LOCK-)Taste ist aktiviert	CAPS bzw. CAPS LOCK drücken
Eingabe erscheint doppelt	Terminal ist auf HALBDUPLEX (HALF DUPLEX) eingestellt	Auf VOLLDUPLEX (FULL DUPLEX) stellen
Tabulatortaste arbeitet nicht richtig	Tabulatoren falsch eingestellt	<code>stty -tabs‡</code> eingeben
Übertragungsverbindung kommt nicht zustande, obwohl hochfrequenter Ton beim Anwählen hörbar ist	Terminal ist auf LOCAL oder OFF-LINE eingestellt.	Terminal auf ON-LINE einstellen; erneut anmelden
Übertragungsverbindung (vom Terminal zum UNIX-System) wird wiederholt unterbrochen	Schwache Telefonleitung oder defekter Datenübertragungskanal	Systemverwalter heranziehen

* Viele Probleme sind auf falsche Konfiguration der Terminals zurückzuführen. Um diese Möglichkeit auszuschließen, sind die Konfigurationsprüfungen unter "Terminalkonfiguration" vor dem Anmeldeversuch durchzuführen.

† Bestimmte Probleme sind dem verwendeten Terminal, Datentelefon oder Modem zuzuschreiben. Kann das Problem mit den vorgeschlagenen Maßnahmen nicht behoben werden, ist im Benutzerhandbuch des betreffenden Gerätes nachzuschlagen.

‡ Die Eingabe von `stty -tabs` berichtigt die Tabulatordefinition nur für die aktuelle Sitzung. Um die Tabulatordefinition für alle weiteren Sitzungen zu korrigieren, ist die Zeile `stty -tabs` in die Datei `.profile` einzufügen (siehe Kapitel 7).

Abbildung 2-5: Behebung von Fehlern bei der Anmeldung*

Einfache Kommandos

Erscheint das Bereit-Zeichen auf dem Bildschirm, dann hat das UNIX-System den Benutzer als berechtigten Benutzer identifiziert und erwartet die Eingabe eines Kommandos zum Ausführen eines Programms.

Beispielsweise kann man nun das Kommando `date` ausführen. Dazu ist das Kommando neben dem Bereit-Zeichen einzugeben und dann die RETURN-Taste zu drücken. Daraufhin greift das Betriebssystem UNIX auf das Programm mit dem Namen `date` zu, führt es aus und gibt das Ergebnis wie im folgenden dargestellt auf dem Bildschirm aus.

```
$ date<CR>
Wed Apr 26 16:26:28 MES 1989
$
```

Mit dem Kommando `date` kann man sich also das Datum und die Uhrzeit im 24-Stunden-Format ausgeben lassen.

Nun das Kommando `who` eingeben und die RETURN-Taste drücken. Auf dem Bildschirm erscheint dann etwa folgende Ausgabe:

```
$ who<CR>
zugvogel    tty000      Apr 26 12:00
jjb         tty011      Apr 26 12:43
petra       tty030      Apr 26 07:08
hgf         tty060      Apr 26 10:48
vera        tty090      Apr 26 15:20
$
```

Mit dem Kommando **who** werden die Benutzernamen aller Benutzer aufgelistet, die gerade an demselben System arbeiten. Die tty-Angaben beziehen sich auf die Gerätedateien, die den Terminals der einzelnen Benutzer zugeordnet sind. Außerdem werden Datum und Uhrzeit der Anmeldung der einzelnen Benutzer angezeigt.

Abmeldung

Am Ende einer Sitzung mit dem UNIX-System ist neben dem Bereit-Zeichen das Steuerzeichen `<^d>` einzugeben (CONTROL-Taste gedrückt halten und den Buchstaben d drücken; als nicht druckbares Steuerzeichen erscheint es nicht auf dem Bildschirm). Nach wenigen Sekunden zeigt das UNIX-System wieder die Eingabeaufforderung `login: an`.

```
$ <^d>
login:
```

Damit wird angezeigt, daß die Abmeldung ordnungsgemäß erfolgte und daß sich ein anderer Benutzer am System anmelden kann.

NOTE

Vor dem Ausschalten des Terminals bzw. Einhängen des Telefons sollte man sich in jedem Fall durch Eingabe von `<^d>` abmelden, da man sonst unter Umständen nicht wirklich abgemeldet ist.

Eine weitere Möglichkeit zum Abmelden bietet das Kommando `exit`, der allerdings nur von wenigen Benutzern verwendet wird. Es bietet sich an, wenn man in einem Shell-Programm ein Kommando zum Abmelden verwenden will. Nähere Angaben dazu sind im Abschnitt über Sonderkommandos unter `sh(1)` im *User's Reference Manual* zu finden.

Kapitel 3: BENUTZUNG DES DATEISYSTEMS

Einführung	3-1
Struktur des Dateisystems	3-2
Die Position des Benutzers im Dateisystem	3-4
Das Home-Verzeichnis	3-4
Das aktuelle Verzeichnis	3-6
Pfadnamen	3-7
Vollständige Pfadnamen	3-8
Relative Pfadnamen	3-11
Benennen von Verzeichnissen und Dateien	3-16
Organisation eines Verzeichnisses	3-17
Anlegen eines Verzeichnisses: das Kommando mkdir	3-17
Auflisten des Inhalts eines Verzeichnisses: das Kommando ls	3-19
Häufig verwendete Optionen zu ls	3-22
Verzeichnis wechseln: das Kommando cd	3-27
Verzeichnis löschen: das Kommando rmdir	3-29
Zugriff auf Dateien und ihre Bearbeitung	3-32
Grundlegende Kommandos	3-32
Ausgabe des Inhalts einer Datei: die Kommandos cat , pg und pr	3-34
Papierausdruck einer Datei anfordern: das Kommando lp	3-48
Erstellen einer Kopie einer Datei: das Kommando cp	3-51
Verschieben und Umbenennen einer Datei: das Kommando mv	3-54
Löschen einer Datei: das Kommando rm	3-57
Zeilen, Wörter und Zeichen einer Datei zählen: das Kommando wc	3-59

Dateien schützen: das Kommando chmod	3-62
Kommandos für Fortgeschrittene	3-70
Unterschiede zwischen Dateien feststellen: das Kommando diff	3-70
Zeichenmuster in einer Datei suchen: das Kommando grep	3-72
Sortieren und Mischen von Dateien: das Kommando sort	3-74

Zusammenfassung	3-78
-----------------	------

Einführung

Um effizient mit dem UNIX-Dateisystem arbeiten zu können, muß der Benutzer mit der Struktur vertraut sein, um sich orientieren zu können. In diesem Kapitel wird der Benutzer in den Umgang mit dem Dateisystem eingeführt.

In den ersten beiden Abschnitten ("Struktur des Dateisystems" und "Die Position des Benutzers im Dateisystem") wird das Dateisystem aus der Sicht des Benutzers während der Arbeit behandelt. In den übrigen Abschnitten des Kapitels werden die UNIX-Systemkommandos beschrieben, mit denen man ein eigenes Verzeichnis einrichten, auf die in ihm organisierten Unterverzeichnisse und Dateien zugreifen und sie verwalten kann sowie den Inhalt von anderen Verzeichnissen, auf die ein Zugriffsrecht besteht, einsehen kann.

Alle Kommandos werden in gesonderten Unterabschnitten erläutert. Am Ende dieser Unterabschnitte befinden sich Tabellen, in denen die Merkmale der Kommandos nochmals zusammengefaßt werden, so daß der Benutzer später schnell die Syntax und die Funktionen der Kommandos nachschlagen kann. Viele der hier vorgestellten Kommandos verfügen noch über weitere leistungsfähige Funktionen, die jedoch in der Regel nur von erfahrenen Benutzern eingesetzt werden und daher in einer anderen Dokumentation des UNIX-Systems beschrieben werden. Die hier eingeführten Kommandos bilden die Grundlage für eine effektive Nutzung des Dateisystems. Die einzelnen Kommandos sollten beim Durcharbeiten stets auch ausprobiert werden.

Struktur des Dateisystems

Das Dateisystem besteht aus einer Reihe von normalen Dateien, Gerätedateien und Verzeichnissen. Mit Hilfe dieser Elemente können Informationen elektronisch organisiert, abgerufen und verwaltet werden. In Kapitel 1 wurden die Eigenschaften von Verzeichnissen und Dateien beschrieben. Diese werden im folgenden Abschnitt nochmals kurz zusammengefaßt; danach wird beschrieben, wie Verzeichnisse und Dateien verwendet werden.

- Eine normale Datei ist eine Sammlung von Zeichen, die auf einem externen Speichermedium gespeichert sind. Sie kann den Text eines Berichts oder den Code eines Programms enthalten.
- Eine Gerätedatei steht für ein physikalisches Gerät wie zum Beispiel ein Terminal oder eine Platte.
- Ein Verzeichnis ist eine Sammlung von Dateien und anderen Verzeichnissen, die dann auch als Unterverzeichnisse bezeichnet werden. In einem Verzeichnis können Dateien nach beliebigen Kriterien zusammengefaßt werden. Beispielsweise kann ein Verzeichnis für jedes von einer Firma verkaufte Produkt oder für jeden Kunden angelegt werden.

Die Gesamtheit aller Verzeichnisse und Dateien ist in einer Struktur organisiert, die die Form eines Baumes hat. Abbildung 3-1 ist ein Beispiel einer Dateistruktur, das von einem Verzeichnis mit dem Namen Root (/) ausgeht. Indem man den vom Root-Verzeichnis ausgehenden Verzweigungen folgt, gelangt man in weitere wichtige Systemverzeichnisse. Folgt man wiederum den Verzweigungen aus diesen, kann man jedes Verzeichnis und jede Datei im Dateisystem erreichen.

Innerhalb dieser Hierarchie stehen die Dateien und (Unter-)Verzeichnisse in einer sogenannten Eltern/Kind-Beziehung zueinander. Diese Form der Beziehung zwischen Dateien und Verzeichnissen besteht auf vielen Ebenen. Tatsächlich gibt es keine Begrenzung der Anzahl der Dateien und Verzeichnisse, die in einem Verzeichnis, auf das man Zugriff hat, angelegt werden können. Ebenfalls unbegrenzt ist die Anzahl der Verzeichnisebenen. Die Dateien können daher in verschiedensten Formen nach dem Beispiel von Abbildung 3-1 organisiert werden.

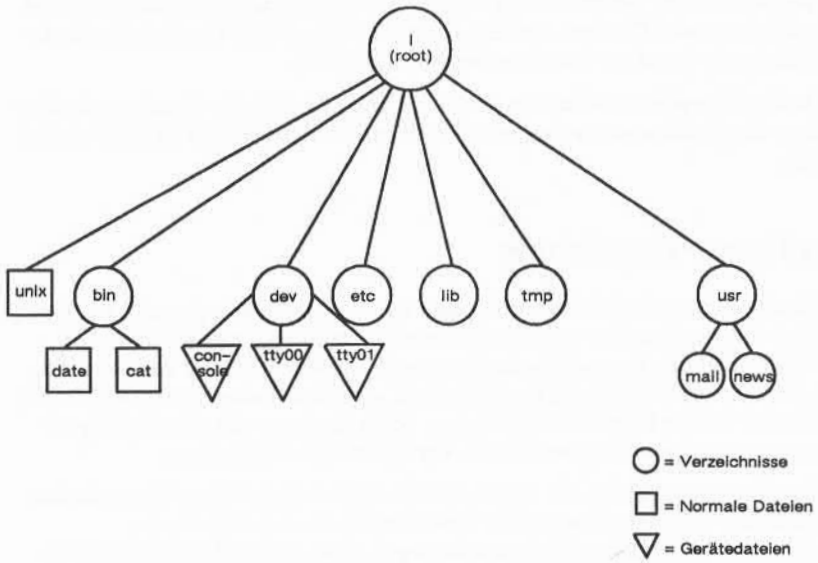


Abbildung 3-1: Beispiel eines Dateisystems

Die Position des Benutzers im Dateisystem

Der Benutzer befindet sich bei der Arbeit mit dem UNIX-System stets an einer bestimmten Stelle in der Struktur des Dateisystems. Das UNIX-System bringt ihn beim Anmelden automatisch an diese Stelle. Von dort aus kann sich der Benutzer in der Hierarchie des Dateisystems bewegen und mit allen eigenen Verzeichnissen und Dateien arbeiten, oder die Dateien und Verzeichnisse anderer Mitarbeiter benutzen, sofern er dazu berechtigt ist.

In den folgenden Abschnitten wird die eigene Position des Benutzers in der Struktur des Dateisystems erläutert, die sich ändert, wenn er sich im Dateisystem bewegt.

Das Home-Verzeichnis

Nachdem die erforderlichen Eingaben für die Anmeldung gemacht wurden, befindet sich der Benutzer an einer bestimmten Stelle des Dateisystems, dem sogenannten Home-Verzeichnis des Benutzers. Üblicherweise ist der Benutzername, der bei der Einrichtung des Benutzerkontos dem Benutzer zugeteilt wurde, auch der Name des Home-Verzeichnisses. Jeder Benutzer mit einem gültigen Benutzernamen hat ein eigenes Home-Verzeichnis im Dateisystem.

Zur Verwaltung durch das UNIX-System werden diese Home-Verzeichnisse in einem oder mehreren Systemverzeichnissen organisiert. Die Home-Verzeichnisse für die Benutzernamen **zugvogel**, **petra** und **jbb** können beispielsweise in einem Systemverzeichnis mit dem Namen **benutzer1** stehen. Abbildung 3-2 zeigt die relative Position eines Systemverzeichnisses wie **benutzer1** in Bezug auf die übrigen wichtigen UNIX-Systemverzeichnisse, die in Kapitel 1 erläutert wurden.

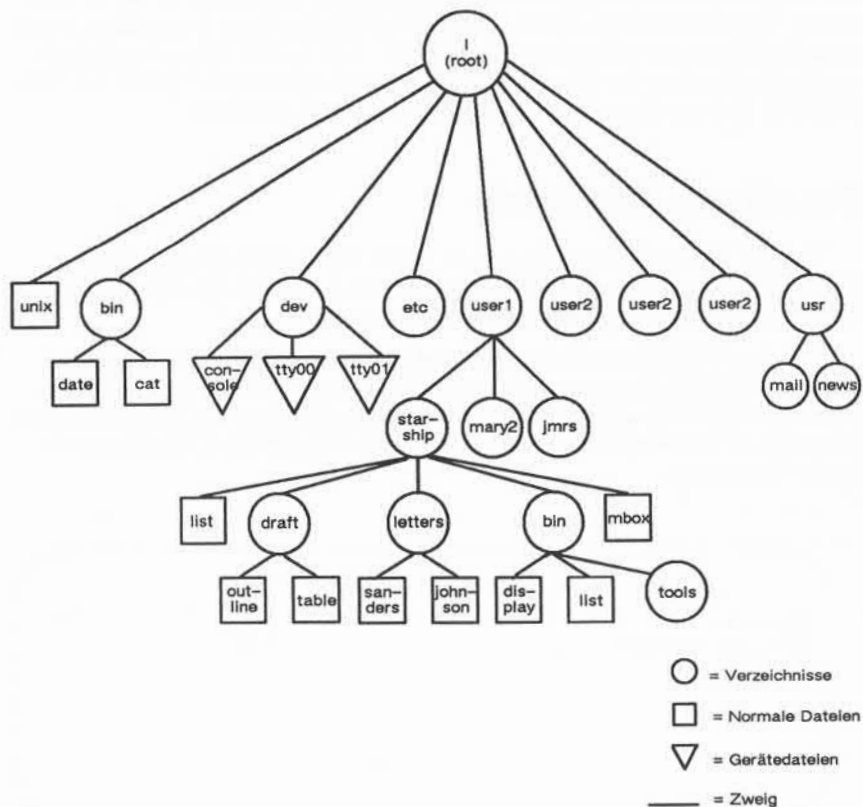


Abbildung 3-2: Verzeichnis der Home-Verzeichnisse

Innerhalb seines Home-Verzeichnisses kann der Benutzer Dateien erstellen und weitere Verzeichnisse (bzw. Unterverzeichnisse) anlegen, in denen sie nach bestimmten Kriterien zusammengefaßt werden. Dateien und Verzeichnisse können verschoben oder gelöscht werden, und man kann Zugriffsrechte auf sie definieren. Für alle Einheiten, die der Benutzer in seinem Home-Verzeichnis anlegt, ist er als Eigentümer selbst voll verantwortlich. Von seinem Home-Verzeichnis aus hat der Benutzer den Überblick über alle darin enthaltenen Dateien und Verzeichnisse und das übrige Dateisystem bis zum Root-Verzeichnis.

Das aktuelle Verzeichnis

Solange sich der Benutzer in seinem Home-Verzeichnis befindet, ist es gleichzeitig sein aktuelles Arbeitsverzeichnis. Wechselt er in ein anderes Verzeichnis, so wird dieses das neue aktuelle Verzeichnis.

Mit dem UNIX-Systemkommando `pwd` ("print working directory") wird der Name des Verzeichnisses angezeigt, in dem man gerade arbeitet. Hat ein Benutzer beispielsweise den Benutzernamen `zugvogel` und führt er das Kommando `pwd` beim ersten Bereit-Zeichen nach dem Anmelden aus, gibt das UNIX-System folgende Antwort aus:

```
$ pwd <CR>
/benutzer1/zugvogel
$
```

Das System zeigt dabei den Namen des Verzeichnisses an, in dem man gerade arbeitet (`zugvogel`), und die Position dieses Verzeichnisses im Dateisystem. Der Pfadname `/benutzer1/zugvogel` gibt dabei an, daß das Verzeichnis `benutzer1` im Home-Verzeichnis (angegeben durch `/` vor der Zeile) steht und das Verzeichnis `zugvogel` enthält. Die weiteren Schrägstriche im Pfadnamen dienen zum Trennen der Verzeichnis- und Dateinamen und zum Anzeigen der

relativen Position der Verzeichnisse gegenüber dem Root-Verzeichnis. Ein Verzeichnisname, der die Position des Verzeichnisses auf diese Weise angibt, wird vollständiger Verzeichnisname oder Pfadname genannt. Auf den folgenden Seiten wird dieser Pfadname erläutert; damit kann der Benutzer beginnen, sich im Dateisystem zu bewegen.

Wie bereits erwähnt, kann man die eigene Position im Dateisystem jederzeit feststellen, indem man das Kommando **pwd** eingibt. Dies ist besonders nützlich, wenn man eine Datei lesen oder kopieren will und das UNIX-System meldet, daß die gesuchte Datei nicht existiert. Dies kann vorkommen, wenn man sich in einem anderen Verzeichnis befindet, als man vermutet hatte.

In Abbildung 3-3 werden die Syntax und die Funktionen des Kommandos **pwd** zusammengefaßt.

Kommandoübersicht		
pwd – Vollständigen Namen des Arbeitsverzeichnisses anzeigen (print working directory)		
<i>Kommando</i>	<i>Optionen</i>	<i>Argumente</i>
pwd	keine	keine
Beschreibung:	Mit pwd wird der vollständige Pfadname des Verzeichnisses ausgegeben, in dem man gerade arbeitet.	

Abbildung 3-3: Übersicht über das Kommando **pwd**

Pfadnamen

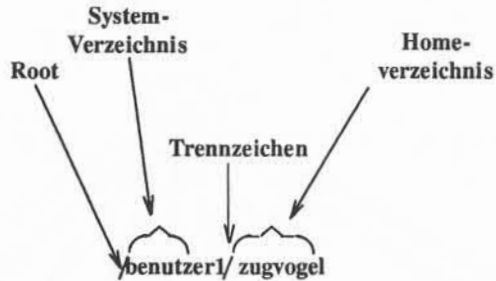
Jede Datei und jedes Verzeichnis im UNIX-System wird mit einem eindeutigen Pfadnamen identifiziert. Der Pfadname zeigt die Position der Datei bzw. des Verzeichnisses an und weist gleichzeitig darauf hin, wie man sie bzw. es erreicht. Grundsätzlich gibt es zwei Typen von Pfadnamen: vollständige und relative Pfadnamen.

Vollständige Pfadnamen

Ein vollständiger Pfadname (auch absoluter Pfadname genannt) enthält Richtungsangaben, die mit dem Root-Verzeichnis beginnen und über eine eindeutige Folge von Verzeichnisnamen zu einem bestimmten Verzeichnis bzw. einer Datei führen. Mit einem vollständigen Pfadnamen erreicht man von einer beliebigen Stelle des Dateisystems aus jede Datei bzw. jedes Verzeichnis.

Da ein vollständiger Pfadname stets mit dem Root-Verzeichnis beginnt, steht immer ein Schrägstrich (/) an der ersten Stelle des Pfadnamens. Der letzte Name eines vollständigen Pfadnamens kann ein Dateiname oder ein Verzeichnisname sein. Alle weiteren Namen der Pfadangabe sind Verzeichnisnamen.

Der Aufbau eines Pfadnamens sowie die Orientierung, die er ermöglicht, werden an folgendem Beispiel deutlich: Angenommen, man arbeitet im Verzeichnis **zugvogel**, das sich wiederum im Verzeichnis **/benutzer1** befindet. Man gibt das Kommando **pwd** ein, und das System gibt den vollständigen Pfadnamen des aktuellen Arbeitsverzeichnisses aus: **/benutzer1/zugvogel**. Die folgende Darstellung und die entsprechenden Erläuterungen beschreiben die einzelnen Elemente des Pfadnamens.



/ (vorangestellt)	= Der Schrägstrich, der als erstes Zeichen im Pfadnamen steht, gibt das Root-Verzeichnis des Dateisystems an.
benutzer1	= Das Systemverzeichnis auf der nächsten Ebene der Hierarchie unter dem Root-Verzeichnis, auf das Root verweist bzw. in das es verzweigt.
/ (nachgestellt)	= Der zweite Schrägstrich trennt die Verzeichnisnamen benutzer1 und zugvogel bzw. begrenzt sie.
zugvogel	= aktuelles Arbeitsverzeichnis

In Abbildung 3-4 wird dieser vollständige Pfad zum Verzeichnis **/benutzer1/zugvogel** durch fette Linien dargestellt.

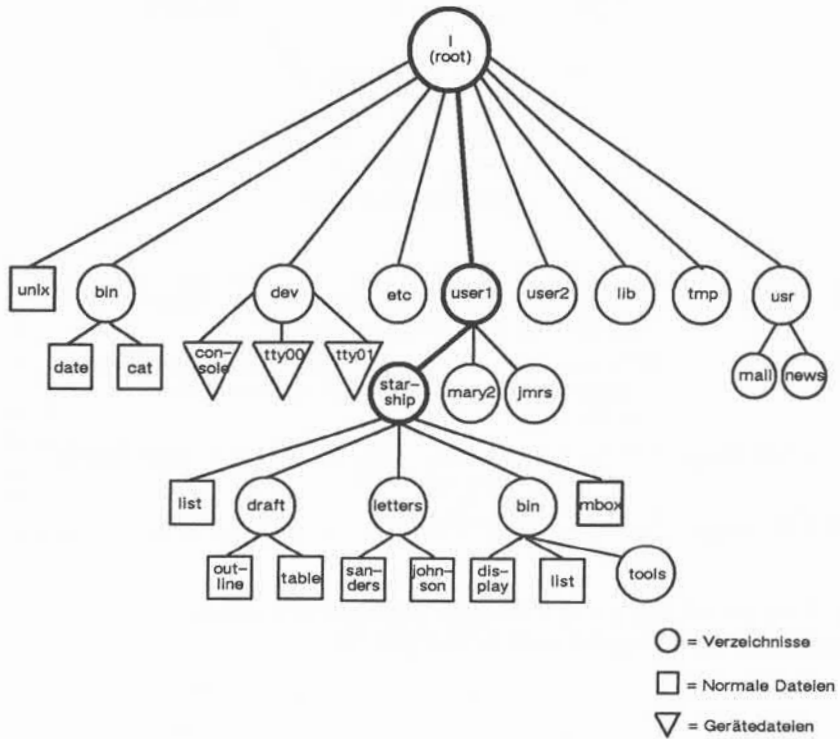


Abbildung 3-4: Vollständiger Pfadname des Verzeichnisses /benutzer1/zugvogel

Relative Pfadnamen

Ein relativer Pfadname gibt einen Weg an, der im aktuellen Arbeitsverzeichnis beginnt und nach oben oder unten in der Verzeichnishierarchie zu einer bestimmten Datei bzw. einem bestimmten Verzeichnis führt. Durch Wechseln des aktuellen Verzeichnisses kann man auf Dateien und Verzeichnisse zugreifen, deren Eigentümer man ist. Bewegt man sich vom aktuellen Verzeichnis aus nach oben, geht man über Ebenen von Elternverzeichnissen zum "Großelternverzeichnis" aller Systemverzeichnisse, d. h. zum Root-Verzeichnis. Von dort aus kann man jeden beliebigen Punkt des Dateisystems erreichen.

Ein relativer Pfadname beginnt mit einer der folgenden Angaben: einem Verzeichnis- oder Dateinamen; einem Punkt (.), der als Kürzel für das aktuelle Verzeichnis dient; oder zwei Punkten (..), die als Kürzel für das direkt über dem aktuellen Verzeichnis liegende Verzeichnis dienen. Dieses Verzeichnis (..) wird als Elternverzeichnis des aktuellen Verzeichnisses (.) bezeichnet.

Angenommen, man befindet sich im Verzeichnis **zugvogel** des Beispiel-Dateisystems, und **zugvogel** enthält Verzeichnisse mit den Namen **entwurf**, **briefe** und **bin** sowie eine Datei mit dem Namen **mbox**. Der relative Pfadname zu diesen Dateien besteht dann lediglich aus ihrem Namen, d. h. **entwurf** oder **mbox**. In Abbildung 3-5 wird der relative Pfad von **zugvogel** zu **entwurf** durch die fette Linie dargestellt.



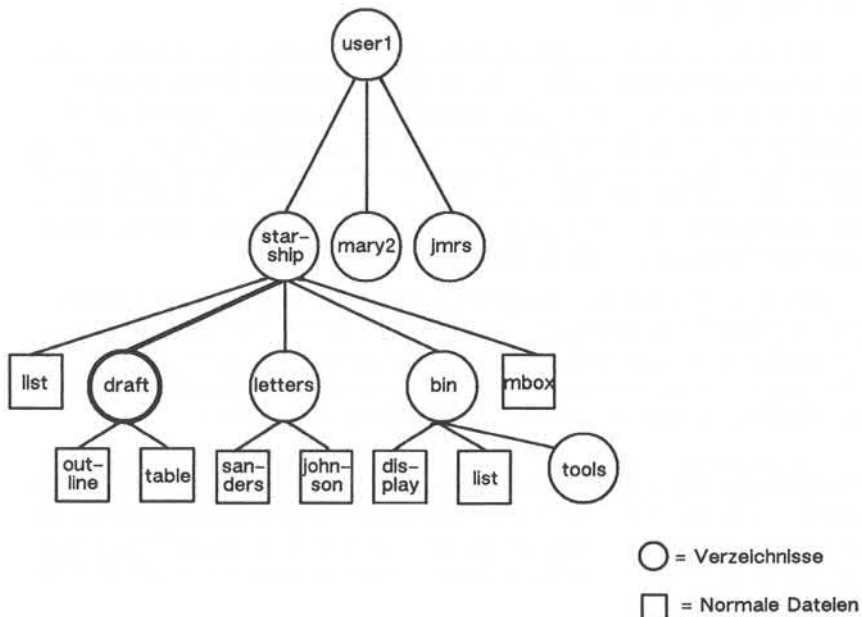


Abbildung 3-5: Relativer Pfadname des Verzeichnisses **entwurf**

Das Verzeichnis **entwurf** des Verzeichnisses **zugvogel** enthält die Dateien **gliederung** und **tabelle**. Der relative Pfadname von **zugvogel** zur Datei **gliederung** ist **entwurf/gliederung**.

Dieser relative Pfad wird in Abbildung 3-6 dargestellt. Der Schrägstrich im Pfadnamen trennt dabei das Verzeichnis **entwurf** von der Datei mit dem Namen **gliederung**. Der Schrägstrich dient als Trennstrich und gibt gleichzeitig an, daß **gliederung** dem Verzeichnis **entwurf** untergeordnet ist; d. h. **gliederung** ist ein Kind des Elternverzeichnisses **entwurf**.

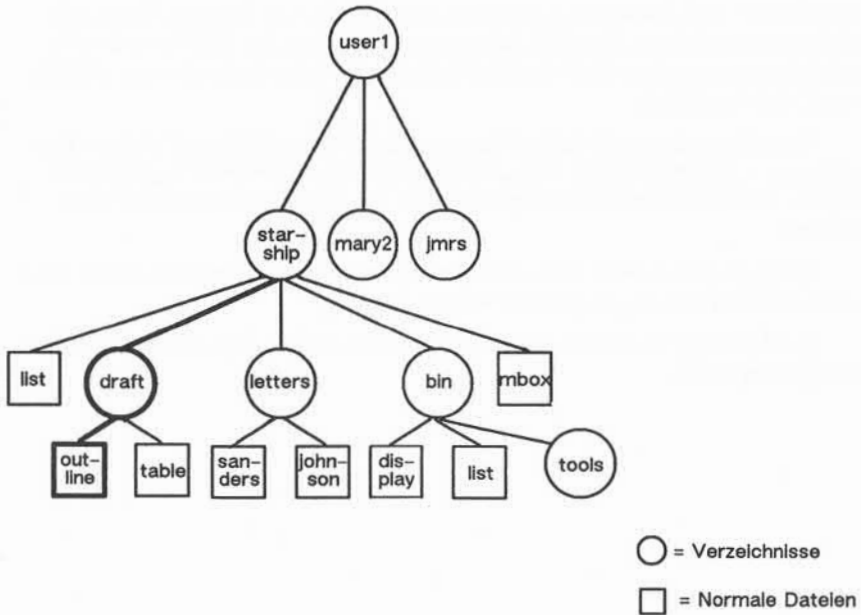


Abbildung 3-6: Relativer Pfadname von zugvogel zu gliederung

Bis hierher wurden die relativen Pfadnamen in bezug auf Dateien behandelt, die im aktuellen Verzeichnis stehen oder diesem untergeordnet sind. Damit ist deutlich geworden, wie man sich in der Systemhierarchie Ebene für Ebene nach unten bewegen kann, bis man sich an der gewünschten Stelle befindet. Man kann sich jedoch auch in der Systemstruktur nach oben bewegen bzw. zuerst nach oben und dann auf einem anderen Pfad wieder nach unten zu anderen Dateien und Verzeichnissen.

Das Elternverzeichnis des aktuellen Verzeichnisses erreicht man durch die Angabe von zwei Punkten (..). Befindet man sich also im Verzeichnis mit dem Namen **entwurf** dieses Beispiel-Dateisystems, dann ist .. der Pfadname des Verzeichnisses **zugvogel**, und ../.. ist der Pfadname des Elternverzeichnisses von **zugvogel**, d. h. **benutzer1**.

Vom Verzeichnis **entwurf** aus kann man auch einen Pfad zur Datei **sanders** angeben: ../**briefe/sanders**. Mit .. erreicht man das Verzeichnis **zugvogel**; die Namen **briefe** und **sanders** führen dann über das Verzeichnis **briefe** zur Datei **sanders**.

Dabei ist zu beachten, daß anstelle eines relativen Pfadnamens stets auch ein vollständiger Pfadname angegeben werden kann.

In Abbildung 3-7 werden einige Beispiele für vollständige und relative Pfadnamen dargestellt.

Pfadname	Bedeutung
/	Vollständiger Pfadname des Root-Verzeichnisses
/bin	Vollständiger Pfadname des Verzeichnisses bin (enthält die meisten ausführbaren Programme und Dienstprogramme)
/benutzer1/zugvogel/bin/tools	Vollständiger Pfadname des Verzeichnisses tools . Es steht im Verzeichnis bin , das wiederum im Verzeichnis zugvogel steht, welches seinerseits in benutzer1 steht, das wiederum zum Root-Verzeichnis gehört.
bin/tools	Relativer Pfadname zur Datei bzw. zum Verzeichnis tools im Verzeichnis bin . Ist / das aktuelle Verzeichnis, sucht das UNIX-System nach /bin/tools . Ist jedoch zugvogel das aktuelle Verzeichnis, sucht das System auf dem vollständigen Pfad /benutzer1/zugvogel/bin/tools .
tools	Relativer Pfadname der Datei bzw. des Verzeichnisses tools im aktuellen Verzeichnis.



Abbildung 3-7: Beispiele für Pfadnamen

Bevor man sich im Dateisystem einigermaßen sicher mit diesen Pfadnamen bewegen kann, bedarf es einiger Übung.

Benennen von Verzeichnissen und Dateien

Der Benutzer kann seinen Verzeichnissen und Dateien beliebige Namen geben, vorausgesetzt, er hält die folgenden Regeln ein:

- Der Name eines Verzeichnisses oder einer Datei darf ein bis vierzehn Zeichen lang sein.
- Alle Zeichen mit Ausnahme von / sind zulässig.
- Bestimmte Zeichen möglichst vermeiden, wie zum Beispiel Leerzeichen, Tabulatoren, Rücktaste (BACKSPACE) und folgende Zeichen:

? @ # \$ ^ & * () ` [] \ | ; ' " < >

Wird ein Leerzeichen oder ein Tabulatorzeichen im Verzeichnis- oder Dateinamen verwendet, muß der Name in der Kommandozeile stets in Anführungszeichen angegeben werden.

- Die Zeichen +, - oder . sollten nicht als erste Zeichen eines Dateinamens verwendet werden.
- Das UNIX-System unterscheidet Groß- und Kleinbuchstaben. Ein Verzeichnis oder eine Datei mit dem Namen **entwurf** wird von einer Datei mit dem Namen **ENTWURF** unterschieden.

Im folgenden einige Beispiele zulässiger Verzeichnis- bzw. Dateinamen:

```
info INFO abschn2 bez:liste
datei.d kap3+4 pkt1-10 gliederung
```

Im weiteren Verlauf dieses Kapitels werden UNIX-Systemkommandos eingeführt, mit denen man das Dateisystem verwalten kann.

Organisation eines Verzeichnisses

In diesem Abschnitt werden vier UNIX-Systemkommandos eingeführt, mit denen eine Verzeichnisstruktur aufgebaut und verwendet werden kann: **mkdir**, **ls**, **cd** und **rmdir**.

mkdir	ermöglicht das Anlegen von neuen Verzeichnissen bzw. Unterverzeichnissen im aktuellen Verzeichnis.
ls	dient zum Auflisten der Namen aller Unterverzeichnisse und Dateien in einem Verzeichnis.
cd	ermöglicht dem Benutzer, im Dateisystem von einem Verzeichnis in ein anderes zu wechseln.
rmdir	dient zum Löschen eines leeren Verzeichnisses.

Diese Kommandos können mit vollständigen oder relativen Pfadnamen ausgeführt werden. Zwei dieser Kommandos, **ls** und **cd**, arbeiten auch ohne Angabe eines Pfadnamens. Die einzelnen Kommandos werden in den folgenden vier Abschnitten ausführlicher beschrieben.

Anlegen eines Verzeichnisses: das Kommando **mkdir**

Unterverzeichnisse des Home-Verzeichnisses sollten in einer logischen und zweckmäßigen Struktur angelegt werden, um das Abrufen von Informationen aus den Dateien zu erleichtern. Stellt man alle Dateien zu einem Thema zusammen in ein Verzeichnis, sind sie später leichter wieder aufzufinden.

Zum Anlegen eines Verzeichnisses dient das Kommando **mkdir** (Abkürzung für "make directory"). Dazu ist einfach der Kommandoname mit dem Namen anzugeben, den das neue Verzeichnis bzw. die Datei erhalten soll. Im Beispieldateisystem wurde das Unterverzeichnis **entwurf** beispielsweise von seinem Eigentümer durch Eingabe des folgenden Kommandos im Home-Verzeichnis (**/benutzer1/zugvogel**) angelegt:

```
$ mkdir entwurf <CR>
$
```

Das zweite Bereit-Zeichen zeigt an, daß das Kommando erfolgreich ausgeführt wurde, d. h. daß das Unterverzeichnis **entwurf** angelegt wurde.

Ebenso hat dieser Benutzer weitere Unterverzeichnisse in seinem Home-Verzeichnis angelegt, wie zum Beispiel **briefe** und **bin**.

```
$ mkdir briefe <CR>
$ mkdir bin <CR>
$
```

Man kann auch alle drei Unterverzeichnisse (**entwurf**, **briefe** und **bin**) gleichzeitig anlegen, indem man sie zusammen in einer Kommandozeile angibt.

```
$ mkdir entwurf briefe bin <CR>
$
```

Wechselt man in ein neu angelegtes Unterverzeichnis, kann man weitere Unterverzeichnisse in diesem anlegen. Beim Anlegen von Verzeichnissen oder Dateien kann man ihnen beliebige Namen zuweisen, sofern man die oben unter "Benennen von Verzeichnissen und Dateien" aufgeführten Richtlinien einhält.

In Abbildung 3-8 werden die Syntax und die Funktionen des Kommandos **mkdir** zusammengefaßt.

Kommandoübersicht		
mkdir – Neues Verzeichnis anlegen.		
<i>Kommando</i>	<i>Optionen</i>	<i>Argumente</i>
mkdir	keine	<i>Verzeichnisname(n)</i>
Beschreibung:	Mit mkdir wird ein neues Verzeichnis (Unterverzeichnis) angelegt.	
Bemerkungen:	Wurde das Verzeichnis erfolgreich angelegt, gibt das System ein Bereit-Zeichen (standardmäßig \$) aus.	

Abbildung 3-8: Übersicht über das Kommando **mkdir**

Auflisten des Inhalts eines Verzeichnisses: das Kommando ls

Alle Verzeichnisse des Dateisystems enthalten Informationen über die Dateien und Verzeichnisse, die in ihnen stehen, wie zum Beispiel Name, Größe und das Datum der letzten Änderung. Diese Informationen kann man sich anzeigen lassen - für das aktuelle Verzeichnis oder andere Systemverzeichnisse - indem man das Kommando `ls` (Abkürzung für "list") eingibt.

Mit dem Kommando `ls` werden die Namen aller Dateien und Unterverzeichnisse in einem bestimmten Verzeichnis aufgelistet. Wird kein Verzeichnis angegeben, werden durch `ls` die Namen der Dateien und Verzeichnisse des aktuellen Verzeichnisses ausgegeben. Zum besseren Verständnis der Arbeitsweise des Kommandos `ls` empfiehlt es sich das Beispiel-Dateisystem (Abbildung 3-2) nochmals zu betrachten.

Angenommen, der Benutzer ist am UNIX-System angemeldet und führt das Kommando `pwd` aus. Das System antwortet dann mit dem Pfadnamen `/benutzer1/zugvogel`. Will er sich die Namen der Dateien und Verzeichnisse in diesem (aktuellen) Verzeichnis anzeigen lassen, gibt er `ls` ein und drückt anschließend die RETURN-Taste. Nach dieser Eingabefolge erscheint folgendes auf dem Terminal:

```
$ pwd<CR>
$/benutzer1/zugvogel
$ ls<CR>
bin
entwurf
briefe
liste
mbox
$
```

Das System gibt also die Namen der Dateien und Verzeichnisse des aktuellen Verzeichnisses **zugvogel** in alphabetischer Reihenfolge aus (haben Datei- oder Verzeichnisnamen eine Zahl oder einen Großbuchstaben als erstes Zeichen, werden sie zuerst ausgegeben).

Sollen die Namen der Dateien und Unterverzeichnisse eines anderen Verzeichnisses angezeigt werden, ohne daß man das aktuelle Verzeichnis verläßt, muß der Name des betreffenden Verzeichnisses wie folgt angegeben werden:

```
ls Pfadname <CR>
```

Als Verzeichnisname kann hier entweder der vollständige oder der relative Pfadname des gewünschten Verzeichnisses angegeben werden. Der Inhalt von **entwurf** kann beispielsweise aus dem Verzeichnis **zugvogel** heraus durch Eingabe von **ls entwurf** mit anschließendem Drücken der RETURN-Taste angezeigt werden. Der Bildschirm sieht dann etwa folgendermaßen aus:

```
$ ls entwurf <CR>
gliederung
tabelle
$
```

Hier ist **entwurf** ein relativer Pfadname von einem Eltern- (**zugvogel**) zu einem Kindverzeichnis (**entwurf**).

Ein relativer Pfadname kann auch dazu verwendet werden, den Inhalt eines Elternverzeichnisses aufzulisten, wenn man sich in einem seiner Kindverzeichnisse befindet. Die Syntaxkonvention Punkt-Punkt (..) erleichtert dies. Mit der folgenden Kommandozeile wird beispielsweise der relative Pfadname von **zugvogel** zu **benutzer1** angegeben:

```
$ ls ..<CR>
jfb
petra
zugvogel
$
```

Dasselbe Ergebnis erhält man, wenn man den vollständigen Pfadnamen vom Root-Verzeichnis zum Verzeichnis **benutzer1** angibt. Gibt man also **ls /benutzer1** ein und drückt anschließend die RETURN-Taste, gibt das System dieselbe Liste aus.

Ebenso kann man den Inhalt jedes Systemverzeichnisses anzeigen lassen, auf das man Zugriff hat, indem man das Kommando **ls** mit einem vollständigen oder relativen Pfadnamen eingibt.

Das Kommando **ls** ist nützlich, wenn man eine lange Liste von Dateien hat und herausfinden möchte, ob eine davon im aktuellen Verzeichnis steht. Befindet man sich beispielsweise im Verzeichnis **entwurf** und möchte feststellen, ob die Dateien **gliederung** und **notiz** in diesem stehen, gibt man das Kommando **ls** wie folgt ein:

```
$ ls gliederung notiz<CR>
gliederung
notiz not found
$
```

Das System bestätigt hier, daß **gliederung** vorhanden ist, indem es seinen Namen auflistet, und meldet, daß die Datei **notiz** nicht gefunden wurde.

Das Kommando `ls` gibt nur den Namen und nicht den Inhalt einer Datei aus. Zum Anzeigen des Inhalts einer Datei dienen die Kommandos `cat`, `pg` oder `pr`. Sie werden unter "Zugriff auf Dateien und ihre Verwaltung" weiter unten in diesem Kapitel beschrieben.

Häufig verwendete Optionen zu `ls`

Zum Kommando `ls` gibt es auch Optionen, durch die die Liste von Dateien oder Unterverzeichnissen nach bestimmten Attributen zusammengestellt werden kann. Es gibt über ein Dutzend Optionen zum Kommando `ls`. Davon werden `-a` und `-l` wohl für den Einsteiger die nützlichsten sein. Einzelheiten zu den weiteren Optionen sind unter `ls(1)` im *User's Reference Manual* zu finden.

Auflisten aller Dateien in einem Verzeichnis

Einige wichtige Dateinamen im Home-Verzeichnis eines Benutzers, wie zum Beispiel `.profile` beginnen mit einem Punkt (`.`). Diese Dateinamen werden nicht mit den anderen Dateinamen angezeigt, die durch das Kommando `ls` aufgelistet werden. Sollen diese Dateien mit angezeigt werden, ist die Option `-a` in der Kommandozeile mit anzugeben.

Will man beispielsweise alle Dateien des aktuellen Verzeichnisses **zugvogel** einschließlich derer, die mit einem Punkt (`.`) beginnen, anzeigen lassen, muß man `ls -a` eingeben und anschließend die RETURN-Taste drücken.

```
$ ls -a <CR>
.
..
.profile
bin
entwurf
briefe
liste
mbox
$
```


Auflisten des Verzeichnisinhalts in Kurzformat

Die Optionen `-C` und `-F` des Kommandos `ls` werden häufig verwendet. Werden sie zusammen angegeben, werden die Unterverzeichnisse und Dateien eines Verzeichnisses im Spaltenformat ausgegeben; dabei werden ausführbare Dateien mit einem Stern (*) und Verzeichnisse mit einem Schrägstrich (/) identifiziert. So können alle Dateien des Arbeitsverzeichnisses **zugvogel** durch Ausführen der folgenden Kommandozeile aufgelistet werden:

```
$ ls -CF<CR>
bin/   briefe/ mbox
entwurf/   liste*
$
```

Auflisten des Verzeichnisinhalts in ausführlichem Format

Die meisten Informationen über ein Verzeichnis erhält man mit der Option `-l` des Kommandos `ls`; damit wird der Inhalt eines Verzeichnisses in "langem" Format ausgegeben, d. h. die Liste enthält den Modus, die Anzahl der Links (Verknüpfungen), Eigentümer, Gruppe, Größe in Byte und den Zeitpunkt der letzten Änderung aller Dateien. Führt man beispielsweise das Kommando `ls -l` aus dem Verzeichnis **zugvogel** aus, erscheint folgende Ausgabe:

```

$ ls -l<CR>
total 30
drwxr-xr-x  3 zugvogel  projekt    96 Oct 27  08:16 bin
drwxr-xr-x  2 zugvogel  projekt    64 Nov  1  14:19 entwurf
drwxr-xr-x  2 zugvogel  projekt    80 Nov  8  08:41 briefe
-rwx----- 2 zugvogel  projekt  12301 Nov  2  10:15 liste
-rw-----  1 zugvogel  projekt    40 Oct 27  10:00 mbox
$
    
```

Die erste Ausgabezeile (total 30) gibt an, wieviele Blöcke an Plattenspeicherplatz belegt sind. Die übrigen Zeilen enthalten die Informationen über die einzelnen Verzeichnisse und Dateien in **zugvogel**. Mit dem ersten Zeichen jeder Zeile (d, -, b oder c) wird der Dateityp angegeben.

- d = Verzeichnis
- = normale Datei
- b = blockorientierte Gerätedatei
- c = zeichenorientierte Gerätedatei

Dabei wird deutlich, daß das Verzeichnis **zugvogel** des obigen Bildschirms drei Verzeichnisse und zwei normale Dateien enthält.

Mit den beiden nächsten Zeichen, die entweder Buchstaben oder Bindestriche sein können, wird angezeigt, wer das Recht zum Lesen oder Benutzen der betreffenden Datei bzw. des Verzeichnisses hat. Zugriffsrechte werden im Zusammenhang mit dem Kommando **chmod** unter "Zugriff auf Dateien und ihre Verwaltung" weiter unten in diesem Kapitel behandelt.

Die darauffolgende Zahl ist die Anzahl der Links. Diese Zahl entspricht der Anzahl der Dateien, die die auf diese Datei verweisen. Bei einem Verzeichnis zeigt diese Zahl an, wieviele Verzeichnisse unmittelbar unter ihm angeordnet sind, zuzüglich zwei (für das Verzeichnis selbst und sein Elternverzeichnis).

Danach wird der Benutzername des Eigentümers der Datei angezeigt (hier zugvogel), gefolgt vom Gruppennamen der Datei bzw. des Verzeichnisses (projekt).

Die Zahl dahinter gibt die Länge des Datei- bzw. Verzeichniseintrags in Informationseinheiten bzw. Speichereinheiten an, d. h. in Byte. Anschließend erscheint Monat, Tag und Uhrzeit der letzten Änderung der Datei. In der letzten Spalte wird schließlich der Name des Verzeichnisses bzw. der Datei angezeigt.

In Abbildung 3-9 werden die einzelnen Spalten der Ausgabezeilen aus dem Kommando `ls -l` erläutert.

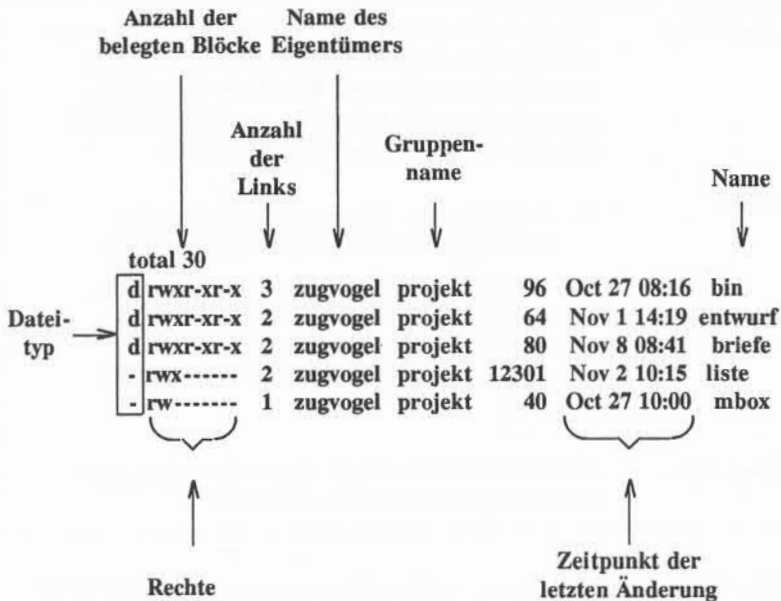


Abbildung 3-9: Erläuterung der Ausgabe des Kommandos `ls -l`

In Abbildung 3-10 werden die Syntax und die Funktionen des Kommandos `ls` und zweier seiner Optionen zusammengefaßt.

Kommandoübersicht		
ls - Auflisten des Inhalts eines Verzeichnisses.		
<i>Kommando</i>	<i>Optionen</i>	<i>Argumente</i>
ls	-a, -l, und andere*	<i>Verzeichnisname(n)</i>
Beschreibung:	Mit <code>ls</code> werden die Namen der Dateien und Unterverzeichnisse der mit dem Kommando angegebenen Verzeichnisse aufgelistet. Wird kein Verzeichnisname als Argument angegeben, wird der Inhalt des aktuellen Arbeitsverzeichnisses aufgelistet.	
Optionen:	<ul style="list-style-type: none"> -a Auflisten aller Einträge einschließlich der Einträge, die mit einem Punkt (.) beginnen. -l Auflisten des Inhalts eines Verzeichnisses in ausführlichem Format unter Angabe des Modus, der Zugriffsrechte, der Größe in Byte sowie dem Zeitpunkt der letzten Änderung. 	
Bemerkungen:	Soll der Inhalt einer Datei angezeigt werden, kann das Kommando <code>cat</code> verwendet werden.	

* Eine Liste aller verfügbaren Optionen zu diesem Kommando mit einer Erläuterung ihrer Funktionen ist unter `ls(1)` im *User's Reference Manual* zu finden.

Abbildung 3-10: Übersicht über das Kommando `ls`

Verzeichnis wechseln: das Kommando `cd`

Beim Anmelden zum UNIX-System wechselt das System zunächst in das Home-Verzeichnis des jeweiligen Benutzers. Solange der Benutzer darin arbeitet, ist es gleichzeitig sein Arbeitsverzeichnis. Mit dem Kommando `cd` (Abkürzung für "change directory") kann man in andere Verzeichnisse wechseln, um darin zu arbeiten. Dieses Kommando wird als `cd` zusammen mit dem Pfadnamen des Verzeichnisses eingegeben, in das man wechseln will.

```
cd Pfadname_neues_Verzeichnis <CR>
```

Als Argument für das Kommando `cd` ist jeder gültige Pfadname (vollständig oder relativ) zulässig. Wird kein Pfadname angegeben, wechselt das System in das Home-Verzeichnis des Benutzers. Nach dem Wechsel in ein anderes Verzeichnis wird dieses zum aktuellen Verzeichnis.

Will man beispielsweise vom Verzeichnis `zugvogel` in dessen Kindverzeichnis `entwurf` wechseln, ist `cd entwurf` einzugeben und anschließend die RETURN-Taste zu drücken (`entwurf` ist hier der relative Pfadname zu diesem Verzeichnis). Erscheint danach das Bereit-Zeichen, kann man überprüfen, ob man sich im anderen Verzeichnis befindet, indem man `pwd` eingibt und die RETURN-Taste drückt. Der Bildschirm des Terminals wird dann etwa folgendes anzeigen:

```
$ cd entwurf<CR>
$ pwd<CR>
/benutzer1/zugvogel/entwurf
$
```

In diesem Verzeichnis `entwurf` können nun Unterverzeichnisse mit dem Kommando `mkdir` angelegt oder neue Dateien mit den Editoren `ed` und `vi` erstellt werden (Anleitungen zu den Kommandos `ed` und `vi` sind in Kapitel 5 bzw. 6 enthalten).

Man muß sich nicht im Verzeichnis **entwurf** befinden, um auf Dateien zuzugreifen, die darin stehen. Eine Datei kann von jedem Verzeichnis aus durch Angabe ihres vollständigen Pfadnamens angegeben werden. Man kann beispielsweise die Datei **sanders** im Verzeichnis **briefe** (**/benutzer1/zugvogel/briefe**) mit dem Kommando **cat** auflisten, während man sich im Verzeichnis **entwurf** befindet (**/benutzer1/zugvogel/entwurf**), indem man den vollständigen Pfadnamen von **sanders** in der Kommandozeile angibt:

```
cat /benutzer1/zugvogel/briefe/sanders <CR>
```

Auch mit dem Kommando **cd** können vollständige Pfadnamen angegeben werden. Um beispielsweise aus dem Verzeichnis **entwurf** in das Verzeichnis **briefe** zu wechseln, gibt man den Pfadnamen **/benutzer1/zugvogel/briefe** in der Kommandozeile wie folgt ein:

```
cd /benutzer1/zugvogel/briefe <CR>
```

Da sowohl **briefe** als auch **entwurf** außerdem Kindverzeichnisse von **zugvogel** sind, kann man auch den relativen Pfadnamen **../briefe** mit dem Kommando **cd** angeben. Durch **..** wechselt man in das Verzeichnis **zugvogel**, der Rest des Pfadnamens dient zum Wechseln in das Verzeichnis **briefe**.

In Abbildung 3-11 werden die Syntax und die Funktionen des Kommandos **cd** zusammengefaßt.

Kommandoübersicht cd – Arbeitsverzeichnis wechseln		
<i>Kommando</i>	<i>Optionen</i>	<i>Argumente</i>
cd	keine	<i>Verzeichnisname</i>
Beschreibung:	Mit cd wechselt man innerhalb des Dateisystems vom aktuellen Verzeichnis in das angegebene Verzeichnis. Wird kein Verzeichnisname als Argument angegeben, wechselt das System durch cd in das Home-Verzeichnis des Benutzers.	
Bemerkungen:	Nachdem die Shell in das angegebene Verzeichnis gewechselt hat, erscheint das Bereit-Zeichen (standardmäßig $\$$) wieder. Um auf ein Verzeichnis zuzugreifen, das sich nicht im Arbeitsverzeichnis befindet, ist der vollständige oder relative Pfadname anstelle eines einfachen Verzeichnisnamens einzugeben.	

Abbildung 3-11: Übersicht über das Kommando cd

Verzeichnis löschen: das Kommando rmdir

Wird ein Verzeichnis nicht mehr benötigt, kann man es mit dem Kommando **rmdir** (Abkürzung für "remove directory") löschen. Dieses Kommando hat folgende Standard-Syntax:

```
rmdir Verzeichnisname(n) <CR>
```

Dabei kann mehr als ein Verzeichnisname in einer Kommandozeile angegeben werden.

Ein Verzeichnisname kann mit dem Kommando **rmdir** nicht gelöscht werden, wenn man nicht Eigentümer des Verzeichnisses ist oder wenn das Verzeichnis nicht leer ist. Um eine Datei in einem Verzeichnis eines anderen Benutzers zu löschen, benötigt man das Schreibrecht für das Elternverzeichnis der Datei, die gelöscht werden soll. Dieses Recht muß vom Eigentümer des Elternverzeichnisses gewährt werden.

Versucht man, ein Verzeichnis zu löschen, in dem immer noch Unterverzeichnisse und Dateien enthalten sind (d. h. es ist nicht leer), gibt das Kommando **rmdir** die Meldung *Verzeichnisname not empty* aus. Dann müssen zuerst alle Unterverzeichnisse und Dateien gelöscht werden, damit man das Verzeichnis entfernen kann.

Hat man beispielsweise ein Verzeichnis mit dem Namen **infos**, in dem ein Unterverzeichnis mit dem Namen **tech** und zwei Dateien, **juni.30** und **juli.31**, enthalten sind, und versucht man nun, das Verzeichnis **infos** zu löschen (**rmdir** aus dem Home-Verzeichnis eingeben), erscheint folgende Anzeige:

```
$ rmdir infos<CR>
rmdir: infos not empty
$
```

Damit man das Verzeichnis **infos** löschen kann, muß zuerst sein Inhalt gelöscht werden: das Unterverzeichnis **tech** und die Dateien **juni.30** und **juli.31**. Dann kann man das Unterverzeichnis **tech** löschen, indem man das Kommando **rmdir** ausführt. Nähere Angaben zum Löschen von Dateien sind unter "Zugriff auf Dateien und ihre Bearbeitung" später in diesem Kapitel zu finden.

Nachdem man den Inhalt des Verzeichnisses **infos** gelöscht hat, kann man **infos** selbst entfernen bzw. löschen. Dazu muß man jedoch zuerst in dessen Elternverzeichnis (d. h. in diesem Fall das Home-Verzeichnis) wechseln. Das Kommando **rmdir** kann nicht ausgeführt werden, wenn man sich noch in dem Verzeichnis befindet, das man löschen will.



Geben Sie folgendes im Home-Verzeichnis ein:

rmdir infos <CR>

Ist **infos** leer, wird es durch das Kommando gelöscht und es erscheint das Bereit-Zeichen.

In Abbildung 3-12 werden die Syntax und die Funktionen des Kommandos **rmdir** zusammengefaßt.

Kommandoübersicht		
rmdir – Löschen eines Verzeichnisses.		
<i>Kommando</i>	<i>Optionen</i>	<i>Argumente</i>
rmdir	keine	<i>Verzeichnisname(n)</i>
Beschreibung:	Mit rmdir werden die angegebenen Verzeichnisse gelöscht, wenn sie keine Dateien und/oder Unterverzeichnisse enthalten.	
Bemerkungen:	Ist das Verzeichnis leer, wird es gelöscht und es erscheint das Bereit-Zeichen. Enthält das Verzeichnis noch Dateien oder Unterverzeichnisse, wird die Meldung rmdir: Verzeichnisname not empty ausgegeben.	

Abbildung 3-12: Übersicht über das Kommando **rmdir**

Zugriff auf Dateien und ihre Bearbeitung

In diesem Abschnitt werden verschiedene UNIX-Systemkommandos eingeführt, mit denen auf Dateien zugegriffen werden kann, und mit denen sie innerhalb der Struktur des Dateisystems verwaltet werden können. Die Informationen in diesem Abschnitt sind in zwei Teile gegliedert: Grundlagen und weiterführende Informationen. Der Teil, in dem die grundlegenden Kommandos beschrieben werden, ist Voraussetzung zur Benutzung des Dateisystems; die Kommandos im zweiten Teil ermöglichen komplexere Methoden der Informationsverarbeitung bei der Arbeit mit Dateien.

Grundlegende Kommandos

In diesem Abschnitt werden die UNIX-Systemkommandos behandelt, die zum Zugriff auf die Dateien in der Verzeichnisstruktur und zu ihrer Nutzung erforderlich sind. Die Kommandos sind in Abbildung 3-13 aufgeführt.

Kommando	Funktion
cat	Ausgabe des Inhalts einer bestimmten Datei auf einem Terminal
pg	Ausgabe des Inhalts einer bestimmten Datei auf einem Terminal in Abschnitten bzw. Seiten
pr	Ausgabe einer teilweise formatierten Version einer bestimmten Datei auf einem Terminal
lp	Anfordern eines Ausdrucks einer Datei auf einem Zeilendrucker
cp	Erstellen einer Kopie einer bestehenden Datei
mv	Verschieben und Umbenennen einer Datei
rm	Löschen einer Datei
wc	Zählen der Anzahl der Zeilen, Wörter und Zeichen in einer Datei
chmod	Ändern des Berechtigungsmodus für eine Datei (oder ein Verzeichnis)

Abbildung 3-13: Grundlegende Kommandos zur Arbeit mit Dateien

Diese Kommandos werden im folgenden ausführlich beschrieben und dann in Tabellen zum späteren Nachschlagen zusammengefaßt. In diesen Tabellen kann man die Syntax und die Funktionen dieser Kommandos auf einen Blick erkennen.

Ausgabe des Inhalts einer Datei: die Kommandos `cat`, `pg` und `pr`

In einem UNIX-System gibt es drei Kommandos zum Anzeigen und Ausdrucken des Inhalts einer oder mehrerer Dateien: `cat`, `pg` und `pr`. Das Kommando `cat` (Abkürzung für "concatenate" - verketten) gibt den Inhalt der angegebenen Datei(en) aus. Die Ausgabe wird auf dem Terminal angezeigt, sofern das Kommando `cat` nicht angewiesen wird, sie in eine andere Datei oder ein weiteres Kommando umzulenken.

Das Kommando `pg` ist insbesondere nützlich, wenn man den Inhalt einer langen Datei lesen will, da es den Text der Datei seitenweise anzeigt. Die angegebenen Dateien werden vom Kommando `pr` formatiert und auf dem Terminal ausgegeben oder, falls man dies wünscht, als formatierte Ausgabe zu einem Drucker umgelenkt (siehe auch unten, Kommando `lp`).

In den folgenden Abschnitten wird beschrieben, wie die Kommandos `cat`, `pg` und `pr` verwendet werden.

Verketteten und Ausdrucken des Inhalts einer Datei: das Kommando `cat`

Mit dem Kommando `cat` wird der Inhalt einer oder mehrerer Dateien angezeigt. Im folgenden Beispiel wird angenommen, daß sich der Benutzer im Verzeichnis `briefe` (unseres Beispiel-Dateisystems) befindet und den Inhalt der Datei `hermann` anzeigen will. Geben Sie dazu die abgebildete Kommandozeile ein; dann erscheint z. B. folgende Ausgabe:

```
$ cat hermann <CR>  
5. März 1989
```

```
Herrn  
Kurt Hermann  
Druck und Verlag  
Berlinerstr. 52  
7000 Stuttgart
```

```
Sehr geehrter Herr Hermann,
```

```
Ich möchte mich nochmals für das Gespräch über die geplante  
Automation des Geschäftsablaufes in Ihrer Firma bedanken.  
Anbei sende ich Ihnen das gewünschte Informationsmaterial  
über das Produktprogramm der Firma ABC zu Computern  
und Software für Büroautomation.
```

```
Zur weiteren Beratung stehe ich jederzeit gerne zu Ihrer Verfügung.
```

```
Mit freundlichen Grüßen,
```

```
Ralf Kern  
$
```

Es können auch mehrere Dateien angezeigt werden, indem man einfach ihre Namen zusammen in der Kommandozeile angibt. Soll beispielsweise der Inhalt der Dateien **hermann** und **sanders** angezeigt werden, folgendes eingeben:

```
$ cat hermann sanders <CR>
```

Dann wird der Inhalt von **hermann** und **sanders** in der angegebenen Reihenfolge auf dem Terminal ausgegeben.

\$ cat hermann sanders<CR>

5. März 1989

Herrn
Kurt Hermann
Druck und Verlag
Berlinerstr. 52
7000 Stuttgart

Sehr geehrter Herr Hermann:

Ich möchte mich nochmals für das Gespräch über die geplante

.
.

Mit freundlichen Grüßen,

Ralf Kern

5. März 1989

Frau
D. Sanders
Sanders Forschung AG
Hannover Str. 53
8000 München 57

Sehr geehrte Frau Sanders:

Meine Kollegen und ich haben mit großem Interesse verfolgt,

.
.

Mit freundlichen Grüßen,

Ralf Kern

\$

Die Ausgabe des Kommandos `cat` kann auch in eine andere Datei oder in ein weiteres Kommando umgelenkt werden. Siehe dazu die Abschnitte über die Umlenkung von Eingaben und Ausgaben in Kapitel 7.

In Abbildung 3-14 werden die Syntax und die Funktionen des Kommandos `cat` zusammengefaßt.

Kommandoübersicht		
<code>cat</code> – Verketteten und Ausgeben des Inhalts einer Datei		
<i>Kommando</i>	<i>Optionen</i>	<i>Argumente</i>
<code>cat</code>	<code>ja</code> *	<i>Dateiname(n)</i>
Beschreibung:	Das Programm <code>cat</code> liest die Namen der in der Kommandozeile angegebenen Dateien und zeigt ihren Inhalt an.	
Bemerkungen:	Ist die angegebene Datei vorhanden und lesbar, wird ihr Inhalt auf dem Terminal ausgegeben; andernfalls erscheint die Meldung <code>cat: cannot open Dateiname</code> auf dem Bildschirm. Der Inhalt eines Verzeichnisses kann mit dem Kommando <code>ls</code> angezeigt werden.	

* Alle verfügbaren Optionen zu diesem Kommando mit einer Erläuterung ihrer Funktionen sind unter `cat(1)` im *User's Reference Manual* zu finden.

Abbildung 3-14: Übersicht über das Kommando `cat`

Blättern in einer Datei: das Kommando pg

Mit dem Kommando `pg` (Abkürzung für "page" - Seite) kann der Inhalt einer oder mehrerer Dateien seitenweise auf einem Terminal durchgesehen werden. Dabei wird der Text einer Datei in Seiten aufgeteilt und angezeigt, gefolgt von einem Doppelpunkt als Aufforderung an den Benutzer, eine Anweisung einzugeben. Hier können Anweisungen wie Weiterführen der seitenweisen Anzeige der Datei oder Suchen nach einer bestimmten Zeichenkette in der/den Datei(en) eingegeben werden. In Abbildung 3-15 werden einige der verfügbaren Anweisungen zusammengefaßt.

Option*	Funktion
h	Hilfefunktion; Anzeigen einer Liste der verfügbaren Optionen von pg [†]
q oder Q	"Lesemodus" von pg verlassen
<CR>	Nächste Textseite anzeigen
l	Nächste Textzeile anzeigen
d oder ^d	Nächste halbe Textseite anzeigen
. oder ^1	Aktuelle Textseite erneut anzeigen
f	Nächste Textseite überspringen und nachfolgende anzeigen
n	Inhalt der nächsten in der Kommandozeile angegebenen Datei anzeigen
p	Vorige in der Kommandozeile angegebene Datei anzeigen
\$	Letzte Seite der aktuellen Datei anzeigen
/Suchmuster	In der Datei vorwärts nach einem bestimmten Zeichenmuster suchen
?Suchmuster	In der Datei rückwärts nach einem bestimmten Zeichenmuster suchen

* Die meisten dieser Anweisungen können mit einer vorangestellten Zahlenangabe ausgeführt werden, wie zum Beispiel +1 (nächste Seite anzeigen) -1 (vorige Seite anzeigen) oder 1 (erste Seite anzeigen).

† Alle verfügbaren Kommandos von **pg** werden im *User's Reference Manual* ausführlich erläutert.

Abbildung 3-15: Übersicht über die Optionen von **pg**

Das Kommando **pg** ist vor allem nützlich, wenn man eine lange Datei oder eine Reihe von Dateien nacheinander durchsehen will, da das Programm nach jeder Seite anhält, so daß genügend Zeit bleibt, sie durchzulesen. Die Größe der angezeigten Seite hängt vom Terminal ab. Bei einem Terminal mit 24zeiliger Anzeige besteht eine Seite aus dreiundzwanzig Zeilen Text und einer Zeile mit dem Doppelpunkt. Hat eine Datei jedoch weniger als 23 Zeilen, so ist die Seite so groß wie die Anzahl der Zeilen der Datei, zuzüglich einer Zeile für den Doppelpunkt.

Das Kommando **pg** zum Durchsehen einer Datei hat folgendes Format:

```
pg Dateiname(n)<CR>
```

Soll beispielsweise der Inhalt der Datei **gliederung** in unserem Beispiel-Dateisystem angezeigt werden, ist einzugeben:

```
pg gliederung<CR>
```

Dann erscheint die erste Seite der Datei auf dem Bildschirm. Da die Datei mehr Zeilen enthält, als auf einer Seite dargestellt werden können, erscheint ein Doppelpunkt am unteren Bildschirmrand. Der Doppelpunkt soll darauf hinweisen, daß die Datei noch länger ist als angezeigt werden kann. Drückt man dann die RETURN-Taste, gibt **pg** die nächste Seite der Datei aus.

Auf dem folgenden Bildschirm wird der bisher beschriebene Ablauf mit dem Kommando **pg** dargestellt.

\$ pg gliederung<CR>

Nachdem man den Gegenstand für einen Bericht analysiert hat, müssen die Informationen, die darin verwendet werden sollen, strukturiert werden.

.
.

Eine Gliederung ist eine wirksames Hilfsmittel zum Strukturieren von Informationen. Die Gliederung ist eine Art Gerüst bzw. ein Rahmen, an dem sich der Schreiber des Berichts orientieren kann; man kann sie auch mit einem Rezept vergleichen,
:<CR>

Nach dem letzten Drücken der RETURN-Taste (siehe Bildschirm) gibt pg den weiteren Inhalt der Datei auf dem Bildschirm aus:

in dem die Zutaten aufgeführt werden und festgelegt wird, in welcher Reihenfolge sie verwendet werden sollen.

.
.

Eine Gliederung braucht nicht ausgefeilt oder sehr detailliert sein; sie dient lediglich zur Orientierung beim Schreiben und kann bei Bedarf, d. h. bei Auftauchen zusätzlicher wichtiger Gedanken während des Schreibens, jederzeit geändert werden.
(EOF):

Hier ist die Zeile am unteren Bildschirmrand mit der Zeichenkette (EOF) : zu beachten. Diese Zeichenkette bedeutet, daß man am Ende der Datei angelangt ist. Das Bereit-Zeichen (Doppelpunkt) ist eine Aufforderung an den Benutzer, ein weiteres Kommando einzugeben.

Nachdem man die Datei durchgesehen hat, drückt man die RETURN-Taste. Dann erscheint ein Bereit-Zeichen auf dem Terminal (dies kann auch durch Eingabe von **q** oder **Q** mit anschließendem Drücken der RETURN-Taste erfolgen). Nach Bedarf kann man auch hier eine der anderen Anweisungen eingeben. Darüber hinaus steht auch eine Anzahl von Optionen zur Verfügung, die in der Kommandozeile des Kommandos **pg** angegeben werden können (siehe dazu **pg(1)** im *User's Reference Manual*).

Die korrekte Ausführung des Kommandos **pg** hängt davon ab, ob der verwendete Terminaltyp richtig definiert wurde. Dies ist darauf zurückzuführen, daß das Programm **pg** so konzipiert wurde, daß es flexibel auf vielen verschiedenen Terminals eingesetzt werden kann; die Form der Ausführung hängt jedoch vom jeweiligen Terminaltyp ab. Mit der Angabe des Typs teilt man dem Kommando **pg** mit:

- wieviele Zeilen ausgegeben werden sollen
- wieviele Spalten ausgegeben werden sollen
- wie der Bildschirm zu löschen ist
- wie Bereit-Zeichen und andere Wörter hervorzuheben sind
- wie die aktuelle Zeile gelöscht wird.

Ein Terminaltyp wird angegeben, indem man den Code für das Terminal der Variablen **TERM** in der Datei **.profile** zuweist. Nähere Angaben zu **TERM** und **.profile** sind in Kapitel 7 zu finden, Anweisungen zur Definition der Variablen **TERM** in Anhang F.

In Abbildung 3-16 werden die Syntax und die Funktionen des Kommandos **pg** zusammengefaßt.

Kommandoübersicht		
pg – Anzeigen des Inhalts einer Datei in Abschnitten bzw. Seiten		
Kommando	Optionen	Argumente
pg	ja*	Dateiname(n)
Beschreibung:	Mit dem Kommando pg wird der Inhalt der angegebenen Datei(en) seitenweise ausgegeben.	
Bemerkungen:	Nach der Ausgabe einer Textseite wartet das Kommando pg auf eine Anweisung des Benutzers, wie zum Beispiel: Anzeige weiterführen, nach einem Zeichenmuster suchen oder den Lesemodus des Kommandos pg verlassen. Darüber hinaus steht eine Anzahl von Optionen zur Verfügung. Beispielsweise kann man eine Datei ab einer bestimmten Zeile oder ab einer bestimmten Zeichenkette bzw. einem Zeichenmuster anzeigen lassen, oder in der Datei zu Textteilen zurückkehren die bereits angezeigt wurden.	

* Alle verfügbaren Optionen zu diesem Kommando mit einer Erläuterung ihrer Funktionen sind unter **pg(1)** im *User's Reference Manual* zu finden.

Abbildung 3-16: Übersicht über das Kommando **pg**

Teilweise formatierten Inhalt einer Datei ausgeben: das Kommando **pr**

Das Kommando **pr** wird verwendet, um Dateien zum Ausdrucken vorzubereiten. Mit ihm werden Titel und Überschriften definiert, ein Seitenumbruch durchgeführt und Dateien mit verschiedenen Seitenlängen und Seitenbreiten auf dem Terminal ausgegeben.

Der Benutzer kann festlegen, daß die Ausgabe des Kommandos auf einem anderen Gerät, wie zum Beispiel einem Zeilendrucker, erfolgt (siehe dazu die Beschreibung des Kommandos `lp` in diesem Abschnitt). Die Ausgabe des Kommandos `pr` kann auch in eine andere Datei geleitet werden (siehe dazu Umlenkung von Eingaben und Ausgaben in Kapitel 7).

Wird keine der verfügbaren Optionen angegeben, erzeugt das Kommando `pr` die Ausgabe in einer Spalte mit 66 Zeilen pro Seite und einem kurzen Kopfteil als Vorspann. Der Kopfteil besteht aus fünf Zeilen: zwei Leerzeilen, einer Zeile mit Datum, Uhrzeit, Dateiname und Seitenzahl und nochmals zwei Leerzeilen. Nach der formatierten Datei werden fünf Leerzeilen ausgegeben. Ein vollständiger Satz von Dienstprogrammen zur Formatierung steht auf UNIX-Systemen zur Verfügung, auf denen die Documenter's Workbench Software installiert ist. Beim Systemverwalter ist zu erfahren, ob diese Software zur Verfügung steht.

Das Kommando `pr` wird häufig zusammen mit dem Kommando `lp` dazu eingesetzt, einen Ausdruck eines Textes auf Papier so zu erzeugen, wie er in die Datei eingegeben wurde (nähere Angaben sind unter dem Kommando `lp` zu finden). Mit dem Kommando `pr` kann der Inhalt einer Datei jedoch auch formatiert und auf dem Terminal ausgegeben werden. Will man beispielsweise den Inhalt der Datei `hermann` des Beispiel-Dateisystems durchsehen, gibt man ein:

```
pr hermann <CR>
```

Der folgende Bildschirm stellt die mit diesem Kommando erzeugte Ausgabe dar.

\$ pr hermann<CR>

Mar 5 15:43 1989 hermann Page 1

5. März 1989

Herrn
Kurt Hermann
Druck und Verlag
Berlinerstr. 52
7000 Stuttgart

Sehr geehrter Herr Hermann,

Ich möchte mich nochmals für das Gespräch über die geplante
Automation des Geschäftsablaufes in Ihrer Firma bedanken.
Anbei sende ich Ihnen das gewünschte Informationsmaterial
über das Produktprogramm der Firma ABC zu Computern
und Software für Büroautomation.

Zur weiteren Beratung stehe ich jederzeit gerne zu Ihrer Verfügung.

Mit freundlichen Grüßen,

Ralf Kern

.
.
\$

Die Auslassungszeichen nach der letzten Zeile der Datei stehen für die übrigen Zeilen (in diesem Fall Leerzeilen), die durch das Kommando **pr** in die Ausgabe eingefügt wurden, so daß jede Seite insgesamt 66 Zeilen hat. Arbeitet man an einem Bildschirmterminal mit 24zeiliger Anzeige, werden alle 66 Zeilen der formatierten Datei ohne Pause ausgegeben.

Das bedeutet, daß die ersten 42 Zeilen über den Bildschirm laufen und sofort wieder verschwinden, so daß man sie nicht lesen kann, wenn man nicht die Möglichkeit hat, einen oder zwei Bildschirminhalte zurückzublättern. Wenn die Datei jedoch besonders lang ist, ist es unter Umständen selbst mit einer solchen Einrichtung nicht möglich, die Datei ganz zu lesen.

In diesen Fällen kann man mit der Tastenkombination <^s> (CONTROL-s) die über den Bildschirm laufende Anzeige anhalten. Gibt man danach <^q> (CONTROL-q) ein, läuft die Anzeige weiter.

In Abbildung 3-17 werden die Syntax und die Funktionen des Kommandos **pr** zusammengefaßt.

Kommandoübersicht		
pr – Ausdrucken des formatierten Inhalts einer Datei		
Kommando	Optionen	Argumente
pr	ja*	Dateiname(n)
Beschreibung:	Mit dem Kommando pr kann eine formatierte Kopie einer oder mehrerer Datei(en) auf dem Terminal oder bei Bedarf auf anderen Geräten ausgegeben werden. Der Text der Dateien wird auf Seiten von 66 Zeilen Länge ausgegeben, gleichzeitig werden fünf Leerzeilen am unteren Seitenrand und ein fünfzeiliger Kopf am oberen Seitenrand integriert. Der Kopf besteht aus zwei Leerzeilen, einer Zeile mit Datum, Uhrzeit, Dateiname und Seitenzahl sowie zwei weiteren Leerzeilen.	
Bemerkungen:	Ist die angegebene Datei vorhanden, wird der Inhalt formatiert und angezeigt; falls nicht, erscheint die Meldung <code>pr: can't open Dateiname</code> . Das Kommando pr wird häufig in Verbindung mit dem Kommando lp verwendet, um eine Datei auf Papier auszudrucken. Er kann auch dazu dienen, eine Datei auf einem Terminal anzuzeigen. Die Anzeige auf dem Bildschirm wird mit <code><^s></code> angehalten und mit <code><^q></code> fortgesetzt.	

* Alle verfügbaren Optionen zu diesem Kommando mit einer Erläuterung ihrer Funktionen sind unter `pr(1)` im *User's Reference Manual* zu finden.

Abbildung 3-17: Übersicht über das Kommando **pr**

Papierausdruck einer Datei anfordern: das Kommando lp

Bei bestimmten Terminals sind Drucker integriert, mit denen man Ausdrücke von Dateien erzeugen kann. Verfügt man über ein solches Terminal, erhält man einen Ausdruck einer Datei, indem man einfach den Drucker einschaltet und das Kommando `cat` oder `pr` ausführt. Arbeitet man jedoch mit einem Bildschirmterminal, muß ein Papierausdruck bei einem Drucker angefordert werden (siehe Abbildung 3-18). Dies erfolgt über das Kommando `lp` (Abkürzung für "line printer" - Zeilendrucker).

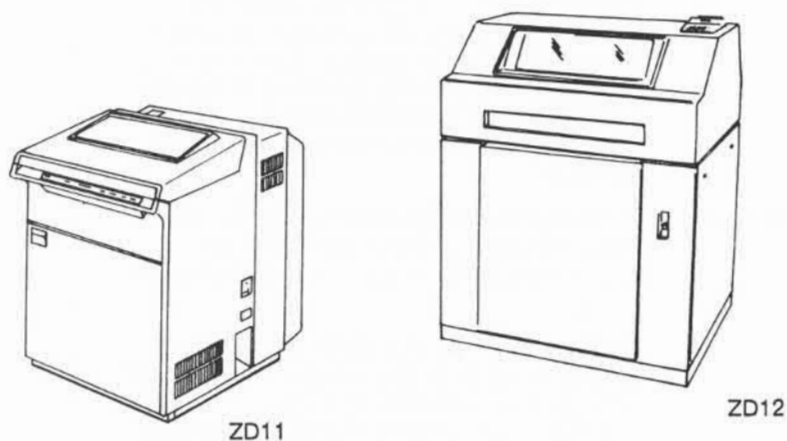


Abbildung 3-18: Zeilendrucker ZD11/ZD12

Für das Kommando **lp** gilt folgendes Format:

```
lp Dateiname <CR>
```

Zum Ausdrucken der Datei **hermann**, auf einem Zeilendrucker beispielsweise, muß folgende Kommandozeile eingegeben werden:

```
lp hermann <CR>
```

Das System antwortet mit dem Namen bzw. Typ des Druckers, auf dem die Datei ausgedruckt wird, und einer Auftragsnummer (ID) für den Auftrag des Benutzers.

```
$ lp hermann <CR>
request id is laser-6885 (1 file)
$
```

In diesem Beispiel wird der Job des Benutzers auf einem Laserdrucker ausgedruckt (der Standarddrucker in diesem System), er erhält die Auftragsnummer (ID) 6885 und umfaßt eine Datei.

Mit der Option **-ddest** (Abkürzung für "destination" - Ziel der Ausgabe) in der Kommandozeile kann die Datei auf einem anderen Gerät ausgedruckt werden; dieses wird als Argument mit **dest** angegeben. Mit der Option **-m** kann man eine Benachrichtigung nach Beendigung des Jobs anfordern.

Will man einen Auftrag an einen Drucker stornieren, gibt man das Kommando **cancel** mit der Auftragsnummer ein. Beispielsweise wird der Auftrag zum Drucken der Datei **briefe** (Auftragsnummer laser-6885) wie folgt storniert:

```
cancel laser-6885 <CR>
```

Mit dem Kommando **lpstat** kann man den Status eines laufenden Druckauftrags überprüfen oder die Auftragsnummer abfragen. Mit diesem Kommando wird auch eine vollständige Liste der Drucker ausgegeben, die im System zur Verfügung stehen. Es hängt von der Konfiguration des UNIX-Systems ab, welche Drucker den einzelnen Benutzern zur Verfügung stehen. Die Namen der verfügbaren Drucker kann man beim Systemverwalter oder durch folgende Eingabe in die Kommandozeile erfahren:

```
lpstat -v <CR>
```

In Abbildung 3-19 werden die Syntax und die Funktionen des Kommandos **lp** zusammengefaßt.



Kommandoübersicht		
lp – Papierausdruck einer Datei auf einem Zeilendrucker anfordern		
Kommando	Optionen	Argumente
lp	-d, -m, und andere*	Datei(en)
Beschreibung:	Mit dem Kommando lp kann man die angegebenen Dateien von einem Zeilendrucker auf Papier ausdrucken lassen.	
Optionen:	<p>-ddest Hier kann man für <i>dest</i> den Drucker bzw. Druckertyp angeben, auf dem ein Druckauftrag ausgeführt werden soll. Wird diese Option nicht angegeben, wählt das Programm lp den Standard-Drucker aus.</p> <p>-m Benachrichtigen des Benutzers über mail, wenn der Druckauftrag ausgeführt ist.</p>	
Bemerkungen:	<p>Ein Auftrag an einen Zeilendrucker kann durch Eingabe von cancel mit der Auftragsnummer (ID) storniert werden, die bei Bestätigung des Auftrags vom System zugeteilt wurde.</p> <p>Informationen über weitere und/oder andere Kommandos für die Drucker, die in einem bestimmten System zur Verfügung stehen, sind jeweils beim Systemverwalter zu erfragen.</p>	

* Alle verfügbaren Optionen zu diesem Kommando mit einer Erläuterung ihrer Funktionen sind unter **lp(1)** im *User's Reference Manual* zu finden.

Abbildung 3-19: Übersicht über das Kommando **lp**

Erstellen einer Kopie einer Datei: das Kommando cp

Es ist häufig erforderlich, eine Kopie einer Datei zu erstellen, so zum Beispiel, wenn eine Datei überarbeitet werden, jedoch die ursprüngliche Version unverändert erhalten bleiben soll. Mit dem Kommando `cp` (Abkürzung für "copy" - kopieren) kann der vollständige Inhalt einer Datei in eine andere kopiert werden. Außerdem kann man mit diesem Kommando eine oder mehrere Dateien von einem Verzeichnis in ein anderes kopieren und die ursprüngliche(n) Datei(en) an der alten Stelle gespeichert lassen.

Die Datei `gliederung` des Beispielverzeichnisses wird in die Datei `neu.gliederung` desselben Verzeichnisses kopiert, indem man einfach `cp gliederung neu.gliederung` eingibt und anschließend die RETURN-Taste drückt. Nach dem Kopiervorgang erscheint wieder das Bereit-Zeichen. Mit dem Kommando `ls` und anschließendem Drücken der RETURN-Taste kann man überprüfen, ob die neue Datei vorhanden ist. Mit diesem Kommando werden die Namen aller Dateien und Verzeichnisse des aktuellen Verzeichnisses (hier `entwurf`) ausgegeben. Dieser Vorgang wird auf dem folgenden Bildschirm veranschaulicht:

```
$ cp gliederung neu.gliederung<CR>
$ ls<CR>
gliederung
neu.gliederung
tabelle
$
```

Im UNIX-System können nicht zwei Dateien mit gleichem Namen in einunddemselben Verzeichnis stehen. Da eine Datei mit dem Namen `neu.gliederung` bei Eingabe des Kommandos `cp` in diesem Beispiel noch nicht existierte, hat das System eine neue Datei dieses Namens angelegt. Wäre jedoch bereits eine Datei mit dem Namen `neu.gliederung` vorhanden gewesen, wäre sie durch eine Kopie der Datei `gliederung` ersetzt worden, und die ursprüngliche Version von `neu.gliederung` wäre damit gelöscht bzw. überschrieben worden.

Versucht man beispielsweise, die Datei **gliederung** in eine andere Datei **gliederung** in demselben Verzeichnis zu kopieren, gibt das System die Meldung aus, daß die Dateinamen dieselben sind, und es erscheint wieder das Bereit-Zeichen; läßt man sich dann eine Liste des Inhalts des Verzeichnisses anzeigen, um festzustellen, wieviele Kopien von **gliederung** vorhanden sind, erscheint etwa folgende Anzeige auf dem Bildschirm:

```
$ cp gliederung gliederung<CR>
cp: gliederung and gliederung are identical
$ ls<CR>
gliederung
tabelle
$
```

Im UNIX-System können zwei Dateien denselben Namen haben, sie müssen dann jedoch in verschiedenen Verzeichnissen stehen. Man kann beispielsweise die Datei **gliederung** des Verzeichnisses **entwurf** in eine andere Datei mit Namen **gliederung** im Verzeichnis **briefe** kopieren. Befindet man sich beispielsweise im Verzeichnis **entwurf**, können dazu vier verschiedene Kommandozeilen verwendet werden. In den ersten beiden dieser Kommandozeilen wird hier der Name der neuen Datei angegeben, die durch Kopieren erstellt wird:

- **cp gliederung /benutzer1/zugvogel/briefe/gliederung<CR>** (vollständiger Pfadname)
- **cp gliederung ../briefe/gliederung<CR>** (relativer Pfadname)

Dabei muß der Name der neuen Datei nicht unbedingt ausdrücklich angegeben werden. Gibt man keinen neuen Dateinamen in der Kommandozeile an, benennt das Kommando **cp** die neue Datei mit demselben Namen. Daher kann auch eine der folgenden Kommandozeilen verwendet werden:

- **cp gliederung /benutzer1/zugvogel/briefe<CR>** (vollständiger Pfadname)

- `cp gliederung ../briefe<CR>` (relativer Pfadname)

In diesen vier Beispielen legt `cp` jeweils eine Kopie der Datei `gliederung` im Verzeichnis `briefe` an und gibt ihr ebenfalls den Namen `gliederung`.

Soll die neue Datei einen anderen Namen erhalten, muß er natürlich angegeben werden. Will man beispielsweise die Datei `gliederung` im Verzeichnis `entwurf` in eine Datei mit dem Namen `gliederung.vs2` im Verzeichnis `briefe` kopieren, ist eine der folgenden Kommandozeilen zu verwenden:

- `cp gliederung /benutzer1/zugvogel/briefe/gliederung.vs2<CR>` (vollständiger Pfadname)
- `cp gliederung ../briefe/gliederung.vs2<CR>` (relativer Pfadname)

Bei der Vergabe neuer Namen sind die entsprechenden Konventionen zu beachten, die unter "Benennen von Verzeichnissen und Dateien" in diesem Kapitel aufgeführt sind.

In Abbildung 3-20 werden die Syntax und die Funktionen des Kommandos `cp` zusammengefaßt.

Kommandoübersicht		
cp – Eine Kopie einer Datei erstellen		
<i>Kommando</i>	<i>Optionen</i>	<i>Argumente</i>
cp	keine	<i>Datei1</i> <i>Datei2</i> <i>Datei(en)</i> <i>Verzeichnis</i>
Beschreibung:	Mit cp kann eine Kopie von <i>Datei1</i> unter dem Namen <i>Datei2</i> angelegt werden, wobei <i>Datei1</i> erhalten bleibt; man kann damit auch eine oder mehrere Dateien in ein anderes Verzeichnis kopieren.	
Bemerkungen:	<p>Kopiert man <i>Datei1</i> in <i>Datei2</i>, und besteht eine Datei mit dem Namen <i>Datei2</i> bereits, überschreibt das Kommando cp die erste Version von <i>Datei2</i> mit einer Kopie von <i>Datei1</i> und nennt sie <i>Datei2</i>. Die alte Version von <i>Datei2</i> wird dabei gelöscht.</p> <p>Mit dem Kommando cp können keine Verzeichnisse kopiert werden.</p>	

Abbildung 3-20: Übersicht über das Kommando cp

Verschieben und Umbenennen einer Datei: das Kommando mv

Mit dem Kommando **mv** (Abkürzung für "move" - verschieben) kann man eine Datei in demselben Verzeichnis umbenennen oder von einem Verzeichnis in ein anderes bewegen. Verschiebt man eine Datei in ein anderes Verzeichnis, kann die Datei umbenannt werden oder ihren ursprünglichen Namen behalten.

Eine Datei wird innerhalb eines Verzeichnisses mit einer Kommandozeile des folgenden Formats umbenannt:

```
mv Datei1 Datei2<CR>
```

Mit dem Kommando **mv** wird der Name einer Datei von *Datei1* in *Datei2* umbenannt und gleichzeitig *Datei1* gelöscht. Für *Datei1* und *Datei2* können alle zulässigen Namen einschließlich Pfadnamen angegeben werden.

Befindet man sich beispielsweise im Verzeichnis **entwurf** im Beispiel-Dateisystem und möchte die Datei **tabelle** in **neu.tabelle** umbenennen, ist einfach **mv tabelle neu.tabelle** einzugeben und anschließend die RETURN-Taste zu drücken. Wird das Kommando erfolgreich ausgeführt, erscheint das Bereit-Zeichen. Um zu überprüfen, ob die Datei **neu.tabelle** existiert, kann man den Inhalt des Verzeichnisses durch Eingabe von **ls** und Drücken der RETURN-Taste auflisten lassen. Auf dem Bildschirm erscheint diese Eingabe und die darauf folgende Ausgabe des Systems wie folgt:

```
$ mv tabelle neu.tabelle <CR>
$ ls <CR>
gliederung
neu.tabelle
$
```

Man kann eine Datei auch von einem Verzeichnis in ein anderes verschieben. Dabei kann der Dateiname beibehalten oder geändert werden. Mit der folgenden Kommandozeile wird eine Datei unter Beibehaltung ihres Namens verschoben:

```
mv Datei(en) Verzeichnis <CR>
```

Als Datei- und Verzeichnisnamen können alle zulässigen Namen einschließlich der Pfadangaben eingesetzt werden.

Will man beispielsweise die Datei **tabelle** aus dem aktuellen Verzeichnis **entwurf** (vollständiger Pfadname **/benutzer1/zugvogel/entwurf**) in eine Datei desselben Namens aus dem Verzeichnis **briefe** verschieben (relativer Pfadname) im Verzeichnis **entwurf** ist **../briefe**), vollständiger Pfadname ist **/benutzer1/zugvogel/briefe**), können folgende verschiedenen Kommandozeilen verwendet werden:

```
mv tabelle /benutzer1/zugvogel/briefe <CR>
```

```
mv tabelle /benutzer1/zugvogel/briefe/tabelle <CR>
```

```
mv tabelle ../briefe <CR>
```

```
mv tabelle ../briefe/tabelle <CR>
```

```
mv /benutzer1/zugvogel/entwurf/tabelle /benutzer1/zugvogel/briefe/tabelle <CR>
```

Soll die Datei **tabelle** in **tabelle2** umbenannt werden, während man sie in das Verzeichnis **briefe** verschiebt, kann eine der folgenden Kommandozeilen verwendet werden:

```
mv tabelle /benutzer1/zugvogel/briefe/tabelle2 <CR>
```

```
mv tabelle ../briefe/tabelle2 <CR>
```

```
mv /benutzer1/zugvogel/entwurf/tabelle/benutzer1/zugvogel/briefe/tabelle2 <CR>
```

Mit dem Kommando **ls** kann man den Inhalt des Verzeichnisses auflisten lassen, um zu überprüfen, ob das Kommando richtig ausgeführt wurde.

In Abbildung 3-21 werden die Syntax und die Funktionen des Kommandos **mv** zusammengefaßt.

Kommandoübersicht		
mv – Verschieben oder Umbenennen von Dateien		
Kommando	Optionen	Argumente
mv	keine	<i>Datei1 Datei2</i> <i>Datei(en) Verzeichnis</i>
Beschreibung:	Mit mv kann der Name einer Datei geändert werden, oder man kann eine oder mehrere Dateien in ein anderes Verzeichnis verschieben.	
Bemerkungen:	Verschiebt man eine <i>Datei1</i> in eine <i>Datei2</i> , wenn eine Datei namens <i>Datei2</i> bereits existiert, wird die ursprüngliche Version von <i>Datei1</i> überschrieben und <i>Datei2</i> genannt. Die erste Version von <i>Datei2</i> wird dabei gelöscht.	

Abbildung 3-21: Übersicht über das Kommando mv

Löschen einer Datei: das Kommando rm

Benötigt man eine Datei nicht mehr, kann man sie mit dem Kommando **rm** (Abkürzung für "remove" - entfernen) löschen. Das Grundformat für dieses Kommando lautet:

```
rm Datei(en) <CR>
```

Man kann gleichzeitig mehrere Dateien löschen, indem man die Dateien in der Kommandozeile zusammen eingibt, jeweils mit einem Leerzeichen zwischen den Dateinamen:

```
rm Datei1 Datei2 Datei3 <CR>
```

Das System legt keine Sicherungskopie einer Datei an, wenn sie gelöscht wird. Nach dem Ausführen dieses Kommandos ist die Datei gelöscht.

Nach dem Ausführen des Kommandos **rm** kann man überprüfen, ob es erfolgreich ausgeführt wurde, indem man das Kommando **ls** eingibt. Mit **ls** werden alle Dateien des Verzeichnisses angezeigt; so sieht man sofort, ob das Kommando **rm** erfolgreich ausgeführt wurde.

Hat man beispielsweise ein Verzeichnis mit den beiden Dateien **gliederung** und **tabelle**, kann man beide Dateien gleichzeitig mit einem **rm**-Kommando löschen. Wird **rm** erfolgreich ausgeführt, ist das Verzeichnis leer. Dies kann man mit dem Kommando **ls** überprüfen:

```
$ rm gliederung tabelle <CR>
$ ls
$
```

Das Bereit-Zeichen zeigt an, daß **gliederung** und **tabelle** gelöscht wurden.

In Abbildung 3-22 werden die Syntax und die Funktionen des Kommandos **rm** zusammengefaßt.

Kommandoübersicht		
rm – eine Datei löschen		
<i>Kommando</i>	<i>Optionen</i>	<i>Argumente</i>
rm	ja*	<i>Datei(en)</i>
Beschreibung:	Mit rm kann man eine oder mehrere Dateien löschen.	
Bemerkungen:	Die Dateien, die als Argumente mit dem Kommando rm angegeben werden, werden dauerhaft gelöscht.	

* Alle verfügbaren Optionen zu diesem Kommando mit einer Erläuterung ihrer Funktionen sind unter **rm(1)** im *User's Reference Manual* zu finden.

Abbildung 3-22: Übersicht über das Kommando **rm**

Zellen, Wörter und Zeichen einer Datei zählen: das Kommando `wc`

Mit dem Kommando `wc` (Abkürzung für "word count" - Wortzählung) kann man sich anzeigen lassen, wieviele Zeilen, Wörter und Zeichen in der oder den Datei(en) stehen, die in der Kommandozeile angegeben werden. Gibt man mehrere Dateien an, zählt das Programm `wc` die Zeilen, Wörter und Zeichen der einzelnen Dateien und bildet dann die Summe daraus. Man kann das Programm `wc` auch anweisen, nur Zeilen, nur Wörter oder nur Zeichen zu zählen, indem man die Optionen `-l`, `-w` bzw. `-c` angibt.

Die Kommandozeile für dieses Kommando hat folgendes Format:

```
wc Datei1 <CR>
```

Das System antwortet mit einer Zeile des folgenden Formats:

```
l w c Datei1
```

Dabei steht

- `l` für die Anzahl der Zeilen (lines) in *Datei1*
- `w` für die Anzahl der Wörter (words) in *Datei1*
- `c` für die Anzahl der Zeichen (characters) in *Datei1*.

Sollen zum Beispiel die Zeilen, Wörter und Zeichen der Datei **hermann** (im aktuellen Verzeichnis **briefe**) gezählt werden, müssen Sie folgende Kommandozeile eingeben:

```
$ wc hermann <CR>
24      66      406 hermann
$
```

Die Ausgabe des Systems bedeutet hier, daß die Datei **hermann** 24 Zeilen, 66 Wörter und 406 Zeichen hat.

Mit folgendem Kommandozeilenformat können die Zeilen, Wörter und Zeichen in mehreren Dateien gezählt werden:

```
wc Datei1 Datei2 <CR>
```

Zugriff auf Dateien und ihre Bearbeitung

Die Ausgabe des Systems sieht dann etwa folgendermaßen aus:

```
l   w   c   Datei1
l   w   c   Datei2
l   w   c   total
```

Die Anzahl der Zeilen, Wörter und Zeichen wird für *Datei1* und *Datei2* jeweils auf einer separaten Zeile angezeigt, und die Summen erscheinen in der letzten Zeile neben dem Wort *total*.

Im folgenden Beispiel wird das Programm *wc* angewiesen, die Zeilen, Wörter und Zeichen in den Dateien **hermann** und **sanders** des aktuellen Verzeichnisses zu zählen.

```
$ wc hermann sanders <CR>
  24    66   406 hermann
  28    92   559 sanders
  52   158   965 total
$
```

In der ersten Zeile wird angezeigt, daß die Datei **hermann** 24 Zeilen, 66 Wörter und 406 Zeichen hat. In der zweiten Zeile wird angezeigt, daß die Datei **sanders** 28 Zeilen, 92 Wörter und 559 Zeichen hat. In der letzten Zeile wird angegeben, daß die beiden Dateien zusammen 52 Zeilen, 158 Wörter und 965 Zeichen haben.

Die folgenden Zeilen enthalten die Formate für Kommandozeilen, wenn man nur Zeilen, Wörter oder Zeichen zählen lassen will:

```
wc -l Datei1 <CR>   (Zeilen)
wc -w Datei1 <CR>   (Wörter)
wc -c Datei1 <CR>   (Zeichen)
```

Gibt man beispielsweise die Option `-l` an, gibt das System nur die Anzahl der Zeilen in der angegebenen Datei (hier `sanders`) aus:

```
$ wc -l sanders <CR>
    28 sanders
$
```

Gibt man stattdessen die Option `-w` oder `-c` an, gibt das System die Anzahl der Wörter bzw. Zeichen der Datei aus.

In Abbildung 3-23 werden die Syntax und die Funktionen des Kommandos `wc` zusammengefaßt.

Kommandoübersicht		
wc – Anzahl der Zeilen, Wörter und Zeichen in einer Datei zählen		
Kommando	Optionen	Argumente
wc	-l, -w, -c	Datei(en)
Beschreibung:	Mit <code>wc</code> werden die Zeilen, Wörter und Zeichen in den angegebenen Dateien gezählt; gibt man mehrere Dateien an, werden außerdem die Summen der einzelnen Spalten gebildet.	
Optionen	<ul style="list-style-type: none"> -l Zählen der Anzahl der Zeilen in der/den angegebenen Datei(en) -w Zählen der Anzahl der Wörter in der/den angegebenen Datei(en) -c Zählen der Anzahl der Zeichen in der/den angegebenen Datei(en) 	
Bemerkungen:	Ein Dateiname, der in der Kommandozeile angegeben wird, wird mit den Zählergebnissen angezeigt.	

Abbildung 3-23: Übersicht über das Kommando `wc`

Dateien schützen: das Kommando `chmod`

Über das Kommando `chmod` (Abkürzung für "change mode" - Modus ändern) kann der Benutzer festlegen, wer seine Dateien lesen, in sie schreiben und sie benutzen darf. Da das Betriebssystem UNIX ein Mehrbenutzersystem ist, arbeitet man in der Regel nicht alleine im Dateisystem. Die Benutzer des Systems können den Pfadnamen in verschiedene Verzeichnisse folgen und die Dateien anderer Benutzer lesen und benutzen, sofern sie das Recht dazu haben.

Der Eigentümer einer Datei kann entscheiden, wer das Recht hat, sie zu lesen, in sie zu schreiben (d. h. Änderungen in ihr vorzunehmen) oder, wenn es sich um eine Programmdatei handelt, sie auszuführen. Auch die Rechte für Verzeichnisse können mit dem Kommando `chmod` eingeschränkt werden. Erteilt man das Ausführbarkeitsrecht für ein Verzeichnis, dann erlaubt man den angegebene Benutzern, mit `cd` in dieses Verzeichnis zu wechseln und den Inhalt mit dem Kommando `ls` anzeigen zu lassen.

Die Form der Berechtigung wird durch Angabe der folgenden Buchstaben bestimmt:

- r** erlaubt den Benutzern des Systems, eine Datei zu lesen oder ihren Inhalt zu kopieren
- w** erlaubt den Benutzern des Systems, Änderungen in einer Datei (oder einer Kopie davon) vorzunehmen
- x** erlaubt den Benutzern des Systems, eine ausführbare Datei auszuführen.

Die Benutzer, denen diese Zugriffsrechte erteilt (bzw. nicht erteilt) werden, werden mit folgenden drei Buchstaben definiert:

- u** Der Benutzer selbst, als Eigentümer der Dateien und Verzeichnisse (u ist die Abkürzung für "user" - Benutzer)
- g** Mitglieder der Gruppe, zu der auch der Benutzer selbst gehört (die Gruppe besteht beispielsweise aus den Mitgliedern einer Arbeitsgruppe, die an demselben Projekt arbeiten, den Angehörigen einer Abteilung oder einer Gruppe von Personen, die von der Person beliebig definiert werden kann, die das Benutzerkonto des betreffenden Benutzers eingerichtet hat).
- o** alle anderen Systembenutzer.

Legt man eine Datei oder ein Verzeichnis an, erteilt oder verweigert das System automatisch die Rechte dem Benutzer selbst, Mitgliedern seiner Gruppe bzw. anderen Systembenutzern. Dieser automatische Vorgang kann vom Benutzer beeinflusst werden, indem er seine Umgebung modifiziert (nähere Angaben dazu sind in Kapitel 7 zu finden). Unabhängig davon, wie die Rechte beim Anlegen einer Datei definiert waren, kann man sie als Eigentümer der Datei bzw. des Verzeichnisses jederzeit ändern. Häufig sollen bestimmte Dateien niemand anders zugänglich sein und ausschließlich der eigenen Verwendung durch den Benutzer selbst vorbehalten bleiben. Man kann das Recht zum Lesen und Schreiben in einer Datei auch an die Mitglieder der eigenen Gruppe oder andere Systembenutzer vergeben. Es kann auch erwünscht sein, ein Programm mit den Mitgliedern der eigenen Gruppe gemeinsam zu nutzen und ihnen daher das Recht zu gewähren, es auszuführen.

Feststellen der geltenden Zugriffsrechte

Mit dem folgenden Kommando kann man feststellen, welche Rechte für eine Datei oder ein Verzeichnis gerade gültig sind, und es wird eine ausführliche Liste des Inhalts eines Verzeichnisses ausgegeben: `ls -l`. Gibt man beispielsweise `ls -l` ein und drückt anschließend die RETURN-Taste, während man sich im Verzeichnis `zugvogel/bin` des Beispiel-Dateisystems befindet, erscheint folgende Ausgabe:

```
$ ls -l<CR>
total 35
-rwxr-xr-x 1 zugvogel projekt 9346 Nov 1 08:06 anzeige
-rw-r--r-- 1 zugvogel projekt 6428 Dec 2 10:24 liste
drwx-x-x 2 zugvogel projekt 32 Nov 8 15:32 tools
$
```

Die Rechte für die Dateien `anzeige` und `liste` und das Verzeichnis `tools` erscheinen links in der Anzeige unter der Zeile `total 35`, in diesem Format:

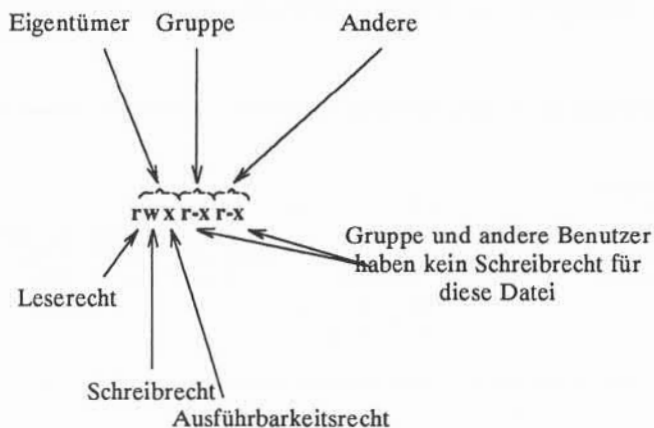
```
-rwxr-xr-x (für die Datei anzeige)
-rw-r--r-- (für die Datei liste)
drwx--x--- (für das Verzeichnis tools)
```

Die Rechte für die Dateien **anzeige** und **liste** und das Verzeichnis **tools** erscheinen links in der Anzeige unter der Zeile **total 35**, in diesem Format:

```
-rwxr-xr-x (für die Datei anzeige)  
-rw-r--r-- (für die Datei liste)  
drwx--x--- (für das Verzeichnis tools)
```

Das erste Zeichen gibt jeweils den Dateityp an (zum Beispiel bezeichnet ein Bindestrich (-) eine normale Datei, der Buchstabe **d** ein Verzeichnis), die weiteren neun Zeichen die Zugriffsrechte. Sie bestehen aus jeweils drei Gruppen zu je drei Zeichen. Die erste dieser Zeichengruppen bezieht sich auf die Rechte für den Eigentümer, die zweite auf die Rechte für Mitglieder der Gruppe, die letzte auf die Rechte für alle anderen Benutzer des Systems. Innerhalb dieser Zeichengruppen werden die geltenden Zugriffsrechte für jede Kategorie mit den Buchstaben **r**, **w** und **x** angezeigt. Steht anstelle eines der Buchstaben **r**, **w** oder **x** ein Bindestrich, ist das Recht zum Lesen, Schreiben oder Ausführen nicht erteilt.

In der folgenden Abbildung werden diese Komponenten für die Datei **anzeige** dargestellt.



Wie aus dieser Darstellung ersichtlich, hat der Eigentümer die Rechte **r**, **w** und **x**, und die Mitglieder der Gruppe sowie die anderen Systembenutzer haben die Rechte **r** und **x**.



Bei dieser Anzeige gibt es zwei Ausnahmen. Gelegentlich kann der Buchstabe *s* oder *l* anstelle von *r*, *w* oder *x* in der Anzeige erscheinen. Der Buchstabe *s* (Abkürzung für "set user ID" oder "set group ID" - "s-bit für Benutzer setzen" oder "s-bit für Gruppe setzen") steht für eine spezielle Form der Berechtigung zum Ausführen einer Datei. Es kann an der Stelle erscheinen, an der normalerweise *x* (oder *-*) für den Eigentümer oder die Gruppe steht (erste und zweite Zeichengruppe). Aus der Sicht des Benutzers ist es gleichbedeutend mit einem *x* an derselben Stelle; es bedeutet, daß das Ausführbarkeitsrecht besteht. Es ist nur für Programmierer und Systemverwalter von Bedeutung (nähere Angaben zum Setzen des s-Bit für Benutzer oder Gruppe sind im *System Administrator's Guide* enthalten).

Der Buchstabe *l* (Abkürzung für "lock enabling") gibt an, daß die Dateisperre aktiviert ist. Dies bedeutet nicht, daß die Datei gesperrt ist, sondern daß die Sperrfunktion für diese Datei aktiv ist, d. h. verwendet werden kann. Am Buchstaben *l* kann man also nicht erkennen, ob eine Datei gesperrt ist oder nicht.

Ändern von bestehenden Zugriffsrechten

Nachdem man festgestellt hat, welche Zugriffsrechte gerade gelten, kann man sie über das Kommando **chmod** mit folgendem Format ändern:

chmod *who* +*Recht* *Datei(en)* <**CR**>

oder

chmod *who* =*Recht* *Datei(en)* <**CR**>

In der folgenden Aufstellung werden die einzelnen Bestandteile dieser Kommandozeile erläutert:

chmod	Name des Programms bzw. Kommandos
who	eine der drei Benutzerkategorien (u , g oder o) u = Eigentümer g = Gruppe o = Andere
+ oder -	Gewähren (+) oder Verweigern (-) des Rechtes
Recht	beliebige Kombination der drei Rechte (r , w und x) r = Leserecht w = Schreibrecht x = Ausführbarkeitsrecht
Datei(en)	Datei- oder Verzeichnisname(n); es wird davon ausgegangen, daß es sich um Verzweigungen des aktuellen Verzeichnisses handelt, sofern man nicht jeweils den vollständigen Pfadnamen angibt.

NOTE

Das Kommando **chmod** arbeitet nicht, wenn ein oder mehrere Leerzeichen zwischen *who*, der Angabe, ob das Recht gewährt (+) oder verweigert (-) werden soll, und dem *Recht* eingegeben werden.

In den folgenden Beispielen wird die Verwendung des Kommandos **chmod** veranschaulicht. Als Eigentümer der Datei **anzeige** kann man diese ausführbare Datei lesen, in sie schreiben und sie ausführen. Man kann diese Datei vor versehentlichem Überschreiben schützen, indem man sich selbst das Schreibrecht (**w**) verweigert. Dazu ist folgende Kommandozeile einzugeben:

```
chmod u-w anzeige <CR>
```

Nachdem das Bereit-Zeichen erscheint, ls **-l** eingeben und anschließend die RETURN-Taste drücken, um zu überprüfen, ob das Zugriffsrecht korrekt geändert wurde, wie auf dem folgenden Bildschirm dargestellt:

```

$ chmod u-w anzeige <CR>
$ ls -l <CR>
total 35
-r-xr-xr-x 1 zugvogel projekt 9346 Nov 1 08:06 anzeige
rw-r--r-- 1 zugvogel projekt 6428 Dec 2 10:24 liste
drwx-x-x 2 zugvogel projekt 32 Nov 8 15:32 tools
$

```

Hier wird deutlich, daß der Eigentümer selbst nun nicht mehr die Berechtigung hat, die Datei zu ändern. Erst wenn der Benutzer für sich selbst das Schreibrecht wieder definiert, kann er sie wieder ändern.

Das Schreibrecht für die Datei **anzeige** wurde den Mitgliedern der Gruppe des Eigentümers und den anderen Systembenutzern verweigert. Sie haben jedoch das Leserecht. Das bedeutet, daß sie die Datei in ihre eigenen Verzeichnisse kopieren und dann Änderungen in ihr vornehmen können. Um zu verhindern, daß andere Benutzer diese Datei in ihr Verzeichnis kopieren, kann man ihnen das Leserecht für diese Datei durch folgende Eingabe entziehen:

```
chmod go-r anzeige <CR>
```

Die Buchstaben **g** und **o** stehen für Mitglieder der Gruppe bzw. alle anderen Benutzer des Systems, mit dem Buchstaben **-r** wird ihnen das Recht zum Lesen oder Kopieren der Datei verweigert. Die Ergebnisse dieses Vorgangs können mit dem Kommando **ls -l** überprüfen werden:

```

$ chmod go-r anzeige<CR>
$ ls -l<CR>
total 35
-rwx--x--x  1 zugvogel   projekt    9346 Nov 1  08:06 anzeige
rw-r--r--   1 zugvogel   projekt    6428 Dec 2  10:24 liste
drwx--x--x  2 zugvogel   projekt     32 Nov 8  15:32 tools
$

```

Hinweis zu Zugriffsrechten auf Verzeichnisse

Wie für Dateien kann das Kommando **chmod** auch verwendet werden, um Zugriffsrechte auf Verzeichnisse zu gewähren oder zu verweigern. Dazu ist lediglich anstelle des Dateinamens in der Kommandozeile der Verzeichnisname anzugeben.

Dabei sind jedoch die Auswirkungen für die anderen Benutzer des Systems zu bedenken. Erteilt man beispielsweise das Leserecht für ein Verzeichnis sich selbst (**u**), Mitgliedern der eigenen Gruppe (**g**) und den anderen Benutzern des Systems (**o**), so kann sich jeder Benutzer, der Zugriff auf das System hat, die Namen der in diesem Verzeichnis enthaltenen Dateien mit dem Kommando **ls -l** auflisten lassen. Ebenso ermöglicht das Schreibrecht es den angegebenen Benutzern, in diesem Verzeichnis neue Dateien anzulegen und bestehende Dateien zu löschen. Wird bestimmten Benutzern das Ausführbarkeitsrecht für ein Verzeichnis gewährt, können sie mit dem Kommando **cd** in dieses Verzeichnis wechseln (und es zu ihrem aktuellen Verzeichnis machen).

Alternative Form von chmod

Das Kommando **chmod** kann in zwei verschiedenen Formen ausgeführt werden. Die oben beschriebene Form, in der Symbole wie **r**, **w** und **x** verwendet werden, um Zugriffsrechte zu definieren, wird die symbolische Methode genannt.

Als alternative Form dazu steht die oktale Methode zur Verfügung. Dabei werden Zugriffsrechte mit drei Oktalziffern (Bereich 0 bis 7) definiert. Das oktale Zahlensystem unterscheidet sich von unserem im Alltag üblichen Dezimalsystem. Nähere Informationen zur oktalen Methode sind unter **chmod(1)** im *User's Reference Manual* zu finden.

In Abbildung 3-24 werden die Syntax und die Funktionen des Kommandos **chmod** zusammengefaßt.

Kommandoübersicht		
chmod – Ändern von Zugriffsrechten für Dateien (und Verzeichnisse)		
<i>Kommando</i>	<i>Anweisung</i>	<i>Argumente</i>
chmod	who + – Recht	<i>Dateiname(n)</i> <i>Verzeichnisname(n)</i>
Beschreibung:	Mit chmod kann das Leserecht, Schreibrecht und Ausführbarkeitsrecht drei Kategorien von Systembenutzern gewährt (+) oder verweigert (–) werden: Eigentümer (der Benutzer selbst), Gruppe (Mitglieder der Gruppe des Eigentümers) und Andere (alle Benutzer des Systems, an dem der Eigentümer arbeitet).	
Bemerkungen:	Der Anweisungssatz kann in oktaler oder symbolischer Form eingegeben werden.	

Abbildung 3-24: Übersicht über das Kommando **chmod**



Kommandos für Fortgeschrittene

Der Grad der Vertrautheit mit dem Dateisystem wächst mit der Häufigkeit der Verwendung der bisher eingeführten Kommandos. Mit wachsender Vertrautheit wird auch der Wunsch nach vielfältigeren Methoden für die Arbeit mit Dateien wachsen. In diesem Abschnitt werden drei Kommandos eingeführt, die die bisherigen Möglichkeiten erweitern.

diff	Vergleichen von zwei Dateien auf unterschiedlichen Inhalt
grep	Suchen eines Zeichenmusters in einer Datei
sort	Sortieren und Mischen von Dateien

Weitere Informationen zu diesen Kommandos sind im *User's Reference Manual* zu finden.

Unterschiede zwischen Dateien feststellen: das Kommando **diff**

Mit dem Kommando **diff** werden alle Unterschiede zwischen zwei Dateien gesucht und ausgegeben; dabei wird angegeben, wie die erste Datei zu ändern ist, um ein Duplikat der zweiten Datei zu erhalten. Das Grundformat des Kommandos lautet:

```
diff Datei1 Datei2<CR>
```

Sind *Datei1* und *Datei2* identisch, gibt das System wieder das Bereit-Zeichen aus. Sind sie nicht identisch, wird dem Benutzer durch das Kommando **diff** mitgeteilt, wie die erste Datei mit Kommandos des Zeileneditors **ed** zu ändern ist, damit sie genau der zweiten Datei entspricht (nähere Einzelheiten zum Zeileneditor sind in Kapitel 5 zu finden). Das UNIX-System markiert die Zeilen in *Datei1* (die zu ändern sind) mit dem Symbol < (kleiner als), und die entsprechenden Zeilen in *Datei2* (der Vorlage) mit dem Symbol > (größer als).

Angenommen, man führt das Kommando **diff** aus, um die Unterschiede zwischen den Dateien **hermann** und **maierhofer** festzustellen. Die Datei **maierhofer** enthält denselben Brief, der auch in der Datei **hermann** steht, mit den entsprechenden Änderungen für den unterschiedlichen Adressaten. Mit dem Kommando **diff** werden diese Änderungen wie folgt ausgegeben:


```

3,6c3,6
< Kurt Hermann
< Druck und Verlag
< Berlinerstr. 52
< 7000 Stuttgart
-----
> J. Maierhofer
> Uhu Presse
> Bahnhofstr. 223
> 4790 Paderborn
9c9
< Sehr geehrter Herr Hermann:
-----
> Sehr geehrter Herr Maierhofer:

```

Die erste Ausgabezeile aus **diff** lautet:

```
3,6c3,6
```

Das bedeutet, daß man die Zeilen 3 bis 6 in **hermann** in die Zeilen 3 bis 6 in **maierhofer** ändern (c - change) müßte, wollte man **hermann** so anpassen, daß sie mit **maierhofer** identisch ist. Danach werden durch **diff** beide Zeilengruppen angezeigt.

Nimmt man diese Änderungen vor (mit einem Texteditor wie **ed** oder **vi**), dann ist die Datei **hermann** mit der Datei **maierhofer** identisch. Das Kommando **diff** zeigt die Unterschiede zwischen Dateien an. Will man eine exakte Kopie einer Datei erstellen, ist das Kommando **cp** zu verwenden.

In Abbildung 3-25 werden die Syntax und die Funktionen des Kommandos **diff** zusammengefaßt.

Kommandoübersicht		
diff – Suchen von Unterschieden zwischen zwei Dateien		
<i>Kommando</i>	<i>Optionen</i>	<i>Argumente</i>
diff	ja*	<i>Datei1 Datei2</i>
Beschreibung:	Mit dem Kommando diff wird angezeigt, welche Zeilen zweier Dateien unterschiedlich sind, und was zu tun wäre, wollte man die erste Datei identisch mit der zweiten machen.	
Bemerkungen:	Die Anweisungen, wie eine Datei zu ändern ist, um sie an eine zweite anzupassen, bestehen aus Kommandos des Zeileneditors (ed): a (anhängen), c (ändern) und d (löschen). Mit a , c oder d angegebene Zahlen geben an, welche Zeilen zu ändern sind. Das Symbol < bezeichnet eine Zeile der ersten Datei, das Symbol > bezeichnet eine Zeile der zweiten Datei.	

- * Alle verfügbaren Optionen zu diesem Kommando mit einer Erläuterung ihrer Funktionen sind unter **diff(1)** im *User's Reference Manual* zu finden.

Abbildung 3-25: Übersicht über das Kommando **diff**

Zeichenmuster in einer Datei suchen: das Kommando **grep**

Mit dem Kommando **grep** (Abkürzung für "globally search for a regular expression and print" - global nach einem regulären Ausdruck suchen und ihn anzeigen). kann man das UNIX-System anweisen, in einer Datei nach einem bestimmten Wort, Ausdruck oder einer Zeichengruppe zu suchen. Mit dem Begriff "regulärer Ausdruck" wird dabei ein beliebiges Muster aus Zeichen (Wort, Ausdruck oder Formel) bezeichnet, das der Benutzer angeben kann.

Das Grundformat für die Kommandozeile lautet:

```
grep Muster Datei(en) <CR>
```

Will man zum Beispiel in der Datei **hermann** alle Zeilen suchen, in denen das Wort **Automation** vorkommt, muß man folgendes eingeben:

```
grep Automation hermann <CR>
```

Das System antwortet mit folgender Ausgabe:

```
$ grep Automation hermann <CR>
die geplante Automation in Ihrer Firma
$
```

Als Ausgabe erscheinen alle Zeilen der Datei **hermann**, die das Muster enthalten, nach dem gesucht wird (hier **Automation**).

Enthält das Muster mehrere Wörter oder ein oder mehrere Zeichen, die für das UNIX-System eine Sonderbedeutung haben (wie zum Beispiel \$, |, *, ?), muß das Muster in einfache Anführungszeichen gesetzt werden (die Sonderbedeutungen dieser und weiterer Zeichen werden unter "Maskierungszeichen" in Kapitel 7 erläutert). Sucht man zum Beispiel die Zeilen, in denen das Muster **Software für Büroautomation** vorkommt, sieht die Kommandozeile und die entsprechende Reaktion des Systems folgendermaßen aus:

```
$ grep 'Software für Büroautomation' hermann <CR>
und Software für Büroautomation.
$
```

Kann man sich nicht mehr daran erinnern, in welchem Brief von **Software** zur **Büroautomation** die Rede war, d. h. im Brief an Herrn **Hermann** oder in dem an Frau **Sanders**, kann man dies mit folgender Kommandozeile herausfinden:

```
$ grep 'Software für Büroautomation' hermann sanders <CR>
hermann:und Software für Büroautomation.
$
```

Mit dieser Ausgabe wird dem Benutzer angezeigt, daß das Muster **Software für Büroautomation** in der Datei **hermann** einmal enthalten ist.

Zum Kommando **grep** bietet das UNIX-System noch zwei Varianten, **egrep** bzw. **fgrep**, die die Suchfunktion mit einigen Optionen verbessern. Nähere Angaben zu diesen Kommandos sind unter **grep(1)**, **egrep(1)** und **fgrep(1)** im *User's Reference Manual* zu finden.

In Abbildung 3-26 werden die Syntax und die Funktionen des Kommandos `grep` zusammengefaßt.

Kommandoübersicht		
grep – In einer Datei nach einem Muster suchen		
<i>Kommando</i>	<i>Optionen</i>	<i>Argumente</i>
grep	ja*	<i>Muster Datei(en)</i>
Beschreibung:	Mit dem Kommando grep werden die angegebenen Datei(en) nach einem bestimmten Muster durchsucht; die Zeilen, die dieses Muster enthalten, werden dann angezeigt. Werden mehrere Dateien zum Suchen angegeben, wird außerdem der Name der Datei mit angezeigt, in der das Muster gefunden wurde.	
Bemerkungen:	Enthält das Suchmuster mehrere Wörter oder Sonderzeichen, ist es in der Kommandozeile in einfache Anführungszeichen zu setzen.	

* Alle verfügbaren Optionen zu diesem Kommando mit einer Erläuterung ihrer Funktionen sind unter `grep(1)` im *User's Reference Manual* zu finden.

Abbildung 3-26: Übersicht über das Kommando `grep`

Sortieren und Mischen von Dateien: das Kommando `sort`

Im UNIX-System steht ein wirkungsvolles Werkzeug zum Sortieren und Mischen von Dateien zur Verfügung: das Kommando `sort`. Das Format für die Kommandozeile lautet:

`sort Datei(en) <CR>`

Mit diesem Kommando können die Zeilen der angegebenen Datei(en) in der folgenden Reihenfolge sortiert und gemischt werden:

- Zeilen, die mit Zahlen beginnen, werden nach der Ziffer sortiert und vor Zeilen aufgelistet, die mit Buchstaben beginnen.
- Zeilen, die mit Großbuchstaben beginnen, werden vor Zeilen aufgelistet, die mit Kleinbuchstaben beginnen.
- Zeilen, die mit Symbolen wie *, % oder @ beginnen, werden nach dem ASCII-Wert der Symbole sortiert.

Im folgenden Beispiel enthalten zwei Dateien, **Gruppe1** und **Gruppe2**, jeweils eine Liste mit Namen. Sie sollen alphabetisch sortiert und dann zu einer Liste vereinigt werden. Zuerst kann man sich den Inhalt der Dateien über das Kommando **cat** anzeigen lassen:

```
$ cat Gruppe1 <CR>
Schmid, Albin
Jehle, Barbara
Christ, Karin
Maier, Peter
Wolf, Robert
$ cat Gruppe2 <CR>
Frank, Manfred
Nauer, Johannes
Westermann, Doris
Hiller, Karl
Morgen, Kristine
$
```

Zum Anzeigen dieser Dateien muß das Kommando nicht zweimal ausgeführt werden, da man die beiden Dateien auch in einer Kommandozeile zusammen angeben kann. In diesem Fall hätte die Eingabe **cat Gruppe1 Gruppe2** (RETURN-Taste) gelaute, die Ausgabe wäre dieselbe.

Nun kann man durch Ausführen des Kommandos `sort` den Inhalt der beiden Dateien sortieren und mischen. Wenn nicht anders angegeben, wird die Ausgabe des Programms `sort` auf dem Bildschirm des Terminals ausgegeben.

```
$ sort Gruppe1 Gruppe2<CR>
Christ, Karin
Frank, Manfred
Hiller, Karl
Jehle, Barbara
Maier, Peter
Morgen, Kristine
Nauer, Johannes
Schmid, Albin
Westermann, Doris
Wolf, Robert
$
```

Neben dem Mischen einfacher Listen wie im obigen Beispiel kann man mit dem Kommando `sort` Zeilen und Teile von Zeilen (Felder genannt) auf der Basis einer Anzahl anderer Angaben in der Kommandozeile neu anordnen. Diese Angaben sind sehr komplex und würden den Rahmen dieses Handbuchs sprengen. Eine vollständige Beschreibung der verfügbaren Optionen ist im *User's Reference Manual* enthalten.

In Abbildung 3-27 werden die Syntax und die Funktionen des Kommandos `sort` zusammengefaßt.

Kommandoübersicht		
sort – Sortieren und Mischen von Dateien		
<i>Kommando</i>	<i>Optionen</i>	<i>Argumente</i>
sort	ja*	<i>Datei(en)</i>
Beschreibung:	Mit dem Kommando sort können Zeilen einer oder mehrerer Dateien sortiert und gemischt werden; die Ausgabe erscheint standardmäßig auf dem Terminal.	
Bemerkungen:	Werden in der Kommandozeile keine Optionen angegeben, werden die Zeilen nach dem ASCII-Wert der Zeichen der Zeile sortiert und gemischt.	

* Alle verfügbaren Optionen zu diesem Kommando mit einer Erläuterung ihrer Funktionen sind unter **sort(1)** im *User's Reference Manual* zu finden.

Abbildung 3-27: Übersicht über das Kommando **sort**



Zusammenfassung

In diesem Kapitel wurde die Struktur des Dateisystems beschrieben, und es wurden Möglichkeiten zu seiner Benutzung und zur Orientierung im Dateisystem mit Hilfe von UNIX-Systemkommandos erläutert. Das nächste Kapitel enthält eine Übersicht über eine Reihe von Möglichkeiten, die dem Benutzer im UNIX-System zur Verfügung stehen: Bearbeitung von Texten, Verwendung der Shell als Kommandosprache, elektronische Kommunikation mit anderen Benutzern des Systems, sowie Software zur Programmierung und Programmentwicklung.

Kapitel 4: ÜBERSICHT ÜBER DIE ANLEITUNGEN

Einführung	4-1
Textbearbeitung	4-2
Allgemeine Informationen zu den Texteditoren	4-2
Arbeitsweise eines Texteditors	4-3
Puffer für die Textbearbeitung	4-3
Betriebsarten	4-4
Der Zeileneditor	4-4
Der Bildschirmeditor	4-5
Die Shell	4-7
Anpassen der Benutzerumgebung	4-8
Programmieren in der Shell	4-9
Elektronische Kommunikation	4-12
Programmieren im System	4-14

Einführung

Mit diesem Kapitel wird von den Übersichten der ersten drei Kapitel zu den Anleitungen in den folgenden vier Kapiteln übergeleitet. Insbesondere wird dem Leser in diesem Kapitel ein Überblick über die Themen dieser Anleitungen gegeben: Textbearbeitung, Arbeiten in der Shell sowie elektronische Kommunikation. Die Bearbeitung von Texten wird in Kapitel 5, "Anleitung zum Zeileneditor `ed`" und Kapitel 6, "Anleitung zum Bildschirmeditor `vi`" behandelt. In Kapitel 7, "Anleitung zur Shell", lernt der Benutzer, mit der Shell zu arbeiten und zu programmieren. Die Verfahren zur elektronischen Kommunikation werden schließlich in Kapitel 8, "Anleitung zur Kommunikation", behandelt.

Textbearbeitung

Die wichtigste Möglichkeit, in einer UNIX-Systemumgebung zu arbeiten, besteht darin, das Dateisystem zu benutzen. In diesem Abschnitt wird behandelt, wie Dateien mit einem Softwarewerkzeug erstellt und geändert werden können; dieses Softwarewerkzeug nennt man einen Texteditor. Am Anfang dieses Abschnitts wird erläutert, was ein Texteditor ist und wie er arbeitet. Danach werden zwei Typen von Texteditoren eingeführt, die mit dem UNIX-System verwendet werden können: der Zeileneditor `ed` und der Bildschirmeditor `vi` ("visual editor"). Die beiden Editoren werden dabei auch vergleichend betrachtet. Ausführliche Angaben zu `ed` und `vi` sind in den Kapiteln 5 und 6 enthalten.

Allgemeine Informationen zu den Texteditoren

Beim Ausarbeiten eines Briefes, einer Mitteilung oder eines Berichtes sind stets eine oder mehrere der folgenden Aufgaben zu erledigen: Einfügen neuer bzw. zusätzlicher Informationen, Löschen überflüssiger Informationen, Übertragen von Informationen und schließlich das Vorbereiten einer sauber gestalteten und korrigierten Version. Texteditoren führen diese Aufgaben nach den Anweisungen des Benutzers durch, und mit ihnen läßt sich das Schreiben und Überarbeiten von Texten viel einfacher und schneller als von Hand erledigen.

Die Texteditoren des UNIX-Systems sind wie die Shell des UNIX-Systems interaktive Programme: sie nehmen die Kommandos des Benutzers entgegen und führen dann die entsprechenden Funktionen aus. Von der Shell aus gesehen sind die Editoren ausführbare Programme.

Ein wichtiger Unterschied zwischen einem Texteditor und der Shell liegt jedoch im Kommandosatz, den sie verarbeiten. Alle bisher eingeführten Kommandos gehören zum Kommandosatz der Shell. Ein Texteditor hat einen anderen, eigenen Kommandosatz, mit dem Text in Dateien erstellt, verschoben, hinzugefügt und gelöscht werden kann; außerdem kann man damit Text aus anderen Dateien übernehmen.

Arbeitsweise eines Texteditors

Zum Verständnis der Arbeitsweise eines Texteditors muß man etwas über die Umgebung wissen, die hergestellt wird, wenn man ein Textbearbeitungsprogramm aufruft, sowie über die Betriebsarten, in denen ein Texteditor arbeitet.

Puffer für die Textbearbeitung

Ruft man einen Texteditor zum Erstellen einer neuen Datei oder zum Ändern einer bestehenden Datei auf, wird zunächst die Shell angewiesen, die Steuerung der Arbeitssitzung an den betreffenden Editor zu übergeben. Daraufhin wird ein temporärer Arbeitsspeicher eingerichtet, der Editierpuffer genannt wird; alle beim Bearbeiten einer Datei eingegebenen Informationen werden in diesem Puffer gespeichert und können dort geändert werden.

Da es sich um einen temporären Arbeitsspeicher handelt, sind alle Eingaben und Änderungen darin ebenfalls nur temporär. Der Puffer und sein Inhalt bestehen nur solange man den Text bearbeitet. Soll die Datei abgespeichert werden, muß der Texteditor angewiesen werden, den Inhalt des Puffers in eine Datei zu schreiben. Diese Datei wird dann im Speicher des Rechners gespeichert. Gibt man keine solche Anweisung, verschwindet der Inhalt des Puffers beim Verlassen des Textbearbeitungsprogramms. Um zu vermeiden, daß dies versehentlich geschieht, erinnert der Texteditor den Benutzer ans Speichern, wenn man ohne diese Anweisung eine Editiersitzung zu beenden versucht.

NOTE

Hat man einen schwerwiegenden Fehler gemacht oder ist man mit der überarbeiteten Version einer Datei nicht zufrieden, kann man den Texteditor auch verlassen, ohne die Datei abzuspeichern. Dadurch bleibt die ursprüngliche Version der Datei erhalten, und die überarbeitete Version existiert nicht mehr.

Unabhängig davon, ob man eine neue Datei erstellt oder eine bestehende überarbeitet, wird der Text im Puffer nach Zeilen strukturiert. Eine Textzeile besteht einfach aus einer Reihe von Buchstaben, die waagrecht auf dem Bildschirm erscheint und durch Drücken der RETURN-Taste abgeschlossen wird. Es kann vorkommen, daß Dateien Textzeilen enthalten, die länger als eine Bildschirmbreite sind. Bei bestimmten Terminals wird die Fortsetzung solcher Zeilen automatisch in der nächsten Zeile des Bildschirms angezeigt. Dies gilt jedoch nicht für alle Terminals.

Betriebsarten

Texteditoren arbeiten in zwei Betriebsarten: Kommandomodus und Texteingabemodus. Zu Beginn einer Editiersitzung befindet man sich stets im Kommandomodus. In diesem Modus kann man sich in einer Datei bewegen, in ihr nach bestimmten Zeichenmustern suchen oder bestehenden Text ändern. Im Kommandomodus kann jedoch kein Text erstellt werden; dazu muß man in den Texteingabemodus wechseln. In diesem Modus werden alle eingegebenen Zeichen als Teil der Textdatei in den Puffer geschrieben. Sollen nach der Eingabe von Text wieder Kommandos zur Textbearbeitung ausgeführt werden, muß man in den Kommandomodus zurückkehren.

Da in einer normalen Editiersitzung häufig zwischen diesen beiden Modi umgeschaltet wird, kann es vorkommen, daß man vergißt, in welchem Modus man gerade arbeitet. Dabei versucht man beispielsweise, Text im Kommandomodus oder ein Kommando im Eingabemodus einzugeben. Dies kommt auch bei geübten Benutzern gelegentlich vor. Man kann jedoch solch einen Fehler schnell erkennen und beheben, wenn man die Anleitungen in Kapitel 5 und 6 durchgearbeitet hat.

Der Zeileneditor

Der Zeileneditor, der mit dem Kommando `ed` aufgerufen wird, ist ein schnelles und vielseitiges Programm zur Erstellung von Textdateien. Er wird Zeileneditor genannt, da er Text zeilenweise bearbeitet. Das bedeutet, daß man die Zeilennummer der Zeile angeben muß, deren Text man ändern will. Der Texteditor `ed` gibt daraufhin die Zeile auf dem Bildschirm aus; dort kann man sie dann ändern.

Dieser Texteditor bietet Kommandos, mit denen Zeilen geändert und gedruckt, Dateien angezeigt und gelesen sowie Text eingegeben werden kann. Außerdem kann man den Zeileneditor aus einem Shell-Programm aufrufen; dies ist mit dem Bildschirmditor nicht möglich (Informationen zu einfachen Programmiermethoden mit der Shell sind in Kapitel 7 enthalten).

Der Zeileneditor (`ed`) ist für Bildschirmterminals und Druckerterminals gut geeignet, ebenso für die Benutzung einer langsamen Telefon-Übertragungsleitung (der Bildschirmditor `vi` kann nur bei Bildschirmterminals eingesetzt werden). In Kapitel 5, "Anleitung zum Zeileneditor", sind Anweisungen zur Verwendung dieses Werkzeugs zur Textbearbeitung enthalten. Eine Zusammenfassung der Kommandos des Zeileneditors ist in Anhang C zu finden.

Der Bildschirmeditor

Der Bildschirmeditor - er wird mit dem Kommando **vi** aufgerufen - ist ein bildschirmorientiertes interaktives Softwarewerkzeug. Er ermöglicht es dem Benutzer, die Datei, die er bearbeitet, jeweils seitenweise anzusehen. Dieser Editor arbeitet am besten, wenn er mit einem Bildschirmterminal eingesetzt wird, das mit 1200 oder mehr Baud betrieben wird.

In der Regel wird mit diesem Editor eine Datei geändert (durch Einfügen, Löschen oder Ändern von Text), indem man den Cursor auf den Punkt des Bildschirms setzt, an dem die Änderung vorgenommen werden soll und diese dann ausführt. Das Ergebnis einer Änderung wird sofort im Kontext angezeigt! Aufgrund dieser Eigenschaften wird ein Bildschirmeditor als leistungsfähiger betrachtet als ein Zeileneditor.

Darüber hinaus stehen im Bildschirmeditor eine Reihe von Kommandos zur Verfügung. Dazu gehören Kommandos, mit denen der Cursor innerhalb einer Datei bewegt wird, Kommandos, mit denen die Datei auf dem Bildschirm aufwärts oder abwärts durchgeblättert werden kann, sowie Kommandos, mit denen man bestehenden Text ändern oder neuen Text erstellen kann. Neben diesem eigenen Kommandosatz kann der Bildschirmeditor auch auf Kommandos des Zeileneditors zugreifen.

Der Nachteil, der für die Geschwindigkeit, die ansprechende Ausgabe, die Effektivität und Leistungsfähigkeit des Bildschirmeditors in Kauf zu nehmen ist, besteht darin, daß er die Rechenkapazität des Rechners stark beansprucht. Bei der kleinsten Änderung muß die ganze Bildschirmanzeige vom Programm **vi** aktualisiert werden. Anweisungen zur Benutzung dieses Editors sind in Kapitel 6, "Anleitung zum Bildschirmeditor" enthalten. In Anhang D werden die Kommandos des Bildschirmeditors zusammengefaßt, und Abbildung 4-1 enthält einen Vergleich zwischen den Eigenschaften des Zeileneditors (**ed**) und denen des Bildschirmeditors (**vi**).

Merkmal	Zelleneditor (ed)	Bildschirmeditor (vi)
Empfohlener Terminaltyp	Bildschirm- oder Druckerterminal	Bildschirmterminal
Geschwindigkeit	Langsame und schnelle Datenübertragungsverbindungen	Am günstigsten mit schnellen Übertragungsverbindungen (> 1200 Baud).
Flexibilität	Aufruf aus Shell-Skripts sowie Benutzung in Editiersitzungen	Interaktive Verwendung in Editiersitzungen
Komplexität	Schnelles Ändern von Texten; relativ geringer Aufwand an Prozessorkapazität	Einfaches Ändern von Texten; beansprucht die Prozessorkapazität jedoch stark
Leistung	Voller Kommandosatz zum Editieren; UNIX-Standardtexteditor	Eigene Editierkommandos und Kommandos des Zelleneditors stehen zur Verfügung
Vorteile	Der Benutzer braucht nur wenige Kommandos zu erlernen, um ed anzuwenden	Mit vi werden die Auswirkungen vorgenommener Änderungen im Zusammenhang einer Textseite sofort sichtbar (beim Editor ed sind Änderungen und Anzeigen der Ergebnisse getrennte Vorgänge)

Abbildung 4-1: Vergleich von Zeilen- und Bildschirmeditor (ed bzw. vi)

Die Shell

Bei jeder Anmeldung am UNIX-System erfolgt die Kommunikation über die Shell, und diese wird solange aufrechterhalten, bis man sich vom System abmeldet. Während man einen Texteditor verwendet, ist die Interaktion mit der Shell jedoch vorübergehend ausgesetzt; sie wird wieder aufgenommen, sobald man die Arbeit mit dem Editor beendet.

Die Shell entspricht in vielem anderen Programmen, anstatt jedoch jeweils nur einen Auftrag auszuführen, wie es zum Beispiel bei `cat` oder `ls` der Fall ist, steht die Shell im Mittelpunkt der Interaktion des Benutzers mit dem UNIX-System. Die wichtigste Funktion der Shell ist die eines Kommandointerpreters zwischen dem Benutzer und dem Rechnersystem. Als Interpreter übersetzt die Shell die Anforderungen des Benutzers in eine Sprache, die der Rechner verstehen kann, lädt aufgerufene Programme in den Speicher und führt sie aus.

In diesem Abschnitt werden Verfahren beschrieben, wie der Benutzer die Shell so einsetzen kann, daß er damit die Systemeinrichtungen besser nutzen kann. Neben dem Aufrufen der gewünschten Programme kann man die Shell auch für folgende Funktionen einsetzen:

- Interpretieren des Namens einer Datei oder eines Verzeichnisses, der in abgekürzter Form eingegeben wird.
- Umlenken der Ein-/Ausgabe der ausgeführten Programme.
- Ausführen mehrerer Programme gleichzeitig oder in Form einer Pipe.
- Anpassen der Rechnerumgebung an die speziellen Anforderungen des Anwenders.

Neben ihrer Funktion als Kommandointerpreter ist die Shell auch eine Programmiersprache. Ausführliche Informationen über die Verwendung der Shell als Kommandointerpreter und Programmiersprache sind in Kapitel 7 enthalten.

Anpassen der Benutzerumgebung

In diesem Abschnitt wird eine weitere Funktion beschrieben, die die Shell bietet: die Benutzerumgebung. Bei der Anmeldung am UNIX-System richtet die Shell automatisch eine Arbeitsumgebung für den Benutzer ein. In der Standardkonfiguration dieser Umgebung sind die folgenden Variablen enthalten:

HOME	das Anmelde- bzw. Benutzerverzeichnis
LOGNAME	der Benutzername
PATH	der Pfad, den die Shell nimmt, wenn sie nach ausführbaren Dateien bzw. Kommandos des Benutzers sucht (in der Regel <code>PATH=:/bin:/usr/bin</code>).

Die Variable `PATH` liefert der Shell die Information, wo sie nach dem mit einem Kommando aufgerufenen ausführbaren Programm suchen soll. Sie wird daher jedesmal verwendet, wenn ein Kommando eingegeben wird. Hat ein Benutzer ausführbare Programme in mehreren Verzeichnissen, muß er die Variable so definieren, daß alle diese Verzeichnisse durchsucht werden, damit das Programm in jedem Fall gefunden wird.

Der Benutzer kann die Standardumgebung verwenden, die das System vorgibt, oder er kann sich eine Umgebung an seine Anforderungen anpassen. Eine Änderung in der Umgebung des Benutzers kann in zwei Formen erfolgen. Soll ein Teil der Umgebung nur für die aktuelle Sitzung geändert werden, sind die Änderungen in einer Kommandozeile einzugeben (siehe Kapitel 7). Soll jedoch ständig eine andere Umgebung als die Standardumgebung verwendet werden, können die Änderungen in einer Datei definiert werden, über die diese Umgebung bei jeder Anmeldung des Benutzers wiederhergestellt wird. Diese Datei muß den Namen `.profile` haben und im Home-Verzeichnis des Benutzers stehen.

Die Datei `.profile` wird für einige oder alle der folgenden Funktionen verwendet: Prüfen, ob Post eingegangen ist; Definition von Datenparametern, Terminalcharakteristika und Tabulatoren; Zuweisen eines Zeichens oder einer Zeichenkette als Eingabeaufforderung bei der Anmeldung; Umlegen der Löschfunktionen für Zeichen und Zeilen auf bestimmte Tasten. In der Datei `.profile` können beliebig viele dieser Funktionen definiert werden. Man kann auch jederzeit Teile davon ändern. Anweisungen zum Ändern einer `.profile`-Datei sind unter "Anpassen der Benutzerumgebung" in Kapitel 7 enthalten.

An dieser Stelle ist zu prüfen, ob eine Datei `.profile` vorhanden ist. Befindet man sich noch nicht im eigenen Home-Verzeichnis, ist mit `cd` in dieses zu wechseln. Dann die Datei `.profile` mit folgendem Kommando anzeigen lassen:

```
cat .profile
```

Besteht eine Datei `.profile`, wird ihr Inhalt auf dem Bildschirm ausgegeben. Besteht noch keine Datei `.profile`, kann man eine solche mit einem Texteditor wie `ed` oder `vi` erstellen (Anweisungen dazu sind unter "Anpassen der Benutzerumgebung" in Kapitel 7 zu finden).

Programmieren in der Shell

Die Shell ist nicht lediglich der Kommandointerpreter, sondern auch eine Programmiersprache auf Kommandoebene. Das bedeutet, daß die Shell nicht nur als direktes Bindeglied zwischen dem Benutzer und dem Rechner dient, sondern auch so programmiert werden kann, daß sie bestimmte Anweisungsfolgen automatisch wiederholt. Dazu müssen ausführbare Dateien erstellt werden, die Listen von Kommandos enthalten. Solche Dateien werden Shell-Prozeduren oder Shell-Skripts genannt. Hat man einmal ein solches Shell-Skript für eine bestimmte Aufgabe erstellt, braucht man lediglich die Shell anzuweisen, den Inhalt des Shell-Skripts zu lesen und auszuführen, wenn diese Aufgabe ausgeführt werden soll.

Wie andere Programmiersprachen bietet die Shell Programmierfunktionen wie zum Beispiel Variablen, Steuerstrukturen, Unterprogramme und Parameterübergabe. Mit diesen Funktionen kann man eigene Werkzeuge entwickeln, indem man Systemkommandos verknüpft.

Beispielsweise kann man die drei UNIX-Systemprogramme `date`, `who` und `wc` zu einem einfachen Shell-Skript mit dem Namen `benutzer` verknüpfen, das das aktuelle Datum und die Uhrzeit anzeigt und angibt, wieviele Benutzer am System arbeiten. Verwendet man den Editor `vi` (in Kapitel 6 beschrieben), um das Skript zu erstellen, kann man das folgende Verfahren anwenden. Zuerst ist die Datei `benutzer` mit dem Editor durch folgende Eingabe anzulegen:

```
vi benutzer<CR>
```

Der Editor gibt dann eine leere Seite auf dem Bildschirm aus und erwartet die Texteingabe.

```
Cursor
~
~
~
~
~
~
~
~
~
~
"benutzer" [New file]
```

Nun sind die drei UNIX-Systemkommandos in einer Zeile einzugeben:

```
date; who |wc -l
```

Dann speichern Sie die Datei ab und verlassen den Editor. Machen Sie die Datei **benutzer** ausführbar, indem Sie mit dem Kommando **chmod** das Ausführbarkeitsrecht ändern.

```
chmod ug+x benutzer<CR>
```

Nun kann man versuchen, das neue Kommando auszuführen. Auf dem folgenden Bildschirm wird gezeigt, welche Ausgabe man damit erhält:

```
$ benutzer<CR>
Wed Apr 26 16:27:10 MES 1989
  4
$
```

Die Ausgabe besagt, daß bei Eingabe des Kommandos am Mittwoch, 26. April 1989 um 16.25 Uhr, vier Benutzer angemeldet waren.

Eine schrittweise Einführung in das Erstellen von Shell-Skripts und Informationen über komplexere Programmierverfahren mit der Shell sind in Kapitel 7, "Anleitung zur Shell", enthalten.

Elektronische Kommunikation

Der Benutzer eines UNIX-Systems kann Nachrichten oder in Dateien gespeicherte Informationen an andere Benutzer desselben oder eines anderen UNIX-Systems übermitteln. Zu diesem Zweck muß man an einem UNIX-System angemeldet sein, das in der Lage ist, mit dem UNIX-System Verbindung aufzunehmen, an das man die betreffenden Informationen schicken will. Das dafür verwendete Kommando hängt davon ab, was für Informationen übermittelt werden sollen. In diesem Handbuch werden folgende Kommunikationsprogramme eingeführt:

- | | |
|--------------------|--|
| mail | Mit diesem Kommando können Nachrichten oder Dateien an andere Benutzer eines UNIX-Systems geschickt werden, und zwar unter Verwendung ihrer Benutzernamen als Adressen. Ebenso kann man damit auch Nachrichten empfangen, die von anderen Benutzern geschickt wurden. Das Programm mail speichert Nachrichten, so daß sie der Empfänger lesen kann dann, wenn es für ihn am günstigsten ist. |
| mailx | Dieses Programm ist eine leistungsfähigere Version des Programms mail . Es verfügt über eine Anzahl von Optionen zur Verwaltung der abgesandten und eingegangenen elektronischen Post. |
| uucp | Dieses Kommando wird verwendet, um Dateien von einem UNIX-System zu einem anderen zu übertragen (der Name leitet sich ab von "UNIX to UNIX system copy" - Kopie von UNIX-System zu UNIX-System). Das Programm uucp dient dazu, eine Datei in ein bestimmtes Verzeichnis auf einem fernen Rechner zu übertragen. Nach der Übertragung wird der Eigentümer dieses Verzeichnisses vom Programm mail benachrichtigt, daß sie angekommen ist. |
| uuto/uupick | Diese Kommandos werden verwendet, um Dateien zu senden bzw. abzurufen. Mit dem Kommando uuto können eine oder mehrere Dateien in ein öffentliches Verzeichnis gesandt werden; wenn die Datei bzw. die Dateien dort angekommen sind, wird der Empfänger vom Programm "mail" benachrichtigt. Er kann dann die Datei(en) mit dem Kommando uupick aus dem öffentlichen Verzeichnis in ein Verzeichnis seiner Wahl kopieren. |

uuu

Mit diesem Programm können Kommandos auf einem fernen Rechner ausgeführt werden. Dateien können von verschiedenen Rechnern gesammelt werden, um dann ein Kommando auf diese Dateien anzuwenden und die Standardausgabe in eine Datei auf dem angegebenen Rechner zu senden.

Anleitungen zu diesen Kommandos sind in Kapitel 8 enthalten.

Programmieren im System

Das UNIX-System bietet eine leistungsfähige und leicht handhabbare Umgebung zur Programmierung und Entwicklung von Software mit den Programmiersprachen C, FORTRAN-77, Pascal und COBOL. Darüber hinaus bietet das UNIX-System einige leistungsfähige Werkzeuge, die die Softwareentwicklung erleichtern und einen systematischen Ansatz zur Programmierung unterstützen.

Allgemeine Informationen über das Programmieren in einer UNIX-Umgebung sind im *Programmer's Guide* zu finden. Neben ergänzenden Hinweisen zu den Programmiersprachen sind im *Programmer's Guide* Anleitungen zu folgenden fünf Werkzeugen enthalten:

SCCS	Source Code Control System - Quellencode-Controll-System
make	zur Pflege von Programmen
lex	zur Generierung von Programmen für einfache lexikalische Aufgaben
yacc	zur Generierung von Parser-Programmen.

Kapitel 5: ANLEITUNG ZUM ZEILENEDITOR (ed)

Einführung in den Zeileneditor	5-1
Hinweise zur Verwendung dieser Anleitung	5-2
Einführung	5-3
Aufrufen des Zeileneditors ed	5-3
Text erstellen	5-4
Text anzeigen	5-5
Löschen einer Textzeile	5-8
Bewegen innerhalb der Datei	5-9
Abspeichern des Pufferinhalts in einer Datei	5-10
Verlassen des Editors	5-12
Übung 1	5-15
Allgemeines Format der Kommandos des Editors ed	5-16
Adressierung von Zeilen	5-17
Numerische Adressen	5-18
Symbolische Adressen	5-19
Symbolische Adresse der aktuellen Zeile	5-19
Symbolische Adresse der letzten Zeile	5-20
Symbolische Adresse: Alle Zeilen	5-21
Symbolische Adresse: Aktuelle Zeile bis letzte Zeile	5-21
Relative Adressen: Addieren bzw. Subtrahieren von Zeilen von der aktuellen Zeile	5-22

Adressierung mit Zeichenketten	5-24
Angabe eines Zeilenbereiches	5-26
Globale Suche	5-28
Übung 2	5-32
Text einer Datei anzeigen lassen	5-33
Text alleine anzeigen: das Kommando p	5-33
Text mit Zeilenadressen anzeigen: das Kommando n	5-34
Text erstellen	5-36
Text anhängen: das Kommando a	5-36
Text einfügen: das Kommando i	5-39
Text ändern: das Kommando c	5-41
Übung 3	5-44
Text löschen	5-46
Löschen von Zeilen: das Kommando d	5-46
Zuletzt eingegebenes Kommando rückgängig machen: das Kommando u	5-48
Löschen von Text im Texteingabemodus	5-50
Löschfunktion außer Kraft setzen	5-50
Text ersetzen	5-52
Ersetzen in der aktuellen Zeile	5-53
Ersetzen in einer bestimmten Zeile	5-55
Ersetzen in einem Zeilenbereich	5-55
Globales Ersetzen	5-57

Übung 4	5-61
Sonderzeichen	5-63
Übung 5	5-74
Text verschieben/kopieren	5-76
Textzeilen verschieben	5-76
Textzeilen kopieren	5-78
Aufeinanderfolgende Zeilen zusammenfügen	5-80
Textzeilen in eine Datei schreiben	5-81
Wichtige Hinweise	5-82
Inhalt einer Datei einlesen	5-83
Übung 6	5-85
Weitere nützliche Kommandos und Hinweise	5-86
Hilfekommandos	5-86
Nicht druckbare Zeichen anzeigen lassen	5-89
Der aktuelle Dateiname	5-90
Shell-Kommandos aus dem Editor ausführen	5-92
Wiederherstellung nach Systemunterbrechungen	5-93
Zusammenfassung	5-94
Übung 7	5-96
Lösungen zu den Übungen	5-97
Übung 1	5-97



Übung 2	5-99
Übung 3	5-102
Übung 4	5-105
Übung 5	5-108
Übung 6	5-111
Übung 7	5-114

Einführung in den Zeileneditor

Dieses Kapitel ist eine Anleitung zur Benutzung des Zeileneditors **ed**. **ed** ist ein vielseitiges Programm, das für die Ausführung von Editieraufgaben nur wenig Prozessorkapazität beansprucht. Es kann mit jedem Terminaltyp verwendet werden. Die Beispiele für Kommandozeilen und die entsprechenden Reaktionen des Systems, die in diesem Kapitel erscheinen, gelten für das verwendete Terminal, unabhängig davon, ob es sich um ein Bildschirmterminal oder ein Druckerterminal handelt. Die Kommandos von **ed** können am Terminal eingegeben oder innerhalb eines Shell-Programms verwendet werden (siehe dazu Kapitel 7, "Anleitung zur Shell").

Das Programm **ed** ist ein Zeileneditor. Bei den Editiersitzungen zeigt es jeweils auf eine bestimmte Zeile in der Datei; diese wird "aktuelle Zeile" genannt. Beim Zugriff auf eine bestehende Datei nimmt **ed** zunächst die letzte Zeile als aktuelle Zeile an, so daß man Text einfach anfügen kann. Gibt man keine andere Zeile bzw. keinen Bereich von Zeilen an, führt **ed** jedes eingegebene Kommando mit der aktuellen Zeile aus. Neben dem Ändern, Löschen oder Einfügen von Text in eine oder mehrere Zeilen ermöglicht **ed** die Übernahme von Text aus einer anderen Datei in den Puffer.

Während einer Editiersitzung mit **ed** ändert man den Inhalt einer Datei in einem temporären Pufferspeicher, in dem man arbeitet, bis man die Erstellung bzw. Korrektur des Textes beendet hat. Editiert man eine bestehende Datei, wird eine Kopie davon in den Puffer gebracht, und die Änderungen werden in dieser Kopie vorgenommen. Diese Änderungen sind in der ursprünglichen Datei erst wirksam, wenn man **ed** mit dem Kommando **write** anweist, den Inhalt des Puffers in der Datei abzuspeichern.

Nach dem Durcharbeiten dieser Anleitung sowie der Beispiele und Übungen verfügt der Benutzer über gute Grundkenntnisse zur Benutzung des Zeileneditors **ed**. Dazu gehören folgende Grundlagen:

- Aufrufen des Zeileneditors **ed**, Erstellen von Text, Schreiben von Text in eine Datei (Abspeichern) und Verlassen des Zeileneditors **ed**
- Adressieren bestimmter Zeilen in der Datei und Anzeigen von Textzeilen
- Löschen von Text
- Alten Text durch neuen Text ersetzen
- Sonderzeichen als Kürzel in Such- und Ersetzmustern verwenden
- Text innerhalb einer Datei verschieben sowie weitere nützliche Kommandos und Hinweise.

Hinweise zur Verwendung dieser Anleitung

Die in den einzelnen Abschnitten behandelten Kommandos werden am Ende des jeweiligen Abschnitts wiederholt. Eine Zusammenfassung aller Kommandos von **ed**, die in diesem Kapitel eingeführt werden, ist in Anhang C enthalten; sie sind dort thematisch geordnet.

Am Ende einiger Abschnitte sind Übungen vorgesehen, damit der Benutzer die Kommandos einüben kann. Die Lösungen zu diesen Übungen befinden sich am Ende dieses Kapitels.

Die Notationskonventionen in diesem Kapitel sind dieselben, wie sie auch in den übrigen Teilen dieses *Handbuches* verwendet werden; sie sind im Vorwort beschrieben.

Einführung

Der einfachste Weg, den Editor **ed** zu erlernen, besteht darin, sich am UNIX-System anzumelden und die Beispiele durcharbeiten, während man diese Anleitung liest. Man sollte die Übungen durchführen und sich nicht scheuen, auch zu experimentieren. Indem man mit Kommandos des Editors **ed** experimentiert und sie anwendet, erlernt man eine schnelle und flexible Methode der Textbearbeitung.

Im folgenden Abschnitt werden die Kommandos für folgende Aufgaben eingeführt:

- Aufrufen von **ed**
- Text anfügen
- Bewegen innerhalb der Datei, um eine bestimmte Textzeile anzuzeigen
- Eine Textzeile löschen
- Den Puffer in einer Datei abspeichern
- **ed** verlassen.

Aufrufen des Zeileneditors **ed**

Der Zeileneditor wird durch Eingabe von **ed** mit einem Dateinamen aufgerufen:

```
ed Dateiname <CR>
```

Erstellt man eine neue Datei, sollte man einen Dateinamen wählen, der zum Inhalt der Datei in Beziehung steht. Das System antwortet mit einem Fragezeichen und dem Dateinamen:

```
$ ed neu-datei <CR>  
?neu-datei
```

Bearbeitet man eine bereits bestehende Datei, antwortet **ed** mit der Anzahl der Zeichen in der Datei:

```
$ ed alt-datei <CR>  
235
```

Text erstellen

Der Editor empfängt zwei Typen von Eingaben vom Terminal des Benutzers: Editierkommandos und Text. Um zu verhindern, daß diese verwechselt werden, kennt **ed** zwei Modi für die Textbearbeitung: Kommandomodus und Texteingabemodus. Im Kommandomodus werden alle eingegebenen Zeichen als Kommandos interpretiert, im Texteingabemodus werden sie als Text zum Einfügen in eine Datei interpretiert.

Beim Aufrufen von **ed** ist stets automatisch der Kommandomodus aktiv. Will man Text in die Datei eingeben, muß man in den Texteingabemodus wechseln, indem man in eine eigene Zeile **a** (für anfügen) eingibt und dann die RETURN-Taste drückt.

a<CR>

Nun befindet man sich im Eingabemodus; alle von nun an eingegebenen Zeichen werden als Textzeichen in die Datei eingefügt. Es ist darauf zu achten, daß das oben erwähnte **a** alleine in eine eigene Zeile eingegeben werden muß, da der Editor das Kommando andernfalls nicht ausführt.

Will man die Eingabe von Text beenden, ist ein Punkt alleine in eine eigene Zeile einzugeben. Damit kehrt man aus dem Texteingabemodus in den Kommandomodus zurück. Dann können weitere Kommandos an den Texteditor **ed** gegeben werden.

Im folgenden Beispiel wird gezeigt, wie man **ed** aufruft, Text in eine neue Datei mit dem Namen **versuch** eingibt und den Texteingabemodus mit einem Punkt wieder verläßt.


```
$ ed versuch<CR>
? versuch
a<CR>
Das ist die erste Textzeile.<CR>
Das ist die zweite Zeile.<CR>
und das die dritte.<CR>
.<CR>
```

Hier ist zu beachten, daß **ed** auf den Punkt nicht reagiert, sondern auf ein neues Kommando wartet. Reagiert **ed** nicht auf ein Kommando, kann es sein, daß man vergessen hat, am Ende der Texteingabe einen Punkt einzugeben und sich daher noch im Texteingabemodus befindet. In diesem Fall am Anfang einer Zeile einen Punkt eingeben und die RETURN-Taste drücken; damit kehrt man in den Kommandomodus zurück, in dem man weitere Editierkommandos ausführen kann. Hat man beispielsweise unerwünschte Zeichen oder Zeilen in den Text eingegeben, kann man sie löschen, nachdem man in den Kommandomodus zurückgekehrt ist.

Text anzeigen

Man kann sich eine Zeile einer Datei anzeigen lassen, indem man **p** (für "print") in eine eigene Zeile eingibt. Mit dem Kommando **p** wird die aktuelle Zeile angezeigt, d. h. die zuletzt bearbeitete Zeile. Hier wird nun das obige Beispiel fortgeführt. Es wurde ein Punkt eingegeben, mit dem man den Eingabemodus verlassen hat. Nun muß das Kommando **p** eingegeben werden, um die aktuelle Zeile anzeigen zu lassen.

```
$ ed versuch<CR>
? versuch
a<CR>
Das ist die erste Textzeile.<CR>
Das ist die zweite Zeile<CR>
und das die dritte.<CR>
.<CR>
p<CR>
und das die dritte.
```

Man kann jede Textzeile der Datei anzeigen lassen, indem man ihre Zeilennummer angibt (diese wird auch als Adresse der Zeile bezeichnet). Die Adresse der ersten Zeile ist 1, die der zweiten 2 usw. Mit folgender Eingabe wird die zweite Zeile der Datei **versuch** angezeigt:

```
2p<CR>
Das ist die zweite Zeile
```

Man kann auch Bereiche von Zeilen angeben, die man anzeigen lassen will; dazu gibt man die Adressen der ersten und letzten Zeile des Bereiches an, den man sich anzeigen lassen will; diese Adressen sind durch Komma voneinander zu trennen. Mit folgender Eingabe werden z. B. die ersten drei Zeilen einer Datei angezeigt:

```
1,3p<CR>
```

So kann man die gesamte Datei anzeigen. Eine Datei von 20 Zeilen Länge kann durch Eingabe von **1,20p** angezeigt werden. Kennt man die Adresse der letzten Zeile der Datei nicht, kann man ein Dollarzeichen (\$), das ed-Symbol für die Adresse der letzten Zeile, dafür einsetzen. Diese Konventionen werden im Einzelnen im Abschnitt "Adressierung von Zeilen" behandelt.

```
1,$p<CR>
Das ist die erste Textzeile.
Das ist die zweite Zeile,
und das die dritte.
```

Vergißt man, vom Texteingabemodus mit einem Punkt wieder zurückzukehren, gibt man ungewollt Text ein. Versuchen Sie mal an dieser Stelle den Fehler zu machen. Geben Sie eine weitere Textzeile in die Datei **versuch** ein und dann das Kommando **p**, ohne den Texteingabemodus vorher zu verlassen. Verlassen Sie danach den Texteingabemodus und lassen sich die gesamte Datei anzeigen.

```
p<CR>
und das die dritte.
a<CR>
Das ist die vierte Zeile.<CR>
p<CR>
.<CR>
1,$p<CR>
Das ist die erste Textzeile.
Das ist die zweite Zeile
und das die dritte.
Das ist die vierte Zeile.
p
```

Nun kann man sich das Ergebnis ansehen. Im nächsten Abschnitt wird erklärt, wie man die unerwünschte Zeile wieder löscht.

Löschen einer Textzeile

Man kann eine Textzeile nur im Kommandomodus von `ed` löschen. Man gibt ein `d` ein und löscht damit die aktuelle Zeile. Dieses Kommando nun auf das letzte Beispiel anwenden: d. h. die unerwünschte Zeile mit dem `p` löschen. Dazu die aktuelle Zeile anzeigen lassen (Kommando `p`), sie löschen (Kommando `d`) und dann die in der Datei verbliebenen Zeilen anzeigen lassen (Kommando `p`). Der Bildschirm sollte dabei folgendermaßen aussehen:

```
p<CR>
p
d<CR>
l,$p<CR>
Das ist die erste Textzeile.
Das ist die zweite Zeile
und das die dritte.
Das ist die vierte Zeile.
```

`ed` gibt keine Meldung darüber aus, daß eine Zeile gelöscht wurde. Die einzige Möglichkeit, festzustellen, ob das Kommando `d` erfolgreich ausgeführt wurde, besteht darin, den Inhalt der Datei mit dem Kommando `p` anzeigen zu lassen. Will man den Löschvorgang überprüfen, kann man die Kommandos `d` und `p` zusammen in einer Kommandozeile angeben. Wiederholt man das obige Beispiel mit diesem Kommando, sollte der Bildschirm wie folgt aussehen:

```
p<CR>
p
dp<CR>
Das ist die vierte Zeile.
```

Bewegen innerhalb der Datei

Zum Anzeigen der Zeile nach der aktuellen Zeile ist im Kommandomodus die RETURN-Taste zu drücken. Folgt nach der aktuellen Zeile keine Zeile mehr, gibt ed ein Fragezeichen (?) aus, und die letzte Zeile der Datei bleibt weiterhin die aktuelle Zeile. Zum Anzeigen der Zeile vor der aktuellen Zeile ist die Minustaste (-) zu drücken.

Der folgende Bildschirm enthält Beispiele für die Wirkung dieser Kommandos:

```
p<CR>
Das ist die vierte Zeile.
-<CR>
und das die dritte.
-<CR>
Das ist die zweite Zeile
-<CR>
Das ist die erste Textzeile.
<CR>
Das ist die zweite Zeile
<CR>
und das die dritte.
```

Man kann also durch Eingabe von -<CR> oder <CR> eine Textzeile anzeigen lassen, ohne das Kommando p eingeben zu müssen. Diese Kommandos beziehen sich also auf Zeilenadressen. Gibt man eine Zeilenadresse ohne ein Kommando ein, geht das Programm ed davon aus, daß die angegebene Zeile angezeigt werden soll. Nun sollte man mit diesen Kommandos experimentieren: Text erstellen, eine Zeile löschen und die Datei anzeigen lassen.

Abspeichern des Pufferinhalts in einer Datei

Wie bereits oben beschrieben hält das System den Text bei einer Editiersitzung in einem temporären Speicher, der Puffer genannt wird. Nach der Bearbeitung des Textes kann man diesen speichern, indem man ihn aus dem temporären Puffer in eine Datei im Rechnerspeicher abspeichert. Dazu wird einfach eine Kopie des Pufferinhalts in die Datei geschrieben. Der Text des Puffers bleibt dabei erhalten, so daß man darin weitere Änderungen vornehmen kann.

NOTE

Man sollte den Puffertext in kurzen Abständen in die Datei abspeichern, da man durch eine Unterbrechung (wie zum Beispiel Stromausfall am Terminal) den Inhalt des Puffers verliert, nicht jedoch die in der Datei abgespeicherte Kopie.

Ein Text wird mit dem Kommando `w` in eine Datei geschrieben (gespeichert). Man braucht dazu keinen Dateinamen anzugeben, sondern gibt einfach `w` ein und drückt die RETURN-Taste. Hat man gerade einen neuen Text erstellt, legt `ed` für diesen eine Datei mit dem Namen an, der beim Aufrufen des Editors angegeben wurde. Hat man eine bestehende Datei bearbeitet, speichert man mit dem Kommando `w` den Pufferinhalt standardmäßig in derselben Datei.

Man kann jedoch in der Kommandozeile auch einen neuen Namen für die Datei als Argument zum Kommando `w` angeben. Dabei ist darauf zu achten, daß man nicht den Namen einer bereits bestehenden Datei angibt, da eine solche dann mit dem Pufferinhalt überschrieben wird. `ed` gibt keinen Warnhinweis aus, wenn eine Datei bereits besteht, sondern überschreibt dann einfach diese Datei mit dem Inhalt des Puffers.

Soll die Datei `versuch` beispielsweise unter dem Namen `allerlei` abgespeichert werden, kann man sie wie folgt umbenennen:

```
$ ed versuch <CR>
? versuch
a <CR>
Das ist die erste Textzeile.<CR>
Das ist die zweite Zeile.<CR>
und das die dritte.<CR>
.
w allerlei <CR>
71
```

Hier ist die letzte Zeile des Bildschirms zu beachten: sie enthält die Anzahl der Zeichen im Text. Zeigt der Editor die Anzahl der Zeichen in dieser Form an, wurde das Kommando zum Speichern korrekt ausgeführt.

Verlassen des Editors

Nachdem man mit der Bearbeitung des Textes fertig ist, speichert man sie mit dem Kommando `w` vom Puffer in eine dauerhafte Datei. Dann verläßt man den Editor und kehrt zur Shell zurück, indem man `q` (für "quit" - verlassen) eingibt.

```
w<CR>
71
q<CR>
$
```

Das System reagiert darauf mit dem Bereit-Zeichen der Shell. Zu diesem Zeitpunkt wird der Editierpuffer gelöscht. Wurde das Kommando zum Speichern (`w`) nicht ausgeführt, ist der Text des Puffers ebenfalls verloren. Hat man während der Editiersitzung im Text keine Änderungen vorgenommen, geht natürlich nichts verloren. Hat man den Text jedoch geändert, dann war die Arbeit der Editiersitzung vergeblich. Gibt man daher nach dem Ändern der Datei `q` ein, ohne daß man den Text in die Datei geschrieben hat, gibt `ed` eine Warnmeldung in der Form eines `?` aus. Dann kann man den Text abspeichern und den Editor verlassen.


```
q<CR>
?
w<CR>
?!
q<CR>
$
```

Gibt man anstelle einer Eingabe nochmals `q` ein, nimmt `ed` an, daß der Inhalt des Puffers nicht in die Datei geschrieben werden soll, und kehrt zur Shell zurück. Die ursprüngliche Datei bleibt unverändert, und der Inhalt des Puffers wird gelöscht.

Damit sind die grundlegenden Kommandos zum Erstellen und Editieren einer Datei mit dem Zeileneditor `ed` eingeführt. Sie werden in Abbildung 5-1 nochmals zusammengefaßt.



Kommando	Funktion
<i>ed datei</i>	Aufrufen von <i>ed</i> um eine <i>Datei</i> zu bearbeiten
a	Text nach der aktuellen Zeile einfügen
.	Texteingabemodus verlassen und in den Kommandomodus von <i>ed</i> zurückkehren.
p	Text auf dem Terminal ausgeben
d	Text löschen
<CR>	die nächste Zeile des Puffers anzeigen (entspricht Zeilenvorschub)
+	die nächste Zeile des Puffers anzeigen
-	die vorige Zeile des Puffers anzeigen
w	den Inhalt des Puffers in der Datei abspeichern
q	Editor <i>ed</i> verlassen und zur Shell zurückkehren

Abbildung 5-1: Übersicht über die Kommandos des Editors *ed*

Übung 1

Die Lösungen für alle Übungen in diesem Kapitel sind am Ende des Kapitels zu finden. Es handelt sich dabei jedoch nicht unbedingt um die einzig möglichen Lösungen. Alle Methoden, die zum gewünschten Ergebnis führen, sind richtig, auch wenn sie mit der hier angegebenen Lösung nicht übereinstimmen.

- 1-1. Rufen Sie den Editor **ed** mit einer Datei mit dem Namen **schrott** auf. Geben Sie eine Textzeile mit **Hallo, schöne Welt** ein und speichern sie in der Datei ab. Anschließend verlassen Sie den Editor **ed** wieder.

Erstellen Sie dann mit **ed** eine Datei des Namens **allerlei**. Geben Sie eine Textzeile mit drei Wörtern, **Adieu, schöne Welt, ein**, speichern Sie den Text in der Datei ab und verlassen Sie den **ed** wieder.

- 1-2. Rufen Sie **ed** wieder mit der Datei **schrott** auf. Wie sieht nun die Anzeige des Editors aus? Stimmt die Anzahl der Zeichen mit der Anzahl überein, die bei der Ausführung des Kommandos **w** in Übung 1-1 ausgegeben wurde? **sp** Lassen Sie sich den Inhalt der Datei anzeigen. Ist es die Datei **schrott**?

Wie kommt man nun zur Shell zurück? **q** verwenden, ohne die Datei abzuspeichern. Weshalb kann man hier den Editor ohne Abspeichern verlassen?

- 1-3. Rufen Sie **ed** mit der Datei **schrott** auf. Fügen Sie eine Zeile ein:

Werners Pferd sprang durch das Fenster.

Hier wurde keine Zeilenadresse angegeben; wo wurde diese Zeile im Puffer eingefügt? Den Inhalt des Puffers anzeigen lassen. Verlassen Sie den Puffer, ohne den Text in der Datei abzuspeichern. Speichern Sie den Puffer in eine andere Datei mit dem Namen **allerlei** ab. Achten Sie dabei darauf, ob **ed** eine Meldung ausgibt, daß eine Datei namens **allerlei** bereits existiert. Der Inhalt von **allerlei** wurde gelöscht und durch den neuen Text ersetzt.

Allgemeines Format der Kommandos des Editors ed

Die Kommandos des Editors **ed** haben ein sehr einfaches und regelmäßiges Format:

[Adresse1[,Adresse2]]Kommando[Argument]<CR>

Die Klammern um *Adresse1*, *Adresse2* und *Argument* bedeuten, daß diese Angaben nicht unbedingt nötig sind. Die Klammern gehören nicht zur Kommandozeile.

Adresse1,Adresse2

Die Adressen geben die Position von Zeilen im Puffer an. Mit *Adresse1* bis *Adresse2* erhält man einen Bereich von Zeilen, auf die das *Kommando* angewandt wird. Wird *Adresse2* nicht angegeben, betrifft das Kommando nur die Zeile, die mit *Adresse1* angegeben wurde.

Kommando Das *Kommando* besteht aus einem Zeichen und gibt an, welche Aufgabe der Editor ausführen soll.

Argument Die *Argumente* zu einem *Kommando* können Teile des Textes sein, die zu ändern sind, oder ein Dateiname oder eine andere Zeilenadresse.

Dieses Format wird anschaulicher werden, wenn man mit den Kommandos des Editors **ed** experimentiert.

Adressierung von Zeilen

Eine Zeilenadresse besteht aus einem Zeichen oder einer Reihe von Zeichen, mit denen eine Textzeile identifiziert wird. Der Editor **ed** kann Kommandos zum Einfügen, Löschen, Verschieben oder Ändern von Text erst ausführen, wenn ihm mitgeteilt wird, auf welche Zeile (Adresse) das Kommando angewandt werden soll. Die Zeilenadresse ist vor dem Kommando einzugeben:

`[Adresse1],[Adresse2]Kommando <CR>`

Sowohl *Adresse1* als auch *Adresse2* sind wahlfreie Angaben. *Adresse1* alleine gibt an, daß die Operation mit einer einzelnen Textzeile ausgeführt werden soll. Gibt man *Adresse1* und *Adresse2* an, wird ein Bereich von Zeilen ausgewählt. Wird keine *Adresse* angegeben, geht **ed** davon aus, daß als Zeilenadresse die aktuelle Zeile gelten soll.

Folgende Methoden werden am häufigsten zur Angabe einer Zeilenadresse in **ed** verwendet:

- Eingabe der Zeilennummern (dabei sind die Zeilen der Dateien kontinuierlich von 1 bis *n* durchnummeriert, und die Numerierung beginnt mit der ersten Zeile der Datei).
- Eingabe von speziellen Symbolen für die aktuelle Zeile, die letzte Zeile, den Bereich von Zeilen
- Addieren oder Subtrahieren von Zeilen zur bzw. von der aktuellen Zeile
- Suchen nach einer Zeichenkette oder einem Wort in der gewünschten Zeile.

Dabei kann man eine einzelne Zeile oder einen Zeilenbereich adressieren oder eine globale Suche nach allen Zeilen ausführen, die eine angegebene Zeichenkette enthalten (eine Zeichenkette ist eine Reihe aufeinanderfolgender Zeichen, wie zum Beispiel ein Wort).

Numerische Adressen

ed ordnet jeder Zeile im Puffer eine numerische Adresse zu. Die erste Zeile im Puffer ist Zeile 1, die zweite hat die Nummer 2 usw. Auf jede Zeile kann der Editor **ed** mit Hilfe der entsprechenden Zeilennummer zugreifen. Zur Veranschaulichung der Zeilenadressierung mit Nummern rufen Sie **ed** mit der Datei **versuch** auf und geben Sie eine Nummer ein.

```
$ ed versuch<CR>
96
1<CR>
Das ist die erste Textzeile.
3<CR>
und das die dritte.
```

Es ist zu beachten, daß **p** als Standardkommando auf eine Zeilenadresse angewendet wird, wenn sie ohne Kommando angegeben wird. Da eine Zeilenadresse eingegeben wurde, nimmt **ed** an, daß die betreffende Zeile auf dem Terminal ausgegeben werden soll.

Im Verlauf einer Editiersitzung ändern sich die Zeilennummern häufig. Weiter unten in diesem Kapitel wird beschrieben, wie man Zeilen einfügt, löscht oder an eine andere Stelle verschiebt. Damit ändern sich die Zeilenadressen einiger Zeilen. Entscheidend ist jeweils die aktuelle Position der betreffenden Zeile im Editierpuffer. Fügt man beispielsweise zwischen Zeile 5 und 6 fünf Textzeilen ein, wird Zeile 6 zu Zeile 11, löscht man Zeile 5, wird Zeile 6 zu Zeile 5.

Symbolische Adressen

Symbolische Adresse der aktuellen Zeile

Die aktuelle Zeile ist die Zeile, auf die am häufigsten Kommandos des Editors `ed` angewandt werden. Direkt nach dem Aufrufen von `ed` mit einer bestehenden Datei ist die letzte Zeile dieser Datei stets die aktuelle Zeile. Das Symbol für die Adresse der aktuellen Zeile ist ein Punkt. Daher kann man die aktuelle Zeile anzeigen lassen, indem man einfach einen Punkt (.) eingibt und anschließend die RETURN-Taste drückt.

Probieren Sie dieses Kommando in der Datei `versuch` aus:

```
$ ed versuch <CR>
96
.<CR>
Das ist die vierte Zeile.
```

Der `.` stellt dabei die Adresse dar. Da nach dem Punkt kein Kommando angegeben wurde, führt `ed` das Standardkommando `p` mit dieser Zeile aus, d. h. die Zeile an dieser Adresse wird angezeigt.

Um die Zeilennummer der aktuellen Zeile zu erfahren, gibt man folgendes Kommando ein:

```
.= <CR>
```

`ed` gibt daraufhin die Zeilennummer aus. In der Datei `versuch` ist die aktuelle Zeile beispielsweise Zeile 4.

```
.<CR>
Das ist die vierte Zeile.
.= <CR>
4
```

Symbolische Adresse der letzten Zeile

Die symbolische Adresse für die letzte Zeile einer Datei ist das Dollarzeichen (\$). Nun überprüfen, ob mit dem Zeichen \$ auf die letzte Zeile einer Datei zugegriffen wird; rufen Sie dazu die Datei **versuch** mit **ed** auf und geben Sie diese Adresse alleine in einer eigenen Zeile an (beim Aufrufen einer Datei ist stets die letzte Zeile die aktuelle Zeile).

```
$ ed versuch <CR>
96
.<CR>
Das ist die vierte Zeile.
$<CR>
Das ist die vierte Zeile.
```

Es ist darauf hinzuweisen, daß das Dollarzeichen (\$) als Adresse innerhalb des Editors **ed** eine andere Funktion hat als das Bereit-Zeichen \$ der Shell.

Symbolische Adresse: Alle Zeilen

Wird ein Komma (,) als Adresse verwendet, bezieht es sich auf alle Zeilen einer Datei - von der ersten bis zur letzten. Dies ist eine Kurzform der bereits oben erwähnten Zeichenkette, die für alle Zeilen einer Datei steht: 1,\$. Mit diesem Kürzel den Inhalt von **versuch** anzeigen lassen:

```
p<CR>
Das ist die erste Textzeile.
Das ist die zweite Zeile
und das die dritte.
Das ist die vierte Zeile.
```

5

Symbolische Adresse: Aktuelle Zeile bis letzte Zeile

Der Strichpunkt (;) steht für eine Reihe von Zeilen, die mit der aktuellen Zeile beginnt und mit der letzten Zeile einer Datei endet. Er entspricht der symbolischen Adresse „,\$. Dies soll nun mit der Datei **versuch** ausprobiert werden:

```
2p<CR>
Das ist die zweite Zeile
;p<CR>
Das ist die zweite Zeile
und das die dritte.
Das ist die vierte Zeile.
```

Relative Adressen: Addieren bzw. Subtrahieren von Zeilen von der aktuellen Zeile

Es ist häufig erforderlich, Zeilen mit Bezug auf die aktuelle Zeile zu adressieren. Dies erfolgt durch Addieren bzw. Subtrahieren einer Anzahl von Zeilen zu bzw. von der Zeilennummer der aktuellen Zeile mit Hilfe eines Pluszeichens (+) bzw. eines Minuszeichens (-). Solche Adressen werden relative Adressen genannt. Zum Experimentieren mit relativen Zeilenadressen sind nun einige weitere Zeilen in die Datei **versuch** zu schreiben, wie auf dem folgenden Bildschirm dargestellt. Speichern Sie den Inhalt des Puffers auch in der Datei ab, damit die Einfügungen gesichert sind:

```
$ ed versuch <CR>
96
.<CR>
Das ist die vierte Zeile.
a<CR>
fünf<CR>
sechs<CR>
sieben<CR>
acht<CR>
neun<CR>
zehn<CR>
.<CR>
w<CR>
123
```

Zeilennummern von der Zeilennummer der aktuellen Zeile subtrahieren bzw. addieren:

```
4<CR>
Das ist die vierte Zeile.
+3<CR>
sieben
-5<CR>
Das ist die zweite Zeile
```

Was geschieht beispielsweise, wenn man eine Zeilenadresse anfordert, deren Nummer größer als die der letzten Zeile ist, oder wenn man versucht, eine Zeilennummer abzuziehen, die größer als die der aktuellen Zeile ist?

```
5<CR>
fünf
-6<CR>
?
.= <CR>
5
+7<CR>
?
```

Die aktuelle Zeile bleibt weiterhin Zeile 5 im Puffer. Die aktuelle Zeile ändert sich nur, wenn man dem Programm `ed` eine korrekte Adresse angibt. Die Anzeige `?` bedeutet, daß ein Fehler vorliegt. Unter "Weitere nützliche Kommandos und Hinweise" am Ende dieses Kapitels wird erläutert, wie man eine Hilfeanfrage aufrufen kann, in der der Fehler beschrieben wird.

Adressierung mit Zeichenketten

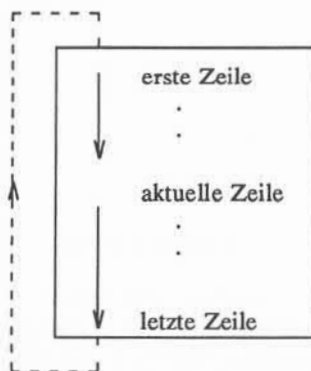
Man kann in einer Datei vorwärts oder rückwärts nach einer Zeile suchen, die eine bestimmte Zeichenkette enthält. Dazu ist eine Zeichenkette anzugeben, der ein Begrenzungszeichen vorangestellt wird.

Begrenzungszeichen kennzeichnen die Grenzen von Zeichenketten; durch sie wird dem Editor **ed** mitgeteilt, wo eine Zeichenkette beginnt und endet. Das üblichste Begrenzungszeichen ist der Schrägstrich (/); er wird folgendermaßen eingegeben:

/Muster

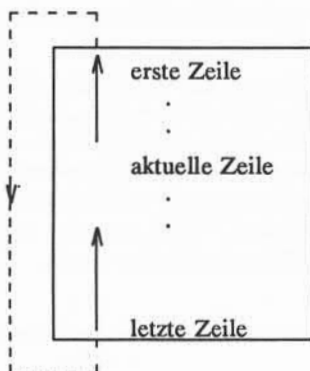
Gibt man ein Muster mit vorangestelltem Schrägstrich (/) an, beginnt **ed** mit der Suche in der aktuellen Zeile und durchsucht die Datei vorwärts (in allen folgenden Zeilen im Puffer) nach der nächsten Zeile, die das *Muster* enthält. Ist der Editor mit der Suche in der letzten Zeile des Puffers angelangt, springt er an den Anfang der Datei und sucht ab Zeile 1 weiter.

Das folgende Rechteck stellt den Editierpuffer dar. Die Pfeile kennzeichnen den Verlauf der Suche, die mit einem Schrägstrich (/) eingeleitet wurde:



Ein weiteres nützliches Begrenzungszeichen ist das Fragezeichen (?). Gibt man ein Suchmuster mit vorangestelltem ? ein (*?Muster*), beginnt **ed** mit der Suche in der aktuellen Zeile und sucht in den vorhergehenden Zeilen im Puffer rückwärts nach der nächsten Zeile, die das *Muster* enthält. Ist der Editor mit der Suche in der ersten Zeile der Datei angelangt, springt er in die letzte Zeile der Datei und sucht rückwärts ab dieser weiter.

Das folgende Rechteck stellt den Editierpuffer dar. Die Pfeile zeigen den Verlauf der Suche, die mit einem Fragezeichen (?) eingeleitet wurde:



Experimentieren Sie nun mit diesen beiden Methoden, d. h. suchen Sie mit Zeichenmustern Zeilenadressen in der Datei **versuch**. Was geschieht beispielsweise, wenn **ed** die angegebene Zeichenkette nicht findet?

```
$ ed versuch <CR>
123
.<CR>
zehn
?erste<CR>
Das ist die erste Textzeile.
/vierte<CR>
Das ist die vierte Zeile.
/schrott<CR>
?
```

In diesem Beispiel hat **ed** die Zeichenketten **erste** und **vierte** gefunden. Da kein Kommando für die Adresse angegeben wurde, wurde standardmäßig das Kommando **p** ausgeführt, d. h. die gefundenen Zeilen wurden angezeigt. Wenn **ed** eine Zeichenkette nicht in der Datei findet (wie zum Beispiel **schrott**), wird ein Fragezeichen (?) ausgegeben.

Mit dem Schrägstrich (/) kann man auch nach mehrmaligem Auftreten eines Zeichenmusters suchen, ohne daß man es mehrmals eingeben muß. Dazu zuerst das Muster durch Eingabe von **/Muster** wie gewohnt eingeben. Nachdem **ed** die Zeile anzeigt, in der das Muster zum ersten Mal auftritt, wartet das Programm auf ein Kommando. Geben Sie nun **/** ein und drücken Sie die RETURN-Taste; dann sucht **ed** weiter vorwärts in der Datei nach dem zuletzt angegebenen **Muster**. Suchen Sie nach dem Wort **die** in der Datei **versuch**:

```
.<CR>
Das ist die erste Textzeile.
/die<CR>
Das ist die zweite Zeile
/<CR>
und das die dritte.
/<CR>
Das ist die vierte Zeile.
/<CR>
Das ist die erste Textzeile.
```

Es ist zu beachten, daß **ed** zum Anfang der Datei zurückspringt und die Suche dort weiterführt, nachdem er jedes Auftreten von **Muster** zwischen der Zeile, an der das Kommando eingegeben wurde, und der letzten Zeile der Datei gemeldet hat.

Angabe eines Zeilenbereiches

Mehrere Zeilen können auf zwei Formen ausgewählt werden. Man kann dazu einen Zeilenbereich, wie zum Beispiel *Adresse1* bis *Adresse2*, oder eine globale Suche nach allen Zeilen angeben, die ein bestimmtes Zeichenmuster enthalten.

Die einfachste Möglichkeit, einen Bereich von Zeilen anzugeben, besteht darin, die Nummern der ersten und letzten Zeile des Bereiches, durch Komma getrennt, anzugeben. Diese Adressenangabe ist vor dem Kommando einzugeben. Sollen beispielsweise Zeilen 2 bis 7 des Editierpuffers angezeigt werden, geben Sie für *Adresse1* die Nummer 2 und für *Adresse2* die Nummer 7 in folgendem Format an:

```
2,7p<CR>
```

Probieren Sie dies mit der Datei **versuch** aus:

```
2,7p<CR>
Das ist die zweite Zeile
und das die dritte.
Das ist die vierte Zeile.
fünf
sechs
sieben
```

5

Wenn man versucht, **2,7** ohne das Kommando **p** einzugeben, gibt **ed** nur *Adresse2* aus, die letzte Adresse des Zeilenbereiches.

Relative Zeilenadressen können auch verwendet werden, um einen Bereich von Zeilen anzugeben. Dazu muß *Adresse1* vor *Adresse2* im Puffer stehen. Relative Adressen werden ausgehend von der aktuellen Zeile berechnet; dies wird im folgenden Beispiel deutlich:

```
4<CR>
Das ist die vierte Zeile.
-2,+3p<CR>
Das ist die zweite Zeile
und das die dritte.
Das ist die vierte Zeile.
fünf
sechs
sieben
```

Globale Suche

Es gibt zwei Kommandos, deren Format sich von dem der sonstigen **ed**-Kommandos unterscheidet: **g** und **v**. Dies sind Kommandos für globales Suchen, bei denen Adressen mit einer Zeichenkette (*Muster*) angegeben werden. Das Kommando **g** sucht nach allen Zeilen, die die Zeichenkette *Muster* enthalten und führt das angegebene *Kommando* mit diesen Zeilen aus. Das Kommando **v** bewirkt eine Suche nach allen Zeilen, die das *Muster* nicht enthalten, und führt das angegebene *Kommando* mit diesen Zeilen aus.

Das allgemeine Format für diese beiden Kommandos lautet:

```
g/Muster/Kommando <CR>
v/Muster/Kommando <CR>
```

Probieren Sie diese Kommandos aus; suchen Sie dazu nach dem Wort **die** in der Datei **versuch**:


```
g/die/p<CR>
Das ist die erste Textzeile.
Das ist die zweite Zeile
und das die dritte.
Das ist die vierte Zeile.
```

```
v/die/p<CR>
fünf
sechs
sieben
acht
neun
zehn
```

Es sei nochmals auf die Funktion des Kommandos `v` hingewiesen: es sucht alle Zeilen, die das in der Kommandozeile angegebene Wort (`die`) nicht enthalten.

Wie oben bereits erwähnt, gilt als Standardkommando für die mit `g` oder `v` adressierten Zeilen das Kommando `p`; man braucht daher `p` in der Kommandozeile nicht anzugeben, wenn man die Zeilen anzeigen lassen will.



```
g/die<CR>
Das ist die erste Textzeile.
Das ist die zweite Zeile
und das die dritte.
Das ist die vierte Zeile.
```

Werden die Zeilenadressen jedoch angegeben, um auf diese Zeilen ein anderes **ed**-Kommando anzuwenden, müssen Begrenzungszeichen vor und hinter dem Suchmuster angegeben werden. Zeilenadressen können für die Kommandos des Editors **ed** mit allen hier beschriebenen Methoden angegeben werden. In Abbildung 5-2 werden die Symbole und die Kommandos zusammengefaßt, mit denen Zeilen adressiert werden können.

Adresse	Beschreibung
<i>n...</i>	Nummer einer Zeile im Puffer.
<i>.</i>	Die aktuelle Zeile (die Zeile, auf die zuletzt ein Kommando des Editors ed angewandt wurde).
<i>.=</i>	Dieses Kommando wird zum Abfragen der Zeilennummer der aktuellen Zeile verwendet.
<i>\$</i>	Die letzte Zeile der Datei.
<i>,</i>	Alle Zeilen von Zeile 1 bis zur letzten Zeile der Datei.
<i>;</i>	Die Zeilengruppe von der aktuellen Zeile bis zur letzten Zeile.
<i>+ n</i>	Die Zeile, die sich <i>n</i> Zeilen nach der aktuellen Zeile befindet.
<i>- n</i>	Die Zeile, die sich <i>n</i> Zeilen vor der aktuellen Zeile befindet.
<i>/abc</i>	Das Kommando, mit dem im Puffer vorwärts nach der ersten Zeile gesucht wird, die das Muster <i>abc</i> enthält.
<i>?abc</i>	Das Kommando, mit dem im Puffer rückwärts nach der ersten Zeile gesucht wird, die das Muster <i>abc</i> enthält.
<i>g/abc</i>	Alle Zeilen, die das Muster <i>abc</i> enthalten.
<i>v/abc</i>	Alle Zeilen, die das Muster <i>abc</i> nicht enthalten.

Abbildung 5-2: Übersicht über die Zeilenadressierung

Übung 2

- 2-1. Erstellen Sie eine Datei mit dem Namen `orte` mit den folgenden Zeilen:

```
Mir liegt ein Ort wie  
Hannover  
Als wäre man nirgendwo in  
Frankfurt  
Ein Ort zum Wohlfühlen ist  
Wiesbaden  
Ich hab mein Herz verloren in  
Heidelberg  
Ich habe $$ verloren in  
München
```

- 2-2. Lassen Sie sich Zeile 3 anzeigen.
- 2-3. Welche Zeilen werden angezeigt, wenn man einen Zeilenbereich mit der relativen Adresse `-2,+3p` angibt?
- 2-4. Wie lautet die Nummer der aktuellen Zeile? Lassen Sie sich die aktuelle Zeile anzeigen.
- 2-5. Was steht in der letzten Zeile?
- 2-6. Welche Zeile wird angezeigt, wenn man die folgende Suchzeile eingibt?

```
?Ort<CR>
```

Geben Sie nach der Antwort von `ed` das folgende Kommando allein in eine Zeile ein:

```
?<CR>
```

Zu welchem Ergebnis führt dies?

- 2-7. Suchen Sie nach allen Zeilen, die das Muster "in" enthalten. Suchen Sie dann nach allen Zeilen, die das Muster "in" nicht enthalten.

Text einer Datei anzeigen lassen

Im Editor `ed` gibt es zwei Kommandos zum Anzeigen von Textzeilen im Editierpuffer: `p` und `n`.

Text alleine anzeigen: das Kommando `p`

Das Kommando `p` wurde schon in mehreren Beispielen angewandt. Daher ist sein allgemeines Format wohl bekannt:

```
[Adresse1,Adresse2]p<CR>
```

Mit `p` können keine Argumente eingegeben werden. Es kann jedoch mit einem Ersetzungskommando kombiniert werden. Dies wird weiter unten in diesem Kapitel behandelt.

Experimentieren Sie nun mit einer Datei im Home-Verzeichnis und den Zeilenadressen in Abbildung 5-3. Beispielsweise kann man versuchen, das Kommando `p` auf jede Adresse anzuwenden, um zu sehen, ob der Editor `ed` so reagiert, wie es in der Abbildung dargestellt ist.

Adresse	Reaktion
1,\$p<CR>	ed sollte die ganze Datei auf dem Terminal ausgeben.
-5p<CR>	ed sollte von der aktuellen Zeile aus fünf Zeilen zurückgehen und die dort stehende Zeile anzeigen.
+2p<CR>	ed sollte von der aktuellen Zeile aus zwei Zeilen vorwärtsgehen und die dort stehende Zeile anzeigen.
1,/x/p<CR>	ed sollte die Gruppe von Zeilen anzeigen, die von Zeile 1 bis zur ersten Zeile nach der aktuellen Zeile reicht, in der das Zeichen <i>x</i> enthalten ist. Dabei ist es wichtig, den Buchstaben <i>x</i> zwischen Schrägstriche zu setzen, damit ed zwischen der Suchmusteradresse (<i>x</i>) und dem Editorkommando (<i>p</i>) unterscheiden kann.

Abbildung 5-3: Beispiele für die Adressierung von Text zum Anzeigen

Text mit Zeilenadressen anzeigen: das Kommando **n**

Mit dem Kommando **n** werden Textzeilen jeweils mit vorangestellter Zeilenadresse angezeigt. Es ist nützlich, wenn man Zeilen löscht, einfügt oder ändert. Das allgemeine Format für die Kommandozeile ist für **n** dasselbe wie für **p**:

[*Adresse1,Adresse2*]**n**<CR>

Ebenso wie **p** wird **n** ohne weitere Argumente eingegeben, das Kommando kann jedoch mit dem Ersetzungskommando kombiniert werden.

Probieren Sie das Kommando **n** in der Datei **versuch** aus:

```
$ ed versuch <CR>
123
1,5n <CR>
1      Das ist die erste Textzeile.
2      Das ist die zweite Zeile
3      und das die dritte.
4      Das ist die vierte Zeile.
5      fünf
6      sechs
7      sieben
8      acht
9      neun
10     zehn
```

5

In Abbildung 5-3 werden die Kommandos des Editors **ed** zum Anzeigen von Text zusammengefaßt.

Kommando	Funktion
p	Anzeigen bestimmter Textzeilen des Editierpuffers auf dem Terminal
n	Anzeigen bestimmter Textzeilen des Editierpuffers mit ihren numerischen Zeilenadressen auf dem Terminal

Abbildung 5-4: Übersicht über die Kommandos zum Anzeigen von Text

Text erstellen

Im Editor `ed` gibt es drei Grundkommandos zum Erstellen neuer Textzeilen:

- a Text anhängen
- i Text einfügen
- c Text ändern

Text anhängen: das Kommando `a`

Mit dem Kommando zum Anhängen von Text, `a`, kann man Zeilen nach der aktuellen Zeile oder einer bestimmten Adresse der Datei einfügen. Dieses Kommando wurde bereits im Abschnitt "Einführung" in diesem Kapitel verwendet. Das allgemeine Format des Kommandos lautet:

```
[Adresse]a<CR>
```

Die Angabe der Adresse ist wahlfrei. Als Standardwert für *Adresse* gilt die aktuelle Zeile.

In einigen der Beispiele wurde dieses Kommando mit der Standardadresse verwendet. Geben Sie nun verschiedene Zeilennummern als *Adresse* an. Im folgenden Beispiel wird eine neue Datei mit dem Namen `neu-datei` erstellt. In der ersten Kommandozeile zum Anhängen von Text wird die aktuelle Zeile, d. h. der Standardwert, als Adresse verwendet. In der zweiten Kommandozeile zum Anhängen von Text wird Zeile 1 als *Adresse* angegeben. Die Zeilen können mit dem Kommando `n` angezeigt werden, so daß ihre Zeilenadressen sichtbar sind. Es ist zu beachten, daß der Modus "Anhängen" durch Eingabe eines Punktes (`.`), alleine in eine Zeile, beendet wird.


```

$ ed neu-datei <CR>
?neu-datei
a <CR> Einige Zeilen Text in diese Datei eingeben.
.<CR> 1,$n <CR>
1      Einige Zeilen
2      Text in diese
3      Datei eingeben.
1a <CR>
Dies wird Zeile 2 <CR>
Dies wird Zeile 3 <CR>
.<CR> 1,$n <CR>
1      Einige Zeilen
2      Dies wird Zeile 2
3      Dies wird Zeile 3
4      Text in diese
5      Datei eingeben.

```

Nach dem Anhängen der beiden neuen Zeilen wird die ursprüngliche Zeile 2 (Text in diese) zur Zeile Nummer 4.

Man kann die Stellen in der Datei, an denen Text angehängt werden soll, in Kurzform durch eine Kombination des Kommandos a und symbolischen Adressen angeben. Mit den folgenden drei Kommandos kann man sich so schnell in der Datei bewegen und schnell Text einfügen.

- .a <CR>** Text hinter der aktuellen Zeile anhängen
- \$a <CR>** Text hinter der letzten Zeile der Datei anhängen
- 0a <CR>** Text vor der ersten Zeile der Datei einfügen (an der symbolischen Adresse "Zeile 0")

Um diese Adressen auszuprobieren können Sie nun eine einzeilige Datei mit dem Namen **zeilen** erstellen und die Beispielkommandos auf den folgenden Bildschirmen eingeben. Die Beispiele erscheinen nur aus Gründen der Übersichtlichkeit in verschiedenen Bildschirmen; die Datei **zeilen** muß nicht dreimal aufgerufen werden, um die einzelnen symbolischen Adressen auszuprobieren. Man kann die Datei **zeilen** einmal aufrufen und versuchen, die drei Adressen nacheinander einzugeben.

\$ ed zeilen <CR>
?zeilen
a <CR>
Dies ist die aktuelle Zeile. <CR>
. <CR>
p <CR>
Dies ist die aktuelle Zeile.
.a <CR>
Dies ist die Zeile nach der aktuellen Zeile. <CR>
. <CR>
-l,p <CR>
Dies ist die aktuelle Zeile.
Dies ist die Zeile nach der aktuellen Zeile.

\$a <CR>
Dies ist jetzt die letzte Zeile. <CR>
. <CR>
\$ <CR>
Dies ist jetzt die letzte Zeile.

```

0a<CR>
Dies ist jetzt die erste Zeile.<CR>
Dies ist jetzt die zweite Zeile.<CR>
Die Zellennummern ändern sich,<CR>
wenn Zeilen hinzugefügt werden.<CR>
.<CR> 1,4n<CR>
1   Dies ist jetzt die erste Zeile.
2   Dies ist jetzt die zweite Zeile.
3   Die Zeilennummern ändern sich,
4   wenn Zeilen hinzugefügt werden.

```

Da das Kommando zum Anhängen von Text diesen nach einer bestimmten Adresse einfügt, wird im obigen Beispiel die Zeile vor Zeile 1 als die Zeile nach Zeile 0 angesprochen. Um solche umständlichen Adressierungen zu vermeiden, kann man ein anderes Kommando des Editors verwenden: das Einfügekommmando **i**.

Text einfügen: das Kommando **i**

Mit dem Einfügekommmando (**i**) kann Text vor einer bestimmten Zeile des Editierpuffers eingefügt werden. Das allgemeine Format der Kommandozeile für **i** entspricht dem des Kommandos **a**.

```
[Adresse1]i<CR>
```

Ebenso wie beim Kommando zum Anhängen von Text können eine oder mehrere Textzeilen eingefügt werden. Der Eingabemodus wird durch Eingabe eines Punktes (.) alleine in eine Zeile beendet.

Erstellen Sie eine Datei mit dem Namen **einfügen**, um das Einfügekommmando (**i**) auszuprobieren:

```
$ ed einfügen <CR>
?einfügen
a <CR>
Zeile 1 <CR>
Zeile 2 <CR>
Zeile 3 <CR>
Zeile 4 <CR>
.<CR> w <CR>
28
```

Fügen Sie nun vor Zeile 2 eine Textzeile ein und eine weitere vor Zeile 1. Lassen Sie sich mit dem Kommando **n** alle Zeilen des Editierpuffers anzeigen:

```
2i <CR>
Dies ist die neue Zeile 2. <CR>
.<CR>
1,$n <CR>
1 Zeile 1
2 Dies ist die neue Zeile 2.
3 Zeile 2
4 Zeile 3
5 Zeile 4
Ii <CR>
Dies ist der Textanfang. <CR>
.<CR>
1,$n <CR>
1 Dies ist der Textanfang.
2 Zeile 1
3 Dies ist die neue Zeile 2.
4 Zeile 2
5 Zeile 3
6 Zeile 4
```

Man sollte nun mit dem Einfügekommando experimentieren, indem man ihn mit symbolischen Zeilenadressen wie folgt kombiniert:

- `.i<CR>`
- `$i<CR>`

Text ändern: das Kommando `c`

Mit dem Kommando zum Ändern von Text (`c`) werden alle angegebenen Zeilen gelöscht, und man kann dann einzelne oder mehrere Textzeilen an deren Stelle eingeben. Da mit `c` ein Zeilenbereich gelöscht werden kann, enthält das allgemeine Format für dieses Kommando zwei Adressen:

`[Adresse1,Adresse2]c<CR>`

Mit dem Änderungskommando kommt der Benutzer in den Texteingabemodus; um diesen zu verlassen, ist ein Punkt alleine in eine Zeile einzugeben.

Adresse1 ist die erste Zeile, *Adresse2* die letzte Zeile des Zeilenbereiches, der durch neuen Text ersetzt werden soll. Soll nur eine Textzeile ersetzt werden, ist nur eine *Adresse1* anzugeben. Wird keine Adresse angegeben, geht der Editor `ed` davon aus, daß die aktuelle Zeile geändert werden soll.

Erstellen Sie nun eine Datei mit dem Namen `ändern`, um dieses Kommando auszuprobieren. Nach der Eingabe des Textes auf dem Bildschirm ändern Sie die Zeilen eins bis vier durch die Eingabe `1,4c`:

```
1,$n<CR>
1 Zeile 1
2 Zeile 2
3 Zeile 3
4 Zeile 4
5 Zeile 5
1,4c<CR>
Zeile 1 und<CR>
Zeilen 2 bis 4 ändern<CR>
.<CR>
1,$n<CR>
1 Zeile 1 und
2 Zeilen 2 bis 4 ändern
3 Zeile 5
```

Änden Sie nun mit dem Kommando c nun die aktuelle Zeile:

```
.<CR>
Zeile 5
c<CR> Dies ist die neue Zeile 5.
.<CR>
.<CR>
Dies ist die neue Zeile 5.
```

Wenn man sich nicht sicher ist, ob man den Texteingabemodus bereits verlassen hat, kann man einen (weiteren) Punkt eingeben. Wird die aktuelle Zeile angezeigt, weiß man, daß man sich im Kommandomodus des Editors ed befindet.

In Abbildung 5-5 werden die Kommandos des Editors `ed` zum Erstellen von Text zusammengefaßt.

Kommando	Funktion
a	Text nach der angegebenen Zeile im Puffer anhängen
i	Text vor der angegebenen Zeile im Puffer einfügen
c	Den Text der angegebenen Zeile(n) in neuen Text ändern
.	Den Texteingabemodus verlassen und in den Kommando- modus des Editors <code>ed</code> zurückkehren

Abbildung 5-5: Übersicht über die Kommandos zum Erstellen von Text

Übung 3

- 3-1. Legen Sie eine neue Datei mit dem Namen **üb3** an. Anstelle des Kommandos "a" zum Anhängen von Text verwenden Sie das Einfügekommmando "i", um neuen Text in den Puffer zu schreiben. Was geschieht?
- 3-2. Rufen Sie den Editor **ed** mit der Datei **orte** auf. Welches ist die aktuelle Zeile?

Fügen Sie vor der dritten Zeile ein:

Stuttgart<CR>

Fügen Sie vor der aktuellen Zeile folgendes ein:

oder<CR>

Dortmund<CR>

Dann fügen Sie vor der letzten Zeile

Hotels in<CR>

ein. Lassen Sie sich den Text des Puffers mit Zeilennummern anzeigen.

- 3-3. Lassen Sie sich in der Datei **orte** die Zeilen 1 bis 5 anzeigen und ersetzen Sie die Zeilen 2 bis 5 durch folgendes:

Hamburg<CR>

Lassen Sie sich die Zeilen 1 bis 3 anzeigen.

- 3-4. Welches ist die aktuelle Zeile nach der Übung 3-3?

Suchen Sie die Textzeile, die folgendes Wort enthält:

Frankfurt

Ersetzen Sie das Wort

Frankfurt

durch

Passau.

Lassen Sie sich die aktuelle Zeile anzeigen.

3-5. Suchen Sie mit einer Kommandozeile

Wiesbaden

und ersetzen es durch:

Berlin



Text löschen

In diesem Abschnitt werden zwei Typen von Kommandos zum Löschen von Text im Editor **ed** beschrieben. Der eine Typ wird verwendet, während man im Kommandomodus arbeitet: mit **d** wird eine Zeile gelöscht, mit **u** wird das zuletzt ausgeführte Kommando rückgängig gemacht. Der andere Kommandotyp wird im Texteingabemodus verwendet: mit **<#>** wird ein Zeichen gelöscht, mit **<@>** wird eine Zeile gelöscht. Als Löschtasten im Eingabemodus werden dieselben Tasten verwendet, die man auch auf der Shell-Ebene nach einem Bereit-Zeichen eingibt. Sie sind ausführlich unter "Korrektur von Tippfehlern" in Kapitel 2 beschrieben.

Löschen von Zeilen: das Kommando **d**

Mit dem Löschkommando **d** wurden bereits im Abschnitt "Einführung" dieses Kapitels Textzeilen gelöscht.

Das allgemeine Format des Kommandos **d** lautet:

`[Adresse1,Adresse2]d<CR>`

Man kann entweder einen Bereich von Zeilen (*Adresse1* bis *Adresse2*) oder einzelne Zeilen (*Adresse1*) löschen. Wird keine Adresse angegeben, löscht **ed** die aktuelle Zeile.

Im folgenden Beispiel werden die Zeilen 1 bis 5 angezeigt und anschließend die Zeilen 2 bis 4 gelöscht:

```

1,$n<CR>
1 1 Pferd
2 2 Hühnchen
3 3 Schinkenrolle
4 4 Senfglas
5 5 Heuballen
2,$d<CR>
1,$n<CR>
1 1 Pferd
2 5 Heuballen

```

Will man nur die letzte Zeile einer Datei löschen, kann man einfach eine symbolische Adresse verwenden:

```
$d<CR>
```

Wie kann man nun die aktuelle Zeile löschen? Einer der verbreitetsten Fehler bei der Verwendung von `ed` besteht darin, daß man vergißt, einen Punkt einzugeben, um den Texteingabemodus zu verlassen. In diesem Fall kann Text versehentlich in den Puffer eingefügt werden. Im folgenden Beispiel wird eine Zeile mit dem Anzeigekommando (`1,$p`) versehentlich in den Text aufgenommen, bevor der Benutzer den Eingabemodus verläßt. Da dies die zuletzt in den Text eingegebene Zeile war, wird sie zur aktuellen Zeile. Sie kann mit der symbolischen Adresse `.` (Punkt) gelöscht werden:

```
a<CR>
Letzte Textzeile.<CR>
l,$p<CR>
.<CR>
p<CR>
l,$p
.d<CR>
p<CR>
Letzte Textzeile.
```

Vor dem Experimentieren mit dem Löschkommando **d** nun zunächst zum Aufhebekommando (letztes Kommando rückgängig machen) **u**.

Zuletzt eingegebenes Kommando rückgängig machen: das Kommando **u**

Mit dem Kommando **u** (Abkürzung für "undo" - rückgängig machen) wird das zuletzt eingegebene Kommando rückgängig gemacht, d. h. Text, der durch dieses Kommando geändert oder gelöscht wurde, wird wiederhergestellt. Das Kommando kann nicht mit Adressen oder Argumenten verwendet werden. Das Format für dieses Kommando lautet:

```
u<CR>
```

Das Kommando **u** ist sehr nützlich, wenn man Text wiederherstellen will, den man versehentlich gelöscht hat. Löscht man alle Zeilen einer Datei und gibt anschließend das Kommando **p** ein, antwortet **ed** mit einem Fragezeichen (?), da sich keine Zeilen mehr in der Datei befinden. Man kann sie mit dem Kommando **u** wiederherstellen.

```

I,$p<CR>
Dies ist die erste Zeile.
Dies ist die mittlere Zeile.
Dies ist die letzte Zeile.
I,$d<CR>
p<CR>
?
u<CR>
p<CR>
Dies ist die letzte Zeile.

```

Experimentieren Sie nun mit dem Kommando **u**: machen Sie das Kommando zum Anhängen von Text (**a**) rückgängig:

```

.<CR>
Dies ist die einzige Textzeile.
a<CR>
Diese Zeile wird angehängt.<CR>
.<CR>
I,$p<CR>
Dies ist die einzige Textzeile
Diese Zeile wird angehängt.
u<CR>
I,$p<CR>
Dies ist die einzige Textzeile.

```

NOTE

u kann nicht eingesetzt werden, um das Kommando zum Speichern (w) oder das Kommando zum Verlassen des Editors (q) rückgängig zu machen; es ist jedoch möglich, mit u ein vorhergehendes Kommando u rückgängig zu machen.

Löschen von Text im Texteingabemodus

Im Texteingabemodus kann man die aktuelle Eingabezeile mit denselben Tasten korrigieren, wie man sie auch verwendet, um eine Shell-Kommandozeile zu korrigieren. In der Standardkonfiguration stehen dafür zwei Tasten zur Verfügung: mit dem Zeichen @ wird die aktuelle Zeile gelöscht, mit dem Zeichen # springt man um ein Zeichen in der aktuellen Zeile zurück, so daß man es überschreiben und damit das ursprünglich an dieser Stelle stehende Zeichen löschen kann (nähere Angaben dazu sind unter "Korrektur von Tippfehlern" in Kapitel 2 zu finden).

Wie bereits in Kapitel 2 erwähnt kann man die Funktionen zum Löschen von Zeilen und Zeichen bei Bedarf auf andere Tasten legen (nähere Angaben dazu sind unter "Anpassen der Benutzerumgebung" in Kapitel 7 zu finden). Wurden diese Funktionen auf andere Tasten gelegt, müssen während der Arbeit mit dem Editor ed auch diese Tasten verwendet werden, da die Standardtasten (@ und #) nicht mehr diese Funktion haben.

Löschfunktion außer Kraft setzen

Es kann vorkommen, daß man die Zeichen @ oder # als Textzeichen verwenden will. Damit diese Zeichen nicht mehr als Löschkommandos interpretiert werden, muß ihnen ein Backslash (\) wie im folgenden Beispiel vorangestellt werden:

```
a<CR>
Die Zeichen \@ und \# sind hier Textzeichen <CR>
.<CR>
p<CR>
Die Zeichen @ und # sind hier Textzeichen
```

In Abbildung 5-6 werden die Kommandos des Editors `ed` und der Shell zum Löschen von Kommandos im Editor `ed` zusammengefaßt.

Kommando	Funktion
Im Kommandomodus:	
<d>	Löschen einzelner oder mehrerer Textzeilen
<u>	Vorheriges Kommando rückgängig machen
<@>	Aktuelle Kommandozeile löschen
Im Texteingabemodus:	
<@>	Aktuelle Zeile löschen
<#> oder <BACKSPACE>	Zuletzt eingegebenes Zeichen löschen

Abbildung 5-6: Übersicht über die Kommandos zum Löschen von Text

Text ersetzen

Ein Text kann auch mit einem Ersetzungskommando geändert werden. Bei diesem Kommando wird eine Zeichenkette an der Stelle ihres ersten Auftretens durch eine neue Zeichenkette ersetzt. Das allgemeine Kommandozeilenformat für dieses Kommando lautet:

```
[Adresse1,Adresse2]s/alt_text/neu_text/[Kommando]<CR>
```

Die einzelnen Bestandteile dieser Kommandozeile werden im folgenden erläutert:

Adresse1,Adresse2

Der Zeilenbereich, der mit dem Kommando *s* adressiert wird. Die Adresse kann aus einer Einzelzeile (*Adresse1*), einem Zeilenbereich (*Adresse1* bis *Adresse2*) oder einer Adresse für globale Suche bestehen. Wird keine Adresse angegeben, führt *ed* die Ersetzung in der aktuellen Zeile aus.

s

Das Ersetzungskommando

/alt_text

Das Argument, das angibt, welcher Text ersetzt werden soll; es wird normalerweise durch Schrägstriche eingegrenzt, jedoch können auch andere Zeichen, wie zum Beispiel ein Fragezeichen (?) oder ein Punkt (.) als Begrenzungszeichen verwendet werden. Das Argument besteht aus den Wörtern oder Zeichen, die durch andere Wörter bzw. Zeichen ersetzt werden sollen. Mit dem Kommando werden diese Wörter bzw. Zeichen an der ersten Stelle ihres Auftretens im Text ersetzt.

/neu_text

Das Argument, das angibt, durch welchen Text der ursprüngliche *alt_text* ersetzt werden soll; es wird durch Schrägstriche oder durch dieselben Begrenzungszeichen eingegrenzt, die auch bei der Angabe von *alt_text* verwendet wurden. Es besteht aus den Wörtern bzw. Zeichen, die *alt_text* ersetzen sollen.

/Kommando

Eins der folgenden vier Kommandos:

- g** *alt_text* bei jedem Auftreten in den angegebenen Zeilen ersetzen.
- l** Die letzte Zeile Text, in der eine Ersetzung vorgenommen wurde, einschließlich der nicht druckbaren Zeichen anzeigen lassen (näheres dazu im letzten Abschnitt dieses Kapitels unter "Weitere nützliche Kommandos und Hinweise").
- n** Die letzte Zeile Text, in der eine Ersetzung vorgenommen wurde, mit vorangestellter Zeilenadresse anzeigen lassen.
- p** Die letzte Zeile Text, in der eine Ersetzung vorgenommen wurde, anzeigen lassen.

Ersetzen in der aktuellen Zeile

Das einfachste Beispiel für das Ersetzungskommando besteht darin, eine Änderung in der aktuellen Zeile vorzunehmen. In diesem Fall braucht keine Zeilenadresse angegeben zu werden:

```
s/alt_text/neu_text/<CR>
```

Im folgenden Beispiel ist ein Tippfehler enthalten. Die Zeile mit dem Fehler ist noch die aktuelle Zeile, und der Fehler wird durch Ersetzen korrigiert. Als *alt_text* ist hier **hä** aus der Zeichenkette **Fhäler**, als *neu_text* ist **eh** einzugeben.

```
a <CR>
Am Anfang habe ich einen Fhäler gemacht.
.<CR>
.p <CR>
Am Anfang habe ich einen Fhäler gemacht.
s/hä/eh <CR>
```

Dabei ist zu beachten, daß `ed` nach einem Ersetzungskommando keine Antwort ausgibt. Will man überprüfen, ob das Kommando erfolgreich ausgeführt wurde, muß man sich die Zeile mit `p` oder `n` anzeigen lassen, bzw. `p` oder `n` bereits in der Kommandozeile mit angeben. Im folgenden Beispiel wird das Kommando `n` verwendet, um zu überprüfen, ob die Zeichenkette `datei` für die Zeichenkette `maschine` eingesetzt wurde:

```
.p<CR>
Das ist die Testmaschine
s/maschine/datei/n<CR>
1      Das ist die Testdatei
```

Man erreicht dies jedoch auch auf einem kürzeren Weg: das Ergebnis des Kommandos wird automatisch angezeigt, wenn man das letzte Begrenzungszeichen nach dem Argument `neu_text` wegläßt:

```
.p<CR>
Das ist die Testdatei
s/datei/strecke<CR>
Das ist die Teststrecke
```

Ersetzen in einer bestimmten Zeile

Will man Text in einer Zeile ersetzen lassen, die nicht die aktuelle Zeile ist, gibt man in der Kommandozeile eine Adresse wie folgt an:

```
[Adresse1]s/alt_text/neu_text/<CR>
```

Im folgenden Beispiel enthält die Kommandozeile eine Adresse für die Zeile, die zu ändern ist (Zeile 1), da Zeile 3 die aktuelle Zeile ist:

```
1,3p<CR>
Das ist die Pestmaschine testen testen
in der Maschine
.<CR>
in der Maschine
1s/Pest/Test<CR>
Das ist die Testmaschine
```

Hier zeigt **ed** die neue Zeile nach der Ersetzung automatisch an, da das letzte Begrenzungszeichen weggelassen wurde.

Ersetzen in einem Zeilenbereich

Man kann eine Ersetzung in einem Bereich von Zeilen vornehmen lassen, indem man den Bereich von der ersten Adresse (*Adresse1*) bis zur letzten Adresse (*Adresse2*) angibt.

```
[Adresse1,Adresse2]s/alt_text/neu_text/<CR>
```

Findet **ed** das zu ersetzende Zeichenmuster in einer dieser Zeilen nicht, wird sie auch nicht geändert.

Im folgenden Beispiel werden alle Zeilen einer Datei mit einem Ersetzungskommando adressiert. Es werden jedoch nur die Zeilen geändert, die die Zeichenkette `es` (das Argument `alt_text`) enthalten:

```
1,$p<CR>
Das ist die Testmaschine
testen testen
in der Maschine
testen 1, 2, 3
1,$s/es/ES/n<CR>
4      tESTen 1, 2, 3
```

Gibt man einen Zeilenbereich mit `p` oder `n` am Ende der Kommandozeile für die Ersetzung an, wird nur die letzte geänderte Zeile angezeigt.

Sollen alle Zeilen angezeigt werden, in denen Text geändert wurde, sind die Kommandos `n` bzw. `p` mit der Adresse `1,$` anzugeben.

```
1,$n<CR>
1      Das ist die TESTmaschine
2      tESTen testen
3      in der Maschine
4      tESTen 1, 2, 3
```

Hier ist zu beachten, daß es in Zeile 2 nur an der ersten Stelle des Auftretens geändert wurde. Soll ein Muster an jeder Stelle, an der es auftritt, geändert werden, ist das Kommando `g` zu verwenden; es wird im folgenden Abschnitt beschrieben.

Globales Ersetzen

Die globale Ersetzung einer Zeichenkette durch eine andere ist eine im Editor `ed` vielseitig verwendbare Funktion. Setzt man das Kommando `g` hinter das letzte Begrenzungszeichen der Kommandozeile zum Ersetzen, kann man ein Muster in den angegebenen Zeilen an jeder Stelle seines Auftretens ersetzen lassen. Nun soll die Zeichenkette des vorigen Beispiels bei jedem Auftreten ersetzt werden. Dazu noch ein Hinweis: mit dem Kommando `u` kann man das zuletzt ausgeführte Ersetzungskommando wieder rückgängig machen.

```
u<CR>
1,$p<CR>
Das ist die Testmaschine
testen testen
in der Maschine
testen 1, 2, 3
1,$s/es/ES/g<CR>
1,$p<CR>
Das ist die TESTmaschine
tESTen tESTen
in der Maschine
tESTen 1, 2, 3
```

Man kann auch ein globales Suchmuster anstelle des durch `1,$` angegebenen Zeilenbereichs als Adresse verwenden.

```
I,$p<CR>
Das ist die Testmaschine
testen testen
in der Maschine
testen 1, 2, 3
g/est/s/es/ES/g<CR>
I,$p<CR>
Das ist die TESTmaschine
tESTen tESTen
in der Maschine
tESTen 1, 2, 3
```

Ist das globale Suchmuster eindeutig und entspricht es dem Argument *alt_text* (zu ersetzender Text), kann man dasselbe Ergebnis auch auf kürzerem Wege erreichen: Das Muster einmal als globale Suchadresse angeben, und es nicht als Argument *alt_text* wiederholen. Der Editor **ed** merkt sich das Muster aus der Suchadresse und verwendet es erneut als zu ersetzendes Muster.

```
g/alt_text/s//neu_text/g<CR>
```

NOTE

Wenn man diese Kurzform verwendet, ist darauf zu achten, daß zwei Schrägstriche (//) nach dem Kommando **s** zu setzen sind.

```

1,$p<CR>
Das ist die Testmaschine
testen testen
in der Maschine
testen 1, 2, 3
g/es/s//ES/g<CR>
1,$p<CR>
Das ist die TESTmaschine
tESTen tESTen
in der Maschine
tESTen 1, 2, 3

```

Nun mit weiteren Suchmusteradressen experimentieren:

```

/Muster<CR>
?Muster<CR>
v/Muster<CR>

```

Diese Suchmusteradressen können mit dem Ersetzungskommando kombiniert werden. Im folgenden Beispiel wird die Suche mit *v/Muster* dazu verwendet, Zeilen zu suchen, die das Muster *testen* nicht enthalten. Anschließend wird das Kommando *s* ausgeführt, um in diesen Zeilen das bestehende Muster (*in*) durch das neue Muster (*an*) zu ersetzen.

```

v/testen/s/in/an<CR>
Das ist die Testmaschine
an der Maschine

```

Dabei ist zu beachten, daß die Zeile `Das ist die Testmaschine` ebenfalls angezeigt wird, obwohl in ihr keine Ersetzung vorgenommen wurde. Wird das letzte Begrenzungszeichen weggelassen, werden alle Zeilen mit der angegebenen Suchadresse ausgegeben, unabhängig davon, ob eine Ersetzung in ihnen vorgenommen wurde oder nicht.

Nun mit dem Kommando `g` nach den Zeilen suchen, die das Muster `testen` enthalten.

```
g/testen/s//springen<CR>
springen testen
springen 1, 2, 3
```

Hier ist zu beachten, daß das Kommando eine Ersetzung nur beim jeweils ersten Auftreten von *Muster* (`testen`) in jeder Zeile vornimmt. Auch hier werden die Zeilen wieder auf dem Terminal ausgegeben, da das letzte Begrenzungszeichen weggelassen wurde.

Übung 4

- 4-1. Ändern Sie in der Datei `orte` jedes Auftreten von `Ort` in `Platz`, und zwar in allen Zeilen außer der, in welcher `Ort` zum Vorkommt.

Die Datei sollte so aussehen:

```
Mir liegt ein Ort wie
Hamburg
Als wäre man irgendwo in
Passau
Ein Ort zum Wohlfühlen ist
Berlin
Ich hab mein Herz verloren in
Heidelberg
Ich habe $$ verloren in
Hotels in
München
```

- 4-2. Verwenden Sie das Fragezeichen (?) als Begrenzungszeichen. Ändern Sie die aktuelle Zeile

```
München
```

in:

```
Frankfurt
```

Da hier die ganze Zeile geändert wird, kann man auch das Änderungskommando `c` verwenden.

- 4-3. Suchen Sie in der Datei rückwärts nach dem Wort

```
verloren
```

und ersetzen Sie es durch

```
gefunden,
```

wobei Sie ? als Begrenzungszeichen verwenden. Funktioniert dies?

4-4. Suchen Sie in der Datei vorwärts nach

irgendwo

und ersetzen Sie es durch

IRGENDWO.

Was geschieht, wenn man versucht, das Fragezeichen (?) als Begrenzungszeichen zu verwenden?

Experimentieren Sie nun mit den verschiedenen Kommandokombinationen, die zum Adressieren von Zeilenbereichen und für globales Suchen zur Verfügung stehen.

Was geschieht, wenn man versucht, \$\$ durch etwas anderes zu ersetzen? Versuchen Sie , in Zeile 9 der Datei \$ durch **viele \$** zu ersetzen. Dazu geben Sie

9s/\$/viele \$<CR>

ein. Was geschieht?

Sonderzeichen

Versucht man, das Dollarzeichen (\$) in der Zeile

Ich habe \$ in München verloren

zu ersetzen, wird man feststellen, daß nicht das \$ ersetzt wird, sondern der neue Text an das Ende der Zeile gesetzt wird. Das Dollarzeichen (\$) ist im Editor **ed** ein Sonderzeichen, das für das Ende der Zeile steht.

In **ed** gibt es mehrere Sonderzeichen, die als Kürzel für Such- und Ersetzungsmuster dienen. Die Zeichen arbeiten wie Globalzeichen. Gibt man sie ein, sieht das Ergebnis wahrscheinlich anders aus als erwartet.

Folgende Sonderzeichen stehen im Editor zur Verfügung:

- . Steht für ein beliebiges Zeichen.
- * Steht für kein oder mehrfaches Auftreten des vorherigen Zeichens.
- . * Steht für kein oder mehrfaches Auftreten eines beliebigen Zeichens nach dem Punkt.
- ^ Zeilenanfang.
- \$ Zeilenende.
- \ Sonderbedeutung des Sonderzeichens nach diesem Zeichen aufheben.
- & Den zu ersetzenden (alten) Text im neuen Text des Musters wiederholen.
- [...] Erstes Auftreten eines in diese Klammern gesetzten Zeichens.
- [^...] Erstes Auftreten eines nicht in die Klammern gesetzten Zeichens.

Im folgenden Beispiel sucht **ed** nach jeder Zeichenkette aus drei Zeichen, die auf "er" endet.

```
l,$p<CR>
Kater
Eber
Katze
Sau
Tier
g/.er<CR>
Kater
Eber
Tier
```

Hier ist zu beachten, daß das Wort "Tier" miterfaßt wird, da das Muster ".er" in der Zeichenkette Tier vorkommt.

Der Stern (*) steht für kein oder mehrfaches Auftreten eines bestimmten Zeichens in einem Such- oder Ersetzungsmuster. Dies ist nützlich, wenn man Wiederholungen von fälschlicherweise eingegebenen Zeichen löschen will. Dies ist beispielsweise der Fall, wenn man die Taste R zu lange drückt, während man das Wort "brechen" tippt. Mit dem Stern (*) kann man durch ein Ersetzungskommando alle überflüssigen r löschen:

```
p<CR>
brrrechen
s/r*/br<CR>
brechen
```

Hier ist zu beachten, daß das Muster den Buchstaben `b` vor dem ersten `r` enthält. Wird es nicht angegeben, wird es bei der Suche mit `*` als "kein Auftreten" eines `r` interpretiert, die Ersetzung wird ausgeführt und das Kommando beendet (ein Muster wird stets nur beim ersten Auftreten ersetzt, wenn man nicht eine globale Suche mit `g` durchführt). Auf dem folgenden Bildschirm wird dargestellt, wie diese Ersetzung ausgeführt wird, wenn man nicht das `b` und das `r` vor dem `*` angibt:

```
p<CR>
brrrechen
s/r*/r<CR>
rbrrrechen
```

Verwendet man den Punkt und den Stern, trifft das Muster auf alle Zeichen zu. Mit dieser Kombination kann man alle Zeichen im letzten Teil einer Zeile ersetzen lassen:

```
p<CR>
Kröten sind schleimige, kalte Tiere
s/sind.*/sind angenehm und warm <CR>
Kröten sind angenehm und warm
```

Mit `.*` kann man auch alle Zeichen zwischen zwei Mustern ersetzen lassen.

```
p<CR>
Kröten sind schleimige, kalte Tiere
s/sind.*Tie/sind
angenehme und warme Tie<CR>
Kröten sind angenehme und warme Tiere
```

Will man ein Wort am Anfang einer Zeile einfügen, kann man den Zirkumflex (^) für den alten, zu ersetzenden Text einsetzen. Dies ist sehr nützlich, wenn man dasselbe Muster vor mehreren Zeilen einfügen will. Im folgenden Beispiel wird das Wort **all** an den Anfang jeder Zeile gesetzt:

```
l,$p<CR>
die großen und kleinen Geschöpfe
die angenehmen und wohltuenden Dinge
die leuchtenden und schönen Dinge
l,$s/^/all /<CR>
l,$p<CR>
all die großen und kleinen Geschöpfe
all die angenehmen und wohltuenden Dinge
all die leuchtenden und schönen Dinge
```

Mit dem Dollarzeichen \$ können Zeichen am Ende einer Zeile oder eines Zeilenbereiches angefügt werden:

```

1,$p<CR>
Ich liebe
Ich brauche
Ich benutze
Das Finanzamt will mein
1,$s/$/ Geld.<CR>
1,$p<CR>
Ich liebe Geld.
Ich brauche Geld.
Ich benutze Geld.
Das Finanzamt will mein Geld.

```

In diesen Beispielen ist darauf zu achten, daß nach dem Wort `all` bzw. vor dem Wort `Geld` ein Leerzeichen eingegeben werden muß, da `ed` die angegebenen Zeichen ganz am Anfang bzw. direkt am Ende des Satzes einfügt. Vergibt man beispielsweise, vor dem Wort `Geld` ein Leerzeichen einzufügen, sieht die Datei folgendermaßen aus:

```

1,$s/$/Geld/<CR>
1,$p<CR>
Ich liebeGeld
Ich braucheGeld
Ich benutzeGeld
Das Finanzamt will meinGeld

```

Das Dollarzeichen (\$) ist auch sehr nützlich, um an das Zeilenende Satzzeichen zu setzen:

```

1,$p<CR>
Ich liebe Geld
Ich brauche Geld
Ich benutze Geld
Das Finanzamt will mein Geld
1,$s/$./.<CR>
1,$p/<CR>
Ich liebe Geld.
Ich brauche Geld.
Ich benutze Geld.
Das Finanzamt will mein Geld.

```

Da der Punkt (.) nicht für ein Zeichen steht (alter Text), sondern ein Zeichen ersetzen soll (neuer Text), hat er keine Sonderbedeutung. Um einen Punkt innerhalb einer Zeile zu ändern, muß die Sonderbedeutung des Punktes im alten Text aufgehoben werden. Dazu stellt man einfach einen Backslash (\) vor den Punkt. Mit diesem hebt man die Sonderbedeutung bestimmter Sonderzeichen auf, die als normale Textzeichen in Such- oder Ersetzungsmustern behandelt werden sollen. Der folgende Bildschirm ist ein Beispiel dafür, wie die Sonderbedeutung des Punktes aufgehoben wird:

```

p<CR>
Nach Hause gehn. Lisa sehn!
s/\./!<CR>
Nach Hause gehn! Lisa sehn!

```

Dasselbe gilt auch für den Backslash selbst. Soll ein \ als normales Textzeichen behandelt werden, ist ein \ voranzustellen. Soll beispielsweise das Symbol \ durch das Wort Backslash ersetzt werden, ist die Kommandozeile des folgenden Bildschirms zu verwenden:


```
1,2p<CR>
In diesem Abschnitt wird
der \ erläutert.
s/\\/Backslash<CR>
der Backslash erläutert.
```

Soll Text hinzugefügt werden, ohne den Rest einer Zeile zu ändern, kann man das Zeichen & als Kürzel verwenden. Das & bedeutet, daß der alte Text im Ersetzungsmuster wiederholt wird, so daß er nicht zweimal eingegeben werden muß. Beispiel:

```
p<CR>
Die sterblichen Überreste des Neanderthalers
s/des/& alten/<CR>
p<CR>
Die sterblichen Überreste des alten Neanderthalers
```

Die zuletzt verwendete Zeichenkette in einem Suchmuster bzw. der alte Text aus einem Ersetzungsvorgang wird von **ed** automatisch beibehalten. Der Editor muß jedoch dazu aufgefordert werden, die Ersetzungszeichen bei einer Ersetzung zu wiederholen; dies geschieht mit dem Prozentzeichen (%). Damit kann man dieselbe Ersetzung auf mehrere Zeilen anwenden, ohne eine globale Ersetzung durchführen zu lassen. Will man beispielsweise das Wort Geld in das Wort Gold umändern, kann man hier die letzte Ersetzung aus Zeile 1 mit Zeile 3 wiederholen, nicht dagegen mit Zeile 4.

```
1,$n<CR>
1   Ich liebe Geld
2   Ich brauche Essen
3   Ich benutze Geld
4   Die IRS will mein Geld
1s/Geld/Gold<CR>
Ich liebe Gold
3s//%<CR>
Ich benutze Gold
1,$n<CR>
1   Ich liebe Gold
2   Ich brauche Essen
3   Ich benutze Gold
4   Das Finanzamt will mein Geld
```

Der Editor **ed** merkt sich das Wort **Geld** (den alten zu ersetzenden Text) automatisch, so daß es zwischen den ersten beiden Begrenzungszeichen nicht wiederholt werden muß. Mit dem Zeichen **%** wird **ed** angewiesen, das zuletzt verwendete Ersetzungsmuster, **Gold**, zu verwenden.

ed sucht nach der ersten Stelle des Auftretens eines der in Klammern gesetzten Zeichen und versucht, den angegebenen alten Text durch den neuen Text zu ersetzen. Die Klammern können an jeder Stelle des zu ersetzenden Musters stehen.

Im folgenden Beispiel werden die Zahlen 6, 7, 8 oder 9 an der ersten Stelle ihres Auftretens in den Zeilen in 4 umgeändert, in denen eine dieser Zahlen auftritt:

```

1,$p<CR>
Montag      33,000
Dienstag    75,000
Mittwoch    88,000
Donnerstag  62,000
1,$s/[6789]/4<CR>
Montag      33,000
Dienstag    45,000
Mittwoch    48,000
Donnerstag  42,000

```

Im nächsten Beispiel wird Mr oder Ms aus einer englischen Namensliste gelöscht:

```

1,$p<CR>
Mr Arthur Middleton
Mr Matt Lewis
Ms Anna Kelley
Ms M. L. Hodel
1,$s/M[rs] //<CR>
1,$p<CR>
Arthur Middleton
Matt Lewis
Anna Kelley
M. L. Hodel

```

Steht ein Zirkumflex (^) als erstes Zeichen in der Klammer, wird er von ed als Anweisung interpretiert, nach Zeichen zu suchen, die nicht in den Klammern stehen. Steht er jedoch an einer anderen Stelle innerhalb der Klammer, wird er von ed als Zirkumflex-Zeichen behandelt.

```
1,$p<CR>
Kurs A Informatik
Kurs B Roboterdesign
Kurs A Boole'sche Algebra
Kurs D Laufen
Kurs C Tennis
1,$s/Kurs [^AB]/Kurs A<CR>
1,$p<CR>
Kurs A Informatik
Kurs B Roboterdesign
Kurs A Boole'sche Algebra
Kurs A Laufen
Kurs A Tennis
```

Verwendet man Sonderzeichen als Globalzeichen in Textteilen, die geändert werden sollen, ist stets darauf zu achten, daß man ein eindeutiges Zeichenmuster auswählt. Hätte man im obigen Beispiel nur das Kommando

`1,$s/[^AB]/A<CR>`

verwendet, wäre das K im Wort Kurs in A geändert worden.

Experimentieren Sie nun mit diesen Sonderzeichen. Dabei sollten Sie auch verschiedene Kombinationen verwenden, um zu sehen, was passiert (bzw. nicht passiert).

In Abbildung 5-7 werden die Sonderzeichen für Such- und Ersetzungsmuster zusammengefaßt.

Kommando	Funktion
.	Steht für ein beliebiges Zeichen in einem Such- oder Ersetzungsmuster.
*	Steht für kein oder mehrfaches Auftreten des vorangestellten Zeichens in einem Such- oder Ersetzungsmuster.
.*	Steht für kein oder mehrfaches Auftreten der Zeichen, die nach dem Punkt stehen.
^	Steht für den Anfang der Zeile in einem Ersetzungsmuster bzw. in einem Suchmuster.
\$	Steht für das Ende der Zeile in einem Ersetzungsmuster.
\	Die Sonderbedeutung des Sonderzeichens, das im Such- oder Ersetzungsmuster folgt, aufheben.
&	Den alten zu ersetzenden Text, der im Ersetzungsmuster angegeben wird, im neuen Text wiederholen.
%	Steht für das zuletzt verwendete Ersetzungsmuster.
[...]	Steht für das erste Auftreten des Zeichens in Klammern.
[^...]	Steht für das erste Auftreten eines Zeichens, das nicht in den Klammern steht.

Abbildung 5-7: Übersicht über die Sonderzeichen

Übung 5

- 5-1. Erstellen Sie eine Datei mit folgenden Textzeilen:

A Informatik
D Joggen
C Tennis

Was geschieht, wenn man folgende Kommandozeile eingibt:

```
1,$s/[^AB]/A/<CR>
```

Machen Sie das obige Kommando wieder rückgängig. Wie kann man dabei C und D eindeutig machen? Ein Tip: Sie befinden sich am Anfang der Zeile an der Stelle, für die der Zirkumflex (^) steht. Keine Angst vor Experimenten!

- 5-2. Folgende Zeile vor Zeile 2 einfügen:

Das sind nicht gerade meine Fächer.

Erstellen Sie mit Klammern und dem Zeichen ^ ein Suchmuster, mit dem man die eingefügte Zeile wieder finden kann. Es gibt mehrere Möglichkeiten, eine Zeile zu adressieren. Beim Bearbeiten von Text sollte man stets die Methode verwenden, die man am schnellsten und einfachsten findet.

- 5-3. Nehmen Sie folgende Zeilen in die Datei auf:

Ich liebe Geld
Ich brauche Geld
Das Finanzamt will mein Geld

Ändern Sie nun diese Zeilen mit einem Kommando in folgende Zeilen:

Es ist mein Geld
Es ist mein Geld
Das Finanzamt will mein Geld

Führen Sie mit zwei Kommandozeilen folgendes aus: in der ersten Zeile soll das Wort **Geld in Gold** geändert werden, und in den letzten beiden Zeilen das Wort **Geld in Gold**, ohne die Wörter **Geld** oder **Gold** selbst zu verwenden.

5-4. Wie können Sie die Zeile

1020231020

in

10202031020

ändern, ohne die alten Ziffern im Ersetzungsmuster zu wiederholen?

5-5. Erstellen Sie eine Textzeile mit folgenden Zeichen:

.\&%^

Ersetzen Sie jedes Zeichen durch einen Buchstaben. Brauchen Sie bei jeder Ersetzung einen Backslash?



Text verschieben/kopieren

Bisher wurde behandelt, wie man Zeilen adressiert, Text erstellt und löscht und Ersetzungen vornimmt. Der Editor **ed** bietet eine weitere Gruppe vielseitiger und wichtiger Kommandos. Man kann im Editierpuffer Zeilen verschieben, kopieren oder zusammenfügen. Außerdem kann man Text aus einer Datei einlesen, die sich nicht im Editierpuffer befindet oder Zeilen der Datei im Puffer in eine andere Datei im aktuellen Verzeichnis schreiben. Mit folgenden Kommandos kann man Text verschieben/kopieren:

m	Textzeilen verschieben
t	Textzeilen kopieren
j	Aufeinanderfolgende Textzeilen zusammenfügen
w	Textzeilen in eine Datei schreiben
r	Inhalt einer Datei einlesen

Textzeilen verschieben

Mit dem Kommando **m** kann man Textblöcke an eine andere Stelle der Datei verschieben. Das allgemeine Format für dieses Kommando lautet:

```
[Adresse1,Adresse2]m[Adresse3]<CR>
```

Die einzelnen Bestandteile dieser Kommandozeile bedeuten folgendes:

Adresse1,Adresse2

Der Bereich von Textzeilen, der verschoben werden soll. Wird nur eine Zeile verschoben, ist nur *Adresse1* anzugeben. Wird keine Adresse angegeben, wird die aktuelle Zeile verschoben.

m Das Kommando zum Verschieben.

Adresse3 Der Text wird nach dieser Zeile eingefügt.

Probieren Sie mit dem folgenden Beispiel aus, wie das Kommando arbeitet. Erstellen Sie dazu eine Datei mit folgenden drei Textzeilen:

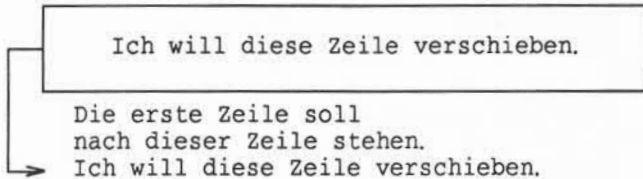
Ich will diese Zeile verschieben.

Die erste Zeile soll nach dieser Zeile stehen.

Das Kommando

1m3<CR>

eingeben. Damit verschiebt ed Zeile 1 hinter Zeile 3.



Auf dem folgenden Bildschirm wird gezeigt, wie dieser Vorgang auf dem Terminal erscheint:

```

1,$p<CR>
  Ich will diese Zeile verschieben.
Die erste Zeile soll
nach dieser Zeile stehen.
1m3<CR>
1,$p<CR>
Die erste Zeile
soll nach dieser Zeile stehen.
Ich will diese Zeile verschieben.

```

Soll ein Absatz verschoben werden, gibt man mit *Adresse1* und *Adresse2* den Zeilenbereich an, der den Absatz umfaßt.

Im folgenden Beispiel wird ein Textblock (Zeilen 8 bis 12) hinter die Zeile 65 verschoben. Dabei ist das Kommando **n** zu beachten; es bewirkt, daß die Zeilennummern der Datei ausgegeben werden:

```
8,12n<CR> 8 Dies ist Zeile 8.  
9 Sie steht am Anfang eines  
10 sehr kurzen Absatzes.  
11 Der Absatz endet  
12 mit dieser Zeile.  
64,65n<CR>  
64 Den Textblock  
65 hinter diese Zeile verschieben.  
8,12m65<CR> 59,65n<CR>  
59 Den Textblock  
60 hinter diese Zeile verschieben.  
61 Dies ist Zeile 8.  
62 Sie steht am Anfang eines  
63 sehr kurzen Absatzes.  
64 Der Absatz endet  
65 mit dieser Zeile.
```

Wie kann man Zeilen vor die erste Zeile der Datei verschieben? Dazu folgendes Kommando versuchen:

```
3,4m0<CR>
```

Wird als *Adresse* 0 angegeben, werden die Zeilen an den Anfang der Datei gesetzt.

Textzeilen kopieren

Das Kopierkommando **t** (Transfer) arbeitet wie das Kommando **m**, mit dem Unterschied, daß der Textblock an der ursprünglichen Adresse nicht gelöscht wird. Statt dessen wird eine Kopie dieses Textblocks nach der angegebenen Textzeile eingefügt.

Auch das allgemeine Format des Kommandos **t** ist dem des Kommandos **m** sehr ähnlich:

```
[Adresse1,Adresse2]t[Adresse3]<CR>
```

Adresse1,Adresse2

Der Zeilenbereich, der kopiert werden soll. Wird nur eine Zeile kopiert, ist nur *Adresse1* anzugeben. Wird keine Adresse angegeben, wird die aktuelle Zeile kopiert.

t Das Kommando zum Kopieren.

Adresse3 Die Kopie des Textes wird nach dieser Zeile eingefügt.

Im folgenden Beispiel wird gezeigt, wie drei Textzeilen hinter die letzte Zeile kopiert werden:

Sicherheitsmaßnahmen:

Bei einem Brand im Gebäude:

Die Zimmertür schließen, um das Feuer zu isolieren

Den nächsten Feuermelder einschlagen.
Den Knopf drücken.
Feuerlöscher holen und Brand bekämpfen.

·
·
·

Bei einem chemischen Brand im Labor gilt:

Den nächsten Feuermelder einschlagen.
Den Knopf drücken.
Feuerlöscher holen und Brand bekämpfen.

Die dafür erforderlichen Kommandos und die Reaktionen von **ed** werden im folgenden Bildschirm dargestellt. Auch hier werden wieder die Zeilennummern durch das Kommando **n** mit angezeigt:

```
5,8n<CR>
5 Die Zimmertür schließen, um das Feuer zu isolieren.
6 Den nächsten Feuermelder einschlagen.
7 Den Knopf drücken.
8 Feuerlöscher holen und Brand bekämpfen.
30n<CR>
30 Bei einem chemischen Brand im Labor gilt:
6,8t30<CR>
30,$n<CR>
30 Bei einem chemischen Brand im Labor gilt:
31 Den nächsten Feuermelder einschlagen.
32 Den Knopf drücken.
33 Feuerlöscher holen und Brand bekämpfen.
6,8n<CR>
6 Den nächsten Feuermelder einschlagen.
7 Den Knopf drücken.
8 Feuerlöscher holen und Brand bekämpfen.
```

Der Text der Zeilen 6 bis 8 bleibt an seiner ursprünglichen Stelle erhalten, und eine Kopie davon wird nach Zeile 30 eingefügt.

Experimentieren Sie nun mit den Kommandos **m** und **t** in einer eigenen Datei.

Aufeinanderfolgende Zeilen zusammenfügen

Mit dem Kommando **j** kann die aktuelle Zeile mit der darauffolgenden Zeile zusammengefügt werden. Das allgemeine Format für dieses Kommando lautet:

```
[Adresse1,Adresse1]j<CR>
```

Im folgenden Beispiel wird gezeigt, wie man mehrere Zeilen zusammenfügen kann. Eine einfache Methode besteht darin, die Zeilen, die man zusammenfügen möchte, mit **p** oder **n** anzuzeigen zu lassen:

```

1,2p<CR>
Nun ist es Zeit, das Team zusammenzustellen.
p<CR>
Team zusammenzustellen.
1p<CR>
Nun ist es Zeit, das
j<CR>
p<CR>
Nun ist es Zeit, dasTeam zusammenzustellen.

```

Dabei ist darauf zu achten, daß im obigen Beispiel nach dem letzten Wort (das) und dem ersten Wort der nächsten Zeile (Team) kein Leerzeichen gesetzt wird. Soll ein Leerzeichen an dieser Stelle stehen, muß es mit dem Kommando s eingefügt werden.

Textzeilen in eine Datei schreiben

Mit dem Kommando w wird Text aus dem Puffer in eine Datei geschrieben. Das allgemeine Format für dieses Kommando lautet:

```
[Adresse1,Adresse2]w [Dateiname]<CR>
```

Adresse1,Adresse2

Der Zeilenbereich, der in eine andere Datei geschrieben werden soll. Wird *Adresse1* oder *Adresse2* nicht angegeben, wird die gesamte Datei in eine neue Datei geschrieben.

w

Das Schreibkommando bzw. Kommando zum Abspeichern

Dateiname

Der Name der neuen Datei, die eine Kopie des Textblockes enthalten soll.

Im folgenden Beispiel wird der Text eines Briefs in einer Datei mit dem Namen **info** gespeichert, um ihn später an verschiedene Personen zu schicken.

```
1,$n<CR>
1      17.03.1989
2      Hallo Gerd,
3      Heute findet um 16:30 h im Konferenzraum
4      eine Versammlung statt.
5      Für Erfrischungen ist gesorgt.
3,$w info<CR>
93
```

Mit dem Kommando **w** kann man eine Kopie der Zeilen 3 bis 5 in eine neue Datei mit dem Namen **info** schreiben. Der Editor **ed** zeigt dann die Anzahl der Zeichen in der neuen Datei an.

Wichtige Hinweise

Durch das Kommando **w** werden bestehende Dateien desselben Namens ohne vorherige Warnung überschrieben; d. h. sie werden gelöscht, und die neue Datei enthält dann den definierten Textblock. Hätte es im obigen Beispiel bereits eine Datei des Namens **info** gegeben, bevor der Textblock in diese Datei geschrieben wurde, wäre sie dadurch gelöscht worden.

Im Abschnitt "Sonderkommandos" weiter unten in diesem Kapitel wird beschrieben, wie man aus dem Editor **ed** Shell-Kommandos ausführen kann. Damit kann man die Dateinamen des Verzeichnisses auflisten lassen, um sicherzustellen, daß man keine bestehende Datei überschreibt.

Eine weitere Schwierigkeit ergibt sich daraus, daß man in die Datei **info** mit diesem Kommando keine weiteren Zeilen schreiben kann. Versucht man, neue Zeilen hinzuzufügen, werden die bestehenden Zeilen (3 bis 5) gelöscht, und die Datei enthält nur die neuen Zeilen.

Inhalt einer Datei einlesen

Mit dem Kommando `r` kann man Text aus einer Datei in den Puffer einfügen. Das allgemeine Format für dieses Kommando lautet:

```
[Adresse1]r Name <CR>
```

Adresse1 Der Text wird nach der Zeile mit dieser *Adresse1* eingefügt. Wird *Adresse1* nicht angegeben, wird die Datei an das Ende des Puffers angehängt.

`r` Das Kommando zum Einlesen.

Name Der Name der Datei, die in den Editierpuffer kopiert werden soll.

Unter Verwendung des obigen Beispiels mit dem Kommando `write` wird im folgenden Bildschirm dargestellt, wie eine Datei durch Einlesen von neuem Text bearbeitet werden kann:

```
1,$n<CR>
1          17.03.1989
2      Hallo Michael,
3      Hast Du heute nachmittag Zeit?
4      Bis dann.
3r info<CR>
93
3,$n<CR>
3      Hast Du heute nachmittag Zeit?
4      Heute findet um 16:30 h im Konferenzraum
5      eine Versammlung statt.
6      Für Erfrischungen ist gesorgt.
7      Bis dann.
```

Die Reaktion von `ed` auf das Lesekommando besteht darin, daß die Anzahl der Zeichen der Datei angezeigt wird, die in den Puffer aufgenommen wurde (in diesem Beispiel von `info`).

Text verschieben/kopieren

Man sollte neue oder geänderte Textzeilen stets anzeigen lassen, um sicherzustellen, daß das Ergebnis korrekt ist.

In Abbildung 5-8 werden die Kommandos von `ed` zum Verschieben/Kopieren von Text zusammengefaßt.

Kommando	Funktion
<code>m</code>	Textzeilen verschieben
<code>t</code>	Textzeilen kopieren
<code>j</code>	Aufeinanderfolgende Zeilen zusammenfügen
<code>w</code>	Text in eine neue Datei schreiben/abspeichern
<code>r</code>	Text aus einer anderen Datei einlesen

Abbildung 5-8: Übersicht über die Kommandos von `ed` zum Verschieben/Kopieren von Text

Übung 6

- 6-1. Es gibt zwei Möglichkeiten, Text im Puffer zu kopieren: erstens durch das Kopierkommando, zweitens durch Schreiben von Text in eine Datei mit dem Kommando zum Abspeichern (w) und anschließendem Einlesen der Datei in den Puffer mit dem Kommando zum Einlesen (r).

Für die zweite Methode benötigt man mehr Zeit. In welchem Fall kann diese Methode jedoch praktischer sein?

Mit welchen Kommandos kann man die Zeilen 10 bis 17 der Datei **übung** in die Datei **übung6** in Zeile 7 kopieren?

- 6-2. Die Zeilen 33 bis 46 sind ein Textblock, der nach Zeile 3, jedoch nicht mehr nach Zeile 32 stehen soll. Mit welchem Kommando kann man dies erreichen?
- 6-3. Angenommen, man befindet sich in Zeile 10 einer Datei und möchte die Zeilen 13 und 14 zusammenziehen. Mit welchen Kommandos kann man dies erreichen?

Weitere nützliche Kommandos und Hinweise

Für Editiersitzungen gibt es vier weitere nützliche Kommandos und eine spezielle Datei.

- | | |
|---------------|--|
| h,H | Aufrufen der Hilfefunktion; durch sie werden Informationen zu Fehlern ausgegeben. |
| l | Anzeigen von Zeichen, die normalerweise nicht angezeigt werden |
| f | Namen der aktuellen Datei anzeigen |
| ! | Den Editor ed vorübergehend verlassen, um ein Shell-Kommando auszuführen |
| ed.hup | Bei einer Systemunterbrechung wird der Puffer von ed in eine spezielle Datei mit dem Namen ed.hup gesichert. |

Hilfefunktionen

Beim Editieren einer Datei wird man feststellen, daß **ed** auf bestimmte Kommandos mit einem Fragezeichen reagiert. Dieses **?** ist eine Meldung, die von **ed** ausgegeben wird, wenn ein Fehler aufgetaucht ist. Mit den Hilfefunktionen kann man eine kurze Meldung anzeigen lassen, in der der letzte Fehler kurz erläutert wird.

Zwei Hilfefunktionen stehen zur Verfügung:

- h** Anzeigen einer kurzen Fehlermeldung mit einer Erläuterung der Ursache für das zuletzt angezeigte **?**.
- H** Die Hilfefunktion des Editors **ed** aktivieren, um bei jedem Erscheinen eines **?** eine Fehlermeldung anzeigen zu lassen (deaktivieren durch nochmalige Eingabe von **H**).

Es wurde bereits erläutert, daß **ed** ein **?** ausgibt, wenn man versucht, den Editor zu verlassen, ohne den Inhalt des Puffers in einer Datei abzuspeichern. Versuchen Sie dies nun, und geben Sie nach dem Erscheinen des **?** das Hilfefunktion **h** ein:

```
q<CR>
?
h<CR>
warning: expecting `w'
```

Das Fragezeichen erscheint auch, wenn man einen neuen Dateinamen in der Kommandozeile des Editors **ed** angibt. Geben Sie einen neuen Dateinamen ein und nach Erscheinen des **?** das Hilfekommando **h**, um festzustellen, was die Meldung bedeutet.

```
ed neudatei<CR>
? neudatei
h<CR>
cannot open input file
```

Diese Meldung kann zwei Dinge bedeuten: entweder ist keine Datei mit dem Namen **neudatei** vorhanden, oder eine solche Datei existiert, darf aber von **ed** nicht gelesen werden.

Wie oben bereits ausgeführt, wird durch das Kommando **H** der Hilfemodus des Editors aktiviert, so daß bei jedem Erscheinen eines **?** sofort danach eine Erläuterung der aufgetretenen Fehlerbedingung ausgegeben wird. Durch nochmalige Eingabe von **H** wird der Hilfemodus wieder deaktiviert. Auf dem nächsten Bildschirm wird dargestellt, wie der Hilfemodus mit **H** aktiviert wird. Außerdem sind einige Beispielmeldungen auf häufig auftretende Fehlerbedingungen darin enthalten:

```
$ ed neudatel<CR>
e neudatel<CR>
?neudatei
H<CR>
cannot open input file
/Hallo<CR>
?
illegal suffix
1,22p<CR>
?
line out of range
a<CR>
Diese Zeile wird an den Pufferinhalt angehängt.
.<CR>
s/$ Kaffeeklatsch<CR>
?
illegal or missing delimiter
,$s/$/ Kaffeeklatsch<CR>
?
unknown command
H<CR>
q<CR>
?
h<CR>
warning: expecting `w'
```

Im folgenden werden einige der am häufigsten auftretenden Fehlermeldungen bei Editiersitzungen aufgeführt:

illegal suffix

ed findet keine Entsprechung zum Suchmuster **Hallo**, weil der Puffer leer ist.

line out of range

ed kann keine Zeilen anzeigen, weil der Puffer leer ist oder sich die angegebene Zeile nicht im Puffer befindet.

Nun wird eine Textzeile in den Puffer geschrieben, um einige Fehlermeldungen zu erläutern, die auf das Kommando `s` hin auftreten können.

`illegal or missing delimiter`

Das Begrenzungszeichen zwischen dem alten, zu ersetzenden Text und dem neuen Text fehlt.

`unknown command`

Adresse1 wurde vor dem Komma nicht eingegeben; `ed` erkennt die Eingabe `,$` nicht.

Nun wird der Hilfemodus ausgeschaltet und das Kommando `h` stattdessen verwendet, um die Bedeutung des letzten `?` erläutern zu lassen. Solange man sich in den Editor `ed` einarbeitet, ist es günstig, den Hilfemodus zu aktivieren. Dazu das Kommando `H` verwenden. Hat man sich jedoch einmal mit der Benutzung von `ed` vertraut gemacht, braucht man sich in der Regel nur noch gelegentlich Erläuterungen zu Fehlern anzeigen zu lassen. Dann kann man von Fall zu Fall das Kommando `h` verwenden.

Nicht druckbare Zeichen anzeigen lassen

Drückt man die Tabulatortaste bzw. die entsprechende Tastenkombination, zeigt das Terminal normalerweise bis zu acht Leerzeichen (die Zahl der Zeichen bis zur nächsten Tabulatorposition) an. Die Tabulatoreinstellung kann auch auf mehr oder auf weniger als acht Zeichen eingestellt sein; nähere Angaben dazu sind in Kapitel 7, "Anleitung zur Shell", unter `stty` zu finden.

Will man sich anzeigen lassen, wieviele Tabulatorzeichen man in einen Text eingegeben hat, kann man das Kommando `l` (`list`) eingeben. Das allgemeine Format für das Kommando `l` ist praktisch dasselbe wie für die Kommandos `n` und `p`.

`[Adresse1,Adresse2]l<CR>`

Die Bestandteile dieser Kommandozeile haben folgende Bedeutungen:

Adresse1,Adresse2

Der Zeilenbereich, der angezeigt werden soll. Wird keine Adresse angegeben, wird die aktuelle Zeile angezeigt. Wird nur *Adresse1* angegeben, wird nur diese Zeile angezeigt.

- l Das Kommando, durch das nicht druckbare Zeichen zusammen mit dem Text angezeigt werden.

Durch das Kommando l werden Tabulatorzeichen mit dem Zeichen > (größer als) dargestellt. Steuerzeichen werden durch Drücken der CONTROL-Taste zusammen mit der betreffenden Buchstabentaste erzeugt. Die Taste für den Signalton ist ^g (CONTROL-g). Sie wird als \07 angezeigt; dies ist der oktale Code für ^g.

Geben Sie zwei Textzeilen mit einem <^g> (CONTROL-g) und einem Tabulatorzeichen ein. Lassen Sie sich dann die Textzeilen mit dem Kommando l auf dem Terminal anzeigen.

```
a<CR>
Ein <^g> (CONTROL-g) in diese Zeile eingeben.<CR>
Ein <tab> (Tabulatorzeichen) in diese Zeile eingeben.<CR>
.<CR>
l,2l<CR>
Ein \07 (CONTROL-g) in diese Zeile eingeben.<CR>
Ein > (Tabulatorzeichen) in diese Zeile eingeben.<CR>
```

Wurde bei der Eingabe von <^g> ein Tonsignal ausgegeben?

Der aktuelle Dateiname

Im Laufe einer langen Editiersitzung kann es vorkommen, daß man den Dateinamen vergißt. Mit dem Kommando f kann man sich den Namen der gerade im Puffer stehenden Datei anzeigen lassen. Oft möchte man auch die ursprünglich in den Puffer geladene Datei im Original erhalten und den Inhalt des Puffers (den geänderten Text) in einer neuen Datei abspeichern. Während einer längeren Editiersitzung kann man dies vergessen und versehentlich die ursprüngliche Datei mit der gewohnten Kommandofolge w und q überschreiben. Dies kann man vermeiden, indem man den Editor während der Editiersitzung anweist, dem Pufferinhalt einen neuen Dateinamen zuzuordnen. Dazu sind das Kommando f und ein neuer Dateiname einzugeben.

Das Format für dieses Kommando ist **f** alleine in einer Zeile:

f<CR>

Rufen Sie nun den Editor mit einer Datei auf, um zu sehen, wie **f** arbeitet. Heißt die Datei beispielsweise **altdatei**, reagiert **ed** wie im folgenden Bildschirm dargestellt:

```
ed altdatei<CR>
323
f<CR>
altdatei
```

Mit folgendem allgemeinen Format des Kommandos wird dem Inhalt des Puffers ein neuer Dateiname zugeordnet:

f neudatei<CR>

Wird mit dem Kommando zum Abspeichern (**w**) kein Dateiname angegeben, verwendet **ed** automatisch den zu Beginn der Editiersitzung eingegebenen Namen und speichert den Inhalt des Puffers in dieser Datei. Soll die ursprüngliche Datei nicht überschrieben werden, muß entweder mit dem Kommando zum Abspeichern ein neuer Dateiname angegeben oder mit dem Kommando **f**, gefolgt vom neuen Dateinamen, der aktuelle Dateiname geändert werden. Das Kommando **f** kann zu jeder Zeit in einer Editiersitzung eingegeben werden; daher kann man den Dateinamen sofort ändern und anschließend weiter editieren, ohne befürchten zu müssen, daß die ursprüngliche Datei überschrieben wird.

Auf dem folgenden Bildschirm wird dargestellt, wie der Editor mit der Datei **altdatei** aufgerufen und anschließend der Dateiname in **neudatei** geändert wird. Danach wird ein Textzeile in den Puffer eingefügt, und schließlich werden die Kommandos **w** und **q** zum Abspeichern der Datei bzw. Verlassen des Editors ausgeführt.



```
ed altdatei<CR>
323 r<CR>
al tdatei
f neudatei<CR>
neudatei
a<CR>
Eine Textzeile einfügen.<CR>
.<CR>
w<CR>
343
q<CR>
```

Nach der Rückkehr zur Shell kann man die Dateien auflisten lassen, um zu prüfen, ob die neue Datei **neudatei** vorhanden ist. In **neudatei** müßte nun eine Kopie des Inhalts von **altdatei** zuzüglich der neu eingegebenen Textzeile stehen.

Shell-Kommandos aus dem Editor ausführen

Um sicherzustellen, daß man nicht eine bestehende Datei überschreibt, wenn man den Inhalt des Editierpuffers einem neuen Namen zuordnet, muß man zur Shell zurückkehren und die Dateien auflisten lassen. Mit **!** kann man vorübergehend zur Shell zurückkehren, ein Shell-Kommando ausführen und anschließend wieder in die aktuelle Zeile des Editors zurückkehren.

Das allgemeine Format für dieses Kommando lautet:

```
!Shell-Kommandozeile <CR> Reaktion der Shell auf diese
Kommandozeile !
```

Das Zeichen **!** wird als erstes Zeichen einer Zeile eingegeben; das Shell-Kommando muß danach in derselben Zeile eingegeben werden. Die Reaktion des Programms auf das Kommando erscheint während seiner Ausführung. Wurde das Kommando ausgeführt, erscheint das Zeichen **!** alleine in einer Zeile. Das bedeutet, daß man sich wieder im Editor in der aktuellen Zeile befindet.

Beispielsweise kann man zur Shell zurückkehren, um sich das aktuelle Datum anzeigen zu lassen; dazu ! und das Shell-Kommando `date` eingeben:

```
p<CR>
Dies ist die aktuelle Zeile.
!date<CR>
Tue Apr 1 14:24:22 MES 1989
!
p<CR>
Dies ist die aktuelle Zeile.
```

Auf dem Bildschirm wird zuerst die aktuelle Zeile angezeigt. Dann wird das Kommando zum Verlassen des Editors und Anzeigen des Datums eingegeben. Nach der Ausgabe des aktuellen Datums erfolgt die Rückkehr in die aktuelle Zeile des Editors.

Sollen mehrere Kommandos in der Shell-Kommandozeile ausgeführt werden, siehe die Erläuterung zu ; im Abschnitt "Sonderzeichen" in Kapitel 7.

Wiederherstellung nach Systemunterbrechungen

Wenn im Verlauf einer Editiersitzung mit `ed` eine Systemunterbrechung auftritt, d. h. wenn es zu einem Systemabsturz kommt oder die Stromversorgung des Terminals unterbrochen wird, versucht das UNIX-System, den Inhalt des Editierpuffers in einer speziellen Datei mit dem Namen `ed.hup` zu sichern. Der Text kann aus dieser Datei später mit zwei verschiedenen Methoden wieder abgerufen werden. Die erste Methode besteht darin, der Datei `ed.hup` mit einem Shell-Kommando einen anderen Namen zuzuweisen, wie zum Beispiel den Namen, den man der Datei vor der Unterbrechung zugewiesen hatte. Die zweite Methode besteht darin, `ed` aufzurufen und den Inhalt des Puffers mit dem Kommando `f` umzubenennen. Auf dem folgenden Bildschirm wird ein Beispiel für die zweite Methode dargestellt:

```
ed ed.hup<CR>
928
f meinedatei<CR>
meinedatei
```

Verwendet man die zweite Methode, um den Inhalt des Puffers wiederherzustellen, ist die Datei **ed.hup** danach zu löschen.

Zusammenfassung

In diesem Kapitel wurden nun viele Kommandos des Editors **ed** eingeführt. Die in dieser Anleitung nicht behandelten Kommandos, wie zum Beispiel **G**, **P**, **Q** und die Verwendung der Klammern **()** und **{ }** sind unter **ed(1)** im *User's Reference Manual* beschrieben. Man kann mit diesen Kommandos experimentieren, um zu sehen, welche Aufgaben sie ausführen können.

In Abbildung 5-9 werden die Funktionen der in diesem Abschnitt eingeführten Kommandos zusammengefaßt.

Kommando	Funktion
h	Anzeigen eines kurzen, erläuternden Textes zur letzten Meldung ?.
H	Hilfemodus einschalten. Bei jeder Fehlermeldung (?) wird ein erläuternder Text ausgegeben. Nochmaliges Drücken von H deaktiviert den Hilfemodus.
l	Anzeigen nicht druckbarer Zeichen im Text.
f	Anzeigen des aktuellen Dateinamens.
f neudatei	Ändern des aktuellen Dateinamens, der dem Inhalt des Editierpuffers zugeordnet ist, in <i>neudatei</i> .
!Kommando	Aus dem Editor heraus ein Shell-Kommando (<i>Kommando</i>) ausführen.
ed.hup	Der Editierpuffer wird in der speziellen Datei ed.hup gesichert, wenn es vor einem Kommando zum Abspeichern zu einer Systemunterbrechung kommt.



Abbildung 5-9: Übersicht über weitere nützliche Kommandos

Übung 7

- 7-1. Legen Sie eine neue Datei unter dem Namen **neudatei1** an. Rufen Sie **ed** auf und ändern Sie den Dateinamen in **aktuell1**. Erstellen Sie dann etwas Text, speichern die Datei ab (schreiben) und verlassen **ed**. Führen Sie das Kommando **ls** aus, um zu prüfen, ob nicht bereits eine Datei unter dem Namen **neudatei1** in dem Verzeichnis steht. Beim Ausführen des Shell-Kommandos **ls** erkennt man, daß das Verzeichnis keine Datei des Namens **neudatei1** enthält.
- 7-2. Legen Sie eine Datei mit dem Namen **datei1** an und hängen Sie einige Textzeilen an (a). Verlassen Sie den Modus "Anhängen", speichern Sie die Datei jedoch noch nicht ab. Schalten Sie das Terminal aus und dann wieder ein. Melden Sie sich im System an. Führen Sie auf Shell-Ebene das Kommando **ls** aus. Ist eine neue Datei mit dem Namen **ed.hup** vorhanden? **ed.hup** mit dem Editor **ed** aufrufen. Wie kann man den aktuellen Dateinamen in **datei1** ändern? Lassen Sie sich den Inhalt der Datei anzeigen. Enthält sie denselben Text wie die Datei **datei1** vor dem Ausschalten des Terminals?
- 7-3. Kehren Sie aus dem Editor **ed** vorübergehend zur Shell zurück und senden Sie eine Nachricht an sich selbst.

Lösungen zu den Übungen

Übung 1

1-1.

```
$ ed schrott <CR>
? schrott
a <CR>
Hallo, schöne Welt. <CR>
.<CR>
w <CR>
19
q <CR>
$
```

1-2.

```
$ ed schrott <CR>
19
l,$p <CR>
Hallo, schöne Welt. <CR>
q <CR>
$
```



Das System hat hier nicht das Fragezeichen als Warnhinweis ausgegeben, da im Puffer keine Änderung vorgenommen wurde.

1-3.

```
$ ed schrott<CR>
19 a<CR>
Werners Pferd sprang durch das Fenster.<CR>
.<CR>
I,$p<CR>
Hallo, schöne Welt. Werners Pferd sprang durch das Fenster.
q<CR>
? w allerlei<CR>
58 q<CR>
$
```

Übung 2

2-1.

```
$ ed orte<CR>
? orte a<CR>
Mir liegt ein Ort wie<CR>
Hannover<CR>
Als wäre man irgendwo in<CR>
Frankfurt<CR>
Ein Ort zum Wohlfühlen ist<CR>
Wiesbaden<CR>
Ich hab mein Herz verloren in<CR>
Heidelberg<CR>
Ich habe $$ verloren in<CR>
München<CR>
.<CR> w<CR>
166
```

2-2.

```
3<CR> Als wäre man irgendwo in
```

2-3.

```
-2,+3p<CR>  
Mir liegt ein Ort wie  
Hannover  
Als wäre man irgendwo in  
Frankfurt  
Ein Ort zum Wohlfühlen ist  
Wiesbaden
```

2-4.

```
. = <CR>  
6  
6<CR>  
Wiesbaden
```


2-5.

```
$<CR>  
München
```

2-6.

```
?Ort<CR>  
Ein Ort zum Wohlfühlen ist  
?<CR>  
Mir liegt ein Ort wie
```

2-7.

g/in<CR>

Mir liegt ein Ort wie
Als wäre man nirgendwo in
Ein Ort zum Wohlfühlen ist
Ich hab mein Herz verloren in
Ich habe \$\$ verloren in

v/in<CR>

Hannover
Frankfurt
Wiesbaden
Heidelberg
München

Übung 3

3-1.

\$ ed üb3<CR>

?Üb3

l<CR>

?

q<CR>

Das Fragezeichen ? nach der Zeile mit dem i bedeutet, daß das Kommando fehlerhaft ist. Es gibt hier keine aktuelle Zeile, vor die Text eingefügt werden kann.

3-2.

```

$ ed orte<CR>
166
.n<CR>
10 München
3i<CR>
Stuttgart<CR>
.<CR>
.i<CR>
oder<CR>
Dortmund<CR>
.<CR>
$!<CR>
Hotels in<CR>
1,$n<CR>
 1 Mir liegt ein Ort wie
 2 Hannover
 3 oder
 4 Dortmund
 5 Stuttgart
 6 Als wäre man irgendwo in
 7 Frankfurt
 8 Ein Ort zum Wohlfühlen ist
 9 Wiesbaden
10 Ich hab mein Herz verloren in
11 Heidelberg
12 Ich habe $$ verloren in
13 Hotels in
14 München
    
```

3-3.

```
1,5n<CR>
1 Mir liegt ein Ort wie
2 Hannover
3 oder
4 Dortmund
5 Stuttgart
2,5e<CR>
Hamburg<CR>
.<CR>
1,3n<CR>
1 Mir liegt ein Ort wie
2 Hamburg
3 Als wäre man irgendwo in
```

3-4.

```
.<CR>
Als wäre man irgendwo in
/Fra<CR>
Frankfurt
e<CR>
Passau<CR>
.<CR>
.<CR>
Passau
```

3-5.

```
.<CR>  
/Wiesb/c<CR>  
Berlin<CR>  
.<CR>  
.<CR>  
Berlin
```

Als Zeichenkette für das Suchmuster muß nicht das ganze Wort bzw. die ganze Zeile angegeben werden; es kommt nur darauf an, daß die Zeichenkette eindeutig ist.

Übung 4

4-1.

```
v/Ort zum/s/Ort/Platz<CR>  
Mir liegt ein Platz wie  
Hamburg  
Als wäre man irgendwo in  
Passau  
Berlin  
Ich hab mein Herz verloren in  
Heidelberg  
Ich habe $$ verloren in Hotels in  
München
```

Die Zeile

Ein Ort zum Wohlfühlen ist

wurde nicht angezeigt, da sie durch das Kommando v NICHT adressiert wurde.

4-2.

```
.<CR>
München
s?München?Frankfurt<CR>
Frankfurt
```

4-3.

```
?verloren?s??gefunden<CR>
Ich habe $$ gefunden in
```

4-4.

```
/irgendwo?s??IRGENDWO<CR>  
?  
/irgendwo/s//IRGENDWO<CR>  
Als wäre man NIRGENDWO in
```

Begrenzungszeichen wie / und ? können in einer Kommandozeile nicht kombiniert werden.

Das Ersetzungskommando mit Zeile 9 führt zu folgendem Ergebnis:

Ich habe \$\$ gefunden inviele \$

Er führte nicht zum beabsichtigten Ergebnis, weil das Zeichen \$ im Editor ed ein Sonderzeichen ist.

5

Übung 5

5-1.

```
$  
ed dateil  
<CR>  
? dateil  
u<CR>  
A Informatik<CR>  
D Joggen<CR>  
C Tennis<CR>  
<CR>  
l,$s/[^AB]/A/<CR>  
l,$p<CR>  
A Informatik  
A Joggen  
A Tennis  
u<CR>
```

```
l,$s/[^AB]/A<CR>  
l,$p<CR>  
A Informatik  
A Joggen  
A Tennis
```


5-2.

```
2i<CR>
Das sind nicht gerade meine Fächer.<CR>
.<CR>
1,$p<CR>
A Informatik
Das sind nicht gerade meine Fächer.
A Tennis
A Joggen
/^[^A]<CR>
Das sind nicht gerade meine Fächer.
?^[F]<CR>
Das sind nicht gerade meine Fächer.
```

5-3.

```
1,$p<CR>
Ich liebe Geld
Ich brauche Geld
Das Finanzamt will mein Geld
g/^i/s/L*G /Es ist mein G<CR>
Es ist mein Geld
Es ist mein Geld
```

```
/s/Geld/Gold<CR>
Es ist mein Gold
2,$s//%<CR>
Das Finanzamt will mein Gold
```

5-4.

```
s/10202/&0<CR>
10202031020
```

5-5.

```
a<CR>
*.\&%^*<CR>
.<CR>
s/*a<CR>
a.\&%^*
s/*b<CR>
a.\&%^b
```

Da keine Zeichen vorausgingen, wurde * durch dasselbe Zeichen ersetzt.

```
s/\./e<CR>
a c \ & % ^ b
s/\\d<CR>
a c d & % ^ b
s/&/e<CR>
a c d e % ^ b
s/%/t<CR>
a c d e f ^ b
```

Die Zeichen & und % sind nur im Ersetzungstext Sonderzeichen.

```
s/\^/g<CR>
a c d e f g b
```

Übung 6

- 6-1. Wenn man Textzeilen mehrmals wiederholen will, ist es einfacher, diese Zeilen in einer Datei abzuspeichern und die Datei an den gewünschten Stellen im Text einzulesen.

Sollen diese Zeilen in eine andere Datei kopiert werden, müssen sie in eine Datei kopiert und aus dieser in den Puffer eingelesen werden, wenn die Datei, in welche die Zeilen kopiert werden sollen, im Puffer steht.

```
ed übung<CR>
725
10,17 w temp<CR>
210
q<CR>
ed übung6<CR>
305
7r temp<CR>
210
```

Die Datei **temp** ist ein hier willkürlich gewählter Name.

6-2.

```
33,46m3<CR>
```

6-3.

```
.= <CR>  
10  
13p<CR>  
Dies ist Zeile 13.  
j<CR>  
-p<CR>  
Dies ist Zeile 13.und Zeile 14.
```

Es ist zu beachten, daß .= die aktuelle Zeile ausgibt.

Übung 7

7-1.

```
$ ed neudate1 <CR>
? neudate1 r
aktuell1 <CR>
aktuell1
a <CR>
Dies ist eine Textzeile <CR>
Wird sie in neudate1 stehen <CR>
oder in aktuell1? <CR>
.<CR>
w <CR>
68
q <CR>
$ !s <CR>
bin
aktuell1
```

7-2.

```
ed date1 <CR>
? date1
a <CR>
Hier füge ich Text in die Datei ein. <CR>
Wird er in ed.hup enthalten sein? <CR>
.<CR>
```

Das Terminal ausschalten.

Erneut anmelden.

```
ed ed.hup<CR>
69
f date1<CR>
date1
l,$p<CR>
Hier füge ich Text in die Datei ein.
Wird er in ed.hup enthalten sein?
```

7-3.

```
$ ed date1<CR>
69
! mail meinname<CR>
Nach dem Editieren ist<CR>
Post für Dich da!<CR>
.<CR>
!
```

Kapitel 6: ANLEITUNG ZUM BILDSCHIRMEDI- TOR (vi)

Einführung	6-1
Zu dieser Anleitung	6-3
Vorbereitungen	6-4
Definieren der Terminalkonfiguration	6-4
Ändern der Umgebung	6-5
Konfigurieren des automatischen Zeilenumbruchs	6-6
Anlegen einer Datei	6-7
Erstellen von Text: Der Modus Anhängen	6-8
Verlassen des Modus Anhängen	6-9
Editieren von Text: Der Kommando-Modus	6-10
Bewegen des Cursors	6-10
Cursor-Bewegungen nach rechts oder links	6-12
Löschen von Text	6-15
Hinzufügen von Text	6-17
Verlassen von vi	6-18
Übung 1	6-21
Steuern des Cursors auf dem Bildschirm	6-22
Positionieren des Cursors auf ein Zeichen	6-22
Den Cursor zum Zeilenanfang oder -ende bewegen	6-23

Ein Zeichen in einer Zeile suchen	6-25
Bewegen des Cursors in eine andere Zeilen	6-27
Das Cursor-Kommando Minus	6-27
Das Cursor-Kommando Plus	6-27
Wortweise springen	6-28
Satzweise springen	6-32
Absatzweise springen	6-34
Bewegen des Cursors in der aktuellen Anzeige	6-35
Positionieren des Cursors in nicht angezeigtem Text	6-41
Blättern im Text	6-41
Das Kommando CONTROL-f	6-41
Das Kommando CONTROL-d	6-42
Das Kommando CONTROL-b	6-42
Das Kommando CONTROL-u	6-43
Suchen einer bestimmten Zeile	6-44
Zeilennummern	6-44
Suche nach einem Zeichenmuster: Die Kommandos / und ?	6-47
Übung 2	6-54
Eingeben von Text	6-56
Anhängen von Text	6-56
Einfügen von Text	6-56
Anlegen einer Zeile für neuen Text	6-58
Übung 3	6-61
Löschen von Text	6-62

Löschen von Text im Textmodus	6-62
Rückgängigmachen des letzten Kommandos	6-64
Löschen von Kommandos im Kommando-Modus	6-64
Löschen von Wörtern	6-65
Löschen von Absätzen	6-66
Löschen von Zeilen	6-67
Löschen von Text ab der Cursor-Position	6-67
Übung 4	6-69
Ändern von Text	6-70
Ersetzen von Text	6-70
Austauschen von Text	6-71
Ändern von Text	6-72
Elektronisches Verschieben von Text	6-78
Verschieben von Text	6-78
Korrektur vertauschter Zeichen	6-78
Kopieren von Text	6-79
Kopieren oder Verschieben von Text mit Registern	6-81
Übung 5	6-83
Spezielle Kommandos	6-84
Wiederholen des letzten Kommandos	6-84
Zusammenfügen von zwei Zeilen	6-84
Löschen und Wiederaufbauen des Bildschirms	6-85
Umwandlung von Klein- in Großschreibung und umgekehrt	6-85

Verwendung von Kommandos des Zeileneditors in vi	6-87
Zeitweilige Rückkehr zur Shell: Die Kommandos :sh und :!	6-87
Eingabe in eine neue Datei: Das Kommando :w	6-88
Suchen der Zeilennummer	6-89
Den restlichen Pufferinhalt löschen	6-90
Aufnehmen einer Datei in den Puffer	6-90
Globale Änderungen	6-91
Verlassen von vi	6-94
Sonderoptionen für vi	6-97
Wiederherstellen einer durch Unterbrechung verlorengegangenen Datei	6-97
Editieren mehrerer Dateien	6-97
Anzeigen einer Datei	6-98
Übung 6	6-100
Lösungen zu den Übungen	6-101
Übung 1	6-101
Übung 2	6-102
Übung 3	6-104
Übung 4	6-105
Übung 5	6-106
Übung 6	6-107

Einführung

Dieses Kapitel enthält die Anleitung zum Bildschirmeditor vi (Abkürzung für "Visual Editor"). Der Editor vi ist ein leistungsstarkes, anspruchsvolles Werkzeug zum Anlegen und Editieren von Dateien. Mit diesem Editor kann der Text von Dateien auf dem Bildschirm angezeigt werden. Mit einigen einfachen Kommandos können Änderungen am Text vorgenommen werden, die dann sofort auf dem Bildschirm erscheinen.

Die Anzahl der Zeilen die vi anzeigen kann ist nicht begrenzt. Der Cursor kann zu jedem Punkt auf dem Bildschirm oder in der Datei bewegt werden (durch Angeben von Positionen wie Beginn oder Ende eines Wortes, einer Zeile, eines Satzes, eines Abschnitts oder einer Datei), und dort kann Text eingegeben, geändert oder gelöscht werden. Es können auch bestimmte Kommandos des Zeileneditors verwendet werden, beispielsweise die wichtigen globalen Kommandos, mit denen mehrmals vorkommende Zeichenfolgen auf einmal geändert werden können. Man kann im Text vorwärts oder rückwärts blättern und dabei die Zeilen unter oder über dem aktuellen Fenster anzeigen, wie in Abbildung 6-1 dargestellt.

NOTE

Nicht bei allen Terminals ist dieses Blättern möglich; ob die Blätter-Funktion von vi eingesetzt werden kann, hängt vom jeweiligen Terminaltyp ab.

6

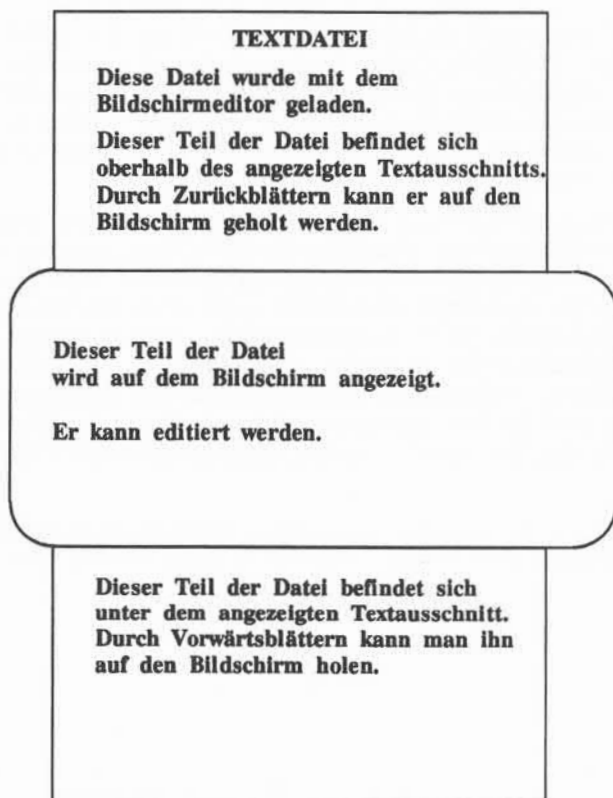


Abbildung 6-1: Textausschnitt einer mit vi geladenen Datei

In vi stehen über 100 Kommandos zur Verfügung. Dieses Kapitel behandelt die Grundkommandos, mit denen ein einfaches aber effizientes Arbeiten mit vi möglich ist. Insbesondere werden folgende Aufgaben erläutert:

- Einrichten des Terminals für vi
- Starten von vi, Eingeben von Text, Korrektur von Fehlern, Sichern des Textes in einer Datei, Beenden des Programms
- Verschieben von Text innerhalb einer Datei
- Kopieren von Text innerhalb einer Datei
- Kommandowiederholungen und Verwendung spezieller Kommandos
- Vorübergehende Rückkehr zur Shell, um Shell-Kommandos auszuführen
- Verwendung von Kommandos des Zeileneditors in vi
- Gleichzeitiges Editieren mehrerer Dateien in einer Sitzung
- Wiederherstellen einer Datei, die durch eine Unterbrechung der Editiersitzung verloren ging
- Ändern der Shell-Umgebung, um die Terminal-Konfiguration und den automatischen Zeilenumbruch zu definieren.

Zu dieser Anleitung

Beim Lesen dieser Anweisung sind die im Vorwort beschriebenen Notationskonventionen zu beachten. In den Bildschirmen in diesem Kapitel dienen Pfeile dazu, die Cursor-Position anzugeben.

Alle Kommandos, die in einem Abschnitt besprochen wurden, werden am Ende des Abschnitts noch einmal zusammengefaßt. Anhang D enthält eine nach Themen geordnete Zusammenfassung aller vi-Kommandos. Einigen Abschnitte enthalten am Schluß Übungen für den Benutzer. Die Lösungen zu diesen Übungen finden Sie am Ende dieses Kapitels. Um die Kommandos von vi möglichst schnell zu lernen, sollten Sie gleich beim Lesen der Anleitung die Beispiele nachvollziehen und die Übungen bearbeiten. Daher sollten Sie sich, bevor Sie mit der Lektüre dieses Kapitels beginnen, an das System anmelden.

Vorbereitungen

Das Betriebssystem UNIX ist flexibel; es kann auf zahlreichen Rechnertypen installiert werden, und die verschiedensten Terminals können darauf zugreifen. Aber gerade weil das System so viele Möglichkeiten zuläßt, muß angegeben werden, welche Hardware in einer bestimmten Situation verwendet wird.

Zudem bietet UNIX verschiedene Zusatzoptionen für das Terminal, die in die Sitzungsroutine aufgenommen werden können. Aus den gewählten Optionen und den Hardware-Spezifikationen ergibt sich die Anmeldeumgebung. Nachdem die Anmeldeumgebung definiert ist, implementiert die Shell diese Spezifikationen und Optionen bei jedem Anmeldevorgang automatisch.

In diesem Abschnitt werden zwei Aufgaben bei der Definition der Anmeldeumgebung beschrieben: Das Definieren der Terminalkonfiguration, die zum ordnungsgemäßen Arbeiten mit `vi` erforderlich ist, und die Definition des automatischen Zeilenumbruchs, der optional gesetzt werden kann.

Definieren der Terminalkonfiguration

Vor dem Starten von `vi` muß die Terminalkonfiguration definiert werden. Dies bedeutet, daß UNIX mitgeteilt wird, welcher Typ von Terminal verwendet wird. Dies ist notwendig, weil `vi` auf unterschiedlichen Terminals unterschiedlich arbeitet.

Für jeden Terminaltyp existieren mehrere Codenamen, die UNIX erkennt. In Anhang F, "Konfiguration des Terminals," sind die gültigen Terminalnamen aufgeführt. Bei vielen Installationen sind jedoch auch andere als diese Standard-Unix-Namen zulässig. Erkundigen Sie sich bei Ihrem Systemverwalter nach der neuesten Liste der zulässigen Terminalnamen.

Zum Definieren der Terminalkonfiguration folgendes eingeben:

```
TERM=terminal_name<CR>  
export TERM<CR>
```

In der ersten Zeile wird ein Wert (der Terminaltyp) der Variablen `TERM` zugewiesen. Mit der zweiten Zeile wird dieser Wert exportiert; er wird allen UNIX-Programmen zur Verfügung gestellt, deren Ausführung vom Terminaltyp abhängt.

Ist das verwendete Terminal beispielsweise ein BA47/BA80, erscheint das Kommando folgendermaßen auf dem Bildschirm:

```
$ TERM = dap4x <CR>  
$ export TERM <CR>
```

Es darf nur der eigene Terminaltyp angegeben werden. Andere Eingaben könnten zu einem Fehler führen, und der Benutzer muß sich abmelden, die Sitzung abbuchen oder den Systemverwalter zu Rate ziehen, um die Anmeldeumgebung wiederherzustellen.

Ändern der Umgebung

Soll vi regelmäßig benutzt werden, muß die Anmeldeumgebung so geändert werden, daß das Terminal nicht bei jedem Anmeldevorgang konfiguriert werden muß. Die Anmeldeumgebung wird von einer Datei im Home-Verzeichnis namens `.profile` gesteuert. Diese Datei ist im Dateisystem noch nicht vorhanden, sondern muß vom Benutzer angelegt werden. Näheres hierzu enthält Kapitel 7.

Werden die Werte für die Terminalkonfiguration in der Datei `.profile` definiert, wird das Terminal bei jeder Anmeldung automatisch konfiguriert. Um das Terminal automatisch zu konfigurieren, müssen die Zuweisung `TERM`, das Kommando `export` und das Kommando `tput` in die Datei `.profile` aufgenommen werden. (Umfassende Anweisungen enthält Kapitel 7.)

Konfigurieren des automatischen Zeilenumbruchs

NOTE

Um den automatischen Zeilenumbruch zu definieren, muß man wissen, wie eine Datei angelegt wird. Kennen Sie bereits einen anderen Texteditor, z. B. `ed`, können Sie die Anweisungen in diesem Abschnitt problemlos ausführen. Haben Sie noch nie mit einem Editor gearbeitet, möchten aber den automatischen Zeilenumbruch definieren, sollten Sie diesen Abschnitt vorläufig überspringen und diese Anweisungen erst ausführen, wenn Sie dieses Kapitel durchgearbeitet und verstanden haben.

Soll der Zeilenumbruch automatisch vorgenommen werden, legt man im Home-Verzeichnis eine Datei namens `.exrc` an. In der Datei `.exrc` können Optionen zum Steuern der Editierumgebung von `vi` gespeichert werden.

Zum Anlegen der Datei `.exrc` wird ein Editor mit diesem Dateinamen aufgerufen. Dann geben Sie die Spezifikation für die Option automatischer Zeilenumbruch (automatische Zeilenschaltung) in eine Zeile ein. Diese Spezifikation hat folgendes Format:

```
wm = n <CR>
```

`n` gibt an, beim wievielten Zeichen vom rechten Bildschirmrand aus die automatische Zeilenschaltung erfolgen soll. Soll der Zeilenumbruch beispielsweise zwanzig Zeichen vom rechten Bildschirmrand erfolgen, geben Sie folgendes ein:

```
wm = 20 <CR>
```

Dann speichern Sie den Inhalt des Puffers in der Datei und verlassen den Editor (siehe Abschnitt "Puffer für die Textbearbeitung" in Kapitel 4). Wenn Sie sich das nächste Mal anmelden, aktiviert diese Datei den automatischen Zeilenumbruch.

Mit dem folgenden Kommando können Sie in `vi` die Werte für das Terminal und den automatischen Zeilenumbruch überprüfen:

```
:set <CR>
```

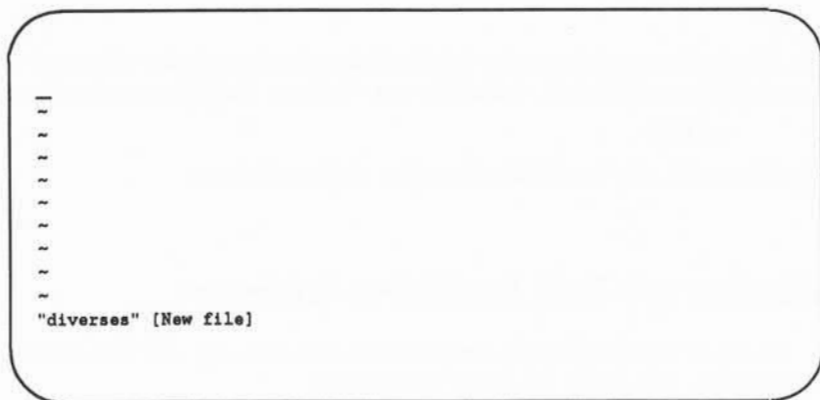
`vi` zeigt den Terminaltyp, die für den automatischen Zeilenumbruch gesetzte Zeichenposition, sowie alle sonstigen Werte an. Zum Anlegen oder Ändern der Zeilenumbruchsfunktion kann auch das Kommando `:set` verwendet werden. Versuchen Sie es einmal.

Anlegen einer Datei

Zunächst rufen Sie den Editor auf: `vi` und den Namen der Datei eingeben, die erstellt oder editiert werden soll.

`vi dateiname <CR>`

So soll beispielsweise eine Datei namens `diverses` angelegt werden. Wird das Kommando `vi` zusammen mit dem Dateinamen `diverses` angegeben, löscht `vi` den Bildschirm und zeigt ein Fenster an, in dem Text eingegeben und editiert werden kann.



Der Unterstrich ~ in der obersten Zeile zeigt die Cursor-Position, an der ein Kommando eingegeben werden kann. (Bei Bildschirmterminals kann der Cursor auch ein blinkender Unterstrich oder ein Farbblock in Umkehranzeige sein.) Jede weitere Zeile ist mit einer Tilde ~, das Symbol für Leerzeile, markiert.

Wurde vor dem Starten von `vi` vergessen, die Konfiguration für das Terminal zu definieren, oder wurde sie falsch definiert, erscheint statt dessen eine Fehlermeldung.

```
$ vi kram<CR>
terminal_name: unknown terminal type

[Using open mode]
"diverses" [New file]
```

Die Terminalkonfiguration kann nicht im Editor definiert werden; dies muß auf der Shell-Ebene geschehen. Geben Sie zum Verlassen des Editors folgendes ein:

```
:q<CR>
```

Nun können Sie die Terminalkonfiguration richtig definieren.

Erstellen von Text: Der Modus Anhängen

Nach dem Starten von **vi** ist der Kommando-Modus aktiv, und **vi** wartet auf Kommandos. Nun wollen wir einen Text erstellen.

- Mit der Taste **A** (**<a>**) in den **vi**-Modus "Anhängen" wechseln. (Nicht die RETURN-Taste drücken.) Nun kann Text in die Datei eingegeben werden. Das **A** wird nicht auf dem Bildschirm angezeigt.
- Text eingeben.
- Mit der RETURN-Taste erzeugt man eine neue Zeile.

Wurde in der Datei **.exrc** der automatische Zeilenumbruch definiert, wird bei jeder automatischen Zeilenschaltung eine neue Zeile erzeugt (siehe Abschnitt "Konfigurieren des automatischen Zeilenumbruchs").

Verlassen des Modus Anhängen

Ist die Texteingabe abgeschlossen, die Taste ESCAPE drücken; damit kehrt man vom Modus "Anhängen" zum Kommando-Modus zurück. Nun kann jeder bereits erstellte Text bearbeitet oder aus dem Puffer in eine Datei gespeichert werden.

```
<a>Text im Editor vi<CR>  
eingeben und zum<CR>  
Kommando-Modus<CR>  
zurückkehren.<ESC>
```

Ertönt beim Drücken der Escape-Taste ein Signalton, ist der Kommando-Modus bereits wieder aktiv. Der Text in der Datei ist davon nicht betroffen, auch wenn die Escape-Taste mehrmals gedrückt wird.

Editieren von Text: Der Kommando-Modus

Zum Editieren einer vorhandenen Datei gehört das Hinzufügen, Ändern und Löschen von Text. Zuvor muß der Benutzer jedoch imstande sein, sich den Teil der Datei anzeigen zu lassen, der bearbeitet werden soll. vi bietet eine Reihe von Kommandos, mit denen der Cursor seitenweise, zeilenweise oder zwischen bestimmten Punkten in einer Zeile bewegt werden kann. Diese Kommandos sowie die Kommandos zum Löschen und Hinzufügen von Text werden in diesem Abschnitt erläutert.

Bewegen des Cursors

Um den Text bearbeiten zu können, muß man den Cursor zu dem Punkt auf dem Bildschirm bewegen, an dem mit der Änderung begonnen werden soll. Hierzu dienen vier Tasten, die auf der Tastatur nebeneinander angeordnet sind: h, j, k und l.

<h> bewegt den Cursor um ein Zeichen nach links

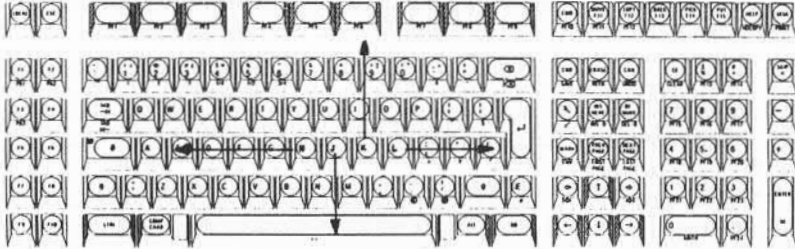
<j> bewegt den Cursor um eine Zeile nach unten

<k> bewegt den Cursor um eine Zeile nach oben

<l> bewegt den Cursor um ein Zeichen nach rechts

Bei den Kommandos <j> und <k> bleibt der Cursor in derselben Spalte. Steht der Cursor beispielsweise auf dem siebten Zeichen von links, springt er bei dem Kommando <j> oder <k> auf das siebte Zeichen der neuen Zeile. Steht an dieser Stelle der neuen Zeile kein Zeichen mehr, springt der Cursor zum letzten Zeichen.

Vielen Benutzern von vi ist es eine Hilfe, wenn sie diese vier Tasten mit Pfeilen in die jeweilige Richtung markieren.



NOTE

Einige Terminals haben spezielle Cursor-Tasten, die mit Pfeilen gekennzeichnet sind. Sie haben die gleiche Wirkung wie die Kommandos `<h>`, `<j>`, `<k>` und `<l>`.

Beobachten Sie die Cursor-Bewegungen beim Drücken der Tasten `<h>`, `<j>`, `<k>` und `<l>`. Soll der Cursor um mehrere Positionen bewegt werden, kann man die Cursor-Taste mehrmals drücken, oder man kann die gewünschte Anzahl von Spalten oder Zeilen vor dem Kommando angeben. Um den Cursor beispielsweise zwei Spalten nach rechts zu bewegen, kann man zweimal `<l>` drücken oder `<2l>` eingeben. Um den Cursor vier Zeilen nach oben zu bewegen, viermal `<k>` drücken oder `<4k>` eingeben. Kann der Cursor nicht weiter in die gewünschte Richtung bewegt werden, ertönt ein akustisches Signal.

Üben Sie nun die Cursor-Kommandos **j** und **k**. Zuerst den Cursor um sieben Zeilen nach oben bewegen. Eingabe:

<7k>

Der Cursor bewegt sich von der aktuellen Zeile um sieben Zeilen nach oben. Sind oberhalb der aktuellen Zeile keine sieben Zeilen vorhanden, bleibt der Cursor an seiner ursprünglichen Position, und es ertönt ein akustisches Signal.

Nun soll der Cursor fünfunddreißig Zeilen nach unten bewegt werden. Eingabe:

<35j>

vi löscht den Bildschirm und baut ihn neu auf. Der Cursor steht in der fünfunddreißigsten Zeile unter der Ausgangszeile, in der Mitte des neuen Textausschnitts auf dem Bildschirm. Sind unterhalb der Ausgangszeile weniger als fünfunddreißig Zeilen vorhanden, ertönt das akustische Signal, und der Cursor bewegt sich nicht. Beobachten Sie bei der Eingabe des nächsten Kommandos den Bildschirm.

<35k>

Wie die meisten **vi**-Kommandos erscheinen auch die Cursor-Kommandos **<h>**, **<j>**, **<k>** und **<l>** nicht auf dem Bildschirm, wenn sie eingegeben werden. Text darf nur dann auf dem Bildschirm erscheinen, wenn er im Modus "Anhängen" in die Datei eingefügt werden soll. Erscheinen die Buchstaben der Cursor-Kommandos auf dem Bildschirm, ist der Modus "Anhängen" noch aktiv. Die Escape-Taste drücken, zum Kommando-Modus zurückkehren und die Kommandos erneut eingeben.

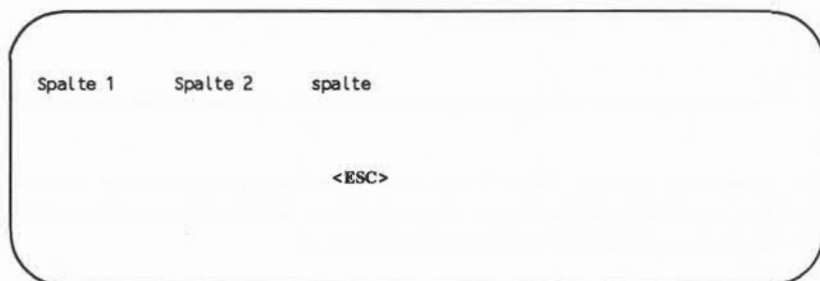
Cursor-Bewegungen nach rechts oder links

Neben den Cursor-Tasten **<h>** und **<l>** dienen auch die Leertaste und die Taste **BACKSPACE** dazu, den Cursor nach rechts oder links auf ein Zeichen in der aktuellen Zeile zu stellen.

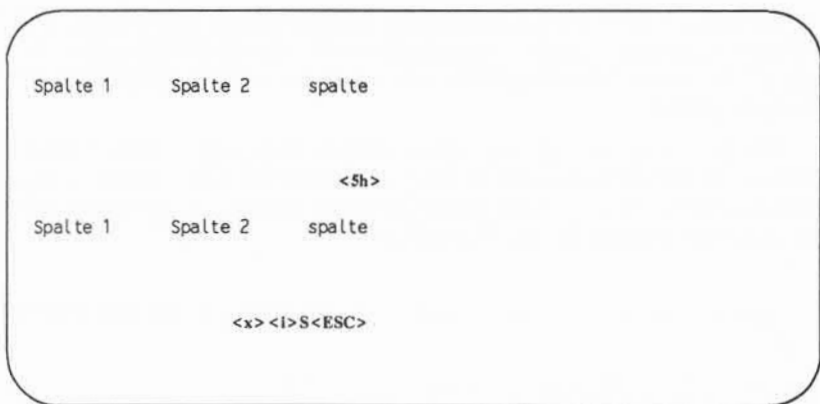
<Leertaste>	bewegt den Cursor um ein Zeichen nach rechts
<nLeertaste>	bewegt den Cursor um <i>n</i> Zeichen nach rechts
<BACKSPACE>	bewegt den Cursor um ein Zeichen nach links
<nBACKSPACE>	bewegt den Cursor um <i>n</i> Zeichen nach links

Auch hier kann man eine Zahl mit der Kommandotaste kombinieren, so daß sich der Cursor um die angegebene Anzahl von Zeichen nach links oder rechts bewegt. In dem nachstehenden Beispiel wird die Cursor-Bewegung durch die Pfeile dargestellt.

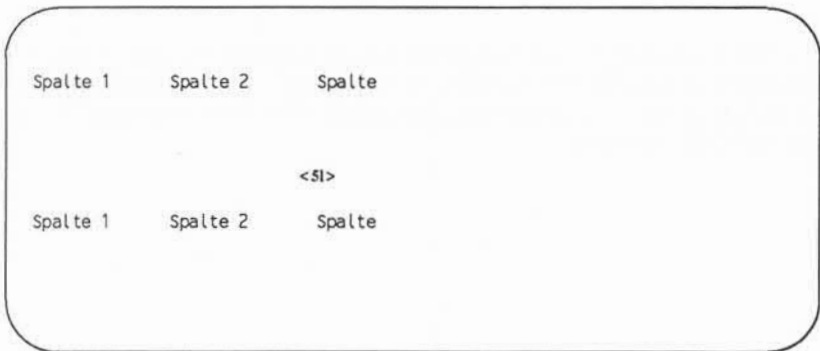
Um den Cursor schnell nach rechts oder links zu bewegen, wird vor dem Kommando eine Zahl eingegeben. Angenommen, auf dem Bildschirm sollen vier Spalten angelegt werden. Nach dem Eingeben der ersten drei Spaltenüberschriften entdecken Sie jedoch einen Tippfehler:



Der Fehler soll korrigiert werden, bevor weitere Eingaben erfolgen. Vom Modus "Anhängen" mit der Escape-Taste in den Kommando-Modus wechseln; der Cursor springt auf das e. Dann mit dem Kommando <h> den Cursor fünf Positionen nach links bewegen.



Das s mit <x> löschen. Dann in den Einfügemodus (<i>) wechseln und ein S eingeben, anschließend die Escape-Taste drücken. Mit dem Cursor-Kommando <l> an die ursprüngliche Position zurückkehren.



Sicher haben Sie bereits festgestellt, daß man den Cursor auch mit der Leertaste und der Taste BACKSPACE innerhalb einer Zeile bewegen kann.

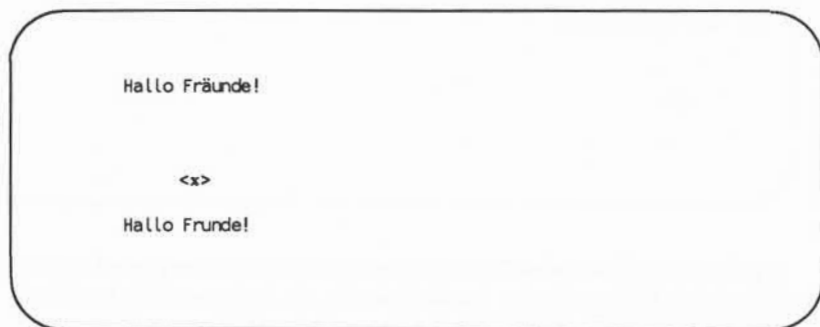
- <Leertaste>** bewegt den Cursor um ein Zeichen nach rechts
- <nLeertaste>** bewegt den Cursor um *n* Zeichen nach rechts
- <BACKSPACE>** bewegt den Cursor um ein Zeichen nach links
- <nBACKSPACE>** bewegt den Cursor um *n* Zeichen nach links

Auch hier kann man, anstatt die Taste mehrmals zu drücken, eine Zahl eingeben, bevor man die entsprechende Taste drückt. Der Cursor bewegt sich um die angegebene Zahl von Zeichen nach links oder rechts.

Löschen von Text

Soll ein Zeichen gelöscht werden, den Cursor auf dieses Zeichen stellen und die Taste **<x>** gedrückt. Beobachten Sie dabei den Bildschirm; das Zeichen verschwindet, und die Zeile wird entsprechend angepaßt. Um drei Zeichen zu löschen, die Taste **<x>** dreimal drücken. Im folgenden Beispiel zeigen die Pfeile unter den Buchstaben die Cursor-Position an.

- <x>** ein Zeichen löschen
- <nX>** *n* Zeichen löschen, wobei *n* die Anzahl der zu löschenden Zeichen angibt



Nun wollen wir vor dem Betätigen der Taste `<x>` eine Zahl für mehrere zu löschende Zeichen eingegeben. So soll beispielsweise das Wort `tiefsten` gelöscht werden, wenn es in dem folgenden Bildschirmtext zum zweitenmal erscheint. Den Cursor auf den ersten Buchstaben der zu löschenden Zeichenfolge stellen und neun Zeichen (für die acht Buchstaben von `tiefsten` und ein Leerzeichen) löschen.

Morgen wird das Ungeheuer von Loch Ness
aus den tiefsten, dunkelsten tiefsten Tiefen
des Sees emporsteigen.

`<9x>`

Morgen wird das Ungeheuer von Loch Ness
aus den tiefsten, dunkelsten Tiefen
des Sees emporsteigen.

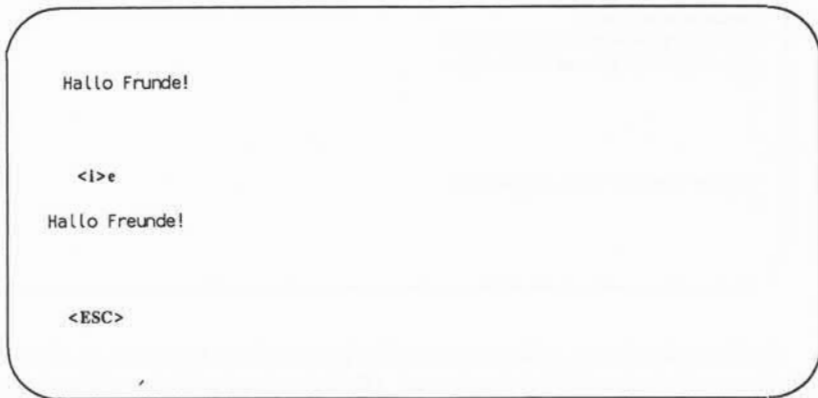
`<9x>`

`vi` paßt den verbleibenden Text so an, daß an der Stelle der gelöschten Zeichenfolge keine Lücke entsteht. Handelt es sich, wie in diesem Fall, bei der zu löschenden Zeichenfolge um ein Wort, kann auch das `vi`-Kommando zum Löschen eines Wortes verwendet werden. Dieses Kommando wird in dem Abschnitt "Wortweises Positionieren des Cursors" erläutert.

Hinzufügen von Text

Für das Hinzufügen von Text stehen zwei Kommandos zur Verfügung: die Kommandos Einfügen (<i>) und Anhängen (<a>). Soll Text mit dem Einfügebefehl an einer Stelle der Datei eingefügt werden, die auf dem Bildschirm sichtbar ist, den Cursor mit <h>, <j>, <k> und <l> zu dieser Stelle bewegen. Dann <i> drücken und den Text eingeben. Während der Eingabe erscheint der Text auf dem Bildschirm links von dem Zeichen, auf das der Cursor gestellt wurde.

Dieses und alle anderen Zeichen rechts vom Cursor werden nach rechts verschoben, um für den neuen Text Platz zu machen. Der Editor vi akzeptiert die eingegebenen Zeichen, bis die Escape-Taste gedrückt wird. Ggf. werden die ursprünglichen Zeichen auch in die nächste Zeile verschoben.



Das Kommando Anhängen funktioniert genauso. Der einzige Unterschied besteht darin, daß dabei der neue Text rechts von dem Zeichen, auf dem der Cursor steht, eingefügt wird.

An anderer Stelle in dieser Anleitung wird erklärt, wie der Cursor auf dem Bildschirm bewegt wird bzw. wie durch die Datei geblättert wird, um Zeichen, Wörter oder Zeilen einzufügen oder zu löschen.

Verlassen von vi

Ist die Eingabe beendet, muß der Benutzer den Pufferinhalt in einer Datei speichern und zur Shell zurückkehren. Dazu die Taste SHIFT gedrückt halten und zweimal die Taste Z drücken (<ZZ>). Im Editor ist der Dateiname gespeichert, der zu Beginn der Editiersitzung zusammen mit dem Kommando vi angegeben wurde, und nun wird der Text im Puffer in diese Datei geschrieben. In einer Meldung unten auf dem Bildschirm werden der Dateiname sowie die Zeilen- und Zeichenzahl der Datei angezeigt. Anschließend gibt die Shell ein Bereit-Zeichen aus.

```
<a>Dies ist eine Testdatei.<CR>
Ich schreibe Text in einen<CR>
temporären Puffer, und<CR>
jetzt ist er fertig.<CR>
Ich will ihn in dieser Datei speichern<CR>
und zur Shell zurückkehren.<ESC><ZZ>
~
~
~
~
~
"diverses" [New file] 7 lines, 178 characters
$
```

Auch mit den Kommandos :w und :q des Zeileneditors kann man in die Datei schreiben und sie anschließend verlassen. (Kommandos des Zeileneditors beginnen mit einem Doppelpunkt und erscheinen in der untersten Bildschirmzeile.) Mit dem Kommando :w wird der Inhalt des Puffers in einer Datei gespeichert. Mit dem Kommando :q verläßt man den Editor und kehrt zur Shell zurück. Diese Kommandos können getrennt eingegeben oder zu dem Kommando :wq zusammengefaßt werden, was die einfachere Methode ist.

<a>Dies ist eine Testdatei.<CR>
Ich schreibe Text in einen<CR>
temporären Puffer, und<CR>
jetzt ist er fertig.<CR>
Ich will ihn in dieser Datei speichern<CR>
und zur Shell zurückkehren.<ESC><ZZ>
~
~
~
~
~
~
~
~<CR>

In Abbildung 6-2 sind die Grundkommandos zusammengefaßt, die Sie zum Starten und Benutzen von vi benötigen.

Kommando	Funktion
TERM = <i>terminal_name</i> export TERM	Definieren der Terminalkonfiguration
tput init	Initialisieren des Terminals wie in <i>terminal_name</i> angegeben
vi dateiname	Starten des Editors vi zum Bearbeiten der Datei <i>dateiname</i>
<a>	Text nach dem Cursor einfügen
<h>	Cursor um ein Zeichen nach links bewegen
<j>	Cursor eine Zeile nach unten bewegen
<k>	Cursor eine Zeile nach oben bewegen
<l>	Cursor um ein Zeichen nach rechts bewegen
<x>	Zeichen löschen
<CR>	Zeilenschaltung
<ESC>	Modus "Anhängen" verlassen, Rückkehr zum Kommando-Modus von vi
:w	Text in einer Datei abspeichern
:q	vi verlassen
:wq	Text in Datei speichern und vi verlassen
<ZZ>	Text in Datei speichern und vi verlassen

Abbildung 6-2: Zusammenfassung der Kommandos für den Editor vi

Übung 1

Die Lösungen zu den Übungen sind am Ende dieses Kapitels zusammengefaßt. Bitte beachten Sie, daß viele Aufgaben in vi auf verschiedene Arten gelöst werden können. Alle Methoden, die zu dem gewünschten Ergebnis führen, sind richtig.

Beobachten Sie beim Eingeben von Kommandos in den folgenden Übungen, wie sich die Anzeige verändert oder sich der Cursor bewegt.

- 1-1. Anmelden, sofern dies noch nicht geschehen ist. Das Terminal konfigurieren.
- 1-2. vi aufrufen und die folgenden fünf Textzeilen in eine neue Datei namens **probl** eingeben.

Das ist eine Übung!
Auf, ab,
links, rechts,
Terminal-Training
Bit für Bit

- 1-3. Den Cursor in die erste Zeile der Datei, auf das siebte Zeichen von rechts stellen. Während der Cursor nach oben bewegt wird, springt er in den kurzen Zeilen zum letzten Zeichen, in der nächsten längeren Zeile bewegt er sich jedoch nicht zum Zeilenende.
- 1-4. Das siebte und achte Zeichen von rechts löschen.
- 1-5. Den Cursor auf das letzte Zeichen in der letzten Textzeile stellen.
- 1-6. Die folgende Textzeile anfügen:

und Byte für Byte

- 1-7. Den Pufferinhalt in eine Datei abspeichern und vi verlassen.
- 1-8. vi erneut starten und zwei weitere Zeilen Text in die Datei **probl** einfügen.
Wie lautet die Meldung am unteren Bildschirmrand, wenn vi zum Bearbeiten von **probl** erneut aufgerufen wurde?

Steuern des Cursors auf dem Bildschirm

Bisher dienten zum Steuern des Cursors die Kommandos <h>, <j>, <k>, <l>, die Taste BACKSPACE und die Leertaste. Es gibt noch verschiedene andere Kommandos, mit denen der Cursor schnell auf dem Bildschirm bewegt werden kann. In diesem Abschnitt wird erklärt, wie man den Cursor

- zeichenweise in einer Zeile
- zeilenweise
- zwischen Textobjekten
 - wortweise
 - satzweise
 - abschnittsweise
- im Fenster

bewegt.

Es gibt auch Kommandos, mit denen der Cursor in Teilen des Editierpuffers von vi, die auf dem Bildschirm nicht sichtbar sind, positioniert werden kann. Diese Kommandos werden im nächsten Abschnitt, "Positionieren des Cursors in Text außerhalb der Bildschirmanzeige" erklärt.

Um den Anweisungen dieses Kapitels zu folgen, den Editor vi mit einer Datei aufrufen, die mindestens vierzig Zeilen lang ist. Ist keine solche Datei vorhanden, muß sie angelegt werden.

Hinweis: Um die hier beschriebenen Kommandos ausführen zu können, muß der Kommando-Modus von vi aktiv sein. Die Escape-Taste drücken, um sicherzugehen, daß nicht der Modus "Anhängen" aktiv ist.

Positionieren des Cursors auf ein Zeichen

Man kann den Cursor auf drei verschiedene Arten auf ein Zeichen in einer Zeile stellen, und zwar kann man:

- den Cursor nach links oder rechts zum Zeichen bewegen

- den Cursor auf das Zeichen am Ende oder Anfang der Zeile bewegen
- das Zeichen in einer Zeile suchen lassen

Die erste Methode wurde im Abschnitt "Cursor-Bewegungen nach rechts oder links" erläutert. In den folgenden Abschnitten werden die beiden anderen Methoden dargestellt.

Den Cursor zum Zeilenanfang oder -ende bewegen

Bei der zweite Methode zum Steuern des Cursors in einer Zeile wird der Cursor mit einem von drei Kommandos auf das erste oder letzte Zeichen einer Zeile gestellt.

- | | |
|------------------|---|
| <\$> | stellt den Cursor auf das letzte Zeichen einer Zeile |
| <0> (null) | stellt den Cursor auf das erste Zeichen einer Zeile |
| <^> (Zirkumflex) | stellt den Cursor auf das erste Zeichen einer Zeile, das kein Leerzeichen ist |

In den folgenden Beispielen wird gezeigt, welche Cursor-Bewegungen jedes dieser Kommandos bewirken.

Cursor an das Zeilenende stellen!

<\$>

Cursor an das Zeilenende stellen!



Cursor an den Zeilenanfang stellen!

<0>
Cursor an den Zeilenanfang stellen!

Cursor auf das erste Zeichen
der Zeile stellen,
das kein Leerzeichen ist!

<^>
Cursor auf das erste Zeichen
der Zeile stellen,
das kein Leerzeichen ist!

Ein Zeichen in einer Zeile suchen

Bei der dritten Methode, den Cursor innerhalb einer Zeile zu positionieren, wird ein bestimmtes Zeichen in der Zeile gesucht. Wird das Zeichen dort nicht gefunden, ertönt ein akustisches Signal, und der Cursor bewegt sich nicht. (Es gibt auch ein Kommando, mit dem man ein Suchmuster in der gesamten Datei suchen kann. Dies wird im nächsten Abschnitt erläutert.) Zum Suchen in einer Zeile stehen sechs Kommandos zur Verfügung: <f>, <F>, <t>, <T>, <;> und <, >. Nach jedem Kommando außer <;> und <, > muß ein zu suchendes Zeichen angegeben werden.

- <fx> Cursor nach rechts zu dem angegebenen Zeichen x bewegen.
- <Fx> Cursor nach links zu dem angegebenen Zeichen x bewegen.
- <tx> Cursor nach rechts zu dem Zeichen direkt vor dem angegebenen Zeichen x bewegen.
- <Tx> Cursor nach links zu dem Zeichen unmittelbar hinter dem angegebenen Zeichen x bewegen.
- <;> Die Suche nach dem im letzten Kommando angegebenen Zeichen in derselben Richtung fortsetzen. Mit ; wird das Suchzeichen gespeichert und seine nächste Entsprechung in der aktuellen Zeile gesucht.
- <, > Die Suche nach dem im letzten Kommando angegebenen Zeichen in der Gegenrichtung fortsetzen. Mit , wird das Suchzeichen gespeichert und seine vorherige Entsprechung in der aktuellen Zeile gesucht.

Auf dem folgenden Bildschirm beispielsweise sucht vi nach rechts in der aktuellen Zeile das erste Auftreten des Buchstaben A.

Buchstaben A in dieser Zeile in Schreibrichtung suchen.

<fA>

Buchstaben A in dieser Zeile in Schreibrichtung suchen.

Üben Sie nun die Suchkommandos in einer anderen Datei.

Bewegen des Cursors in eine andere Zeilen

Neben den bereits bekannten Kommandos <j> und <k> können auch die Kommandos <+>, <-> und <CR> verwendet werden, um den Cursor in eine andere Zeile zu bewegen.

Das Cursor-Kommando Minus

Mit dem Kommando <-> wird der Cursor um eine Zeile nach oben auf das erste Zeichen der Zeile, das kein Leerzeichen ist, gestellt. Um mehrere Zeilen auf einmal zu überspringen, wird vor dem Kommando <-> die Anzahl der Zeilen angegeben, um die der Cursor bewegt werden soll. Soll der Cursor z. B. um dreizehn Zeilen nach oben bewegt werden, folgendes eingeben:

<13->

Der Cursor geht dreizehn Zeilen nach oben. Liegt die gewünschte Zeile nicht mehr in dem angezeigten Textausschnitt, wird die Anzeige entsprechend verschoben, bis die Zeile zu sehen ist. Auf diese Art kann der Cursor rasch in einer Datei nach oben bewegt werden.

Versuchen Sie nun, den Cursor 100 Zeilen nach oben zu bewegen. Dazu folgendes eingeben:

<100->

Haben Sie die Anzeige beobachtet? Sind oberhalb der aktuellen Zeile weniger als 100 Zeilen vorhanden, meldet ein Signalton, daß ein Fehler vorliegt, und der Cursor bleibt auf der aktuellen Zeile stehen.

Das Cursor-Kommando Plus

Mit dem Cursor-Kommando Plus (<+>) oder dem Kommando <CR> wird der Cursor um eine Zeile nach unten bewegt. Sollen mehrere Zeilen übersprungen werden, die entsprechende Zahl vor dem Kommando <+> eingeben. Um den Cursor beispielsweise um neun Zeilen nach unten zu bewegen, folgendes eingeben:

<9+>

Der Cursor bewegt sich um neun Zeilen nach unten. Sind die entsprechenden Zeilen auf dem Bildschirm nicht mehr sichtbar, wird der Text weitergeblättert.

Versuchen Sie nun das gleiche mit der RETURN-Taste. Bewirkt diese Methode die gleichen Ergebnisse wie die +-Taste?

Wortweise springen

Für den Editor vi ist ein Wort eine Zeichenfolge, die aus Buchstaben, Zahlen und Unterstrichen bestehen kann. Es gibt sechs Kommandos, um den Cursor wortweise zu positionieren: <w>, , <e>, <W>, und <E>. Bei den Kommandos in Kleinbuchstaben (<w>, und <e>) gilt jedes Zeichen, das weder Buchstabe, Zahl noch Unterstrich ist, als Begrenzungszeichen für Anfang oder Ende eines Wortes. Satzzeichen vor oder nach einem Leerzeichen gelten als Wort. Anfang und Ende einer Zeile gelten ebenfalls als Begrenzungszeichen.

Die Kommandos in Großbuchstaben (<W>, , and <E>) behandeln Satzzeichen als Bestandteile des Wortes; Wörter werden nur durch Leerzeichen und Zeilenschaltungen begrenzt.

Hier eine Zusammenfassung der Kommandos für wortweise Cursor-Bewegungen.

- <w> Stellt den Cursor auf das erste Zeichen des nächsten Wortes. Um das gewünschte Wort zu erreichen, kann <w> beliebig oft gedrückt werden; man kann aber auch die entsprechende Zahl vor dem <w> angeben.
- <nw> Der Cursor springt *n* Wörter weiter auf den ersten Buchstaben des gewünschten Wortes. Das Zeilenende setzt dieses Kommando nicht außer Kraft; der Cursor springt in die nächste Zeile und zählt ab dem neuen Zeilenanfang weiter.

Mit dem Kommando <w> bewegt sich
der Cursor Wort für Wort durch die
Datei. Von DIESEM Wort

<6w>

sechs Wörter weiter zu DIESEM Wort springen.

Mit dem Kommando <w> bewegt sich
der Cursor Wort für Wort durch die
Datei. Von DIESEM Wort
sechs Wörter weiter zu DIESEM Wort springen.

6

- <W> Ignoriert alle Satzzeichen und bewegt den Cursor vorwärts zum Wort nach dem nächsten Leerzeichen.
- <e> Bewegt den Cursor in der Zeile zum letzten Zeichen des nächsten Wortes.

Ein Wort weiter zum Ende des
nächsten Wortes in dieser Zeile

<e>

Ein Wort weiter zum Ende des
nächsten Wortes in dieser Zeile

Zum Ende des dritten Wortes nach dem aktuellen Wort springen.

<3e>

Zum Ende des dritten Wortes nach dem aktuellen Wort springen.

- <E> Ignoriert alle Satzzeichen außer Leerzeichen, die die einzigen Begrenzungszeichen für Wörter sind.
- Bewegt den Cursor innerhalb der Zeile zurück zum ersten Zeichen des vorhergehenden Wortes.
- <nb> Bewegt den Cursor *n* Wörter zurück zum ersten Zeichen des *n*ten Wortes. Beim Kommando überspringt der Cursor auch den Zeilenanfang und springt vom Ende der vorhergehenden Zeile weiter zurück.
- Anwendung wie das Kommando , allerdings gelten als Wortbegrenzungszeichen nur Leerzeichen und New-Line-Zeichen. Satzzeichen gelten als Elemente eines Wortes.

Wort für Wort durch die Datei rückwärts gehen. Von hier aus vier Wörter zurückgehen.

<4b>

gehen. Von hier aus vier Wörter zurückgehen.

Satzweise springen

Der Editor vi erkennt auch Sätze. Für vi endet ein Satz mit ! , . oder ?. Erscheinen diese Begrenzungszeichen zwischen Text in einer Zeile, müssen danach zwei Leerzeichen folgen, damit vi sie erkennt. Es empfiehlt sich, mit der Konvention von vi, daß zwei Leerzeichen nach einem Punkt als Satzende gelten, zu arbeiten, denn es ist oft nützlich, wenn ein Satz als Einheit behandelt werden kann.

Um den Cursor satzweise durch die Datei zu bewegen, verwendet man die Kommandos <(> (Klammer auf) und <)> (Klammer zu).

- < (> Stellt den Cursor auf den Anfang des aktuellen Satzes.
- < n(> Stellt den Cursor auf den Anfang des nten Satzes vor dem aktuellen Satz.
- <) > Stellt den Cursor auf den Anfang des nächsten Satzes.
- < n) > Stellt den Cursor auf den Anfang des nten Satzes nach dem aktuellen Satz.

Das Beispiel in den folgenden Bildschirmen zeigt, wie der Cursor mit dem Kommando "Klammer auf" auf dem Bildschirm bewegt werden kann.

Plötzlich entdeckten wir weit draußen
Wale. Daniel sah sie als erster.

<(>

Wale. Daniel sah sie als erster.

6

Wiederholen Sie nun das Kommando, geben aber davor eine Zahl ein. Beispielsweise:

<3(> (oder)

<5(>

Hat der Cursor sich um die richtige Anzahl Sätze vor oder zurück bewegt?

Absatzweise springen

vi erkennt Absätze, die nach einer Leerzeile beginnen. Soll der Cursor zum Anfang eines Absatzes bewegt werden (oder im weiteren Verlauf dieser Anleitung ein ganzer Absatz gelöscht oder geändert werden), muß jeder Absatz mit einer Leerzeile enden.

- < { > Stellt den Cursor auf den Anfang eines Absatzes, dessen obere Begrenzung eine Leerzeile ist.
- < n { > Stellt den Cursor auf den Anfang des nten Absatzes vor dem aktuellen Absatz.
- < } > Stellt den Cursor auf den Anfang des nächsten Absatzes.
- < n } > Bewegt den Cursor zum nten Absatz unterhalb der aktuellen Zeile.

In den beiden folgenden Bildschirmen wird gezeigt, wie der Cursor zum Anfang eines anderen Absatzes bewegt wird.

Plötzlich entdeckten wir weit draußen
Wale. Daniel sah sie als erster.

< } >
"He, schaut mal! Da sind Wale!" schrie er aufgeregt.

Plötzlich entdeckten wir weit draußen

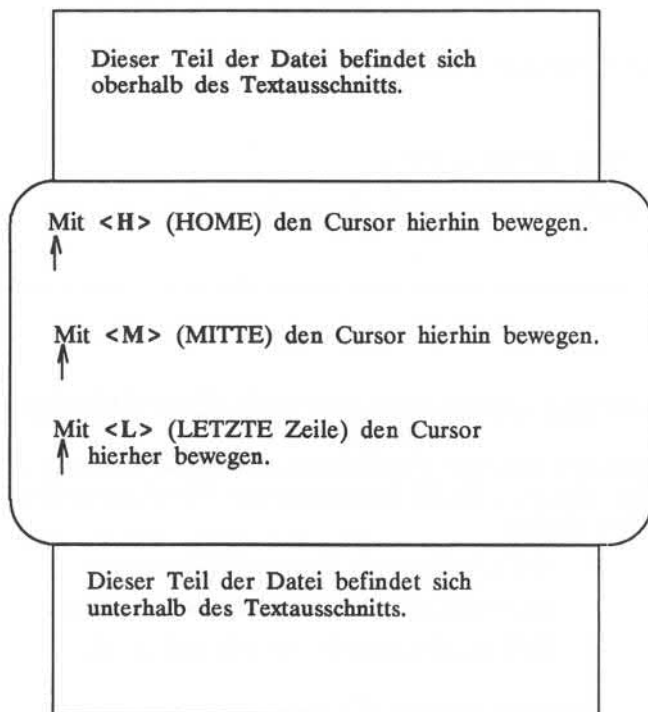
Wale. Daniel sah sie als erster.

"He, schaut mal! Da sind Wale!" schrie er aufgeregt.

Bewegen des Cursors in der aktuellen Anzeige

Der Editor vi verfügt auch über drei Kommandos zum Steuern des Cursors in einer Anzeige. Probieren Sie alle Kommandos aus. Sie müssen in Großbuchstaben eingegeben werden.

- <H>** Stellt den Cursor in die erste Bildschirmzeile.
- <M>** Stellt den Cursor in die mittlere Bildschirmzeile.
- <L>** Stellt den Cursor in die letzte Bildschirmzeile.



In den Abbildungen 6-3 bis 6-6 werden die vi-Kommandos, durch die der Cursor zeilen-, wort-, oder absatzweise bewegt oder im aktuellen Textausschnitt gesteuert wird. Weitere vi-Kommandos zur Cursor-Steuerung werden in Abbildung 6-7 an anderer Stelle in diesem Kapitel zusammengefaßt.

Positionieren auf ein Zeichen	
<h>	Bewegt den Cursor um ein Zeichen nach links.
<l>	Bewegt den Cursor um ein Zeichen nach rechts.
<BACKSPACE>	Bewegt den Cursor um ein Zeichen nach links.
<Leertaste>	Bewegt den Cursor um ein Zeichen nach rechts.
<fx>	Bewegt den Cursor nach rechts zum angegebenen Zeichen <i>x</i> .
<Fx>	Bewegt den Cursor nach links zum angegebenen Zeichen <i>x</i> .
<tx>	Bewegt den Cursor nach rechts zum Zeichen unmittelbar vor dem angegebenen Zeichen <i>x</i> .
<Tx>	Bewegt den Cursor nach links zum Zeichen unmittelbar nach dem angegebenen Zeichen <i>x</i> .
<;>	Setzt die Suche nach dem zuletzt mit <f>, <F>, <t> oder <T> angegebenen Zeichen in derselben Zeile und derselben Richtung fort. ; speichert das Zeichen und sucht die nächste Entsprechung in der aktuellen Zeile.
<, >	Setzt die Suche nach dem zuletzt mit <f>, <F>, <t> oder <T> angegebenen Zeichen in derselben Zeile in der Gegenrichtung fort. , speichert das Zeichen und sucht die vorhergehende Entsprechung in der aktuellen Zeile.

Abbildung 6-3: Zusammenfassung der vi-Cursor-Steuerkommandos (Seite 1 von 4)

Positionieren in einer Zeile	
<k>	Bewegt den Cursor in derselben Spalte eine Zeile nach oben (sofern dort ein Zeichen steht).
<j>	Bewegt den Cursor in derselben Spalte in die nächste Zeile (sofern dort ein Zeichen steht).
<->	Bewegt den Cursor zum Anfang der vorherigen Zeile.
<+>	Bewegt den Cursor zum Anfang der nächsten Zeile.
<CR>	Bewegt den Cursor zum Anfang der nächsten Zeile.

Abbildung 6-4: Zusammenfassung der vi-Cursor-Steuerkommandos (Seite 2 von 4)

Wortweise positionieren	
<w>	Stellt den Cursor auf das erste Zeichen im nächsten Wort.
<W>	Ignoriert alle Satzzeichen und stellt den Cursor auf das nächste Wort, das nur durch Leerzeichen begrenzt wird.
	Stellt den Cursor auf das erste Zeichen des vorherigen Wortes.
	Bewegt den Cursor nach links um ein Wort, das nur durch Leerzeichen begrenzt wird.
<e>	Stellt den Cursor ans Ende des aktuellen Wortes.
<E>	Wörter nur durch Leerzeichen begrenzen. Der Cursor wird auf das letzte Zeichen vor dem nächsten Leerzeichen oder dem Zeilenende gestellt.

Abbildung 6-5: Zusammenfassung vi-Cursor-Steuerkommandos (Seite 3 von 4)

Satzweise positionieren	
<(>	Stellt den Cursor an den Anfang des aktuellen Satzes.
<)>	Stellt den Cursor an den Anfang des nächsten Satzes.
Absatzweise positionieren	
<{>	Stellt den Cursor an den Anfang des aktuellen Absatzes.
<}>	Stellt den Cursor an den Anfang des nächsten Absatzes.
Positionieren in einem angezeigten Textausschnitt	
<H>	Stellt den Cursor in die erste Bildschirmzeile (Position 1).
<M>	Stellt den Cursor in die mittlere Bildschirmzeile.
<L>	Stellt den Cursor in die letzte Bildschirmzeile.

Abbildung 6-6: Zusammenfassung der vi-Cursor-Steuerkommandos (Seite 4 von 4)

Positionieren des Cursors in nicht angezeigtem Text

Eine Möglichkeit, den Cursor in einem Textteil zu positionieren, der nicht im aktuellen Editierfenster angezeigt wird, ist die Verwendung der Kommandos <20j> oder <20k>. Soll jedoch eine größere Datei editiert werden, muß man den Cursor schnell und präzise steuern können. In diesem Abschnitt werden die Kommandos erläutert, mit denen man den Cursor auf folgende Arten in der Datei bewegen kann:

- Vorwärts oder rückwärts blättern
- eine bestimmte Zeile in der Datei suchen
- die Datei nach einem Suchmuster durchsuchen

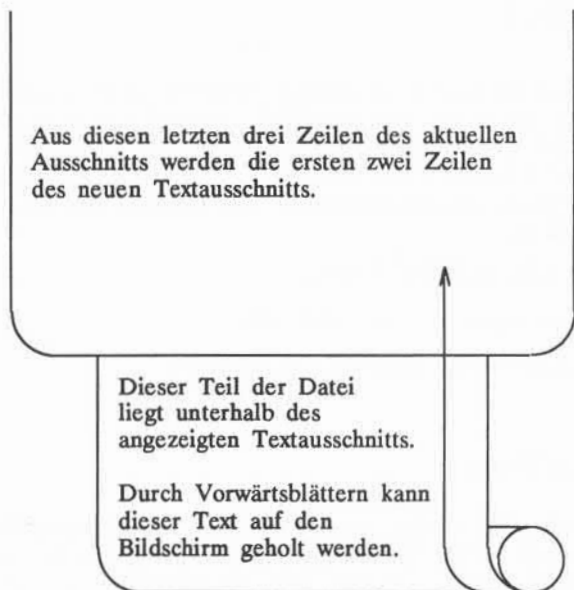
Blättern im Text

Zum Blättern im Text einer Datei stehen vier Kommandos zur Verfügung. Mit den Kommandos <^f> (CONTROL-f) und <^d> (CONTROL-d) wird vorwärts geblättert. Mit den Kommandos <^b> (CONTROL-b) und <^u> (CONTROL-u) wird rückwärts geblättert.

Das Kommando CONTROL-f

Mit dem Kommando <^f> (CONTROL-f) wird der Text um ein ganzes Fenster vorwärts geblättert. Dazu löscht vi den Bildschirm und baut ihn neu auf. Die drei untersten Zeilen des aktuellen Textausschnitts erscheinen dann ganz oben im nächsten Ausschnitt. Sind nun nicht mehr genügend Zeilen vorhanden, um den Bildschirm zu füllen, werden die leeren Zeilen mit einer ~ gekennzeichnet.

Zum Löschen und Aufbauen des Bildschirms geht vi folgendermaßen vor:

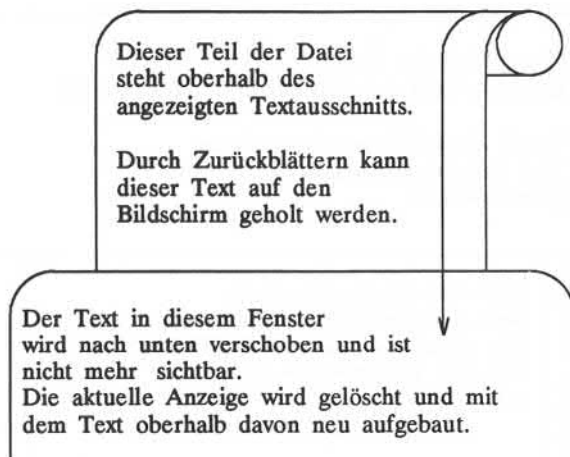


Das Kommando CONTROL-d

Mit dem Kommando `<^d>` (CONTROL-d) kann um einen halben Bildschirm vorwärts geblättert werden. Nach der Eingabe `<^d>` verschwindet der Text oben auf dem Bildschirm, und von unten rückt Text nach. Sind dafür nicht genügend Textzeilen vorhanden, wird ein Signalton ausgegeben.

Das Kommando CONTROL-b

Mit dem Kommando `<^b>` (CONTROL-b) um eine ganze Bildschirmseite zurückblättern. Dazu löscht vi den Bildschirm und baut ihn mit dem Text oberhalb des aktuellen Textausschnitts wieder auf. Anders als bei dem Kommando `<^f>` werden bei `<^b>` keine Referenzzeilen aus dem vorherigen Textausschnitt übernommen. Sind oberhalb des aktuellen Fensters nicht genügend Zeilen vorhanden, wird dies mit einem Signalton gemeldet, und die aktuelle Anzeige bleibt bestehen.



Blättern Sie nun zurück. Dazu

`<^b>`

eingeben. `vi` löscht den Bildschirm und baut einen neuen auf.

Dieser Teil der Datei
steht oberhalb des angezeigten Textausschnitts.

Durch Zurückblättern kann dieser
Text auf den Bildschirm
geholt werden.

Der Text dieses Abschnitts
wird nach unten verschoben
Die aktuelle Anzeige wird gelöscht und
mit dem Text oberhalb neu
aufgebaut.

Der gesamte Text des Anzeigefensters wird nach unten verschoben.

Das Kommando CONTROL-u

Mit dem Kommando <^u> (CONTROL-u) wird um einen halben Bildschirm zurückgeblättert. Die untersten Bildschirmzeilen werden gelöscht. Blättern Sie vorwärts, um den Text unterhalb des aktuellen Textausschnitts auf den Bildschirm zu holen.

<^u>

eingeben. Wenn der Cursor den Anfang der Datei erreicht, meldet ein Signalton, daß weiteres Blättern nicht möglich ist.

Suchen einer bestimmten Zeile

Mit dem Kommando <G> wird der Cursor in eine bestimmte Zeile im Fenster gestellt; befindet sich diese nicht auf dem aktuellen Bildschirm, löscht <G> den Bildschirm und baut ihn mit der neuen Zeile neu auf. Wird keine Zeilennummer angegeben, springt <G> in die letzte Zeile der Datei.

- <G> springt zur letzten Zeile der Datei
- <nG> springt zur nten Zeile der Datei

Zeilennummern

Jede Zeile der Datei hat eine Zeilennummer entsprechend ihrer Position im Puffer. Um die Nummer einer bestimmten Zeile abzufragen, wird der Cursor in diese Zeile gestellt und <^g> eingegeben. Mit dem Kommando <^g> wird unten am Bildschirm eine Statusmeldung angezeigt, die folgendes angibt:

- den Dateinamen
- ob die Datei geändert wurde
- die Nummer der Zeile, in der der Cursor steht
- die Gesamtzahl der Zeilen im Puffer
- die Position der aktuellen Zeile im Verhältnis zur Gesamtlänge des Puffers in Prozent

Dies ist die 35.-ste Zeile des Puffers.
Der Cursor befindet sich in dieser Zeile.

<^g>

Es sind noch einige Zeilen mehr Puffer.
Die letzte Zeile im Puffer ist Zeile 116.

Dies ist die 35.-ste Zeile des Puffers.
Der Cursor befindet sich in dieser Zeile.

Es sind noch einige Zeilen mehr Puffer.
Die letzte Zeile im Puffer ist Zeile 116.

"datei.name" [modified] line 36 of 116 —34%—

Suche nach einem Zeichenmuster: Die Kommandos / und ?

Am schnellsten gelangt man zu einer bestimmten Textstelle mit einem der Suchkommandos: /, ?, <n> oder <N>. Mit diesen Kommandos kann der Puffer vorwärts oder rückwärts nach dem nächsten Auftreten eines bestimmten Suchmusters durchsucht werden. Die Kommandos / und ? werden bei der Eingabe zusammen mit dem Suchmuster unten am Bildschirm angezeigt. Die Kommandos <n> und <N>, mit denen eine Suche mit den Kommandos / oder ? wiederholt werden kann, werden nicht angezeigt.

Mit dem Kommando /, gefolgt von einem *Suchmuster (/Muster)*, wird der Puffer ab der Cursor-Position vorwärts nach dem nächsten Auftreten der Zeichen *Muster* durchsucht und der Cursor auf das erste Zeichen des Suchmusters gestellt. Auf die Kommandozeile

/Hallo Freunde <CR>

hin wird beispielsweise das nächste Auftreten der Wörter **Hallo Freunde** im Puffer gesucht und der Cursor unter dem **H** positioniert.

Mit ?, gefolgt von einem *Muster (?Muster)*, wird im Puffer rückwärts nach den in *Muster* angegebenen Zeichen gesucht und der Cursor auf das erste Zeichen gestellt. Mit der Kommandozeile

?Datensatz <CR>

beispielsweise wird das nächste Auftreten des Wortes **Datensatz** (vor der aktuellen Cursor-Position) gesucht und der Cursor unter das **D** positioniert.

Wird nach zwei Wörtern gesucht, können diese Kommandos sie nicht finden, wenn die Wörter auf zwei Zeilen verteilt sind. Werden z. B. die Wörter **Hallo Freunde** gesucht, findet das Suchkommando das Muster nicht, wenn **Hallo** am Ende einer Zeile und **Freunde** am Anfang der nächsten Zeile steht.

Keinen Unterschied macht es allerdings, wo die gesuchte Zeichenfolge im Puffer steht. Wird gegen Ende des Puffers ein Muster mit dem Kommando */Muster* gesucht, das am Beginn des Puffers steht, wird es gefunden.

Mit den Kommandos <n> und <N> können Suchanforderungen, die mit /Muster oder ?Muster gestartet wurden, ohne erneutes Eingeben wiederholt werden.

<n> Wiederholung des letzten Suchkommandos.

<N> Wiederholung des letzten Suchkommandos in der Gegenrichtung.

Soll z. B. die Datei rückwärts nach der Zeichenfolge er durchsucht werden, wird die Suche mit ?er gestartet und mit <n> fortgesetzt. Mit den folgenden Bildschirmen wird Schritt für Schritt gezeigt, wie das Kommando <n> die Datei rückwärts durchsucht und vier Stellen findet, an denen die Zeichenfolge er auftritt.

Plötzlich entdeckten wir weit draußen
Wale. Daniel sah sie als erster.

"He, schaut mal! Da sind Wale!" schrie er aufgeregt.

?er

Plötzlich entdeckten wir weit draußen
Wale. Daniel sah sie als erster.

"He, schaut mal! Da sind Wale!" schrie er aufgeregt.

(1)

Plötzlich entdeckten wir weit draußen
Wale. Daniel sah sie als erster.

"He, schaut mal! Da sind Wale!" schrie er aufgeregt.

<a>

Plötzlich entdeckten wir weit draußen
Wale. Daniel sah sie als erster.

"He, schaut mal! Da sind Wale!" schrie er aufgeregt.

(2)

Plötzlich entdeckten wir weit draußen
Wale. Daniel sah sie als erster.

"He, schaut mal! Da sind Wale!" schrie er aufgeregt.

<n>

Plötzlich entdeckten wir weit draußen
Wale. Daniel sah sie als erster.

(3)

"He, schaut mal! Da sind Wale!" schrie er aufgeregt.

Plötzlich entdeckten wir weit draußen
Wale. Daniel sah sie als erster.

<n>

"He, schaut mal! Da sind Wale!" schrie er aufgeregt.

Plötzlich entdeckten wir weit draußen
Wale. Daniel sah sie als erster.

(4)

"He, schaut mal! Da sind Wale!" schrie er aufgeregt.

Mit den Suchkommandos / und ? können bestimmte Stellen, an denen ein *Muster* zum n-ten Mal auftritt, angefordert werden. Man kann beispielsweise nicht das dritte Auftreten eines *Musters* ab der aktuellen Position abrufen.

In Abbildung 6-7 sind die vi-Kommandos zum Blättern im Text, zum Angeben einer Zeilennummer und zur Suche nach einem *Muster* zusammengefaßt.

Blättern	
<^f>	Vorwärtsblättern um einen ganzen Bildschirm, Anzeige des Textausschnitts unterhalb des aktuellen Ausschnitts.
<^d>	Vorwärtsblättern um einen halben Bildschirm, Anzeige der Zeilen unterhalb des aktuellen Ausschnitts.
<^b>	Zurückblättern um einen ganzen Bildschirm, Anzeige des Textes oberhalb des aktuellen Ausschnitts.
<^u>	Zurückblättern um einen halben Bildschirm, Anzeige der Zeilen oberhalb des aktuellen Ausschnitts.
Positionieren in eine bestimmte Zeile	
<1G>	Springt zur ersten Zeile der Datei.
<G>	Springt zur letzten Zeile der Datei.
<^g>	Anzeige der Zeilennummer und des Dateistatus.
Suche nach einem Suchmuster	
/Muster	Sucht vorwärts nach dem nächsten Auftreten des <i>Musters</i> im Puffer. Stellt den Cursor auf das erste Zeichen des <i>Musters</i> .
?Muster	Sucht rückwärts nach dem ersten Auftreten des <i>Musters</i> im Puffer von der Cursor-Position aus. Stellt den Cursor auf das erste Zeichen des <i>Musters</i> .
<n>	Wiederholt das letzte Suchkommando.
<N>	Wiederholt das letzte Suchkommando in der Gegenrichtung.

Abbildung 6-7: Zusammenfassung der zusätzlichen vi-Cursor-Steuerkommandos

Übung 2

- 2-1. Legen Sie eine Datei namens **prob2** an. Numerieren Sie die Zeilen von 1 bis 50 durch. Die Datei sollte ähnlich aussehen wie das folgende Beispiel.

```
1
2
3
.
.
.
48
49
50
```

- 2-2. Üben Sie alle Blätter-Kommandos und achten Sie darauf, um wieviele Zeilen jeweils geblättert wird. Führen Sie folgende Kommandos aus:

```
<^f>
<^b>
<^u>
<^d>
```

- 2-3. Springen Sie ans Ende der Datei, und fügen Sie folgende Textzeile an.

```
123456789 123456789
```

Auf welche Zahl wird der Cursor mit dem Kommando <7h> gestellt?
Auf welche Zahl wird der Cursor mit dem Kommando <3l> gestellt?

- 2-4. Üben Sie die Kommandos <\$> und <0> (null).
2-5. Zum ersten Zeichen der Zeile gehen, das kein Leerzeichen ist. Zum ersten Zeichen im nächsten Wort springen. Zum ersten Zeichen des links davon stehenden Wortes zurückgehen. Zum Ende des Wortes springen.

- 2-6. Stellen Sie den Cursor in die erste Zeile der Datei. Geben Sie die Kommandos ein, mit denen der Cursor in die Mitte, in die letzte Zeile und in die erste Zeile des Fensters gestellt wird.
- 2-7. Suchen Sie die Zahl 8 und dann das nächste Auftreten der Zahl 8. Suchen Sie die Zahl 48.

Eingeben von Text

Zum Eingeben von Text stehen drei Grundkommandos zur Verfügung:

- <a> Text anhängen
- <i> Text einfügen
- <o> Eine neue Zeile anlegen, in die Text eingegeben werden kann

Wenn der Text mit einem dieser Kommandos eingegeben wurde, drücken Sie die Taste ESCAPE, um in den Kommando-Modus von vi zurückzukehren.

Anhängen von Text

- <a> Text nach dem Cursor anhängen
- <A> Text am Ende der aktuellen Zeile anhängen

Mit dem Kommando <a> wurde bereits im Abschnitt "Anlegen einer Datei" geübt. Legen Sie eine Datei namens **müll2** an. Hängen Sie Text mit dem Kommando <a> an. Kehren Sie mit der Taste ESCAPE zum Kommando-Modus von vi zurück. Vergleichen Sie dann das Kommando <a> mit dem Kommando <A>.

Einfügen von Text

- <i> Text vor der Cursor-Position einfügen
- <I> Text am Anfang der aktuellen Zeile, vor dem ersten Zeichen, das kein Leerzeichen ist, einfügen

Mit ESCAPE kann man zum Kommando-Modus von vi zurückkehren.

In den folgenden Beispielen können die Kommandos zum Anhängen und Einfügen miteinander verglichen werden. Die Pfeile zeigen die Cursor-Position an, an der der neue Text hinzugefügt wird.

NACH dem H von Hier drei Leerzeichen anhängen

<a>

NACH dem H von H hier drei Leerzeichen einfügen.

<ESC>

VOR dem H von Hier drei Leerzeichen einfügen.

<I>.

VOR dem H von hier drei Leerzeichen einfügen.

<ESC>

In beiden Fällen hat der Benutzer den Textmodus mit der Taste ESCAPE verlassen.

Anlegen einer Zeile für neuen Text

- <o> Dient zum Eingeben von Text ab dem Anfang einer neuen Zeile unterhalb der aktuellen Zeile. Dieses Kommando kann an jeder Position der aktuellen Zeile eingegeben werden.
- <O> Dient zum Eingeben von Text ab dem Anfang einer neuen Zeile oberhalb der aktuellen Zeile. Auch dieses Kommando kann an jeder Position der aktuellen Zeile eingegeben werden.

Mit diesen Kommandos wird eine neue Zeile unmittelbar über oder unter der aktuellen Zeile angelegt und der Textmodus aktiviert. Auf den folgenden Bildschirmen wird beispielsweise mit dem Kommando <O> eine Zeile über der aktuellen Zeile und mit dem Kommando <o> eine Zeile unter der aktuellen Zeile angelegt. In beiden Fällen wartet der Cursor am Anfang der neuen Zeile auf die Eingabe.

Text ÜBER der aktuellen Zeile eingeben.

<O>

[Leerzeile]

Text ÜBER der aktuellen Zeile eingeben.

Nun Text UNTER der aktuellen Zeile eingeben.

<o>

Nun Text UNTER der aktuellen Zeile eingeben.
[Leerzeile]

In Abbildung 6-8 sind die Kommandos zum Eingeben und Hinzufügen von Text mit dem Bildschirmditor vi zusammengefaßt.

6

Kommando	Funktion
<a>	Text hinter der Cursor-Position eingeben.
<A>	Text am Ende der aktuellen Zeile eingeben.
<i>	Text vor dem Cursor eingeben.
<I>	Text vor dem ersten Zeichen der aktuellen Zeile, das kein Leerzeichen ist, eingeben.
<o>	Text am Anfang einer neuen Zeile unter der aktuellen Zeile eingeben.
<O>	Text am Anfang einer neuen Zeile über der aktuellen Zeile eingeben.
<ESC>	Mit vi aus einem der obigen Texteingabemodi in den Kommando-Modus zurückkehren.

Abbildung 6-8: Zusammenfassung der vi-Kommandos für die Texteingabe

Übung 3

3-1. Legen Sie eine Textdatei namens **prob3** an.

3-2. Fügen Sie folgende vier Textzeilen ein:

**Text anhängen
Text einfügen
Der Computer
übernimmt Routineaufgaben.**

3-3. Fügen Sie die folgende Zeile über der letzten Zeile ein:

und Bilanzen erstellen

3-4. Mit einem Kommando zum Einfügen von Text fügen Sie die folgende Zeile oberhalb der dritten Zeile ein:

Text löschen

3-5. Fügen Sie die folgende Textzeile unter der aktuellen Zeile ein:

Budgetierung

3-6. Fügen Sie mit einem Kommando "Anhängen" die folgende Textzeile nach der letzten Zeile an:

Aber Computer sind interessante Maschinen.

3-7. Geben Sie in der ersten Zeile vor dem Wort "Text" das Wort "Etwas" ein.

Üben Sie nun alle sechs Kommandos zum Eingeben von Text.

3-8. Verlassen Sie **vi** und lesen Sie im nächsten Abschnitt nach, wie Fehler, die bei der Texteingabe gemacht wurden, zu löschen sind.

Löschen von Text

Zum Löschen von Text stehen im Kommando-Modus mehrere Kommandos zur Verfügung, kleinere Textteile können auch im Textmodus gelöscht werden. Auch können die Ergebnisse des zuletzt eingegebenen Kommandos vollständig aufgehoben werden.

Löschen von Text im Textmodus

Zum Löschen jeweils eines Zeichens wird im Textmodus die Taste BACKSPACE verwendet.

<BACKSPACE> Löscht das aktuelle Zeichen (das Zeichen, auf dem der Cursor steht).

Im Textmodus setzt die Taste BACKSPACE den Cursor um ein Zeichen zurück und löscht dabei das entsprechende Zeichen. Das gelöschte Zeichen verschwindet jedoch erst vom Bildschirm, wenn es überschrieben oder die Taste ESCAPE zur Rückkehr in den Kommando-Modus gedrückt wird.

Im folgenden Beispiel stellen die Pfeile den Cursor dar.

Hänschen klein ging alle

<BACKSPACE> <BACKSPACE>

Hänschen klein ging alle

<ESC>

Hänschen klein ging all

Hier wird deutlich, daß die Zeichen erst vom Bildschirm gelöscht werden, wenn die Taste ESCAPE gedrückt wird.

Im Textmodus kann noch mit zwei weiteren Tasten Text gelöscht werden. Auch wenn sie nicht oft benutzt werden, sollten Sie wissen, wie sie funktionieren. Der Abschnitt über Sonderkommandos gibt Auskunft darüber, wie die Sonderfunktionen aufgehoben und die Tasten als Textzeichen verwendet werden können.

<^w> macht die Eingabe des aktuellen Wortes rückgängig

<@> löscht den seit der Aktivierung des Textmodus eingegebenen Text in der aktuellen Zeile

Bei der Eingabe <^w> geht der Cursor zum ersten Zeichen des zuletzt geschriebenen Wortes zurück. Damit löscht er das Wort nicht, sondern wartet, bis die Taste ESCAPE gedrückt oder das Wort überschrieben wird. Ähnlich funktioniert das Zeichen <@>, hier wird jedoch der gesamte Text gelöscht, der seit der letzten Aktivierung des Textmodus in der aktuellen Zeile eingegeben wurde.

Rückgängigmachen des letzten Kommandos

Bevor Sie die Kommandos zum Löschen üben, sollten Sie sich mit dem Kommando **u** vertraut machen. Damit wird das letzte eingegebene Kommando rückgängig gemacht.

<u> macht das letzte Kommando rückgängig

<U> stellt den Zustand der aktuellen Zeile vor der Änderung wieder her

Haben Sie versehentlich Zeilen gelöscht, geben Sie **<u>** ein und die Zeilen erscheinen wieder auf dem Bildschirm. Wurde das falsche Kommando eingegeben, kann es mit **<u>** zurückgenommen werden. Mit dem Kommando **<U>** werden alle Änderungen in der aktuellen Zeile rückgängig gemacht, solange der Cursor noch in dieser Zeile steht.

Wird **<u>** zweimal hintereinander eingegeben, annulliert das zweite Kommando das erste; der Widerruf wird widerrufen! Wurde z.B. eine Zeile versehentlich gelöscht und mit der Eingabe **<u>** wiederhergestellt, wird sie erneut gelöscht, wenn **<u>** zum zweitenmal eingegeben wird. Mit diesem Kommando kann man sich viel unnötige Arbeit ersparen.

Löschen von Kommandos im Kommando-Modus

Es wurde bereits erklärt, daß man Kommandos eine Zahl voranstellen kann. In **vi** können nach zahlreichen Kommandos wie denen zum Löschen und Ändern auch Cursor-Steuerkommandos eingegeben werden. Mit dem Cursor-Steuerkommando kann ein Textobjekt, z.B. ein Wort, eine Zeile, ein Satz oder ein Absatz angegeben werden. **vi**-Kommandos haben folgendes allgemeine Format:

[Zahl][Kommando]Textobjekt

Die eckigen Klammern um einige Elemente des Kommandoformates zeigen an, daß diese Elemente wahlweise angegeben werden können.

Im Kommando-Modus wird bei allen Kommandos zum Löschen unerwünschter Text sofort vom Bildschirm gelöscht und der entsprechende Bildschirmteil angepaßt.

Das Kommando zum Löschen richtet sich nach dem allgemeinen Kommando-Format von vi.

[Zahl]dTextobjekt

Löschen von Wörtern

Mit dem Kommando `<dw>` kann ein Wort oder ein Teil davon gelöscht werden. Stellen Sie den Cursor auf das erste zu löschende Zeichen und geben Sie `<dw>` ein. Das Zeichen, auf dem der Cursor steht, und alle folgenden Zeichen dieses Wortes werden gelöscht.

den tiefen dunklen Tiefen des Sees.

`<2dw>`

den Tiefen des Sees.

Löschen von Text

Das Kommando `<dw>` löscht ein Wort oder ein Interpunktionszeichen und das Leerzeichen danach. Wird vor dem Kommando eine Zahl angegeben, können mehrere Wörter gleichzeitig gelöscht werden. Um beispielsweise drei Wörter und zwei Kommata zu löschen, geben Sie `<5dw>` ein.

den tiefen, tiefen, dunklen Tiefen des Sees

`<5dw>`

den Tiefen des Sees

Löschen von Absätzen

Zum Löschen von Absätzen dienen die folgenden Kommandos.

`<d{>` oder `<d>`

Beobachten Sie, wie sich die Datei verändert. Vergessen Sie nicht, daß gelöschter Text sich mit dem Kommando `<u>` wiederhergestellt werden kann.

Löschen von Zeilen

Zum Löschen einer Zeile wird `<dd>` eingegeben. Um mehrere Zeilen gleichzeitig zu löschen, wird vor dem Kommando eine Zahl angegeben. Mit

`<10dd>`

z.B. werden zehn Zeilen gelöscht. Wird eine größere Anzahl von Zeilen gelöscht, zeigt vi unten am Bildschirm folgende Meldung an:

```
10 lines deleted
```

Enthält die Datei unterhalb der aktuellen Zeile weniger als zehn Zeilen, wird dies mit einem Signalton gemeldet, und es wird keine Zeile gelöscht.

Löschen von Text ab der Cursor-Position

Um den gesamten Text einer Zeile nach der Cursor-Position zu löschen, wird der Cursor auf das erste zu löschende Zeichen gestellt und folgendes eingegeben:

`<D>` oder `<d$>`.

Bei diesen beiden Kommandos können nicht mehrere Zeilen auf einmal gelöscht werden, sondern sie gelten jeweils nur für die aktuelle Zeile.

In Abbildung 6-9 sind die vi-Kommandos zum Löschen von Text zusammengefaßt.

Kommando	Funktion
<p>Im EINFÜGE-Modus:</p> <p><BACKSPACE></p> <p><^h></p> <p><^W></p> <p><@></p>	<p>Löscht das aktuelle Zeichen.</p> <p>Löscht das aktuelle Zeichen.</p> <p>Löscht das aktuelle Wort.</p> <p>Löscht die aktuelle Zeile neu eingegebenen Textes oder löscht den gesamten neuen Text in der aktuellen Zeile.</p>
<p>Im KOMMANDO-Modus:</p> <p><u></p> <p><U></p> <p><x></p> <p><ndx></p> <p><dw></p> <p><dW></p> <p><dd></p> <p><D></p> <p><d></p> <p><d}></p>	<p>Annulliert das zuletzt eingegebene Kommando.</p> <p>Stellt den früheren Zustand der aktuellen Zeile wieder her.</p> <p>Löscht das aktuelle Zeichen.</p> <p>Löscht <i>n</i> Textobjekte des Typs <i>x</i>.</p> <p>Löscht das Wort an der Cursor-Position samt dem folgenden Leerzeichen oder bis zum nächsten Interpunktionszeichen.</p> <p>Löscht das Wort und Interpunktionszeichen an der Cursor-Position einschließlich des nächsten Leerzeichens.</p> <p>Löscht die aktuelle Zeile.</p> <p>Löscht in der aktuellen Zeile alle Zeichen rechts vom Cursor.</p> <p>Löscht den aktuellen Satz.</p> <p>Löscht den aktuellen Absatz.</p>

Abbildung 6-9: Zusammenfassung der Kommandos zum Löschen

Übung 4

- 4-1. Legen Sie eine Datei namens **prob4** an und geben Sie die folgend vier Zeilen Text ein:

**Im Haushalt gibt es
viele langweilige, immer wiederkehrende
Arbeiten, daher sollte man einen
Roboter dafür anschaffen.**

- 4-2. Stellen Sie den Cursor in die zweite Zeile und fügen Sie am Zeilenende folgendes an:

mühsame und unangenehme

Löschen Sie das Wort "unangenehme" im Modus "Anhängen".

Löschen Sie das Wort "langweilige" im Kommando-Modus.

Auf welche Art könnte man das Wort "langweilige" noch löschen?

- 4-3. Fügen Sie am Anfang der vierten Zeile folgendes ein:

vollautomatischen, elektronischen

Löschen Sie die Zeile.

Wie läßt sich der Zeileninhalt entfernen, ohne die Zeile selbst zu löschen?

Löschen Sie alle Zeilen mit einem Kommando.

- 4-4. Verlassen Sie den Bildschirmditor und löschen Sie die leere Datei aus dem Verzeichnis.

Ändern von Text

Die Kommandos zum Löschen und zur Texteingabe sind eine Möglichkeit, Text zu ändern. Eine weitere Möglichkeit besteht darin, mit einem Kommando Text gleichzeitig zu löschen und einzugeben. Es gibt drei Grundkommandos zum Ersetzen: `<r>`, `<s>` und `<c>`.

Ersetzen von Text

- `<r>` Ersetzt das aktuelle Zeichen (auf dem der Cursor steht). Mit diesem Kommando wird der Textmodus nicht aufgerufen, daher muß auch anschließend nicht die Taste ESCAPE gedrückt werden.
- `<nr>` Ersetzt *n* Zeichen durch einen bestimmten Buchstaben. Dieses Kommando endet, wenn das *nte* Zeichen ersetzt ist. Danach braucht nicht die Taste ESCAPE gedrückt zu werden.
- `<R>` Ersetzt nur die Zeichen, die mit neuem Text überschrieben werden, bevor das Kommando ESCAPE eingegeben wird. Am Zeilenende wird die Eingabe als neuer Text angehängt.

Mit dem Kommando `<r>` wird das aktuelle Zeichen durch das nächste eingegebene Zeichen ersetzt. Soll beispielsweise im folgenden Satz das Wort Tiere durch Biere ersetzt werden:

Im Zirkus gibt es viele Tiere.

Stellen Sie den Cursor unter das T von Tiere und geben Sie

`<r>B`

ein. Dann lautet der Satz

Im Zirkus gibt es viele Biere.

Soll viele in 77777 geändert werden, stellt man den Cursor auf das v von viele und gibt folgendes ein

`<5r7>`

Das Kommando `<r>` ersetzt die fünf Buchstaben des Wortes viele durch fünfmal die Zahl sieben.

Im Zirkus gibt es 7777 Biere.

Austauschen von Text

Mit dem Kommando "Austauschen" werden Zeichen ersetzt, danach kann aber an dieser Stelle weiter Text eingegeben werden, bis die Taste ESCAPE gedrückt wird.

- <s> Löscht das Zeichen, auf dem der Cursor steht, und hängt Text an. Der Textmodus wird mit der Taste ESCAPE beendet.
- <ns> Löscht *n* Zeichen und hängt Text an. Mit der Taste ESCAPE wird der Textmodus beendet.
- <S> Ersetzt alle Zeichen in der Zeile.

Bei dem Kommando <s> wird das letzte Zeichen der zu ersetzenden Zeichenfolge durch ein Dollar-Zeichen \$ ersetzt. Die Zeichen werden erst dann vom Bildschirm gelöscht, wenn sie überschrieben werden, oder wenn man den Textmodus mit der Taste ESCAPE verläßt.

Es ist zu beachten, daß mit <r> und <s> keine Argumente verwendet werden dürfen.

Angenommen, in dem Satz Ich bekomme vierhundert DM Gehalt, soll "hundert" durch zigtausend" ersetzt werden. Stellen Sie den Cursor auf das *h* von *hundert* und geben Sie <7s> ein. Achten Sie darauf, wo das Zeichen \$ erscheint.

Ich bekomme vierhundert DM Gehalt.

<7s>

Nun zigtausend eingeben

Ich bekomme vierhunder\$ DM Gehalt.

zigtausend

Ich bekomme vierzigtausend DM Gehalt.

Ändern von Text

Mit dem Kommando zum Austauschen werden Zeichen ersetzt. Mit dem Kommando zum Ändern werden Textobjekte ausgetauscht, und ab dieser Stelle wird weiter Text eingegeben, bis die Taste ESCAPE gedrückt wird. Die Taste ESCAPE beendet das Kommando.

Zu dem Kommando "Ändern" kann ein Argument eingegeben werden. Zeichen, Wörter und ganze Zeilen können durch neuen Text ersetzt werden.

<nex> Ersetzt *n* Textobjekte des Typs *x*, z.B. Sätze (die mit <> markiert sind) und Absätze (markiert mit < } >).

- <cw>** Ersetzt ein Wort oder die verbleibenden Zeichen eines Wortes durch neuen Text. Der Editor vi markiert das letzte zu ändernde Zeichen mit einem Dollarzeichen \$.
- <ncw>** Ersetzt *n* Wörter.
- <cc>** Ersetzt alle Zeichen in der Zeile.
- <ncc>** Ersetzt alle Zeichen in der aktuellen Zeile und bis zu *n* Zeilen Text.
- <C>** Ersetzt die restlichen Zeichen in der Zeile von der Cursor-Position bis zum Zeilenende.
- <nC>** Ersetzt die restlichen Zeichen ab der Cursor-Position und alle Zeilen ab der aktuellen bis zu *n* Zeilen.

Die Kommandos zum Ändern, **<cw>** und **<C>**, markieren das letzte zu ändernde Zeichen mit dem Zeichen \$. Dies sieht im folgenden Beispiel so aus:

Sie kommen jetzt wohl am Dienstag.

<cw>

Sie kommen jetzt wohl am Diensta\$.

Donnerstag <ESC>

Sie kommen jetzt wohl am Donnerstag.

Das neue Wort (Donnerstag) ist länger als das ersetzte (Dienstag). Sobald das Kommando zum Ändern ausgeführt wird, ist der Textmodus aktiv, und es kann beliebig langer Text eingegeben werden. Der Puffer nimmt Text auf, bis die Taste ESCAPE gedrückt wird.

Ähnlich wirkt das Kommando <C>, wenn es zum Ändern des restlichen Textes in der Zeile verwendet wird. Nach der Eingabe des Kommandos wird das Ende des zu löschenden Textes mit einem \$ markiert, dann der Textmodus aktiviert und gewartet, daß der alte Text mit neuem überschrieben wird. Die folgenden Bildschirme zeigen ein Beispiel für das Kommando C.

Dies ist die 1. Zeile.
Oh, ich muß bei der falschen Nummer sein.

<C>
Dies ist die 3. Zeile.
Dies ist die 4. Zeile.

Dies ist die 1. Zeile.
Oh, ich muß bei der falschen Nummer sein\$

Dies ist die 2. Zeile.<ESC>
Dies ist die 3. Zeile.
Dies ist die 4. Zeile.

Dies ist die 1. Zeile.
Dies ist die 2. Zeile.
Dies ist die 3. Zeile.
Dies ist die 4. Zeile.

Nun sollen Sie versuchen, Argumente zu kombinieren. Geben Sie z.B. folgendes ein:

`<c{>`

Da das Kommando "Annullieren" bekannt ist, kann mit verschiedenen Argumenten oder vorangestellten Zahlen geübt werden. Bevor Sie das Kommando `<u>` verwenden, müssen Sie die Taste ESCAPE drücken, da `<c>` den Textmodus aktiviert.

`<S>` mit `<cc>` vergleichen. Die beiden Kommandos müßten dieselben Ergebnisse erzielen.

In Abbildung 6-10 sind die vi-Kommandos zum Ändern von Text zusammengefaßt.

Kommando	Funktion
<r>	Ersetzt das aktuelle Zeichen.
<R>	Ersetzt nur die Zeichen, die bis zum Drücken der Taste ESCAPE mit neuem Text überschrieben werden.
<s>	Löscht das Zeichen, auf dem der Cursor steht, und hängt neuen Text an. Zum Beenden des Modus "Anhängen" die Taste ESCAPE drücken.
<S>	Ersetzt alle Zeichen in der Zeile.
<cc>	Ersetzt alle Zeichen in der Zeile.
<nex>	Ersetzt <i>n</i> Textobjekte des Typs <i>x</i> , z.B. Sätze (markiert mit < >) und Absätze (markiert mit < } >).
<cw>	Ersetzt ein Wort oder die restlichen Zeichen eines Wortes durch neuen Text.
<C>	Ersetzt die restlichen Zeichen in der Zeile von der Cursor-Position bis zum Zeilenende.

Abbildung 6-10: Zusammenfassung der vi-Kommandos zum Ändern von Text

Elektronisches Verschieben von Text

vi bietet einige Kommandos zum Verschieben von Text in einer Datei. Mit anderen Kommandos kann Text kopiert und in einen anderen Abschnitt der Datei verschoben werden.

Verschieben von Text

Text kann innerhalb des vi-Puffers verschoben werden, indem man die entsprechenden Zeilen an ihrer ursprünglichen Position löscht und an der gewünschten Stelle einfügt. Der zuletzt gelöschte Text wird in einem temporären Puffer gespeichert. Wird der Cursor an die Stelle der Datei gesetzt, an der die gelöschten Zeilen stehen sollen, und die Taste **p** gedrückt, werden diese Zeilen unter der aktuellen Zeile eingefügt.

<p> Fügt den Inhalt des temporären Puffers hinter dem Cursor ein.

Ein Teil eines Satzes, der mit dem Kommando **<D>** gelöscht wurde, kann an eine beliebige Stelle in einer anderen Zeile gestellt werden. Stellen Sie den Cursor auf das Leerzeichen zwischen zwei Wörtern und drücken Sie **<p>**. Der zuvor gelöschte Text wird hinter dem Cursor eingefügt.

Zeichen, die mit **<nx>** gelöscht wurden, werden ebenfalls in einen temporären Puffer aufgenommen. Jedes gerade gelöschte Textobjekt kann mit **<p>** an anderer Stelle in den Text eingefügt werden.

Das Kommando **<p>** muß unmittelbar nach einem Kommando zum Löschen ausgeführt werden, da der temporäre Puffer immer nur die Ergebnisse eines Kommandos aufnehmen kann. Mit dem Kommando **<p>** kann auch Text kopiert werden, der mit dem Kommando Kopieren in den Puffer gestellt wurde. Dieses Kommando (**<y>**) wird unter "Kopieren von Text" erläutert.

Korrektur vertauschter Zeichen

Vertauschte Zeichen lassen sich rasch mit einer Kombination der Kommandos **<x>** und **<p>** zu **<xp>** korrigieren. **<x>** löscht das Zeichen. **<p>** stellt es hinter das nächste Zeichen.

Die folgende Zeile enthält einen Fehler.

Eine Zeile Tetx

Zum raschen Korrigieren wird der Cursor auf das t in tx gestellt und die Tasten <x> und <p> gedrückt - diese Reihenfolge muß eingehalten werden. Das Ergebnis lautet:

Eine Zeile Text

Geben Sie in der Übungsdatei einen Tippfehler ein und korrigieren Sie ihn mit dem Kommando <xp>. Warum funktioniert dieses Kommando?

Kopieren von Text

Eine oder mehrere Zeilen Text können in einen temporären Puffer und von dort aus an eine beliebige Position in der Datei gestellt werden. Mit dem Kommando <p> wird der Text an seine neue Position gebracht, und zwar unter der aktuellen Zeile.

Das Format des Kommandos Kopieren entspricht dem allgemeinen Kommandoformat von vi.

[Zahl]y[Textobjekt]

Durch das Kopieren wird der Text an seiner ursprünglichen Position nicht gelöscht. Soll derselbe Text an mehreren Stellen erscheinen, kann man mit diesem Kommando vermeiden, ihn mehrmals schreiben zu müssen. Soll der Text jedoch nur einmal vorkommen, muß darauf geachtet werden, daß er nach dem Kopieren an der ursprünglichen Position gelöscht wird.

In Abbildung 6-11 sind die Möglichkeiten zur Verwendung des Kommandos "Kopieren" zusammengefaßt.

Kommando	Funktion
<code><n>x</code>	Kopiert <i>n</i> Textobjekte des Typs <i>x</i> (z.B. Sätze) und Absätze).
<code><yw></code>	Kopiert ein Wort.
<code><yy></code>	Kopiert die aktuelle Zeile.
<code><n>yy</code>	Kopiert <i>n</i> Zeilen.
<code><y></code>	Kopiert den gesamten Text bis zum Satzende.
<code><y}></code>	Kopiert den gesamten Text bis zum Ende des Absatzes.

Abbildung 6-11: Zusammenfassung des Kommandos Kopieren

Hier gilt, daß bei diesem Kommando die Anzahl der zu kopierenden Textobjekte angegeben werden kann.

Geben Sie die folgenden Kommandozeilen ein und beobachten Sie das Ergebnis auf dem Bildschirm. (Nicht vergessen, daß das letzte Kommando immer zurückgenommen werden kann.) Folgendes eingeben:

`<5yw>`

Den Cursor an eine andere Stelle positionieren. Folgendes eingeben:

`<p>`

Nun einen Absatz auswählen und hinter dem aktuellen Absatz einfügen `<y>`. Nun zum Dateiende gehen `<G>` und denselben Absatz dort anfügen.

Kopieren oder Verschieben von Text mit Registern

Das Verschieben oder Kopieren von mehreren Textteilen in der Datei ist eine mühselige Arbeit. Für solche Fälle bietet vi eine Abkürzungsmöglichkeit: benannte Register, in denen der Text gespeichert werden kann, bis er gebraucht wird. Zum Speichern muß der Text entweder mit den Kommandos zum Kopieren oder zum Löschen in das Register gestellt werden.

Die Verwendung von Registern ist sinnvoll, wenn ein Textabschnitt mehrmals in der Datei vorkommen soll. Der gewählte Text bleibt in dem angegebenen Register, bis die Editiersitzung beendet oder ein anderer Textabschnitt durch Löschen oder Kopieren in dieses Register gestellt wird.

Das Kommando hat folgendes allgemeine Format:

`[Zahl][x]Kommando[Textobjekt]`

Das *x* steht für den Namen des Registers, der ein beliebiger Einzelbuchstabe sein kann. Diesem Namen muß ein Anführungszeichen vorangestellt werden. Den Cursor beispielsweise an den Anfang einer Zeile stellen und folgendes eingeben:

`<3"ayy>`

Weiteren Text eingeben und ans Dateiende gehen. Folgendes eingeben:

`<"ap>`

Sind die im Register *a* gespeicherten Zeilen am Ende der Datei erschienen?

In Abbildung 6-12 sind die Kommandos zum Verschieben von Text zusammengefaßt.

Kommando	Funktion
<p>	Stellt den Inhalt des temporären Puffers, der sich aus dem letzten Kommando "Löschen" oder "Kopieren" ergibt, hinter dem Cursor in den Text.
<yy>	Kopiert eine Textzeile in einen temporären Puffer.
<nxx>	Kopiert <i>n</i> Textobjekte des Typs <i>x</i> in einen temporären Puffer.
<"xyi">	Stellt die Kopie eines Textobjekts vom Typ <i>n</i> in das benannte Register <i>x</i> .
<"xp">	Fügt den Inhalt des Registers <i>x</i> nach dem Cursor ein.

Abbildung 6-12: Zusammenfassung der vi-Kommandos zum Verschieben von Text

Übung 5

- 5-1. Rufen Sie den vi mit dem Namen der Datei **prob2**, die in Übung 2 angelegt wurde, auf.

Ändern Sie den Inhalt von Zeile 8 in **ENDE DER DATEI**.

- 5-2. Kopieren Sie die ersten acht Zeilen der Datei in das Register z. Fügen Sie den Inhalt von Register z nach der letzten Zeile der Datei an.

- 5-3. Ändern Sie den Inhalt von Zeile 8 in **acht ist Macht**.

- 5-4. Stellen Sie den Cursor in die letzte Zeile der Datei. **DATEI** durch **ÜBUNG** ersetzen **DER** durch **EINER** ersetzen

Spezielle Kommandos

Im folgenden werden einige spezielle Kommandos vorgestellt, die Ihnen die Arbeit erleichtern.

- <. > wiederholt das letzte Kommando
- <J > fügt zwei Zeilen zusammen
- <^I > löscht den Bildschirm und baut ihn neu auf
- <~ > wandelt Klein- in Großschreibung um und umgekehrt

Wiederholen des letzten Kommandos

Mit dem Punkt . wird das zuletzt eingegebene Kommando zum Anlegen, Löschen oder Ändern von Text in der Datei wiederholt. Oft wird es zusammen mit dem Suchkommando verwendet.

Angenommen, es wurde versehentlich das r in "Recht haben" groß geschrieben, und dies soll in ein "r" geändert werden. Bei der Wendung "zu Recht" soll das "R" jedoch groß bleiben. Eine Korrekturmöglichkeit besteht darin, das Wort "Recht" zu suchen und, wo es zum erstenmal in der Formulierung "recht haben" vorkommt, "R" in "r" zu ändern. Danach wird die Suche fortgesetzt. Tritt das Wort wieder in dieser Formulierung auf, wird nur ein Punkt eingegeben: vl hat das letzte Kommando gespeichert und ersetzt wieder "R" durch "r".

Üben Sie dieses Kommando. Geben Sie z. B. im Kommando-Modus am Ende eines Satzes einen Punkt ein, erscheint auf dem Bildschirm plötzlich die letzte Textänderung. Verfolgen Sie auf dem Bildschirm, welche Auswirkungen dies auf den Text hat.

Zusammenfügen von zwei Zeilen

Mit dem Kommando <J > werden Zeilen zusammengefügt. Dazu wird der Cursor in die gewünschte Zeile gestellt, und die Tasten SHIFT und j werden gleichzeitig gedrückt. Die nachfolgende Zeile wird an die aktuelle angefügt.

Angenommen, es sind die folgenden beiden Zeilen vorhanden:

```
Sehr geehrter Herr
Schmidt!
```

Um diese beiden Zeilen zu einer zu verbinden, wird der Cursor auf ein Zeichen in der ersten Zeile gestellt und folgendes eingegeben:

```
<J>
```

Sofort erscheint auf dem Bildschirm:

```
Sehr geehrter Herr Schmidt!
```

vi setzt automatisch ein Leerzeichen zwischen das letzte Wort der ersten Zeile und das erste Wort der zweiten Zeile.

Löschen und Wiederaufbauen des Bildschirms

Schickt ein UNIX-Benutzer eine Nachricht an einen anderen Benutzer, der gerade mit **vi** arbeitet, erscheint sie im aktuellen Fenster und überdeckt einen Teil des bearbeiteten Textes. Damit der Benutzer seinen Text wiederherstellen kann, wenn er die Nachricht gelesen hat, muß er sich im Kommando-Modus befinden. (Im Textmodus muß man die Taste ESCAPE drücken und zum Kommando-Modus zurückkehren.) Dort **<^I>** (CONTROL-I) eingeben. **vi** löscht die Nachricht und baut das Fenster wie vorher auf.

Umwandlung von Klein- in Großschreibung und umgekehrt

Um einen Kleinbuchstaben rasch in einen Großbuchstaben umzuwandeln oder umgekehrt, wird der Cursor auf den zu ändernden Buchstaben gestellt und eine Tilde **<~>** eingegeben. Mit **~** wird z. B. "a" in "A" umgewandelt. Mehrere Buchstaben werden durch mehrmalige Eingabe von **~** geändert, aber es ist nicht möglich, vor dem Kommando eine Zahl einzugeben, um mehrere Buchstaben auf einmal umzuwandeln.

In Abbildung 6-13 sind diese speziellen Kommandos zusammengefaßt.

Kommando	Funktion
<. >	Wiederholt das letzte Kommando.
<J >	Fügt die Zeile unter der aktuellen Zeile an diese an.
<^I >	Löscht den Bildschirm und baut ihn neu auf.
<~ >	Wandelt Klein- in Großbuchstaben um oder umgekehrt.

Abbildung 6-13: Zusammenfassung spezieller Kommandos

Verwendung von Kommandos des Zeileneditors in vi

Der Editor **vi** kann zahlreiche Kommandos des Zeileneditors **ex** ebenfalls verwenden. Eine vollständige Liste der **ex**-Kommandos finden sie unter **ex(1)** im *User's Reference Manual*. In diesem Abschnitt werden einige der am häufigsten verwendeten Kommandos erklärt.

Die **ex**-Kommandos sind den **ed**-Kommandos, die in Kapitel 5 erläutert wurden sehr ähnlich. Wenn Sie mit **ed** vertraut sind, können Sie in einer Testdatei ausprobieren, welche **ed**-Kommandos auch in **vi** verwendet werden können.

Zeileneditor-Kommandos beginnen mit einem Doppelpunkt **:**. Nach der Eingabe des Doppelpunktes springt der Cursor zum unteren Bildschirmrand, und der Doppelpunkt wird angezeigt. Auch die übrigen Eingaben zu dem Kommando erscheinen an dieser Stelle.

Zeitweilige Rückkehr zur Shell: Die Kommandos **:sh** und **:!**

Beim Starten von **vi** füllt der Pufferinhalt den Bildschirm, so daß keine Shell-Kommandos mehr abgesetzt werden können. Dies ist jedoch bisweilen erforderlich. Sie wollen beispielsweise Informationen aus einer anderen Datei in den aktuellen Text übernehmen. Dazu könnte man mit einem Shell-Kommando wie **cat** oder **pg** den Text einer Datei auf dem Bildschirm anzeigen lassen. Den Editor zu verlassen und wieder aufzurufen ist jedoch umständlich und kostet Zeit. **vi** bietet zwei Methoden, den Editor zeitweilig zu verlassen, um Shell-Kommandos abzusetzen oder andere Dateien zu bearbeiten, ohne den Puffer verlassen zu müssen: die Kommandos **!** und **:sh**.

Mit dem Kommando **!** kann man den Editor verlassen und in einer Kommandozeile ein Shell-Kommando eingeben. Geben Sie dazu im Kommando-Modus von **vi** **!** ein. Diese Zeichen erscheinen am unteren Bildschirmrand. Direkt nach dem **!** das Shell-Kommando eingeben. Die Shell aktiviert das Kommando, gibt das Ergebnis aus und zeigt folgende Meldung an: [Hit return to continue]. Wird die Taste RETURN gedrückt, erneuert **vi** den Bildschirm, und der Cursor steht an derselben Stelle wie vor dem Verlassen des Editors.

Die gleiche Funktion hat auch das `ex`-Kommando `:sh`, die Bildschirmanzeige ist jedoch anders. Geben Sie im Kommando-Modus von `vi` `:sh` ein und drücken Sie die Taste RETURN. In der nächsten Zeile erscheint daraufhin ein Bereit-Zeichen der Shell. Wie bei der normalen Arbeit mit der Shell das benötigte Kommando/die Kommandos nach dem Bereit-Zeichen eingeben. Um zu `vi` zurückzukehren, geben Sie `<^d>` oder `exit` ein; der Bildschirm wird erneuert, und der Cursor steht an derselben Stelle wie zuvor.

Auch wenn beim zeitweiligen Arbeiten in der Shell ein anderes Verzeichnis aktiviert wird, kann man anschließend mit `exit` oder `<^d>` in den `vi`-Puffer zurückkehren, in dem die Datei editiert wird.

Eingabe in eine neue Datei: Das Kommando `:w`

Mit dem Kommando `:w` (write = schreiben) kann eine Datei angelegt werden, indem Zeilen aus der Datei, die gerade bearbeitet wird, in eine angegebene Datei kopiert werden. Zum Anlegen der neuen Datei muß eine Zeile oder ein Zeilenbereich (mit Zeilennummern) zusammen mit dem Namen der neuen Datei in die Kommandozeile eingeben. Die Zeilenzahl unterliegt keiner Beschränkung. Das allgemeine Format lautet:

```
:Zeilennummer[,Zeilennummer]w dateiname
```

Soll z. B. die dritte Zeile des Puffers in eine Datei namens `drei` gestellt werden, geben Sie folgendes ein:

```
:3w drei<CR>
```

Wurde die neue Datei angelegt, gibt `vi` die folgende Meldung aus:

```
"drei" [New file] 1 line, 20 characters
```

Soll die aktuelle Zeile in eine Datei geschrieben werden, kann als Zeilenadresse ein Punkt `.` angegeben werden:

```
.:w schrott<CR>
```

Eine neue Datei `schrott` wird angelegt. Sie enthält nur die aktuelle Zeile des `vi`-Puffers.

Wird ein Bereich von Zeilen angegeben, kann auch ein ganzer Abschnitt des Puffers in eine neue Datei gestellt werden. Um beispielsweise die Zeilen 23 bis 37 in eine Datei zu schreiben, wird folgendes eingegeben:

```
:23,37w neue_datei<CR>
```

Suchen der Zeilennummer

Die Zeilennummer einer Zeile wird bestimmt, indem der Cursor in die Zeile gestellt und ein Doppelpunkt `:` eingegeben wird. Dieser erscheint unten am Bildschirm. Dahinter `:=` eingeben und die Taste RETURN drücken.

Um die Nummer dieser Zeile
abzufragen, `:=` <CR> eingeben.

`:=`

6

Sobald die Taste RETURN gedrückt wird, erscheint in der untersten Bildschirmzeile anstelle der Kommandozeile die Nummer der aktuellen Zeile im Puffer.

Um die Nummer der aktuellen Zeile abzufragen, `:.<CR>` eingeben.

34

Der Cursor kann in eine beliebige Zeile im Puffer gestellt werden, wenn : und die Zeilennummer eingegeben wird. Mit der Kommandozeile

`n<CR>`

geht man zur *n*ten Zeile des Puffers.

Den restlichen Pufferinhalt löschen

Soll der Text zwischen der aktuellen Zeile und dem Ende des Puffers gelöscht werden, ist es am einfachsten, das Kommando `d` des Zeileneditors mit den Sondersymbolen für aktuelle und letzte Zeile zu verwenden.

`:$d<CR>`

. steht für die aktuelle, das Dollarzeichen \$ für die letzte Zeile.

Aufnehmen einer Datei in den Puffer

Das Kommando `:r` (read = lesen) dient dazu, Text aus einer anderen Datei unterhalb einer angegebenen Zeile im Editierpuffer einzufügen. Um beispielsweise den Inhalt einer Datei namens **daten** in die aktuelle Datei zu stellen, wird der Cursor eine Zeile über der Stelle positioniert, an der der Text erscheinen soll.

`:r daten<CR>`

eingeben. Anstatt den Cursor zu bewegen, kann man auch die Zeilennummer angeben. Soll z. B. die Datei **daten** nach Zeile 56 des Puffers folgen,

:56r daten<CR>

Auch hier gilt wieder: Keine Angst vor Experimenten; auch ex-Kommandos können mit dem Kommando **<u>** rückgängig gemacht werden.

Globale Änderungen

Eines der leistungsfähigsten Kommandos in ex ist das Global-Kommando. Dieses Kommando wird hier für Benutzer angegeben, die den Zeileneditor kennen. Aber auch wer nicht mit diesem Editor vertraut ist, kann das Kommando in einer Testdatei ausprobieren.

Angenommen, es geht um einige Seiten eines Textes über das DNS-Molekül, dessen Struktur als Helix beschrieben wird. Nun soll durchgehend das Wort Helix in Doppelhelix geändert werden. Durch das Global-Kommando des Editors ex kann dies mit einer einzigen Kommandozeile erreicht werden. Dazu müssen Sie jedoch einige Kommandos kennen:

:g/Muster/Kommando<CR>

In jeder Zeile, die das *Muster* enthält, wird das ex-Kommando *Kommando* ausgeführt. Bei der Eingabe **:g/Helix<CR>** werden beispielsweise alle Zeilen ausgegeben, in denen das Suchmuster Helix vorkommt.

:s/Muster/neue_Wörter/<CR>

Dies ist das Kommando zum Ersetzen. Der Zeileneditor sucht in der aktuellen Zeile, wo zum erstenmal die Zeichen *Muster* vorkommen, und ändert sie in *neue_Wörter*.

:s/Muster/neue_Wörter/g<CR>

Wird nach dem letzten Begrenzungszeichen dieser Kommandozeile der Buchstabe "g" angefügt, ändert ex alle *Muster* in der aktuellen Zeile. Andernfalls ändert ex das Suchmuster nur beim ersten Vorkommen.

`:g/Helix/s//Doppelhelix/g<CR>`

Mit dieser Kommandozeile wird nach dem Wort "Helix" gesucht. Bei jedem Vorkommen ersetzt das Kommando "Ersetzen" dieses Wort innerhalb der Zeile durch das Wort "Doppelhelix". Zwischen den Begrenzungszeichen nach dem s braucht "Helix" nicht nochmals eingegeben zu werden. In dem Kommando ist das Wort von den Begrenzungszeichen hinter dem Globalkommando g her gespeichert. Es handelt sich um ein leistungsfähiges Kommando. Eine ausführlichere Erläuterung zu Global- und Ersetzungskommandos enthält Kapitel 5.

In Abbildung 6-14 sind die in vi zur Verfügung stehenden Kommandos des Zeileneditors zusammengefaßt.

Kommando	Funktion
:	Zeigt an, daß nun Kommandos des Zeileneditors folgen.
:sh <CR>	Zeitweilige Rückkehr zur Shell, damit Shell-Kommandos ausgeführt werden können.
<^d>	Rückkehr von der temporären Shell zum aktuellen Bildschirm von vi.
:n <CR>	Stellt den Cursor in die nte Zeile des Puffers.
:x,yw daten <CR>	Schreibt die Zeilen von x bis y in eine neue Datei (daten).
:\$ <CR>	Stellt den Cursor in die letzte Zeile des Puffers.
:\$d <CR>	Löscht alle Zeilen im Puffer zwischen der aktuellen und der letzten Zeile.
:r shell.datei <CR>	Fügt den Inhalt der shell.datei nach der aktuellen Zeile im Puffer ein.
:s/Text/neue_Wörter/<CR>	Ersetzt das erste Vorkommen der Zeichen Text in der aktuellen Zeile durch neue_Wörter.
:s/Text/neue_Wörter/g<CR>	Ersetzt jedes Vorkommen von Text in der aktuellen Zeile durch neue_Wörter.
:g/Text/s/ neue_Wörter/g<CR>	Ersetzt jedes Vorkommen von Text in der Datei durch neue_Wörter.

Abbildung 6-14: Zusammenfassung der Zeileneditor-Kommandos

Verlassen von vi

Zum Verlassen des Editors vi stehen fünf Kommandos zur Verfügung. Kommandos, die mit einem Doppelpunkt (:) eingeleitet werden, sind Kommandos des Zeileneditors.

- | | |
|---|---|
| <code><ZZ></code> oder <code>:wq<CR></code> | Schreibt den Inhalt des vi-Puffers in die gerade bearbeitete UNIX-Datei und verläßt vi. |
| <code>:w dateiname <CR></code>
<code>:q <CR></code> | Schreibt den Inhalt des temporären Puffers in eine neue Datei namens <i>dateiname</i> und verläßt vi. |
| <code>:w! dateiname <CR></code>
<code>:q <CR></code> | Überschreibt eine vorhandene Datei namens <i>dateiname</i> mit dem Inhalt des Puffers und verläßt vi. |
| <code>:q! <CR></code> | Verläßt vi, ohne den Pufferinhalt in eine Datei zu stellen, und löscht alle im Puffer vorgenommenen Änderungen. |
| <code>:q <CR></code> | Verläßt vi, ohne den Pufferinhalt in eine UNIX-Datei zu stellen. Dies ist nur möglich, wenn im Puffer keine Änderungen vorgenommen wurden; ansonsten gibt vi eine Warnung aus, daß der Pufferinhalt gesichert oder zum Verlassen des Editors das Kommando <code>:q! <CR></code> verwendet werden muß. |

Mit dem Kommando `<ZZ>` und der Kommandofolge `:wq` sichert man den Inhalt des Puffers in einer Datei, verläßt vi und kehrt zur Shell zurück. Das Kommando `<ZZ>` wurde bereits geübt. Verlassen Sie nun vi mit dem Kommando `:wq`. In vi ist der Name der gerade bearbeiteten Datei gespeichert, so daß er nicht neu eingegeben werden muß, wenn der Pufferinhalt wieder in die Datei gespeichert wird.

`:wq<CR>`

eingeben. Das System reagiert genauso wie bei dem Kommando `<ZZ>`. Es zeigt den Namen der Datei sowie die Anzahl der Zeilen und Zeichen in der Datei an.

Wie muß man vorgehen, um die Datei umzubenennen? Soll z. B. in eine neue Datei namens **schrott** Text geschrieben werden, geben Sie:

```
:w schrott<CR>
```

ein. Um nach dem Sichern in die neue Datei den vi zu verlassen, geben Sie folgendes ein:

```
:q<CR>
```

Wird versucht, in eine bereits vorhandene Datei zu sichern, wird eine Warnung ausgegeben. Wird beispielsweise versucht, in eine Datei namens **krause** zu sichern, meldet das System:

```
"krause" File exists - use "w! krause" to overwrite
```

Soll der Inhalt der bestehenden Datei durch den Pufferinhalt ersetzt werden, überschreiben Sie mit dem Kommando **:w!** die Datei **krause**.

```
:w! krause<CR>
```

Die neue Datei überschreibt die vorhandene.

Editiert man eine Datei namens **memo**, nimmt einige Änderungen vor und kommt dann zu dem Schluß, daß die Änderungen nicht gespeichert werden sollen, oder falls versehentlich ein Kommando an vi gegeben wurde, das sich nicht rückgängig machen läßt, kann man vi ohne zu speichern verlassen. Dazu

```
:q!<CR>
```

eingeben. In Abbildung 6-15 sind die Kommandos zum Verlassen des Editors zusammengefaßt.

Kommando	Funktion
<ZZ>	Sichert die Datei und verläßt vi.
:wq<CR>	Sichert die Datei und verläßt vi.
:w <i>dateiname</i> <CR> :q<CR>	Sichert den Editierpuffer in eine neue Datei (<i>dateiname</i>) und verläßt vi.
:w! <i>dateiname</i> <CR> :q<CR>	Überschreibt eine vorhandene Datei (<i>dateiname</i>) mit dem Inhalt des Editierpuffers und verläßt vi.
:q!<CR>	Verläßt vi, ohne den Puffer in eine Datei zu sichern.
:q<CR>	Verläßt vi, ohne den Puffer in eine Datei zu sichern.

Abbildung 6-15: Zusammenfassung der Kommandos zum Verlassen des Editors

Sonderoptionen für vi

Zum Kommando vi gibt es einige spezielle Optionen. Damit kann man:

- eine Datei wiederherstellen, die durch eine Unterbrechung des Betriebssystems UNIX verlorenging
- mehrere Dateien in den Editierpuffer stellen und nacheinander bearbeiten
- die Datei mit den Cursor-Steuerkommandos von vi in beliebiger Geschwindigkeit durchgehen

Wiederherstellen einer durch Unterbrechung verlorengegangenen Datei

Bei einer Unterbrechung oder einem Stromausfall verläßt das System den Editor vi, ohne den Text im Puffer in die entsprechende Datei zurückzuspeichern. Das Betriebssystem UNIX speichert jedoch eine Kopie des Puffers. Meldet man sich wieder in UNIX an, kann man die Datei mit der Option -r zum Kommando vi wiederherstellen. Dazu

```
vi -r dateiname<CR>
```

eingeben. Die Änderungen, die vor der Unterbrechung in *dateiname* vorgenommen wurden, stehen nun im vi-Puffer. Man kann nun die Datei weiter editieren oder die Datei sichern und vi verlassen. Der Editor vi speichert den Dateinamen und sichert in diese Datei.

Editieren mehrerer Dateien

Sollen in einer Editiersitzung mehrere Dateien bearbeitet werden, wird das Kommando vi zusammen mit den gewünschten Dateinamen eingegeben. Folgendes ist einzugeben:

```
vi datei1 datei2<CR>
```

vi meldet daraufhin, wieviele Dateien editiert werden. Zum Beispiel:

```
2 files to edit
```

Ist die erste Datei editiert, sind die im Puffer befindlichen Änderungen in die Datei (*datei1*) zu sichern. Dazu ist

:w<CR>

einzugeben. Auf das Kommando **:w <CR>** hin erscheint unten am Bildschirm eine Meldung mit dem Namen der Datei und der Anzahl der Zeilen und Zeichen in der Datei. Nun kann die nächste Datei mit dem Kommando **:n** in den Editierpuffer geholt werden. Dazu ist

:n<CR>

einzugeben. Daraufhin erscheint unten am Bildschirm eine Meldung mit der Angabe des Namens der nächsten zu bearbeitenden Datei sowie der Anzahl der Zeilen und Zeichen in dieser Datei.

Wählen Sie zwei Dateien aus dem aktuellen Verzeichnis aus. Rufen Sie nun **vi** auf und laden die beiden Dateien gleichzeitig in den Editierpuffer. Achten Sie auf die Meldungen des Systems zu den eingegebenen Kommandos.

Anzeigen einer Datei

Oft ist es günstig, eine Datei mit Hilfe der leistungsstarken Such- und Blätterfunktionen von **vi** durchzugehen. Andererseits möchte man verhindern, daß man die Datei während einer Editiersitzung versehentlich ändert. Mit der Option für Lesezugriff ist die Datei vor solchen unerwünschten Änderungen geschützt. Um dies zu erreichen, wird diese Option aktiviert, indem der Editor als **view** anstatt als **vi** aufgerufen wird.

In Abbildung 6-16 sind die Sonderoptionen für **vi** zusammengefaßt.

Option	Funktion
<code>vi datei1 datei2 datei3 <CR></code>	Stellt drei Dateien (<i>datei1</i> , <i>datei2</i> und <i>datei3</i>) zum Editieren in den vi-Puffer.
<code>:w <CR></code> <code>:n <CR></code>	Sichert die aktuelle Datei und holt die nächste Datei in den Puffer.
<code>vi -r datei1 <CR></code>	Stellt die Änderungen in <i>datei1</i> wieder her.

Abbildung 6-16: Zusammenfassung der Sonderoptionen für vi

Übung 6

- 6-1. Stellen Sie eine durch Unterbrechung verloren gegangene Datei wieder her.

Rufen Sie **vi** auf und geben in einer Datei namens **prob6** Text ein. Schalten Sie das Terminal aus, ohne zu sichern oder den **vi** zu verlassen. Schalten Sie das Terminal wieder ein und melden Sie sich wieder an. Versuchen Sie, den **vi** wieder auf und **prob6** zu editieren.

- 6-2. Stellen Sie die Dateien **prob1** und **prob2** zum Editieren in den **vi**-Puffer. Sichern Sie **prob1** und holen Sie die nächste Datei, **prob2**, in den Puffer.

Sichern Sie **prob2** in eine Datei **schrott**.

Verlassen Sie den **vi**.

- 6-3. Probieren Sie folgendes Kommando aus:

vi prob*<CR>

Was geschieht? Versuchen Sie, alle Dateien so schnell wie möglich zu verlassen.

- 6-4. **prob4** im Lesemodus durchgehen.

Vorwärts blättern.

Abwärts blättern.

Zurück blättern.

Aufwärts blättern.

Verlassen Sie den Editor und kehren Sie zur Shell zurück.

Lösungen zu den Übungen

Oft können in vi Aufgaben auf verschiedene Arten gelöst werden. Alle Methoden, die zum gewünschten Ergebnis führen, sind richtig. Die folgenden Lösungen sind daher nicht die einzig gültigen Möglichkeiten.

Übung 1

- 1-1. Erfragen Sie beim Systemverwalter den Systemnamen des Terminals.
Geben Sie folgendes ein:

```
TERM=terminalname <CR>
```

- 1-2. Geben Sie das Kommando vi für eine Datei namens prob1 ein:

```
vi prob1 <CR>
```

Geben Sie anschließend das Kommando "Anhängen" (<a>) und den folgenden Text in die Datei ein:

```
Das ist eine Übung! <CR>  
Auf, ab, <CR>  
links, rechts, <CR>  
Terminal-Training <CR>  
Bit für Bit <ESC>
```

- 1-3. Verwenden Sie die Kommandos <k> und <h>.
1-4. Verwenden Sie das Kommando <x>.
1-5. Verwenden Sie die Kommandos <j> und <l>.

- 1-6. Rufen Sie vi auf und geben im Modus "Anhängen" (<a>) folgenden Text ein:

und Byte für Byte<ESC>

Stellen Sie anschließend mit <J> und <I> den Cursor auf das letzte Zeichen der letzten Zeile der Datei. Geben Sie erneut mit dem Kommando <a> Text ein. Legen Sie eine neue Zeile wird mit der Taste RETURN an. Drücken Sie zum Verlassen des Textmodus die Taste ESCAPE.

- 1-7. Geben Sie folgendes ein:

<ZZ>

- 1-8. Geben Sie folgendes ein:

vi prob1<CR>

Achten Sie auf die Antwort des Systems:

"prob1" 7 lines, 93 characters

Übung 2

- 2-1. Eingabe:

```
vi prob2<CR>
<a>1<CR>
2<CR>
3<CR>
.
.
.
48<CR>
49<CR>
50<ESC>
```

2-2. Eingabe:

<^f>
<^b>
<^u>
<^d>

Achten Sie beim Bildschirmwechsel auf die Zeilennummern.

2-3. Eingabe:

<G>
<o>
123456789 123456789<ESC>
<7h>
<3l>

Die Eingabe <7h> stellt den Cursor
auf die 2 der zweiten Zahlenreihe.
Bei Eingabe <3l> springt der Cursor
auf die 5 in der zweiten Zahlenreihe.

2-4. \$ = Zeilenende
0 = erstes Zeichen der Zeile

2-5. Eingabe:

<^>
<w>

<e>

2-6. Eingabe:

<1G>
<M>
<L>
<H>

2-7. Eingabe:

/8
<n>
/48

Übung 3

- 3-1. Eingabe:
vi prob3 <CR>
- 3-2. Eingabe:
<a> Text anhängen <CR>
Text einfügen <CR>
Der Computer übernimmt <CR>
Routine Aufgaben. <ESC>
- 3-3. Eingabe:
<O>
und Bilanzen erstellen <ESC>
- 3-4. Eingabe:
<3G>
<i>Text löschen <CR> <ESC>

Der Text in der Datei lautet nun:

```
Text anhängen
Text einfügen
Text löschen
Der Computer übernimmt
und Bilanzen erstellen
Routineaufgaben.
```

- 3-5. Die aktuelle Zeile ist Der Computer übernimmt. Um eine neue Textzeile darunter anzulegen, dient das Kommando <o>.
- 3-6. Die aktuelle Zeile ist Budgetierung.
Mit <G> stellt man den Cursor in die unterste Zeile.
Mit <A> kann man am Zeilenende mit dem Anhängen beginnen.
Mit <CR> wird eine neue Zeile angelegt.
Fügen Sie den folgenden Satz ein: **Aber Computer sind interessante Maschinen.**
Mit <ESC> verläßt man den Modus "Anhängen".

- 3-7. Eingabe:
 <1G>
 /Text
 <i>Etwas <Leertaste> <ESC>
- 3-8. Sichern Sie mit <ZZ> den Pufferinhalt in prob3 und kehren zur Shell zurück.

Übung 4

- 4-1. Eingabe:
 vi prob4<CR>
 <a> Im Haushalt gibt es<CR>
 viele langweilige, immer wiederkehrende<CR>
 Arbeiten, daher sollte man einen<CR>
 Roboter dafür anschaffen.<ESC>
- 4-2. Eingabe:
 <2G>
 <A> mühsame und unangenehme<8BACKSPACE><CR>
 <ESC>

Drücken Sie <h> so oft, bis das l von langweilige erreicht ist.
 Geben Sie dann folgendes ein:
 <dw>. (Auch <6x> ist möglich.)

- 4-3. Der Cursor steht in der zweiten Zeile. Eingabe:
 <2j>
 <I> vollautomatischen, elektronischen<ESC>
 <dd>

Um alle Zeichen einer Zeile, nicht aber die Zeile selbst zu löschen,
 geben Sie folgendes ein:

<0> (Mit null bewegt sich der Cursor zum Zeilenanfang)
 <D>

<H>
 <3dd>

- 4-4. Sichern und verlassen Sie den vi.

`<ZZ>`

Löschen Sie die Datei.

`rm prob4<CR>`

Übung 5

- 5-1. Eingabe:

`vi prob2<CR>`
`<8G>`
`<cc> ENDE DER DATEI <ESC>`

- 5-2. Eingabe:

`<1G>`
`<8"zyy>`
`<G>`
`<"zp>`

- 5-3. Eingabe:

`<8G>`
`<cc> acht ist Macht<ESC>`

- 5-4. Eingabe:

`<G>`
`<2w>`
`<cw>`
`ÜBUNG<ESC>`
`<2b>`
`<cw>`
`EINER<ESC>`

Übung 6

6-1. Eingabe:

```
vi prob6 <CR>  
<a> (mehrere Textzeilen anhängen)  
<ESC>
```

Schalten Sie das Terminal aus.

Schalten Sie das Terminal ein.

Melden Sie sich im UNIX-System an. Eingabe:

```
vi -r prob6 <CR>  
:wq <CR>
```

6-2. Eingabe:

```
vi prob1 prob2 <CR>  
:w <CR>  
:n <CR>
```

```
:w schrott <CR>  
<ZZ>
```

6-3. Eingabe:

```
vi prob* <CR>
```

(Meldung:)

8 files to edit (vi ruft alle Dateien auf, deren Namen mit prob beginnen.)

```
<ZZ>  
<ZZ>
```

6-4. Eingabe:

```
view prob4 <CR>  
<^f>  
<^d>  
<^b>  
<^u>  
:q <CR>
```

Kapitel 7: ANLEITUNG ZUR SHELL

Einführung	7-1
Die Shell-Kommandosprache	7-2
Maskierungszeichen	7-4
Das Maskierungszeichen für alle Zeichen: der Stern (*)	7-4
Das Maskierungszeichen für ein Zeichen: das Fragezeichen (?)	7-7
Korrektur von Tippfehlern mit * oder ?	7-8
Maskierungszeichen für ein Zeichen in einer Menge:	
eckige Klammern ([])	7-9
Sonderzeichen	7-10
Kommando im Hintergrund ausführen: das kaufmännische und (&)	7-10
Ausführen von Kommandos in Folge: das Semikolon (;)	7-11
Sonderbedeutungen aufheben: der Backslash (\)	7-12
Sonderbedeutungen aufheben: Anführungszeichen	7-13
Anführungszeichen zum Aufheben der Sonderbedeutung des Leerzeichens	7-13
Umlenkung von Eingabe und Ausgabe	7-15
Umlenkung der Eingabe: das Zeichen <	7-16
Umlenkung der Ausgabe in eine Datei: das Zeichen >	7-16
Anhängen der Ausgabe an eine bestehende Datei: das Symbol >>	7-17
Nützliche Anwendungsmöglichkeiten für die Ausgabeumlenkung	7-18
Kombination von Hintergrundmodus und Ausgabeumlenkung	7-21
Umlenkung der Ausgabe in ein Kommando: die Pipe ()	7-22
Pipeline mit den Kommandos cut und date	7-23
Ausgabe als Argument einsetzen	7-28
Aufrufen und Beenden von Prozessen	7-29
Programme mit den Kommandos batch und at zur späteren Ausführung vormerken	7-29

Abrufen des Status laufender Prozesse	7-35
Abbrechen aktiver Prozesse	7-36
Das Kommando nohup	7-37
Übungen zur Kommandosprache	7-39
Shell-Programmierung	7-40
Shell-Programme	7-41
Erstellen einfacher Shell-Programme	7-41
Aufrufen von Shell-Programmen	7-42
Verzeichnis bin für ausführbare Dateien anlegen	7-43
Hinweise zur Benennung von Shell-Programmen	7-44
Variablen	7-45
Positionsparameter	7-46
Sonderparameter	7-50
Benannte Variablen	7-55
Variablenwerte zuweisen	7-56
Shell-Programmstrukturen	7-65
Kommentare	7-65
Das Here Document	7-66
Aufrufen des Editors ed in einem Shell-Programm	7-69
Rückgabecodes	7-72
Schleifen	7-73
Der Papierkorb der Shell: /dev/null	7-79
Bedingte Programmstrukturen	7-79
Unbedingte Steueranweisungen: die Kommandos break und continue	7-91
Debugger-Programme	7-92
Anpassen der Benutzerumgebung	7-97
Einfügen von Kommandos in die Datei .profile	7-97

Definition der Terminalcharakteristika	7-98
Anlegen eines Verzeichnisses <code>rje</code>	7-100
Verwendung von Shell-Variablen	7-100
Übungen zur Shell-Programmierung	7-103
Lösungen zu den Übungen	7-105
Übungen zur Kommandosprache	7-105
Übungen zur Shell-Programmierung	7-106

Einführung

In diesem Abschnitt wird beschrieben, wie man die Shell des Betriebssystems UNIX für die Durchführung von Routineaufgaben einsetzt. Beispielsweise wird erläutert, wie man mit der Shell Dateien verwaltet, den Inhalt von Dateien ändert und Kommandos zu Programmen zusammenstellt, die die Shell anschließend ausführen kann.

Dieses Kapitel besteht aus zwei Hauptabschnitten. Im ersten dieser Abschnitte, "Die Shell-Kommandosprache", wird die Funktion der Shell als Kommandointerpreter ausführlich beschrieben. Dabei wird die Verwendung der Shell-Kommandos und der Zeichen mit Sonderbedeutungen zur Verwaltung von Dateien, zum Umlenken der Standardeingabe und -ausgabe, zum Aufrufen und Beenden von Verarbeitungsvorgängen beschrieben. Im zweiten Hauptabschnitt, "Shell-Programmierung", wird die Funktion der Shell als Programmiersprache ausführlich beschrieben. Es wird beschrieben, wie man Programme, die aus Kommandos, Variablen und Programmstrukturen wie Schleifen und Fallanweisungen bestehen, erstellt, ausführt und Fehler in ihnen korrigiert. Schließlich wird noch beschrieben, wie man die Benutzerumgebung bei der Anmeldung verändert.

In diesem Kapitel sind viele Beispiele enthalten. Der Benutzer sollte sich am UNIX-System anmelden und die Beispiele praktisch nachvollziehen, während er diesen Text durcharbeitet. Wie bei den anderen Beispielen dieses Handbuchs werden Eingaben des Benutzers von den Ausgaben des UNIX-Systems durch unterschiedlichen Druck (**fett**, *kursiv* oder nicht proportional) unterschieden. Nähere Angaben dazu sind im Vorwort unter "Notationskonventionen" enthalten.

Zusätzlich zu den Beispielen befinden sich am Ende der Hauptabschnitte "Die Shell-Kommandosprache" und "Shell-Programmierung" einige Übungen. Diese Übungen sollen das Verständnis der behandelten Themen vertiefen. Die Lösungen zu den Übungen befinden sich am Ende dieses Kapitels.

NOTE

Bei einem bestimmten UNIX-System stehen vielleicht nicht alle in diesem Kapitel aufgeführten Kommandos zur Verfügung. Kann ein Kommando nicht aufgerufen werden, sollte man sich beim Systemverwalter informieren.

Um einen ersten Überblick über die Funktionen der Shell als Kommandointerpreter und als Programmiersprache zu erhalten, empfiehlt es sich Kapitel 1 und 4 sowie in Anhang E, "Übersicht über die Shell-Kommandosprache" zu lesen.

Die Shell-Kommandosprache

In diesem Abschnitt werden Kommandos und, was noch wichtiger ist, bestimmte Zeichen mit Sonderbedeutungen eingeführt, mit denen man folgende Aufgaben durchführen kann:

- Suchen und Bearbeiten von Gruppen von Dateien unter Verwendung von Zeichenmustern
- Ausführen eines Kommandos im Hintergrund oder zu einem bestimmten Zeitpunkt
- Ausführen einer Reihe von Kommandos nacheinander
- Umlenken der Standardeingabe und -ausgabe in Dateien/Kommandos und aus Dateien/Kommandos
- Beenden von Prozessen

Zuerst werden die Zeichen behandelt, die für die Shell eine Sonderbedeutung haben, danach werden die Kommandos und die Syntax zum Ausführen der vorgenannten Aufgaben beschrieben. Die in diesem Kapitel behandelten Zeichen mit Sonderbedeutungen sind in Abbildung 7-1 zu einer Übersicht zusammengefaßt.

Zeichen	Funktion
* ? []	Maskierungszeichen zur Abkürzung von Dateinamen mittels Zeichenmustern
&	Damit werden Kommandos in den Hintergrundmodus gesetzt, so daß das Terminal für andere Aufgaben verwendet werden kann.
;	Trennen mehrerer Kommandos in einer Kommandozeile
\	Aufheben der Sonderbedeutungen der Sonderzeichen, wie z. B. *, ?, [], &, ;, >, < und .
'...'	Mit einfachen Anführungszeichen werden die Bedeutung "Begrenzungszeichen" eines Leerzeichens und die Sonderbedeutungen aller Sonderzeichen aufgehoben.
"..."	Mit doppelten Anführungszeichen werden die Bedeutung "Begrenzungszeichen" eines Leerzeichens und die Sonderbedeutungen aller Sonderzeichen <i>außer</i> \$ und ` aufgehoben.
>	Umlenken der Ausgabe eines Kommandos in eine Datei (Inhalt wird überschrieben)
<	Die Eingabe für ein Kommando kommt aus einer Datei
>>	Umlenken der Ausgabe eines Kommandos in eine Datei (Anhängen ans Ende der Datei)
	Anlegen einer Pipe, so daß die Ausgabe eines Kommandos zur Eingabe eines anderen Kommandos wird
`...`	Der Gravis ermöglicht die direkte Verwendung der Ausgabe eines Kommandos als Argument in einer Kommandozeile.
\$	Wird für Positionsparameter und benutzerdefinierte Variablen verwendet; Bereit-Zeichen der Shell.

Abbildung 7-1: Zeichen mit Sonderbedeutungen in der Shell-Sprache

Maskierungszeichen

Maskierungszeichen gehören zu den Sonderzeichen und stehen für andere Zeichen. Man nennt sie Globalzeichen, weil sie global für andere Zeichen eingesetzt werden können. Hier werden die Maskierungszeichen * (Stern), ? (Fragezeichen) und [] (eckige Klammer) behandelt.

Diese Zeichen werden für Dateinamen oder Teile davon eingesetzt; damit vereinfacht man die Angabe von Dateien oder Gruppen von Dateien als Argumente zu Kommandos. Dabei müssen die Dateien, die dem aus diesen Maskierungszeichen gebildeten Muster entsprechen, bereits vorhanden sein. Man kann damit beispielsweise alle Dateinamen angeben, die den Buchstaben "a" enthalten, alle Dateinamen, die aus fünf Buchstaben bestehen, usw.

Das Maskierungszeichen für alle Zeichen: der Stern (*)

Der Stern (*) steht für jede Zeichenkette, einschließlich einer leeren Zeichenkette. Mit dem Stern (*) kann man einen vollständigen Dateinamen oder einen Teil davon angeben. Der * alleine bezieht sich auf alle Datei- und Verzeichnisnamen im aktuellen Verzeichnis. Das folgende Beispiel zeigt die Wirkung des Sterns (*). Er wird als Argument zum Kommando `echo(1)` angegeben:


```
echo *  
<CR>
```

Beim Kommando `echo` werden die angegebenen Argumente auf dem Bildschirm angezeigt. Die Reaktion des Systems auf das Kommando `echo *` ist eine Liste aller Dateinamen des aktuellen Verzeichnisses. Die Dateinamen erscheinen jedoch im Unterschied zum Kommando `ls` in mehreren Spalten über die ganze Bildschirmbreite anstatt in einer vertikalen Spalte.

In Abbildung 7-2 werden die Syntax und die Funktionen des Kommandos `echo` zusammengefaßt.

Kommandoübersicht		
echo – Die angegebenen Argumente ausgeben		
Kommando	Optionen	Argumente
echo	keine	beliebige(s) Zeichen
Beschreibung:	Mit echo werden die mit dem Kommando angegebenen Argumente in die Ausgabe geschrieben; sie sind durch Leerzeichen zu trennen und mit <CR> abzuschließen.	
Bemerkungen:	Bei der Programmierung mit der Shell wird echo dazu verwendet, Anweisungen auszugeben, Wörter oder Daten in eine Datei umzuleiten und Daten über eine Pipe in ein Kommando einzugeben. Diese Funktionen werden weiter unten in diesem Kapitel ausführlicher beschrieben.	

Abbildung 7-2: Übersicht über das Kommando echo

 Der * ist ein äußerst wirkungsvolles Zeichen. Gibt man beispielsweise **rm *** ein, löscht man damit alle Dateien des aktuellen Verzeichnisses. Daher ist es nur mit Vorsicht einzusetzen!

Hat man beispielsweise mehrere Berichte geschrieben und ihnen die Namen **bericht**, **bericht1**, **bericht1a**, **bericht1b.01**, **bericht25** und **bericht316** zugeordnet, kann man durch die Angabe **bericht1*** global auf alle Dateien Bezug nehmen, die zu **bericht1** gehören. Will man feststellen, wieviele Berichte man geschrieben hat, kann man mit dem Kommando **ls** alle Dateien, die mit der Zeichenkette "bericht" beginnen, auflisten lassen:

```
$ ls bericht*<CR>
bericht
bericht1
bericht1a
bericht1b.01
bericht25
bericht316
$
```

Der * steht für beliebige Zeichen oder kein Zeichen nach der Zeichenkette "bericht". Hier ist zu beachten, daß durch * die Dateien in numerischer und alphabetischer Reihenfolge geordnet werden. Man kann sich durch folgendes Kommando den Inhalt aller Berichtsdateien schnell und einfach ausgeben lassen:

```
pr bericht*<CR>
```

Wählen Sie für das nächste Beispiel ein Zeichen aus, das alle Dateinamen des aktuellen Verzeichnisses gemeinsam haben, wie z. B. ein kleines "a". Lassen Sie sich diese Dateien auflisten lassen, indem Sie nur auf dieses Zeichen Bezug nehmen. Hier ein Beispiel mit "a":

```
ls *a*<CR>
```

Das System gibt dann die Namen aller Dateien im aktuellen Verzeichnis aus, die ein kleines "a" enthalten.

Der * kann für Zeichen an beliebigen Stellen des Dateinamens eingesetzt werden. Ist beispielsweise bekannt, daß mehrere Dateien den ersten und den letzten Buchstaben gemeinsam haben, kann man eine Liste dieser Dateien nach diesem Kriterium anzeigen lassen:

```
ls F*E<CR>
```

Das System gibt in diesem Fall eine Liste aller Dateinamen aus, die mit F beginnen und mit E enden, und zwar in der folgenden Reihenfolge:

F123E
FATE
FE
Fig3.4E

Die Reihenfolge wird von der ASCII-Sortierfolge bestimmt:

1. Zahlen
2. Großbuchstaben
3. Kleinbuchstaben.

Das Maskierungszeichen für ein Zeichen: das Fragezeichen (?)

Das Fragezeichen (?) kann für ein beliebiges Einzelzeichen eines Dateinamens eingesetzt werden. Hat man beispielsweise mehrere Kapitel eines Buches geschrieben, das 12 Kapitel umfaßt, und will man sich eine Liste der Dateien bis Kapitel 9 anzeigen lassen, kann man das Kommando `ls` mit dem Fragezeichen ? und der Zeichenkette "kapitel" verwenden, wie im folgenden Beispiel:

```
$ ls kapitel? <CR>
kapi tel1
kapi tel2
kapi tel5
kapi tel9
$
```

Das System gibt daraufhin eine Liste aller auf dieses Muster passenden Dateinamen aus.

Das Fragezeichen ? steht zwar nur für ein Zeichen, kann jedoch in einem Dateinamen mehrmals verwendet werden. Die übrigen Kapitel des erwähnten Buches kann man beispielsweise mit folgender Eingabe anzeigen lassen:

```
ls kapitel?? <CR>
```

Sollen die Dateien aller Kapitel in diesem Verzeichnis angezeigt werden, kann man natürlich das Maskierungszeichen `*` verwenden:

```
ls kapitel*
```

Korrektur von Tippfehlern mit `*` oder `?`

Es kann vorkommen, daß man beim Verschieben einer Datei mit dem Kommando `mv(1)` versehentlich ein Zeichen in den Dateinamen eingibt, das nicht auf dem Bildschirm angezeigt wird. Das System behandelt dieses nicht druckbare Zeichen als Teil des Dateinamens; es muß daher auch später wieder mit angegeben werden, wenn man die Datei aufruft. Andernfalls erscheint eine Fehlermeldung. Mit `*` oder `?` kann man den Dateinamen mit dem nicht druckbaren Zeichen erfassen und dann umbenennen.

Probieren Sie folgendes Beispiel aus:

1. Legen Sie eine kurze Datei unter dem Namen `versuch` an.
2. Eingabe: `mv versuch versuch<^g>1<CR>`

(Das Steuerzeichen `<^g>` gibt man ein, indem man die CONTROL-Taste niederhält, während man die Taste `g` drückt).

3. Eingabe: `ls versuch1<CR>`

Das System gibt daraufhin eine Fehlermeldung aus:

```
$ ls versuch1<CR>
versuch1: no such file or directory
$
```

4. Eingabe: `ls versuch?1<CR>`

Das System gibt den Dateinamen `versuch1` aus (einschließlich des nicht druckbaren Zeichens) und zeigt damit an, daß diese Datei existiert. Mit dem Fragezeichen (`?`) können Sie nun auch den Dateinamen korrigieren:

```
$ mv versuch?1 versuch1<CR>
$ ls versuch1<CR>
versuch1
$
```

Maskierungszeichen für ein Zeichen in einer Menge: eckige Klammern ([])

Man verwendet eckige Klammern ([]), wenn das Muster auf eines von mehreren möglichen Zeichen zutreffen soll, die sich an einer bestimmten Stelle des Dateinamens befinden. Gibt man beispielsweise an einer Stelle des Suchmusters [crf] ein, sucht die Shell nach den Dateinamen, die den Buchstaben "c", den Buchstaben "r" oder den Buchstaben "f" an der angegebenen Stelle enthalten, wie im folgenden Beispiel gezeigt:

```
$ ls [kpv]ater <CR>
kater
pater
vater
$
```

Mit diesem Kommando werden alle Dateinamen angezeigt, die mit einem der Buchstaben "k", "p" oder "v" beginnen und auf "ater" enden. Zeichen, die in dieser Form in Klammern gruppiert werden können, werden zusammen eine "Zeichenklasse" genannt.

Mit eckigen Klammern kann man auch einen Bereich von Zeichen angeben, unabhängig davon, ob es Zahlen oder Buchstaben sind. Gibt man beispielsweise an:

```
kapitel[1-5]
```

trifft das Suchmuster auf alle Dateien von **kapitel1** bis **kapitel5** zu. Das ist eine einfache Methode, nur einige wenige Kapitel gleichzeitig anzugeben.

Probieren Sie das Kommando **pr** mit einem Argument in eckigen Klammern aus:

```
$ pr kapitel [2-4] <CR>
```

Durch dieses Kommando wird der Inhalt von **kapitel2**, **kapitel3** und **kapitel4** in dieser Reihenfolge auf dem Terminal ausgegeben.

Mit einer Zeichenklasse kann man auch einen Bereich von Buchstaben angeben. Gibt man [A-Z] an, sucht die Shell nur nach Großbuchstaben, bei [a-z] nur nach Kleinbuchstaben.

Die Verwendungsmöglichkeiten der Maskierungszeichen werden in Abbildung 7-3 zusammengefaßt. Der Leser sollte nun mit diesen Maskierungszeichen im aktuellen Verzeichnis experimentieren.

Zeichen	Funktion
*	steht für eine beliebige Zeichenkette, einschließlich einer leeren Zeichenkette (kein Zeichen)
?	steht für ein beliebiges Einzelzeichen
[]	steht für eines der in Klammern gesetzten Einzelzeichen
[-]	steht für den Bereich der angegebenen Zeichen

Abbildung 7-3: Übersicht über die Maskierungszeichen

Sonderzeichen

Die Shell-Sprache bietet noch einige weitere spezielle Zeichen für bestimmte Funktionen. Einige dieser Sonderzeichen werden in diesem Abschnitt beschrieben, weitere folgen im nächsten Abschnitt unter "Umlenkung von Eingabe und Ausgabe."

Kommando im Hintergrund ausführen: das kaufmännische und (&)

Einige Shell-Kommandos benötigen relativ viel Zeit zur Ausführung. Das kaufmännische UND (&) wird eingesetzt, um Kommandos im Hintergrundmodus ausführen zu lassen und damit das Terminal für andere Aufgaben freizugeben. Das allgemeine Format für dieses Kommando lautet:

Kommando &<CR>

NOTE

Interaktive Shell-Kommandos, wie zum Beispiel `read` (siehe "Das Kommando `read`" in diesem Kapitel), sollten nicht im Hintergrund ausgeführt werden.

Im folgenden Beispiel führt die Shell eine lange Suchaktion im Hintergrundmodus aus: mit dem Kommando `grep(1)` wird in der Datei `konten` nach der Zeichenkette "delinquent" gesucht. Dabei ist zu beachten, daß das `&` als letztes Zeichen in der Kommandozeile steht:

```
$ grep delinquent konten &<CR>
21940
$
```

Bei der Ausführung eines Kommandos im Hintergrund gibt das UNIX-System eine Prozeß-Nummer aus; die Prozeß-Nummer in diesem Beispiel ist 21940. Mit dieser Nummer kann man die Ausführung eines Hintergrundkommandos stoppen. Dies wird ausführlicher im Abschnitt "Aufrufen und Beenden von Prozessen" beschrieben. Das Bereit-Zeichen in der letzten Zeile bedeutet, daß man auf das Terminal zugreifen kann bzw. daß es eine Eingabe erwartet, während `grep` im Hintergrund bereits ausgeführt wird.

Die Ausführung eines Kommandos im Hintergrund beeinflusst lediglich die Verfügbarkeit des Terminals, nicht dagegen die Ausgabe des Kommandos. Die Ausgabe eines Kommandos erfolgt auf dem Terminal, unabhängig davon, ob er im Vorder- oder Hintergrund ausgeführt wird, außer wenn man die Ausgabe in eine Datei umlenkt (nähere Angaben dazu sind unter "Umlenkung der Ausgabe" weiter unten in diesem Kapitel zu finden).

Soll ein Kommando im Hintergrund weiter ausgeführt werden, nachdem man sich abgemeldet hat, kann man dies mit dem Kommando `nohup(1)` eingeben. Dies wird unter "Das Kommando `nohup`" weiter unten in diesem Kapitel ausführlich beschrieben.

Ausführen von Kommandos in Folge: das Semikolon (;)

Man kann zwei oder mehr Kommandos in eine Zeile eingeben; dabei muß zwischen je zwei Kommandos ein Semikolon gesetzt werden (`;`), wie im folgenden Format:

```
Kommando1; Kommando2; Kommando3<CR>
```

Das UNIX-System führt die Kommandos in der Reihenfolge ihres Auftretens in der Kommandozeile aus und gibt alle Ausgaben auf dem Bildschirm aus. Diesen Vorgang nennt man sequentielle Ausführung von Kommandos.

Zur Übung, und um die Funktion des ; zu veranschaulichen, geben Sie folgendes ein:

```
cd; pwd; ls <CR>
```

Die Shell führt diese Kommandos sequentiell, d. h. nacheinander aus:

1. Mit `cd` wird in das Anmelde- bzw. Benutzerverzeichnis gewechselt.
2. Mit `pwd` wird der vollständige Pfadname des aktuellen Verzeichnisses ausgegeben.
3. Mit `ls` werden die Dateien im aktuellen Verzeichnis ausgegeben.

Sollen die Reaktionen des Systems auf diese Kommandos nicht auf dem Bildschirm erscheinen, sind unter "Umlenkung der Ausgabe" geeignete Maßnahmen zu finden.

Sonderbedeutungen aufheben: der Backslash (\)

Die Shell interpretiert den Backslash (\) als Fluchtsymbol, mit dem man eine eventuelle Sonderbedeutung eines nachfolgenden Zeichens aufheben kann. Zur Veranschaulichung eine zweizeilige Datei unter dem Namen `versuch` mit folgendem Text erstellen:

```
Der * am Himmel  
leuchtet hell.
```

Mit dem Kommando `grep` nach dem Stern in der Datei suchen, wie im folgenden Beispiel gezeigt:

```
$ grep \* versuch <CR>  
Der * am Himmel  
$
```

Das Kommando `grep` sucht im Text nach dem Stern (*) und zeigt die Zeile an, die ihn enthält. Ohne das Zeichen \ wäre * für die Shell ein Maskierungszeichen, das hier für alle Dateinamen des aktuellen Verzeichnisses stünde.

Sonderbedeutungen aufheben: Anführungszeichen

Eine weitere Möglichkeit, die Sonderbedeutung von Sonderzeichen zu umgehen, sind Anführungszeichen. Dabei heben einfache Anführungszeichen ('...') die Sonderbedeutung jedes Zeichens auf, doppelte Anführungszeichen ("...") die Sonderbedeutung jedes Zeichens außer \$ und ` (Gravis), deren Sonderbedeutung auch innerhalb doppelter Anführungszeichen erhalten bleibt. Gegenüber dem umgekehrten Schrägstrich haben Anführungszeichen den Vorteil, daß viele Sonderzeichen in Anführungszeichen gesetzt werden können und sie präziser wirken.

Wenn beispielsweise die Datei mit dem Namen `versuch` noch die Zeile

```
Er fragte sich nur warum? Warum???
```

enthält, kann man mit dem Kommando `grep` die Zeile mit den drei Fragezeichen wie folgt suchen:

```
$ grep '???' versuch <CR>
Er fragte sich nur warum? Warum???
```

Hätte man dasselbe mit folgender Eingabe versucht:

```
grep ??? versuch <CR>
```

wären die drei Fragezeichen als Shell-Maskierungszeichen interpretiert worden, d. h. hier für alle Dateinamen, die drei Zeichen lang sind.

Anführungszeichen zum Aufheben der Sonderbedeutung des Leerzeichens

Ein häufig auftretender Fall, daß Anführungszeichen als Fluchtsymbole verwendet werden, ist das Aufheben der Sonderbedeutung des Leerzeichens. Die Shell interpretiert ein Leerzeichen in einer Kommandozeile als Begrenzungszeichen zwischen den Argumenten eines Kommandos. Diese Sonderbedeutung kann man mit einfachen oder doppelten Anführungszeichen aufheben.

Sucht man beispielsweise nach zwei oder mehr Wörtern, die zusammen in einem Text erscheinen, definiert man die Wörter als Einzelargument (für das Kommando `grep`), indem man sie in Anführungszeichen setzt. Mit folgender Kommandozeile sucht man die Wörter "am Himmel" in der Datei `versuch`:

```
$ grep 'am Himmel' versuch <CR>
Der * am Himmel
$
```

grep sucht die Zeichenkette "am Himmel" und gibt die Zeile aus, in der sie enthalten ist. Was würde wohl passieren, wenn man die Zeichenkette nicht in Anführungszeichen setzte?

Die Möglichkeit, die Sonderbedeutung eines Leerzeichens aufzuheben, ist vor allem beim Kommando **banner**(1) nützlich. Durch dieses Kommando wird eine Nachricht in großen Plakatbuchstaben auf einem Terminalbildschirm ausgegeben.

Das Kommando **banner** wird mit einer Nachricht aufgerufen, die aus einem oder mehreren Argumenten besteht (in diesem Fall in der Regel Wörter), die in der Kommandozeile durch Leerzeichen getrennt werden. Das Kommando **banner** interpretiert diese Leerzeichen als Begrenzungszeichen für die Argumente, und es gibt jedes Argument in einer eigenen Zeile aus.

Sollen mehrere Argumente in derselben Zeile ausgegeben werden, sind die Wörter zusammen in doppelte Anführungszeichen zu setzen. Beispielsweise kann man mit folgender Eingabe einen Geburtstagsgruß an einen anderen Benutzer schicken:

```
banner Alles Gute zum Geburtstag <CR>
```

Durch dieses Kommando wird die Nachricht als vierzeilige Plakatschrift ausgegeben.

Soll dieselbe Meldung als dreizeilige Plakatschrift ausgegeben werden, ist einzugeben:

```
banner Alles Gute "zum Geburtstag" <CR>
```

Nun erscheinen die Wörter "zum" und "Geburtstag" in derselben Zeile, d. h. das Leerzeichen zwischen ihnen hat nicht mehr die Sonderbedeutung eines Begrenzungszeichens.

In Abbildung 7-4 werden die Syntax und die Funktionen des Kommandos **banner** zusammengefaßt.

Kommandoübersicht		
banner – Ausgabe in Plakatschrift erzeugen		
<i>Kommando</i>	<i>Optionen</i>	<i>Argumente</i>
banner	keine	Zeichen
Beschreibung:	Mit banner können bis zu zehn Zeichen als große Buchstaben angezeigt werden.	
Bemerkungen:	Weiter unten in diesem Kapitel wird beschrieben, wie man die Ausgabe des Kommandos banner in eine Datei umlenkt, die als Plakat verwendet werden soll.	

Abbildung 7-4: Übersicht über das Kommando banner

Umlenkung von Eingabe und Ausgabe

Im UNIX-System erwarten einige Kommandos ihre Eingabe von der Tastatur (Standardeingabe), und die meisten Kommandos zeigen ihre Ausgabe auf dem Terminal an (Standardausgabe). Man kann jedoch als Standardeingabe und Standardausgabe auch Dateien und Programme definieren. Dieser Vorgang wird Umlenkung oder Umleitung genannt. Mit der Umlenkung kann man die Shell anweisen,

- die Eingabe aus einer Datei anstelle von der Tastatur zu lesen,
- die Ausgabe in eine Datei anstelle an das Terminal zu senden,
- ein Programm als Eingabequelle für ein anderes Programm verwenden.

Zur Umlenkung von Ein- und Ausgabe werden einige Operatoren verwendet: das Zeichen "kleiner als" (<), das Zeichen "größer als" (>), zwei Zeichen "größer als" (>>) und eine Pipe (|).

Umlenkung der Eingabe: das Zeichen <

Zur Umlenkung der Eingabe wird in der Kommandozeile ein Zeichen "kleiner als" (<), gefolgt von einem Dateinamen eingegeben:

Kommando < Datei<CR>

Soll beispielsweise eine Nachricht mit dem Kommando `mail(1)` (in Kapitel 8 beschrieben) an einen anderen Benutzer mit dem Benutzernamen `kollege` übermittelt werden und ist die Nachricht bereits in einer Datei mit dem Namen `bericht` vorhanden, kann man sich das erneute Eingeben der Nachricht ersparen, indem man den Dateinamen als die Quelle der Eingabe angibt:

`mail kollege < bericht<CR>`

Umlenkung der Ausgabe in eine Datei: das Zeichen >

Die Ausgabe kann man umlenken, indem man einen Dateinamen nach einem Zeichen "größer als" (>) in der Kommandozeile angibt:

Kommando > Datei<CR>



Wird die Ausgabe in eine bereits bestehende Datei umgelenkt, wird diese Datei durch die Ausgabe überschrieben, da die Shell es nicht zuläßt, zwei Dateien desselben Namens in demselben Verzeichnis zu speichern.

Daher sollte man vor der Umlenkung der Ausgabe eines Kommandos stets prüfen, ob nicht bereits eine Datei dieses Namens besteht, es sei denn, es spielt keine Rolle, wenn sie überschrieben wird. Die Shell gibt keine Warnung aus, wenn eine bestehende Datei überschrieben wird.

Man sollte daher zuerst das Kommando `ls` mit dem gewünschten Dateinamen als Argument ausführen, um sich zu vergewissern, ob eine Datei dieses Namens bereits vorhanden ist. Existiert bereits eine Datei unter diesem Namen, wird sie durch `ls` angezeigt; falls nicht, erscheint eine Meldung, daß die Datei im aktuellen Verzeichnis nicht gefunden wurde. Prüft man beispielsweise das Verzeichnis auf die Existenz der Dateien `temp` und `schrott`, erhält man folgende Ausgabe:

```
$ ls temp<CR>
temp
$ ls schrott<CR>
schrott: no such file or directory
$
```

Das bedeutet, daß man die neue Ausgabedatei zwar *schrott*, nicht aber *temp* nennen kann, es sei denn, der Inhalt der bestehenden Datei *temp* wird nicht mehr benötigt.

Anhängen der Ausgabe an eine bestehende Datei: das Symbol >>

Eine weitere Möglichkeit, zu vermeiden, daß eine bestehende Datei überschrieben wird, besteht darin, das doppelte Symbol zur Umlenkung (>>) wie folgt einzugeben:

Kommando >> *Datei*<CR>

Damit wird die Ausgabe eines Kommandos an das Ende der Datei *Datei* angehängt. Gibt es die *Datei* noch nicht, wird sie damit angelegt.

Im folgenden Beispiel wird gezeigt, wie man die Ausgabe aus dem Kommando *cat* an eine bestehende Datei anhängen kann. Zuerst wird das Kommando *cat* auf beide Dateien ohne Umlenkung angewandt, um den Inhalt der beiden Dateien anzuzeigen. Dann wird der Inhalt von *versuch2* nach der letzten Zeile von *versuch1* angehängt; dazu führt man das Kommando *cat* mit der Datei *versuch2* aus und lenkt die Ausgabe um in *versuch1*.

```
$ cat versuch1<CR>
Dies ist die erste Zeile von versuch1.
Hallo.
Dies ist die letzte Zeile von versuch1.
$
$ cat versuch2<CR>
Dies ist der Anfang von versuch2.
Hallo.
Das ist der Schluß von versuch2.
$
$ cat versuch2 >> versuch1<CR>
$ cat versuch1<CR>
Dies ist die erste Zeile von versuch1.
Hallo.
Dies ist die letzte Zeile von versuch1.
Dies ist der Anfang von versuch2.
Hallo.
Das ist der Schluß von versuch2.
$
```

Nützliche Anwendungsmöglichkeiten für die Ausgabeumlenkung

Die Umlenkung von Ausgaben ist insbesondere nützlich, wenn man verhindern will, daß sie sofort auf dem Bildschirm erscheinen oder wenn man sie abspeichern will. Auch bei Kommandos, die lästige Routinearbeiten mit Textdateien ausführen, ist die Umlenkung der Ausgabe von Nutzen. Zwei solche Kommandos sind **spell** und **sort**.

Das Kommando **spell**

Das Programm **spell** vergleicht jedes Wort einer Datei mit seiner eigenen Vokabularliste und gibt eine Liste aller möglicherweise falsch geschriebenen Wörter auf dem Bildschirm aus. Ist ein Wort nicht in der Vokabularliste von **spell** enthalten (wie z. B. Eigennamen), wird auch dieses Wort als falsch geschrieben ausgegeben.

Die Ausführung von **spell** mit einer langen Textdatei kann sehr lange dauern, und die Liste der möglicherweise falsch geschriebenen Wörter wird dann oft länger sein als der Bildschirm. Das Programm **spell** gibt die gesamte Ausgabe auf einmal aus; reicht der Bildschirm nicht aus, rollt die Ausgabe kontinuierlich über den Bildschirm, bis sie beendet ist. Eine solche Liste läuft so schnell über den Bildschirm, daß sie kaum lesbar ist.

Dieses Problem kann man vermeiden, indem man die Ausgabe des Kommandos **spell** in eine Datei umlenkt. Im folgenden Beispiel wird mit **spell** eine Datei des Namens **info** durchsucht, und die Liste der falsch geschriebenen Wörter wird in eine Datei mit dem Namen **fehler** geschrieben:

```
$ spell info > fehler<CR>
```

In Abbildung 7-5 werden die Syntax und die Funktionen des Kommandos **spell** zusammengefaßt.

Kommandoübersicht		
spell - Suchen von Rechtschreibfehlern		
<i>Kommando</i>	<i>Optionen</i>	<i>Argumente</i>
spell	ja*	<i>Datei</i>
Beschreibung:	Das Programm spell sammelt die Wörter aus Datei/en und vergleicht sie mit einer Rechtschreibliste. Die Wörter, die nicht in der Rechtschreibliste enthalten sind, werden auf dem Bildschirm angezeigt.	
Optionen:	spell verfügt über mehrere Optionen, darunter eine Option für britische Schreibweise.	
Bemerkungen:	Die Liste der falsch geschriebenen Wörter kann in eine Datei umgelenkt werden.	

* Eine Liste aller verfügbaren Optionen mit einer Erläuterung ihrer Funktionen ist unter **spell(1)** im *User's Reference Manual* zu finden.

Abbildung 7-5: Übersicht über das Kommando **spell**

Das Kommando **sort**

Mit dem Kommando **sort** werden die Zeilen einer Datei alphabetisch geordnet (nähere Angaben sind in Kapitel 3 enthalten). Da man Dateien nach einem Sortiervorgang in der Regel auch erhalten will, ist hier die Umlenkung der Ausgabe von besonderem Nutzen.

Es ist darauf zu achten, daß ein neuer Name für die Datei angegeben wird, in die die Ausgabe des Kommandos **sort** umgelenkt wird (d. h. die alphabetisch sortierte Liste). Nach dem Aufrufen von **sort** löscht die Shell zuerst den Inhalt der Datei, in die die umgelenkte Ausgabe geschrieben werden soll. Dann wird der Sortiervorgang durchgeführt und die Ausgabe in die leere Datei geschrieben. Gibt man beispielsweise folgendes ein:

```
sort liste > liste<CR>
```

löscht die Shell den Inhalt von **liste** und kann daher auch nichts sortieren, da Ausgangs- und Zielfile den gleichen Namen haben.

Kombination von Hintergrundmodus und Ausgabeumlenkung

Die Ausführung eines Kommandos im Hintergrund hat alleine keine Wirkung auf die Ausgabe des Kommandos, sofern sie nicht umgelenkt wird, wird die Ausgabe stets auf dem Terminal ausgegeben. Benutzt man das Terminal für andere Aufgaben, während ein Kommando im Hintergrund läuft, wird man daher unterbrochen, wenn das Kommando seine Ausgabe auf den Bildschirm schickt. Lenkt man dagegen die Ausgabe in eine Datei um, kann man ungestört am Terminal arbeiten.

Im Abschnitt "Sonderzeichen" wurde beispielsweise beschrieben, wie man das Kommando **grep** mit **&** im Hintergrund ausführen läßt. Soll beispielsweise eine Datei mit dem Namen **zeitplan** nach dem Wort "test" durchsucht werden, kann man das Kommando **grep** im Hintergrund laufen lassen und die Ausgabe in eine Datei mit dem Namen **testdatei** umlenken:

```
$ grep test zeitplan > testdatei &<CR>
```

Dann kann man das Terminal für andere Arbeiten benutzen und die Datei **testdatei** ansehen, wenn man die anderen Arbeiten abgeschlossen hat.

Umlenkung der Ausgabe in ein Kommando: die Pipe (|)

Das Zeichen | steht für eine Pipe. Eine Pipe ist eine vielseitig einsetzbare Funktion, bei der die Ausgabe aus einem Kommando sofort als Eingabe für ein anderes Kommando dient, ohne daß Zwischendateien angelegt werden müssen. Eine Kommandozeile mit mehreren Pipes wird Pipeline genannt.

Das allgemeine Format einer Pipeline lautet:

Kommando1 | *Kommando2* | *Kommando3*...<CR>

Die Ausgabe von *Kommando1* wird als Eingabe für *Kommando2* verwendet, und die Ausgabe von *Kommando2* wird dann als Eingabe für *Kommando3* verwendet.

Die Effizienz und Leistungsfähigkeit einer Pipeline wird am besten deutlich, wenn man sie mit einer anderen Methode vergleicht, die zu demselben Ergebnis führt.

- Bei der Methode mit Umlenkung der Eingabe und Ausgabe wird zuerst ein Kommando ausgeführt und seine Ausgabe in einer Zwischendatei gespeichert. Dann wird das zweite Kommando ausgeführt, bei dem der Inhalt der Zwischendatei als Eingabe angegeben wird. Nachdem das zweite Kommando ausgeführt ist, wird schließlich die Zwischendatei gelöscht.
- Bei der Methode mit einer Pipeline wird ein Kommando ausgeführt und das Ergebnis über eine Pipe direkt an ein zweites Kommando übergeben.

Im folgenden Beispiel soll ein Geburtstagsgruß in Plakatschrift an den Eigentümer des Benutzernamens **david** per elektronischer Post versandt werden. Ohne Pipeline sind dafür drei Einzelschritte erforderlich:

1. Das Kommando **banner** aufrufen und seine Ausgabe in eine Zwischendatei umlenken:

```
banner Alles Gute > zwischen.tmp
```

2. Das Kommando **mail** mit der Datei **zwischen.tmp** als Eingabe aufrufen:

```
mail david < zwischen.tmp
```

3. Die Zwischendatei löschen:

```
rm zwischen.tmp
```

Im Gegensatz dazu kann man dies mit einer Pipeline in einem einzigen Schritt ausführen lassen:

```
banner Alles Gute | mail david <CR>
```

Pipeline mit den Kommandos cut und date

Die Kommandos **cut** und **date** liefern ein gutes Beispiel dafür, wie man durch Pipelines die Vielseitigkeit von einzelnen Kommandos vergrößern kann. Mit dem Kommando **cut** kann man einen Ausschnitt jeder Zeile in einer Datei heraussuchen lassen. Dabei wird nach Zeichen gesucht, die sich an einer bestimmten Stelle der Zeilen befinden, und diese werden ausgegeben. Mit der Option **-c** gibt man an, welcher Bereich einer Zeile herausgesucht werden soll. Dabei ist die Anzahl der Stellen in der Zeile anzugeben, ausgehend vom linken Rand.

Im folgenden Beispiel soll jeweils nur das Datum aus den Zeilen einer Datei mit dem Namen **geburtstage** herausgesucht werden. Die Datei enthält folgende Liste:

Anne	26.12.
Klaus	7.4.
Maria	18.10.
Peter	11.9.
Nina	23.4.
Samuel	8.12.

Die Geburtstagsdaten stehen zwischen der neunten und der vierzehnten Spalte der einzelnen Zeilen. Mit folgender Kommandozeile werden sie herausgesucht:

```
cut -c9-14 geburtstage <CR>
```

Daraus ergibt sich folgende Ausgabe:

26.12.
7.4.
18.10.
11.9.
23.4.
8.12.

In Abbildung 7-6 werden die Syntax und die Funktionen des Kommandos `cut` zusammengefaßt.

Kommandoübersicht		
cut – Ausschnitte aus bestimmten Feldern der einzelnen Zeilen einer Datei bilden		
<i>Kommando</i>	<i>Optionen</i>	<i>Argumente</i>
cut	-cListe -fListe [-d]	<i>Datei</i>
Beschreibung:	Mit cut werden bestimmte Spalten einer Tabelle oder Felder aus Dateizeilen "ausgeschnitten", daher der Name "cut".	
Optionen:	<p>-c : Liste der Zeichenpositionen vom linken Rand aus. Mit -c1-9 kann man beispielsweise auch den Bereich der Zeichen 1-9 angeben.</p> <p>-f : Auflisten der Feldnummern von links, getrennt durch ein Begrenzungszeichen, das mit -d angegeben wird.</p> <p>-d gibt die Feldbegrenzung für die Option -f an. Standardwert ist ein Leerzeichen. Soll das Begrenzungszeichen ein Doppelpunkt sein, wird er folgendermaßen eingegeben: -d : .</p>	
Bemerkungen:	Wenn der Benutzer das Kommando cut nutzbringend einsetzen kann, sind häufig auch die Kommandos paste und split nützlich.	

Abbildung 7-6: Übersicht über das Kommando cut

Das Kommando `cut` wird normalerweise auf eine Datei angewendet. Mit einer Pipe kann man dieses Kommando jedoch auch auf die Ausgabe anderer Kommandos anwenden. Dies ist günstig, wenn man nur einen Teil der Informationen benötigt, die von einem anderen Kommando ausgegeben werden. Im folgenden Beispiel soll nur die Zeit ausgegeben werden. Mit dem Kommando `date` werden Wochentag, Datum und Uhrzeit wie folgt ausgegeben:

```
$ date<CR>
Sat May 20 13:12:32 MES 1989
```

Die Zeitangabe erscheint zwischen den Spalten 12 und 19 der Zeile. Nun kann man die Zeit (ohne Datum) ausgeben lassen, indem man eine Pipe verwendet, durch die die Ausgabe von `date` in das Kommando `cut` eingegeben wird, und die Stellen 12–19 mit der Option `-c` auswählt. Die entsprechende Kommandozeile und die Ausgabe sehen folgendermaßen aus:

```
$ date | cut -c12-19<CR>
13:14:56
```

In Abbildung 7-7 werden die Syntax und die Funktionen des Kommandos `date` zusammengefaßt.

Kommandoübersicht		
date - Datum und Uhrzeit ausgeben		
<i>Kommando</i>	<i>Optionen</i>	<i>Argumente</i>
date	+%m%d%y* +%H%%M%S	ja*
Beschreibung:	Mit date wird das aktuelle Datum und die Uhrzeit auf dem Terminal ausgegeben.	
Optionen:	Mit +% gefolgt von m (für Monat), d (für Tag), y (für Jahr), H (für Stunde), M (für Minute), und S (für Sekunde) werden diese Daten auf dem Terminal ausgegeben. Man kann auch Erläuterungen hinzufügen:	
	date '+%H:%M ist die aktuelle Uhrzeit'	
Bemerkungen:	Arbeitet man an einem kleinen Rechnersystem, an dem man gleichzeitig Benutzer und Systemverwalter ist, kann man meist das Datum und die Uhrzeit durch wahlweise verwendbare Argumente zum Kommando date einstellen. Nähere Informationen dazu sind im Reference Manual zu finden. In einer Mehrplatzumgebung stehen diese Argumente nur dem Systemverwalter zur Verfügung.	

Abbildung 7-7: Übersicht über das Kommando **date**

- * Eine Liste aller verfügbaren Optionen mit einer Erläuterung ihrer Funktionen ist unter **date(1)** im *User's Reference Manual* zu finden.

Ausgabe als Argument einsetzen

Die Ausgabe jedes Kommandos kann festgehalten und als Argument in einer Kommandozeile verwendet werden. Dazu wird das Kommando zwischen Akzentzeichen (`...`) an die Stelle in der Kommandozeile gesetzt, an der die Ausgabe als Argument dienen soll. Dieser Vorgang wird als "Kommandosubstitution" bezeichnet.

Beispielsweise kann man die Ausgabe der oben verwendeten Pipe aus den Kommandos `date` und `cut` als Argument in einem Ausdruck mit `banner` verwenden. Geben Sie dazu folgende Kommandozeile ein:

```
$ banner `date | cut -c12-19` <CR>
```

Ergebnis: das System gibt das aktuelle Datum in Plakatschrift aus.

Im Abschnitt "Shell-Programmierung" in diesem Kapitel wird gezeigt, wie man außerdem die Ausgabe einer Kommandozeile als Variablenwert einsetzen kann.

Aufrufen und Beenden von Prozessen

In diesem Abschnitt werden folgende Themen behandelt:

- Programme zur späteren Ausführung bestimmen; dazu dienen die Kommandos **batch** oder **at**
- Abfragen des Status aktiver Prozesse
- Beenden von aktiven Prozessen
- Prozesse nach dem Abmelden im Hintergrund weiter ausführen lassen.

Programme mit den Kommandos **batch** und **at** zur späteren Ausführung vormerken

Mit den Kommandos **batch** und **at** kann ein Kommando oder eine Reihe von Kommandos zur späteren Ausführung vorgesehen werden. Beim Kommando **batch** bestimmt dabei das System, wann das Kommando aufgerufen wird, beim Kommando **at** legt dies der Benutzer fest. Die Eingabe für beide Kommandos kommt von der Standardeingabe (dem Terminal); die Liste der am Terminal eingegebenen Kommandos wird durch Drücken von **<^d>** (CONTROL-d) beendet.

Das Kommando **batch** ist vor allem nützlich, wenn man einen Prozeß oder ein Shell-Programm ausführen lassen will, das viel Rechnerzeit benötigt. Mit dem Kommando **batch** wird ein Stapeljob (mit den auszuführenden Kommandos) an das System übergeben. Der Auftrag wird in eine Warteschlange gestellt und ausgeführt, sobald die Systemauslastung auf ein bestimmtes Niveau fällt. Damit kann das System schnell auf andere Eingaben reagieren, und andere Benutzer werden nicht unnötig behindert.

Das allgemeine Format für das Kommando **batch** lautet:

```
batch <CR>
  erstes Kommando <CR>
  .
  .
  .
  letztes Kommando <CR>
  <^d>
```

Soll nur ein Kommando mit **batch** aufgerufen werden, kann es wie folgt eingegeben werden:

```
batch Kommandozeile <CR>  
<^d>
```

Im folgenden Beispiel wird **batch** verwendet, um das Kommando **grep** zum nächsten günstigen Zeitpunkt ausführen zu lassen. In diesem Beispiel durchsucht **grep** alle Dateien des aktuellen Verzeichnisses, und die Ausgabe wird in die Datei **dol.datei** umgelenkt.

```
$ batch grep dollar * > dol.datei <CR>  
<^d>  
job 155223141.b at Sun May 14 11:14:54 1989  
$
```

Nachdem ein Auftrag mit **batch** an das System übergeben wurde, gibt es eine Auftragsnummer, das Datum und die Uhrzeit aus. Die Auftragsnummer ist nicht identisch mit der Prozeßnummer, die vom System generiert wird, wenn man ein Kommando im Hintergrund ausführt.

In Abbildung 7-8 werden die Syntax und die Funktionen des Kommandos **batch** zusammengefaßt.

Kommandoübersicht		
batch – Kommandos zu einem späteren Zeitpunkt ausführen		
<i>Kommando</i>	<i>Optionen</i>	<i>Eingabe</i>
batch	keine	<i>Kommandozeilen</i>
Beschreibung:	Mit batch wird ein Stapeljob an das System übergeben, der in eine Warteschlange gestellt und aufgerufen wird, wenn die Belastung des Systems auf ein bestimmtes Niveau fällt.	
Bemerkungen:	Die Liste der Kommandos ist mit <code><^d></code> (CONTROL-d) abzuschließen.	

Abbildung 7-8: Übersicht über das Kommando **batch**

Mit dem Kommando **at** kann man einen genauen Zeitpunkt für die Ausführung der angegebenen Kommandos festlegen. Das allgemeine Format für das Kommando **at** lautet:

```
at Zeit <CR>
  erstes Kommando <CR>
  .
  .
  .
  letztes Kommando <CR>
  <^d>
```

Für das Argument *Zeit* ist die Tageszeit und, falls nicht das aktuelle Datum gelten soll, das Datum einzugeben.

Im folgenden Beispiel wird gezeigt, wie man mit dem Kommando **at** einen Geburtstagsgruß in Plakatschrift am Geburtstag des Empfängers mit dem Benutzernamen **emil** auf seinem Terminal ausgeben läßt:

```
$ at 8:15am Feb 27 <CR>
banner Alles Gute |mail emll <CR>
<^d>
job 453400603.a at Thurs May 18 08:15:00 1989
$
```

Wie beim Kommando **batch** werden daraufhin auch bei **at** die Auftragsnummer, das Datum und die Uhrzeit ausgegeben.

Sollen die Kommandos eines Stapeljobs mit **batch** oder **at**, die noch in der Warteschlange stehen, doch nicht ausgeführt werden, kann man sie mit der Option **-r** des Kommandos **at** unter Angabe der Auftragsnummer löschen. Das allgemeine Format dafür lautet:

```
at -r Auftragsnummer <CR>
```

Mit folgender Eingabe wird der oben eingegebene **at**-Auftrag für den Geburtstagsgruß storniert:

```
at -r 453400603.a <CR>
```

Hat man die Auftragsnummer vergessen, kann man sich mit dem Kommando **at -l** eine Liste der gerade in der Warteschlange für **batch** oder **at** vorgemerkten Aufträge anzeigen lassen, wie im folgenden Bildschirm gezeigt:

```
$ at -l<CR>
user = Benutzername 168302040.a at Sat May 20 13:00:00 1989
user = Benutzername 453400603.a at Fri May 19 08:15:00 1989
$
```

Das System zeigt dabei die Auftragsnummer und die Uhrzeit der Ausführung an.

Schicken Sie sich zur Übung nun mit dem Kommando **at** selbst die Datei **info** per elektronischer Post zu - als Erinnerung, daß Mittagszeit ist (dazu muß die Datei in das Kommando **mail** umgelenkt werden, sofern man nicht das "here document" verwendet, das im Abschnitt "Shell-Programmierung" beschrieben wird). Geben Sie dann das Kommando **at** mit der Option **-l** ein:

```
$ at 12:00pm<CR>
mail Benutzername < info<CR>
<^d>
job 263131754.a at May 08 12:00:00 1989
$
$ at -l<CR>
user = Benutzername 263131754.a at May 08 12:00:00 1989
$
```

In Abbildung 7-9 werden die Syntax und die Funktionen des Kommandos **at** zusammengefaßt.

Kommandoübersicht		
at – Ausführen von Kommandos zu einem bestimmten Zeitpunkt		
<i>Kommando</i>	<i>Optionen</i>	<i>Argumente</i>
at	-r -l	<i>Zeit (Datum)</i> <i>Auftragsnummer</i>
Beschreibung:	Mit dem Kommando werden Kommandos zu einem bestimmten Zeitpunkt ausgeführt. Als Zeitangabe können ein bis vier Ziffern und am (vormittags) und pm (nachmittags) angegeben werden. Zur Angabe des Datums ist ein Monatsname (englisch), gefolgt vom Datum des Tages einzugeben. Ein Datum braucht nicht angegeben werden, wenn der Auftrag am selben Tag ausgeführt werden soll. Weitere Standardwerte für diese Zeitangaben sind unter at(1) im <i>User's Reference Manual</i> zu finden.	
Optionen:	Mit der Option -r kann man unter Angabe der Auftragsnummer bereits übergebene Aufträge wieder löschen. Durch die Option -l (ohne Argumente) wird die Auftragsnummer sowie der Status aller vorge-merkten Aufträge mit at und batch angezeigt.	
Bemerkungen:	Beispiele für die Eingabe der Zeitangaben beim Kommando at: at 08:15am Feb 27 at 5:14pm Sept 24	

Abbildung 7-9: Übersicht über das Kommando at

Abrufen des Status laufender Prozesse

Mit dem Kommando `ps` kann man den Status aller von einem Benutzer gerade ausgeführten Prozesse abrufen. Beispielsweise kann man sich mit dem Kommando `ps` den Status aller Prozesse ausgeben lassen, die mit `&` im Hintergrund ausgeführt werden (dies wurde bereits im Abschnitt "Sonderzeichen" beschrieben).

Im folgenden Abschnitt, "Abbrechen aktiver Prozesse", wird beschrieben, wie man mit der PID (Prozeßnummer) die Ausführung eines Kommandos beendet. Eine PID ist eine Zahl von 1 bis 30.000, die vom UNIX-System jedem aktiven Prozeß zugeordnet wird.

Im folgenden Beispiel wird `grep` im Hintergrund ausgeführt und dann das Kommando `ps` eingegeben. Das System gibt daraufhin die Prozeßnummer (PID) und die Terminalnummer (TTY) aus. Außerdem gibt es die kumulative Ausführungszeit für jeden Prozeß (TIME) und den Namen des ausgeführten Kommandos (COMMAND) an.

```
$ grep Wort * > temp &<CR>
28223
$
$ ps<CR>
PID      TTY      TIME COMMAND
28124    tty10    0:00  sh
28223    tty10    0:04  grep
28224    tty10    0:04  ps
$
```

Das System gibt dabei eine PID für das Kommando `grep` und alle weiteren gerade laufenden Prozesse aus: das Kommando `ps` selbst und das Kommando `sh` (Shell), das läuft, solange man angemeldet ist. Das Shell-Programm `sh` interpretiert die Shell-Kommandos; es wird in den Kapiteln 1 und 4 ausführlicher beschrieben.

In Abbildung 7-10 werden die Syntax und die Funktionen des Kommandos `ps` zusammengefaßt.

Kommandoübersicht		
ps – Prozeßstatus anzeigen		
<i>Kommando</i>	<i>Optionen</i>	<i>Argumente</i>
<code>ps</code>	mehrere*	keine
Beschreibung:	Mit <code>ps</code> werden Informationen über aktive Prozesse ausgegeben.	
Optionen:	Mehrere. Wird keine angegeben, gibt <code>ps</code> den Status aller vom Benutzer aufgerufenen Prozesse aus.	
Bemerkungen:	Zeigt die PID (Prozeßnummer) an. Diese wird benötigt, um einen Prozeß abubrechen (kill).	

* Die verfügbaren Optionen sowie eine Erläuterung ihrer Funktionen sind unter `ps(1)` im *User's Reference Manual* zu finden.

Abbildung 7-10: Übersicht über das Kommando `ps`

Abbrechen aktiver Prozesse

Mit dem Kommando `kill` werden aktive Shell-Prozesse abgebrochen. Das allgemeine Format des Kommandos `kill` lautet:

`kill PID<CR>`

Mit dem Kommando `kill` werden Prozesse abgebrochen, die im Hintergrund laufen. Hintergrundprozesse können nicht mit den Tasten `BREAK` oder `DELETE` abgebrochen werden.

Im folgenden Beispiel wird gezeigt, wie das Kommando **grep** abgebrochen wird, das im vorigen Beispiel im Hintergrund aufgerufen wurde.

```
$ kill 28223 <CR>
28223 Terminated
$
```

Das System gibt eine Meldung und das Bereit-Zeichen \$ aus und zeigt damit an, daß der Prozeß abgebrochen wurde. Findet das System die angegebene PID nicht, gibt es eine Fehlermeldung aus:

```
kill:28223:No such process
```

In Abbildung 7-11 werden die Syntax und die Funktionen des Kommandos **kill** zusammengefaßt.

Kommandoübersicht		
kill – Einen Prozeß beenden		
Kommando	Optionen	Argumente
kill	ja*	Auftragsnummer oder PID
Beschreibung:	Durch kill wird der mit der Prozeßnummer angegebene Prozeß abgebrochen.	

* Die verfügbaren Optionen sowie eine Erläuterung ihrer Funktionen sind unter **kill(1)** im *User's Reference Manual* zu finden.

Abbildung 7-11: Übersicht über das Kommando **kill**

Das Kommando **nohup**

Beim Abmelden werden alle laufenden Prozesse abgebrochen. Soll ein Hintergrundprozeß nach dem Abmelden weiterlaufen, muß das Kommando **nohup** mit dem betreffenden Hintergrundkommando eingegeben werden.

Das Kommando **nohup** hat folgendes Format:

nohup *Kommando* & <CR>

Das Kommando **nohup** ist dabei vor dem Kommando anzugeben, das als Hintergrundprozeß weitergeführt werden soll.

Im folgenden Beispiel sollen alle Dateien im aktuellen Verzeichnis mit dem Kommando **grep** nach der Zeichenkette "Wort" durchsucht und die Ausgabe in die Datei **wort.liste** umgelenkt werden, und der Benutzer will sich sofort wieder abmelden. Dazu ist folgende Kommandozeile einzugeben:

nohup grep Wort * > wort.liste & <CR>

Das Kommando **nohup** kann mit dem Kommando **kill** abgebrochen werden. In Abbildung 7-12 werden die Syntax und die Funktionen des Kommandos **nohup** zusammengefaßt.

Kommandoübersicht		
nohup – Kommandos werden nach der Trennung vom System weiter ausgeführt		
<i>Kommando</i>	<i>Optionen</i>	<i>Argumente</i>
nohup	<i>keine</i>	<i>Kommandozeile</i>
Beschreibung:	Ein Kommando wird ausgeführt, selbst wenn man vom System getrennt wird oder sich abmeldet.	

Abbildung 7-12: Übersicht über das Kommando **nohup**

Nachdem diese grundlegenden Shell-Kommandos und ihre Syntax nun eingeführt wurden, sollte man sie in eigenen Shell-Programmen verwenden. Die folgenden Übungen erleichtern die Einarbeitung in die praktische Anwendung der Shell-Kommandosprache. Die Lösungen zu den Übungen befinden sich am Ende des Kapitels.

Übungen zur Kommandosprache

- 1-1. Was geschieht, wenn man einen Stern (*) am Anfang eines Dateinamens verwendet? Versuchen Sie, einige der Dateien eines Verzeichnisses unter Angabe eines * mit dem letzten Buchstaben eines der Dateinamen auflisten zu lassen. Was geschieht dabei?
- 1-2. Geben Sie folgende beiden Kommandos wie folgt ein:

```
cat[0-9]*<CR>
echo *<CR>
```

- 1-3. Ist ein Fragezeichen (?) am Anfang oder in der Mitte eines Dateinamens zulässig? Probieren Sie sie aus.
- 1-4. Sind Dateien vorhanden, deren Name mit einer Zahl beginnt? Können solche Dateien aufgelistet werden, ohne die übrigen Dateien des Verzeichnisses mit aufzulisten? Ist es möglich, nur die Dateien aufzulisten, die mit einem Kleinbuchstaben zwischen a und m beginnen? (Tip: setzen Sie einen Bereich von Zahlen oder Buchstaben in []).
- 1-5. Ist es zulässig, ein Kommando im Hintergrundmodus mit einer Zeile einzugeben, durch die mehrere weitere Kommandos nacheinander abgearbeitet werden? Ausprobieren und feststellen, was geschieht (Tip: ; und &. verwenden). Kann das Kommando, das im Hintergrund ausgeführt werden soll, an jeder Stelle der Kommandozeile stehen? Probieren Sie verschiedene Stellen aus. Experimentieren Sie mit jedem neu gelernten Zeichen, um die Möglichkeiten kennenzulernen, die das Zeichen bietet.
- 1-6. Lenken Sie die Ausgabe der Kommandos `pwd` und `ls` mit folgender Kommandozeile in eine Datei um:

```
cd; pwd; ls; ed versuch<CR>
```

Dabei ist zu beachten, daß, falls Sie die Ausgabe umlenken wollen, bei der zweiten Umlenkung das Symbol `>>` (anhängen) verwendet werden muß, wenn die Ausgabe beider Kommandos in dieselbe Datei geschrieben werden soll, da die Informationen aus dem Kommando `pwd` andernfalls gelöscht werden.

- 1-7. Versuchen Sie anstatt aus der Antwort den Teil Datum (`date`) herauszuziehen, nur das Datum ohne die Zeit in das Kommando `banner` umzulenken. Dazu muß nur ein Teil der Kommandozeile mit der Zeitangabe geändert werden; welcher?

```
banner `date |cut -c12-19` <CR>
```

Shell-Programmierung

Mit der Shell kann der Benutzer selbst Programme –d.h. neue Kommandos erstellen. Solche Programme werden auch "Shell-Prozeduren" genannt. In diesem Abschnitt wird beschrieben, wie Shell-Programme geschrieben und ausgeführt werden, und wie man die Kommandos, Variablen, Positionsparameter, Rückgabecodes und die grundlegenden Programmsteuerstrukturen dafür einsetzt.

Die Beispiele der Shell-Programme in diesem Abschnitt erscheinen in zwei Formen. Zunächst wird mit dem Kommando `cat` in einem Bildschirm der Inhalt einer Datei gezeigt, die ein Shell-Programm enthält.

```
$ cat testdatei <CR>
erstes Kommando
.
.
.
letztes Kommando
$
```

Dann, nach einer Kommandozeile, wird das Ergebnis des ausgeführten Programms angezeigt:

```
$ testdatei <CR>
Programmausgabe
$
```

Man sollte einen Editor bedienen können, bevor man versucht, Shell-Programme zu erstellen. Editoren sind in den Anleitungen von Kapitel 5 (zum Editor `ed`) und Kapitel 6 (zum Editor `vi`) beschrieben.

Shell-Programme

Erstellen einfacher Shell-Programme

Zur Einführung soll ein einfaches Shell-Programm erstellt werden, das die folgenden Aufgaben in der angegebenen Reihenfolge ausführt:

- Aktuelles Verzeichnis anzeigen
- Inhalt dieses Verzeichnisses auflisten
- Folgende Nachricht auf dem Terminal ausgeben: "Das ist das Ende des Shell-Programms".

Legen Sie eine Datei mit dem Namen `vl` (kurz für Verzeichnisliste) mit einem Editor an und geben Sie folgendes ein:

```
pwd<CR>
ls<CR>
echo Das ist das Ende des Shell-Programms.<CR>
```

Speichern Sie die Datei ab und verlassen Sie den Editor. Damit wurde bereits ein Shell-Programm erstellt. Sie können sich nun den Inhalt der Datei mit dem Kommando `cat` anzeigen lassen, wie auf dem folgenden Bildschirm dargestellt:

```
$ cat vl<CR>
pwd
ls
echo Das ist das Ende des Shell-Programms.
$
```

Aufrufen von Shell-Programmen

Eine Möglichkeit, ein Shell-Programm aufzurufen, ist das Kommando **sh**. Folgendes eingeben:

```
sh vl<CR>
```

Das Kommando **vl** wird mit **sh** aufgerufen, dann wird der Pfadname des aktuellen Verzeichnisses angezeigt, anschließend die Liste der Dateien des aktuellen Verzeichnisses und schließlich der Hinweis Das ist das Ende des Shell-Programms. Mit dem Kommando **sh** kann man Shell-Programme testen, um sicherzustellen, daß sie arbeiten.

Ist das Shell-Programm **vl** ein nützliches Kommando für den Benutzer, kann es mit dem Kommando **chmod** in eine ausführbare Datei umgewandelt werden. Dann genügt es, **vl** alleine einzugeben, damit das darin enthaltene Programm ausgeführt wird. Im folgenden Beispiel wird gezeigt, wie man eine Datei mit dem Kommando **chmod** ausführbar macht; danach wird **ls -l** aufgerufen, um die Änderungen in den Zugriffsrechten zu überprüfen.

```
$ chmod u+x vl<CR>
$ ls -l<CR>
total 2
-rw----- 1      Benutzername  Gruppe  3661  Nov  2  10:28 mbox
-rwx----- 1      Benutzername  Gruppe   48   Nov 15  10:50 vl
$
```

Mit diesem Kommando **chmod** wird das Recht zur Ausführung (**+x**) für den Eigentümer (**u**) aktiviert. Nun ist **vl** ein ausführbares Programm. Mit folgender Eingabe kann man es aufrufen:

```
vl<CR>
```

Das Ergebnis ist dasselbe wie oben, als es mit **sh vl** aufgerufen wurde. Weitere Angaben zum Kommando **chmod** sind in Kapitel 3 zu finden.

Verzeichnis `bin` für ausführbare Dateien anlegen

Sollen Shell-Programme von allen Verzeichnissen aus aufrufbar sein, kann man ein Verzeichnis `bin` im Benutzerverzeichnis anlegen und die Shell-Dateien in dieses Verzeichnis `bin` verschieben.

Dann muß das Verzeichnis `bin` noch in die Shell-Variable `PATH` aufgenommen werden:

```
PATH=$PATH:$HOME/bin
```

Weitere Informationen zur Shell-Variablen `PATH` sind unter "Variablen" und "Verwendung von Shell-Variablen" in diesem Kapitel enthalten.

Im folgenden Beispiel wird nochmals gezeigt, welche Kommandos erforderlich sind. Dabei steht `vl` im Benutzerverzeichnis. Geben Sie folgende Kommandozeilen ein:

```
cd <CR>  
mkdir bin <CR>  
mv vl bin/vl <CR>
```

Wechseln Sie in das Verzeichnis `bin` und geben Sie das Kommando `ls -l` ein. Besteht für `vl` noch immer Ausführbarkeitsrecht?

Wechseln Sie nun in ein anderes Verzeichnis (nicht das Benutzerverzeichnis) und geben Sie folgendes Kommando ein:

```
vl <CR>
```

In Abbildung 7-13 wird das neue Shell-Programm `vl` kurz zusammengefaßt.

Shell-Programm	
vl – Verzeichnispfad und Verzeichnisinhalt anzeigen (vom Benutzer definiert)	
<i>Kommando</i>	<i>Argumente</i>
vl	keine
Beschreibung:	Mit vl wird die Ausgabe der Shell-Kommandos pwd und ls angezeigt.

Abbildung 7-13: Übersicht über das Shell-Programm vl

Das Verzeichnis `bin` kann umbenannt werden; dann muß jedoch auch die Shell-Variablen `PATH` angepaßt werden.

Hinweise zur Benennung von Shell-Programmen

Shell-Programme können mit einem beliebigen zulässigen Dateinamen benannt werden. Man sollte jedoch einem Programm nicht den Namen eines Systemkommandos geben, da das System in diesem Fall das vom Benutzer definierte Shell-Kommando anstelle des Systemkommandos ausführt. Hätte man im obigen Beispiel dem Programm vl den Namen mv gegeben, würde das System jedesmal, wenn man versucht, eine Datei umzubenennen (Systemkommando mv), stattdessen den Inhalt des Verzeichnisses auflisten.

Ein weiteres Problem kann auftreten, wenn man beispielsweise das Programm vl mit ls benennt. Dann wird bei der Ausführung eine Endlosschleife erzeugt, da das vom Benutzer definierte Programm das Kommando ls aufruft. In solch einem Fall gibt das System nach einiger Zeit folgende Fehlermeldung aus:

```
Too many processes, cannot fork
```

Den ablaufenden Vorgang kann man sich wie folgt vorstellen: Der Benutzer gibt das neue Kommando, hier ls ein. Die Shell liest das Kommando pwd und führt es aus. Dann liest sie das Kommando ls im Programm des Benutzers und versucht, dieses Kommando ls erneut auszuführen. Dadurch wird eine Endlosschleife erzeugt.

Um solchen Fällen vorzubeugen, wurde im UNIX-System eine Begrenzung für die Anzahl der Durchläufe von Endlosschleifen vorgesehen. Eine Möglichkeit, dies zu verhindern, besteht darin, beim Schreiben eines Shell-Programms den vollständigen Pfadnamen des Systemkommandos `ls`, d. h. hier `/bin/ls`, anzugeben.

Das folgende Shell-Programm `ls` wird daher arbeiten:

```
$ cat ls <CR>
pwd
/bin/ls
echo Das ist das Ende des Shell-Programms.
```

Benennt man also das eigene Shell-Programm mit `ls`, kann man das Systemkommando `ls` nur ausführen lassen, wenn man den vollständigen Pfadnamen, `/bin/ls`, angibt.

Variablen

Variablen sind die Grundeinheiten von Daten, mit denen Shell-Programme arbeiten, im Unterschied zu Dateien. Im folgenden werden drei Typen von Variablen und ihre Verwendung beschrieben:

- Positionsparameter
- Sonderparameter
- Benannte Variablen

Positionsparameter

Ein Positionsparameter ist eine Variable innerhalb eines Shell-Programms, dessen Wert beim Aufrufen über ein in der Kommandozeile angegebenes Argument bestimmt wird. Positionsparameter sind durchnummeriert und werden jeweils mit vorangestelltem **\$**: **\$1**, **\$2**, **\$3**, usw. angegeben.

Ein Shell-Programm kann bis zu neun Positionsparameter adressieren. Wird ein Shell-Programm mit einer Kommandozeile der folgenden Art aufgerufen:

```
shell.prog pp1 pp2 pp3 pp4 pp5 pp6 pp7 pp8 pp9<CR>
```

wird dem Positionsparameter **\$1** im Programm der Wert **pp1**, dem Positionsparameter **\$2** im Programm der Wert **pp2**, usw. zugewiesen werden, wenn das Shell-Programm aufgerufen wird.

Um den Umgang mit Positionsparametern zu üben, legen Sie eine Datei mit dem Namen **pp** (kurz für Positionsparameter) an. Geben Sie dann die auf dem folgenden Bildschirm dargestellten Kommandos mit **echo** ein. Geben Sie dabei die Kommandozeilen so ein, daß folgende Ausgabe erscheint, wenn das Kommando **cat** mit der fertiggestellten Datei ausgeführt wird:

```
$ cat pp<CR>
echo Erster Positionsparameter: $1<CR>
echo Zweiter Positionsparameter: $2<CR>
echo Dritter Positionsparameter: $3<CR>
echo Vierter Positionsparameter: $4<CR>
$
```

Führt man dann das Shell-Programm mit den Argumenten **eins**, **zwei**, **drei** und **vier** aus, erhält man das folgende Ergebnis; das Shell-Programm **pp** ist zuvor noch mit dem Kommando **chmod** in ein ausführbares Programm umzuwandeln:

```
$ chmod u+x pp <CR>
$
$ pp eins zwei drei vier <CR>
Erster Positionsparameter: eins
Zweiter Positionsparameter: zwei
Dritter Positionsparameter: drei
Vierter Positionsparameter: vier
$
```

Auf dem folgenden Bildschirm wird das Shell-Programm **ghtag** gezeigt, mit dem ein Geburtstagsgruß an den in der Kommandozeile angegebenen Benutzernamen geschickt wird.

```
$ cat ghtag <CR>
banner Alles Gute | mail $1
```

Versuchen Sie, einen Geburtstagsgruß an sich selbst zu senden. Ist Ihr eigener Benutzername **susi**, lautet die Kommandozeile:

```
ghtag susi <CR>
```

In Abbildung 7-14 werden die Syntax und die Funktionen des Shell-Programms **ghtag** zusammengefaßt.

Shell-Programm	
gbtag – Geburtstagsgruß per E-Post schicken (vom Benutzer definiert)	
<i>Kommando</i>	<i>Argumente</i>
gbtag	<i>Benutzername</i>
Beschreibung:	Mit gbtag wird der Geburtstagsgruß "Alles Gute" per elektronischer Post in Plakatbuchstaben an den angegebenen Benutzer (Benutzername) geschickt.

Abbildung 7-14: Übersicht über das Kommando **gbtag**

Mit dem Kommando **who** kann man sich anzeigen lassen, welche Benutzer gerade am System angemeldet sind. Im folgenden Beispiel wird ein einfaches Shell-Programm mit dem Namen **werda** erstellt, mit dem man anzeigen lassen kann, ob ein bestimmter Benutzer (Benutzername) gerade am System arbeitet.

Geben Sie folgende Kommandozeile in eine Datei mit dem Namen **werda** ein:

```
who | grep $1 <CR>
```

Mit dem Kommando **who** werden alle gerade angemeldeten Benutzer des Systems aufgelistet, und mit dem Kommando **grep** wird die Ausgabe des Kommandos **who** nach einer Zeile abgesucht, die die Zeichenkette enthält, die als Wert für den Positionsparameter **\$1** eingegeben wird.

Übergeben Sie nun den eigenen Benutzernamen als Argument an das neue Programm **werda**. Ist der eigene Benutzername beispielsweise **susi**, setzt das Shell-Programm **werda** den Benutzernamen **susi** für den Parameter **\$1** im Programm ein und führt es wie folgende Kommandozeile aus:

```
who | grep susi <CR>
```

Die Ausgabe wird auf dem folgenden Bildschirm dargestellt:

```
$ werda susi<CR>
susi  tty26   Jan 24 13:35
$
```

Arbeitet der angegebene Benutzer gerade nicht am System, findet **grep** keine Zeichenkette, und **werda** erzeugt keine Ausgabe.

In Abbildung 7-15 werden die Syntax und die Funktionen des Kommandos **werda** zusammengefaßt.

Shell-Programm	
werda – Anmeldeinformationen ausgeben, wenn der angegebene Benutzer angemeldet ist (vom Benutzer definiertes Programm)	
<i>Kommando</i>	<i>Argumente</i>
werda	<i>Benutzername</i>
Beschreibung:	Ist der angegebene Benutzer angemeldet, zeigt werda den Benutzernamen, die TTY-Nummer, sowie Datum und Uhrzeit der Anmeldung des Benutzers an.

Abbildung 7-15: Übersicht über das Kommando **werda**

Eine Kommandozeile der Shell kann bis zu 128 Argumente enthalten, die Anzahl der gleichzeitig verwendbaren Positionsparameter ist jedoch auf neun (\$1 bis \$9) beschränkt. Diese Einschränkung kann man mit dem Kommando **shift** umgehen.

Eine weitere Möglichkeit, die Werte der Argumente der Kommandozeile zu adressieren, bietet sich mit dem Sonderparameter `$*`; er wird im folgenden Abschnitt beschrieben.

Sonderparameter

`$#` Dieser Parameter wird in ein Shell-Programm integriert und enthält die Anzahl der Argumente, mit denen das Shell-Programm aufgerufen wurde. Sein Wert kann an beliebiger Stelle im Shell-Programm verwendet werden.

Geben Sie die auf dem folgenden Bildschirm dargestellte Kommandozeile in ein ausführbares Shell-Programm mit dem Namen `arg.anz` ein und führen anschließend das Kommando `cat` mit der Datei aus.

```
$ cat arg.anz <CR>
echo Anzahl der Argumente: $#
$
```

Mit dem Programm wird einfach die Anzahl der Argumente angezeigt, mit der es aufgerufen wurde, wie im folgenden Beispiel dargestellt:

```
$ arg.anz teste dieses kleine programm <CR>
Anzahl der Argumente: 4
$
```

In Abbildung 7-16 wird das Shell-Programm `arg.anz` zusammenfassend dargestellt.

Shell-Programm	
arg.anz – Anzahl der angegebenen Argumente zählen und ausgeben (vom Benutzer definiertes Programm)	
<i>Kommando</i>	<i>Argumente</i>
arg.anz	<i>(Zeichenkette)</i>
Beschreibung:	<code>arg.anz</code> zählt die Argumente, die an das Kommando übergeben wurden und gibt die Summe aus.
Bemerkungen:	Mit diesem Kommando wird die Verwendung des Sonderparameters <code>\$#</code> veranschaulicht.

Abbildung 7-16: Übersicht über das Shell-Programm `arg.anz`

\$* Dieser Sonderparameter wird in ein Shell-Programm integriert und steht für eine Zeichenkette mit allen Argumenten, mit denen das Shell-Programm aufgerufen wurde, in der Reihenfolge ihrer Eingabe. Im Unterschied zu den Positionsparametern **\$1** bis **\$9** besteht hier keine Einschränkung auf neun Argumente.

Die Funktion des Sonderparameters **\$*** wird am folgenden Beispiel deutlich: Ein Shell-Programm mit dem Namen **anzg.param** erstellen, mit dem alle Parameter angezeigt werden. Dazu die Kommandozeile der folgenden Datei verwenden:

```
$ cat anzg.param <CR>
echo Die Parameter für dieses Kommando sind: $*
$
```

Durch **anzg.param** werden alle an das Kommando übergebenen Argumente angezeigt.

Wandeln Sie **anzg.param** in ein ausführbares Programm um und rufen Sie es mit folgenden Parametern auf:

Hallo. Wie geht's?


```
$ anzg.param Hallo. Wie geht's? <CR>
Die Parameter für dieses Kommando sind: Hallo. Wie geht's?
$
```

Im folgenden Beispiel wird **anzg.param** mit mehr als neun Argumenten aufgerufen:

```
$ anzg.param eins zwei 3 4 5 sechs 7 8 9 10 11 <CR>
Die Parameter für dieses Kommando sind: eins zwei 3 4 5 sechs 7 8 9 10 11
$
```

Auch hier zeigt **anzg.param** alle übergebenen Argumente auf dem Bildschirm an. Der Parameter **\$*** ist insbesondere nützlich, wenn man Argumente an das Shell-Kommando mit der Dateinamen-Expansion übergibt.

Im folgenden Beispiel wird die Dateinamen-Expansion mit dem Kommando **anzg.param** eingesetzt. Angenommen, es stehen mehrere Dateien in einem Verzeichnis, die nach den Kapiteln eines Buches benannt sind, **kap1**, **kap2** usw. bis **kap7**, wird durch **anzg.param** eine Liste dieser Dateien ausgegeben:

```
$ anzg.param kap7 <CR>
Die Parameter für dieses Kommando sind: kap1 kap2 kap3 kap4 kap5
kap6 kap7
$
```

In Abbildung 7-17 wird das Shell-Programm `anzg.param` zusammenfassend dargestellt.

Shell-Programm	
<code>anzg.param</code> – Alle Positionsparameter anzeigen (vom Benutzer definiert)	
<i>Kommando</i>	<i>Argumente</i>
<code>anzg.param</code>	(beliebige Positionsparameter)
Beschreibung:	Mit <code>anzg.param</code> werden alle angegebenen Parameter angezeigt.
Bemerkungen:	Werden Ersatzparameter für Dateinamen eingegeben, werden alle zutreffenden Dateinamen angezeigt.

Abbildung 7-17: Übersicht über das Shell-Programm `anzg.param`

Benannte Variablen

Eine weitere Form von Variablen, die in einem Shell-Programm verwendet werden können, sind benannte Variablen. Benannten Variablen werden Werte vom Benutzer selbst zugewiesen. Dabei ist folgendes Format einzuhalten:

benannte Variable = Wert <CR>

Vor oder nach dem Gleichheitszeichen (=) dürfen keine Leerzeichen stehen.

Im folgenden Beispiel heißt die benannte Variable `var1`, und der Wert bzw. die Zeichenkette, der/die dieser Variablen zugewiesen wird, ist `meinname`.

`var1=meinname<CR>`

In einem Shell-Programm wird der Wert der Variablen durch ein vorangestelltes Dollarzeichen (\$) gekennzeichnet. Nach dem obigen Beispiel wird die Shell durch die Angabe `$var1` im Programm angewiesen, den Wert `meinname` bei jedem Auftreten der Zeichenkette `$var1` für diese einzusetzen.

Das erste Zeichen eines Variablennamens muß stets ein Buchstabe oder ein Unterstrich sein. Der Rest des Namens kann sich aus Buchstaben, Unterstrichen und Ziffern zusammensetzen. Wie für die Dateinamen von Shell-Programmen ist es auch hier nicht ratsam, Namen bestehender Shell-Kommandos als Variablennamen zu wählen. Daneben sind in der Shell bereits bestimmte Variablennamen reserviert; auch sie sollten nicht für Variablen verwendet werden, die vom Benutzer selbst definiert werden. Im folgenden eine kurze Erläuterung dieser reservierten Shell-Variablen:

- **CDPATH** definiert den Suchpfad für das Kommando `cd`.
- **HOME** ist die Standardvariable für das Kommando `cd` (Home-Verzeichnis).
- **IFS** definiert die internen Feldtrennzeichen (normalerweise Leerzeichen, Tabulator- und Carriage-Return).
- **LOGNAME** ist der Benutzername.
- **MAIL** gibt die Datei an, die die elektronische Post für den Benutzer enthält.
- **PATH** bestimmt den Suchpfad für die Shell zum Aufrufen von Kommandos.

- **PS1** definiert das primäre Bereit-Zeichen (Standardwert ist \$).
- **PS2** definiert das sekundäre Bereit-Zeichen (Standardwert ist >).
- **TERM** gibt den Terminaltyp an. Es ist wichtig, diese Variable zu definieren, wenn man mit dem Editor *vi* arbeitet.
- **TERMINFO** gibt das Verzeichnis an, in dem nach den Informationen über das Terminal gesucht werden soll, wie beispielsweise nach der Bildschirmgröße.
- **TZ** definiert die Zeitzone (Standardwert ist **EST5EDT**).

Die meisten dieser Variablen werden unter "Anpassen der Benutzerumgebung" weiter unten in diesem Kapitel näher erläutert. Weitere Informationen dazu sind unter *sh(1)* im *User's Reference Manual* zu finden.

Den Wert dieser Variablen kann man sich in der Shell in zwei Formen anzeigen lassen. In der ersten Form ist einzugeben:

```
echo $Variablenname
```

Das System gibt daraufhin den Wert von *Variablenname* aus. Die zweite Möglichkeit besteht darin, die Werte aller in der Shell definierten Variablen mit dem Kommando *env(1)* ausgeben zu lassen. Dazu ist *env* alleine in eine Kommandozeile einzugeben; daraufhin gibt das System eine Liste der Variablennamen und ihrer Werte aus.

Variablenwerte zuweisen

Mit dem Editor *vi* kann man die Variable **TERM** durch folgende Kommandozeile definieren:

```
TERM = Terminalname <CR>
```

Dies ist die einfachste Möglichkeit, einer Variablen einen Wert zuzuweisen.

Es gibt jedoch noch mehrere weitere Möglichkeiten:

- Mit dem Kommando **read** der Variablen eine Eingabe zuweisen.
- Die Ausgabe eines Kommandos in eine Variable umlenken; dazu wird die Kommandosubstitution mit den Akzentzeichen (` ... `) verwendet.
- Der Variablen einen Positionsparameter zuweisen.

In den folgenden Abschnitten werden diese Methoden im Einzelnen erläutert.

Das Kommando read

Mit dem Kommando **read** kann ein Benutzer im Programm zur Eingabe von Werten für Variablen aufgefordert werden. Das allgemeine Format für das Kommando **read** lautet:

```
read Variable <CR>
```

Die mit **read** der *Variablen* zugewiesenen Werte werden überall im Programm für *\$Variable* eingesetzt. Mit einem Kommando **echo** unmittelbar vor dem Kommando **read** kann man im Programm Anweisungen an den Benutzer geben, wie zum Beispiel Eingabe: Das Kommando **read** wird erst ausgeführt, wenn man eine Zeichenkette eingegeben und anschließend die RETURN-Taste gedrückt hat; dann wird diese Zeichenkette der Variablen als Wert zugewiesen.

Im folgenden Beispiel wird gezeigt, wie man mit einem einfachen Shell-Programm mit dem Namen **tel.num** Telefonnummern abrufen kann. Dabei werden folgende Kommandos verwendet:

echo	Als Aufforderung zur Eingabe des Nachnamens des Gesprächspartners
read	Zur Übergabe des Eingabewertes an die Variable name
grep	Zum Durchsuchen der Datei liste nach dieser Variablen

Das Programm sollte ungefähr so aussehen:

```
$ cat tel.num <CR>
echo Nachnamen eingeben:
read name
grep $name liste
$
```

Eine Datei mit dem Namen **liste** mit mehreren Nachnamen und Telefonnummern erstellen und anschließend **tel.num** ausführen.

Im folgenden Beispiel wird ein Programm mit dem Namen **telnum.eing** erstellt; mit ihm kann man eine Liste anlegen. In **telnum.eing** werden folgende Kommandos verwendet:

- **echo** fordert den Benutzer zur Eingabe eines Namens auf
- **read** weist der Variablen *name* den Namen des Gesprächspartners zu
- **echo** fordert zur Eingabe der Telefonnummer des Gesprächspartners auf
- **read** weist die Telefonnummer der Variablen *num* zu
- **echo** trägt die Werte der Variablen *name* und *num* in die Datei **liste** ein.

Soll die Ausgabe des Kommandos **echo** an das Ende der Datei **liste** angehängt werden, muß sie mit **>>** umgelenkt werden. Verwendet man **>**, wird **liste** nur die zuletzt eingegebene Telefonnummer enthalten.

Den Inhalt des Programms **telnum.eing** kann man sich mit dem Kommando **cat** anzeigen lassen. Sieht das Programm wie im folgenden Bildschirm aus, kann man es in ein ausführbares Programm umwandeln (mit dem Kommando **chmod**):

```
$ cat telnum.eing<CR>
echo Namen eingeben:
read name
echo Telefonnummer eingeben:
read num
echo $name $num >> liste
$ chmod u+x telnum.eing<CR>
$
```

Probieren Sie nun die neuen Programme an einer eigenen Telefonliste aus. Im folgenden Beispiel wird mit **telnum.eing** ein neuer Eintrag für Herrn Liebermann eingegeben. Danach wird seine Telefonnummer mit **tel.num** abgerufen:

```

$ telnum.eing<CR>
Namen eingeben:
Herr Liebermann<CR>
Telefonnummer eingeben:
668-0007<CR>
$ tel.num<CR>
Nachnamen eingeben:
Liebermann<CR>
Herr Liebermann 668-0007
$

```

Die Variable `name` akzeptiert **Herr** und **Liebermann** als einen Wert.

In den Abbildungen 7-18 und 7-19 werden die Shell-Programme `telnum.eing` bzw. `tel.num` kurz zusammengefasst.

Shell-Programm	
<code>telnum.eing</code> – Namen und Telefonnummer in eine Liste eingeben	
<i>Kommando</i>	<i>Argumente</i>
<code>telnum.eing</code>	(interaktiv)
Beschreibung:	Das Programm fordert zur Eingabe des Namens und der Telefonnummer einer Person auf und trägt die Werte in eine Liste ein.
Bemerkungen:	Das Kommando arbeitet interaktiv.

Abbildung 7-18: Übersicht über das Shell-Programm `telnum.eing`

Shell-Programm	
tel.num – Name und Telefonnummer einer Person anzeigen	
<i>Kommando</i>	<i>Argumente</i>
tel.num	(interaktiv)
Beschreibung:	Das Programm fordert zur Eingabe des Nachnamens einer Person auf und zeigt dann den vollen Namen sowie die Telefonnummer der betreffenden Person an.
Bemerkungen:	Das Kommando arbeitet interaktiv.

Abbildung 7-19: Übersicht über das Shell-Programm **tel.num****Einsetzen einer Kommandoausgabe für einen Variablenwert**

Die Ausgabe eines Kommandos kann für einen Variablenwert eingesetzt werden. Dazu verwendet man die *Kommandosubstitution* im folgenden Format:

Variable = `Kommando` <CR>

Die Ausgabe von *Kommando* wird dadurch zum Wert der *Variablen*.

In einem der weiter oben angeführten Beispiele zur Verwendung von Pipes wurde die Ausgabe des Kommandos **date** über eine Pipe in das Kommando **cut** eingegeben, um die Uhrzeit zu erhalten. Die Kommandozeile dafür lautete:

date | cut -c12-19<CR>

Diese Kommandos können in ein einfaches Shell-Programm mit dem Namen **z** geschrieben werden; damit kann man sich dann die Zeit anzeigen lassen.


```
$ cat z<CR>
time=`date | cut -c12-19`
echo Uhrzeit: $time
$
```

Vor oder nach dem Gleichheitszeichen dürfen keine Leerzeichen stehen. Nachdem die Datei in ein ausführbares Programm umgewandelt wurde, kann man sich die Zeit anzeigen lassen:

```
$ chmod u+xz<CR>
$ z<CR>
Uhrzeit: 10:36
$
```

In Abbildung 7-20 wird das benutzerdefinierte Programm `z` zusammengefaßt.

Shell-Programm z – Anzeigen der Uhrzeit	
<i>Kommando</i>	<i>Argumente</i>
t	keine
Beschreibung:	Mit z kann man sich die Uhrzeit in Stunden und Minuten anzeigen lassen.

Abbildung 7-20: Übersicht über das Shell-Programm z

Zuweisen von Werten durch Positionsparameter

Ein Positionsparameter kann einem benannten Parameter zugewiesen werden; dabei ist folgendes Format zu verwenden:

```
var1=$1<CR>
```

Im folgenden Beispiel wird mit einem einfachen Programm mit dem Namen `zuweis.parm` einer Variablen ein Positionsparameter zugewiesen. Im folgenden Bildschirm sind die Kommandos zu sehen, die in `zuweis.parm` verwendet werden:

```
$ cat zuweis.parm<CR>
var1=$1
echo $var1
$
```

Selbstverständlich kann man auch die Ausgabe eines Kommandos, das mit Positionsparametern arbeitet, einer Variablen zuweisen:

```
person=`who | grep $1` <CR>
```

Im folgenden Beispiel protokolliert das Programm `prot.zeit` die Ergebnisse von Aufrufen des Programms `werda`. Die Ausgabe von `werda` wird der Variablen `person` zugewiesen und mit dem Kommando `echo` in die Datei `benutzer.datei` eingefügt. Das letzte Kommando `echo` dient dazu, den Wert von `$person` anzuzeigen; es ist derselbe Wert wie bei der Ausgabe aus dem Kommando `werda`:

```
$ cat prot.zeit <CR>
person=`who | grep $1`
echo $person >> benutzer.datei
echo $person
$
```

Die Antwort des Systems auf Eingabe von `prot.zeit` wird auf dem folgenden Bildschirm dargestellt:

```
$ prot.zeit marianne <CR>
marianne  tty61      Apr 11 10:26
$
```

In Abbildung 7-21 wird das Shell-Programm `prot.zeit` zusammengefaßt.

Shell-Programm	
prot.zeit – Protokollieren und Anzeigen eines Benutzernamens (vom Benutzer definiert)	
<i>Kommando</i>	<i>Argumente</i>
prot.zeit	<i>Benutzername</i>
Beschreibung:	Ist der Benutzer mit dem angegebenen Benutzernamen gerade am System angemeldet, schreibt <code>prot.zeit</code> die Ausgabe des Kommandos <code>who</code> in die Datei <code>benutzer.datei</code> und zeigt die Zeile außerdem auf dem Terminal an.

Abbildung 7-21: Übersicht über das Shell-Programm `prot.zeit`

Shell-Programmstrukturen

In der Shell-Programmiersprache stehen verschiedene Strukturen zur Verfügung, die die Flexibilität von Programmen steigern:

- Mit Kommentaren kann man Funktionen im Programm dokumentieren.
- Mit dem "Here Document" kann man Zeilen in das Shell-Programm integrieren, die zur Eingabe in ein Kommando des Shell-Programms umgelenkt werden sollen.
- Mit dem Kommando **exit** kann man ein Programm vorzeitig verlassen und dabei Rückgabecodes ausgeben lassen.
- Mit den Schleifenstrukturen **for** und **while** kann man Gruppen von Kommandos wiederholt durchlaufen lassen, die in einer Schleife angeordnet sind.
- Die bedingten Steuerkommandos **if** und **case** führen eine Gruppe von Kommandos nur aus, wenn bestimmte Bedingungen erfüllt sind.
- Mit dem Kommando **break** kann ein Programm eine Schleife ohne Bedingung verlassen.

Kommentare

Kommentare können in ein Shell-Programm in zwei Formen eingegeben werden. Text, der in einer Zeile nach einem Nummernzeichen (#) steht, wird vom Shell-Programm bei der Ausführung ignoriert. Das Nummernzeichen kann am Anfang einer Zeile oder nach einem Kommando stehen. Im ersten Fall wird die gesamte Zeile ignoriert, im zweiten Fall wird das Kommando noch ausgeführt, der Rest der Zeile wird dagegen ignoriert. Ein Kommentar endet stets am Zeilenende. Das allgemeine Format für eine Kommentarzeile lautet:

```
#Kommentar<CR>
```

In einem Programm, das beispielsweise folgende Zeilen enthält, werden diese bei der Ausführung ignoriert:

```
# Dieses Programm übermittelt einen Geburtstagsgruß.<CR>
# Bei diesem Programm ist ein Benutzername<CR>
# als Positionsparameter anzugeben.<CR>
```

Kommentare sind wichtig für die Dokumentation von Programmfunktionen und sollten in jedes Programm eingefügt werden, das der Benutzer schreibt.

Das Here Document

Mit einem "Here Document" können in ein Shell-Programm Zeilen aufgenommen werden, die als Eingabe für ein Kommando in diesem Programm verwendet werden sollen. Damit verfügt man über eine Möglichkeit, eine Eingabe für ein Kommando im Shell-Programm zu liefern, ohne eine separate Datei zu verwenden. Die Syntax besteht aus dem Umlenkungszeichen << und einem Begrenzungszeichen, mit dem Anfang und Ende der Eingabezeilen eingegrenzt werden. Als Begrenzungszeichen kann ein Zeichen oder eine Zeichenkette verwendet werden; häufig wird dafür das Ausrufezeichen (!) verwendet.

In Abbildung 7-22 wird das allgemeine Format für ein "Here Document" dargestellt.

```
Kommando <<Begrenzungszeichen <CR>  
...Eingabezeilen...<CR>  
Begrenzungszeichen <CR>
```

Abbildung 7-22: Format eines "Here Document"

Im folgenden Beispiel wird im Programm **gebtag** mit einem "Here Document" ein allgemeiner Geburtstagsgruß übermittelt, indem Eingabezeilen in das Kommando **mail** umgelenkt werden:

```
$ cat gebtag<CR>
mail $1 <<!
Die besten Wünsche zum Geburtstag.
!
$
```

Beim Aufrufen dieses Kommandos muß der Benutzername des Empfängers als Argument für das Kommando angegeben werden. Der folgende Textteil ist die Eingabe, die hier mit dem "Here Document" integriert wird:

Die besten Wünsche zum Geburtstag.

Diesen Geburtstagsgruß kann man beispielsweise an den Benutzer **maria** übermitteln; dazu geben Sie folgendes ein:

```
$ gebtag maria<CR>
```

Der Benutzer **maria** wird den Gruß erhalten, sobald er das nächste Mal die per elektronischer Post eingegangenen Nachrichten liest:

```
$ mail<CR>
From meinname Wed May 14 14:31 MES 1989
Die besten Wünsche zum Geburtstag
$
```

In Abbildung 7-23 werden das Format und die Funktionen des Kommandos **gebtag** zusammengefaßt.

Shell-Programm	
gebtag - Übermitteln eines Geburtstagsgrußes (vom Benutzer definiert)	
<i>Kommando</i>	<i>Argumente</i>
gebtag	<i>Benutzername</i>
Beschreibung:	Mit gebtag wird ein allgemeiner Geburtstagsgruß an den Benutzer übermittelt, der als Argument angegeben wurde.

Abbildung 7-23: Übersicht über das Kommando **gebtag**

Aufrufen des Editors **ed** in einem Shell-Programm

Mit dem "Here Document" kann man den Editor **ed** in einem Shell-Skript aufrufen. Man kann beispielsweise ein Shell-Programm erstellen, das folgende Schritte ausführt:

- Editor **ed** aufrufen
- globale Ersetzung in einer Datei durchführen
- die Datei speichern
- den Editor verlassen

Auf dem folgenden Bildschirm wird ein Programm mit dem Namen **ers.text** dargestellt, das genau diese Funktion hat.

```
$ cat ers.text<CR>
echo Dateiname eingeben:
read datei1
echo Text, der geändert werden soll, genau eingeben:
read alt_text
echo Neuen Text genau eingeben:
read neu_text
ed - $datei1 <<!
g/$alt_text/s//$neu_text/g
w
q
!
$
```

Hier ist die Option - (minus) des Kommandos `ed` zu beachten. Damit wird verhindert, daß die Anzahl der Zeichen auf dem Bildschirm ausgegeben wird. Auch das genaue Format des Editorkommandos zur globalen Ersetzung ist zu beachten:

```
g/alt_text/s//neu_text/g<CR>
```

In diesem Programm werden drei Variablen verwendet: *datei1*, *alt_text* und *neu_text*. Nachdem das Programm aufgerufen wird, holt es sich mit dem Kommando `read` die Werte dieser Variablen. Diese liefern folgende Informationen:

datei Name der Datei, die bearbeitet werden soll
alt_text Text, der ersetzt werden soll (alter Text)
neu_text Text, der den alten Text ersetzen soll (neuer Text)

Nach der Eingabe der Variablen in das Programm werden die globale Ersetzung, das Kommando zum Speichern (`w`) und das Kommando zum Verlassen des Editors vom "Here Document" als Eingabe in das Kommando `ed` umgelenkt. Probieren Sie das neue Kommando `ers.text` aus. Auf dem folgenden Bildschirm werden Beispiele für Eingaben gezeigt, nach denen das Programm den Benutzer fragt:

```

$ ers.text <CR>
Dateiname eingeben:
info <CR>
Text, der geändert werden soll, genau eingeben:
Hallo Johannes: <CR>
Neuen Text genau eingeben:
An alle Mitarbeiter: <CR>
$ cat info <CR>
An alle Mitarbeiter:
$

```

Die Ergebnisse der globalen Ersetzung kann man sich mit dem Kommando `cat` anzeigen lassen.

In Abbildung 7-24 werden das Format und die Funktionen des Kommandos `ers.text` zusammengefasst.

Shell-Programm	
ers.text – Text in einer Datei ändern	
<i>Kommando</i>	<i>Argumente</i>
ers.text	(interaktiv)
Beschreibung:	Ersetzen von Text in einer Datei durch neuen Text.
Bemerkungen:	Dieses Shell-Programm arbeitet interaktiv, d. h. es fordert zur Eingabe der Argumente auf.

Abbildung 7-24: Übersicht über das Kommando `ers.text`

Nähere Informationen zum Editor `ed` sind in Kapitel 5, "Anleitung zum Zeileneditor (`ed`)" zu finden. Beim Programmieren mit der Shell kann auch der Editor im Prozedurbetrieb `sed` verwendet werden.

Rückgabecodes

Die meisten Shell-Kommandos geben Rückgabecodes aus, mit denen angezeigt wird, ob sie richtig ausgeführt wurden. Als Konvention gilt, daß der Rückgabecode 0 (Null) bedeutet, daß das Kommando richtig ausgeführt wurde; alle anderen Werte bedeuten, daß er nicht korrekt ausgeführt wurde. Der Rückgabecode wird nicht automatisch angezeigt, sondern steht als Wert des Sonderparameters `$?` der Shell zur Verfügung.

Abfragen von Rückgabecodes

Nachdem man ein Kommando interaktiv ausgeführt hat, kann man seinen Rückgabecode durch folgende Eingabe anzeigen lassen:

```
echo $?
```

Dies wird im folgenden Beispiel veranschaulicht:

```
$ cat hallo
Dies ist die Datei hallo.
$ echo $?
0
$ cat hello
cat: cannot open hello
$ echo $?
2
$
```

Im ersten Fall ist die Datei `hallo` im betreffenden Verzeichnis vorhanden und das Leserecht für den Benutzer besteht. Das Kommando `cat` wird wie erwartet ausgeführt und gibt den Inhalt der Datei aus. Es wird mit dem Rückgabecode 0 beendet; diesen kann man mit dem `$?` abrufen. Im zweiten Fall ist die Datei entweder nicht vorhanden oder der Benutzer hat kein Leserecht. Das Kommando `cat` gibt eine Diagnosemeldung aus und wird mit einem Rückgabecode von 2 beendet.

Verwendung von Rückgabecodes mit dem Kommando `exit`

Ein Shell-Programm endet normalerweise mit der Ausführung des letzten Kommandos in der Datei. Man kann das Programm jedoch auch mit dem Kommando `exit` an einer beliebigen anderen Stelle beenden. Der wichtigste Aspekt dabei ist, daß man mit dem Kommando `exit` Rückgabecodes für ein Shell-Programm ausgeben lassen kann. Weitere Informationen zu `exit` sind unter `exit(2)` im *Programmer's Reference Manual* enthalten.

Schleifen

In den bisher in diesem Kapitel dargestellten Beispielen wurden die Kommandos der Shell-Programme sequentiell, d. h. einer nach dem andern, ausgeführt. Mit den Schleifenstrukturen `for` und `while` ist es möglich, ein Kommando oder eine Kommandofolge mehrmals auszuführen.

Die Schleife mit `for`

In der Schleife mit `for` wird eine Kommandofolge einmal für jede Einheit einer Liste ausgeführt. Sie hat folgendes Format:

```
for Variable <CR>
  in Werteliste <CR>
do <CR>
  Kommando 1 <CR>
  Kommando 2 <CR>
  .
  .
  letztes Kommando <CR>
done <CR>
```

Abbildung 7-25: Format der Schleifenstruktur `for`

Bei jedem Schleifendurchlauf wird der nächste Wert der Liste der Variablen in der Zeile `for` zugeordnet. Innerhalb der Anweisung `do` kann diese Variable für jedes Kommandos beliebig verwendet werden.

Ein Shell-Programm ist einfacher zu lesen, wenn die Schleifenstrukturen übersichtlich angeordnet werden. Da die Shell Leerzeichen am Zeilenanfang ignoriert, kann jeder Programmabschnitt, wie oben dargestellt, eingerückt werden. Dadurch wird es auch einfacher, festzustellen, ob zu jedem **do** das entsprechende **done** am Schleifenende vorhanden ist.

Als Variable kann ein beliebiger Name gewählt werden. Nennt man sie beispielsweise **var**, werden die Werte der nach dem Schlüsselwort **in** angegebenen Liste nacheinander der Variablen **var** zugewiesen; Verweise der Form **\$var** im Programm liefern dann den betreffenden Wert. Wird **in** weggelassen, bestehen die Werte für **var** aus der gesamten Gruppe der an das Kommando übergebenen Argumente, die mit dem Sonderparameter **\$*** abgerufen werden können. Die Kommandoliste zwischen den Schlüsselwörtern **do** und **done** wird jeweils einmal für jeden Wert ausgeführt.

Nachdem die Kommandos mit dem letzten Wert der Liste ausgeführt wurden, fährt das Programm mit der Zeile, die auf **done** folgt, fort. Ist keine weitere Zeile mehr vorhanden, endet das Programm.

Am einfachsten lernt man Shell-Programmstrukturen kennen, indem man an einem Beispiel übt: Erstellen Sie ein Programm, mit dem Dateien in ein anderes Verzeichnis verschoben werden können. Setzen Sie dabei folgende Kommandos ein:

echo	Den Benutzer auffordern, den Pfadnamen des neuen Verzeichnisses einzugeben.
read	Den Pfadnamen der Variablen pfad zuweisen.
for Variable	Aufrufen der Variablen datei ; auf sie kann mit \$datei in der Kommandofolge verwiesen werden.
in Werteliste	Zur Eingabe einer Liste von Werten. Wird in weggelassen, gilt als Werteliste \$* , d. h. alle in der Kommandozeile eingegebenen Argumente.
do Kommandofolge	Ein oder mehrere Kommandos, die ausgeführt werden sollen. Die Struktur in diesem Programm ist: <pre>do mv \$datei \$pfad/\$datei<CR> done</pre>

Auf dem folgenden Bildschirm wird der Text für das Shell-Programm `vs.datei` dargestellt:

```
$ cat vs.datei<CR>
echo Bitte den Verzeichnispfad eingeben:
read pfad
for datei
  in info1 info2 info3
do
  mv $datei $pfad/$datei
done
$
```

In diesem Programm sind die Werte für die Variable `datei` bereits im Programm enthalten. Sollen beim Aufrufen des Programms verschiedene Dateien angegeben werden, sind die Werte mit Positionsparametern oder mit dem Kommando `read` zuzuweisen. Bei Verwendung von Positionsparametern ist das Schlüsselwort `in` nicht erforderlich; dies wird auf dem folgenden Bildschirm gezeigt:

```
$ cat vs.datei<CR>
echo Verzeichnispfad eingeben:
read pfad
for datei
do
  mv $datei $pfad/$datei
done
$
```

Mit diesem Kommando kann man mehrere Dateien gleichzeitig verschieben, indem man eine Liste von Dateinamen als Argumente mit dem Kommando eingibt. Am einfachsten geschieht dies über die weiter oben bereits beschriebene Dateinamen-Expansion.

In Abbildung 7-26 wird das Shell-Programm `vs.datei` kurz zusammengefaßt.

Shell-Programm	
vs.datei – Verschieben von Dateien in ein anderes Verzeichnis (vom Benutzer definiert)	
<i>Kommando</i>	<i>Argumente</i>
vs.datei	<i>Dateinamen</i> (interaktiv)
Beschreibung:	Dateien werden in ein anderes Verzeichnis verschoben.
Bemerkungen:	Bei diesem Programm müssen Dateinamen als Argumente an das Kommando übergeben werden. Das Programm fordert zur Eingabe des Pfades für das neue Verzeichnis auf.

Abbildung 7-26: Übersicht über das Shell-Programm `vs.datei`

Die Schleife mit `while`

Für eine weitere Schleifenstruktur, die Schleife mit `while`, werden zwei Gruppen von Kommandos verwendet. Dabei wird die Kommandofolge der zweiten Gruppe, der Gruppe `do...done`, ausgeführt, solange das letzte Kommando in der ersten Gruppe, der `while`-Gruppe, den Wert wahr liefert (d. h. das Kommando kann ausgeführt werden).

Das allgemeine Format der **while**-Schleife wird in Abbildung 7-27 dargestellt.

```
while<CR>
  Kommando 1<CR>
  .
  .
  .
  letztes Kommando<CR>
do<CR>
  Kommando 1<CR>
  .
  .
  .
  letztes Kommando<CR>
done<CR>
```

Abbildung 7-27: Format der **while**-Schleife

Im Programm mit dem Namen **eing.name** des folgenden Beispiels wird eine Schleife mit **while** zur Eingabe einer Liste von Namen in eine Datei verwendet. Das Programm besteht aus folgenden Kommandozeilen:

```
$ cat eing.name<CR>
while
  read x
do
  echo $x>>xdatei
done
$
```

Das richtige Programm sieht später etwa so aus:

```
$ cat eing.name<CR>
echo Bitte die Namen der Personen, jeweils gefolgt von <CR>, eingeben:
echo Am Ende der Namensliste <^d> eingeben.
while read x
do
    echo $x>>xdatei
done
echo xdatei enthält folgende Namen:
cat xdatei
$
```

Nachdem die Schleife beendet ist, führt das Programm die Kommandos nach **done** aus.

In den beiden ersten **echo**-Kommandozeilen wurden Sonderzeichen verwendet; um ihre Sonderbedeutung aufzuheben, müssen Anführungszeichen verwendet werden. Auf dem folgenden Bildschirm werden die Ergebnisse des Programms **eing.name** dargestellt:

```
$ eing.name<CR>
Bitte die Namen der Personen, jeweils gefolgt von <CR>, eingeben:
Am Ende der Namensliste <^d> eingeben.
Marianne<CR>
Jeanette<CR>
<^d>
xdatei enthält folgende Namen:
Marianne
Jeanette
$
```

Nach Beendigung der Schleife gibt das Programm alle in `xdatei` enthaltenen Namen aus.

Der Papierkorb der Shell: `/dev/null`

Im Dateisystem gibt es eine Datei mit dem Namen `/dev/null`, in die man unerwünschte Ausgaben leiten kann.

Im folgenden Beispiel wird die Ausgabe des Kommandos `who` nach `/dev/null` umgelenkt. Zuerst das Kommando `who` normal eingeben. Das System zeigt an, wer am System angemeldet ist. Dann das Kommando `who` nochmals eingeben, diesmal jedoch die Ausgabe in die Datei `/dev/null` umlenken:

```
who > /dev/null<CR>
```

Das System gibt daraufhin ein Bereit-Zeichen aus. Die Ausgabe aus dem Kommando `who` wurde in die Datei `/dev/null` geleitet, d. h. sie wurde wirkungslos abgeleitet.

Bedingte Programmstrukturen

`if...then`

Mit dem Kommando `if` wird das Shell-Programm angewiesen, die Kommandofolge nach `then` nur auszuführen, wenn das letzte Kommando der Kommando-Liste nach `if` erfolgreich ausgeführt wurde. Die `if`-Programmstruktur endet mit dem Schlüsselwort `fi`.

Das allgemeine Format für die Programmstruktur `if` wird in Abbildung 7-28 dargestellt.

```
if<CR>
  Kommando1<CR>
  .
  .
  letztes Kommando<CR>
then<CR>
  Kommando1<CR>
  .
  .
  letztes Kommando<CR>
fi<CR>
```

Abbildung 7-28: Format der bedingten Programmstruktur **if...then**

Mit dem folgenden Shell-Programm **suche** wird die Verwendung der Programmstruktur **if...then** gezeigt. In **suche** wird das Kommando **grep** eingesetzt, um nach einem Wort in einer Datei zu suchen. Findet **grep** das Wort, zeigt das Programm mit dem Kommando **echo** an, daß das Wort in der Datei gefunden wurde:

```
$ cat suche<CR>
echo Wort und Dateiname eingeben.
read wort datei
if grep $wort $datei
then echo $wort steht in $datei
fi
$
```

Mit dem Kommando **read** werden hier zwei Variablen Werte zugewiesen. Die erste eingegebene Zeichenkette bis zum ersten Leerzeichen wird der Variablen **wort** zugewiesen. Die übrigen Zeichen werden einschließlich eventueller Leerzeichen der Variablen **datei** zugewiesen.

In diesem Programm stellt sich das Problem einer unerwünschten Ausgabe aus dem Kommando **grep**, die auf dem Bildschirm angezeigt wird. Soll die Antwort des Systems auf das Kommando **grep** des Programms nicht angezeigt werden, ist die Datei **/dev/null** zu verwenden, indem man die Kommandozeile von **if** wie folgt ändert:

```
if grep $wort $datei > /dev/null<CR>
```

Rufen Sie nun das Programm **suche** auf. Es sollte nur die nach dem Kommando **echo** eingegebene Meldung erscheinen.

if...then...else

Bei der Programmstruktur **if...then** kann mit dem zusätzlichen Kommando **else** in eine alternative Kommandofolge verzweigt werden, wenn die Kommandofolge von **if** den Wert "falsch" liefert. Dabei gilt folgendes allgemeine Format:

```
if<CR>
  Kommando1<CR>
  .
  .
  letztes Kommando<CR>
then<CR>
  Kommando1<CR>
  .
  .
  letztes Kommando<CR>
else<CR>
  Kommando1<CR>
  .
  .
  letztes Kommando<CR>
fi<CR>
```

Abbildung 7-29: Format der bedingten Programmstruktur **if...then...else**

Dieses Programm *suche* kann man nun noch weiter verbessern, so daß es auch meldet, wenn es ein Wort nicht gefunden hat. Auf dem folgenden Bildschirm wird eine so verbesserte Version des Programms gezeigt:

```

$ cat suche<CR>
echo Wort und Dateiname eingeben.
read wort datei
if
  grep $wort $datei >/dev/null
then
  echo $wort steht in $datei
else
  echo $wort steht NICHT in $datei
fi
$

```

In Abbildung 7-30 wird das erweiterte Programm `suche` kurz zusammengefaßt.

Shell-Programm	
<code>suche</code> - Anzeigen, ob ein Wort in einer Datei steht (vom Benutzer definiert)	
<i>Kommando</i>	<i>Argumente</i>
<code>suche</code>	interaktiv
Beschreibung:	Meldet, ob ein bestimmtes Wort in einer bestimmten Datei steht.
Bemerkungen:	Das Programm fordert zur Eingabe der Argumente (Wort und Datei) auf.

Abbildung 7-30: Übersicht über das Shell-Programm `suche`

Das Prüfkommando für Schleifen test

Mit dem Kommando **test** kann man prüfen, ob bestimmte Bedingungen erfüllt sind; das Kommando eignet sich für bedingte Programmstrukturen. Ist eine Bedingung erfüllt (wahr), wird die Schleife weitergeführt, ist die Bedingung nicht erfüllt (falsch), wird die Schleife beendet und das nächste Kommando ausgeführt. Für das Kommando **test** stehen einige nützliche Optionen zur Verfügung:

test -r datei <CR>	Wahr, wenn die Datei existiert und Leserecht definiert ist.
test -w datei <CR>	Wahr, wenn die Datei existiert und Schreibrecht definiert ist.
test -x datei <CR>	Wahr, wenn die Datei existiert und Ausführbarkeitsrecht besteht.
test -s datei <CR>	Wahr, wenn die Datei existiert und mindestens ein Zeichen enthält.
test var1 -eq var2 <CR>	Wahr, wenn <i>var1</i> gleich <i>var2</i> ist.
test var1 -ne var2 <CR>	Wahr, wenn <i>var1</i> ungleich <i>var2</i> ist.

Im folgenden soll ein Shell-Programm erstellt werden, mit dem alle ausführbaren Dateien des aktuellen Verzeichnisses in das Verzeichnis **bin** verschoben werden. Mit dem Kommando **test -x** kann man dabei die ausführbaren Dateien auswählen. Auf dem folgenden Bildschirm wird die Programmstruktur mit **for** dargestellt, die in der Datei **vs.datei** verwendet wurde:


```
$ cat vs.datei<CR>
echo Verzeichnispfad eingeben:
read pfad
for datei
do
  mv $datei $pfad/$datei
done
$
```

Auf dem folgenden Bildschirm wird das Programm `vs.ex` dargestellt, in das eine Anweisung `if test -x` in die Schleife `do...done` eingefügt wird, so daß nur ausführbare Dateien verschoben werden:

```
$ cat vs.ex<CR>
echo Verzeichnispfad eingeben:
read pfad
for datei
do
  if test -x $datei
  then
    mv $datei $pfad/$datei
  fi
done
$
```

Als Verzeichnispfad ist der Pfad vom aktuellen Verzeichnis zum Verzeichnis `bin` anzugeben. Verwendet man dagegen den Wert für die Shell-Variablen `HOME`, braucht der Pfadname nicht jedesmal erneut eingegeben zu werden. `$HOME` gibt den Pfad zum Benutzerverzeichnis an. `$HOME/bin` gibt den Pfad zum Verzeichnis `bin` an.

Im folgenden Beispiel fordert `vs.ex` nicht dazu auf, den Verzeichnisnamen anzugeben und liest daher auch nicht die Variable `pfad` ein:

```
$ cat vs.ex<CR>
for datei
do
  if test -x $datei
  then
    mv $datei $HOME/bin/$datei
  fi
done
$
```

Probieren Sie das Kommando nun mit allen Dateien im aktuellen Verzeichnis aus; geben Sie dabei ist das Maskierungszeichen `*` als Argument zum Kommando ein. Mit den Kommandozeilen des folgenden Beispiels wird zuerst das Kommando aus dem aktuellen Verzeichnis ausgeführt, dann ins Verzeichnis `bin` gewechselt, und anschließend werden die Dateien in diesem Verzeichnis aufgelistet. Danach sollten sich alle ausführbaren Dateien dort befinden.

```
$ vs.ex *<CR>
$ cd; cd bin; ls<CR>
Liste_der ausführbaren_Dateien
$
```

In Abbildung 7-31 werden das Format und die Funktionen des Shell-Programms `vs.ex` zusammengefaßt.

Shell-Programm	
vs.ex – Verschieben aller ausführbaren Dateien des aktuellen Verzeichnisses in das Verzeichnis bin .	
<i>Kommando</i>	<i>Argumente</i>
vs.ex	* (alle Dateinamen)
Beschreibung:	Alle Dateien des aktuellen Verzeichnisses, für die das Ausführbarkeitsrecht gilt, werden in das Verzeichnis bin verschoben.
Bemerkungen:	Die ausführbaren Dateien im Verzeichnis bin (bzw. jedem in der Variablen PATH definierten Verzeichnis) können von jedem anderen Verzeichnis aus ausgeführt werden.

Abbildung 7-31: Übersicht über das Shell-Programm `vs.ex`

`case..esac`

Bei der Programmstruktur `case...esac` kann aus einem von mehreren Mustern ausgewählt werden; auf dieses Muster wird dann eine Reihe von Kommandos angewendet. Die Anweisung für das Muster muß mit dem Schlüsselwort **in** beginnen, und nach dem letzten Zeichen des Musters muß ein **)** stehen. Die Kommandofolge für jedes Muster wird mit **;;** beendet. Die Programmstruktur `case` wird mit `esac` beendet (Umkehrung von "case").

Das allgemeine Format für die Programmstruktur `case` wird aus Abbildung 7-32 ersichtlich:

```
case wort <CR>
in <CR>
    Muster1) <CR>
        Kommandozeile 1 <CR>
        .
        .
        letzte Kommandozeile <CR>
;; <CR>
Muster2) <CR>
    Kommandozeile 1 <CR>
    .
    .
    letzte Kommandozeile <CR>
;; <CR>
Muster3) <CR>
    Kommandozeile 1 <CR>
    .
    .
    letzte Kommandozeile <CR>
;; <CR>
*) <CR>
    Kommando 1 <CR>
    .
    .
    letztes Kommando <CR>
;; <CR>
esac <CR>
```

Abbildung 7-32: Die bedingte Programmstruktur case...esac

Die Anweisung `case` sucht eine Entsprechung für das `wort` neben dem Wort `case`, und zwar nach dem *Muster* des ersten Abschnitts der Struktur. Findet das Programm eine Entsprechung, führt das Programm die Kommandozeilen des ersten Abschnitts aus (bis zum ersten `;;`).

Trifft das erste Muster nicht zu, versucht es das Programm mit dem zweiten Muster. Sobald eine Entsprechung gegeben ist, werden die weiteren Muster nicht mehr geprüft; das Programm springt dann zu dem Kommando, das auf `esac` folgt.

Setzt man den Stern (*) als Muster ein, gilt jedes *wort* als Entsprechung; damit kann man eine Kommandofolge für den Fall vorsehen, daß keines der vorgegebenen Muster zutrifft. Es muß jedoch an die letzte Stelle der möglichen Muster der Programmstruktur `case` gesetzt werden, damit die anderen Muster zuerst geprüft werden. Damit kann man falsche oder unerwartete Eingaben gut erkennen.

In den *Muster*-Angaben der einzelnen Abschnitte können die Maskierungszeichen *, ? und [] ebenso eingesetzt werden, wie weiter oben in diesem Kapitel für die Dateinamen-Expansion der Shell beschrieben wurde. Damit erreicht man eine hohe Flexibilität.

Das Programm `set.term` enthält ein gutes Beispiel für die Programmstruktur `case...esac`. Mit dem Programm wird die Shell-Variable `TERM` auf den verwendeten Terminaltyp eingestellt. Dabei wird die folgende Kommandozeile eingesetzt:

```
TERM=Terminalname <CR>
```

Eine Erläuterung zu den hier verwendeten Kommandos ist in der Anleitung zu `vi` in Kapitel 6 enthalten. Im folgenden Beispiel handelt es sich um den Terminaltyp `BA47/BA80`.

Das Programm `set.term` prüft zuerst, ob der Wert von `term` "dap4x" ist. Wenn ja, weist das Programm den Wert "dap4x" der Variablen `TERM` zu und ist damit beendet. Ist der Wert von `term` nicht "dap4x", prüft das Programm auf weitere Werte: `ba80-08` und `vt100`. Es führt die Kommandos aus, die unter dem ersten Muster stehen, das zutrifft, und springt dann zum ersten Kommando nach der Anweisung `esac`.

Das Muster *, d. h. alle sonstigen Möglichkeiten, wird am Ende der Muster für den Terminaltyp vorgesehen. Damit wird der Benutzer darauf aufmerksam gemacht, daß für den Terminaltyp kein zutreffendes Muster vorliegt; gleichzeitig kann man damit die Programmstruktur `case` verlassen:

```
$ cat set.term<CR>
echo Bei einem Terminal TTY BA47 folgendes eingeben: dap4x
echo Bei einem Terminal TTY BA80 folgendes eingeben: ba80-08
echo Bei einem Terminal TTY BA47 folgendes eingeben: vt100
read term
case $term
  in
    dap4x)
        TERM=dap4x
        ;;
    ba80-08)
        TERM=dap4x
        ;;
    vt100)
        TERM=vt100
        ;;
    *)
        echo Eingabe nicht zulässig - falscher Terminaltyp
        ;;
  esac
export TERM
echo Programm beendet
$
```

Hier ist das Kommando **export** zu beachten. Mit **export** stellt man eine Variable für die eigene Umgebung und für andere Shell-Prozeduren zur Verfügung. Würde man hier das Muster ***** (alle Werte) an die erste Stelle setzen, könnte das Programm **set.term** der Variablen **TERM** nie einen Wert zuweisen, da das erste Muster, d. h. *****, stets zutrifft.

In Abbildung 7-33 werden das Format und die Funktionen des Shell-Programms **set.term** zusammengefaßt.

Shell-Programm	
set.term - Der Variablen TERM einen Wert zuweisen (vom Benutzer definiert)	
<i>Kommando</i>	<i>Argumente</i>
set.term	interaktiv
Beschreibung:	Mit dem Programm wird der Shell-Variablen TERM ein Wert zugewiesen; dann wird dieser Wert an andere Shell-Prozeduren exportiert.
Bemerkungen:	Das Programm fordert zur Eingabe eines bestimmten Codes für ein Terminal auf, der dann als Muster in der Programmstruktur case eingesetzt wird.

Abbildung 7-33: Übersicht über das Shell-Programm **set.term**

Unbedingte Steueranweisungen: die Kommandos **break** und **continue**

Mit dem Kommando **break** wird die Ausführung jeder Schleife beendet, in der es auftritt, ohne daß eine Bedingung erfüllt sein muß, und das Programm springt zum nächsten Kommando nach der Anweisung **done**, **fi** bzw. **esac**. Steht nach dieser Anweisung kein Kommando mehr, ist das Programm beendet.

Im obigen Beispiel mit **set.term** kann man anstelle des Kommandos **echo** das Kommando **break** einsetzen, um das Programm zu beenden. Dies wird im folgenden Beispiel gezeigt:

```
$ cat set.term<CR>
echo Bei einem Terminal TTY BA47 folgendes eingeben: dap4x
echo Bei einem Terminal TTY BA80 folgendes eingeben: ba80-08
echo Bei einem Terminal TTY BA47 folgendes eingeben: vt100
read term
case $term
  in
    dap4x)
        TERM=dap4x
        ;;
    ba80-08)
        TERM=dap4x
        ;;
    vt100)
        TERM=vt100
        ;;
    *)
        break
    ;;
esac
export TERM
echo Programm beendet
$
```

Das Kommando **continue** bewirkt, daß das Programm unmittelbar mit dem nächsten Durchlauf einer Schleife mit **do** oder **for** beginnt, ohne die übrigen Kommandos der Schleife zuerst auszuführen.

Debugger-Programme

Es kann gelegentlich vorkommen, daß man Fehler in einem Programm suchen und beheben muß. Dazu kann das Kommando **sh** mit zwei Optionen eingesetzt werden:

sh -v Shell-Programm-Name gibt die Eingabezeilen der Shell so aus, wie sie vom System gelesen werden

`sh -x Shell-Programm-Name` gibt Kommandos und ihre Argumente während ihrer Ausführung aus

Diese Optionen kann man ausprobieren, indem man ein Shell-Programm mit einem Fehler erstellt, wie im folgenden Beispiel mit der Datei **fehler**:

```
$ cat fehler<CR>
heute=`date`
echo Person eingeben:
read person
mail $1
$person
Bitte kommen Sie nach dem Abmelden in mein Büro.
$heute.
MLH
$
```

Hier wird **heute** mit der Ausgabe des Kommandos **date** gleichgesetzt; dieses Kommando muß zwischen Akzentzeichen (``...`) gesetzt werden, damit eine Kommandosubstitution erfolgen kann.

Die Nachricht, die per elektronischer Post an Thomas (\$1) unter dem Benutzernamen **tommy** (\$2) geschickt werden soll, sollte folgendermaßen aussehen:

```
$ mail<CR>
From mlh Tue Apr 25 15:16:23 MES 1989
Tom
Bitte kommen Sie nach dem Abmelden in mein Büro.
Wed Apr 26 17:17:19 MES 1989
MLH
?
```

Versucht man nun, das Programm **fehler** auszuführen, wird man es mit der Taste **BREAK** oder **DELETE** abbrechen müssen.

Dieses Programm kann man nun korrigieren, indem man **fehler** mit **sh -v** aufruft. Dadurch werden die Zeilen der Datei so ausgegeben, wie sie vom System gelesen werden:

```
$ sh -v fehler thomas <CR>
heute=`date`
echo Person eingeben:
Person eingeben:
read person
tommy
mail $1
```

Die Ausgabe bricht hier mit dem Kommando **mail** ab, da es mit dem Kommando **mail** ein Problem gibt. Hier muß die Eingabe mit einem "Here Document" in das Kommando **mail** umgelenkt werden.

Vor dem Korrigieren des Programms `fehler` kann man es nun auch mit `sh -x` aufrufen; damit werden die Kommandos und Argumente so ausgegeben, wie sie vom System gelesen werden:

```
$ sh -x fehler thomas tommy<CR>
+date
heute=Thu Apr 10 11:07:23 MES 1989
+ echo Person eingeben:
Person eingeben:
+ read person
tommy
+ mail thomas
$
```

Auch hier bricht das Programm wieder mit dem Kommando `mail` ab. Die Substitutionen mit den Variablen wurden ausgeführt und werden angezeigt.

Das korrigierte Programm `fehler` sieht wie folgt aus:

```
$ cat fehler<CR>
heute=`date`
echo Person eingeben:
read person
mail $1 <<!
$person
Bitte kommen Sie nach dem Abmelden in mein Büro.
$heute
MLH
!
$
```

Das Kommando **tee** eignet sich für die Fehlerbehebung bei Pipelines. Die Standardeingabe wird einfach an die Standardausgabe übergeben, dabei wird jedoch eine Kopie der Eingabe in die Datei geschickt, deren Name als Argument angegeben wurde.

Das allgemeine Format des Kommandos **tee** lautet:

```
Kommando1 | tee Kopie | Kommando2<CR>
```

Kopie steht hier für die Datei, in die die Ausgabe von *Kommando1* kopiert wird, so daß man sie später untersuchen kann.

Im folgenden Beispiel soll die Ausgabe des Kommandos **grep** der folgenden Kommandozeile geprüft werden:

```
who | grep $1 | cut -c1-9<CR>
```

Mit dem Kommando **tee** kann man die Ausgabe von **grep** in eine Datei kopieren (hier **test**), ohne die übrigen Teile der Pipeline zu beeinflussen:

```
who | grep $1 | tee test | cut -c1-9<CR>
```

Die Datei **test** erhält eine Kopie der Ausgabe von **grep**; diese wird auf dem folgenden Bildschirm dargestellt:

```
$ who |grep mlhmo |tee test |cut -c1-9<CR>
mlhmo
$ cat test<CR>
mlhmo  tty61  Apr 10  11:30
$
```

Anpassen der Benutzerumgebung

Im UNIX-System kann die Benutzerumgebung auf verschiedene Arten geändert werden. Ein häufig geändertes Merkmal sind die Standardwerte für die Löschrzeichen # (Erase) und @ (Kill).

Bei der Anmeldung eines Benutzers untersucht die Shell zuerst die Datei im Benutzerverzeichnis mit dem Namen `.profile`. Diese Datei enthält Kommandos, mit denen die Umgebung des Benutzers in der Shell definiert wird.

Die Datei `.profile` kann wie andere Dateien bearbeitet und an die speziellen Anforderungen des Benutzers angepaßt werden. An manchen Systemen kann der Benutzer diese Datei selbst ändern, an anderen Systemen übernimmt der Systemverwalter diese Aufgabe. Mit folgender Eingabe kann man feststellen, ob im eigenen Home-Verzeichnis eine Datei `.profile` enthalten ist:

```
ls -al $HOME
```

Ändert man die Datei selbst, wird man zu Anfang vorsichtig vorgehen wollen. Daher sollte man zuerst eine Kopie davon unter dem Namen `safe.profile` anlegen, bevor man die Datei `.profile` selbst ändert. Geben Sie folgendes ein:

```
cp .profile safe.profile <CR>
```

In die Datei `.profile` können Kommandos wie in jedes andere Shell-Programm eingefügt werden. Weiterhin kann man bestimmte Terminal-Charakteristika mit dem Kommando `stty` und einige Shell-Variablen definieren.

Einfügen von Kommandos in die Datei `.profile`

Zur Übung rufen Sie die Datei `.profile` mit einem Editor aufrufen und geben das folgende Kommando `echo` in die letzte Zeile der Datei ein:

```
echo Guten Morgen! Ich bin eingabebereit.
```

Speichern Sie die Datei und verlassen Sie den Editor.

Nimmt man in der Datei `.profile` Änderungen vor, die sofort für die aktuelle Arbeitssitzung gelten sollen, kann man die Kommandos in der Datei `.profile` direkt mit dem Shell-Kommando `.` (Punkt) ausführen lassen. Die Shell initialisiert die Umgebung dann neu, indem sie die Kommandos in der Datei `.profile` liest und ausführt. Geben Sie folgendes ein:

```
./profile <CR>
```

Das System sollte nun folgendes ausgeben:

```
Guten Morgen! Ich bin eingabebereit.  
$
```

Definition der Terminalcharakteristika

Mit dem Kommando `stty` kann man die Arbeit mit dem Terminal vereinfachen. Zum Kommando `stty` gibt es drei Optionen: `-tabs`, `erase <^h>` und `echoe`.

<code>stty -tabs</code>	Mit dieser Option bleiben Tabulatoren bei der Ausgabe erhalten. Ein Tabulator wird auf acht Leerzeichen erweitert; dies ist der Standardwert für die Tabulatoren. Die Leerzeichen, die für einen Tabulator eingesetzt werden, können einzeln geändert werden. Nähere Angaben dazu sind unter <code>stty(1)</code> im <i>User's Reference Manual</i> zu finden.
<code>stty erase <^h></code>	Diese Option ermöglicht es, die Löschtaste auf der Tastatur anstellen des Standardzeichens <code>#</code> zum Löschen eines Zeichens einzusetzen. Normalerweise ist die Taste <code>BACKSPACE</code> die Löschtaste.
<code>stty echoe</code>	Bei einem Terminal mit einem Bildschirm werden mit dieser Option Zeichen auch vom Bildschirm gelöscht, sobald die Taste <code>BACKSPACE</code> gedrückt wird.

Sollen diese Optionen zum Kommando `stty` verwendet werden, kann man die entsprechenden Kommandozeilen in die Datei `.profile` genauso wie in ein Shell-Programm eingeben. Verwendet man dazu das Kommando `tall` (Anzeigen der letzten Zeilen einer Datei), kann man sehen, ob die vier Kommandozeilen richtig in die Datei `.profile` eingegeben wurden:

```

$ tail -4 .profile<CR>
echo Guten Morgen! Ich bin eingabebereit.
stty -tabs
stty erase <^h>
stty echoe
$

```

In Abbildung 7-34 werden das Format und die Funktionen des Kommandos `tail` zusammengefaßt.

Kommandoübersicht		
<code>tail</code> – Anzeigen des letzten Teils einer Datei		
<i>Kommando</i>	<i>Optionen</i>	<i>Argumente</i>
<code>tail</code>	<code>-n</code>	<i>Dateiname</i>
Beschreibung:	Anzeigen der letzten Zeilen einer Datei.	
Optionen:	Mit <code>-n</code> kann man die Anzahl der Zeilen angeben, die angezeigt werden sollen (Standardwert ist zehn Zeilen). Anstelle von Zeilen kann man auch eine Anzahl von Blöcken (<code>-nb</code>) oder Zeichen (<code>-nc</code>) angeben.	

Abbildung 7-34: Übersicht über das Kommando `tail`

Anlegen eines Verzeichnisses rje

In diesem Kapitel war schon mehrfach die Rede von der gemeinsamen Nutzung von Programmen mit anderen Benutzern. Ebenso können diese Benutzer über Programme oder andere Dateien verfügen, die sie für einen selbst zugänglich machen wollen. Um die Übermittlung dieser Dateien zu vereinfachen, sollte man ein Verzeichnis **rje** (remote job entry - Job-Ferneingabe) anlegen:

```
mkdir rje chmod go+w rje
```

Hier ist zu beachten, daß die Zugriffsrechte für das Verzeichnis mit **chmod** geändert werden müssen. Nachdem das Verzeichnis **rje** mit den richtigen Zugriffsrechten besteht, können andere Benutzer mit dem Kommando **uucp** Dateien in dieses senden. Nähere Angaben dazu sind unter **uucp(1)** im *User's Reference Manual* enthalten.

Verwendung von Shell-Variablen

Verschiedene der für die Shell reservierten Variablen werden in der Datei **.profile** verwendet. Mit folgendem Kommando kann man sich den aktuellen Wert jeder Shell-Variablen anzeigen lassen:

```
echo $Variablenname
```

Vier der wichtigsten Variablen werden im folgenden erläutert.

HOME

Mit dieser Variablen wird der Pfadname des eigenen Benutzerverzeichnisses definiert. Wechseln Sie mit dem Kommando **cd** in das eigene Benutzerverzeichnis und geben Sie folgendes ein:

```
pwd <CR>
```

Achten Sie auf die Antwort des Systems; geben Sie dann folgendes ein:

```
echo $HOME <CR>
```

Ist die Antwort des Systems hier dieselbe wie auf das Kommando **pwd**?

\$HOME ist das Standardargument zum Kommando **cd**. Gibt man also kein Verzeichnis ausdrücklich an, wechselt man durch Eingabe von **cd** in das Home-Verzeichnis **\$HOME**.

PATH

Mit dieser Variablen wird der Suchpfad für auszuführende Programme definiert. Die aktuellen Werte der Variablen **PATH** kann man sich mit folgendem Kommando anzeigen lassen:

```
echo $PATH<CR>
```

Das System gibt daraufhin den aktuellen Wert der Variablen **PATH** aus.

```
$ echo $PATH<CR>
:/meiname/bin:/bin:/usr/bin:/usr/lib
$
```

Der Doppelpunkt (:) wird als Begrenzungszeichen zwischen den einzelnen Pfadnamen in der Zeichenkette verwendet, die der Variablen **\$PATH** zugewiesen wird. Steht vor einem : kein anderer Wert, gilt das aktuelle Verzeichnis. Im letzten Beispiel sucht das System nach Kommandos zuerst im aktuellen Verzeichnis, dann in **/meiname/bin/**, danach in **/bin**, danach in **/usr/bin** und schließlich in **/usr/lib**.

Arbeitet man zusammen mit mehreren anderen Mitarbeitern an demselben Projekt, kann man eine Gruppe **bin** definieren, d. h. ein Verzeichnis mit Shell-Programmen, die nur für die Mitglieder der Gruppe vorgesehen sind. Als Pfad kann man beispielsweise **/projekt1/bin** definieren. Im folgenden Beispiel wird in der Datei **.profile** das Verzeichnis **:/projekt1/bin** an das Ende der Variablen **PATH** gesetzt:

```
PATH=":/meiname/bin:/bin:/usr/lib:/projekt1/bin"<CR>
```

TERM

Mit dieser Variablen teilt man der Shell mit, welchen Terminaltyp man verwendet. Man weist ihr einen Wert zu, indem man die folgenden Kommandos in der angegebenen Reihenfolge ausführt:

```
TERM=Terminalname <CR>
export TERM <CR>
```

Die ersten beiden Zeilen geben dem Rechner an, welchen Terminaltyp man verwendet.

Will man die Variable **TERM** nicht jedesmal beim Anmelden eingeben, nimmt man diese beiden Kommandozeilen in die eigene Datei **.profile** auf; dann werden sie automatisch bei jedem Anmelden ausgeführt. Der Wert, der der Variablen **TERM** zuzuweisen ist, kann in Anhang F, "Konfiguration des Terminals", nachgeschlagen werden. In diesem Anhang sind auch Informationen zum Kommando **tput** enthalten.

Verwendet man häufiger mehrere Terminaltypen, wird es auch günstig sein, das Kommando **set.term** in die Datei **.profile** aufzunehmen.

PS1

Mit dieser Variablen wird das primäre Bereit-Zeichen der Shell definiert (Standardwert ist das Dollarzeichen \$). Dieses kann man ändern, indem man die Variable **PS1** in der Datei **.profile** ändert.

Dies kann man an dem folgenden Beispiel üben. Dabei ist zu beachten, daß ein Bereit-Zeichen, das aus mehreren Wörtern bestehen soll, in Anführungszeichen zu setzen ist. Geben Sie folgende Variablenzuweisung in die Datei **.profile** ein:

```
PS1="Dein Wunsch ist mein Befehl <CR>"
```

Führen Sie nun die Datei **.profile** (mit dem Kommando **.**) aus. Achten Sie auf das neue Bereit-Zeichen.

```
$ . .profile <CR>
Dein Wunsch ist mein Befehl
```

Nun taucht das allgemeine Bereit-Zeichen \$ nicht mehr auf, bis man die Variable **PS1** aus der Datei **.profile** wieder löscht.

Übungen zur Shell-Programmierung

- 2-1. Erstellen Sie ein Shell-Programm mit dem Namen `zeit` mit folgender Kommandozeile:

```
banner `date | cut -c12-19` <CR>
```

- 2-2. Erstellen sie ein Shell-Programm, das nur das Datum in Plakatschrift ausgibt. Das Programm darf nicht denselben Namen wie ein UNIX-Systemkommando erhalten.
- 2-3. Erstellen Sie ein Shell-Programm, mit dem eine kurze Nachricht an mehrere andere Benutzer des Systems geschickt wird.
- 2-4. Lenken Sie das Kommando `date` ohne die Zeitangabe in eine Datei um.
- 2-5. Schreiben Sie den Ausdruck "Lieber Kollege" in dieselbe Datei, die das Kommando `date` enthält, ohne das Datum dabei zu löschen (`echo`).
- 2-6. Erstellen Sie mit den obigen Übungen ein Shell-Programm, mit dem eine Info an dieselben Benutzer des Systems wie in Übung 2-3 gesendet wird. Nehmen Sie folgendes in die Info auf:

Das aktuelle Datum und die Worte "Lieber Kollege" am Anfang der Info.

Den Rumpf des Infotextes (aus einer bestehenden Datei).

Die Meldung, daß das Programm beendet ist.

- 2-7. Wie kann man Variablen mit `read` in das Programm `vs.datei` einlesen?
- 2-8. Verschieben Sie mit einer `for`-Schleife eine Liste von Dateien des aktuellen Verzeichnisses in ein anderes Verzeichnis. Wie verschiebt man alle Dateien in ein anderes Verzeichnis?
- 2-9. Wie kann man das Programm `suche` so ändern, daß es in mehreren Dateien sucht?

Tip:

```
for datei  
in $*
```

- 2-10. Definieren Sie die Optionen von **stty** für die eigene Umgebung.
- 2-11. Ändern Sie das Bereit-Zeichen in das Wort "Hallo".
- 2-12. Lassen Sie sich die Definitionen der Variablen **\$HOME**, **\$TERM** und **\$PATH** in der eigenen Umgebung anzeigen.

Lösungen zu den Übungen

Übungen zur Kommandosprache

- 1-1. Der * am Anfang eines Dateinamens bezieht sich auf alle Dateien, die auf diesen Dateinamen enden, einschließlich des betreffenden Dateinamens.

```
$ ls *t<CR>
kat 123t neu.t t
$
```

- 1-2. Das Kommando `cat [0-9]*` erzeugt folgende Ausgabe:

```
1info
100data
9
05name
```

Das Kommando `echo *` erzeugt eine Liste aller Dateien im aktuellen Verzeichnis.

- 1-3. Das ? kann an jeder Stelle eines Dateinamens eingesetzt werden.
- 1-4. Mit dem Kommando `ls [0-9]*` werden nur die Dateien aufgelistet, die mit einer Zahl beginnen.
- Mit dem Kommando `ls [a-m]*` werden nur die Dateien aufgelistet, die mit einem der Buchstaben von "a" bis "m" beginnen.
- 1-5. Wird die Zeile mit der Kommandofolge in den Hintergrundmodus versetzt, gibt das System unmittelbar darauf die Prozeßnummer (PID) des Auftrags aus.
- Nein, das Zeichen & muß am Ende der Kommandozeile stehen.

1-6. Die Kommandozeile sieht folgendermaßen aus:

```
cd; pwd > schrott; ls >> schrott; ed versuch<CR>
```

1-7. Ändern Sie die Option `-c` der Kommandozeile folgendermaßen:

```
banner `date | cut -c1-10` <CR>
```

Übungen zur Shell-Programmierung

2-1.

```
$ cat zeit<CR>
banner `date | cut -c12-19`
$
$ chmod u+x zeit<CR>
$ zeit<CR>
(Zeitanzeige in Plakatschrift 10:26)
$
```

2-2.

```
$ cat datum<CR>
banner `date | cut -c1-10`
$
```

2-3.

```
$ cat kollegen<CR>
echo Name der Datei mit der Nachricht eingeben.
read note mail jeanette marianne berthold < $note
$
```

Wurden für die Benutzernamen Parameter anstelle der Benutzernamen selbst verwendet, sieht das Programm etwa so aus:

```
$ cat kollegen<CR>
echo Name der Datei mit der Nachricht eingeben.
read note mail $* < $note
$
```

- 2-4. `date | cut -c1-10 > date1<CR>`
2-5. `echo Lieber Kollege >> date1<CR>`
2-6.

```
$ cat send.info<CR>
date | cut -c1-10 > info1
echo Lieber Kollege >> info1
cat info >> info1
echo Ein Info von M. L. Keller >> info1
mail jeanette marianne berthold < info1
$
```

- 2-7.

```
$ cat vs.date1<CR>
echo Verzeichnispfad eingeben: read pfad
echo Dateinamen eingeben, mit <^d> beenden
while
  read datei
do
  mv $datei $pfad/$datei
done echo Fertig
$
```


2-8.

```
$ cat vs.datei<CR>
echo Bitte Verzeichnispfad eingeben:
read pfad
for datei in $*
do
  mv $datei $pfad/$datei
done $
```

Mit folgender Kommandozeile werden alle Dateien im aktuellen Verzeichnis verschoben:

```
$ vs.datei *<CR>
```

2-9. Siehe Tip zur Übung 2-9.

```
$ cat suche<CR> for datei
in $*
do
  if grep $wort $datei >/dev/null
  then echo $wort steht in $datei
  else echo $wort steht NICHT in $datei
  fi
done
$
```

2-10. Nehmen Sie folgende Zeilen in die Datei `.profile` auf:

```
stty -tabs <CR>
stty erase <^h> <CR>
stty echoe <CR>
```

2-11. Nehmen Sie folgende Kommandozeilen in die Datei `.profile` auf:

```
PS1=Hallo <CR>
export PS1
```

2-12. Mit folgenden Kommandos werden die Werte der Variablen der eigenen Umgebung ausgegeben:

- `$ echo $HOME <CR>`
- `$ echo $TERM <CR>`
- `$ echo $PATH <CR>`

Kapitel 8: ANLEITUNG ZUR KOMMUNIKATION

Einführung	8-1
Austauschen von Nachrichten	8-2
mail	8-3
Senden von Nachrichten	8-3
Unzustellbare Post	8-4
Post an eine Person senden	8-6
Post an mehrere Personen gleichzeitig senden	8-7
Post an ferne Systeme senden: die Kommandos uname und uuname	8-8
Verwaltung eingegangener Post	8-12
mailx	8-16
Übersicht über mailx	8-17
Optionen in der Kommandozeile	8-19
Senden von Nachrichten: die Tilde-Escape-Sequenzen	8-20
Editieren von Nachrichten	8-22
Einfügen bestehender Texte in Nachrichten	8-24
Einlesen von Dateien in Nachrichten	8-25
Einlesen von eingegangenen Nachrichten in Antworten	8-26
Änderungen in der Kopfzeile von Nachrichten	8-27
Unterschrift	8-28
Abgesandte Nachrichten aufzeichnen	8-28

mailx verlassen	8-31
Zusammenfassung	8-31
Verwaltung eingegangener Post	8-32
Das Argument msglist	8-32
Kommandos zum Lesen und Löschen von Post	8-33
Post lesen	8-33
Postfach durchsehen	8-35
Bearbeiten anderer Postdateien	8-36
Post löschen	8-37
Kommandos zum Speichern von Post	8-38
Kommandos zum Beantworten von Post	8-39
Kommandos zum Verlassen von mailx	8-40
Zusammenfassung	8-41
Die Datei .mailrc	8-42
Senden und Empfangen von Dateien	8-47
Übertragung kleiner Dateien: das Kommando mail	8-47
Übertragung großer Dateien	8-48
Übertragung einer Datei vorbereiten	8-49
Das Kommando uucp	8-51
Kommandozeilensyntax	8-52
Beispiele zur Verwendung von uucp mit Optionen	8-54
Funktion des Kommandos uucp	8-55
Das Kommando uuto	8-59
Senden einer Datei: Option m und Kommando uustat	8-59
Empfang von mit uuto übertragenen Dateien: das Kommando uupick	8-64

Vernetzung	8-68
Anschluß eines fernen Terminals: das Kommando <code>ct</code>	8-68
Kommandozeilenformat	8-69
Beispiele	8-69
Andere UNIX-Systeme anrufen: das Kommando <code>cu</code>	8-71
Kommandozeilenformat	8-72
Beispiele	8-75
Kommandos auf fernen Systemen ausführen: das Kommando <code>uux</code>	8-77
Kommandozeilenformat	8-77
Beispiele	8-77

Einführung

Das Betriebssystem UNIX bietet eine Auswahl von Kommandos, durch die man mit anderen Benutzern des UNIX-Systems kommunizieren kann; insbesondere bieten sie die folgenden Möglichkeiten: Senden und Empfangen von Nachrichten an bzw. von anderen Benutzern an demselben oder einem anderen UNIX-System, Austauschen von Dateien und Bildung von Netzwerken mit anderen UNIX-Systemen. Eine solche Vernetzung ermöglicht den Austausch von Nachrichten und Dateien zwischen verschiedenen Rechnern und die Ausführung von Kommandos auf fernen (remote) Rechnern.

Die in diesem Kapitel eingeführten Kommandos ermöglichen es dem Benutzer, diese Funktionen zu nutzen; folgende Kommandos werden beschrieben:

Zum Nachrichtenaustausch: **mail, mailx, uname** und **uname**

Zum Dateiaustausch: **uucp, uuto, uupick** und **uustat**

Für den Netzwerkbetrieb: **ct, cu** und **uux**

Austauschen von Nachrichten

Nachrichten können entweder mit dem Kommando **mail** oder **mailx** übermittelt werden. Durch diese Kommandos wird die Nachricht in eine Datei geschrieben, deren Eigentümer der Empfänger ist. Der Empfänger erhält bei der Anmeldung, oder während er angemeldet ist, die Meldung, daß Post angekommen ist (**you have mail**). Dann kann er entweder das Kommando **mail** oder **mailx** aufrufen, um die Nachricht zu lesen und gegebenenfalls darauf zu antworten.

Der Hauptunterschied zwischen **mail** und **mailx** liegt darin, daß nur **mailx** folgende Funktionen bietet:

- Eine Auswahl aus Texteditoren (**ed** oder **vi**) zur Bearbeitung eingegangener und abzusendender Nachrichten.
- Eine Liste vorhandener Nachrichten, anhand derer der Benutzer entscheiden kann, welche Nachrichten er in welcher Reihenfolge bearbeitet.
- Verschiedene Optionen zum Sichern von Dateien.
- Kommandos zum Beantworten von Nachrichten und zum Senden von Kopien eingehender und ausgehender Nachrichten an andere Benutzer.

Mit **mail** oder **mailx** kann man auch kurze Dateien mit Memos, Berichten usw. übermitteln. Soll jedoch eine Datei übermittelt werden, die über eine Seite lang ist, ist eines der Kommandos zur Dateiübertragung zu verwenden: **uuto** oder **uucp**. Eine Beschreibung dieser Kommandos ist unter "Übermittlung großer Dateien" weiter unten in diesem Kapitel zu finden.

mail

In diesem Abschnitt wird das Kommando **mail** beschrieben. Dabei werden die Grundlagen der Übermittlung elektronischer Post an einen oder mehrere Benutzer gleichzeitig eingeführt. Diese Benutzer können an demselben System wie der Absender oder an einem fernen System arbeiten. Weiterhin wird beschrieben, wie man eingegangene Post abrufen und bearbeitet.

Senden von Nachrichten

Das allgemeine Format einer Kommandozeile zum Senden von Nachrichten lautet:

mail *Benutzername* <CR>

Dabei steht *Benutzername* für den Benutzernamen des Empfängers, der an einem UNIX-System arbeitet. Dieser Benutzername kann bestehen aus:

- einem Benutzernamen, wenn der Empfänger an demselben System arbeitet (zum Beispiel **bert**),
- einem Systemnamen und Benutzernamen, wenn sich der Empfänger an einem anderen UNIX-System befindet, das mit dem System des Absenders kommunizieren kann (zum Beispiel **sys2!bert**)

Zunächst soll davon ausgegangen werden, daß der Empfänger Benutzer des lokalen Systems ist. Auf die Übermittlung von Post an Benutzer ferner Systeme wird später eingegangen. Geben Sie das Kommando **mail** auf das Bereit-Zeichen des Systems ein, drücken Sie die RETURN-Taste und geben Sie den Text der Nachricht in die nächste Zeile ein. Die Länge der Nachricht ist nicht begrenzt. Nachdem man die Eingabe beendet hat, wird sie durch Eingabe eines Punktes (.) oder <^d> (CONTROL-d) am Anfang einer neuen Zeile abgesandt.

Im folgenden Beispiel wird gezeigt, wie dieser Vorgang auf dem Bildschirm dargestellt wird:

```
$ mail phillip<CR>
Mein Treffen mit der Gruppe<CR>
von Herrn Schmid morgen wurde auf<CR>
15:00 verlegt; ich kann daher<CR>
nicht kommen. Können wir uns<CR>
stattdessen morgens treffen?<CR>
.<CR>
$
```

Das Bereit-Zeichen in der letzten Zeile bedeutet, daß die Nachricht vorgemerkt wurde, d. h. sie wurde in eine Warteschlange von Nachrichten gestellt, und daß sie abgesandt werden wird.

Unzustellbare Post

Macht man einen Fehler bei der Eingabe des Benutzernamens des Empfängers, kann das Programm **mail** die Post nicht zustellen. In diesem Fall werden zwei Meldungen ausgegeben: daß die Übermittlung nicht erfolgreich war, und daß die Post zurückgeht. Dann wird die Post an den Absender zurückgeschickt, mit einer Nachricht, in der der Systemname und Benutzername des Absenders und des Empfängers mit der Angabe, weshalb die Post nicht zugestellt werden konnte, enthalten sind.

Im folgenden Beispiel wird angenommen, daß ein Benutzer mit dem Benutzernamen **kol** eine Nachricht an den Benutzer mit dem Benutzernamen **chris** an einem System mit dem Namen **marmaduk** senden will. Die Nachricht lautet Die Konferenz wurde auf 14:00 verschoben. Dabei wird jedoch der Benutzername als **cris** falsch eingegeben:

```

$ mail cris <CR>
Die Konferenz wurde auf 14:00 verschoben
.<CR>
mail: Can't send to cris
mail: Return to kol
you have mail in /usr/mail/kol
$

```

Die Post, die damit in `/usr/mail` angekommen ist, enthält nützliche Informationen für den Fall, daß man nicht weiß, weshalb das Kommando `mail` nicht ausgeführt werden konnte; außerdem kann man daraus die Nachricht für einen neuen Versuch abrufen, ohne sie nochmals eintippen zu müssen. Folgende Informationen sind darin enthalten:

```

$ mail <CR>

From kol Wed Apr 26 15:44:48 MES 1989
>From kol Wed Apr 26 17:44:48 MES 1989 forwarded by kol
***** UNDELIVERABLE MAIL sent to cris, being returned by marmaduk!kol *****
mail: ERROR # 8 'Invalid recipient' encountered on system marmaduk

Die Konferenz wurde auf 14:00 verschoben.

?

```

Nähere Angaben zur Ausgabe und zur Bearbeitung dieser Post sind unter "Verwaltung eingegangener Post" weiter unten in diesem Kapitel enthalten.

Post an eine Person senden

Auf dem folgenden Bildschirm wird eine typische Nachricht dargestellt:

```
$ mail tommy<CR>
Hallo Thomas,<CR>
Heute um 15:00 tagt der Projektausschuß.<CR>
D.F. möchte Deine Meinung hören und wissen,<CR>
wieviel Zeit für die Fertigstellung des<CR>
Projektes noch benötigt wird.<CR>
B.K.<CR>
.<CR>
$
```

Sobald sich Thomas an seinem Terminal anmeldet, bzw. bereits während er angemeldet ist, erhält er eine Meldung, daß Post für ihn angekommen ist:

```
$ you have mail
```

Unter "Verwaltung eingegangener Post" weiter unten in diesem Kapitel wird beschrieben, wie man eingegangene Post einsehen kann.

Man kann das Kommando **mail** sehr gut üben, indem man Post an sich selbst schickt. Dazu ist das Kommando **mail** mit dem eigenen Benutzernamen einzugeben; dann kann man eine kurze Nachricht an sich selbst eingeben. Nachdem man zum Abschluß den Punkt oder `<^d>` eingegeben hat, wird die Post in eine Datei geschickt, die nach dem eigenen Benutzernamen benannt ist und sich im Verzeichnis `/usr/mail` befindet; außerdem wird eine Meldung ausgegeben, daß Post angekommen ist.

Mit Post an sich selbst kann man sich auch sehr gut an bestimmte Dinge erinnern lassen. Wenn man (als Benutzer **bert**) jemand anders am nächsten Morgen anrufen möchte, kann man sich selbst mit elektronischer Post daran erinnern lassen:

```
$ mail bert <CR>  
Buchhaltung anrufen, weshalb <CR>  
meine Zahlen von 1985 noch nicht bearbeitet sind! <CR>  
<CR>  
$
```

Meldet man sich dann am nächsten Tag an, erscheint die Meldung auf dem Bildschirm, daß Post angekommen ist.

Post an mehrere Personen gleichzeitig senden

Man kann dieselbe Nachricht an mehrere Personen schicken, indem man ihre Benutzernamen in der Kommandozeile mit **mail** angibt, wie im folgenden Beispiel:

```
$ mail tommy jeanette werner david <CR>  
Handballmannschaft! <CR>  
Heute abend um 20:30 im Bistro. <CR>  
Terminkalender nicht vergessen! <CR>  
Euer Trainer <CR>  
<CR>  
$
```

In Abbildung 8-1 werden die Syntax und die Funktionen des Kommandos **mail** zusammengefaßt.

Kommandoübersicht		
mail – Senden einer Nachricht an einen anderen Benutzer		
<i>Kommando</i>	<i>Optionen</i>	<i>Argumente</i>
mail	keine	<i>[Systemname!]Benutzername</i>
Beschreibung:	Durch Eingabe von mail mit einem oder mehreren Benutzernamen wird die nach der Kommandozeile eingegebene Nachricht an die angegebenen Benutzer(namen) übermittelt.	
Bemerkungen:	Nach Eingabe eines Punktes oder <code><^d></code> (gefolgt von RETURN-Taste) am Anfang einer neuen Zeile wird die Nachricht abgesandt.	

Abbildung 8-1: Übersicht über das Senden von Nachrichten mit dem Kommando **mail**

Post an ferne Systeme senden: die Kommandos **uname** und **uname**

Bis hierher wurde davon ausgegangen, daß die Nachrichten an Benutzer des lokalen UNIX-System geschickt werden. Man kann jedoch auch innerhalb einer Firma Post versenden, wenn zum Beispiel drei verschiedene Computersysteme in unterschiedlichen Teilen eines Gebäudes oder an verschiedenen Standorten stehen.

Man kann Post an Benutzer anderer Systeme schicken, indem man einfach den Namen des Systems des Empfängers vor seinem Benutzernamen in der Kommandozeile eingibt.

```
mail sys2!bert<CR>
```

Der Name des Systems und der Benutzername des Empfängers sind durch Ausrufezeichen zu trennen.

Bevor man dieses Kommando ausführen kann, benötigt man jedoch drei Informationen:

- den Namen des fernen Systems
- ob das eigene und das ferne System eine Verbindung haben
- den Benutzernamen des Empfängers.

Mit den Kommandos **uname** und **uuname** kann man diese Informationen herausfinden.

Den Namen des fernen Systems und den Benutzernamen des Empfängers erfragt man am besten vom Empfänger. Weiß dieser den Systemnamen nicht, braucht er auf seinem System nur folgendes Kommando eingeben:

```
uname -n <CR>
```

Daraufhin wird der Name des Systems angezeigt, wie im folgenden Beispiel:

```
$ uname -n <CR>
jumbo
$
```

Kennt man den Namen des fernen Systems, kann man mit dem Kommando **uuname** in Erfahrung bringen, ob das eigene System mit diesem fernen System kommunizieren kann. Beim Bereit-Zeichen

```
uuname <CR>
```

eingeben. Daraufhin wird eine Liste der fernen Systeme ausgegeben, mit denen das eigene System kommunizieren kann. Ist das System des Empfängers in dieser Liste enthalten, kann man an ihn mit dem Kommando **mail** Post übermitteln.

Diese Suche kann man noch vereinfachen, indem man mit dem Kommando **grep** die Ausgabe des Kommandos **uuname** nach dem Namen des fernen Systems durchsuchen läßt. Neben dem Bereit-Zeichen

```
uuname |grep system <CR>
```

eingeben. Hier steht *system* für den Systemnamen des Empfängers. Findet **grep** den angegebenen Systemnamen, wird er auf dem Bildschirm ausgegeben, wie im folgenden Beispiel:

```
$ uuname |grep jumbo<CR>
jumbo
$
```

Das bedeutet, daß die Kommunikation zwischen **jumbo** und dem eigenen System möglich ist. Kann **jumbo** mit dem eigenen System nicht in Verbindung treten, wird nach dem Kommando **uuname** nur das Bereit-Zeichen ausgegeben:

```
$ uuname |grep jumbo<CR>
$
```

Die Kommandos **uname** und **uuname** werden hier an einem Beispiel nochmals erläutert. Dabei soll eine Nachricht an den Empfänger mit dem Benutzer-namen **sabine** auf das ferne System "jumbo" geschickt werden. Zunächst ist zu prüfen, ob "jumbo" und das eigene System kommunizieren können, dann ist die Nachricht abzuschicken. Auf dem folgenden Bildschirm werden diese beiden Schritte dargestellt:

```
$ uuname |grep jumbo<CR>
jumbo
$ mail jumbo@sabine<CR>
Hallo Sabine,<CR>
Hier die endgültigen Zahlen für das Seminar:<CR>
<CR>
Unsere Abteilung - 18<CR>
Eure Abteilung - 20<CR>
<CR>
Thomas<CR>
.<CR>
$
```

In den Abbildungen 8-2 und 8-3 werden die Syntax und die Funktionen der Kommandos **uname** bzw. **uuname** zusammengefaßt.

Kommandoübersicht		
uname – Anzeigen des Systemnamens		
<i>Kommando</i>	<i>Optionen</i>	<i>Argumente</i>
uname	-n u.a.*	keine
Beschreibung:	Mit uname -n wird der Name des Systems angezeigt, auf dem man als Benutzer angemeldet ist.	

Abbildung 8-2: Übersicht über das Kommando **uname**

* Eine Liste aller verfügbaren Optionen mit einer Erläuterung ihrer jeweiligen Funktionen ist unter **uname(1)** im *User's Reference Manual* enthalten.

Kommandoübersicht		
uuname – Anzeigen einer Liste der erreichbaren fernen Systeme		
<i>Kommando</i>	<i>Optionen</i>	<i>Argumente</i>
uuname	keine	keine
Beschreibung:	Mit uuname wird eine Liste der fernen Systeme ausgegeben, die mit dem eigenen System kommunizieren können.	

Abbildung 8-3: Übersicht über das Kommando **uuname**

Verwaltung eingegangener Post

Wie oben bereits erwähnt, kann man mit dem Kommando **mail** auch die Nachrichten auf dem Bildschirm anzeigen lassen, die von anderen Benutzern geschickt wurden. Ist man in der Zeit angemeldet, zu der jemand Post schickt, erscheint folgende Meldung auf dem Bildschirm:

```
you have mail
```

Das bedeutet, daß eine oder mehrere Nachrichten in der Datei mit dem Namen `/usr/mail/Benutzername` vorliegen; diese Datei wird häufig auch als das Postfach (mailbox) des Benutzers bezeichnet. Sollen diese Nachrichten auf dem Bildschirm angezeigt werden, ist einfach das Kommando **mail** ohne sonstige Argumente einzugeben:

```
mail <CR>
```

Die Nachrichten werden dann nacheinander einzeln angezeigt; dabei wird die zuletzt eingegangene Nachricht zuerst ausgegeben. Auf dem folgenden Bildschirm wird eine typische Ausgabe des Kommandos **mail** dargestellt:

```
$ mail
From tommy Wed May 21 15:33 MES 1989
Bert,
Die Konferenz wurde anscheinend abgesagt.
Brauchst Du noch das Material für die technische Prüfung?
Thomas
?
```

Die erste Zeile, d. h. die Kopfzeile, enthält Informationen über die Nachricht: den Benutzernamen des Absenders sowie Datum und Uhrzeit der Absendung. Die Nachricht selbst steht zwischen der Kopfzeile und der Zeile mit dem Fragezeichen (?).

Wird eine längere Nachricht auf dem Terminal ausgegeben, kann man die Ausgabe durch Eingeben von `<^s>` (CONTROL-s) anhalten. Hat man den betreffenden Abschnitt bzw. Bildschirmausschnitt gelesen, kann man die Ausgabe mit `<^q>` fortsetzen lassen.

Nach jeder Nachricht wird ein Fragezeichen als Bereit-Zeichen ausgegeben; das System wartet dann auf eine Eingabe. An diesem Punkt stehen viele Optionen zur Auswahl; man kann beispielsweise die aktuelle Meldung im Postfach lesen und die nächste Meldung lesen, man kann die aktuelle Meldung löschen oder für spätere Verwendung abspeichern. Eine Liste aller Optionen, die beim Kommando `mail` zur Verfügung stehen, erhält man, indem man auf das Bereit-Zeichen `?` des Kommandos ebenfalls ein Fragezeichen `?` eingibt.

Soll die nächste Nachricht angezeigt werden, ohne die aktuelle Nachricht zu löschen, ist auf das Fragezeichen hin die RETURN-Taste zu drücken.

`? <CR>`

Damit verbleibt die aktuelle Nachricht im Postfach, und die nächste Nachricht wird angezeigt. Nachdem man alle Nachrichten im Postfach durchgelesen hat, erscheint ein Bereit-Zeichen.

Soll eine Nachricht gelöscht werden, ist auf das Fragezeichen ein `d` einzugeben:

`? d <CR>`

Damit wird die aktuelle Nachricht aus dem Postfach gelöscht. Liegt noch eine weitere Nachricht vor, wird diese angezeigt, liegt keine mehr vor, erscheint ein Bereit-Zeichen, daß besagt, daß alle Nachrichten gelesen wurden.

Soll eine Nachricht zur späteren Verwendung abgespeichert werden, ist auf das Fragezeichen ein `s` einzugeben:

`? s <CR>`

Damit wird die Nachricht standardmäßig in eine Datei mit dem Namen `mbox` im Home-Verzeichnis des Benutzers abgespeichert. Soll die Nachricht in eine andere Datei geschrieben werden, ist der Name dieser Datei nach dem Kommando `s` anzugeben.

Soll eine Nachricht beispielsweise in einer Datei mit dem Namen **postsich** (im aktuellen Verzeichnis) gespeichert werden, ist nach dem Fragezeichen folgendes einzugeben:

? s postsich <CR>

Besteht die Datei **postsich** bereits, wird die Nachricht durch das Kommando **mail** an diese angehängt. Besteht noch keine Datei unter diesem Namen, wird sie durch das Kommando **mail** angelegt, und die Nachricht wird in die Datei geschrieben. Mit dem Kommando **ls** kann man später überprüfen, ob die neue Datei vorhanden ist (mit **ls** wird der Inhalt des aktuellen Verzeichnisses aufgelistet).

Man kann die Nachricht auch in einer Datei in einem anderen Verzeichnis abspeichern, indem man einen Pfadnamen wie folgt angibt:

? s projekt1/info <CR>

Mit diesem relativen Pfadnamen wird eine Datei mit dem Namen **info** im Unterverzeichnis **projekt1** des aktuellen Verzeichnisses angegeben. Beim Speichern von elektronischer Post kann man relative oder vollständige Pfadnamen verwenden. Nähere Angaben zur Verwendung von Pfadnamen sind in Kapitel 3 enthalten.

Mit folgender Eingabe (nach dem Fragezeichen) kann man das Lesen der Nachrichten beenden:

? q <CR>

Alle noch nicht gelesenen Nachrichten verbleiben im Postfach, bis man das Kommando **mail** zum nächsten Mal aufruft.

Man kann die Ausgabe einer Nachricht abbrechen, indem man die Taste **BREAK** drückt. Damit wird die Anzeige angehalten und ein Fragezeichen erscheint als Eingabeaufforderung an den Benutzer.

In Abbildung 8-4 werden die Syntax und die Funktionen des Kommandos **mail** zum Lesen von Nachrichten zusammengefaßt.

Kommandoübersicht		
mail – Lesen von eingegangenen Nachrichten		
<i>Kommando</i>	<i>Optionen</i>	<i>Argumente</i>
mail	ja*	keine
Beschreibung:	Wird mail ohne Option angegeben, werden alle im Postfach (Systemdatei <i>/usr/mail/Benutzername</i>) eingegangenen Nachrichten angezeigt.	
Bemerkungen:	Das Fragezeichen (?) nach jeder Nachricht ist eine Eingabeaufforderung. Eine vollständige Liste der möglichen Eingaben ist im <i>User's Reference Manual</i> enthalten.	

Abbildung 8-4: Übersicht über das Lesen von Nachrichten mit dem Kommando **mail**

- * Eine Liste aller verfügbaren Optionen und eine Erläuterung ihrer Funktionen ist unter **mail(1)** im *User's Reference Manual* zu finden.

mailx

In diesem Abschnitt wird das Programm **mailx** eingeführt. Es wird erläutert, wie die Umgebung für **mailx** konfiguriert wird, wie Nachrichten mit dem Kommando **mailx** übermittelt werden, und wie man eingegangene Nachrichten damit behandelt. Die Informationen sind in vier Teile gegliedert:

- Übersicht über **mailx**
- Senden von Nachrichten
- Verwaltung eingegangener Post
- Die Datei **.mailrc**

Übersicht über mailx

Das Kommando **mailx** ist eine erweiterte Version des Kommandos **mail**. In **mailx** stehen gegenüber **mail** viele weitere Optionen zum Senden und Lesen von Post zur Verfügung. Man kann damit beispielsweise einen Aliasnamen für einen einzelnen Benutzernamen oder eine Gruppe definieren. Dadurch kann man mit **mail** elektronische Post an eine andere Person unter Verwendung eines anderen Namens oder Wortes als ihrem Benutzernamen übermitteln; außerdem kann man Post an eine ganze Gruppe von Personen unter einem einzigen Namen oder Wort senden. Verwendet man **mailx** zum Lesen eingegangener Post, kann man diese in verschiedenen Dateien speichern, sie bearbeiten, sie an Dritte weiter senden, die Nachricht beantworten usw. Mit den Umgebungsvariablen von **mailx** kann man sich eine Umgebung nach seinen eigenen Anforderungen einrichten.

Das Kommando **mailx** kann mit einem oder mehreren Benutzernamen als Argumente eingegeben werden. Nach der Eingabe fordert **mailx** zur Eingabe einer Kurzinformation zum Thema der Nachricht (Betreff) auf; danach ist die Nachricht oder ein Kommando einzugeben. Im Abschnitt "Senden von Nachrichten" werden die Funktionen zum Editieren der Nachrichten, zum Einfügen anderer Dateien, Einfügen von Namen in Verteilerlisten usw. beschrieben.

Gibt man das Kommando **mailx** ohne Argumente ein, prüft **mailx**, ob Post in der Datei mit dem Namen `/usr/mail/Benutzername` eingegangen ist. Ist Post gekommen, wird eine Liste der Nachrichten angezeigt, und man kann sie dann lesen, abspeichern, löschen oder in eine andere Datei verschieben. Im Abschnitt "Verwaltung eingegangener Post" sind einige Beispiele enthalten; außerdem werden die verfügbaren Optionen beschrieben.

Soll **mailx** angepaßt werden, legt man im Home-Verzeichnis eine Umgebungsdatei mit dem Namen `.mailrc` an. Im Abschnitt "Die Datei `.mailrc`" werden die Variablen beschrieben, die in diese Umgebungsdatei aufgenommen werden können.

mailx arbeitet in zwei Betriebsarten: Eingabemodus und Kommandomodus. Zum Erstellen und Absenden von Nachrichten muß man sich im Eingabemodus befinden. Der Kommandomodus dient zum Lesen eingegangener Post. Die Funktion von **mailx** kann folgendermaßen gesteuert werden:

- Eingabe von Optionen in der Kommandozeile (siehe **mailx(1)** im *User's Reference Manual*)

- Eingabe von Kommandos im Eingabemodus, wie zum Beispiel zum Erstellen einer Nachricht. Diesen Kommandos geht stets eine Tilde (~) voraus; sie werden Tilde-Escape-Sequenzen genannt (siehe **mailx(1)** im *User's Reference Manual*)
- Eingabe von Kommandos im Kommandomodus, um beispielsweise eingegangene Post zu lesen.
- Speichern von Kommandos und Umgebungsvariablen in einer Umgebungsdatei im Home-Verzeichnis mit dem Namen **\$HOME/.mailrc**.

Tilde-Escape-Sequenzen werden unter "Senden von Nachrichten" erläutert, Kommandos für den Kommandomodus unter "Verwaltung eingegangener Post", die Datei **.mailrc** unter "Die Datei **.mailrc**."

Optionen in der Kommandozeile

In diesem Abschnitt werden die Optionen in der Kommandozeile beschrieben.

Die Syntax für das Kommando **mailx** lautet:

```
mailx [Optionen] [Name...]
```

Die *Optionen* sind hier Signale für die Funktion des Kommandos, *Name...* steht für die adressierten Empfänger.

Alle Eingaben in die Kommandozeile, die nicht Optionen sind (diesen geht ein Bindestrich voraus) werden von **mailx** als *Name* interpretiert, d. h. als Benutzername oder Aliasname einer Person, an die eine Nachricht geschickt wird.

Auf zwei der Optionen der Kommandozeile soll besonders hingewiesen werden:

- **-f** [*Dateiname*]: Damit kann man Nachrichten aus *Dateiname* anstatt aus dem Postfach lesen.

Da mit **mailx** Nachrichten in einer beliebigen Datei gespeichert werden können, verwendet man die Option **-f** zum Lesen dieser gespeicherten Nachrichten. Standarddatei für solche Nachrichten ist **\$HOME/mbx**; mit dem Kommando

```
mailx -f
```

werden also die dort gespeicherten Nachrichten gelesen.

- **-n**: Nicht mit der Standard-systemdatei **mailx.rc** initialisieren.

Hat man eine eigene Umgebungsdatei **.mailrc** definiert (siehe "Die Datei **.mailrc**"), sucht **mailx** bei Verwendung der Option **-n** nicht in der Standard-Umgebungsdatei nach Werten, sondern direkt in der benutzerdefinierten Datei **.mailrc**. Dadurch ergibt sich eine schnellere Initialisierung, was besonders bei stark ausgelasteten Systemen ins Gewicht fällt.

Senden von Nachrichten: die Tilde-Escape-Sequenzen

Mit folgendem Kommando wird eine Nachricht an einen anderen Benutzer eines UNIX-Systems gesandt:

```
$ mailx david<CR>
```

Der angegebene Benutzername bezeichnet die Person, die die Nachricht erhalten soll. Das System geht daraufhin in den Eingabemodus und fordert zur Eingabe eines Betreffs auf. Bei starker Auslastung des Systems kann es einige Sekunden dauern, bis diese Eingabeaufforderung (Subject:) erscheint. Dies ist die einfachste Form des Kommandos **mailx**; sie unterscheidet sich kaum vom Kommando **mail**.

In den folgenden Beispielen wird gezeigt, wie man Nachrichten erstellen und editieren, bestehende Texte in die Nachrichten einfügen, die Informationen in der Kopfzeile der Nachricht ändern und weitere Aufgaben mit **mailx** ausführen kann. Den einzelnen Beispielen folgt eine Erläuterung der wichtigsten Punkte, die anhand des Beispiels veranschaulicht werden.

```
$ mailx david<CR>
Subject:
```

Ein Betreff (Subject) muß nicht unbedingt angegeben werden; in diesem Fall ist die RETURN-Taste zu drücken. Der Cursor wird dann in die nächste Zeile gesetzt, und das Programm erwartet die Eingabe der Nachricht.

```
$ mailx david<CR>
Subject: Einführungskurs<CR>
Morgen um 9:00 soll ein Einführungskurs<CR>
für mailx-Neulinge im Konferenzsaal stattfinden.<CR>
Könntest Du eine Demo machen?<CR>
Bert<CR>
~. <CR>
EOT<CR>
$
```

In diesem Beispiel sind zwei wichtige Dinge zu beachten:

- Man teilt die Nachricht in Zeilen auf, indem man am Ende einer Eingabezeile die RETURN-Taste drückt. Damit wird die Nachricht für den Empfänger leichter lesbar, und ein Überlaufen der Zeilenpuffer wird vermieden.
- Die Texteingabe wird beendet, und die Nachricht wird abgesandt, indem man eine Tilde und einen Punkt zusammen (~.) am Anfang einer neuen Zeile eingibt. Das System gibt daraufhin eine Textende-Meldung (EOT) und das Bereit-Zeichen aus.

Im Eingabemodus (wie im obigen Beispiel) stehen mehrere Kommandos zur Verfügung. Sie bestehen alle aus einer Tilde (~) und einem Buchstaben, und sie werden am Anfang einer Zeile eingegeben. Diese Zeichenketten werden als Tilde-Escape-Sequenzen bezeichnet (siehe `mailx(1)` im *User's Reference Manual*). Die meisten davon werden in den Beispielen dieses Abschnittes verwendet.

Mit der Option `-s` kann man den Betreff der Nachricht in der Kommandozeile angeben; folgende Kommandozeile:

```
$ mailx -s "Einführungskurs" david<CR>
```

ist beispielsweise gleichbedeutend mit

```
$ mailx david<CR>
Subject: Einführungskurs<CR>
```

Für den Empfänger sieht die Betreffzeile in beiden Fällen gleich aus. Dabei ist zu beachten, daß man einen Betreff mit mehreren Wörtern in Anführungszeichen setzen muß, wenn man ihn in der Kommandozeile mit angibt.

Editieren von Nachrichten

Im Eingabemodus des Programms **mailx** kann man einen Editor aufrufen, indem man die Tilde-Escape-Sequenz **~e** am Anfang einer Zeile eingibt. Im folgenden Beispiel wird gezeigt, wie die Tilde verwendet wird:

```
$ mailx david <CR>
Subject: Tilde ausprobieren <CR>
Macht man bei der Eingabe eines Textes <CR>
für eine Nachricht einen Fähler, <CR>
kann man einen Editor aufrufen, indem <CR>
man ~e (Tilde e) eingibt.
```

```
.
.
.
```

In dieser Nachricht wurde ein Wort falsch geschrieben. Zur Korrektur ruft man mit **~e** einen Editor auf, in diesem Beispiel den Standardeditor **ed**.

```

.
.
.
~e<CR>
12
/Fähler/p
für eine Nachricht einen Fähler,
s/Ähl/ehl/p
für eine Nachricht einen Fehler,
w
132
q
(continue)
Was bleibt noch zu sagen?
.
.
.

```

In diesem Beispiel wurde der Editor **ed** verwendet. In der Datei **.profile** oder **.mailrc** wird festgelegt, welcher Editor bei Eingabe eines Kommandos **~e** aufgerufen wird. Mit dem Kommando **~v** (Tilde v) kann ein alternativer Editor aufgerufen werden (meist **vi**).

Nach dem Verlassen des Editors **ed** (durch Eingabe von **q**) wird man vom Programm **mailx** wieder in den Eingabemodus gebracht und dazu aufgefordert, die Nachricht fortzusetzen (**continue**). An diesem Punkt möchte man in der Regel die korrigierte Nachricht überprüfen; dazu ist das Kommando **~p** (Tilde p) einzugeben. Mit der Tilde-Escape-Sequenz **~p** wird die ganze bis dahin eingegebene Nachricht angezeigt. Damit kann man bei der Texteingabe jederzeit den Inhalt der Nachricht überprüfen.

```
~p
Message contains:
To: david
Subject: Tilde ausprobieren
```

```
Macht man bei der Eingabe eines Textes
für eine Nachricht einen Fehler,
kann man einen Editor aufrufen, indem
man ~e (Tilde e) eingibt.
Was bleibt noch zu sagen?
(continue)
~,
EOT
$
```

Einfügen bestehender Texte in Nachrichten

mailx bietet vier Möglichkeiten, Informationen aus einer anderen Quelle in die Nachricht zu integrieren, die man erstellt:

- Einlesen einer Datei in eine Nachricht
- Einlesen einer eingegangenen Nachricht in eine Antwort
- Einfügen des Wertes einer benannten Umgebungsvariablen in eine Nachricht
- Ausführen eines Shell-Kommandos mit Einfügen der Ausgabe des Kommandos in eine Nachricht

In den folgenden Beispielen werden die ersten beiden Möglichkeiten veranschaulicht; sie werden am häufigsten verwendet. Informationen zu den beiden anderen Möglichkeiten sind unter **mailx(1)** im *User's Reference Manual* zu finden.

Einlesen von Dateien in Nachrichten

```
$ mailx david<CR>
Subject: Arbeitsplan<CR>
Wie aus dem folgenden ersichtlich:<CR>
~r briefe/date11
"briefe/date11" 10/725
haben wir unseren Arbeitsanteil herausgearbeitet.
Bitte um Kommentar.
- Bert
~.
EOT
$
```

Wie im Beispiel dargestellt, wird nach der Tilde-Escape-Sequenz `~r` der Name der Datei angegeben, die eingefügt werden soll. Das System gibt den Namen der Datei und die Anzahl der Zeilen und Zeichen der Datei aus. Der Benutzer befindet sich dann noch immer im Eingabemodus und kann die Nachricht weiterbearbeiten. Wenn der Empfänger die Nachricht erhält, ist der Text von `briefe/date11` mit in der Nachricht enthalten. Mit `~p` (Tilde p) kann man sich vor dem Absenden den Inhalt anzeigen lassen.



Einlesen von eingegangenen Nachrichten in Antworten

```
$ mailx<CR>
mailx version 2.14 2/9/85 Type ? for help.
"usr/mail/roberts": 2 messages 1 new
>N 1 abc      Tue May 1 08:09 8/155 Einführungskurs
    2 zentr   Mon Apr 30 16:57 4/127 Terminplan
? m jansen<CR>
Subject: Terminplan Zentrale<CR>
Hier ist eine Kopie des Terminplans der Zentrale...<CR>
~f 2<CR>
Interpolating: 2
(continue)
Du kannst daraus entnehmen, daß der Chef unseren Bezirk<CR>
am 14. und 15. besuchen wird.<CR>
- Robert
~.
EOT
?
```

In diesem Beispiel werden verschiedene wichtige Punkte dargestellt:

- Die Sequenz beginnt mit dem Kommandomodus; in diesem wird die eingegangene Post gelesen und bearbeitet. Dann wird durch Eingabe des Kommandos **m jansen**, d. h. eine Nachricht an "jansen" senden, in den Eingabemodus umgeschaltet.
- Mit **~f** wird im Eingabemodus eine der Nachrichten im Postfach eingelesen, um sie zum Bestandteil der abgehenden Nachricht zu machen. Die Zahl 2 nach **~f** bedeutet, daß Nachricht 2 eingelesen werden soll.
- **mailx** meldet, daß Nachricht 2 eingelesen wird und fordert dann dazu auf, fortzufahren.
- Nachdem die Nachricht fertig und abgesandt ist, befindet man sich wieder im Kommandomodus; dies wird an der Eingabeaufforderung **?** deutlich. Nun kann man ein anderes Kommando im Kommandomodus ausführen oder **mailx** durch Eingabe von **q** verlassen.

Ein alternativ verwendbares Kommando, `~m` (Tilde m), arbeitet ebenso wie `~f`, nur wird die eingelesene Nachricht um eine Tabulatorposition eingerückt. Beide Kommandos, `~m` und `~f`, arbeiten nur, wenn man im Kommandomodus beginnt und dann ein Kommando eingibt, das auf Eingabemodus umschaltet. Weitere derartige Kommandos werden im Abschnitt "Verwaltung eingegangener Post" behandelt.

Änderungen in der Kopfzeile von Nachrichten

Die Kopfzeile einer `mailx`-Nachricht besteht aus vier Teilen:

- Betreff
- Empfänger
- Verteilerliste
- Blindverteiler (eine Liste von Empfängern, die nicht auf den Kopien der Nachricht für die anderen Empfänger erscheint)

Nach dem Aufrufen des Kommandos `mailx` mit einem Benutzernamen oder Aliasnamen wird der Benutzer in den Eingabemodus versetzt und zur Eingabe des Betreffs der Nachricht aufgefordert. Nachdem die Betreffzeile durch Drücken der RETURN-Taste beendet wird, erwartet `mailx` den Text der Nachricht. Soll an irgendeinem Punkt während der Eingabe (im Eingabemodus) die Kopfzeile geändert oder ergänzt werden, kann man unter vier Tilde-Escape-Sequenzen auswählen: `~h`, `~t`, `~c` und `~b`.

- `~h` Anzeigen der Felder der Kopfzeile: Betreff, Empfänger, Verteilerliste und Blindverteiler, mit den aktuellen Werten. Man kann diese Werte ändern, ergänzen oder durch Drücken der RETURN-Taste übernehmen.
- `~t` Ergänzen der Liste der Empfänger; es können entweder Benutzernamen oder Aliasnamen eingegeben werden.
- `~c` Erstellen oder Ergänzen einer Verteilerliste für die Nachricht. Den Benutzernamen oder Aliasnamen derjenigen eingeben, denen die Nachricht zugestellt werden soll.
- `~b` Erstellen oder Ergänzen eines Blindverteilers für die Nachricht.

Alle Tilde-Escape-Sequenzen müssen in einer Zeile ganz vorne stehen. Bei den Sequenzen `~t`, `~c` oder `~b` werden alle weiteren Eingaben in dieselbe Zeile als Eingabe für die betreffende Liste behandelt. Weitere Eingaben in einer Zeile, die mit `~h` beginnt, werden ignoriert.

Unterschrift

Bei Bedarf kann man mit den Umgebungsvariablen `sign` und `Sign` zwei verschiedene Unterschriften definieren. Diese können in der Nachricht dann mit `~a` (Tilde a) bzw. `~A` (Tilde A) aufgerufen werden. Im folgenden Beispiel wird der Wert "Chef" durch das Kommando `~A` aufgerufen:

```
$ mailx -s befehle alle <CR>
Um 04:00 Uhr beginnt die Konferenz. <CR>
~A <CR>
Chef
~. <CR>
EOT
$
```

Mit den beiden Unterschriftssequenzen (`~a` und `~A`) kann man zwei verschiedene Formen der Unterschrift definieren. Da jedoch der Benutzername des Absenders automatisch in der Nachrichtenkopfzeile erscheint, wenn die Nachricht gelesen wird, wird zur Identifikation des Absenders keine Unterschrift benötigt.

Abgesandte Nachrichten aufzeichnen

Das Kommando `mailx` bietet verschiedene Möglichkeiten, Kopien abgesandter Nachrichten zu speichern. Zwei dieser Möglichkeiten, bei denen keine speziellen Umgebungsvariablen definiert werden müssen, sind die Verwendung von `~w` (Tilde w) und der Option `-F` in der Kommandozeile.

Durch Eingabe von `~w` mit einem Dateinamen wird die Nachricht in diese Datei geschrieben, wie im folgenden Beispiel:

```
$ mailx bdr<CR>
Subject: Kopien Sichern<CR>
Soll eine Kopie des Nachrichtentextes<CR>
gesichert werden, Tilde-w verwenden.<CR>
~w sichpost
"sichpost" 2/71
~.
EOT
$
```

Nun kann man den Inhalt von `sichpost` anzeigen lassen:

```
$ cat sichpost<CR>
Soll eine Kopie des Nachrichtentextes
gesichert werden, Tilde-w verwenden.
$
```

Der Nachteil dieser Methode besteht darin, daß die Kopfzeile nicht mit gesichert wird.

Mit der Option `-F` in der Kommandozeile kann man die Kopfzeile erhalten:

```
$ mailx -F -s Sichern bdr<CR>
Mit dieser Methode wird die Nachricht
in eine Datei mit dem Namen bdr in
meinem aktuellen Verzeichnis gespeichert.
~.
EOT
$
```

Nun kann man sich den Inhalt der Datei **bdr** anzeigen lassen.

```
$ cat bdr<CR>
From: kol Fri May 2 11:14:45 1986
To: bdr
Subject: Sichern

Mit dieser Methode wird die Nachricht
in eine Datei mit dem Namen bdr in
meinem aktuellen Verzeichnis gespeichert.
$
```

Mit der Option **-F** wird der Text der Nachricht an eine Datei angehängt, die nach dem ersten Empfänger angegeben wird. Wurde ein Aliasname für den/die Empfänger verwendet, wird er zuerst in den/die betreffenden Benutzername(n) umgewandelt, und der erste Benutzername wird als Dateiname verwendet. Besteht eine Datei dieses Namens im aktuellen Verzeichnis, wird der Text der Nachricht an sie angehängt.

mailx verlassen

Nachdem die Nachricht erstellt ist, kann man durch Eingabe eines der folgenden drei Kommandos **mailx** wieder verlassen:

- ~. Tilde-Punkt (~.) verwendet man in der Regel, um den Eingabemodus zu verlassen. Damit wird gleichzeitig die Nachricht abgesandt. Hat man aus dem Kommandomodus des Kommandos **mailx** in den Eingabemodus gewechselt, kehrt man nun wieder zum Kommandomodus zurück (dies erkennt man an der Eingabeaufforderung ? nach der Eingabe dieses Kommandos). Hat man im Eingabemodus begonnen, kehrt man an dieser Stelle zur Shell zurück (erkennbar am Bereit-Zeichen der Shell).
- ~q Mit Tilde-q (~q) wird eine Unterbrechung simuliert. Damit kann man den Eingabemodus von **mailx** verlassen. Hat man Text für eine Nachricht eingegeben, wird er in einer Datei mit dem Namen **dead.letter** im Home-Verzeichnis des Benutzers gespeichert.
- ~x Mit Tilde-x (~x) wird ebenfalls eine Unterbrechung simuliert; damit kann man den Eingabemodus von **mailx** verlassen, ohne etwas zu sichern.

Zusammenfassung

In den vorangegangenen Abschnitten wurden einige der Tilde-Escape-Sequenzen anhand von Beispielen beschrieben, die beim Versenden von Nachrichten mit dem Kommando **mailx** zur Verfügung stehen. Weitere Informationen sind unter **mailx(1)** im *User's Reference Manual* zu finden.



Verwaltung eingegangener Post

Im Programm **mailx** stehen über fünfzig Kommandos zur Verwaltung eingehender Post zur Verfügung. Eine alphabetische Liste dieser Kommandos mit ihren Synonymen ist unter **mailx(1)** im *User's Reference Manual* enthalten. Die am häufigsten verwendeten Kommandos und Argumente werden in den folgenden Unterabschnitten beschrieben:

- das Argument *msglist*
- Kommandos zum Lesen und Löschen von Post
- Kommandos zum Speichern von Post
- Kommandos zum Beantworten von Post
- Kommandos zum Verlassen des Programms **mailx**

Das Argument *msglist*

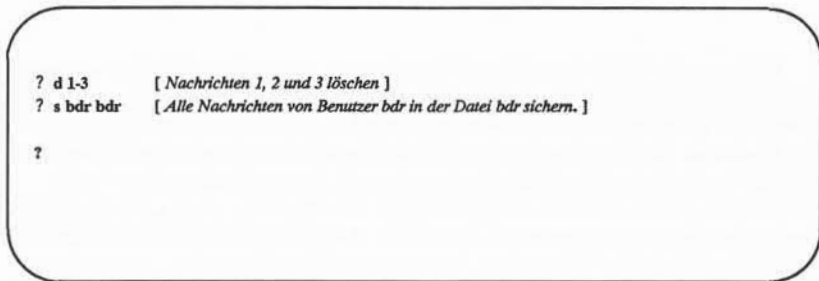
Bei vielen Kommandos von **mailx** kann eine Form des *msglist*-Arguments angegeben werden. Mit diesem Argument wird eine Liste von Nachrichten an ein Kommando übergeben, auf die es angewendet werden soll. Ist ein *msglist*-Argument für ein Kommando erforderlich, und wird keines angegeben, wird das Kommando auf die aktuelle Nachricht angewendet. Für ein *msglist*-Argument kann eines der folgenden Formate verwendet werden:

<i>n</i>	Nachricht Nummer <i>n</i> als aktuelle Nachricht
^	die erste wiederhergestellte Nachricht
\$	die letzte Nachricht
*	alle Nachrichten
<i>n-m</i>	eine Reihe von Nachrichtennummern (einschließlich)
<i>Benutzer</i>	alle Nachrichten von <i>Benutzer</i>
<i>/Zeichenkette</i>	Alle Nachrichten mit dieser <i>Zeichenkette</i> in der Betreffzeile (unabhängig von Groß- oder Kleinschreibung)
<i>:c</i>	alle Nachrichten des Typs <i>c</i> ; dabei steht <i>c</i> für: d - gelöschte Nachrichten n - neue Nachrichten

- o - alte Nachrichten
- r - gelesene Nachrichten
- u - nicht gelesene Nachrichten

Aus dem Kontext des Kommandos ergibt sich, ob die betreffende Angabe sinnvoll ist.

Auf dem folgenden Bildschirm werden zwei Beispiele dargestellt (? ist die Eingabeaufforderung im Kommandomodus):



In den drei folgenden Unterabschnitten finden sich weitere Beispiele.

Kommandos zum Lesen und Löschen von Post

Wenn eine Nachricht im Postfach ankommt, erscheint die folgende Meldung auf dem Bildschirm des Benutzers:

```
you have mail
```

Diese Meldung erscheint bei der Anmeldung, oder wenn man aus einem anderen Programm zur Shell zurückkehrt.

Post lesen

Mit dem Kommando **mailx** mit oder ohne Argumente kann man die Post lesen. Ruft man das Kommando auf, wird man in den Kommandomodus von **mailx** gebracht. Dann erscheint auf dem Bildschirm eine Anzeige im folgenden Format:

```
mailx version 2.14 10/19/86  Type ? for help
"/usr/mail/bdr":  3 messages 3 new
> N 1 rbt          Thur Apr 30 14:20  8/190 Konferenz
  N 2 verwalt     Thur Apr 30 15:56  5/84  Neuer Drucker
  N 3 david       Fri May 1  08:39  64/1574 Reorganisation
?
```

In der ersten Zeile wird die auf dem System verwendete Version von **mailx** und das Datum angegeben, und es wird darauf hingewiesen, daß man durch Eingabe eines Fragezeichens (?) Hilfetexte aufrufen kann. In der zweiten Zeile wird der Pfadname der Datei angezeigt, die als Eingabe für die Anzeige verwendet wird (normalerweise identisch mit dem eigenen Benutzernamen) sowie die Gesamtzahl der Nachrichten und ihr Status. Der Rest der Anzeige besteht aus Kopfzeilen der eingegangenen Nachrichten. Die Nachrichten sind fortlaufend numeriert; dabei steht die zuletzt eingegangene Nachricht unten am Ende der Liste. Links von den Nummern kann ein Statusindikator stehen: N für "neu", U für "ungelesen". Ein Zeichen > (größer als) zeigt auf die aktuelle Nachricht. Weitere Felder in der Kopfzeile enthalten den Benutzernamen des Absenders, den Wochentag, das Datum und die Uhrzeit der Zustellung sowie die Anzahl der Zeilen und Zeichen der Nachricht und den Betreff. Dieses letzte Feld kann leer sein.

Nachdem die Kopfzeilen auf dem Bildschirm angezeigt werden, kann man Nachrichten anzeigen lassen, indem man die RETURN-Taste drückt oder ein Kommando, gefolgt von einem *msglist*-Argument, eingibt. Gibt man ein Kommando ohne *msglist*-Argument an, wird das Kommando auf die Nachricht angewendet, auf die das Zeichen > zeigt. Die RETURN-Taste hat hier dieselbe Funktion wie die Eingabe des Kommandos **p** ohne *msglist*-Argument; es wird die Nachricht angezeigt, auf die das Zeichen > zeigt. Soll eine andere Nachricht (oder mehrere nacheinander) angezeigt werden, ist **p** oder **t** mit der/den Nachrichtennummer(n) einzugeben, wie in den folgenden Beispielen dargestellt:

? <CR>	[Aktuelle Nachricht anzeigen.]
? p 2 <CR>	[Nachricht Nummer 2 anzeigen.]
? p david <CR>	[Alle Nachrichten von Benutzer david anzeigen.]

Das Kommando **t** ist ein Synonym des Kommandos **p**.

Postfach durchsehen

Mit dem Kommando **mailx** kann man die Nachrichten im Postfach durchsehen und dabei entscheiden, welche man sofort ansehen will.

Wenn man den Kommandomodus des Kommandos **mailx** aktiviert, wird angezeigt, wie viele Nachrichten angekommen sind, und die Kopfzeilen von bis zu zwanzig Nachrichten werden auf einem Bildschirm angezeigt. Bei einer Telefonverbindung mit dem Computersystem werden nur zehn Kopfzeilen angezeigt. Übersteigt die Gesamtzahl der Nachrichten eine Bildschirmlänge, kann man den nächsten Bildschirm mit dem Kommando **z** ansehen. Mit **z-** kann man gegebenenfalls den vorherigen Bildschirm anzeigen lassen. Sollen die Kopfzeilen für eine bestimmte Gruppe von Nachrichten angezeigt werden, ist das Kommando **f** ("von Absender") mit dem zutreffenden *msglist*-Argument einzugeben.

Im folgenden einige Beispiele für diese Kommandos:

? z	[Nächsten Bildschirm mit Kopfzeilen anzeigen.]
? z-	[Vorigen Bildschirm anzeigen.]
? f david	[Kopfzeilen aller Nachrichten von Benutzer david anzeigen.]

Bearbeiten anderer Postdateien

Ruft man **mailx** mit folgendem Kommando auf:

```
$ mailx<CR>
```

wird die eigene Datei `/usr/mail/Benutzername` bearbeitet.

Mit **mailx** kann man auch andere Postdateien aufrufen und die Kommandos von **mailx** auf ihren Inhalt anwenden. Auch andere Dateitypen können damit aufgerufen werden, jedoch können sie nicht mit den **mailx**-Kommandos bearbeitet werden. Versucht man dies, erscheint die Nachricht "No applicable messages". Eine andere Postdatei wird mit den Kommandos **fl** oder **fold** (Synonyme), gefolgt vom *Dateinamen*, aktiviert. Für das Argument *Dateiname* kann eines der folgenden Sonderzeichen eingegeben werden:

%	das aktuelle Postfach
%Benutzername	das Postfach des Eigentümers von <i>Benutzername</i> (sofern man über die erforderlichen Zugriffsrechte verfügt)
#	die vorherige Datei
&	die aktuelle Datei "mbox"

Auf dem folgenden Bildschirm ist ein Beispiel dafür zu sehen:

```

$ mailx<CR>

mailx version 2.14 10/19/86  Type ? for help.
"/usr/mail/david":  3 messages 2 new 3 unread
  U 1 jaf           Sat May 9 07:55   7/137   Test25
> N 2 tobias       Sat May 9 08:59   9/377   Anforderungen für UNITS
  N 3 hasler       Sat May 9 11:08  29/1214 Zugriff auf Bailey

? fl &           [ Mit diesem Kommando ins eigene Postfach übertragen. ]

Held 3 messages in /usr/mail/david
"/fs1/david/mbox": 74 messages 10 unread
.
.
.
? q<CR>
$

```

Post löschen

Eine Nachricht wird gelöscht, indem man **d** gefolgt von einem *msglist*-Argument eingibt. Läßt man das *msglist*-Argument weg, wird die aktuelle Nachricht gelöscht. Der eigentliche Löschvorgang findet erst statt, wenn man die bearbeitete Postfachdatei verläßt. Vor diesem Zeitpunkt kann man das Löschen durch Eingabe von **u** (wiederherstellen) wieder rückgängig machen. Nach dem Kommando **q** (verlassen) oder nach dem Aktivieren einer anderen Datei sind die Nachrichten jedoch endgültig gelöscht.

In **mailx** kann man die Kommandos zum Löschen und Anzeigen kombinieren: dazu ist **dp** einzugeben. Dies bedeutet soviel wie: "Gerade gelesene Nachricht löschen und die nächste anzeigen". Im folgenden einige Beispiele für das Löschkommando:

- ? d* [Alle eigenen Nachrichten löschen.]
- ? dr [Alle gelesenen Nachrichten löschen.]
- ? dp [Aktuelle Nachricht löschen und nächste anzeigen.]
- ? d2-5 [Nachrichten 2 bis 5 löschen.]

Kommandos zum Speichern von Post

Alle Nachrichten, die nicht ausdrücklich gelöscht werden, werden beim Verlassen von **mailx** gesichert. Die Nachrichten, die man gelesen hat, werden im Home-Verzeichnis in einer Datei mit dem Namen **mbox** gesichert, Nachrichten, die man noch nicht gelesen hat, verbleiben im Postfach (**/usr/mail/Benutzername**).

Das Kommando zum Speichern von Nachrichten steht in zwei Formen zur Verfügung: als **S** (Großbuchstabe) und als **s** (Kleinbuchstabe). Die Syntax für die Form in Großschreibung lautet:

S [*msglist*]

Die im Argument *msglist* angegebenen Nachrichten werden in einer Datei im aktuellen Verzeichnis gespeichert, die nach dem Namen des Absenders der ersten Nachricht der Liste benannt wird.

Die Syntax für die Form in Kleinschreibung lautet:

s [*msglist*] [*Dateiname*]

Die im Argument *msglist* angegebenen Nachrichten werden in der mit *Dateiname* angegebenen Datei gespeichert. Läßt man das Argument *msglist* weg, wird die aktuelle Nachricht gespeichert. Verwendet man hier Benutzernamen als Dateinamen, kann dies zu Verwechslungsmöglichkeiten führen. Kann **mailx** die Namen

Kommandos zum Beantworten von Post

Das Kommando zum Beantworten von Post steht in zwei Formen zur Verfügung: als großes oder kleines **r**. Der Hauptunterschied zwischen den beiden Formen liegt darin, daß mit dem Großbuchstaben (**R**) die Antwort nur an den Absender der Nachricht, mit dem Kleinbuchstaben (**r**) dagegen nicht nur an den Absender, sondern auch an alle weiteren Empfänger gesandt wird. Die weiteren Unterschiede zwischen den beiden Formen sind unter **mailx(1)** im *User's Reference Manual* zu finden.

Beim Beantworten einer Nachricht wird die ursprüngliche Betreffzeile wieder aufgegriffen und als Betreffzeile der Antwort verwendet. Dazu ein Beispiel:

```
$ mailx<CR>
```

```
mailx version 2.14 10/19/83 Type ? for help.
```

```
"usr/mail/david": 3 messages 2 new 3 unread
```

```
U 1 jaf      Wed May 9 07:55  7/137  test25
> N 2 tobias Wed May 9 08:59  9/377  Anforderungen für UNITS
N 3 hasler   Wed May 9 11:08  29/1214 Zugriff auf Bailey
```

```
? R 2
```

```
To: tobias
```

```
Subject: Re: Anforderungen für UNITS
```

Nimmt man an, daß die Nachricht über Anforderungen für UNITS auch an weitere Personen gesandt wurde, und daß für die Antwort **r** verwendet wurde, erscheint die Kopfzeile etwa so:

```
? r2
To: tobias eg hasler jcb bdr
Subject: Re: Anforderungen für UNITS
```

Kommandos zum Verlassen von mailx

Zum Verlassen des Programms **mailx** gibt es zwei Kommandos: **q** oder **x**. Verläßt man **mailx** durch Eingabe von **q**, werden Meldungen angezeigt, mit denen zusammengefaßt wird, wie man die Post bearbeitet hat. Sie sehen folgendermaßen aus:

```
? q<CR>
Saved 1 message in /fs1/bdr/mbx
Held 1 message in /usr/mail/bdr
$
```

Aus dieser Anzeige wird ersichtlich, daß der Benutzer **bdr** mindestens zwei Nachrichten erhalten, davon eine gelesen und die andere nicht gelesen oder das Kommando eingegeben hat, daß sie in **/usr/mail/bdr** verbleiben soll. Waren es mehr als zwei Nachrichten, wurden die anderen gelöscht oder in anderen Dateien abgespeichert. Über diese gibt **mailx** keine Meldungen aus.

Verläßt man **mailx** durch Eingabe von **x**, bleibt praktisch der ursprüngliche Zustand erhalten. Die gelesenen und die gelöschten Nachrichten bleiben im Postfach erhalten. Nachrichten, die in anderen Dateien abgespeichert wurden, werden jedoch nicht wieder in ihren ursprünglichen Zustand zurückversetzt.

Zusammenfassung

In den vorangegangenen Unterabschnitten wurden einige der am häufigsten verwendeten Kommandos von **mailx** beschrieben. Eine vollständige Liste der Kommandos ist unter **mailx(1)** im *User's Reference Manual* enthalten. Benötigt man Hilfeinformationen während der Arbeit im Kommandomodus von **mailx** ist entweder ein **?** oder das Kommando **help** neben der Eingabeaufforderung **?** einzugeben. Dann wird eine Liste der Kommandos von **mailx** mit Hinweisen zu ihrer Funktion auf dem Terminal angezeigt.

Die Datei .mailrc

Die Datei **.mailrc** enthält Kommandos, die beim Aufrufen von **mailx** ausgeführt werden sollen.

Es kann eine systemübergreifende Umgebungsdatei (**/usr/lib/mailx/mailx.rc**) im System definiert sein. In einer solchen Datei werden allgemeine Variablen vom Systemverwalter definiert. Variablen, die in der Benutzerdatei **.mailrc** definiert sind, haben in seiner Umgebung Vorrang vor den in **mailx.rc** definierten Variablen.

Die meisten Kommandos von **mailx** können in der Datei **.mailrc** verwendet werden; folgende Kommandos sind jedoch nicht zulässig:

! (oder) shell	Auf Shell-Ebene zurückgehen
Copy	Mit <i>msglist</i> angegebene Nachrichten in einer Datei speichern, deren Name vom Namen des Absenders abgeleitet wird
edit	Editor aufrufen
visual	vi aufrufen
followup	Eine Nachricht beantworten
Followup	Eine Nachricht beantworten und eine Kopie an die mit <i>msglist</i> angegebenen Empfänger senden
mail	Umschalten auf Eingabemodus
reply	Eine Nachricht beantworten
Reply	Allen Absendern der in <i>msglist</i> angegebenen Nachrichten antworten

Eine Datei **.mailrc** kann man mit einem beliebigen Editor erstellen oder von einem anderen Benutzer kopieren. In Abbildung 8-5 wird ein Beispiel einer Datei **.mailrc** gezeigt:


```

if r
    cd $HOME/mail
endif
set allnet append asksub askcc autoprint dot
set metoo quiet save showto header hold keep keepsave
set outfolder
set folder='mail'
set record='outbox'
set crt=24
set EDITOR='/bin/ed'
set sign='Rohleder'
set Sign='Jakob Rohleder, Supervisor'
set toplines=10
alias franz    fjs
alias bert     rcm
alias alice    ap
alias markus   mct
alias doris    dr
alias peter    pat
group rohledergp    franz bert alice peter markus
group buchhalt    rohledergp doris

```

Abbildung 8-5: Beispiel einer Datei `.mailrc`

In dem Beispiel von Abbildung 8-5 sind die Kommandos enthalten, die sehr häufig verwendet werden: das Kommando `set` und die Kommandos `alias` oder `group`.

Mit dem Kommando `set` werden Umgebungsvariablen Werte zugewiesen. Die Kommandosyntax lautet:

```

set
set Name
set Name = Zeichenkette
set Name = Zahl

```

Gibt man das Kommando `set` ohne Argumente ein, wird eine Liste aller definierten Variablen mit ihren Werten erstellt. Das Argument *Name* bezieht sich auf eine Umgebungsvariable. Bei einem Kommando `set` können mehrere *Namen* angegeben werden. Bei einigen Variablen ist eine Zeichenkette oder ein numerischer Wert einzugeben. Zeichenketten werden dabei in einfache Anführungszeichen gesetzt.

Weist man einer Umgebungsvariablen einen Wert zu, wie beispielsweise durch `HOME=meinname`, teilt man der Shell mit, wie sie diese Variable interpretieren soll. Diese Form der Zuweisung in der Shell bedeutet jedoch nicht, daß der Wert dieser Variablen auch anderen UNIX-Systemprogrammen zur Verfügung steht, die mit Zugriffen auf Variablen arbeiten. Um dies zu erreichen, muß die Variable exportiert werden. In Kapitel 6 bzw. 7 wurde die Variable `TERM` definiert; nach diesem Beispiel wird das Kommando `export` verwendet:

```
$ TERM=dap4x
$ export TERM
```

Exportiert man Variablen aus der Shell in dieser Form, spricht man bei Programmen, die auf diese Umgebungsvariablen zugreifen, davon, daß sie die Variablen importieren. Einige dieser Variablen (wie zum Beispiel `EDITOR` und `VISUAL`) gelten nicht nur für `mailx`, sondern können als allgemeine Umgebungsvariablen aus der Benutzerumgebung importiert werden. Wird für eine importierte Variable ein Wert in der Datei `.mailrc` definiert, hat dieser Vorrang vor dem importierten Wert. Mit dem Kommando `unset` kann man eine Zuweisung aufheben, das Kommando kann jedoch nur auf Variablen angewendet werden, die in der Datei `.mailrc` definiert wurden; er hat auf importierte Variablen keinen Einfluß.

In einer Datei `.mailrc` können 41 Variablen definiert werden; bei dieser hohen Zahl können in diesem Handbuch nicht alle beschrieben werden. Ausführliche Informationen dazu sind unter `mailx(1)` im *User's Reference Manual* zu finden.

Drei der im Beispiel von Abbildung 8-5 enthaltenen Variablen sollen hier näher betrachtet werden, da man an ihnen das Ablegen von Nachrichten darstellen kann: `folder`, `record` und `outfolder`. Die drei Variablen hängen miteinander zusammen; über sie wird bestimmt, in welchen Verzeichnissen und Dateien Kopien der Nachrichten abgelegt werden.

Der Variablen **folder** wird mit folgendem Format ein Wert zugewiesen:

```
set folder=Verzeichnis
```

Damit wird das Verzeichnis angegeben, in dem normale Postdateien abgespeichert werden sollen. Beginnt der Verzeichnisname nicht mit einem Schrägstrich (/), wird angenommen, daß es ein Unterverzeichnis von **\$HOME** ist. Ist **folder** eine exportierte Shell-Variable, kann man (in Kommandos, für die ein *Dateiname* als Argument erforderlich ist) Dateinamen mit einem Schrägstrich (/) vor dem Namen angeben. Der Name wird dann so erweitert, daß die Datei in das Verzeichnis **folder** geschrieben wird.

Der Variablen **record** wird mit folgendem Format ein Wert zugewiesen:

```
set record=Dateiname
```

Damit wird **mailx** angewiesen, eine Kopie aller abgesandten Nachrichten in der angegebenen Datei zu speichern. Dabei werden die Kopfzeilen mit dem Text der Nachrichten gespeichert. Standardmäßig ist diese Variable nicht definiert.

Mit der Variablen **outfolder** wird festgelegt, daß die Datei, in der Kopien abgesandter Nachrichten gespeichert werden (die über die Variable **record=** definiert wird), im Verzeichnis **folder** untergebracht wird. Sie wird mit einem Kommando **set** definiert. Standardwert ist **nooutfolder**.

Die Kommandos **alias** und **group** sind Synonyme. In Abbildung 8-5 wird das Kommando **alias** verwendet, um einem Aliasnamen einen einzelnen Benutzernamen zuzuordnen, das Kommando **group** dagegen, um mehrere Namen anzugeben, die mit einem einzigen gemeinsamen Namen adressiert werden sollen. So kann man einfach zwischen Aliasnamen für Einzelpersonen und solchen für Gruppen unterscheiden; die Kommandos sind jedoch austauschbar. Aliasnamen können auch verschachtelt werden.

In der Datei **.mailrc** in Abbildung 8-5 steht der Aliasname **rohledergrp** für fünf Benutzer; davon sind drei durch zuvor angegebene Aliasnamen definiert, und einer ist durch seinen Benutzernamen definiert. Der fünfte Benutzer, **peter**, wird sowohl über einen Benutzernamen als auch über einen Aliasnamen definiert. Im nächsten Kommando **group** der Datei, d. h. für die Gruppe **buchhalt**, werden die beiden Aliasnamen **rohledergrp** und **doris** verwendet; sie stehen für insgesamt zwölf Benutzernamen.

In der Datei `.mailrc` in Abbildung 8-5 ist ein Kommando `if-endif` enthalten. Die vollständige Syntax für dieses Kommando lautet:

```
if s |r mail-Kommandos
    else mail-Kommandos
endif
```

Dabei stehen `s` und `r` für Senden bzw. Empfangen; damit kann man bewirken, daß bestimmte Initialisierungskommandos in Abhängigkeit davon ausgeführt oder nicht ausgeführt werden, ob `mailx` im Eingabemodus (Senden) oder im Kommandomodus (Empfangen) aktiviert wird. Im obigen Beispiel bewirkt das Kommando einen Wechsel ins Verzeichnis `$HOME/mail`, wenn Post gelesen werden soll (Empfangen). In diesem Fall hatte der Benutzer zur Bearbeitung eingehender Post ein Unterverzeichnis angelegt.

Die in diesem Abschnitt beschriebenen Umgebungsvariablen sind solche, die am häufigsten in der Datei `.mailrc` vorkommen. Sie können jedoch alle auch nur für eine Sitzung definiert werden; dazu muß man sich im Kommandomodus befinden. Eine vollständige Liste der Umgebungsvariablen, die in `mailx` definiert werden können, ist unter `mailx(1)` im *User's Reference Manual* zu finden.

Senden und Empfangen von Dateien

In diesem Abschnitt werden die Kommandos zur Übermittlung von Dateien beschrieben: das Kommando **mail** für kleine Dateien (bis zu einer Seite) und die Kommandos **uucp** und **uuto** für längere Dateien. Mit dem Kommando **mail** können Dateien lokal oder zu einem fernen System übermittelt werden, mit den Kommandos **uucp** und **uuto** werden Dateien von einem System auf ein anderes übertragen.

Übertragung kleiner Dateien: das Kommando **mail**

Damit eine Datei mit **mail** übertragen werden kann, muß die Eingabe in diese Datei in der Kommandozeile umgelenkt werden. Dazu ist das Umlenkungssymbol **<** (kleiner als) wie folgt zu verwenden:

```
mail Benutzername < Dateiname <CR>
```

Weitere Informationen zur Eingabeumlenkung sind in Kapitel 7 enthalten. Für *Benutzername* ist hier der Benutzername des Empfängers, für *Dateiname* der Name der zu übertragenden Datei einzusetzen. Soll beispielsweise eine Kopie einer Datei mit dem Namen **tagesordnung** an den Eigentümer des Benutzernamens **sabine** (auf dem eigenen System) gesandt werden, ist folgende Kommandozeile einzugeben:

```
$ mail sabine < tagesordnung <CR>
$
```

Das Bereit-Zeichen in der zweiten Zeile bedeutet, daß der Inhalt von **tagesordnung** übertragen wurde. Sobald **sabine** das Kommando **mail** eingibt, um ihre Post zu lesen, wird die Datei **tagesordnung** angezeigt.

Soll dieselbe Datei an mehrere Benutzer des eigenen Systems übermittelt werden, ist dasselbe Format in der Kommandozeile mit einer Abweichung zu verwenden: anstatt eines Benutzernamens sind mehrere Benutzernamen durch Leerzeichen getrennt einzugeben, wie im folgenden Beispiel:

```
$ mail sabine tommy ingo werner < tagesordnung <CR>
$
```

Auch hier bedeutet das Bereit-Zeichen als Antwort auf das Kommando, daß die Nachricht übermittelt wurde.

Mit einer Erweiterung dient dasselbe Format in der Kommandozeile zur Übermittlung einer Datei an einen Benutzer an einem fernen System, das mit dem eigenen System verbunden ist. In diesem Fall ist der Name des fernen Systems vor dem Benutzernamen des Empfängers anzugeben. Der Systemname und der Benutzername sind durch ein Ausrufezeichen (!) zu trennen:

```
mail System!Benutzername < Dateiname < CR >
```

Beispiel:

```
$ mail jumbo!werner < tagesordnung < CR >
$
```

Das Bereit-Zeichen in der zweiten Zeile bedeutet, daß die Nachricht mit der Datei zur Übermittlung in eine Warteschlange gestellt wird.

Verwendet man **mailx**, kann die Syntax des Kommandos **mail** nicht zur Übermittlung einer Datei verwendet werden; statt dessen ist die Option **-r** wie folgt zu verwenden:

```
$ mailx philipp
Subject: Info
-r Info
$
```

Übertragung großer Dateien

Mit den Kommandos **uucp** und **uuto** können Dateien an einen fernen Rechner übertragen werden. **uucp** ermöglicht die Übermittlung von Dateien in das gewünschte Verzeichnis auf dem Zielsystem. Übermittelt man eine Datei in ein Verzeichnis, dessen Eigentümer man ist, hat man auch das erforderliche Zugriffsrecht, um die Datei in dieses Verzeichnis zu stellen. Informationen zu den Zugriffsrechten auf Verzeichnisse und Dateien sind in Kapitel 3 enthalten. Will man jedoch eine Datei in ein Verzeichnis eines anderen Benutzers übermitteln,

muß man zuvor sicherstellen, daß man das Recht hat, eine Datei in sein Verzeichnis zu schreiben. Außerdem sind die erforderlichen Pfadnamen häufig sehr lang, und da sie vollständig eingegeben werden müssen, sind Kommandozeilen mit **uucp** häufig kompliziert und fehlerträchtig.

Für diese Zwecke ist das Kommando **uuto** besser geeignet. Er ist eine erweiterte Version des Kommandos **uucp**; mit ihm kann man Dateien automatisch in ein öffentliches Verzeichnis mit dem Namen **/usr/spool/uucppublic** auf dem System des Empfängers übermitteln lassen. Das bedeutet, daß man die Zieldatei zwar nicht frei wählen kann, daß man jedoch jederzeit eine Datei übermitteln kann, ohne vom Eigentümer des Zielverzeichnisses zuerst das Schreibrecht einholen zu müssen.

Schließlich ist eine Kommandozeile mit dem Kommando **uuto** kürzer und weniger kompliziert als eine Kommandozeile mit dem Kommando **uucp**. Daher ist die Wahrscheinlichkeit einer fehlerhaften Eingabe bei **uuto** viel geringer.

Übertragung einer Datei vorbereiten

Bevor man eine Datei mit dem Kommando **uucp** oder **uuto** senden kann, muß man feststellen, ob die Datei übertragbar ist. Dazu sind die Zugriffsrechte der Datei zu prüfen. Sind sie nicht entsprechend definiert, muß man sie mit dem Kommando **chmod** ändern, sofern man Eigentümer der Datei ist (Zugriffsrechte und das Kommando **chmod** sind in Kapitel 3 beschrieben).

Damit eine Datei mit den Kommandos **uucp** oder **uuto** übertragen werden kann, müssen zwei Kriterien in bezug auf Zugriffsrechte erfüllt sein.

- Für die zu übertragende Datei muß das Leserecht (**r**) für andere Benutzer gelten.
- Für das Verzeichnis, in dem die Datei enthalten ist, muß das Lese- (**r**) und das Ausführbarkeitsrecht (**x**) für andere Benutzer definiert sein.

Im folgenden Beispiel wird davon ausgegangen, daß eine Kopie einer Datei mit dem Namen **huhn** aus dem Verzeichnis mit dem Namen **suppe** (im Home-Verzeichnis) mit dem Kommando **uuto** an einen anderen Benutzer übertragen werden soll. Dazu sind zuerst die Zugriffsrechte auf **suppe** zu prüfen:

```
$ ls -l<CR>
total 4
drwxr-xr-x    2      leser  gruppe1  45 Feb 9 10:43  suppe
$
```

Durch das Kommando `ls` wird hier angezeigt, daß für das Verzeichnis `suppe` für alle drei Gruppen Lese- (r) und Ausführbarkeitsrecht (x) gelten; es müssen keine Änderungen vorgenommen werden. Nun mit dem Kommando `cd` aus dem Home-Verzeichnis in das Verzeichnis `suppe` wechseln und die Zugriffsrechte auf die Datei `huhn` prüfen:

```
$ ls -l huhn<CR>
total 4
-rw-----    1      leser  gruppe1 3101   Mar 1 18:22  huhn
$
```

Die Ausgabe aus diesem Kommando bedeutet, daß der Eigentümer, jedoch niemand anders, das Leserecht auf die Datei `huhn` hat. Daher ist hier das Leserecht für die eigene Gruppe (g) und andere (o) mit dem Kommando `chmod` zu definieren:

```
$ chmod go+r huhn<CR>
```

Nun die Zugriffsrechte nochmals mit dem Kommando `ls -l` prüfen:


```

$ ls -lhuhn<CR>
total 4
-rw-r--r--  1      leser  gruppe1 3101   Mar01 18:22  huhn
$

```

Damit steht fest, daß die Datei nun übertragbar ist, und man kann sie mit dem Kommando **uucp** oder **uuto** absenden. Nachdem die Kopien der Datei abgesandt sind, kann man den Vorgang umkehren und die vorherigen Zugriffsrechte wiederherstellen.

Das Kommando uucp

Mit dem Kommando **uucp** (kurz für "UNIX-to-UNIX system copy") kann man eine Datei direkt in das Home-Verzeichnis eines anderen Benutzers an einem anderen Rechner oder in ein beliebiges Verzeichnis kopieren, für das man das Schreibrecht hat.

uucp ist kein interaktives Kommando, sondern arbeitet für den Benutzer unsichtbar. Nach dem Aufrufen des Kommandos kann man andere Prozesse ausführen.

Zur Übertragung von Dateien zwischen Rechnern sind mehrere Schritte erforderlich. Zuerst wird eine Arbeitsdatei mit Anweisungen für die Dateiübertragung erstellt. Auf Anforderung wird auch eine Datendatei (eine Kopie der zu übertragenden Datei) erstellt. Dann ist die Datei bereit zur Übertragung. Nach dem Aufrufen des Programms **uucp** führt es die vorbereitenden Schritte wie oben beschrieben aus (Erstellen der erforderlichen Dateien in einem speziellen Verzeichnis, das **Spool-Verzeichnis** genannt wird) und ruft dann den Dämon **uucico** auf, der die eigentliche Übertragung vornimmt. Dämonen sind Systemprozesse, die im Hintergrund laufen. Die Datei wird in einer Warteschlange vorgemerkt und von **uucico** zum nächstmöglichen Zeitpunkt übertragen.

Daher kann man mit dem Kommando **uucp** Dateien auf einen fernen Rechner übertragen, ohne daß man nähere Informationen benötigt, abgesehen vom Namen des fernen Rechners und eventuell dem Benutzernamen des/der fernen Benutzer, an den/die die Datei übertragen wird.

Kommandozeilensyntax

Mit **uucp** sind folgende Übertragungsformen möglich:

- Übertragen einer Datei in eine Datei oder ein Verzeichnis
- Übertragen mehrerer Dateien in ein Verzeichnis.

Damit die Dateien zugestellt werden können, muß das Programm **uucp** den vollständigen Pfadnamen der *Quelldatei* und der *Zieldatei* kennen. Das bedeutet jedoch nicht, daß man jedesmal den vollständigen Pfadnamen eingeben muß, wenn man das Kommando **uucp** verwendet. Es gibt verschiedene Abkürzungen, die man einsetzen kann, sobald man das jeweilige Format kennt; durch **uucp** werden sie in die vollständigen Pfadnamen erweitert.

Die Angabe der richtigen Bezeichnungen für die *Quelldatei* und *Zieldatei* beginnt mit der Angabe der Position der *Quelldatei* relativ zur eigenen aktuellen Position im Dateisystem. Hier wird davon ausgegangen, daß sich die *Quelldatei* auf dem lokalen System befindet. Steht die *Quelldatei* im aktuellen Verzeichnis, kann man ihren Namen ohne Pfadangabe eingeben. Steht die *Quelldatei* nicht im aktuellen Verzeichnis, ist sie mit vollständigem Pfadnamen anzugeben.

Da sich die *Zieldatei* auf einem fernen System befindet, muß sie stets mit einem Pfadnamen angegeben werden, der mit dem Namen des fernen Systems beginnt. Danach läßt das Programm **uucp** dem Benutzer jedoch die Wahl, den vollständigen Pfadnamen oder eine von zwei Formen von Abkürzungen zu verwenden. Die *Zieldatei* kann daher in einem der folgenden Formate angegeben werden:

- *Systemname!vollständiger_Pfadname*
- *Systemname!~Benutzername[/Verzeichnis/Dateiname]*
- *Systemname!~/Benutzername[/Verzeichnis/Dateiname]*

Der Benutzername bezieht sich hier auf den Empfänger der Datei.

Bisher ging es um das Verfahren bei der Übertragung einer Datei vom lokalen System zu einem fernen System. Man kann jedoch mit **uucp** auch eine Datei von einem fernen System zum lokalen System übertragen lassen. In beiden Fällen können die oben beschriebenen Formate zur Angabe der *Quelldateien* oder *Zieldateien* verwendet werden. Der Hauptunterschied bei der Auswahl eines dieser Formate besteht nicht darin, ob eine Datei eine *Quelldatei* oder *Zieldatei* ist, sondern wo sich der Benutzer im Dateisystem, relativ zu den Dateien, die er angibt, gerade befindet. Daher kann sich in den obigen Formaten der *Benutzername* auf den Benutzernamen des Eigentümers oder des Empfängers einer *Quelldatei* oder *Zieldatei* beziehen.

Im folgenden Beispiel hat der Eigentümer den Benutzernamen **kol** auf einem System mit dem Namen **minna**. Sein Home-Verzeichnis ist **/usr/kol**, und er will eine Datei mit dem Namen **kap1** (aus einem Verzeichnis mit dem Namen **text** im Home-Verzeichnis) an den Benutzernamen **wsm** auf einem System mit dem Namen **mistel** übertragen. Im Moment befindet er sich in **/usr/kol/text** und kann daher die *Quelldatei* mit ihrem relativen Pfadnamen, **kap1**, angeben. Die *Zieldatei* kann mit einer der folgenden Kommandozeilen angegeben werden:

- Angabe der *Zieldatei* mit vollständigem Pfadnamen:
uucp kap1 mistel!/usr/wsm/empfang/kap1
- Angabe der *Zieldatei* mit dem *~Benutzernamen* (wird zum Namen des Home-Verzeichnisses des Empfängers erweitert) und einem Namen für die neue Datei:
uucp kap1 mistel!~wsm/empfang/kap1
 (die Datei wird in **mistel!/usr/wsm/empfang/kap1** übertragen).
- Angabe der *Zieldatei* mit dem *~Benutzernamen* (erweitert zum Home-Verzeichnis des Empfängers), jedoch ohne Namen für die neue Datei; in diesem Fall erhält die neue Datei denselben Namen wie die *Quelldatei*:
uucp kap1 mistel!~wsm/empfang
 (die Datei wird in **mistel!/usr/wsm/empfang/kap1** übertragen).
- Angabe der *Zieldatei* mit *~/Benutzername*. Diese Angabe wird erweitert zum Unterverzeichnis des Empfängers im öffentlichen Verzeichnis auf dem fernen System:
uucp kap1 mistel!~/wsm
 (die Datei wird in **mistel!/usr/usr/spool/uucppublic/wsm** übertragen).

Beispiele zur Verwendung von uucp mit Optionen

Im folgenden Beispiel wird eine Datei mit dem Namen **protokoll** auf einen fernen Rechner mit dem Namen **adler** übertragen; dazu ist folgende Kommandozeile einzugeben:

```
$ uucp -m -s status -j protokoll adler!usr/gws/protokoll<CR>
adlerN3f45
$
```

Die Datei **protokoll** (im aktuellen Verzeichnis des lokalen Rechners) wird zum fernen Rechner **adler** übertragen und unter dem Pfadnamen **/usr/gws** in eine Datei mit dem Namen **protokoll** geschrieben. Nach dem Ende der Übertragung wird der Benutzer **gws** am fernen Rechner durch die elektronische Post benachrichtigt.

Die Option **-m** bewirkt, daß der Absender ebenfalls durch die elektronische Post benachrichtigt wird, ob die Übertragung erfolgreich ausgeführt wurde. Die Option **-s**, gefolgt von einem Dateinamen (hier **status**), bewirkt, daß ein Statusbericht der Dateiübertragung in die angegebene Datei (**status**) geschrieben wird.

NOTE

Nach der Option **-s** ist stets ein Dateiname einzugeben, da andernfalls die Übertragung mit folgender Fehlermeldung abgebrochen wird: `uucp failed completely.`

Die Option **-j** bewirkt, daß die Auftragsnummer (**adlerN3f45**) angezeigt wird.

Selbst wenn `uucp` nicht schon kurze Zeit nach dem Absenden einer Datei die erfolgreiche Ausführung meldet, braucht man nicht anzunehmen, daß die Übertragung fehlgeschlagen ist. Nicht alle Systeme mit Netzwerksoftware haben auch die Hardware-Einrichtungen, die zum Anrufen anderer Systeme erforderlich sind. Dateien, die von diesen sogenannten passiven Systemen übertragen werden, werden in bestimmten Abständen von aktiven Systemen abgerufen, die mit der erforderlichen Hardware ausgerüstet sind (nähere Angaben sind unter "Funktion des Kommandos `uucp`" zu finden). Bei der Übertragung von Dateien von einem passiven System kann es daher gewisse Verzögerungen geben. Beim Systemverwalter ist zu erfahren, ob es sich bei einem System um ein passives oder aktives System handelt.

Im obigen Beispiel wird die *Zieldatei* mit ihrem vollständigen Pfadnamen angegeben. Die *Zieldatei* kann auch in zwei weiteren Formen angegeben werden:

- Das Benutzerverzeichnis von `gws` kann mit der Tilde (`~`) wie folgt angegeben werden:

```
adler!~gws/protokoll
```

Dies wird als folgende Angabe interpretiert:

```
adler!/usr/gws/protokoll
```

- Der Bereich `uucppublic` wird ähnlich durch die Tilde vor dem Pfadnamen angegeben, wie im folgenden Beispiel:

```
adler!~/gws/protokoll
```

Dies wird als folgende Angabe interpretiert:

```
/usr/spool/uucppublic/gws/protokoll
```

Funktion des Kommandos `uucp`

Dieser Abschnitt enthält eine Übersicht über die Arbeitsweise des Kommandos `uucp`. Wenn man die entsprechenden Verarbeitungsvorgänge versteht, kann man die Einschränkungen und die Voraussetzungen für die Verwendung des Kommandos besser verstehen, z. B. weshalb bestimmte Aufgaben ausgeführt werden können und andere nicht, weshalb bestimmte Aufgaben wann ausgeführt werden, und weshalb man ihn unter bestimmten Umständen einsetzen kann oder nicht. Nähere Informationen dazu sind in *System Administrator's Guide* und *System Administrator's Reference Manual* enthalten.

Gibt man ein **uucp**-Kommando ein, erstellt das Programm **uucp** eine Arbeitsdatei und in der Regel eine Datendatei für die angeforderte Übertragung; eine Datendatei wird von **uucp** nicht erstellt, wenn die Option **-c** verwendet wird. Die Arbeitsdatei enthält die für die Übertragung der Datei(en) erforderlichen Informationen, die Datendatei ist einfach eine Kopie der angegebenen Quelldatei. Nachdem diese Dateien im Spool-Verzeichnis erstellt sind, wird der Dämon **uucico** aufgerufen.

Der Dämon **uucico** versucht, eine Verbindung zu dem fernen Rechner herzustellen, an den die Datei(en) geschickt werden soll(en). Dazu werden zuerst die Informationen aus der Datei **Systems** geholt, die erforderlich sind, um eine Verbindung zum fernen Rechner aufzubauen. Dabei erhält **uucico** die Information, welche Einheit zum Einrichten der Verbindung zu verwenden ist. Das Programm **uucico** sucht in der Datei **Devices** nach den Einheiten, die die Anforderungen erfüllen, die in der Datei **Systems** aufgeführt sind. Nachdem **uucico** eine passende Einheit gefunden hat, versucht das Programm, die Verbindung herzustellen und sich am fernen Rechner anzumelden.

Bei der Anmeldung von **uucico** am fernen Rechner wird der Dämon **uucico** des fernen Rechners gestartet. Die beiden Dämonen **uucico** vereinbaren dann das Leitungsprotokoll, das für die Dateiübertragung(en) verwendet werden soll. Der lokale Dämon **uucico** überträgt dann die Datei(en) zum fernen Rechner und der ferne Dämon **uucico** transportiert die Dateien an die mit Pfadnamen angegebenen Ziele im fernen Rechner. Nachdem der lokale Rechner die Übertragung abgeschlossen hat, kann der ferne Rechner Dateien übermitteln, die für die Übertragung zum lokalen Rechner vorgemerkt sind. Durch einen Eintrag in der Datei **Permissions** kann man dem fernen Rechner das Recht zur Übertragung auch verweigern. In diesem Fall muß vom fernen Rechner zuerst eine Verbindung zum lokalen Rechner eingerichtet werden, um Dateien zu übertragen.

Ist der ferne Rechner oder die für die Einrichtung der Verbindung zum fernen Rechner gewählte Einheit nicht verfügbar, bleibt der Auftrag im Spool-Verzeichnis vorgemerkt. In Abständen von einer Stunde (Standardwert) wird **uudemon.hour** durch **cron** aufgerufen, und durch diesen Prozeß wird wiederum der Dämon **uusched** aufgerufen. Der Dämon **uusched** durchsucht das Spool-Verzeichnis nach verbliebenen Arbeitsdateien, legt auf Zufallsbasis eine Reihenfolge für ihre Verarbeitung fest und ruft dann den in den vorigen Absätzen beschriebenen Übertragungsprozeß (**uucico**) auf.

Der hier beschriebene Übertragungsvorgang gilt allgemein für einen aktiven Rechner. Ein aktiver Rechner, d. h. einer, der mit Hardware zum Anrufen anderer Rechner und mit Netzwerksoftware ausgestattet ist, kann so konfiguriert werden, daß er einen passiven Rechner nach Dateien abfragt. Ein passiver Rechner mit Netzwerksoftware kann Dateiübertragungen vormerken, jedoch keine fernen Rechner anrufen, da er nicht über die erforderliche Hardware verfügt. In die Datei **Poll** (Pfad: `/usr/lib/uucp/Poll`) werden die Rechner eingetragen, die in dieser Form abgefragt werden sollen.

In Abbildung 8-6 werden die Syntax und die Funktionen des Kommandos **uucp** zusammengefaßt.

Kommandoübersicht		
uucp – Kopieren einer Datei von einem Rechner zu einem anderen		
<i>Kommando</i>	<i>Optionen</i>	<i>Argumente</i>
uucp -j1, -m, -s u.a.*	<i>Quelldatei</i>	
Beschreibung:	<p>uucp führt die erforderlichen Vorbereitungen für das Kopieren einer Datei von einem Rechner zu einem anderen aus und ruft uucico auf, den Dämon (Hintergrundprozeß), der die eigentliche Übertragung der Datei durchführt. Der Benutzer braucht dabei lediglich das Kommando uucp mit der zu kopierenden Datei einzugeben.</p>	
Bemerkungen:	<p>Standardmäßig kann man Dateien nur in das Verzeichnis /usr/spool/uucppublic übertragen lassen. Sollen Dateien in Verzeichnisse anderer Benutzer geschrieben werden, muß man das Schreibrecht von diesem Benutzer erhalten. Es gibt zwar mehrere Möglichkeiten, Pfadnamen als Argumente einzugeben, es wird jedoch empfohlen, vollständige Pfadnamen einzugeben, um Verwechslungen zu vermeiden.</p>	

Abbildung 8-6: Übersicht über das Kommando **uucp**

* Eine Beschreibung aller verfügbaren Optionen ist unter **uucp(1)** im *User's Reference Manual* zu finden.

Das Kommando `uuto`

Mit dem Kommando `uuto` können Dateien in das öffentliche Verzeichnis eines anderen Systems übertragen werden. Das allgemeine Format für das Kommando `uuto` lautet:

```
uuto Dateiname System!Benutzername <CR>
```

Dabei steht *Dateiname* für den Namen der zu übertragenden Datei, *System* für das System des Empfängers und *Benutzername* für den Benutzernamen des Empfängers.

Soll eine Datei an einen Benutzer am lokalen System übertragen werden, kann der Systemname weggelassen werden; dann gilt folgendes Format:

```
uuto Dateiname Benutzername <CR>
```

Senden einer Datei: Option `-m` und Kommando `uustat`

Es wurde bereits erläutert, wie festgestellt werden kann, ob eine Datei übergelaufen ist. Im folgenden wird der Ablauf einer Dateiübertragung anhand eines Beispiels dargestellt.

Der Vorgang einer Dateiübertragung mit `uuto` wird als Auftrag bezeichnet. Nach der Eingabe eines Kommandos `uuto` wird der Auftrag nicht sofort ausgeführt, sondern die Datei wird zuerst in einer Warteschlange von Aufträgen vorge­merkt und erhält eine Auftragsnummer. Wenn die Auftragsnummer an der Reihe ist, wird die Datei zum fernen System übertragen und dort in ein öffentliches Verzeichnis gestellt. Dem Empfänger wird der Eingang der Datei durch eine `mail`-Nachricht gemeldet; er kann die Datei dann mit dem Kommando `uupick` (weiter unten in diesem Kapitel beschrieben) abrufen.



Für die folgenden Ausführungen sollen folgende Namen gelten:

werner	der eigene Benutzername
sys1	der Name des eigenen Systems
marie	der Benutzername des Empfängers
sys2	der Name des Empfänger-Systems
geld	zu übertragende Datei

Es wird auch davon ausgegangen, daß die beiden Systeme miteinander kommunizieren können.

Mit folgender Eingabe wird die Datei **geld** an den Benutzernamen **marie** auf dem System **sys2** übertragen:

```
$ uuto geld sys2!marie<CR>
$
```

Das Bereit-Zeichen in der zweiten Zeile bedeutet, daß die Datei in eine Warteschlange eingereiht wurde. Der Auftrag ist damit abgesandt; man kann nun lediglich auf die Bestätigung warten, daß er am Ziel angekommen ist.

Die einfachste Methode, dies zu erfahren, besteht darin, in die Kommandozeile von **uuto** die Option **-m** mit einzugeben, wie im folgenden Beispiel:

```
$ uuto -m geld sys2!marie<CR>
$
```

Damit erhält man eine Rückmeldung durch **mail**, wenn der Auftrag am System des Empfängers angekommen ist. Die Meldung sieht etwa so aus:

```
$ mail<CR>
From uucp Wed Apr 26 09:45 MES 1989
file /sys1/werner/geld, system sys1
copy succeeded
?
```

Soll geprüft werden, ob der Auftrag abgesandt wurde, kann man das Kommando **uustat** verwenden; damit kann man sich für alle laufenden Aufträge mit **uucp** und **uto** den Status anzeigen lassen, wie im folgenden Beispiel:

```
$ uustat <CR>
1145 werner sys2 10/05-09:31 10/05-09:33 JOB IS QUEUED
$
```

Diese Beispielmeldung enthält folgende Bestandteile:

- 1145 ist die Auftragsnummer, die der Übertragung der Datei **geld** an **marie** am System **sys2** zugewiesen wurde.
- **werner** ist der Benutzername der Person, die den Auftrag in Auftrag gegeben hat.
- **sys2** ist das System des Empfängers.
- 10/05-09:31 sind Datum und Uhrzeit der Eingabe des Auftrags.
- 10/05-09:33 sind Datum und Uhrzeit der Übermittlung dieser **uustat**-Nachricht.
- Der letzte Teil gibt den Status des Auftrags an. Hier wird angezeigt, daß der Auftrag noch in der Warteschlange steht, jedoch noch nicht übermittelt wurde.

Man kann sich auch den Status für einen einzelnen Auftrag mit **uto** ausgeben lassen, indem man die Option **-j** mit der betreffenden Auftragsnummer in der Kommandozeile eingibt:

```
uustat -jAuftragsnummer <CR>
```

Für den Auftrag des obigen Beispiels wäre die Auftragsnummer 1145 nach der Option **-j** einzugeben:

```
$ uustat -j1145<CR>  
1145 werner sys2 10/05-09:31 10/05-09:37 COPY FINISHED, JOB DELETED  
$
```

In dieser Statusmeldung wird angezeigt, daß der Auftrag ausgeführt und aus der Warteschlange gelöscht wurde; die Datei befindet sich nun im öffentlichen Verzeichnis des Systems des Empfängers. Weitere Statusmeldungen und Optionen zum Kommando `uustat` sind im *User's Reference Manual* beschrieben.

Dies ist alles, was zur Übertragung von Dateien erforderlich ist. Man sollte diesen Vorgang nun üben; dazu kann man eine Datei an sich selbst senden.

In den Abbildungen 8-7 und 8-8 werden die Syntax und die Funktionen der Kommandos `uuto` bzw. `uustat` zusammengefaßt.

Kommandoübersicht		
uuto – Übertragen von Dateien an andere Benutzer		
<i>Kommando</i>	<i>Optionen</i>	<i>Argumente</i>
uuto	-m u.a.*	<i>Dateisystem!Benutzername</i>
Beschreibung:	Mit uuto kann man eine Datei in das öffentliche Verzeichnis eines anderen Systems übertragen und den Empfänger benachrichtigen lassen (durch elektronische Post mit seinem/ihrer Benutzernamen als Adresse), wenn die Datei angekommen ist.	
Bemerkungen:	Für zu übertragende Dateien muß das Leserecht für andere Benutzer definiert sein; für das Elternverzeichnis der Datei muß das Leserecht und Ausführbarkeitsrecht für andere Benutzer definiert sein. Die Option -m bewirkt, daß der Absender durch elektronische Post benachrichtigt wird, wenn die Datei angekommen ist.	

Abbildung 8-7: Übersicht über das Kommando **uuto**

- * Eine Beschreibung aller verfügbaren Optionen ist unter **uuto(1)** im *User's Reference Manual* zu finden.

Kommandoübersicht		
uustat – Prüfen des Status eines Auftrags mit uucp oder uuto		
<i>Kommando</i>	<i>Optionen</i>	<i>Argumente</i>
uustat	-j u.a.*	keine
Beschreibung:	Durch uustat wird der Status aller in Auftrag gegebenen Aufträge mit uucp und uuto angezeigt.	
Bemerkungen:	Mit der Option -j , gefolgt von der Auftragsnummer, kann man sich den Status nur für diesen bestimmten Auftrag anzeigen lassen.	

Abbildung 8-8: Übersicht über das Kommando **uustat**

- * Eine Beschreibung aller verfügbaren Optionen ist unter **uustat(1)** im *User's Reference Manual* zu finden.

Empfang von mit **uuto** übertragenen Dateien: das Kommando **uupick**

Wenn eine Datei, die mit **uuto** übertragen wurde, im öffentlichen Verzeichnis des UNIX-Systems ankommt, erhält der Empfänger eine **mail**-Nachricht. Im obigen Beispiel erhält die Eigentümerin des Benutzernamens **marie** die folgende Nachricht, sobald die Datei **geld** im öffentlichen Verzeichnis ihres Systems angekommen ist:

```

$ mail
From uucp Wed Apr 26 17:46:18 MES 1989
/usr/spool/uucppublic/empfang/marie/sys1//geld from sys1!werner arrived
$

```

Diese Meldung enthält folgende Informationen:

- In der ersten Zeile wird angezeigt, wann die Datei am Ziel angekommen ist.
- Der erste Teil der zweiten Zeile, bis zu den zwei Schrägstrichen (//), gibt den Pfadnamen zu dem Teil des öffentlichen Verzeichnisses an, in dem die Datei gespeichert wurde.
- Der Rest der Zeile (nach den beiden Schrägstrichen) gibt den Namen der Datei und den Absender an.

Nachdem man die **mail**-Meldung wieder gelöscht hat, kann man die Datei mit dem Kommando **uupick** an der gewünschten Stelle speichern; dazu ist folgendes Kommando neben dem Bereit-Zeichen einzugeben:

```
uupick <CR>
```

Das Programm sucht im öffentlichen Verzeichnis nach Dateien, die für den Benutzer eingegangen sind. Findet es solche, werden die Dateinamen ausgegeben. Dann fordert ein Fragezeichen (?) dazu auf, weitere Anweisungen einzugeben.

In diesem Beispiel gibt die Eigentümerin des Benutzernamens **marie** das Kommando **uupick** ein, um die Datei **geld** abzurufen. Dabei erscheint etwa folgende Anzeige auf dem Bildschirm:

```

$ uupick <CR>
from system sys1: file geld
?

```

Hier gibt es mehrere Möglichkeiten der Reaktion; diese sollen im folgenden erläutert werden.

Zunächst sollte man die Datei aus dem öffentlichen Verzeichnis in das Benutzerverzeichnis verschieben; dazu ist neben dem Fragezeichen ein **m** einzugeben:

```
?  
m <CR>  
$
```

Damit wird die Datei in das aktuelle Verzeichnis verschoben; soll sie in ein anderes Verzeichnis geschrieben werden, ist nach dem **m** der betreffende Verzeichnisname einzugeben:

```
?  
m Verzeichnis <CR>
```

Sind noch andere Dateien für den Benutzer im öffentlichen Verzeichnis enthalten, wird die nächste angezeigt, gefolgt von einem Fragezeichen. Falls nicht, gibt **uupick** wieder ein Bereit-Zeichen aus.

Soll zu diesem Zeitpunkt die Datei unverändert im öffentlichen Verzeichnis belassen werden, ist auf das Fragezeichen hin die RETURN-Taste zu drücken:

```
?  
<CR>
```

Die aktuelle Datei bleibt im öffentlichen Verzeichnis, bis das Kommando **uupick** zum nächsten Mal wieder eingegeben wird. Liegen keine weiteren Nachrichten vor, gibt das System wieder ein Bereit-Zeichen aus.

Weiß man zu diesem Zeitpunkt bereits, daß man die Datei nicht sichern will, kann man sie durch Eingabe von **d** neben dem Fragezeichen löschen:

```
?  
d <CR>
```

Damit wird die aktuelle Datei aus dem öffentlichen Verzeichnis gelöscht und die nächste Nachricht (sofern vorhanden) angezeigt. Liegen keine weiteren Meldungen über eingegangene Dateien vor, erscheint das Bereit-Zeichen.

Schließlich kann man das Programm **uupick** beenden, indem man neben dem Fragezeichen ein **q** eingibt:

```
?  
q <CR>
```

Alle nicht verschobenen und nicht gelöschten Dateien bleiben im öffentlichen Verzeichnis erhalten, bis man das Kommando **uupick** wieder eingibt.

Weitere mögliche Reaktionen sind im *User's Reference Manual* aufgeführt.

In Abbildung 8-9 werden die Syntax und die Funktionen des Kommandos **uupick** zusammengefaßt.

Kommandoübersicht		
uupick – Suchen nach Dateien, die mit uuto oder uucp übertragen wurden		
<i>Kommando</i>	<i>Optionen</i>	<i>Argumente</i>
uupick	-s	
Beschreibung:	uupick sucht im öffentlichen Verzeichnis des eigenen Systems nach Dateien, die mit uuto oder uucp übertragen wurden. Sind solche vorhanden, werden Informationen über sie ausgegeben, und der Benutzer wird zur Eingabe einer Anweisung aufgefordert.	
Bemerkungen:	Das Fragezeichen (?) nach der Nachricht bedeutet, daß eine Anweisung des Benutzers erwartet wird. Eine vollständige Liste der möglichen Eingaben ist im <i>User's Reference Manual</i> enthalten.	

Abbildung 8-9: Übersicht über das Kommando **uupick**

Vernetzung

Bei der Vernetzung werden Rechner und Terminals miteinander verbunden, um den Benutzern folgende Möglichkeiten zu geben:

- Anmelden an einem fernen Rechner wie am lokalen Rechner
- Anmelden und Arbeiten an zwei Rechnern in einer Arbeitssitzung, ohne daß man sich immer wieder aufs Neue vom einen abmelden und am anderen anmelden muß
- Austauschen von Daten zwischen Rechnern.

Die in diesem Abschnitt eingeführten Kommandos ermöglichen diese Funktionen. Mit dem Kommando `ct` kann man den eigenen Rechner mit einem fernen Terminal verbinden, das mit einem Modem ausgestattet ist; mit dem Kommando `cu` kann man den eigenen Rechner mit einem fernen Rechner verbinden, und mit dem Kommando `uux` kann man Programme (Kommandos) auf einem fernen System ausführen, ohne an diesem angemeldet zu sein.



Bei bestimmten kleineren Rechnern hängt die Verfügbarkeit dieser Kommandos davon ab, ob Netzwerksoftware installiert ist. Gibt man an einem System, auf dem keine Netzwerksoftware installiert ist, ein Netzwerkkommando ein, erscheint eine Fehlermeldung folgender Art:

```
cu: not found
```

Beim Systemverwalter ist zu erfahren, ob solche Netzwerkkommandos auf dem eigenen UNIX-System verfügbar sind.

Anschluß eines fernen Terminals: das Kommando `ct`

Mit dem Kommando `ct` wird der eigene Rechner mit einem fernen Terminal verbunden, das mit einem Modem ausgestattet ist, so daß sich ein Benutzer an diesem Terminal anmelden kann. Dazu wählt das Kommando die Telefonnummer des Modems an. Das Modem muß den Anruf automatisch beantworten können; stellt das Programm `ct` fest, daß der Anruf angenommen wird, startet es den Anmeldeprozeß `getty` für das ferne Terminal, so daß ein Benutzer sich über das Terminal am Rechner anmelden kann.

Das Kommando kann auch vom anderen Ende der Leitung aus eingegeben werden, d. h. vom fernen Terminal selbst. Benutzt man ein Terminal, das weit entfernt vom Rechner steht, und will man hohe Kosten für Ferngespräche vermeiden, kann man sich durch Eingabe von `ct` vom Rechner anrufen lassen. Dazu muß man einfach den Rechner anrufen, sich anmelden und das Kommando `ct` eingeben. Der Rechner beendet dann die aktuelle Verbindung und ruft seinerseits das ferne Terminal zurück.

Findet das Programm `ct` keine verfügbare Wähleinheit, meldet es, daß alle Wähleinheiten belegt sind, und fragt, ob es warten soll, bis eine frei wird. Antwortet man mit ja, fragt es, wie lange gewartet werden soll (in Minuten).

Kommandozeilenformat

Das Kommando `ct` wird nach folgendem Format aufgerufen:

```
ct [Optionen] TelNr <CR>
```

Das Argument *TelNr* steht hier für die Telefonnummer des fernen Terminals.

Beispiele

Im folgenden Beispiel wird davon ausgegangen, daß man an einem lokalen Terminal angemeldet ist, und daß man ein fernes Terminal mit dem eigenen Rechner verbinden will; die Telefonnummer des Modem am fernen Terminal lautet 932-3497. Folgende Kommandozeile ist einzugeben:

```
ct -h -w5 -s1200 9=9323497 <CR>
```

NOTE

Das Gleichheitszeichen (=) steht für einen sekundären Wählton, Bindestriche (-) nach der Telefonnummer stehen für Wartezeiten (bei Fernverbindungen).

Durch `ct` wird hier das Modem über eine Wähleinheit mit einer Übertragungsrate von 1200 Baud angewählt. Für den Fall, daß keine Wähleinheit verfügbar ist, bewirkt die Option `-w5`, daß `ct` fünf Minuten darauf wartet, daß eine Wähleinheit frei wird; sonst wird der Vorgang abgebrochen. Mit der Option `-h` wird `ct` angewiesen, die Verbindung mit dem lokalen Terminal (über das das Kommando eingegeben wurde) nicht zu unterbrechen.

Nun soll die Anmeldung am Rechner vom fernen Terminal aus erfolgen. Durch folgende Eingabe mit `ct` erreicht man, daß der Rechner das ferne Terminal anruft:

```
ct -s1200 9=9323497 <CR>
```

Ist keine Wähleinheit verfügbar, gibt `ct` folgende Meldung aus, da die Option `-w` nicht angegeben wurde:

```
1 busy dialer at 1200 baud Wait for dialer?
```

Gibt man hier `n` (nein) ein, wird `ct` beendet, gibt man `y` (ja) ein, fordert `ct` den Benutzer auf, eine Wartezeit in Minuten anzugeben:

```
Time, in minutes?
```

Wird in dieser Zeit eine Wähleinheit frei, gibt `ct` folgende Meldung aus:

```
Allocated dialer at 1200 baud
```

Das bedeutet, daß eine Wähleinheit zur Verfügung steht. In jedem Fall fragt `ct` danach, ob die Verbindung vom fernen Terminal zum Rechner beendet werden soll:

```
Confirm hangup?
```

Gibt man hier `y` (ja) ein, erfolgt die Abmeldung, und `ct` ruft das ferne Terminal zurück, sobald eine Wähleinheit zur Verfügung steht. Gibt man `n` (nein) ein, wird das Programm `ct` beendet und man bleibt am Rechner angemeldet.

In Abbildung 8-10 werden die Syntax und die Funktionen des Kommandos `ct` zusammengefaßt.

Kommandoübersicht		
ct – Verbinden eines Rechners mit einem fernen Terminal		
Kommando	Optionen	Argumente
ct	-h, -w, -s u.a.*	TelNr
Beschreibung:	Mit ct kann ein Rechner mit einem fernen Terminal verbunden werden, so daß sich ein Benutzer über dieses Terminal am Rechner anmelden kann.	
Bemerkungen:	Das ferne Terminal muß mit einem Modem ausgestattet sein, der Anrufe automatisch beantworten kann.	

Abbildung 8-10: Übersicht über das Kommando ct

* Eine Beschreibung aller verfügbaren Optionen ist unter ct(1) in *User's Reference Manual* zu finden.

Andere UNIX-Systeme anrufen: das Kommando cu

Mit dem Kommando **cu** kann ein ferner Rechner mit dem eigenen Rechner verbunden werden, und man kann gleichzeitig an beiden Rechnern angemeldet sein. Das bedeutet, daß man zwischen den Rechnern hin- und herspringen, zwischen ihnen Dateien übertragen und auf beiden Kommandos ausführen kann, ohne die Verbindung zwischendurch abbrechen zu müssen.

Die Funktion des Kommandos **cu** hängt von den Angaben in der Kommandozeile ab. In der Kommandozeile ist die Telefonnummer oder der Systemname des fernen Rechners anzugeben. Gibt man eine Telefonnummer an, wird sie an das Selbstwählmodem übergeben, gibt man einen Systemnamen an, sucht sich **cu** die Telefonnummer aus der Datei **Systems** heraus. Wird kein Selbstwählmodem zur Herstellung der Verbindung verwendet, kann die Leitung (der Anschluß) in der Kommandozeile angegeben werden, die der direkten Übertragungsverbindung zum fernen Rechner zugeordnet ist.

Nachdem die Verbindung hergestellt ist, fordert der Rechner den Benutzer dazu auf, sich anzumelden. Hat man die Arbeit am fernen Terminal beendet, meldet man sich durch Eingabe von `<~.>` ab; gleichzeitig wird die Verbindung beendet. Die Anmeldung am lokalen Rechner besteht jedoch weiter.

NOTE

Das Programm `cu` kann keine Fehler erkennen oder beheben; bei der Dateiübertragung können Dateien verlorengehen oder verstümmelt werden. Nach der Übertragung kann man deshalb prüfen, ob Daten verlorengegangen sind, indem man das Kommando `sum` oder `ls -l` mit der übertragenen Datei und der empfangenen Datei ausführt. Mit diesen Kommandos wird jeweils die Gesamtzahl von Byte in den Dateien ausgegeben; stimmen sie überein, war die Übertragung fehlerfrei. Das Kommando `sum` ist dabei schneller und erzeugt eine leichter interpretierbare Ausgabe. Nähere Angaben dazu sind unter `sum(1)` und `ls(1)` im *User's Reference Manual* enthalten.

Kommandozeilenformat

Das Kommando `cu` wird mit folgendem Format aufgerufen:

```
cu [Optionen] TelNr | Systemname <CR>
```

Diese Kommandozeile hat folgende Bestandteile:

TelNr Die Telefonnummer eines fernen Rechners

Gleichheitszeichen (=) stehen für sekundäre Wähltöne, Bindestriche (-) für Wartezeiten von vier Sekunden.

Systemname Ein Systemname aus der Datei `Systems`.

Das Programm `cu` holt sich die Telefonnummer und die Baudrate aus der Datei `Systems` und sucht nach einer entsprechenden Wähleinheit. Die Optionen `-s`, `-n` und `-l` sollten nicht zusammen mit `Systemname` verwendet werden. Die Liste der Rechner, die in der Datei `Systems` stehen, kann man sich mit dem Kommando `uname` ausgeben lassen.

Nachdem die Verbindung vom Terminal besteht und der Benutzer am fernen Rechner angemeldet ist, gehen alle Standardeingaben (Eingaben über die Tastatur) an den fernen Rechner. In den Abbildungen 8-11 und 8-12 werden die Kommandos dargestellt, die man aufrufen kann, während man über `cu` mit einem fernen Rechner verbunden ist.

Kommando	Funktion
~.	Übertragungsverbindung beenden.
~!	Zum lokalen Rechner wechseln, ohne die Verbindung abubrechen. Die Rückkehr zum fernen Rechner erfolgt durch Eingabe von <^d> (CONTROL-d).
~!Kommando	Das Kommando auf dem lokalen Rechner ausführen.
~\$Kommando	Das Kommando lokal ausführen und die Ausgabe zum fernen System senden.
~%cd Pfad	Auf dem lokalen Rechner das Verzeichnis wechseln; dabei steht Pfad für den Pfadnamen oder Verzeichnisnamen.
~%take von [nach]	Eine Datei mit dem Namen von (ferner Rechner) in eine Datei mit dem Namen nach (lokaler Rechner) kopieren. Wird nach weggelassen, gilt das Argument von an beiden Stellen.
~%put von [nach]	Eine Datei mit dem Namen von (lokaler Rechner) in eine Datei mit dem Namen nach (ferner Rechner) kopieren. Wird nach weggelassen, gilt das Argument von an beiden Stellen.
~ ~ ...	Eine Zeile, die mit ~ (~ ~ ...) beginnt, an den fernen Rechner senden.
~%break	Ein BREAK-Zeichen (Unterbrechung) an den fernen Rechner senden; kann auch als ~%b angegeben werden.

Abbildung 8-11: Kommandos zur Verwendung mit cu (Seite 1 von 2)

Kommando	Funktion
~%nostop	Das Handshake-Protokoll für den Rest der Sitzung deaktivieren. Dies ist sinnvoll, wenn der ferne Rechner nicht richtig auf die Protokollzeichen reagiert.
~%debug	Die Option für Testhilfe -d aktivieren oder deaktivieren; kann auch als ~%d angegeben werden.
~t	Die Werte der Strukturvariablen für die Ein-/Ausgabe am eigenen Terminal anzeigen lassen (zur Testhilfe).
~l	Die Werte der Strukturvariablen für die Ein-/Ausgabe am Terminal für die Fernverbindung anzeigen lassen (zur Testhilfe).

Abbildung 8-12: Kommandos zur Verwendung mit cu (Seite 2 von 2)

NOTE

Für ~%put sind stty und cat auf dem fernen Rechner erforderlich. Außerdem müssen die aktuellen Tastenkombinationen zum Löschen eines Zeichens bzw. einer Zeile auf dem fernen Rechner mit denen des aktuellen Rechners übereinstimmen.

Für ~%take müssen die Kommandos echo und cat auf dem fernen Rechner verfügbar sein. Außerdem sollte der Modus stty tabs auf dem fernen Rechner definiert sein, wenn Tabulatoren beim Kopieren nicht in Leerzeichen umgewandelt werden sollen.

Beispiele

Im folgenden soll der Rechner an einen fernen Rechner mit dem Namen **adler** angeschlossen werden; die Telefonnummer für "adler" lautet 847-7867. Dazu ist folgende Kommandozeile einzugeben:

```
cu -s1200 9=8477867 <CR>
```

Mit der Option `-s1200` wird festgelegt, daß **cu** eine Wähleinheit mit 1200 Baud zum Anrufen von **adler** verwenden soll. Wird die Option `-s` nicht angegeben, verwendet **cu** eine Wähleinheit mit der Standardrate 300 Baud.

Nachdem **adler** den Anruf angenommen hat, benachrichtigt **cu** den Benutzer, daß die Verbindung hergestellt ist und fordert ihn zur Eingabe eines Benutzernamens auf:

```
connected
login:
```

Hier sind der Benutzername und das Paßwort einzugeben.

Mit dem Kommando **take** können Dateien vom fernen Rechner auf den lokalen Rechner kopiert werden. Im folgenden Beispiel soll eine Kopie der Datei mit dem Namen **angebot** auf den lokalen Rechner geholt werden. Dazu wird die Datei **angebot** mit dem folgenden Kommando aus dem aktuellen Verzeichnis auf dem fernen Rechner in das aktuelle Verzeichnis auf dem lokalen Rechner kopiert; wird dabei für die neue Datei kein Dateiname angegeben, erhält sie ebenfalls den Namen **angebot**.

```
~%take anbot <CR>
```

Das Kommando **put** wird für den umgekehrten Vorgang verwendet, d. h. zum Kopieren von Dateien vom lokalen auf den fernen Rechner. Mit folgender Eingabe wird eine Datei mit dem Namen **protokoll** aus dem aktuellen Verzeichnis auf dem lokalen Rechner zum fernen Rechner kopiert:

```
~%put protokoll protokoll.9-18 <CR>
```

In diesem Fall wurde für die neue Datei ein anderer Name definiert (**protokoll.9-18**). Die Kopie der Datei **protokoll**, die auf dem fernen Rechner angelegt wird, heißt daher **protokoll.9-18**.

In Abbildung 8-13 werden die Syntax und die Funktionen des Kommandos **cu** zusammengefaßt.

Kommandoübersicht		
cu – Verbinden eines Rechners mit einem fernen Rechner		
<i>Kommando</i>	<i>Optionen</i>	<i>Argumente</i>
cu	-s u.a.*	<i>TelNr (oder) Systemname</i>
Beschreibung:	Mit cu kann man einen Rechner mit einem fernen Rechner verbinden und an beiden gleichzeitig angemeldet sein. Nach der Anmeldung kann man zwischen den Rechnern hin- und herspringen, um Kommandos auszuführen und Dateien zu übertragen, ohne die Verbindung zwischendurch unterbrechen zu müssen.	

Abbildung 8-13: Übersicht über das Kommando **cu**

* Eine Beschreibung aller verfügbaren Optionen ist unter **cu(1)** im *User's Reference Manual* zu finden.

Kommandos auf fernen Systemen ausführen: das Kommando uux

Mit dem Kommando **uux** (kurz für "UNIX-to-UNIX system command execution") kann man UNIX-Systemkommandos auf fernen Rechnern ausführen lassen. Dabei können Dateien von verschiedenen Rechnern zusammengezogen, ein Kommando mit ihnen auf einem bestimmten Rechner ausgeführt und die Standardausgabe in eine Datei auf einem bestimmten Rechner geschrieben werden. Die Ausführung bestimmter Kommandos kann auf dem fernen Rechner eingeschränkt werden. Darf ein angefordertes Kommando nicht ausgeführt werden, wird der Benutzer durch elektronische Post darüber informiert.

Kommandozeilenformat

Für das Kommando **uux** gilt folgendes allgemeine Format:

```
uux [Optionen] Kommandofolge <CR>
```

Die *Kommandofolge* besteht dabei aus einem oder mehreren Argumenten. Alle Sonderzeichen der Shell (wie z. B. "<>|^") müssen in Anführungszeichen gesetzt werden; dazu ist entweder die gesamte *Kommandofolge* oder nur das Zeichen als separates Argument in Anführungszeichen zu setzen.

Innerhalb der *Kommandofolge* kann dem Kommando und den Dateinamen ein *Systemname!* vorangestellt sein. Alle Argumente, die keinen *Systemnamen* enthalten, werden als Argumente für das Kommando interpretiert. Als Dateinamen kann entweder ein vollständiger Pfadname oder der Name einer Datei im aktuellen Verzeichnis (des lokalen Rechners) angegeben werden.

Beispiele

Ist der eigene Rechner fest mit einem größeren Host verbunden, kann man mit **uux** Ausdrücke von Dateien erhalten, die auf dem eigenen Rechner gespeichert sind, indem man beispielsweise folgendes eingibt:

```
pr protokoll | uux -p Host!lp <CR>
```

Mit diesem Kommando wird die Datei **protokoll** zum Ausdrucken auf dem Drucker des **Host** vorgemerkt.

In Abbildung 8-14 werden die Syntax und die Funktionen des Kommandos **uux** zusammengefaßt.

Kommandoübersicht		
uux – Ausführen von Kommandos auf einem fernen Rechner		
<i>Kommando</i>	<i>Optionen</i>	<i>Argumente</i>
uux	-1, -p und andere*	<i>Kommandofolge</i>
Beschreibung:	Mit uux können UNIX-Systemkommandos auf fernen Rechnern ausgeführt werden. Dateien können von verschiedenen Rechnern gesammelt werden, ein Kommando kann auf einem bestimmten Rechner ausgeführt, und die Standardausgabe kann in eine Datei auf einem bestimmten Rechner geschrieben werden.	
Bemerkungen:	Standardmäßig gilt, daß Benutzer beim Kommando uux nur das Recht zur Ausführung der Kommandos mail und mailx haben. Beim Systemverwalter ist zu erfahren, ob Benutzern auf dem eigenen System auch das Ausführbarkeitsrecht für andere Kommandos eingeräumt wurde.	

Abbildung 8-14: Übersicht über das Kommando **uux**

* Eine Beschreibung aller verfügbaren Optionen ist unter **uux(1)** im *User's Reference Manual* zu finden.

Die UNIX-Systemdateien

In diesem Anhang wird das Dateisystem zusammengefaßt dargestellt, das in Kapitel 1 ausführlich beschrieben wurde. Dabei werden die wichtigsten Systemverzeichnisse im Root-Verzeichnis beschrieben.

Struktur des Dateisystems

Die UNIX-Systemdateien sind hierarchisch geordnet; ihre Struktur wird oft mit der eines umgekehrten Baumes verglichen. An der Spitze dieses Baumes befindet sich das Root-Verzeichnis, der "Ursprung" des gesamten Dateisystems. Das Root-Verzeichnis wird durch einen Schrägstrich (/) gekennzeichnet. Alle weiteren Verzeichnisse und Dateien sind diesem untergeordnet und verzweigen von root wie in Abbildung A-1 dargestellt.

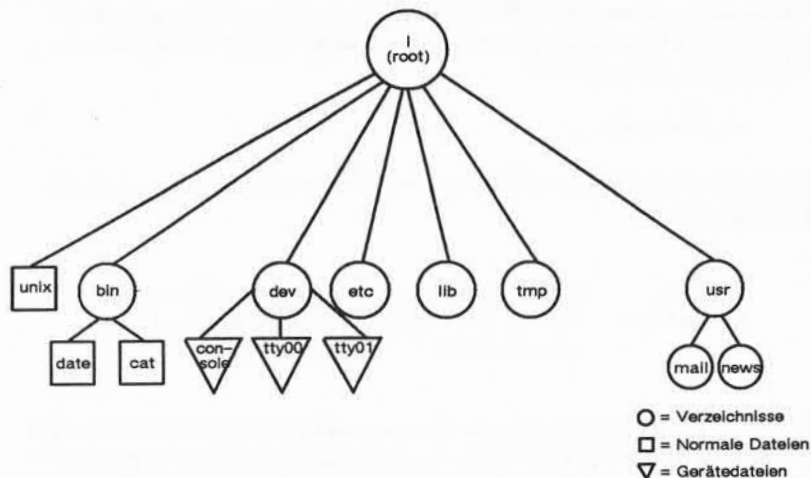


Abbildung A-1: Verzeichnisbaum, von root ausgehend

Pfad von **root** führt in das Home-Verzeichnis jedes einzelnen Benutzers. Unter seinem Home-Verzeichnis kann jeder Benutzer seine eigene Hierarchie von Verzeichnissen und Dateien aufbauen.

Weitere Pfade führen von **root** in Systemverzeichnisse, die allen Benutzern zur Verfügung stehen. Die in diesem Handbuch beschriebenen Systemverzeichnisse sind auf allen UNIX-Systemen vorhanden und werden vom Betriebssystem selbst eingerichtet und gepflegt.

Neben diesen Standardverzeichnissen können auf einem bestimmten UNIX-System noch weitere Systemverzeichnisse vorhanden sein. Mit folgender Kommandozeile erhält man eine Liste der Verzeichnisse und Dateien im Verzeichnis **root** auf dem eigenen UNIX-System:

```
ls -l / <CR>
```

Man verwendet Pfadnamen, um sich innerhalb der Dateistruktur zu bewegen, d. h. Verzeichnisse zu wechseln. Beispielsweise kann man mit folgender Kommandozeile in das Verzeichnis **/bin** wechseln, in dem ausführbare Dateien des UNIX-Systems gespeichert sind:

```
cd /bin <CR>
```

Den Inhalt eines Verzeichnisses kann man mit einer der folgenden Kommandozeilen anzeigen lassen:

ls <CR>	für eine Liste von Datei- und Verzeichnisnamen
ls -l <CR>	für eine ausführliche Liste von Datei- und Verzeichnisnamen

Den Inhalt eines Verzeichnisses, in dem man sich gerade nicht befindet, kann man mit dem Kommando **ls** nach einem der folgenden Beispiele anzeigen lassen:

ls /bin <CR>	für eine Liste in Kurzform
ls -l /bin <CR>	für eine ausführliche Liste

libc.a Im folgenden Abschnitt werden das Verzeichnis **root** und die unter ihm angeordneten Systemverzeichnisse kurz beschrieben (vgl. Abbildung A-1).

UNIX-Systemverzeichnisse

- /** Der Ursprung des Dateisystems (**Root-Verzeichnis** genannt)
- /bin** Enthält viele ausführbare Programme und Dienstprogramme, wie zum Beispiel:
- cat**
 - date**
 - login**
 - grep**
 - mkdir**
 - who**
- /lib** Enthält die zur Verfügung stehenden Programmbibliotheken und Sprachenbibliotheken, wie zum Beispiel:
- libc.a** Systemaufrufe, Standard-E/A
 - libm.a** Mathematische Routinen und Unterstützung für Programmiersprachen wie FORTRAN, C und PASCAL.
- /dev** Enthält Gerätedateien, die für Peripheriegeräte stehen, wie zum Beispiel:
- console** Konsole
 - lp** Zeilendrucker
 - ttty???** Benutzerterminal(s)
 - wd??** Laufwerke
- /etc** Enthält Programme und Datendateien zur Systemverwaltung
- /tmp** Dient zur Aufnahme temporärer Dateien, wie zum Beispiel der Puffer beim Editieren einer Datei
- /usr** Enthält folgende Unterverzeichnisse, die ihrerseits Dateien mit den angegebenen Daten enthalten:
- news** wichtige Nachrichten
 - mail** elektronische Post
 - spool**

Grundlegende UNIX-Systemkommandos

at Ausführen eines Kommandos im Hintergrundmodus zu einem in der Kommandozeile angegebenen Zeitpunkt. Wird kein Zeitpunkt angegeben, zeigt **at(1)** die Auftragsnummern aller laufenden Aufträge unter **at(1)**, **batch(1)** oder im Hintergrundmodus an.

Beispiel:

```
at 8:45am Jun 09 <CR>
Kommando1 <CR>
Kommando2 <CR>
<^d>
```

Gibt man das Kommando **at** ohne Datum an, wird das Kommando zur angegebenen Zeit innerhalb der nächsten 24 Stunden ausgeführt.

banner Eine Nachricht (in Wörtern von bis zu 10 Zeichen Länge) in großer Schrift auf der Standardausgabe angeben.

batch Ein oder mehrere Kommandos übergeben, die ausgeführt werden sollen, sobald die Systembelastung es zuläßt. Eingabe nach dem folgenden Muster:

```
batch <CR>
Kommando1 <CR>
Kommando2 <CR>
<^d>
```

Als Kommando in **batch(1)** kann auch ein Shell-Skript verwendet werden. Dies kann sinnvoll sein und viel Zeit ersparen, wenn man eine bestimmte Gruppe von Kommandos häufig mit diesem Kommando übergibt.

cat Anzeigen des Inhalts einer bestimmten Datei auf dem Terminal. Die Ausgabe auf einem ASCII-Terminal kann man vorübergehend durch **<^s>** anhalten und danach mit **<^q>** wieder weiterlaufen lassen. Zum Abbrechen der Ausgabe mit Rückkehr zur Shell ist am ASCII-Terminal die Taste **BREAK** oder **DELETE** zu drücken.

- cd** Aus dem aktuellen Verzeichnis zurück ins eigene Home-Verzeichnis wechseln. Gibt man einen Verzeichnisnamen an, wechselt man vom aktuellen Verzeichnis in das angegebene Verzeichnis. Mit einem Pfadnamen anstelle des Verzeichnisnamens kann man mit einem Kommando über mehrere Ebenen wechseln.
- cp** Kopieren einer bestimmten Datei in eine neue Datei, wobei die ursprüngliche Datei unverändert erhalten bleibt.
- cut** Bestimmte Felder aus jeder Zeile einer Datei löschen. Damit können beispielsweise Spalten einer Tabelle gelöscht werden.
- date** Aktuelles Datum und Uhrzeit anzeigen.
- diff** Zwei Dateien vergleichen. Die Ausgabe des Kommandos **diff(1)** gibt an, welche Zeilen unterschiedlich sind, und welche Änderungen vorgenommen werden müßten, bis die zweite Datei mit der ersten identisch wäre.
- echo** Eine Eingabe an der Standardausgabereinheit (dem Terminal) anzeigen, einschließlich <CR>; anschließend wird das Bereit-Zeichen angezeigt.
- ed** Eine bestimmte Datei mit dem Zeileneditor bearbeiten. Ist keine Datei mit dem angegebenen Namen vorhanden, wird von **ed(1)** eine solche angelegt. Eine ausführliche Anleitung zur Verwendung des Editors **ed(1)** ist in Kapitel 5 enthalten.
- grep** Eine oder mehrere Datei(en) nach einem bestimmten Muster durchsuchen und die Zeilen anzeigen, die das Muster enthalten. Gibt man mehrere Dateien an, zeigt **grep(1)** den Namen der Datei an, in der das Muster enthalten ist.
- kill** Einen Hintergrundprozeß mit der angegebenen Prozeßnummer (PID) abbrechen. Die PID erhält man, indem man das Kommando **ps(1)** ausführt.
- lex** Generieren von Programmen, die zur einfachen lexikalischen Analyse von Text verwendet werden sollen, beispielsweise als erster Schritt zu einem Compiler. Nähere Angaben dazu sind im *User's Reference Manual* enthalten.

- lp** Den Inhalt einer bestimmten Datei auf einem Zeilendrucker auf Papier ausdrucken.
- lpstat** Den Status der Aufträge an den Zeilendrucker anzeigen. Für ausführlichere Statusinformationen stehen Optionen zur Verfügung.
- ls** Auflisten der Namen aller Dateien und Verzeichnisse im aktuellen Verzeichnis mit Ausnahme derer, deren Name mit einem Punkt (.) beginnt. Für ausführlichere Informationen über die Dateien im Verzeichnis stehen Optionen zur Verfügung. Nähere Angaben sind unter **ls(1)** im *User's Reference Manual* zu finden.
- mail** Die eingegangene elektronische Post am Terminal anzeigen; dabei wird jeweils eine Nachricht ausgegeben. Auf jede Nachricht folgt die Eingabeaufforderung `?`; damit zeigt **mail(1)** an, daß eine Anweisung vom Benutzer erwartet wird, wie z.B. Sichern, Weiterleiten oder Löschen der Nachricht. Eine Liste der verfügbaren Optionen kann man durch Eingabe von `?` abrufen.
- Steht nach dem Kommando ein Benutzername, kann mit **mail(1)** eine Nachricht an den Eigentümer dieses Namens gesandt werden. Es können beliebig viele Zeilen eingegeben werden. Mit `<^d>` wird die Nachricht beendet und an den Empfänger abgesandt. Eine Sitzung mit "mail" kann durch Drücken der Taste **BREAK** unterbrochen werden.
- mailx** **mailx(1)** ist eine vielseitigere, erweiterte Form der elektronischen Post
- make** Große Programme oder Dokumente auf der Basis von kleineren pflegen und unterstützen. Nähere Angaben sind unter **make(1)** im *User's Reference Manual* zu finden.
- mkdir** Ein neues Verzeichnis anlegen. Das neue Verzeichnis wird zum Unterverzeichnis des Verzeichnisses, in dem man das Kommando **mkdir** eingibt. Sollen Unterverzeichnisse oder Dateien im neuen Verzeichnis angelegt werden, muß man zuerst mit dem Kommando **cd** in dieses wechseln.

- mv** Eine Datei an eine neue Stelle im Dateisystem verschieben. Man kann eine Datei in eine Datei mit einem anderen Namen in demselben Verzeichnis oder in ein anderes Verzeichnis verschieben; verschiebt man eine Datei in ein anderes Verzeichnis, kann man denselben Dateinamen oder einen neuen Dateinamen wählen.
- nohup** Die Ausführung eines Kommandos in den Hintergrund versetzen, so daß es weiter ausgeführt wird, auch nachdem man sich vom System abgemeldet hat. Fehlermeldungen werden gegebenenfalls in eine Datei mit dem Namen **nohup.out** geschrieben.
- pg** Den Inhalt einer bestimmten Datei auf dem Terminal seitenweise anzeigen. Nach jeder Seite wartet das System auf eine Anweisung des Benutzers.
- pr** Anzeigen einer teilweise formatierten Version einer bestimmten Datei auf dem Terminal. Durch das Kommando **pr(1)** werden Seitenumbrüche angezeigt, es werden jedoch keine für Textformatierprogramme eingegebenen Makros ausgeführt.
- ps** Anzeigen des Status und der Nummer aller Prozesse, die gerade laufen. Vom Kommando **ps(1)** sind Aufträge in den Warteschlangen von **at(1)** oder **batch(1)** nicht betroffen, sie werden jedoch berücksichtigt, wenn sie ausgeführt werden.
- pwd** Anzeigen des vollständigen Pfadnamens des aktuellen Arbeitsverzeichnisses.
- rm** Eine Datei aus dem Dateisystem löschen. Mit dem Kommando **rm(1)** können Maskierungszeichen verwendet werden; sie sollten jedoch nur mit Vorsicht angewandt werden, da gelöschte Dateien nicht leicht wiederhergestellt werden können.
- rmdir** Ein Verzeichnis löschen. Man kann nicht das Verzeichnis löschen, in dem man sich gerade befindet. Ebenso kann man ein Verzeichnis auch nur löschen, wenn es leer ist. Daher müssen alle Unterverzeichnisse und Dateien aus diesem Verzeichnis gelöscht werden, bevor dieses Kommando für das Verzeichnis angegeben werden kann. Eine Möglichkeit zum Löschen von Verzeichnissen, die nicht leer sind, ist unter **rm -r** im *User's Reference Manual* aufgeführt.

sort	<p>Eine Datei nach dem ASCII-Code sortieren und die Ergebnisse auf dem Terminal ausgeben. Die Reihenfolge nach dem ASCII-Code sieht folgendermaßen aus:</p> <ol style="list-style-type: none"> 1. Zahlen vor Buchstaben 2. Großschreibung vor Kleinschreibung 3. Alphabetische Reihenfolge <p>Es stehen noch weitere Optionen zum Sortieren einer Datei zur Verfügung. Eine vollständige Liste der Optionen zu sort(1) ist unter sort(1) im <i>User's Reference Manual</i> enthalten.</p>
spell	<p>Die Wörter aus einer bestimmten Datei sammeln und gegen eine Rechtschreibliste prüfen. Wörter, die nicht in der Liste enthalten sind oder keine Verwandtschaft mit diesen aufweisen (über Vorsilben, Nachsilben usw.) werden angezeigt.</p>
stty	<p>Die Werte bestimmter Eingabe-/Ausgabeoptionen für das Terminal anzeigen. Mit den entsprechenden Optionen und Argumenten werden diese Eingabe-/Ausgabeoptionen durch stty(1) auch definiert (nähere Angaben sind unter stty(1) im <i>User's Reference Manual</i> zu finden).</p>
uname	<p>Anzeigen des Namens des UNIX-Systems, auf dem man gerade arbeitet.</p>
uucp	<p>Eine bestimmte Datei auf ein anderes UNIX-System übertragen. Nähere Angaben sind unter uucp(1) im <i>User's Reference Manual</i> enthalten.</p>
uuname	<p>Die Namen der fernen UNIX-Systeme auflisten, mit denen das eigene UNIX-System in Verbindung treten kann.</p>
uupick	<p>Im öffentlichen Verzeichnis nach Dateien suchen, die mit dem Kommando uuto(1) übertragen wurden und an den Benutzer gerichtet sind. Ist eine Datei eingegangen, zeigt uupick(1) den Namen der Datei und das System an, von dem sie übertragen wurde, und fordert mit einem ? zur Eingabe einer Anweisung auf.</p>
uustat	<p>Den Status des Kommandos uuto(1) anzeigen lassen, mit dem man Dateien an einen anderen Benutzer übertragen hat.</p>

- uuto** Eine bestimmte Datei an einen anderen Benutzer übertragen. Das Ziel wird im Format *System!Benutzername* angegeben. Das *System* muß in der Liste der Systeme enthalten sein, die vom Kommando **uname(1)** erzeugt wird.
- vi** Eine bestimmte Datei mit dem Bildschirmditor **vi(1)** bearbeiten. Ist keine Datei mit dem angegebenen Namen vorhanden, legt **vi(1)** eine solche an. In Kapitel 6 ist eine ausführliche Anleitung zur Verwendung des Editors **vi(1)** enthalten.
- wc** Zählen der Anzahl von Zeilen, Wörtern und Zeichen in einer bestimmten Datei; die Ergebnisse werden auf dem Terminal angezeigt.
- who** Anzeigen der Benutzernamen der Benutzer, die gerade am UNIX-System angemeldet sind. Mitangezeigt werden die Terminaladresse für jeden Benutzernamen und die Uhrzeit, zu der sich die Benutzer angemeldet haben.
- yacc** Eine Struktur über die Eingabe eines Programms legen. Nähere Angaben dazu sind im *User's Reference Manual* enthalten.

Die ed-Kommandos

Das allgemeine Format für die Kommandos von **ed** lautet:

[Adresse1,Adresse2]Kommando[Parameter]... <CR>

Dabei werden mit *Adresse1* und *Adresse2* Zeilenadressen angegeben, mit *Parameter* die Daten, auf die das Kommando angewandt werden soll. Die Kommandos erscheinen bei der Eingabe auf dem Bildschirm. Eine umfassende Beschreibung der Verwendung der Kommandos von **ed** ist in Kapitel 5, "Anleitung zum Zeileneditor" enthalten.

Es folgt eine Übersicht über die Kommandos von **ed**. Die Kommandos sind nach ihrer Funktion geordnet.

Allgemeine Kommandos

- ed** *Dateiname* Aufrufen des Zeileneditors **ed** zur Bearbeitung einer bestimmten Datei.
- a** Text nach der aktuellen Zeile anhängen (einfügen).
- .** Texteingabemodus verlassen und in den Kommandomodus zurückkehren.
- p** Aktuelle Zeile anzeigen.
- d** Aktuelle Zeile löschen.
- <CR>** Um eine Zeile im Puffer vorwärts gehen.
- Um eine Zeile im Puffer zurückgehen.
- w** Inhalt des Puffers in der ihm gerade zugeordneten Datei abspeichern (sichern).
- q** Beenden einer Editiersitzung. Wurden Änderungen im Puffer nicht in einer Datei abgespeichert, wird eine Meldung (?) ausgegeben. Gibt man daraufhin **q** nochmals ein, wird die Sitzung ohne Speichervorgang beendet.

Kommandos zur Adressierung von Zeilen

1, 2, 3...	Angabe von Zeilenadressen im Puffer.
.	Adresse der aktuellen Zeile im Puffer.
.=	Anzeigen der Adresse der aktuellen Zeile.
\$	Die letzte Zeile im Puffer.
,	Von der ersten bis zur letzten Zeile.
;	Von der aktuellen bis zur letzten Zeile.
+x	Relative Adresse, die sich durch Addition von x zur Nummer der aktuellen Zeile ergibt.
-x	Relative Adresse, die sich durch Subtraktion von x von der Nummer der aktuellen Zeile ergibt.
/abc	Die nächste Zeile ab der aktuellen Zeile vorwärts im Puffer suchen, die das Muster <i>abc</i> enthält.
?abc	Die nächste Zeile ab der aktuellen Zeile rückwärts im Puffer suchen, die das Muster <i>abc</i> enthält.
g/abc	Alle Zeilen im Puffer, die das Muster <i>abc</i> enthalten.
v/abc	Alle Zeilen im Puffer, die das Muster <i>abc</i> nicht enthalten.

Anzeigekommandos

p	Anzeigen der angegebenen Zeilen des Puffers.
n	Anzeigen der angegebenen Zeilen mit vorangestellter Zeilenadresse und einem Tabulatorvorschub.

Texteingabe

- a Text nach der angegebenen Zeile im Puffer einfügen.
- i Text vor der angegebenen Zeile im Puffer einfügen.
- c Text in den angegebenen Zeilen durch neuen Text ersetzen.
- . Bei Eingabe alleine in einer eigenen Zeile wird damit der Texteingabemodus beendet und es erfolgt die Rückkehr in den Kommandomodus.

Text löschen

- d Löschen einer oder mehrerer Textzeilen (Kommandomodus).
- u Zuletzt eingegebenes Kommando rückgängig machen (Kommandomodus).
- @ Die aktuelle Zeile (im Texteingabemodus) oder eine Kommandozeile (im Kommandomodus) löschen.
- # oder BACKSPACE
Zuletzt eingegebenes Textzeichen löschen (im Eingabemodus).

Text ersetzen

Adresse1,Adresse2s/alt_text/neu_text/Kommando

Die Zeichenkette *neu_text* anstelle von *alt_text* in dem Zeilenbereich einsetzen, der durch *Adresse1,Adresse2* (die mit Zahlen, Symbolen oder Text angegeben werden können) eingegrenzt ist. Als *Kommando* kann *g*, *l*, *n*, *p* oder *gp* verwendet werden.

Sonderzeichen

- . Steht für ein beliebiges Einzelzeichen in einem Such- oder Ersetzungsmuster.
- * Steht für kein oder mehrfaches Auftreten des vorangestellten Zeichens in einem Such- oder Ersetzungsmuster.

[...]	Das erste Auftreten des in Klammern gesetzten Musters.
[^...]	Das erste Auftreten eines nicht in Klammern gesetzten Zeichens.
.*	Kein oder mehrfaches Auftreten beliebiger Zeichen nach dem Punkt in Such- oder Ersetzungsmustern.
^	Der Zirkumflex (^) steht in Such- oder Ersetzungsmustern für den Anfang der Zeile.
\$	Steht in Such- oder Ersetzungsmustern für das Ende der Zeile.
\	In Such- und Ersetzungsmustern die Sonderbedeutung des Sonderzeichens aufheben, das auf dieses Zeichen folgt.
&	Das letzte zu ersetzende Muster wiederholen.
%	Das letzte Ersetzungsmuster wiederholen.

Kommandos zum Verschieben/Kopieren von Text

m	Die angegebenen Textzeilen hinter die adressierte Zeile verschieben; dabei werden die Textzeilen an der ursprünglichen Stelle gelöscht.
t	Die angegebenen Textzeilen kopieren und die Kopie hinter die adressierte Zeile setzen.
j	Die aktuelle Zeile und die darauffolgende Zeile zu einer Zeile zusammenfügen.
w	Inhalt des Puffers in einer Datei schreiben (abspeichern).
r	Text aus einer anderen Datei einlesen und an den Puffer anhängen.

Weitere nützliche Kommandos und Hinweise

h	Anzeigen einer kurzen Erläuterung zur vorangegangenen Meldung ?.
H	Hilfsmodus aktivieren; dabei wird während einer Editiersitzung zu jeder Meldung ? automatisch eine Erläuterung angezeigt.

- l** Anzeigen nicht druckbarer Zeichen im Text.
- f** Anzeigen des Namens der aktuellen Datei.
- f *neudatei*** Den dem Puffer zugeordneten Namen der aktuellen Datei in *neudatei* ändern.
- !*Kommando*** Den Editor vorübergehend verlassen, um ein Shell-Kommando ausführen zu lassen.
- ed.hup** Wird die Verbindung zwischen Terminal und System vor dem Abspeichern unterbrochen, wird der Editierpuffer in der Datei *ed.hup* gesichert.



Übersicht über vi

Dieser Anhang enthält eine Zusammenfassung der Kommandos des Bildschirm-Editors vi. Die Kommandos sind nach Funktion geordnet.

Das allgemeine Format eines vi-Kommandos ist:

`[x][Kommando]Textobjekt`

x ist eine Zahl und *Textobjekt* gibt den Teil des Textes an, auf den das Kommando angewandt wird. Die Kommandos erscheinen bei der Eingabe auf dem Bildschirm. Eine Einführung in die vi-Kommandos ist in Kapitel 6, "Anleitung zum Bildschirm-Editor", zu finden.

Kommandos für die Vorbereitung des Terminals

Shell-Kommandos

TERM=Code	Schreibt einen Codenamen für das Benutzerterminal in die Variable TERM .
export TERM	Überträgt den Wert von TERM (den Terminalcode) an jedes beliebige terminalabhängige UNIX-Systemprogramm.
tput init	Initialisiert das Terminal, so daß es mit verschiedenen UNIX-Systemprogrammen arbeiten kann.

NOTE

Bevor vi verwendet werden kann, müssen die drei oben genannten Schritte abgeschlossen sein: die Variable **TERM** muß gesetzt werden, der Wert von **TERM** muß exportiert werden, und das Kommando **tput init** muß ausgeführt werden.

vi dateiname	Greift auf den Bildschirm-Editor vi zu, so daß eine angegebene Datei editiert werden kann.
---------------------	--

Grundlegende vi-Kommandos

<a>	Wechselt in den Textmodus und fügt Text nach dem Cursor an.
<ESC>	Escape-Taste; verläßt den Textmodus und kehrt in den Kommandomodus zurück.
<h>	Bewegt den Cursor um ein Zeichen nach links.
<j>	Bewegt den Cursor in derselben Spalte eine Zeile nach unten.
<k>	Bewegt den Cursor in derselben Spalte eine Zeile nach oben.
<l>	Bewegt den Cursor um ein Zeichen nach links.
<x>	Löscht das aktuelle Zeichen.
<CR>	Zeilenschaltung, RETURN-Taste; stellt den Cursor an den Anfang der nächsten Zeile.
<ZZ>	Schreibt Änderungen des Pufferinhalts in die Datei und verläßt vi.
:w	Schreibt Änderungen des Pufferinhalts in die Datei.
:q	Verläßt vi, wenn Änderungen des Pufferinhalts in eine Datei geschrieben wurden.

Kommandos zum Bewegen des Cursors in einem angezeigten Textausschnitt

Zeichenweise springen

<h>	Bewegt den Cursor um ein Zeichen nach links.
<BACKSPACE>	Backspace-Taste; bewegt den Cursor um ein Zeichen nach links.
<l>	Bewegt den Cursor um ein Zeichen nach rechts.

<Leertaste>	Bewegt den Cursor um ein Zeichen nach rechts.
<fx>	Bewegt den Cursor nach rechts auf das angegebenen Zeichen <i>x</i> .
<Fx>	Bewegt den Cursor nach links auf das angegebene Zeichen <i>x</i> .
<tx>	Bewegt den Cursor nach rechts auf das Zeichen vor dem angegebenen Zeichen <i>x</i> .
<Tx>	Bewegt den Cursor nach links auf das Zeichen nach dem angegebenen Zeichen <i>x</i> .
<;>	Setzt die Suche nach dem mit den Kommandos <f>, <F>, <t> oder <T> angegebenen Zeichen fort. Mit dem Kommando ; wird das angegebene Zeichen gespeichert und das nächste Auftreten dieses Zeichens in der aktuellen Zeile gesucht.
<, >	Setzt die Suche nach dem mit den Kommandos <f>, <F>, <t> oder <T> angegebenen Zeichen fort. Mit dem Zeichen , wird das angegebene Zeichen gespeichert und das nächste Auftreten dieses Zeichens in der aktuellen Zeile gesucht.

Zeilenweise springen

<j>	Bewegt den Cursor in derselben Spalte um eine Zeile nach unten.
<k>	Bewegt den Cursor in derselben Spalte um eine Zeile nach oben.
<+>	Bewegt den Cursor nach unten an den Anfang der nächsten Zeile.
<CR>	Zeilenschaltung, RETURN-Taste; bewegt den Cursor nach unten an den Anfang der nächsten Zeilen.
<->	Bewegt den Cursor nach oben an den Anfang der nächsten Zeile.

Wortweise springen

- <w> Bewegt den Cursor nach rechts auf das erste Zeichen des nächsten Wortes.
- Bewegt den Cursor zurück auf das erste Zeichen des vorigen Wortes.
- <e> Bewegt den Cursor an das Ende des aktuellen Wortes.

Satzweise springen

- <(> Bewegt den Cursor an den Anfang des Satzes.
- <)> Bewegt den Cursor an den Anfang des nächsten Satzes.

Absatzweise springen

- <{> Bewegt den Cursor an den Anfang des Absatzes.
- <}> Bewegt den Cursor an den Anfang des nächsten Absatzes.

Positionieren auf dem Bildschirm

- <H> Bewegt den Cursor in die erste Zeile auf dem Bildschirm.
- <M> Bewegt den Cursor in die mittlere Zeile auf dem Bildschirm.
- <L> Bewegt den Cursor in die letzte Zeile auf dem Bildschirm.

Kommandos zum Positionieren in der Datei

Blättern

- | | |
|-------------------------|--|
| <code><^f></code> | Blättert den Text um einen ganzen Bildschirm vorwärts, so daß der Text unter dem aktuellen Textausschnitt angezeigt wird. |
| <code><^d></code> | Blättert den Text um einen halben Bildschirm vorwärts, so daß einige Textzeilen unter dem aktuellen Textausschnitt angezeigt werden. |
| <code><^b></code> | Blättert den Text um einen ganzen Bildschirm zurück, so daß der Text über dem aktuellen Textausschnitt angezeigt wird. |
| <code><^u></code> | Blättert den Text um einen halben Bildschirm zurück, so daß einige Textzeilen über dem aktuellen Textausschnitt angezeigt werden. |

Positionieren in eine nummerierte Zeile

- | | |
|-------------------------|--|
| <code><G></code> | Bewegt den Cursor an den Anfang der letzten Zeile im Puffer. |
| <code><nG></code> | Bewegt den Cursor an den Anfang der <i>n</i> ten Zeile der Datei (<i>n</i> = Zeilennummer). |

Suchen nach einem Muster

- | | |
|------------------------|--|
| <code>/Muster</code> | Sucht im Puffer vorwärts nach dem nächsten Auftreten des <i>Musters</i> im Text. Bewegt den Cursor unter das erste Zeichen des <i>Musters</i> . |
| <code>?Muster</code> | Sucht im Puffer rückwärts nach dem nächsten Auftreten des <i>Musters</i> im Text. Bewegt den Cursor unter das erste Zeichen des <i>Musters</i> . |
| <code><n></code> | Wiederholt das letzte Suchkommando. |
| <code><N></code> | Wiederholt das Suchkommando in die entgegengesetzte Richtung. |

Kommandos zum Einfügen von Text

- <a>** Wechselt in den Textmodus und fügt nach dem Cursor Text an.
- <I>** Wechselt in den Textmodus und fügt vor dem Cursor Text ein.
- <o>** Wechselt durch das Öffnen einer neuen Zeile direkt unter der aktuellen Zeile in den Textmodus.
- <O>** Wechselt durch das Öffnen einer neuen Zeile direkt über der aktuellen Zeile in den Textmodus.
- <ESC>** Escape-Taste; kehrt vom Textmodus in den Kommando-Modus zurück (wird nach einem der oben genannten Kommandos eingegeben).

Kommandos zum Löschen von Text

Im Textmodus

- <BACKSPACE>** Backspace-Taste; löscht das aktuelle Zeichen.
- <^w>** Löscht das aktuelle durch Leerzeichen begrenzte Wort.
- <@>** Löscht die aktuelle Textzeile.

In Kommandomodus

- <x>** Löscht das aktuelle Zeichen.
- <dw>** Löscht ein Wort (oder einen Teil eines Wortes) vom Cursor an bis zum nächsten Leerzeichen oder Satzzeichen.
- <dd>** Löscht die aktuelle Zeile.

- <ndr>** Löscht n Textobjekte vom Typ x , wobei x ein Wort, eine Zeile, ein Satz oder ein Absatz sein kann.
- <D>** Löscht die aktuelle Zeile vom Cursor bis zum Ende der Zeile.

Kommandos zum Ändern von Text

Zeichen, Wörter, Textobjekte

- <r>** Ersetzt das aktuelle Zeichen.
- <s>** Löscht das aktuelle Zeichen und fügt Text an, bis das Kommando **<ESC>** eingegeben wird.
- <S>** Ersetzt alle Zeichen in der aktuellen Zeile.
- <~>** Ändert Groß- in Kleinschreibung oder Klein- in Großschreibung.
- <cw>** Ersetzt das aktuelle Wort oder die übrigen Zeichen des aktuellen Wortes mit neuem Text, und zwar von der Cursor-Position bis zum nächsten Leer- oder Satzzeichen.
- <cc>** Ersetzt alle Zeichen in der aktuellen Zeile.
- <nex>** Ersetzt n Textobjekte des Typs x , wobei x ein Wort, eine Zeile, ein Satz oder ein Absatz sein kann.
- <C>** Ersetzt die übrigen Zeichen der aktuellen Zeile von der Cursor-Position bis zum Ende der Zeile.

Verschieben von Text

- <p>** Fügt den Inhalt des temporären Puffers (der die Ausgabedaten des letzten Lösch- oder Kopierkommandos enthält) nach der Cursor-Position oder unterhalb der aktuellen Zeile in den Text ein.

- <yy> Schreibt eine angegebene Textzeile in einen temporären Puffer.
- <n!yx> Schreibt eine Kopie von *n* Textobjekten des Typs *x* in einen temporären Puffer.
- <"/yx> Schreibt eine Kopie des Textobjektes *x* in ein mit dem Buchstaben *l* bezeichnetes Register. *x* kann ein Wort, eine Zeile, ein Satz oder ein Absatz sein.
- <"xp> Fügt den Inhalt des Registers *x* nach der Cursor-Position oder unterhalb der aktuellen Zeile in den Text ein.

Andere Kommandos

Spezielle Kommandos

- <^g> Gibt die Zeilennummer der aktuellen Cursor-Position im Puffer und den Änderungsstatus der Datei aus.
- <.> Wiederholt das letzte Kommando.
- <u> Macht das letzte Kommando rückgängig.
- <U> Stellt den Zustand der aktuellen Zeile vor den letzten Änderungen wieder her.
- <J> Fügt die Zeile direkt unter der aktuellen Zeile mit der aktuellen Zeile zusammen.
- <^l> Löscht das aktuelle Fenster und baut es neu auf.

Kommandos des Zeileneditors

- : Teilt vi mit, daß das nächste Benutzerkommando ein Kommando des Zeileneditors ist.
- :sh Bewirkt die zeitweilige Rückkehr zur Shell, um Shell-Kommandos auszuführen, ohne vi zu verlassen.
- <^d> Kehrt von der Shell zum Editor vi zurück, so daß der aktuelle Textausschnitt editiert werden kann.

<i>n</i>	Sprung zur <i>n</i> ten Zeile des Puffers.
<i>x,zw dateiname</i>	Schreibt die Zeilen <i>x</i> bis <i>z</i> in eine neue Datei mit dem Namen <i>dateiname</i> .
<i>:\$</i>	Bewegt den Cursor an den Anfang der letzten Zeile im Puffer.
<i>:\$d</i>	Löscht alle Zeilen von der aktuellen Zeile bis zur letzten Zeile.
<i>:r dateiname</i>	Fügt den Inhalt der Datei <i>dateiname</i> unterhalb der aktuellen Zeile des Puffers ein.
<i>:s/Text/Neuer_Text/</i>	Ersetzt das erste Auftreten von <i>Text</i> in der aktuellen Zeile durch <i>Neuer_Text</i> .
<i>:s/Text/Neuer_Text/g</i>	Ersetzt jedes Auftreten von <i>Text</i> in der aktuellen Zeile durch <i>Neuer_Text</i> .
<i>:g/Text/s//Neuer_Text /g</i>	Ersetzt jedes Auftreten von <i>Text</i> im Puffer durch <i>Neuer_Text</i> .

Kommandos zum Verlassen von vi

<ZZ>

<i>:wq</i>	Schreibt den Pufferinhalt in die Datei und verläßt vi.
<i>:w dateiname</i>	Schreibt den Pufferinhalt in die neue Datei <i>dateiname</i> und verläßt vi.
<i>:w! dateiname</i>	Überschreibt die bestehende Datei <i>dateiname</i> mit dem Inhalt des Puffers und verläßt vi.
<i>:q!</i>	Verläßt vi, egal ob die Änderungen des Pufferinhalts in eine Datei geschrieben wurden oder nicht. Berücksichtigt die Änderungen des Pufferinhalts seit dem letzten Schreibkommando (:w) nicht.

:q Verläßt vi, wenn die Änderungen des Pufferinhalts in eine Datei geschrieben wurden.

Sonderoptionen für vi

vi *datei1 datei2 datei3*

Stellt drei Dateien, *datei1*, *datei2* und *datei3*, zum Editieren in den vi-Puffer.

:w

:n

Wurde mehr als eine Datei in einer einzigen vi-Kommandozeile aufgerufen, wird hiermit der Puffer in die Datei geschrieben, die gerade editiert wird, und dann wird die nächste Datei in den Puffer gestellt. (:n nur nach :w verwenden).

vi -r *datei1*

Stellt die Änderungen der Datei *datei1* wieder her, wenn diese durch eine Systemunterbrechung verloren gegangen sind.

vlew *datei1*

Zeigt *datei1* im Nur-Lese-Modus von vi an. Änderungen, die am Pufferinhalt vorgenommen werden, dürfen nicht in die Datei geschrieben werden.

Übersicht über die Shell-Kommandosprache

In diesem Abschnitt werden die Shell-Kommandosprache und die Programmstrukturen zusammengefaßt, die in Kapitel 7, "Anleitung zur Shell", beschrieben wurden. Im ersten Abschnitt werden die Maskierungszeichen, Sonderzeichen, die Umlenkung von Ein- und Ausgaben sowie Variablen und Prozesse aufgeführt. Sie sind thematisch nach der Reihenfolge geordnet, in der sie in Kapitel 7 erscheinen. Im zweiten Abschnitt werden Shell-Programmstrukturen dargestellt.

Das Vokabular der Shell-Kommandosprache

Sonderzeichen der Shell

- * ? [] . Maskierungszeichen; sie werden als Abkürzungen zur Angabe von Dateinamen mittels Zeichenmustern verwendet.
- & Ausführen von Kommandos im Hintergrundmodus.
- ; Ausführen von mehreren Kommandos in Folge, die in derselben Kommandozeile eingegeben werden; dabei werden die Kommandos durch ; getrennt.
- \ Aufheben der Sonderbedeutung des darauffolgenden Sonderzeichens.
- '... ' Mit einfachen Anführungszeichen werden die Sonderbedeutungen aller darin eingeschlossenen Sonderzeichen aufgehoben.
- "..." Mit doppelten Anführungszeichen werden die Sonderbedeutungen aller darin enthaltenen Sonderzeichen mit Ausnahme von \$ und ` aufgehoben.

Umlenkung von Eingabe und Ausgabe

- < Umlenken des Inhalts einer Datei in ein Kommando.
- > Umlenken der Ausgabe eines Kommandos in eine neue Datei, bzw. Ersetzen des Inhalts einer bestehenden Datei durch die Ausgabe.



- >> Umlenken der Ausgabe eines Kommandos in eine Datei, wobei die Ausgabe an das Ende der Datei angehängt wird.
- | Umlenken der Ausgabe eines Kommandos, so daß sie direkt als Eingabe für das nächste Kommando dient.
- Kommando* Einsetzen der Ausgabe des in Ausführungszeichen gesetzten Kommandos anstelle des *Kommandos*.

Aufrufen und Beenden von Prozessen

- batch** Die danach angegebenen Kommandos werden als Stapeljob an das System übergeben und ausgeführt, sobald die Systembelastung dies zuläßt. Mit <^d> wird das Kommando **batch** abgebrochen.
- at** Die danach angegebenen Kommandos werden zu einem bestimmten Zeitpunkt ausgeführt. Mit <^d> wird das Kommando **at** abgebrochen.
- at -l** Anzeigen, welche Aufträge gerade in der Warteschlange von **at** oder **batch** vorgemerkt sind.
- at -r** Löschen eines **at**- oder **batch**-Auftrags aus der Warteschlange.
- ps** Anzeigen des Status der Shell-Prozesse.
- kill PID** Abbrechen des Shell-Prozesses mit der angegebenen Prozeßnummer (PID).
- nohup** *Kommandoliste &*
Weiterführen von Hintergrundprozessen nach dem Abmelden.

Shell-Programmdateien

- chmod u+x** *Dateiname*
Damit erhält der Benutzer das Ausführbarkeitsrecht für die angegebene Datei (nützlich bei Shell-Programmdateien).
- mv** *Dateiname* **\$HOME/bin/***Dateiname*
Die angegebene Datei wird in das Verzeichnis **bin** des Home-Verzeichnis des Benutzers verschoben. Dieses Verzeichnis **bin** enthält ausführbare Shell-Programme, die stets zur Verfügung stehen sollen. Dabei ist sicherzustellen, daß die Variable **PATH** in der Datei **.profile** die Angabe **bin** enthält, damit die Shell im

Verzeichnis `$HOME/bin` nach einer Datei sucht, die man ausführen will. Enthält die Variable `PATH` diese Angabe nicht, kann die Shell die Datei nicht finden und daher auch nicht ausführen.

Dateiname Der Name einer Datei, die ein Shell-Programm enthält, ist gleichzeitig das Kommando, das einzugeben ist, um dieses Shell-Programm auszuführen.

Variablen

Positionsparameter

Eine numerierte Variable in einem Shell-Programm, mit der auf Werte verwiesen wird; diese Werte werden durch die Shell automatisch auf der Grundlage der Argumente in der Kommandozeile für das Shell-Programm zugewiesen.

echo Mit diesem Kommando kann der Wert einer Variablen auf dem Terminal angezeigt werden.

\$# Ein Parameter, der die Anzahl der Argumente angibt, mit denen das Shell-Programm ausgeführt wurde.

\$* Ein Parameter, der die Werte aller Argumente angibt, mit denen das Shell-Programm ausgeführt wurde.

benannte Variable

Eine Variable, der vom Benutzer ein Name und Werte zugewiesen werden können.

Systemvariablen

HOME Damit wird das Home-Verzeichnis des Benutzers angegeben; der Standardwert für das Kommando `cd`.

PATH Mit dieser Variablen wird der Pfad definiert, auf dem die Benutzer-Shell nach den Programmen/Dateien zu eingetragenen Kommandos sucht.

CDPATH Zur Definition des Suchpfades für das Kommando `cd`.

MAIL Zur Definition der Datei, in der die elektronische Post des Benutzers gespeichert wird.

PS1 PS2	Zur Definition der primären und sekundären Zeichenketten für das Bereit-Zeichen.
TERM	Zur Definition des Terminaltyps.
LOGNAME	Name, mit dem sich der Benutzer anmeldet (Benutzername).
IFS	Zur Definition der internen Feldtrennzeichen (normalerweise Leerzeichen, Tabulator und Carriage-Return).
TERMINFO	Damit kann festgelegt werden, daß die Unterroutinen <i>curses</i> und <i>terminfo</i> zuerst in einem bestimmten Verzeichnispfad nach dem Terminaltyp des Benutzers suchen, und dann erst im Standardverzeichnis.
TZ	Zur Definition der Ortszeit.

Shell-Programmstrukturen

Kommando <<!
Eingabezeilen
!

Schleife mit For

```
for Variable <CR>
  in dieser Werteliste <CR>
do folgende Kommandos ausführen <CR>
  Kommando 1 <CR>
  Kommando 2 <CR>
  .<CR>
  .<CR>
  letztes Kommando <CR>
done <CR>
```

Schleife mit While

```
while Kommandoliste <CR>
do (ausführen:) <CR>
  Kommando1 <CR>
  Kommando2 <CR>
  .<CR>
  .<CR>
  letztes Kommando <CR>
done <CR>
```

Bedingung If...Then

```
if Kommando A (ausgeführt wird)<CR>  
then Kommando1<CR>  
    Kommando2<CR>  
        .<CR>  
        .<CR>  
    letztes Kommando<CR>  
fi<CR>
```

Bedingung If...Then...Else

```
if Kommandoliste<CR>  
    then Kommandoliste<CR>  
    else Kommandoliste<CR>  
fi<CR>
```

Programmstruktur Case

```
case Wort <CR>
in <CR>
  Muster1) <CR>
    Kommandozeile 1 <CR>
      .<CR>
      .<CR>
    letzte Kommandozeile <CR>
  ;;<CR>
  Muster2) <CR>
    Kommandozeile 1 <CR>
      .<CR>
      .<CR>
    letzte Kommandozeile <CR>
  ;;<CR>
  Muster3) <CR>
    Kommandozeile 1 <CR>
      .<CR>
      .<CR>
    letzte Kommandozeile <CR>
  ;;<CR>
esac <CR>
```

Anweisungen break und continue

Durch die Anweisungen "break" oder "continue" verläßt das Programm eine Schleife und führt das Kommando aus, das unmittelbar auf die Schleife folgt.

Definition der Variablen TERM

Nixdorf unterstützt die Verwendung vieler Terminaltypen mit dem Betriebssystem UNIX. Da die Ausführung einiger der Kommandos vom Terminal abhängt, muß dem System bei jeder Anmeldung mitgeteilt werden, welchen Terminaltyp der Benutzer verwendet. Das System bestimmt dann die Eigenschaften des Terminals, indem es den Wert der Variablen **TERM** abfragt; diese Variable enthält den Namen eines Terminals. Gibt man als Wert für diese Variable den Namen des verwendeten Terminals an, kann das System alle Programme zufriedenstellend mit diesem Terminal ausführen.

Diese Form der Angabe des Terminaltyps gegenüber dem UNIX-System nennt man die Konfiguration des Terminals. Mit der Kommandozeile des folgenden Bildschirms wird das Terminal konfiguriert; dabei ist für *Terminalname* der Name des verwendeten Terminals einzusetzen.

```
$ TERM=Terminalname<CR>
$ export TERM<CR>
$ tput init<CR>
```

Diese Zeilen müssen in der angegebenen Reihenfolge eingegeben werden, da sie sonst nicht verarbeitet werden. Dieser Vorgang muß auch bei jeder Anmeldung wiederholt werden. Daher schreibt man diese Zeilen in der Regel in eine Datei mit dem Namen *.profile*, die automatisch bei jedem Anmelden ausgeführt wird. Nähere Angaben zur Datei *.profile* sind in Kapitel 7 zu finden.

Mit den ersten beiden Zeilen dieses Bildschirms wird der Shell des UNIX-Systems mitgeteilt, welcher Terminaltyp verwendet wird. Mit der Kommandozeile *tput init* wird das Terminal angewiesen, sich so zu verhalten, wie das UNIX-System es von einem Terminal dieses Typs erwartet; beispielsweise werden der linke Rand und die Tabulatoren eingestellt, falls diese Charakteristika am Terminal eingestellt werden können.

Das Kommando **tput** greift auf den Eintrag in der Datenbasis für das Terminal zu, um der Shell terminalabhängige Funktionen und Informationen zur Verfügung zu stellen. Da sich die Werte dieser Funktionen bei den einzelnen Terminaltypen unterscheiden, muß die Kommandozeile **tput init** jedesmal ausgeführt werden, wenn die Variable **TERM** geändert wird.

Für jeden Terminaltyp ist eine Reihe von Funktionen in der Datenbasis definiert. Diese Datenbasis befindet sich in **/usr/lib/terminfo**.

In den folgenden Abschnitten wird beschrieben, wie man feststellen kann, welche *Terminalnamen* zulässig sind. Weitere Informationen über die Funktionen in der Datenbasis **terminfo** sind unter **terminfo(4)** im *Programmer's Reference Manual* zu finden.

Zulässige Terminalnamen

Das UNIX-System erkennt eine Vielzahl von Terminaltypen. Bevor man der Variablen **TERM** einen Terminalnamen zuweist, ist sicherzustellen, daß dieser Name zulässig ist.

Es gibt in der Regel mindestens zwei zulässige Namen für ein Terminal: den Namen des Herstellers und die Modellnummer. Diese Namen können jedoch auf mehrere Arten dargestellt werden: durch Variationen in der Groß- und Kleinschreibung, durch Abkürzungen usw. Der Variablen **TERM** darf erst dann ein Name zugewiesen werden, wenn geprüft wurde, ob das System ihn erkennt.

Mit dem Kommando **tput** kann man sehr schnell feststellen, ob das Terminal vom System unterstützt wird. Dazu eingeben:

```
tput -TTerminalname longname <CR>
```

Wird der angegebene Terminaltyp unterstützt, gibt das System den vollen Namen des Terminals aus; andernfalls erscheint eine Fehlermeldung.

Einen zulässigen Namen für das Terminal in der Variablen **TERM** kann man herausfinden, indem man einen Eintrag für das betreffende Terminal in folgendem Verzeichnis sucht: **/usr/lib/terminfo**. Diese Verzeichnisse enthalten eine Reihe von Dateien mit Namen, die nur aus einem Buchstaben bestehen. Jede Datei enthält wiederum eine Liste von Terminalnamen, die mit dem Buchstaben der betreffenden Datei beginnen. Dieser Name kann ein Buchstabe sein, wie beispielsweise der Anfangsbuchstabe **d** in **dap4x**. In diesem Verzeichnis ist nach der Datei zu suchen, deren Name mit dem ersten Buchstaben des Terminalnamens

übereinstimmt. Dann ist der Inhalt der Datei aufzulisten und nach dem gewünschten Terminalnamen zu suchen.

Auch beim Systemverwalter kann man erfahren, welche Terminals vom betreffenden System unterstützt werden, und welche Namen für die Variable **TERM** zulässig sind.

Beispiel

In diesem Beispiel wird von folgenden Annahmen ausgegangen: Es handelt sich um ein Terminal des Typs Nixdorf BA47/BA48, der eigene Benutzername lautet **jakob**, und der Benutzer befindet sich gerade in seinem Home-Verzeichnis. Zuerst ist zu überprüfen, ob das System das Terminal unterstützt, indem man `ls /usr/lib/term` nach dem Kommando `tput` ausführt. Auf dem folgenden Bildschirm wird dargestellt, welche Kommandos dafür erforderlich sind:

```
$ tput dap4x longname <CR>
Nixdorf BA47
$ cd /usr/lib/terminfo/b <CR>
$ ls
ba47
ba48
ba80
$
```

Definition der Variablen TERM

Nun kann der gefundene Name, `dap4x`, der Variablen `TERM` zugewiesen werden. Dabei muß in jedem Fall auch die Variable `TERM` exportiert und `tput init` ausgeführt werden.

```
$ TERM=dap4x<CR>
$ export TERM<CR>
$
```

Damit ist dem UNIX-System bekannt, welchen Terminaltyp der Benutzer verwendet, und es kann daher die eingegebenen Kommandos ordnungsgemäß ausführen.

Glossar

abmelden

Das Verfahren, mit dem der Benutzer das Betriebssystem UNIX verläßt.

Adresse

Allgemein eine Nummer, die die Stelle angibt, an der sich eine Information im Rechnerspeicher befindet. Im UNIX-System ist eine Adresse ein Teil eines Editierkommandos, mit dem eine Zeilennummer oder ein Bereich von Zeilennummern angegeben wird.

aktuelles Verzeichnis

Das Verzeichnis, in dem man gerade arbeitet. Man hat jeweils direkten Zugriff auf die Dateien und Unterverzeichnisse, die im aktuellen Verzeichnis enthalten sind. In Kurzform wird das aktuelle Verzeichnis in bestimmten Situationen mit einem Punkt (.) angegeben.

Akustikkoppler

Ein Gerät zur Datenübertragung über eine Telefonleitung. Der Telefonhörer wird in den Akustikkoppler eingelegt, um einen Rechner an einem Ende der Leitung mit einem Peripheriegerät wie einem Terminal am anderen Ende der Leitung über diese Leitung zu verbinden.

Anmeldung

Das Verfahren, durch das der Benutzer Zugriff auf das Betriebssystem UNIX erhält.

Arbeitsverzeichnis

Siehe **aktuelles Verzeichnis**.

Argument

Das Element einer Kommandozeile, mit dem die Daten angegeben werden, auf die ein Kommando angewandt werden soll. Argumente stehen nach dem Kommando und können aus Zahlen, Buchstaben oder Zeichenketten bestehen. Im Kommando `lp -m meinedatei` ist beispielsweise `lp` das Kommando und `meinedatei` das Argument. Vgl. **Option**.

ASCII

(Aussprache: as'-kii) "American Standard Code for Information Interchange", ein amerikanischer Standardcode für den Datenaustausch zwischen Systemen, der im UNIX-System verwendet wird. Im ASCII-Zeichensatz werden 128 Zeichen, darunter die alphabetischen Zeichen, Ziffern und Sonderzeichen wie #, \$, % und &, als Binärwerte dargestellt.

ausführbare Datei

Eine Datei, die ohne weiteren Übersetzungsvorgang vom Rechner ausgeführt bzw. verarbeitet werden kann. Nach Eingabe des Dateinamens werden die Kommandos in der Datei ausgeführt. Siehe auch **Shell-Prozedur**.

ausführen

Der Rechner führt ein Programm bzw. ein Kommando aus, indem er die darin enthaltenen Anweisungen verarbeitet und die gewünschten Operationen vornimmt.

Ausgabe

Informationen, die in einer bestimmten Form von einem Rechner verarbeitet wurden und dem Benutzer über einen Drucker, ein Terminal oder eine ähnliche Einheit ausgegeben werden.

Baudrate

Eine Maßeinheit für die Geschwindigkeit der Datenübertragung von einem Rechner zu einem Peripheriegerät (wie z. B. einem Terminal) oder von einer Einheit zu einer anderen. Verbreitete Baudraten sind 300, 1200, 4800 und 9600. Als Faustregel kann gelten, daß die Baudrate geteilt durch 10 die ungefähre Anzahl der Schriftzeichen ergibt, die pro Sekunde übertragen werden.

Begrenzungszeichen

Ein Zeichen, mit dem Wörter oder Argumente in einer Kommandozeile logisch voneinander getrennt werden. Zwei im UNIX-System häufig verwendete Begrenzungszeichen sind das Leerzeichen und das Tabulatorzeichen.

benutzerdefinierte Variable

Eine benannte Variable, der vom Benutzer ein Wert zugewiesen wird. Siehe auch **Variable**.

benutzerdefiniert

Vom Benutzer festgelegt.

Benutzername

Eine Zeichenkette, mit der ein Benutzer identifiziert wird. Der Benutzername unterscheidet sich von den Benutzernamen anderer Benutzer.

Benutzerverzeichnis

Siehe **Home-Verzeichnis**.

Benutzer

Jeder, der einen Rechner oder ein Betriebssystem verwendet.

Bereit-Zeichen

Ein Zeichen, das von der Shell auf dem Terminal ausgegeben wird, um anzuzeigen, daß die Shell für die nächste Eingabe bereit ist. Das Bereit-Zeichen kann ein einzelnes oder eine Folge von Zeichen sein. Das Standardzeichen dafür im Betriebssystem UNIX ist das Dollarzeichen (\$).

Betriebssystem

Das Softwaresystem auf einem Rechner, unter dem alle anderen Programme laufen. UNIX ist ein Betriebssystem.

Bildschirmeditor

Ein Programm zur Textbearbeitung, in dem Text in Abhängigkeit von der Cursorposition auf einem Bildschirm bearbeitet wird. Dabei wird der Cursor an die Stelle bewegt, an der Text eingegeben, geändert oder gelöscht werden soll. Die vorgenommenen Änderungen erscheinen dabei sofort auf dem Bildschirm. Vgl. **Texteditor** und **Zelleneditor**.

Bildschirmterminal

Ein Terminal, bei dem ein Bildschirm ähnlich dem eines Fernsehgerätes (ein Monitor) dazu dient, Informationen anzuzeigen. Auf einem Bildschirmterminal können die Informationen viel schneller ausgegeben werden als auf einem Druckerterminal.

Cursor

Ein Zeichen auf dem Terminal, mit dem die Position angezeigt wird, an der man beispielsweise ein Zeichen eingeben oder löschen kann. Es hat normalerweise die Form eines Rechtecks oder eines blinkenden Unterstrichs.



Dateiname

Eine Folge von Zeichen, mit der eine Datei bezeichnet wird. Im Betriebssystem UNIX kann der Schrägstrich (/) nicht als Teil eines Dateinamens verwendet werden.

Dateisystem

Eine Sammlung von Dateien und die Strukturen, mit denen sie verknüpft werden. Das UNIX-Dateisystem weist eine hierarchische Struktur auf. Nähere Angaben sind in Anhang A, "Übersicht über das Dateisystem", zu finden.

Datei

Eine Sammlung von Informationen in Form einer Zeichenfolge. Dateien können Daten, Programme oder andere Texte enthalten. Dateien im UNIX-System werden durch ihren Namen adressiert. Siehe **normale Datei**, **permanente Datei** und **ausführbare Datei**.

Diagnosemeldung

Eine Meldung, die auf dem Terminal erscheint, um darauf hinzuweisen, daß bei der Ausführung eines Kommandos bzw. Programms ein Fehler aufgetreten ist. Im allgemeinen braucht man auf eine solche Diagnosemeldung nicht direkt zu reagieren.

Dienstprogramm

Ein Programm, mit dem Routineaufgaben erledigt werden, oder das einen Programmierer oder Benutzer des Systems bei der Erstellung von Programmen für Routineaufgaben unterstützt.

Drucker

Ein Ausgabegerät, mit dem die Daten, die vom Rechner eingehen, auf Papier ausgedruckt werden.

Einfügemodus

Ein Modus der Textbearbeitung, in dem die eingegebenen Zeichen als Textzeichen in den Puffer des Editors eingegeben werden. In diesem Modus wird der Text vor der aktuellen Position im Puffer eingefügt. Vgl. **Texteingabemodus**, **Modus "Anhängen"** und **Kommandomodus**.

Eingabe/Ausgabe

Der Weg, über den Informationen in ein Rechnersystem gelangen (Eingabe) bzw. das Rechnersystem verlassen (Ausgabe). Eine Eingabeinheit ist beispielsweise die Terminaltastatur, eine Ausgabeinheit das Terminal.

elektronische Post

Eine Einrichtung eines Betriebssystems, durch die Benutzer von Rechnern über diese Rechner schriftliche Nachrichten austauschen können. Im UNIX-System dient das Kommando **mail** zur Übermittlung elektronischer Post, in der die Benutzernamen für die Anmeldung als Adressen dienen.

Elternverzeichnis

Das Verzeichnis unmittelbar über einem Unterverzeichnis oder einer Datei in der Struktur des Dateisystems. Kurzform für die Angabe des Elternverzeichnisses sind zwei Punkte (..).

Endausdruck

Die vollständige, ausgedruckte Version einer Textdatei.

Erase-Zeichen

Das Zeichen, mit dem das zuvor eingegebene Zeichen gelöscht wird. Der Standardwert im UNIX-System für das Erase-Zeichen ist #; viele Benutzer legen dieses Löschrzeichen auf die BACKSPACE-Taste.

Escape

Ein Mittel zur Rückkehr zur Shell aus einem Texteditor oder einem anderen Programm.

fernes System (remote System)

Ein anderes System als das, an dem man gerade arbeitet.

Festplatte

Ein Gerät zur magnetischen Speicherung von Daten, das aus mehreren runden Platten besteht, die in gewisser Weise Schallplatten ähneln. Auf solchen Platten können große Datenmengen gespeichert werden, und sie ermöglichen den schnellen Zugriff auf jede Informationseinheit.

Filter

Ein Kommando, das die Standardeingabe liest, sie in bestimmter Form bearbeitet und das Ergebnis als Standardausgabe ausgibt.

Gerätedatei

Eine Datei, die als Schnittstelle zu einer Ein-/Ausgabeeinheit wie zum Beispiel einem Benutzerterminal, einem Plattenlaufwerk oder einem Zeilendrucker dient; auch Einheitsreiber genannt.

global

Bezieht sich auf die gesamte Datei. Normale Editierkommandos gelten nur für das jeweils erste Auftreten eines Musters in der Datei, globale Kommandos führen die betreffende Operation bei jedem Auftreten eines Musters in einer Datei durch.

Hardware

Die physikalischen Geräte eines Rechnersystems und der damit verbundenen Einheiten.

Hintergrundmodus

Eine Form der Programmausführung, bei der die Shell angewiesen wird, ein Kommando unabhängig von einer Interaktion zwischen dem Benutzer und dem Rechner (d. h. "im Hintergrund") auszuführen. Nachdem dieses Kommando aufgerufen wurde, fordert die Shell den Benutzer zur Eingabe anderer Kommandos über das Terminal auf.

Home-Verzeichnis

Das Verzeichnis, in dem man sich sofort nach dem Anmelden am UNIX-System befindet; auch Benutzer- bzw. Anmeldeverzeichnis genannt.

interaktiv

Eine Methode der Interaktion zwischen dem Benutzer und dem Betriebssystem, die in Form eines Dialoges abläuft. Das Betriebssystem UNIX ist ein interaktives Betriebssystem.

Kindverzeichnis

Siehe Unterverzeichnis.

Kommandodatei

Siehe ausführbare Datei.

Kommandointerpreter

Ein Programm, das als direkte Schnittstelle zwischen dem Benutzer und dem Rechner fungiert. Im Betriebssystem UNIX nimmt ein Programm, das Shell genannt wird, die Kommandos entgegen und übersetzt sie in eine Sprache, die der Rechner versteht.

Kommandomodus

Ein Modus der Textbearbeitung, in dem die eingegebenen Zeichen als Editierkommandos interpretiert werden. In diesem Modus kann man sich beispielsweise im Puffer vor- und zurückbewegen, Text löschen, Textzeilen kopieren oder verschieben. Vgl. **Texteingabemodus**, **Modus "Anhängen"** und **Einfügemodus**.

Kommandozeile

Eine Zeile, die ein oder mehrere Kommandos enthält; sie wird durch ein Carriage-Return (<CR>) abgeschlossen; in der Kommandozeile können außerdem Optionen und Argumente für das/die Kommandos enthalten sein. Eine Kommandozeile wird über die Shell in den Rechner eingegeben, um eine oder mehrere Aufgaben ausführen zu lassen.

Kommando

Der Name einer Datei, die ein Programm enthält, das auf Anforderung vom Rechner ausgeführt werden kann. Compilierte Programme und Shell-Programme sind bestimmte Formen solcher Kommandos.

Kontextsuche

Eine Methode zum Auffinden eines bestimmten Zeichenmusters (Zeichenkette) bei der Textbearbeitung mit einem Editor. Editierkommandos, die eine Kontextsuche auslösen, durchsuchen den Puffer nach einer Entsprechung für die im Kommando angegebene Zeichenkette. Siehe auch **Zeichenkette**.

Maskierungszeichen

Bestimmte Sonderzeichen, die für die Shell eine Sonderbedeutung haben. Zu den Maskierungszeichen gehören *, ? und []. Maskierungszeichen werden in Mustern eingesetzt, die für Dateinamen stehen.

Mehrplatzsystem

Ein Betriebssystem, das mehrere Benutzer gleichzeitig an einem System unterstützen kann.

Modem

Modulator/Demodulator. Eine Einheit, durch die ein Terminal und ein Rechner über eine Telefonleitung miteinander verbunden werden können. Ein Modem wandelt digitale Signale in Töne und umgekehrt Töne in digitale Signale um, so daß ein Terminal und ein Rechner Daten über normale Telefonleitungen austauschen können.

Modus "Anhängen"

Ein Modus in der Textbearbeitung, in dem die eingegebenen Zeichen als Text in den Puffer des Editors geschrieben werden; dabei wird der Text nach der aktuellen Position im Puffer eingefügt bzw. angehängt. Siehe **Texteingabemodus**, vgl. auch **Kommandomodus** und **Einfügemodus**.

Modus

Allgemein eine Betriebsart (beispielsweise der Modus "Anhängen" eines Editors). Im Zusammenhang mit einem Dateisystem ist der Modus eine Oktalzahl, mit der festgelegt wird, wer in welcher Form Zugriff auf die Dateien eines Benutzers hat. Siehe **Zugriffsrechte**.

Multi-Tasking-Fähigkeit

Die Fähigkeit eines Betriebssystems, mehrere Programme gleichzeitig auszuführen.

normale Datei

Eine Datei, die Text oder Daten enthält und nicht ausführbar ist. Vgl. **ausführbare Datei**.

nroff

Ein Textformatierprogramm, das als Zusatzeinrichtung zum Betriebssystem UNIX erhältlich ist. Mit dem Programm **nroff** kann man eine formatierte Bildschirmanzeige oder einen formatierten Papierausdruck einer Datei erzeugen. Siehe **Textformatierprogramm**.

Option

Spezielle Anweisungen, wie ein Kommando ausgeführt werden soll. Optionen sind Typen von Argumenten, die nach einem Kommando und in der Regel vor anderen Argumenten in der Kommandozeile stehen. Konventionsgemäß wird einer Option ein Minuszeichen (-) vorangestellt, um sie von anderen Argumenten zu unterscheiden. Bei bestimmten Kommandos im Betriebssystem UNIX können mehrere Optionen gleichzeitig angegeben werden. Im Kommando **ls -l -a** Verzeichnis sind beispielsweise die Teile **-l** und **-a** Optionen, mit denen das Kommando **ls** modifiziert wird. Vgl. **Argument**.

Parameter

Eine besondere Form einer Variablen, die in Shell-Programmen eingesetzt wird, um auf Werte zuzugreifen, die in Beziehung zu den Argumenten in der Kommandozeile oder zur Umgebung stehen, in der das Programm ausgeführt wird. Siehe auch **Positionsparameter**.

Parität

Ein Verfahren, mit dem der Rechner prüft, ob die empfangenen Daten mit den abgesandten Daten übereinstimmen.

Paßwort

Eine Codewort, das nur der Benutzer kennt, und zu dessen Eingabe er während der Anmeldeprozedur aufgefordert wird. Der Rechner prüft mit dem Paßwort, ob der betreffende Benutzer wirklich dazu berechtigt ist, das System zu benutzen.

Peripheriegerät

Zusatzgeräte, die vom Hauptrechner gesteuert werden und meist für Funktionen wie Eingabe, Ausgabe und Datenspeicherung verwendet werden. Dazu gehören Terminals, Drucker und Plattenlaufwerke.

permanente Datei

Eine Datei mit Daten, die permanent in der Struktur des Dateisystems gespeichert ist. Eine permanente Datei kann mit einem Texteditor geändert werden; dieser legt zum Ändern eine temporäre Kopie der Datei in einem Puffer an. Nachdem die Änderungen im Puffer vorgenommen wurden, müssen sie in die permanente Datei abgespeichert werden, um die Änderungen festzuschreiben. Siehe auch **Puffer**.

Pfadname

Eine Folge von Verzeichnisnamen, die durch Schrägstrich (/) getrennt sind und mit dem Namen einer Datei oder eines Verzeichnisses enden. Mit dem Pfadnamen wird der Weg von einem bestimmten Verzeichnis zur angegebenen Datei definiert.

Pipeline

Eine Folge von Filtern, die jeweils durch | (Pipe-Zeichen) getrennt sind. Die Ausgabe aus jedem Filter wird zur Eingabe des nächsten Filters der Folge. Die Ausgabe des letzten Filters der Pipeline wird in die Standardausgabe geschrieben oder kann nochmals in eine Datei umgelenkt werden. Siehe **Filter**.

Pipe

Eine Form der Umlenkung der Ausgabe eines Kommandos, so daß sie als Eingabe in ein weiteres Kommando verwendet wird. Die Ausgabe wird mit dem Zeichen | umgelenkt. Beim Shell-Kommando `who | wc -l` wird beispielsweise die Ausgabe aus dem Kommando `who` in das Kommando `wc` umgelenkt, so daß als Endergebnis die Anzahl der Benutzer angezeigt wird, die gerade am UNIX-System angemeldet sind.

Positionsparameter

Numerierte Variablen, mit denen in einer Shell-Prozedur auf die Zeichenketten zugegriffen wird, die als Argumente in der Kommandozeile angegeben wurden, mit welcher die Shell-Prozedur aufgerufen wurde. Der Name der Shell-Prozedur ist dabei der Positionsparameter \$0. Siehe **Variable** und **Shell-Prozedur**.

Programm

Die Anweisungen, mit denen einem Rechner mitgeteilt wird, wie er eine bestimmte Aufgabe ausführen soll. Programme bilden die Software, die vom Benutzer ausgeführt werden kann.

Prozeß

Im allgemeinen wird damit ein Programm bezeichnet, das sich in einer bestimmten Phase der Ausführung befindet. Im Betriebssystem UNIX sind darin auch Werte der Rechnerumgebung eingeschlossen, darunter Speicherinhalt, Registerwerte, Name des aktuellen Verzeichnisses, Status von Dateien, bei der Anmeldung aufgezeichnete Daten u.a.

Puffer

Ein temporärer Speicherbereich des Rechners, der bei Editoren verwendet wird; darin werden Änderungen in einer Kopie einer bestehenden Datei vorgenommen. Der Inhalt einer Datei wird zum Bearbeiten in den Puffer gelesen; dort kann man den Text ändern. Damit die Änderungen festgeschrieben werden, muß der Inhalt des Puffers wieder in die Datei geschrieben bzw. abgespeichert werden. Vgl. **permanente Datei**.

Quellcode

Die nicht compilierte Version eines Programms, die in einer Programmiersprache wie C oder Pascal geschrieben ist. Der Quellcode muß von einem Compiler-Programm in Maschinensprache übersetzt werden, damit das Programm vom Rechner ausgeführt werden kann.

relativer Pfadname

Der Pfadname zu einer Datei oder einem Verzeichnis, der von dem Verzeichnis ausgeht, in dem man gerade arbeitet.

root

Das Verzeichnis, unter dem alle weiteren Verzeichnisse und Dateien des Dateisystems angeordnet sind; es wird durch einen Schrägstrich (/) bezeichnet.

sekundäres Bereit-Zeichen

Ein Zeichen, das von der Shell auf dem Bildschirm ausgegeben wird, um anzuzeigen, daß das auf das primäre Bereit-Zeichen eingegebene Kommando unvollständig ist. Der Standardwert für das sekundäre Bereit-Zeichen im Betriebssystem UNIX ist das Zeichen "größer als" (>).

Shell-Prozedur

Eine ausführbare Datei, die kein kompiliertes Programm ist (d. h. kein Programm in Maschinensprache). Mit einer Shell-Prozedur wird die Shell angewiesen, die Kommandos zu lesen und auszuführen, die sich in einer Datei befinden. Damit kann man eine häufig benötigte Folge von Kommandos in einer Datei speichern. Wird auch Shell-Programm oder Kommandodatei genannt. Siehe **ausführbare Datei**.

Shell

Ein UNIX-Systemprogramm, das die Kommunikation zwischen dem Benutzer und dem Rechner verwaltet. Die Shell wird auch als Kommandointerpreter bezeichnet, da sie die vom Benutzer eingegebenen Kommandos in eine Sprache übersetzt, die der Rechner versteht. Die Shell nimmt Kommandos entgegen und ruft das entsprechende Programm auf.

Software

Anweisungen und Programme, mit denen der Rechner zu einer bestimmten Aktion veranlaßt wird. Gegensatz zu **Hardware**.

Sonderzeichen

Ein Zeichen, das für die Shell eine Sonderbedeutung hat; Sonderzeichen werden für häufige Shell-Funktionen wie Dateiumlenkung, Pipes, Hintergrundausführung und Dateinamen-Expansion verwendet. Zu den Sonderzeichen gehören: <, >, |, ;, &, *, ?, [und].

Standardausgabe

Eine offene Datei, die normalerweise direkt einer primären Ausgabeinheit wie zum Beispiel einem Terminaldrucker oder -bildschirm zugeordnet ist. Die Ausgabe aus dem Rechner geht standardmäßig in diese Datei und von dort zur Ausgabeinheit. Man kann die Standardausgabe in eine andere Datei anstatt zum Drucker oder Bildschirm umlenken; dazu wird ein Argument im Format `> Datei` verwendet. Die Ausgabe wird dann in die angegebene Datei geleitet.

Standardeingabe

Eine offene Datei, die normalerweise direkt der Tastatur zugeordnet ist. Die Eingabe für ein Kommando geht standardmäßig von der Tastatur in diese Datei und von dort in die Shell. Die Standardeingabe kann umgelenkt werden, so daß sie von einer anderen Datei anstatt von der Tastatur kommt; dazu wird ein Argument im Format `< Datei` verwendet. Dann kommt die Eingabe für das Kommando aus der angegebenen Datei.

Standardwert

Ein Wert oder eine Bedingung, der bzw. die automatisch gilt, wenn man ihn bzw. sie nicht ausdrücklich ändert. Beispielsweise hat das Bereit-Zeichen der Shell den Standardwert `$`, solange man ihn nicht ändert.

Steuerzeichen

Ein nicht druckbares Zeichen, das eingegeben wird, indem man die CONTROL-Taste (Steuertaste) gedrückt hält und dann ein Zeichen eingibt. Steuerzeichen werden für Sonderfunktionen eingesetzt. Wenn man beispielsweise eine lange Datei mit dem Kommando `cat` auf dem Bildschirm anzeigen läßt, kann man die Anzeige mit CONTROL-s (`^s`) anhalten, so daß man sie lesen kann, und mit CONTROL-q (`^q`) weiterlaufen lassen.

stummes Zeichen

Siehe **verdecktes Zeichen**.

Suchmuster

Siehe **Zeichenkette**.

Systemverwalter

Die Person, die den Rechner überwacht und steuert, auf dem das Betriebssystem UNIX läuft; wird manchmal auch als Superuser bezeichnet.

Targon /31 Rechner

Rechner des Herstellers NCAG.

Timesharing

Eine Form der Datenverarbeitung, bei der mehrere Benutzer scheinbar gleichzeitig ein Rechnersystem gemeinsam nutzen. Der Rechner arbeitet dabei für die Benutzer nacheinander, durch die hohe Geschwindigkeit der Verarbeitung erscheint es jedoch so, als ob er jedem Benutzer die gesamte Verarbeitungskapazität bietet.

Terminal

Ein Ein-/Ausgabegerät an einem Rechnersystem, das normalerweise aus einer Tastatur mit einem Bildschirm oder einem Drucker besteht. Über das Terminal werden dem Rechner Anweisungen übergeben und vom Rechner Informationen ausgegeben.

Texteditor

Ein Programm zum Erstellen, Ändern und Löschen von Text mit Hilfe eines Rechners. Bei den meisten Texteditoren gibt es zwei Modi - einen Eingabemodus zur Eingabe von Text und einen Kommandomodus zum Verschieben, Kopieren und Ändern von Text. Beispiele für Texteditoren sind die Editoren des Betriebssystems UNIX, ed und vi. Siehe **Zelleneditor** und **Bildschirmeditor**.

Texteingabemodus

Ein Modus in der Textbearbeitung, in dem die eingegebenen Zeichen als Textzeichen in den Puffer des Editors eingegeben werden. Um ein Kommando auszuführen, muß man den Texteingabemodus verlassen. Vgl. **Kommandomodus**, **Modus "Anhängen"** und **Einfügemodus**.

Textformatierprogramm

Ein Programm, mit dem eine Textdatei zur Druckausgabe vorbereitet wird. Dazu muß die Datei auch bestimmte Sonderkommandos zur Strukturierung der Druckausgabe enthalten. Mit solchen Sonderkommandos werden dem Formatierprogramm beispielsweise Randeinstellungen, der Beginn neuer Absätze, das Format von Listen und Tabellen, die Anordnung von Abbildungen usw. mitgeteilt. Als Zusatzprogramme sind für das Betriebssystem UNIX die Textformatierprogramme **nroff** und **troff** erhältlich.

troff

Ein Textformatierprogramm, das als Zusatz zum Betriebssystem UNIX erhältlich ist. Mit dem Programm **troff** kann eine entsprechend formatierte Datei auf einem Drucker in höchster Druckqualität ausgegeben werden. Siehe auch **Textformatierprogramm**.

tty

Ursprünglich die Abkürzung für "teletype terminal", d. h. Fernschreiberterminal. Wird heute allgemein zur Bezeichnung eines Benutzerterminals verwendet.

Umgebung

Die Bedingungen, unter denen man mit dem UNIX-System arbeitet. Zur Umgebung gehören die Einrichtungen zur Überprüfung der Anmeldung wie auch die Form der Interaktion des Benutzers mit dem Betriebssystem UNIX und dem Rechner. Zur Shell-Umgebung gehört beispielsweise die Zeichenkette für das Bereit-Zeichen der Shell, spezielle Angaben für die Zeichen zum Löschen eines Einzelzeichens oder einer Zeile sowie Kommandos zur Übermittlung einer Ausgabe vom Terminal an den Rechner.

UNIX

Ein interaktives Vielzweck-Betriebssystem für Mehrplatz-Timesharing, das von AT&T Bell Laboratories entwickelt wurde. Das Betriebssystem UNIX ermöglicht die gemeinsame Nutzung begrenzter Rechner-Ressourcen durch mehrere Benutzer und bietet eine effektive Schnittstelle vom Benutzer zum Rechnersystem.

Unterverzeichnis

Ein Verzeichnis, das sich unter einem anderen Verzeichnis in der Struktur des Dateisystems befindet; auch Kindverzeichnis genannt.

Variable

Ein Symbol, dessen Wert veränderlich ist. In der Shell ist eine Variable ein Symbol, das für eine bestimmte Zeichenkette (d. h. einen Wert) steht. Variablen können interaktiv mit der Shell oder in einer Shell-Prozedur verwendet werden. Zwei Formen von Variablen in einer Shell-Prozedur sind Positionsparameter und Schlüsselwort-Parameter (eine Beschreibung der Schlüsselwort-Parameter ist in "Anleitung zur Shell" enthalten).

verdeckte Zeichen

Eine Gruppe von Zeichen innerhalb des ASCII-Zeichensatzes, die nicht druckbar sind. Beispiele sind BACKSPACE, ESCAPE und <^d>.

Verzeichnis

Ein Dateityp, der dazu dient, andere Dateien oder Verzeichnisse zusammenzufassen und zu ordnen. In ein Verzeichnis können keine Texte oder anderen Daten direkt eingegeben werden. Nähere Angaben sind in Anhang A, "Übersicht über das Dateisystem", enthalten.

Vollduplex

Eine Form der Datenkommunikation, bei der ein Rechnersystem Daten gleichzeitig senden und empfangen kann. Terminals und Modems verfügen normalerweise über die Möglichkeit der Einstellung für Halbduplex (jeweils nur eine Richtung) und Vollduplex (beide Richtungen gleichzeitig). Für das UNIX-System wird die Einstellung Vollduplex verwendet.

vollständiger Pfadname

Ein Pfadname, der mit dem Root-Verzeichnis des UNIX-Systems beginnt und bis zu einer bestimmten Datei oder einem Verzeichnis führt. Jeder Datei und jedem Verzeichnis im UNIX-System ist ein eindeutiger Pfadname zugeordnet; er wird auch absoluter Pfadname genannt. Siehe **Pfadname**.

Vorauslesen

Die Fähigkeit des UNIX-Systems, Eingaben des Benutzers zu lesen und zu interpretieren, während verarbeitete Informationen von vorherigen Eingaben am Terminal ausgegeben werden. Das UNIX-System behandelt Eingaben und Ausgaben separat, so daß sie korrekt verarbeitet werden.

Vordergrundverarbeitung

Die normale Form der Kommandoausführung. Beim Ausführen eines Kommandos im Vordergrund wartet die Shell die Ausführung eines Kommandos ab, bevor sie den Benutzer zur Eingabe eines weiteren Kommandos auffordert. Mit anderen Worten, der Benutzer gibt etwas in den Rechner ein und dieser "antwortet", bevor der Benutzer etwas anderes eingeben kann.

Werkzeug

Ein Paket von Softwareprogrammen.

Zeichenkette

Eine Gruppe oder ein Muster von Zeichen, wie zum Beispiel ein Wort oder ein Ausdruck; darin können auch Sonderzeichen enthalten sein. In einem Texteditor werden solche Sonderzeichen dazu verwendet, im Rahmen einer Kontextsuche nach dem Auftreten eines solchen Musters im Editierpuffer zu suchen.

Zeichenvariable

Eine Folge von Zeichen, die den Wert einer Shell-Variablen darstellen können. Siehe **Variable**.

Zeileneditor

Ein Programm zur Textbearbeitung, bei der Text in einer Datei zeilenweise verarbeitet wird. Die Kommandos zum Erstellen, Ändern und Löschen von Text arbeiten dabei mit Zeilenadressen, mit denen angegeben wird, an welcher Stelle der Datei die Änderungen vorgenommen werden sollen. Die Änderungen können überprüft werden, indem man die geänderten Zeilen anzeigen läßt. Vgl. **Texteditor** und **Bildschirmeditor**.

Zugriffsrechte

Zugriffsmodi, die Verzeichnissen und Dateien zugeordnet sind und mit denen Benutzern des Systems die Möglichkeit, Verzeichnisse und Dateien zu lesen, zu ändern und/oder sie auszuführen, eingeräumt oder verweigert wird. Man legt die Zugriffsrechte für die eigenen Verzeichnisse und Dateien fest, indem man den Modus mit dem Kommando **chmod** ändert.

Bestell-Nr.: 10919.00.8.93