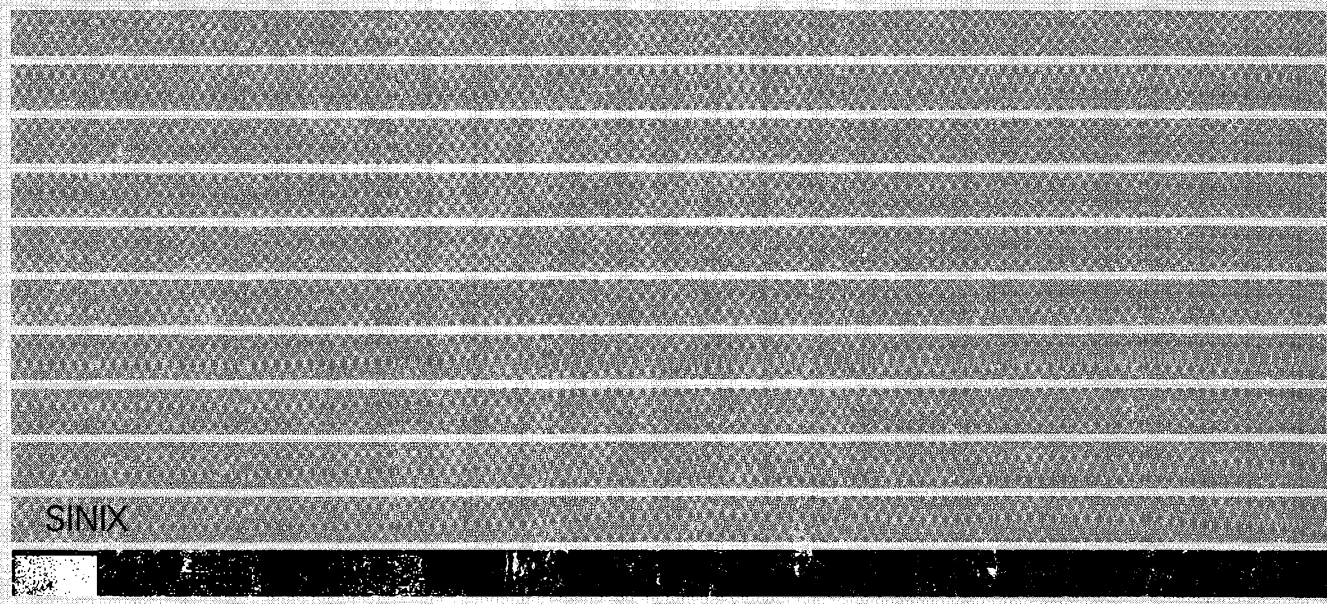
SIEMENS
NIXDORF



SINIX V5.41

Kommandos Band 1
A – K

Beschreibung

Sie haben

uns zu diesem Handbuch etwas mitzuteilen?
Schicken Sie uns bitte Ihre Anregungen unter
Angabe der Bestellnummer dieses Handbuches.

Siemens Nixdorf Informationssysteme AG
Manualredaktion STM QM 2
Otto-Hahn-Ring 6
W-8000 München 83

Fax: (089) 636-40443

email im EUnet:
man@sieqm2.uucp

Dieses Handbuch ist auf chlorfrei gebleichtem Papier gedruckt.

SINIX V5.41

Kommandos Band 1
A – K

Beschreibung

Ausgabe Februar 1992 (SINIX V5.41)

... und Schulung?

Zu dem nachstehend beschriebenen Produkt, wie zu fast allen DV-Themen, bieten unsere regionalen Training Center in Berlin, Essen, Frankfurt, Hannover, Hamburg, München, Mainz, Stuttgart, Wien und Zürich Kurse an.

Auskunft und Info-Material:

Systemfamilien 7·500 und 8890	**Telefon (0 89) 6 36 - 4 89 87**
Ein- und Mehrplatzsysteme	**Telefon (0 89) 6 36 - 4 24 80**

Siemens Nixdorf Training Center
Postfach 83 09 51, W-8000 München 83

X/Open
PLUS XPG3

Inhalt

1 Einleitung

Die Kommandos des Betriebssystems SINIX stellen eine umfangreiche Sammlung von vielseitig verwendbaren Programmen und Prozeduren zur Ansteuerung des Rechners dar, die meist jeder Benutzer aufrufen kann.
Das vorliegende Handbuch enthält die Beschreibung der SINIX-Benutzerkommandos. Es ist ein grundlegendes Nachschlagewerk für alle Benutzer von SINIX, die bereits Grundkenntnisse in diesem Betriebssystem haben. Die Grundkenntnisse sollten dem Wissensstand entsprechen, der durch den *Leitfaden für Benutzer* [3] vermittelt wird. Weiterführende Literatur ist im *Literaturverzeichnis* aufgeführt, das sich im Band *Tabellen und Verzeichnisse* des vorliegenden Handbuchs befindet.

1.1 Konzept des Handbuchs

Die Kommandos sind in alphabetischer Reihenfolge beschrieben. Als Suchhilfe erscheint im äußeren Kolumnentitel jeder Seite der Name des jeweils beschriebenen Kommandos und bei sehr langen Beschreibungen zusätzlich ein Schlagwort.
Das Handbuch ist in fünf Teile gegliedert, verteilt auf drei Bände:

Teil 1:	Band 1:	Einleitung
Teil 2:		Kommandoeingabe
Teil 3:		NLS (Native Language System)
Teil 4:		Kommandos A - K
	Band 2:	Kommandos L - Z
Teil 5:	Band 3:	Tabellen und Verzeichnisse: Inhalt Kommandoübersicht Reguläre Ausdrücke Sonderzeichen der Bourne-Shell sh Gerätedateien für Datenträger Dateien des SPOOL-Systems Zeichensatz ISO 646 Fachwörter Literatur Stichwörter

1.2 Voraussetzungen

Die Leistungsfähigkeit des Betriebssystems und damit Ihres Rechners wird durch verschiedene Faktoren beeinflußt. Um den reibungslosen Ablauf der Arbeit mit Ihrem Rechner und dem Betriebssystem zu ermöglichen, bedarf es bestimmter Voraussetzungen:

- Einige Dienste, die SINIX bereithält, müssen durch den zuständigen Verwalter zuerst eingerichtet werden und bedürfen anschließend ständiger Pflege. Zu nennen ist hier z.B. das *mail*-System, mit dessen Hilfe Sie Post im lokalen Netz oder auch in entfernten Netzen verschicken und empfangen können.
- Einige Kommandos können Sie nur mit bestimmten Hardware-Voraussetzungen verwenden. Sie arbeiten nur mit bestimmten Rechnertypen oder mit bestimmten Datensichtstationen. So benötigen z.B. alle Kommandos, die unter *layers* arbeiten, grafikfähige Bildschirme. Netzkommandos funktionieren nur, wenn die betreffende Rechneranlage an einem Netz angeschlossen ist.
- Bei der Installation von SINIX auf einem Rechner kann festgelegt werden, welche Softwarepakete auf die Festplatte kopiert werden und welche nicht. Es könnte deshalb sein, daß ein Kommando auf Ihrem System nicht verfügbar ist. Wenden Sie sich in einem solchen Fall an Ihren Systemverwalter.

1.3 Beschreibungsformat

Die Beschreibung eines jeden Kommandos hält sich, soweit möglich, an ein festes Raster:

- Äußerer Kolumnentitel
- Innerer Kolumnentitel (optional)
- Hauptüberschrift
- Beschreibung des Kommandos
- Syntax
- Syntaxbeschreibung
- Ende-Status (optional)
- Fehlermeldung(en) (optional)
- Datei(en) (optional)
- Umgebungsvariable(n) (optional)
- Internationale Umgebung (optional)
- Beispiel(e) (optional
- Siehe auch (optional)

Die hier aufgeführten Bestandteile werden im folgenden erläutert.

Äußerer Kolumnentitel

Der äußere Kolumnentitel enthält den Kommandonamen. Bei umfangreichen Kommandos dient der äußere Kolumnentitel außerdem als Orientierungshilfe. Dem Kommandonamen folgt dann ein Stichwort zum Seiteninhalt.

Innerer Kolumnentitel (optional)

Der innere Kolumnentitel wird bei einer zum Verständnis nötigen Klassifizierung eines Kommandos verwendet, z.B. *eingebautes sh-Kommando.*

Hauptüberschrift

Die Hauptüberschrift enthält:

- den Namen des Kommandos
- eine Kurzbeschreibung des Kommandos
- die englische Bezeichnung des Kommandos, wenn Sie mnemotechnisch sinnvoll ist. Oft sind die Kommandonamen englische Abkürzungen ihrer Funktion, z.B. *mkdir* für *make directory*.

Beschreibung des Kommandos

Hier ist beschrieben:

- die Arbeitsweise des Kommandos
- die unterschiedlichen Aufgaben der verschiedenen Kommando-Formate, falls mehrere Formate vorhanden sind
- in welcher Umgebung das Kommando zu verwenden ist (z.B. Einträge in Dateien, Zugriffsrechte, ...)
- Hintergrundinformationen
- was vor oder nach dem Kommandoaufruf zu beachten ist.

Bei Kommandos, die komplexe Programme aufrufen (z.B. *awk, ced, sh*), ist hier nur die SINIX-Kommandoebene beschrieben. Weitergehende Informationen, z.B. die Arbeitsweise eines Editors wie *vi*, finden Sie im Anschluß an die Syntax.

Syntax

cmd[␣**-a**][␣**-b**][␣**-c**][␣**-d**arg1][␣**-f**␣arg2]␣datei␣...

Die Syntax des Kommandos ist grau unterlegt.

Entnehmen Sie die Bedeutung der Metasyntax der folgenden Beschreibung.

In der Syntax:

halbfette Zeichen
: Konstanten: Diese Zeichen sind direkt so einzugeben, wie sie gedruckt sind.

normale Zeichen
: Variablen: Diese Zeichen sind Stellvertreter für andere Zeichen, die Sie auswählen und eingeben.

[ ]
: optional: Alles, was zwischen den eckigen Klammern steht, können, aber müssen Sie nicht eingeben. Welche Auswirkungen das hat, entnehmen Sie der Beschreibung der Optionen und Argumente. Die eckigen Klammern selbst dürfen Sie nicht eingeben, es sei denn, es ist ausdrücklich angegeben.

␣
: Ein Leerzeichen, das Sie eingeben müssen.

...
: Der vorhergehende Ausdruck kann wiederholt werden. Falls zwischen den Wiederholungen Leerzeichen eingegeben werden müssen, die nicht im Ausdruck enthalten sind, steht vor ... ein ␣

Beispiel

cmd[␣**-a**][␣**-b**][␣**-c**][␣**-d**arg1][␣**-f**␣arg2]␣datei␣...

Sie müssen eingeben:

- *cmd*
- für *datei* eine oder mehrere Dateien, die jeweils durch ein Leerzeichen getrennt werden.

Sie können zusätzlich angeben:

- eine oder mehrere der Optionen *-a, -b, -c*; diese Optionen können Sie einzeln:
  -a -b -c
  oder zusammengefaßt angeben:
  -abc
- die Option *-d*, dabei muß *arg1* durch ein Argument ersetzt werden
- die Option *-f*, dabei muß *arg2* durch ein Argument ersetzt werden.

Im Fließtext:

Im Fließtext wird nicht zwischen Konstanten und Variablen unterschieden, alle Elemente der Syntax sowie sonstige Dateinamen, Pfadnamen und Kommandos sind dort in *kursiver* Schrift wiedergegeben.

Syntaxbeschreibung

option
(Beschreibung der Optionen siehe *Kapitel 2, Kommandoeingabe, Eingabe von Optionen*)

argument
Beschreibung der übrigen Argumente, die Sie beim Aufruf an ein Kommando übergeben können, z.B. Eingabedateien, Ausgabedateien, Parameter, Variablen, Feldtrenner usw..

Ende-Status (optional)

Der Ende-Status ist der Wert, den ein Kommando nach seiner Ausführung an den aufrufenden Prozeß zurückliefert. Der Ende-Status gibt Auskunft darüber, wie das Kommando abgelaufen ist. Der Ende-Status ist ein Zahlenwert und wird in der Variablen ? abgelegt. Sie fragen den Ende-Status mit dem Befehl *echo $?* ab.

Der Abschnitt Ende-Status ist nur angegeben, wenn er von folgendem Regelfall abweicht:

0 nach korrekter Durchführung des Kommandos
>0 bei Fehler

Fehlermeldung(en) (optional)

Hier werden wichtige Fehlermeldungen angegeben und erläutert, sowie Hinweise zur Fehlervermeidung und -behebung gegeben.

Fehlermeldungen werden standardmäßig auf die Standard-Fehler-Ausgabe ausgegeben. Normalerweise ist der Bildschirm die Standard-Fehlerausgabe.

Datei(en) (optional)

Hier werden solche Dateien angegeben, auf die das betreffende Kommando zugreift oder die dieses anlegt.

Umgebungsvariable(n) (optional)

Einige Kommandos greifen auf Umgebungsvariablen zu. Diese werden hier aufgeführt.

Internationale Umgebung (optional)

Hier sind die Auswirkungen des NLS auf das Kommando beschrieben (siehe *Kapitel 3, Internationale Umgebung - NLS*).

Beispiel(e) (optional)

Beispiele sollen veranschaulichen:

- die Hauptfunktion des Kommandos
- den Einsatz der wesentlichen Optionen
- sinnvolle komplexe Kombinationen von Optionen und Argumenten.

Siehe auch (optional)

In diesem Abschnitt finden Sie Verweise auf andere Kommandos, die eine ähnliche Funktionsweise haben oder mit dem betreffenden Kommando zusammenarbeiten. Außerdem wird auf weitere Literatur zu diesem Kommando verwiesen.
Systemaufrufe und Bibliotheksfunktionen für den C-Software-Entwickler sind durch ein Paar runder Klammern gekennzeichnet, z.B. *alarm()*.

1.4 Verweise

Die folgenden Beispiele zeigen, in welcher Form auf andere Textabschnitte, Handbücher, Kommandos, Funktionen, Systemaufrufe, Dateiformate sowie auf weiterführende Literatur verwiesen wird.

Andere Textabschnitte

siehe *Kapitel 3, Internationale Umgebung - NLS*	Verweis auf den Abschnitt *Kapitel 3, Internationale Umgebung - NLS* im vorliegenden Handbuch.
siehe *Tabellen und Verzeichnisse, Literatur, Literatur zu UNIX*	Verweis auf den Teil *Tabellen und Verzeichnisse*, Abschnitt *Literatur*, Unterabschnitt *Literatur zu UNIX* im vorliegenden Handbuch.

Andere Handbücher

siehe *Leitfaden für Benutzer* [3]	Verweis auf ein Handbuch, das im Literaturverzeichnis unter der Nummer [3] aufgeführt ist.

Kommandos, Funktionen, Systemaufrufe, Dateiformate

siehe *awk*	Verweis auf das Kommando *awk* im vorliegenden Handbuch.
siehe *printf()* [19]	Verweis auf die C-Funktion *printf()* in dem Handbuch, das im Literaturverzeichnis unter der Nummer [19] aufgeführt ist.
siehe *creat()* [19]	Verweis auf den Systemaufruf *creat()* in dem Handbuch, das im Literaturverzeichnis unter der Nummer [19] aufgeführt ist.
siehe *a.out* [19]	Verweis auf das Dateiformat von *a.out* im dem Handbuch, das im Literaturverzeichnis unter der Nummer [19] aufgeführt ist.

Weiterführende Literatur

siehe Kernighan, Ritchie, *The C Programming Language* [54]	Verweis auf ein Buch, das im Literaturverzeichnis unter der Nummer [54] aufgeführt ist.

2 Kommandoeingabe

2.1 Kommandoeingabe von der Shell aus

Kommandos schreiben Sie anschließend an das Bereitzeichen der Shell (Standard: $␣) in eine Zeile und schließen die Eingabe mit der Taste ↵ ab. Die Taste ↵ ist für die Shell das Zeichen, alles vorhergehende als Kommando zu interpretieren. Wenn die Zeile am Bildschirm zu kurz ist, schreiben Sie am Zeilenende einfach weiter, ohne ↵ zu drücken, oder Sie setzen die Zeile mit \↵ fort. Das Zeichen Gegenschrägstrich \ entwertet die Kommandoabschlußfunktion von ↵. Sie können nun die Eingabe Ihres Kommandos fortsetzen. Nachdem Sie die Taste ↵ (ohne \) gedrückt haben, wird das Kommando ausgeführt.

Die meisten Kommandos geben eine Usage-Meldung aus, wenn sie mit einer falschen Option aufgerufen werden. Der Usage-Meldung können Sie entnehmen, mit welchen Optionen und Operanden Sie ein Kommando aufrufen können.

Beispiel

```
$ ls -y
ls: illegal option --y
usage: ls -1RadCxmnlogrtucpFbqisfL [files]
```

Wenn ein Kommando auf den Bildschirm Meldungen ausgibt und die Ausgabe länger als eine Bildschirmseite ist, dann können Sie

- die Ausgabe anhalten durch gleichzeitiges Betätigen der Tasten CTRL S
- die Ausgabe fortsetzen durch gleichzeitiges Betätigen der Tasten CTRL Q.

Eine seitenweise Ausgabe erhalten Sie mit dem Kommando *pg*, das Sie über eine Pipeline (siehe *Eingabe mehrerer Kommandos*) an das Kommando anhängen können:

```
kommando | pg ↵
```

2.2 Weitere Eingaben nach Kommandoaufruf

Manche Kommandos warten nach Abschluß mit ↵ auf weitere Eingaben von der Tastatur. Das sehen Sie daran, daß die Schreibmarke auf den Beginn der nächsten Zeile gesetzt wird, wobei entweder kein Bereitzeichen erscheint oder ein für das Kommando spezifisches Bereitzeichen ausgegeben wird. Die Zeilen, die Sie nun eingeben, schließen Sie jeweils mit der Taste ↵ ab. Beenden können Sie die Eingabe in der Regel mit der Taste END oder durch eine spezielle Anweisung des Kommandos.

Schließen Sie ein Kommando mit ↵ ab und ist diese Eingabe noch keine vollständige Kommandozeile oder haben Sie ↵ durch einen Gegenschrägstrich \ entwertet, dann gibt die Shell den Wert der Shell-Variablen `PS2` (Standard ist das Größer-Zeichen >) als Bereitzeichen aus.

2.3 Eingabe mehrerer Kommandos

Sie können mehrere Kommandos, durch Strichpunkt ; und beliebig viele Leerzeichen getrennt, in einer Zeile eingeben:

```
kommando; kommando; ... kommando ↵
```

Die Kommandos werden dann nacheinander ausgeführt, so als ob Sie diese einzeln eingegeben hätten.

Wenn Sie mehrere Kommandos mit einem senkrechten Strich | verbinden, wird die Standard-Ausgabe des ersten Kommandos als Standard-Eingabe für das folgende Kommando verwendet (siehe *sh*, *Pipelines*).

2.4 Eingabe von Optionen

Beim Aufruf von Kommandos geben Sie entweder keine, eine oder mehrere Optionen an.

Optionen modifizieren die Funktionsweise eines Kommandos. Sie bestehen in der Regel aus einem Buchstaben. Diese sind so in der Syntax angegeben, wie Sie sie eingeben müssen (halbfette Schrift in der Syntax) oder sie sind im Parameter *option* zusammengefaßt (normale Schrift in der Syntax).

Für die Angabe mehrerer Optionen gelten im allgemeinen die folgenden Regeln:

- Sie können die Optionen in beliebiger Reihenfolge angeben.
- Optionen ohne Argumente können auf zwei Arten eingegeben werden:

  einzeln: z.B. `cmd -a -b -c`

  oder

  zusammengefaßt: z.B. `cmd -abc`

- Optionen, die ein Argument benötigen, z.B. *-dargument* oder *-f argument*, können nicht mit Optionen ohne Argument zusammengefaßt werden. Sie werden einzeln, jeweils durch ein Leerzeichen getrennt, eingegeben:

  ```
  cmd -abc -dargument -f argument
  ```

- Optionen bzw. zusammengefaßte Optionen beginnen mit einem Bindestrich: *-abc*.

Wenn eine Kommandobeschreibung von diesen Regeln abweicht, gilt die abweichende Beschreibung des betreffenden Kommandos.

2.5 Eingabe von Dateinamen

Um Namen von Dateien oder Dateiverzeichnissen einzugeben, haben Sie mehrere Möglichkeiten:

- Sie geben einen relativen Pfadnamen an. Relative Pfadnamen gehen immer vom aktuellen Dateiverzeichnis aus.
- Sie geben einen absoluten Pfadnamen an. Absolute Pfadnamen beginnen mit einem Schrägstrich / und geben damit den Pfadnamen ausgehend vom das Root-Dateiverzeichnis an.

Beispiel

Sie befinden sich in */home/gast* und wollen die Dateien */home/gast/post/brief1* und */home/gast/protokoll* löschen.

```
$ pwd
/home/gast
$ ls -R
post
protokoll

./post:
brief1
```

Löschen mit Angabe der relativen Pfadnamen:

```
$ rm post/brief1 protokoll
$ ls
post
```

Löschen mit Angabe der absoluten Pfadnamen:

```
$ rm /usr/gast/post/brief1 /usr/gast/protokoll
$ ls
post
```

Außerdem können Sie die Sonderzeichen der Shell (siehe *sh* und *Tabellen und Verzeichnisse, Sonderzeichen der Bourne-Shell sh*) zur Erzeugung von Dateinamen verwenden. Dies ist besonders nützlich, wenn Sie mehrere Dateien gleichzeitig angeben möchten.

Für einfache Dateinamen gelten folgende Konventionen:

Erlaubt sind alle Zeichen bis auf die Zeichen / und \0 (Nullbyte als Abschluß von Zeichenreihen). Vermeiden sollten Sie in Dateinamen alle Sonderzeichen der Shell (siehe *Tabellen und Verzeichnisse, Sonderzeichen der Bourne-Shell sh*). Falls der Dateiname Leer- oder Tabulatorzeichen enthält, müssen Sie ihn in Anführungsstriche "..." setzen. Vermeiden Sie +, - und . am Anfang. Beachten Sie bitte auch, daß SINIX zwischen Groß- und Kleinschreibung unterscheidet.

3 Internationale Umgebung - NLS (Native Language System)

3.1 Definition des NLS

Bisher basierte das Betriebssystem SINIX, wie die meisten UNIX-Derivate, auf dem ASCII-Zeichensatz und auf dem amerikanischen Englisch als Sprache, in der der Benutzer mit dem Rechner kommunizieren konnte. Im ASCII-Zeichensatz ist jedes zu verarbeitende Zeichen durch eine eindeutige Folge von 7 Bits codiert und damit die Anzahl möglicher Zeichen auf maximal 128 beschränkt. Es gab keine Möglichkeit, Zeichen aus 8-bit-Zeichensätzen, die einen größeren Zeichenvorrat abdecken, zu verarbeiten und darzustellen oder andere Landessprachen und Konventionen zu unterstützen. Für die europäischen Sprachen braucht man z.B einen 8-bit-Zeichensatz, für die meisten östlichen Sprachen (wie Chinesisch oder Japanisch) sogar einen 16-bit-Zeichensatz oder noch größere Zeichensätze.

Die zunehmende internationale Verbreitung von UNIX erforderte jedoch eine Erweiterung des Systems, die den länderspezifischen Eigenheiten und Gewohnheiten der unterschiedlichen Anwender gerecht wird. So erwarten z.B. Anwender aus Deutschland ein anderes Format bei Datumsangaben als Anwender aus den USA.

Mit dem NLS (Native Language System) hat X/Open eine Schnittstelle definiert, die es ermöglicht,

- Anwendungsprogramme zu entwickeln, die in verschiedenen Landessprachen mit dem Benutzer kommunizieren und den dazugehörigen länderspezifischen Konventionen entsprechen. Solche Programme machen keine Annahmen über die Sprache des Benutzers und halten die Datenspezifikationen getrennt von der Programmlogik. Sie werden als *internationalisierte Programme* bezeichnet.
- die Ablaufumgebung eines internationalisierten Anwendungsprogramms zur Laufzeit mit der richtigen Landessprache und den entsprechenden länderspezifischen Konventionen zu versorgen. Dieses Verfahren nennt man *Lokalisierung*. Daher wird oft die Menge der länder- bzw. sprachspezifischen Konventionen als *Locale* bezeichnet.
- 8-bit-Zeichensätze zu verarbeiten. Ein 8-bit-Zeichensatz (256 verschiedene Byte) reicht aus, um die meisten westeuropäischen Sprachen zu unterstützen.

Das NLS umfaßt dafür insbesondere:

- Einen Bindungsmechanismus, der es dem Benutzer erlaubt, die Landessprache, die Verarbeitung nach regionalen Konventionen und den zugrundeliegenden Zeichensatz zur Laufzeit eines Anwendungsprogramms nach individuellem Bedarf einzustellen. Dies erfolgt mit Hilfe von Umgebungsvariablen.
- Meldungskataloge, die es erlauben, Meldungen eines Anwendungsprogramms getrennt von der Programmlogik zu halten, in verschiedene Landessprachen zu übersetzen und zur Laufzeit an die Anwendung zu binden.
- Internationalisierte C-Bibliotheksfunktionen, die keine Annahmen über Landessprache, Land und Zeichensatz machen und deshalb geeignet sind, universelle Zeichenklassifizierung, Umwandlung von Groß-/Kleinbuchstaben und Zahlenformaten durchzuführen sowie Zeichenketten zu sortieren.
- Einen Satz von C-Bibliotheksfunktionen, die es erlauben, zur Laufzeit eines Anwendungsprogramms die persönliche Sprachumgebung einzustellen, zu verändern und wieder abzustellen.
- 8-bit-transparente Standardkommandos, die auch über ASCII hinausgehende Zeichensätze verarbeiten können.

3.2 8-bit-Transparenz

Soll ein Programm die Zeichen aus 8-bit-Zeichensätzen verarbeiten können, darf es nicht - wie bei der Verarbeitung von 7-bit-ASCII-Zeichen möglich - das 8. Bit (das höchstwertige) eines Byte zu Prüfzwecken verwenden. Alle 8 Bits eines Byte müssen als Bestandteil der Zeichencodierung selbst betrachtet werden. Programme, die diese Bedingung erfüllen, heißen *8-bit-transparent* oder *8-bit-clean*.

Folgende SINIX-Kommandos aus vorliegendem Handbuch sind 8-bit-transparent:

ar	false	ps	uname
awk	fgrep	pwd	uniq
cat	find	red	unpack
ced	gencat	rm	uucp
cd	grep	rmdir	uudecode
chgrp	ic	sed	uuencode
chmod	iconv	sh (+ alle eingeb.	uulog
chown	kill	Kommandos)	uuname
cmp	ln	sleep	uupick
cp	lpr	sort	uustat
cpio	ls	stty	uuto
date	mail	tail	uux
diff	mailx	tar	wait
dumpmsg	mkdir	tee	wc
echo	mv	test	who
ed	pack	tr	
egrep	pcat	true	
expr	pg	umask	
extract	pr	tty	

3.3 Die NLS Schnittstelle

3.3.1 Mögliche Zeichensätze

Die NLS-Schnittstelle von SINIX ermöglicht es standardmäßig, die deutsche, englische und amerikanische Variante des 7-bit-Zeichensatzes ISO 646 (siehe *Tabellen und Verzeichnisse, Zeichensatz ISO 646*) sowie die 8-bit-Zeichensätze ISO 8859-1 und ISO 6937 zu verarbeiten.

Für die Übertragung von 8-bit-Zeichensätzen und für die Darstellung der 8-bit-Zeichen auf dem Bildschirm (z.B. Buchstaben mit Akzenten) brauchen Sie eine Datensichtstation des Typs 97801-480, die als 8-bit-Terminal eingestellt und konfiguriert sein muß. Wenden Sie sich hierfür an Ihren Systemverwalter.

3.3.2 Sprache der Meldungstexte

Falls Sie mit internationalisierten Programmen arbeiten, kommunizieren Sie mit diesem Programm in Ihrer Landessprache bzw. in der Sprache, die Sie in Ihrer persönlichen Arbeitsumgebung eingestellt haben (siehe unten, *Die persönliche internationalisierte Umgebung*): Die Programme geben Meldungstexte in der eingestellten Sprache aus, und Sie können Ihre Antworten, z.B. *ja* bzw. *yes*, in derselben Sprache eingeben.

Die Meldungstexte eines internationalisierten Programms stehen nicht im eigentlichen Programmcode, sondern getrennt davon in einem eigenen Meldungskatalog. Für alle einstellbaren Landessprachen gibt es jeweils eigene Meldungskataloge. Zur Laufzeit des internationalisierten Programms wird der Meldungskatalog, der zu der in der aktuellen Umgebung eingestellten Landessprache gehört, in das Programm eingebunden (siehe unten, *Meldungskataloge*).

3.3.3 Länderspezifische Eigenheiten

Das NLS berücksichtigt nicht nur verschiedene Landessprachen, sondern auch die folgenden länderspezifischen Eigenheiten:

- Das Format von Datums- und Zeitangaben

  In verschiedenen Ländern werden in einer Datumsangabe die Elemente *Tag*, *Monat* und *Jahr* in verschiedener Reihenfolge und mit verschiedenen Trennzeichen angegeben. So wird z.B. der 7.Oktober 1990 in den USA als 10/7/1990, in Japan als 1990/10/7 und in Deutschland als 7.10.1990 dargestellt. Ähnlich verhält es sich mit Zeitangaben:

- Namen der Wochentage und Monate

- Das Währungszeichen

- Die Darstellung des Dezimalpunkts, das Tausendertrennzeichen und das Exponentzeichen

  Das Dezimalkomma wird im englischsprachigen Raum durch einen Punkt (z.B. 2.15), im deutschsprachigen Raum durch ein Komma (z.B. 2,15) dargestellt. Das Tausendertrennzeichen ist im englischsprachigen Raum ein Komma (z.B. 1,350,000), im deutschsprachigen Raum dagegen ein Punkt (z.B. 1.350.000).

- Die Sortierreihenfolge

  Internationalisierte Programme, die Zeichen sortieren, entnehmen die Sortierkriterien Ihrer persönlichen Arbeitsumgebung. Standardmäßig können Sie in Ihrer Arbeitsumgebung festlegen, ob nach der Reihenfolge der Zeichen im ASCII-Zeichensatz oder nach englischen oder deutschen Sortierregeln sortiert werden soll. Bei den deutschen Regeln können Sie zwischen der Standard-Sortierreihenfolge und der Sortierreihenfolge des Telefonbuchs wählen (siehe unten, *@TELEPHONE*). Die verschiedenen Sortierregeln unterscheiden sich durch die Reihenfolge von Groß- und Kleinbuchstaben, Umlauten und Sonderzeichen (siehe unten, *Die persönliche internationalisierte Umgebung*).

- Die Umwandlung von Klein- in Großbuchstaben und umgekehrt

  Für jeden Zeichensatz, den Sie wählen können, legt eine interne Tabelle die jeweils gültige Umwandlung von Klein- in Großbuchstaben und umgekehrt fest.

3.4 Die NLS-Datenbank

In einer NLS-Datenbank sind die Elemente einer sogenannten *Locale* zusammengefaßt. Dabei steht *Locale* für einen Ort und beschreibt hier die persönlich einstellbare Arbeitsumgebung eines Anwenders ähnlich der von der Shell bekannten Arbeitsumgebung. Zu einer *Locale* werden die Beschreibungen der in einem Land oder in einer Region gebräuchlichen Konventionen oder auch die für einen individuellen Benutzer bevorzugte Arbeitsumgebung zusammengefaßt.

Eine *Locale* umfaßt:

- Konfigurationsdaten

  Konfigurationsdaten erkennen die Sprachen, die von einem System unterstützt werden, indem sie die gesetzten Werte für *Sprache*, *Territorium* und *Zeichensatz* auswerten (siehe unten, *Einstellung der internationalisierten Umgebung*). Jede gültige Kombination dieser drei Werte hat seine eigene Sortierreihenfolge, Zeichenklassifikation, Umwandlung von Klein- in Großbuchstaben und umgekehrt, Sprachinformation und seine eigenen Meldungskataloge.

- Tabellen mit der Sortierreihenfolge

  Diese Tabellen definieren die Sortierreihenfolge für jede unterstützte Landessprache. Um auch andere Zeichen als die des ASCII-Zeichensatzes sortieren zu können, werden folgende Fähigkeiten unterstützt :

  1. 1-zu-1-Zeichen-Abbildung
  2. 1-zu-2-Zeichen-Abbildung
     Bestimmte Zeichen werden in der Sortierreihenfolge behandelt, als wären sie zwei Zeichen, z.B. wird das deutsche *ß* zu *ss*.
  3. n-zu-1-Zeichen-Abbildung
     Bestimmte Zeichenfolgen werden in der Sortierreihenfolge behandelt, als wären sie nur ein Zeichen, z.B. das spanische *ch* und *ll*, die nach c und nach l sortiert werden.
  4. Vernachlässigung bestimmter Zeichen beim Sortieren (Don't-care-Zeichen)
     Ist z.B. der Bindestrich - als Don't-care-Zeichen definiert, dann werden die Zeichenfolgen *re-locate* und *relocate* gleich sortiert.

- Zeichenklassifikationstabellen

  Diese Tabellen ordnen die einzelnen Zeichen den Zeichenklassen zu, wie z.B. die C-Makros *isupper*, *islower*, *isnumeral*, *isspace*, *ispunct* ...

- Umwandlungstabellen für Groß-/Kleinbuchstaben (shift tables)

  Diese Tabellen ordnen jedem Kleinbuchstaben den dazugehörigen Großbuchstaben zu und umgekehrt.

- Informationen über Datums- und Zeitangaben

  1. Das Format von Datums- und Zeitangaben
  2. Die Namen der Wochentage und Monate
  3. Die Abkürzungen der Wochentags- und Monatsnamen

- Sonstige Information

  1. Die Darstellung von Dezimalpunkt und Tausendertrennzeichen
  2. Bestätigende und verneinende Antworten auf ja/nein-Fragen
  3. Währungszeichen

Standardmäßig liegen die NLS-Datenbanken unter */usr/lib/locale/Locale*. Darunter stehen Dateiverzeichnisse bzw. Dateien mit den gleichen Namen wie die Umgebungsvariablen bzw. *setlocale*-Parameter, mit denen man die internationale Umgebung einstellen kann.

Der Internationalisierungs-Compiler *ic* erzeugt aus einer Datenbank-Quelle *datenbank.in* ein Dateiverzeichnis *datenbank*. Dieses Dateiverzeichnis enthält folgende Dateien:

Xopen_info
Meldungstextquelldatei, die dem AT&T-Format (siehe unten, *Meldungskataloge*) entspricht. Sie enthält Informationen der *LC_MESSAGES*-Kategorie, wie z.B Antworten auf ja/nein-Fragen.

LC_TIME
Datenbankdatei, die die Tabelle mit Informationen über Datums- und Zeitangaben enthält.

chrtbl
Eingabedatei für das Kommando *chrtbl* (siehe *Referenzhandbuch für Systemverwalter* [7]), die Informationen über Zeichenklassifikation, Umwandlung von Klein- in Großbuchstaben und umgekehrt, Dezimalpunktdarstellung, Exponentzeichen und Tausendertrennzeichen enthält.

colltbl
Eingabedatei für das Kommando *colltbl* (siehe *Referenzhandbuch für Systemverwalter* [7]), die Informationen über die Umwandlung von Klein- in Großbuchstaben und umgekehrt enthält.

montbl
Eingabedatei für das Kommando *montbl* (siehe *Referenzhandbuch für Systemverwalter* [7]), die Informationen über Währungszeichen und Währungsformat enthält.

install_locale

Shell-Prozedur, die folgendes ausführt:

1. Sie bestimmt den Namen der Datenbank.
2. Sie übersetzt den Meldungskatalog *Xopen_info* mit dem Kommando *mkmsgs* (siehe unten, *Meldungskataloge*) und legt den Meldungskatalog unter */usr/lib/locale/Locale/LC_MESSAGES* ab.
3. Sie erzeugt die Datenbankdateien *LC_CTYPE, LC_NUMERIC, LC_COLLATE,* und *LC_MONETARY* mit den Kommandos *chrtbl, colltbl* und *montbl*.
4. Sie legt die Datenbankdateien *LC_CTYPE, LC_COLLATE, LC_MONETARY, LC_NUMERIC* und *LC_TIME* unter dem Dateiverzeichnis */usr/lib/locale/Locale* ab.

Damit wird also eine *Locale* installiert. Die Shell-Prozedur *install_locale* kann nur der Systemverwalter aufrufen.

3.5 Einstellung der internationalisierten Umgebung

3.5.1 Die persönliche internationalisierte Umgebung

Die NLS-Schnittstelle ermöglicht es jedem Benutzer, eine individuelle, seinen Erfordernissen entsprechende internationalisierte Arbeitsumgebung zu definieren. Hierfür steht u.a die Umgebungsvariable *LANG* zur Verfügung. Mit ihr lassen sich die Werte für *Sprache, Territorium* und *Zeichensatz* einstellen. Mit jedem Wert von *LANG* sind eine bestimmte Sortierreihenfolge, Konvertierung, Klassifikation und bestimmte Meldungskataloge verbunden.
Sie weisen der Variablen *LANG* die gewünschte Sprache, das Territorium und den Zeichensatz wie folgt zu:

LANG=sprache_territorium.zeichensatz

sprache_territorium.zeichensatz
: Name eines Dateiverzeichnisses unter */usr/lib/locale*. Die Länge dieser Zeichenkette darf *{NL_LANGMAX}* nicht überschreiten.

LANG kann vom Systemverwalter eingestellt werden, um eine Voreinstellung für das System bereitzustellen, oder vom Benutzer verändert werden.

LANG unterstützt einen allgemeinen Anforderungsmechanismus, mit dem die Benutzer ihre gesamten Anforderungen an die internationale Umgebung festlegen können. Das genügt, wenn eine einzige Umgebungseinstellung alle Anforderungen des Benutzers an Sortierreihenfolge, Zeichenklassifikation und Meldungsdarstellung erfüllt. Es deckt aber nicht den Fall ab, daß ein Benutzer in einer Sprache mit dem System kommunizieren und Textdateien in einer anderen Sprache sortieren will. Dieser Fall wird durch die Definition folgender Umgebunsvariablen gelöst, die die Veränderung einzelner Aspekte der internationalen Umgebung ermöglichen:

Variable	Einfluß auf:
LC_CTYPE	Zeichenklassen und Umwandlung von Klein- in Großbuchstaben und umgekehrt
LC_COLLATE	Sortierreihenfolge
LC_TIME	Datums- und Zeitangaben
LC_MONETARY	Währungszeichen und Währungsformat
LC_NUMERIC	Dezimalpunktdarstellung, Exponentzeichen und Tausendertrennzeichen
LC_MESSAGES	Meldungstexte und Antworten auf ja/nein-Fragen

Diese Variablen definieren Sie in folgendem Format:

```
LC_CTYPE
LC_COLLATE
LC_TIME
LC_MONETARY      =sprache_territorium.zeichensatz[@parameter]
LC_NUMERIC
LC_MESSAGES
```

sprache_territorium.zeichensatz
Name eines Dateiverzeichnisses unter */usr/lib/locale*. Die Länge dieser Zeichenkette darf *{NL_LANGMAX}* nicht überschreiten.

@parameter kann zum Beispiel folgenden Wert annehmen :

@TELEPHONE
Es wird nach denselben Kriterien wie im deutschen Telephonbuch sortiert, d.h.: Kleinbuchstaben werden wie Großbuchstaben behandelt, ä wird behandelt wie ae, ö wie oe, ü wie ue, ß wie ss, Ä wie Ae, Ö wie Oe und Ü wie Ue. Wenn sich Wörter nur durch die Groß-/Kleinschreibung unterscheiden, gilt die Reihenfolge: a, A, b, B, ... z, Z. Wenn sich Wörter nur durch die Umlautdarstellung oder durch ss/ß unterscheiden, gilt jeweils folgende Reihenfolge:

ae, ä, Ae, Ä, AE
oe, ö, Oe, Ö, OE
ue, ü, Ue, Ü, UE
ss, ß

@TELEPHONE nicht angegeben:
Es gilt folgende Sortierreihenfolge der Buchstaben:

bei *En_US.ASCII*:
wie in der ASCII-Tabelle (siehe *Tabellen und Verzeichnisse, Zeichensatz ISO 646*).

bei *En_GB.646* und *En_GB.88591*:
a, b, ... z, ... A, B, ... Z

bei *De_DE.646* und *De_DE.88591*:
Alle Kleinbuchstaben kommen vor allen Großbuchstaben (a, ... z, A, ... Z), wobei ä behandelt wird wie a, ö wie o, ü wie u, ß wie ss, Ä wie A, Ö wie O und Ü wie U. Wenn sich Wörter nur durch die Umlautdarstellung oder durch ss/ß unterscheiden, gilt die Reihenfolge: a, ä, b, c, ... , ss, ß, ... z, A, Ä, B, ... Z

Ist eine der Variablen *LC_CTYPE, LC_COLLATE, LC_TIME, LC_MONETARY, LC_NUMERIC, LC_MESSAGES* nicht definiert oder ist ihr die leere Zeichenkette zugewiesen, dann gilt für diese Variable als Standardwert der Wert von *LANG*.

Hat eine der Variablen *LC_CTYPE, LC_COLLATE, LC_TIME, LC_MONETARY, LC_NUMERIC, LC_MESSAGES* einen ungültigen Wert oder ist die Variable *LANG* nicht definiert bzw. leer, dann verhält sich das System, als wäre es nicht internationalisiert, d.h. es wird dann nach der Reihenfolge der Zeichen im ASCII-Zeichensatz sortiert; es gelten das amerikanische Datumsformat, englische Wochentags- und Monatsnamen etc.

3.5.2 Die internationalisierte Umgebung eines Programms

Ein internationalisiertes Programm erfragt zur Ablaufzeit sein Einsatzfeld aus der NLS-Datenbank. Dazu muß der Zugriff auf die NLS-Datenbank initialisiert werden. Dies geschieht mit der Funktion

setlocale(category,locale)

category

Konstante, die in der Include-Datei <*locale.h*> definiert ist und einen der folgenden Werte annehmen kann :

Konstante	Einfluß auf:
LC_ALL	Gesamte internationale Umgebung
LC_COLLATE	Verhalten der regulären Ausdrücke und die Funktionen, die in *strcoll()* und *strxfrm()* definiert sind.
LC_CTYPE	Verhalten der regulären Ausdrücke und die mit Zeichen arbeitenden Funktionen, die in *_tolower(),_toupper(), tolower(), toupper(), isalpha(), isupper(), islower(), isalnum(), isspace(), ispunct(), isprint(), isgraph()* und *iscntrl()* definiert sind.
LC_MESSAGES	Verhalten von Funktionen, die mit Meldungstexten oder mit Antworten auf ja/nein-Fragen arbeiten.
LC_MONETARY	Verhalten von Funktionen, die mit Währungen arbeiten.
LC_NUMERIC	Dezimalpunktzeichen für die Ein-/Ausgabe-Funktionen, die in *printf()* und *scanf()* definiert sind, und diein *gcvt(), strtod()* und *atof()* definierten Konvertierungsfunktionen.
LC_TIME	Verhalten der in *strftime()* definierten Zeitfunktionen.

locale

Das Argument *locale* ist ein Zeiger auf eine Zeichenkette, die den verlangten Wert von *category* in der folgenden Form enthält:

[sprache[_territorium[.zeichensatz]][@parameter]]

Ein fester Wert für *locale* ermöglicht es, ein *Locale*-spezifisches Programm zu schreiben.

Beispiel

Stellt man in einem Programm mit *setlocale()* die englische Sortierreihenfolge ein, dann sortiert dieses Programm nach der englischen Sortierreihenfolge, auch wenn in den Umgebungsvariablen eine andere Sortierreihenfolge definiert ist.

Für alle Werte von *category* werden *sprache,territorium* und *zeichensatz* definiert, während *parameter* für LC_ALL nicht definiert ist. Drei weitere Werte sind für *locale* definiert:

C

Dies ist die minimale Umgebung für die C-Übersetzung. Wenn *setlocale()* nicht aufgerufen wird, wird dieser Wert für *locale* eingesetzt.

NULL

Dies veranlaßt *setlocale()*, die aktuelle internationalisierte Umgebung abzufragen und zurückzugeben.

""

Dies setzt *category* beim Ablauf auf den Wert der entsprechenden Umgebungsvariablen. So kann man internationalisierte Programme erstellen.

Außerdem sind die folgenden Bibliotheksfunktionen festgelegt bzw. erweitert worden, um internationalisierte Programme zu entwickeln:

```
atof()          islower()       sprintf()
catclose()      isprint()       scanf()
catgets()       ispunct()       strcoll()
catopen()       isspace()       strerror()
fprintf()       isupper()       strtime()
fscanf()        nl_langinfo()   strtod()
gettxt()        perror()        strxfrm()
gcvt()          printf()        tolower()
isalnum()       regexp()        toupper()
isalpha()       scanf()         vfprintf()
iscntrl()       setlocale()     vprintf()
isgraph()       setlocale()     vsprintf()
```

Die folgenden Include-Dateien werden benutzt, um Definitionen und Deklarationen zum Umgang mit dem NLS bereitzustellen.

```
<ctype.h>
<langinfo.h>
<limits.h>
<locale.h>
<nl_types.h>
```

3.6 Meldungskataloge

Meldungskatalogsysteme erlauben es, Programmeldungen getrennt von der Programmlogik zu halten, diese Meldungen in verschiedene Sprachen zu übersetzen und sie zur Ablaufzeit je nach der vom Benutzer geforderten Sprache zum Programm zu binden.

SINIX V5.41 unterstützt zwei Arten von Meldungskatalogen:
- Meldungskataloge, die dem X/Open-Standard entsprechen (wie in SINIX V5.22)
- Meldungskataloge wie bei AT&T System V, Rel.4.0

3.6.1 Meldungskataloge, die dem X/Open-Standard entsprechen

Kommandos

gencat
: erzeugt einen binär kodierten Meldungskatalog aus einer oder mehreren Meldungstext-Quelldateien.

dumpmsg
: erzeugt aus einem binär kodierten Meldungskatalog eine lesbare Meldungstext-Quelldatei. Dies ist nützlich, wenn man z.B nur den binären Meldungskatalog hat und eine "neue Sprache" erzeugen oder etwas korrigieren möchte.

extract
: eignet sich zur Internationalisierung eines bereits bestehenden Programms hinsichtlich seiner Meldungstexte.

iecho
: gibt eine Meldung eines Meldungskatalogs auf die Standard-Ausgabe aus.

C-Bibliotheksfunktionen

catopen()
: sucht einen bestimmten Meldungskatalog im Dateispeicher und öffnet ihn für *catgets()* und *catclose()*.

catgets()
: holt Meldungen aus einem Meldungskatalog, der von *catopen()* geöffnet wurde.

catclose()
: schließt den Meldungskatalog wieder.

Format

Zur Übersetzung mit *gencat* müssen die Meldungstext-Quelldateien folgendes Format haben:

Die einzelnen Felder werden durch ein Leerzeichen getrennt. Alle anderen Leerzeichen werden als zum nächsten Feld gehörig betrachtet.

$set n kommentar
Diese Zeile identifiziert den Satz der folgenden Meldungen, bis ein weiteres *$set*, *$delset* oder das Dateiende auftritt. *n* ist die Nummer des Satzes und muß im Bereich [1,*{NL_SETMAX}*] liegen. Die Satznummern müssen in einer Datei aufsteigend erscheinen, brauchen aber nicht lückenlos zu sein. Jede Zeichenreihe, die nach der Satznummer kommt, wird als Kommentar aufgefaßt. Falls in einer Datei kein *$set* definiert ist, werden alle Meldungen einer Standardsatznummer *NL_SETD* zugewiesen. Der Wert von *NL_SETD* ist in der Include-Datei *nl_types.h* definiert.

$delset n kommentar
Diese Zeile löscht den Meldungssatz *n* aus dem bestehenden Meldungskatalog. *n* bezeichnet dabei die Satznummer. Jede Zeichenreihe, die nach der Satznummer kommt, wird als Kommentar aufgefaßt.

$ kommentar
Eine Zeile, die mit *$* gefolgt von einem Leerzeichen beginnt, wird als Kommentar behandelt.

m meldungstext
Das *m* bezeichnet die Meldungsnummer, die in dem Bereich [1,*{NL_MSGMAX}*] liegen muß. Der *meldungstext* wird im Meldungskatalog mit der vorher durch *$set* definierten Satznummer und der Meldungsnummer gespeichert. Wenn der *meldungstext* leer ist und der Meldungsnummer ein Leer- oder Tabulatorzeichen folgt, wird die leere Zeichenreihe im Meldungskatalog gespeichert. Falls auf die Meldungsnummer weder ein Leer- oder Tabulatorzeichen noch ein *meldungstext* folgt, wird die Meldung mit dieser Meldungsnummer aus der Datei gelöscht. Die Meldungsnummern müssen in einer Datei in aufsteigender Reihenfolge erscheinen, brauchen aber nicht lückenlos zu sein. Die Länge von *meldungstext* muß in dem Bereich [0,*{NL_TEXTMAX}*] liegen.

$quote c
Diese Zeile definiert ein optionales Begrenzungszeichen *c*. Dieses kann vor und nach dem Meldungstext gesetzt werden, um vorangehende oder abschließende Leerzeichen beziehungsweise leere Meldungstexte sichtbar zu machen.

Leerzeilen in einer Meldungstext-Quelldatei werden ignoriert. Die Meldungstexte können die folgenden Sonderzeichen und Escape-Sequenzen enthalten :

Beschreibung		Symbol	Sequenz
newline	Neue-Zeile-Zeichen	NL(LF)	\n
horizontal tab	Horizontaltabulatorzeichen	HT	\t
vertical tab	Vertikaltabulatorzeichen	VT	\v
backspace	Korrekturtaste	BS	\b
carriage return	Wagenrücklauf	CR	\r
form-feed	Seitenvorschub	FF	\f
backslash	Gegenschrägstrich	\	\\
octal bit pattern	Bitmuster oktal	ddd	\ddd
hex bit pattern	Bitmuster hexadezimal	0xnn	\0xnn

Die Escape-Sequenz *ddd* besteht aus einem Gegenschrägstrich (backslash) gefolgt von ein, zwei oder drei oktalen Ziffern, die das gewünschte Zeichen beschreiben.
Die Escape-Sequenz *0xnn* besteht aus einem Gegenschrägstrich (backslash) gefolgt von ein oder zwei hexadezimalen Ziffern, die das gewünschte Zeichen beschreiben.
Der Gegenschrägstrich gefolgt von einem Neue-Zeile-Zeichen wird auch benutzt, um eine Zeichenreihe in der nächsten Zeile fortzusetzen.

Also beschreiben die nächsten zwei Zeilen eine Meldung:

```
1 Diese Zeile \
  geht in der nächsten Zeile weiter.
```

und ist äquivalent zu:

```
1 Diese Zeile geht in der nächsten Zeile weiter.
```

Beispiel einer Meldungstext-Quelldatei:

```
$quote "
$set 1
1 " Dies ist die Meldung 1 "
2 " Dies ist die Meldung 2 "
5 " Dies ist die Meldung 5 "
$set 3
3 " Dies ist die Meldung 3 "
4 " Dies ist die Meldung 4 "
```

Zugriff

Meldungskataloge werden geöffnet und gebrauchsfertig gemacht, indem man die Bibliotheksfunktion *catopen()* aufruft, die den gewünschten Meldungskatalog lokalisiert. Dies geschieht nach den Such- und Namensgebungskonventionen, die in der Umgebungsvariablen *NLSPATH* definiert sind.

Beispiel

```
nl_catd=catopen("katalog_name",0);
```

catopen() gibt einen Katalogdeskriptor *nl_catd* zurück. Dieser wird von nachfolgenden Aufrufen der Funktion *catgets()* benutzt, um den vorbereiteten Meldungskatalog zu identifizieren. Der Meldungskatalog wird von der Funktion *catclose()* wieder geschlossen.

Die Umgebungsvariable NLSPATH

Mit dieser Variablen können Sie, ähnlich wie bei *PATH* (siehe *sh*), die Pfadnamen der Meldungskataloge angeben, aus denen internationalisierte Programme zur Laufzeit ihre Meldungstexte holen. Sie können auch mehrere Pfadnamen nacheinander - durch Doppelpunkt : getrennt - angeben. Die Pfade werden dann von links nach rechts durchsucht. Im Pfadnamen können Sie die folgenden Metazeichen verwenden:

Metazeichen	wird expandiert zu
%L	$LANG
%N	1. Parameter der Funktion catopen(), d.i. in der Regel der Programm- bzw. Kommandoname, zu dem der Meldungskatalog gesucht wird

Die Zeichenketten :: und :.: im Wert von *NLSPATH* bewirken, daß im aktuellen Dateiverzeichnis nach einer Datei *%N* gesucht wird. Beginnt der Wert von *NLSPATH* mit einem Doppelpunkt :, dann wird zuerst im aktuellen Dateiverzeichnis nach der Datei *%N* gesucht.

Beispiel

```
NLSPATH=:/usr/lib/nls/msg/%L/%N.cat:/mein_dvz/%N.cat
```

Wenn *LANG* den Wert *deutsch* hat und das Kommando *date* aufgerufen wird, wird nach dem Meldungskatalog *date.cat* zuerst im aktuellen Dateiverzeichnis, dann im Dateiverzeichnis */usr/lib/nls/msg/De*, dann im Dateiverzeichnis */mein_dvz* gesucht.

Kann der Meldungskatalog nicht unter den durch *NLSPATH* angegebenen Pfadnamen gefunden werden, dann wird standardmäßig nach

```
/usr/lib/nls/msg.products/%l/%N.cat
```

gesucht und dann nach

```
/usr/lib/nls/msg/%l/%N.cat
```

3.6.2 Meldungskataloge wie bei AT&T System V, Rel.4.0

Kommandos

mkmsgs
: erstellt eine Meldungsdatei, auf die die Textsuch-Kommandos *gettxt, srchtxt, exstr* und die C-Funktion *gettxt()* zugreifen können. Diese Datei hat ein anderes Format als die, die mit *gencat* und *dumpmsg* bearbeitet wird.

gettxt
: sucht in einer von *mkmsgs* erzeugten Meldungsdatei die einer Meldungsnummer zugeordnete Zeichenkette.

exstr
: eignet sich zum Erstellen von internationalisierten Programmen.

srchtxt
: zeigt den Inhalt von Meldungskatalogen an oder sucht nach speziellen Zeichenketten in den Meldungskatalogen.

C-Bibliotheksfunktion

gettxt()
: verhält sich wie das Kommando *gettxt*.

Arbeitsschritte für den Programmierer

Die folgenden Schritte sollten Sie unternehmen, um als Programmierer ein Programm zur internationalisierten Anwendung zu erstellen:

- Trennen der Meldungstexte von der Programmlogik mittels der Funktionen *catopen(), catgets()* und *catclose()* bzw. *gettxt()*. Diese binden die Meldungen zur Ablaufzeit des Programms ein.
- Erstellen einer Meldungstext-Quelldatei nach einem bestimmten Format (siehe unten, *Format*).
- Erzeugen eines Meldungskatalogs aus der Meldungstext-Quelldatei mittels der Kommandos *gencat* bzw. *mkmsgs*.

Daraus können Sie nun andere internationalisierte Umgebungen erstellen, indem Sie die Meldungstext-Quelldatei in verschiedene Sprachen übertragen und diese Dateien mit *gencat* bzw. *mkmsgs* übersetzen.

Format

Zur Übersetzung mit *mkmsgs* müssen die Meldungstext-Quelldateien folgendes Format haben:

- die Meldungen sind weder in Sätze unterteilt, noch durch Meldungsnummern gekennzeichnet.
- jede Zeichenreihe (Meldung) muß in einer neuen Zeile stehen.
- die Meldungen sind, im Gegensatz zu dem anderen Format, aufsteigend und lückenlos geordnet, da die Zeilennummern zur Identifikation der Meldungen benutzt werden.
- nicht druckbare Zeichen sind durch Escape-Folgen dargestellt. (siehe oben, *Meldungskataloge, die dem X/Open-Standard entsprechen, Format*)

Beispiel einer Meldungstext-Quelldatei:

```
Dies ist die Meldung 1
Dies ist die Meldung 2
Dies ist die Meldung 3
Dies ist die Meldung 4
```

Zugriff

Der Zugriff auf einen Meldungskatalog erfolgt mit Hilfe der Bibliotheksfunktion *gettxt()*. Sie sucht aus der angegebenen Datei die Meldung mit der angegebenen *meldungsnummer* heraus. Die Datei wird im Dateiverzeichnis */usr/lib/locale/Locale/LC_MESSAGES* gesucht. Der Sprachraum auf den zugegriffen wird, wird über die Umgebungsvariable *LC_MESSAGES* bestimmt. Ist diese nicht gesetzt, ist der Wert von *LANG* ausschlaggebend. Wenn auch diese nicht gesetzt ist, so wird standardmäßig das Dateiverzeichnis */usr/lib/locale/C/LC_MESSAGES* durchsucht, welches Standardmeldungstexte enthält. Die *meldungsnummer* gibt die Zeilennummer an, in der die gesuchte Zeichenreihe steht. Die *gettxt()*-Funktion ist vergleichbar mit *catgets()* in dem anderen Meldungskatalogsystem. Es fehlen hier aber Funktionen, die mit *catopen()* und *catclose()* vergleichbar sind.

Beispiel

```
gettxt("UX:10","fehlertext");
```

Hier wird in der Datei *UX*, die unter */usr/lib/locale/Locale/LC_MESSAGES* liegt, nach der Meldung in der Zeile *10* gesucht. Falls die Meldung nirgends gefunden wird, wird die Zeichenreihe *fehlertext* ausgegeben.

4 Kommandos

acctcom
Suchen und Drucken von Prozeßabrechnungsstatistiken

acctcom liest Daten, die in einem bestimmten Format vorliegen müssen (siehe *acct* [7]), von den als Argumente angegebenen Dateien, von der Standard-Eingabe oder aus einer Systemdatei und bereitet die darin enthaltene Prozeßinformation auf. Die aufbereitete Information wird auf die Standard-Ausgabe geschrieben.

acctcom gibt nur Informationen über bereits beendete Prozesse aus.
Informationen über aktive Prozesse erhalten Sie mit dem Kommando *ps*.

acctcom[_option]...[_datei]...

Keine Option angegeben
Auflistung von Standardinformationen zu den einzelnen Prozessen. Ein Prozeßdatensatz besteht aus Einträgen zu folgenden Feldern: COMMAND NAME, USER, TTYNAME, START TIME, END TIME, REAL (SEC), CPU (SEC), MEAN SIZE.

Die Reihenfolge der Ausgabe entspricht dem *first in - first out - Prinzip*; die älteren Prozeßinformationen werden zuerst ausgegeben.

option

-a
(a - average) Anzeigen von Mittelwertsstatistiken der selektierten Prozesse. Diese Statistik wird ausgegeben, nachdem die Prozeßdatensätze aufgelistet wurden.

-b
(b - backwards) Die Lesereihenfolge wird umgedreht, die aktuellen Kommandos werden zuerst angezeigt. Diese Option hat keine Auswirkung, wenn von der Standard-Eingabe gelesen wird.

-f
Die Ausgabe enthält zusätzliche Spalten für die fork- und exec-Flags sowie für den Ende-Status des Systems. Die numerischen Ausgaben werden oktal dargestellt.

-h
(h - hog factor) Anstatt des gemittelten Speicherbedarfs des Prozesses wird der Anteil der gesamten zur Verfügung stehenden CPU-Zeit angegeben, der vom Prozeß zur Laufzeit verbraucht wurde. Dieser Wert wird als Monopolisierungsfaktor (hog factor) bezeichnet und wird folgendermaßen berechnet:
Gesamt-CPU-Zeit/verbrauchte Zeit.

-i
Die Ausgabe enthält zusätzliche Spalten mit Eingabe- und Ausgabezählern.

-k
Zeigt die kumulierte Speichergröße in Kilobyte-Segmenten, die ein Prozeß insgesamt pro Laufzeitminute beansprucht hat.

-m
(m - mean core size) Der Mittelwert des Hauptspeicherbedarfs wird angezeigt. Diese Option ist standardmäßig eingeschaltet.

-r
Ausgabe des CPU Faktors (user-time/(system-time + user-time)).

-t
Die CPU-Zeit, die das System verbraucht, wird getrennt von der Zeit aufgelistet, die der Benutzer verbraucht.

-v
Es werden keine Spaltenüberschriften ausgegeben.

-l␣line
Nur die Prozesse, die über den Bildschirm mit dem Geräteeintrag /dev/term/*line* verbunden sind, werden ausgegeben.

-u␣benutzer
(u - user) Nur die Prozesse, die dem Benutzer *benutzer* gehören, werden angezeigt. *benutzer* kann angegeben werden als
- Benutzernummer
- Benutzerkennung (die vom Kommando selbst in eine Benutzernummer konvertiert wird)
- Nummernzeichen # (für Prozesse, die Systemverwalter-Privilegien haben)
- Fragezeichen (für Prozesse, die unter unbekannten Benutzernummern ablaufen)

-g␣gruppe
(g - group) Nur die Prozesse, die der Gruppe *gruppe* gehören, werden angezeigt. Als *gruppe* kann eine Gruppennummer oder ein Gruppenname angegeben werden.

Die Zeitangabe *time* in den folgenden Optionen muß im Format *stunde[:minute[:sekunde]]* angeben werden.
Falls der Wert von *time* größer ist als die aktuelle Zeit, wird *time* als Vortageswert interpretiert.

-s␣time
Nur die Prozesse, die zum Zeitpunkt *time* oder danach existieren, werden selektiert.

-e␣time
Nur die Prozesse, die bis zum Zeitpunkt *time* oder davor existierten, werden selektiert.

-S␣time
Nur die Prozesse, die zum Zeitpunkt *time* oder danach gestartet wurden, werden selektiert.

-E␣time
Nur die Prozesse, die zum Zeitpunkt *time* oder davor beendet wurden, werden selektiert.

Werden die Optionen *-E* und *-S* zusammen angegeben und wird beiden der gleiche Wert zugeordnet, so werden die Prozesse angezeigt, die zum Zeitpunkt *time* existiert haben.

-n␣muster
Nur die Kommandos, deren Namen dem angegebenen *muster* entsprechen, werden angezeigt. Für *muster* können Sie reguläre Ausdrücke verwenden (siehe *regcmp* [19]), allerdings steht das Pluszeichen + für ein- oder mehrmaliges Auftreten des Musters.

-q
Nur die Mittelwertsstatistiken wie in der Option *-a* werden angegeben. Es erfolgt keine Ausgabe der Prozeßdatensätze.

-o␣datei
Die Informationen werden nicht auf die Standard-Ausgabe ausgegeben, sondern in der Datei *datei* abgelegt. Das Format entspricht dem Eingabedatenformat.

-H␣faktor
(H - Hog factor) Es werden nur die Prozesse angezeigt, deren Monopolisierungsfaktor größer ist als der angegebene Wert *factor*. Zur Berechnung des Monopolisierungsfaktors siehe Option *-h*.

-O␣sekunden
Es werden nur die Prozesse angezeigt, deren CPU-Systemzeit größer als *sekunden* ist.

-C␣sekunden
Es werden nur die Prozesse angezeigt, deren gesamte CPU-Zeit (Systemzeit + Benutzerzeit) größer als *sekunden* ist.

-I␣anzahl-zeichen
Es werden nur die Prozesse angezeigt, die mehr als *anzahl-zeichen* Zeichen übertragen.

datei
Name der Datei, die die Prozeßinformationen für die Aufbereitung durch *acctcom* enthält. Das Format der Informationen muß dem in *acct* [7] beschriebenen Format entsprechen.

Es können mehrere Dateien angegeben werden. Sie werden der angegebenen Reihenfolge nach gelesen.

datei nicht angegeben:
Wenn die Standard-Eingabe mit der kontrollierenden Datensichtstation verbunden ist, aber keine Prozeßinformationen enthält, oder wenn die Standard-Eingabe bei Hintergrundprozessen der Shell mit */dev/null* verbunden ist, wird die Datei */var/adm/pacct* gelesen, die normalerweise die aktuellen Prozeßinformationen enthält. Bei einem ausgelasteten System kann es vorkommen, daß mehrere Dateien benötigt werden, die dann in der Datei */var/adm/pacct*inkrement abgelegt werden, wobei *inkrement* eine fortlaufende Numerierung ist.

Andernfalls wird von der Standard-Eingabe gelesen.

Ausgabe

Im folgenden werden die Überschriften der Spalten und die Bedeutung der Spalten in der Ausgabe von *acctcom* erläutert. Die Buchstaben hinter den Spaltenüberschriften bezeichnen die Option, bei der die entsprechende Spalte in der Ausgabe erscheint. *alle* bedeutet, daß die Spalte bei allen Optionen erscheint.
Jeder Datensatz der Ausgabe steht für die Ausführung eines Prozesses.

COMMAND NAME (alle)
Name des ausgeführten Kommandos, zu dem die folgende Prozeßinformation gehört. Ein Nummernzeichen # hinter dem Kommandonamen bedeutet, daß das Kommando mit Systemverwalter-Privilegien ausgeführt wurde.

USER (alle)
Benutzerkennung, von der aus der Prozeß gestartet wurde.

TTYNAME (alle)
Die kontrollierende Datensichtstation des Prozesses. Falls der Prozeß nicht mit einem bekannten Gerät verbunden ist, wird ein Fragezeichen ? ausgegeben.

START TIME (alle)
Zeitpunkt, zu dem der Prozeß gestartet wurde.

END TIME (alle)
Zeitpunkt, zu dem der Prozeß beendet wurde.

REAL (SEC) (alle)
Gesamtlaufzeit des Prozesses in Sekunden.

CPU (SEC) (alle)
Systemzeit des Prozesses in Sekunden.

MEAN SIZE(K) (alle)
Gemittelte Speichergröße des Prozesses.

F (f)
Die fork/exec-Flags, 1 steht für fork ohne exec.

STAT (f)
System-Ende-Status

HOG FACTOR (h,H)
Monopolisierungsfaktor

KCORE MIN (k)
Kumulierte Speichergröße in Kilobyte-Segmenten, die ein Prozeß insgesamt pro Laufzeit-Minute beansprucht hat.

CPU FACTOR (r)
CPU-Faktor

CHARS TRNSFD (I)
Anzahl der übertragenen Zeichen

BLOCKS READ
Gesamtanzahl der gelesenen und geschriebenen Blöcke

DATEIEN

/etc/passwd
Datei mit Benutzerkennungen

/etc/group
Datei mit Gruppenkennungen

*/var/adm/pacct*inkrement
Datei mit den aktuellen Prozeßinformationen. Bei ausgelasteten Systemen kann es mehrere Dateien geben, die fortlaufend numeriert werden. Zu unterscheiden sind diese Dateien durch das Suffix *inkrement*.

BEISPIELE

Sie wollen Standardinformationen zu allen aktuellen, aber abgeschlossenen Prozessen ausgeben:

```
$ acctcom
ACCOUNTING RECORDS FROM:  Wed Feb  6 10:22:30 1991
COMMAND                          START    END          REAL      CPU     MEAN
NAME       USER     TTYNAME      TIME     TIME        (SECS)   (SECS) SIZE(K)
#accton    root      tty001        10:22:30 10:22:30     0.13     0.03     8.17
turnacct   root      tty001        10:22:29 10:22:30     1.27     0.12     0.92
rm         root      tty001        10:22:30 10:22:30     0.03     0.03     1.50
rm         root      tty001        10:22:31 10:22:31     0.02     0.02     2.25
startup    root      tty001        10:22:28 10:22:30     2.51     0.20     0.53
acctcom    root      tty001        10:23:21 10:23:21     0.39     0.20     1.43
getopt     root      tty001        10:24:09 10:24:09     0.05     0.03     8.00
expr       root      tty001        10:24:09 10:24:09     0.07     0.03     0.67
...
```

Soll die Ausgabe seitenweise erfolgen, verbinden Sie einfach das Kommando *acctcom* über eine Pipe mit dem Kommando *pg*:

```
$ acctcom | pg
...
```

Sie wollen Prozessinformationen über alle Prozesse, deren CPU-Systemzeit größer als eine Sekunde ist, in der Datei *langeweile* ablegen. Danach sollen die in der Datei abgelegten Informationen ausgegeben werden:

```
$ acctcom -O 1 -o langeweile

ACCOUNTING RECORDS FROM:  Wed Feb  6 10:22:30 1991

$ acctcom langeweile

ACCOUNTING RECORDS FROM:  Wed Feb  6 10:24:10 1991
COMMAND                          START    END          REAL      CPU     MEAN
NAME        USER     TTYNAME     TIME     TIME        (SECS)   (SECS) SIZE(K)
find        root      pts/0       10:24:10 10:25:48    98.40     1.66     0.17
acctdusg    root      pts/0       10:24:10 10:26:21   131.20     1.36     0.24
```

SIEHE AUCH

ps, *su*
acct, *regcmp* [19]
acct, *acctcms*, *acctcon*, *acctmerg*, *acctprc*, *fwtmp*, *runacct*, *acct*, *utmp* [7]

ar
Bibliotheken verwalten (archive maintainer)

ar verwaltet Bibliotheken.
Im einzelnen können Sie mit *ar* folgende Verwaltungsaufgaben erledigen:

Verwaltungsaufgabe	Hauptoption
Bibliothek erstellen	-q oder -r
Datei schnell in die Bibliothek eintragen	-q
Datei ersetzen bzw. eintragen	-r
Datei aus der Bibliothek entfernen	-d
Datei in der Bibliothek verschieben	-m
Dateiinhalt ausgeben	-p
Inhaltsverzeichnis der Bibliothek ausgeben	-t
Datei aus der Bibliothek herauskopieren	-x

ar[␣**-V**]␣hauptoption[zusatzoption].. [position]␣bibliothek[␣datei]..

-V
ar gibt seine Versionsnummer auf die Standard-Fehlerausgabe aus.

Hauptoptionen

Bei einem *ar*-Aufruf müssen Sie genau eine Hauptoption angeben (*-d, -m, -p, -q, -r, -t* oder *-x*). Der Hauptoption können eine oder mehrere Zusatzoptionen folgen. Die Zusatzoptionen folgen ohne Bindestrich und Leerzeichen unmittelbar auf die Hauptoption.

-d
(d - delete) *ar* entfernt die angegebenen Dateien aus der Bibliothek. Sind keine Dateien angegeben, wird *keine* Datei entfernt.

-m
(m - move) *ar* verschiebt die angegebenen Dateien innerhalb der Bibliothek.

Mit *position*:
Die Dateien werden hinter (*a*) bzw. vor (*b* oder *i*) der Datei *posdatei* eingefügt (siehe *position*).

Ohne *position*:
Die Dateien werden am Ende der Bibliothek eingefügt.

-p
(p - print) *ar* gibt den Inhalt der angegebenen Dateien aus. Sind keine Dateien angegeben, gibt *ar* den Inhalt aller Dateien aus.

-q

(q - quickly) *ar* trägt die angegebenen Dateien "schnell" in die Bibliothek ein, d.h.:

- Wenn die Bibliothek bereits existiert, hängt *ar* die Dateien ans Ende der Bibliothek an, ohne zu überprüfen, ob die Dateien bereits in der Bibliothek vorhanden sind.
- Wenn die Bibliothek noch nicht existiert, wird sie erstellt.

Angaben für *position* sind nicht erlaubt.
Die Option *-q* ist sinnvoll, wenn Sie große Bibliotheken schrittweise erstellen wollen.

-r

(r - replace) Diese Option hat drei verschiedene Wirkungen, abhängig davon, ob die angegebene Bibliothek existiert und die angegebenen Dateien enthält:

- Wenn die Bibliothek existiert und die Dateien enthält, ersetzt *ar* die angegebenen Dateien.
- Wenn die Bibliothek existiert und eine der angegebenen Dateien nicht enthält, trägt *ar* diese Datei in die Bibliothek ein.
- Wenn die Bibliothek noch nicht existiert, erstellt *ar* die Bibliothek aus den angegebenen Dateien.

Mit Zusatzoption *u*:
ar ersetzt eine Datei in der Bibliothek nur dann, wenn die beim *ar*-Aufruf angegebene Datei neueren Datums ist als die Version, die in der Bibliothek steht. (Das Datum bezieht sich auf den Zeitpunkt der letzten Änderung.)

Mit *position*:
Dateien, die noch nicht in der Bibliothek enthalten sind, werden hinter (*a*) bzw. vor (*b* oder *i*) der Datei *posdatei* eingefügt (siehe *position*).

Ohne *position*:
Dateien, die noch nicht in der Bibliothek enthalten sind, werden am Ende der Bibliothek eingefügt.

-t

(t - table) *ar* gibt ein Inhaltsverzeichnis der Bibliothek aus. Sind keine Dateien angegeben, listet *ar* alle Dateien der Bibliothek auf, sonst nur die angegebenen Dateien.

-x

(x - extract) *ar* kopiert die angegebenen Dateien aus der Bibliothek heraus. Sind keine Dateien angegeben, kopiert *ar* alle Dateien heraus. Die Bibliothek wird dadurch nicht verändert.

Zusatzoptionen

c

(c - create) Die Standard-Meldung von *ar* beim Erstellen einer Bibliothek wird unterdrückt.

s

(s - symbol table) *ar* erstellt die Symboltabelle der Bibliothek neu (auch dann, wenn die Bibliothek nicht verändert wird).
Diese Option ist sinnvoll, um die Symboltabelle nach einem *strip*-Aufruf wiederherzustellen (siehe *strip* [19]).

u

(u - update) Siehe Hauptoption *-r*.

v

(v - verbose) *ar* meldet, welche Datei es gerade in die Bibliothek einträgt, in der Bibliothek verschiebt, aus der Bibliothek entfernt usw.. Wenn Sie *v* zusammen mit *-t* verwenden, gibt *ar* ähnlich wie *ls -l* (siehe *ls*) ausführliche Informationen über die Dateien aus.

position
position legt fest, wo eine Datei in die Bibliothek eingefügt wird. *position* kann sein:

a␣posdatei
ar fügt die Dateien hinter *posdatei* ein.

b␣posdatei
i␣posdatei
ar fügt die Dateien vor *posdatei* ein.

posdatei ist der Name einer Datei, die in der Bibliothek steht.

bibliothek
Name der Bibliothek, die erstellt bzw. bearbeitet werden soll.

datei
Name der Datei, die aufgelistet, in die Bibliothek eingetragen, aus der Bibliothek entfernt, aus der Bibliothek herauskopiert, in der Bibliothek verschoben bzw. deren Inhalt ausgegeben werden soll.
Sie können mehrere Dateien angeben. Geben Sie bei einem *ar*-Aufruf mit Hauptoption *-q* oder *-r* dieselbe Datei mehrmals an, trägt *ar* diese Datei auch mehrmals in die Bibliothek ein. Wenn mehrere Dateien mit gleichem Namen in der Bibliothek enthalten sind, und Sie *ar* ohne Positionsangabe verwenden, wird auf die erste Datei mit diesem Namen zugegriffen (und nicht z.B. auf die neueste oder älteste etc).

Aufbau einer Bibliothek

Eine Bibliothek ist eine Datei, in der mehrere Dateien zusammengefaßt sind. Gemäß Konvention endet der Name einer Bibliothek mit *.a*. Der Sinn einer Bibliothek besteht darin, Dateien zusammenzufassen, die bzgl. ihrer Verwendung eine Gruppe bilden. Die Verwendung von Bibliotheken erleichtert die Verwaltung der Dateien, da Sie oft nur die Bibliothek angeben müssen, statt alle Elemente einzeln aufzuführen.
Sehr häufig sind die Elemente einer Bibliothek Objektmodule, die üblicherweise zu einem Programm oder Programmsystem gehören.

Die Magic-Zeichenkette (magic string) und der Dateivorspann (header), die *ar* benutzt, bestehen aus druckbaren ASCII-Zeichen. Enthält die Bibliothek nur druckbare Dateien, so ist die ganze Bibliothek druckbar.
Wenn *ar* eine Bibliothek erzeugt, dann erzeugt es den Dateivorspann (header) in einem Format, das über alle Rechner portabel ist (zum portablen Format und Aufbau einer Bibliothek siehe *ar* [19]).

Enthält die Bibliothek mindestens eine Objektdatei, erstellt und verwaltet *ar* eine Symboltabelle. Der Binder *ld* benutzt die Symboltabelle, um bei mehrfach notwendigem Durchlaufen der Bibliothek dieses Durchlaufen zu beschleunigen. Die Symboltabelle steht in einer eigenen Datei, die immer die erste Datei in der Bibliothek ist. Auf diese Datei können Sie nicht zugreifen, und sie wird auch nicht wie andere Bibliotheksdateien aufgelistet oder ausgegeben. Wird die Bibliothek mit *ar* neu erzeugt oder aktualisiert, dann erstellt *ar* auch die Symboltabelle neu. *ar* mit Option *-s* erstellt ebenfalls die Symboltabelle neu.

BEISPIELE

1. Eine Bibliothek schnell erstellen:

```
$ ar -qv archiv.a atoi.o itoa.o
ar: creating archiv.a
a - atoi.o
a - itoa.o
```

   ar erstellt die Bibliothek *archiv.a* aus den Dateien *atoi.o* und *itoa.o*. Die Zusatzoption *v* bewirkt, daß *ar* die Namen der eingetragenen Dateien ausgibt.

2. Eine neue Datei an einer angegebenen Stelle in eine Bibliothek einfügen:

```
$ ar -rvb atoi.o archiv.a atof.o
a - atof.o
```

   ar kopiert *atof.o* vor *atoi.o* in die Bibliothek *archiv.a*.

3. Inhaltsverzeichnis der Bibliothek ausgeben:

```
$ ar -tv archiv.a
rw-r--r-- 104/     1      2276 Jul 13 12:17 1990 atof.o
rw-r--r-- 104/     1       759 Jul 13 12:17 1990 atoi.o
rw-r--r-- 104/     1      1280 Jul 13 12:17 1990 itoa.o
```

4. Eine Datei aus der Bibliothek holen:

```
$ ar -xv archiv.a atoi.o
x - atoi.o
```

Die Datei *atoi.o* wird aus der Bibliothek *archiv.a* in das aktuelle Dateiverzeichnis kopiert. Die Kopie heißt auch *atoi.o*.

SIEHE AUCH

ld, *lorder*, *strip*, *a.out* [19]
ar [19], [7]

at
Kommandos zu einem späteren Zeitpunkt ausführen

Das Kommando *at*

- liest Kommandos von der Standard-Eingabe und führt sie zu einem späteren Zeitpunkt, den Sie angeben, aus (Format 1)
- liest Kommandos aus einem Shell-Skript und führt sie zu einem späteren Zeitpunkt, den Sie angeben, aus (Format 2)
- gibt auf die Standard-Ausgabe aus, welche mit *at* oder *batch* (siehe *batch*) erteilten Kommandoaufträge noch nicht bearbeitet wurden (Format 3)
- löscht Kommandoaufträge, die mit *at* oder *batch* erteilt wurden (Format 4)

Wenn Sie die Standard-Ausgabe und Standard-Fehlerausgabe der auszuführenden Kommandos nicht umgelenkt haben, wird Ihnen die Ausgabe von Format 1 und Format 2 mit *mail* geschickt. Die Umgebungsvariablen, das aktuelle Dateiverzeichnis, die für neue Dateien gültigen Zugriffsrechte (siehe *umask*) und die maximal zulässige Dateigröße (siehe *ulimit*) bleiben erhalten. Offene Dateien und Prioritäten werden nicht vererbt. Das Kommando *trap* (eingebautes Shell-Kommando zum Abfangen von Signalen) wird aufgehoben.
at schreibt die Auftragsnummer und die angegebene Ausführungszeit auf die Standard-Fehlerausgabe. Aufträge, die mit *at* erteilt werden, werden auch dann abgearbeitet, wenn der Auftraggeber sich vom System abgemeldet hat.

Vor dem Aufruf beachten

Wenn die Datei */etc/cron.d/at.allow* existiert, dann dürfen Sie das Kommando *at* nur dann aufrufen, wenn Ihre Benutzerkennung in dieser Datei steht.
Wenn die Datei */etc/cron.d/at.allow* nicht existiert, dann dürfen Sie das Kommando *at* nur dann aufrufen, wenn Ihre Benutzerkennung *nicht* in der Datei */etc/cron.d/at.deny* steht.
Wenn weder */etc/cron.d/at.allow* noch */etc/cron.d/at.deny* existieren, dann darf nur der Systemverwalter *at* aufrufen.

Existiert z.B. nur die leere deny-Datei, so dürfen alle Benutzer *at* aufrufen.

Die allow/deny-Dateien darf nur der Systemverwalter anlegen und ändern. Sie enthalten pro Zeile eine Benutzerkennung.

```
at␣[-m]␣[-qwarteschlange]␣zeit[␣datum][␣+inkrement] ↵                Format 1
kommando... ↵
END
at␣-f␣skript␣[-qwarteschlange]␣[-m]␣zeit[␣datum][␣+inkrement]        Format 2
at␣-l[␣auftragsnummer]...                                            Format 3
at␣-r␣auftragsnummer␣...                                             Format 4
```

Format 1: Kommandos zu einer späteren Zeit ausführen

at␣[**-m**]␣[**-q**warteschlange]␣zeit[␣datum][␣+inkrement] ↵
kommando... ↵
END

-m
Wenn sich ein Auftrag beendet hat, schickt *at* dem Benutzer über *mail* eine entsprechende Nachricht. Dies geschieht jedoch nur dann, wenn der Auftrag nicht bereits das Senden einer Nachricht veranlaßt hat.

-qwarteschlange
Durch die Option *-q* wird die Warteschlange im Dateiverzeichnis */var/spool/cron* spezifiziert, in die der Auftrag eingereiht werden soll.

warteschlange kann sein:

a
für die standardmäßige Warteschlange für Aufträge des Kommandos *at*.

b
für die standardmäßige Warteschlange für Aufträge des Kommandos *batch*.

c
für die standardmäßige Warteschlange für Aufträge des Kommandos *crontab*.

zeit
ziffern[suffix] oder sondername

ziffern

[h]h	Eine und zwei Ziffern werden als Stunden interpretiert.
hhmm	Vier Ziffern werden als Stunden und Minuten interpretiert.
[h]h:[m]m	Durch Doppelpunkt : getrennte Ziffern werden als Stunden und Minuten interpretiert.

suffix

am	Interpretation als vor 12 Uhr mittags
pm	Interpretation als nach 12 Uhr mittags
ohne am, pm	Interpretation im 24-Stunden-Format
zulu[am][pm]	Interpretation als Greenwich Meantime

sondername

noon	mittags
midnight	mitternachts
now	jetzt *at now* ohne *inkrement* führt allerdings zu der Fehlermeldung "Too late" (zu spät).

datum
monat_tag[,jahr] oder
wochentag[_**nextweek**] oder
spezieller tag

monat

jan	Januar
feb	Februar
mar	März
apr	April
may	Mai
jun	Juni
jul	Juli
aug	August
sep	September
oct	Oktober
nov	November
dec	Dezember
nextmonth	der auf den aktuellen Monat folgende Monat

tag
Zahl zwischen 1 und 31, entsprechend der Länge des Monats

jahr
Jahreszahl, für die die Datumsangabe gelten soll

nextyear das auf das aktuelle Jahr folgende Jahr

jahr nicht angegeben:
Wenn die Datumsangabe vor dem aktuellen Datum liegt, geht *at* vom nächsten Jahr, sonst vom aktuellen Jahr aus.

wochentag

mon[day]	Montag
tue[sday]	Dienstag
wed[nesday]	Mittwoch
thu[rsday]	Donnerstag
fri[day]	Freitag
sat[urday]	Samstag
sun[day]	Sonntag

nextweek
die auf die aktuelle Woche folgende Woche

spezieller tag

today	heute
tomorrow	morgen
nextday	der auf den aktuellen Tag folgende Tag

Die Angabe *nextday* bedeutet, daß der Auftrag einen Tag später ausgeführt wird. Wenn die Zeitangabe vor der aktuellen Uhrzeit liegt, interpretiert *at* den nächsten Tag als aktuellen Tag. Wird z.B. am 1.7.90 um 11:00 Uhr der Auftrag *at 10 nextday* erteilt, so wird er erst am 3.7.90 um 10:00 Uhr ausgeführt. *at 14 nextday* bewirkt hingegen, daß die Ausführung am 2.7.90 um 14:00 Uhr erfolgt.

datum nicht angegeben:
entspricht der Angabe *today*, wenn die Zeitangabe (gerundet auf die Minute) nach der aktuellen Uhrzeit liegt.

entspricht der Angabe *tomorrow*, wenn die Zeitangabe (gerundet auf die Minute) vor der aktuellen Uhrzeit liegt.

entspricht der Angabe *now*, wenn die Zeitangabe (gerundet auf die Minute) gleich der aktuellen Uhrzeit ist.

+inkrement
inkrement ist eine positive ganze Zahl, auf die eine der folgenden Zeiteinheiten folgen muß:

minute[s]	Minute(n)
hour[s]	Stunde(n)
day[s]	Tag(e)
week[s]	Woche(n)
month[s]	Monat(e)
year[s]	Jahr(e)

Beispiel

at können Sie u.a. in folgenden Formen angeben:

```
at 0815am jan 24
at 8:15am jan 24
at 5pm friday
at now +1hour
```

kommando
Beliebiges Kommando oder Shell-Prozedur. Sie können *kommando* mehrmals, jeweils durch Strichpunkt ; oder Neue-Zeile-Zeichen getrennt, angeben. Eine so entstandene Kommandoliste läuft unter *einer* Auftragsnummer.

Format 2: Shell-Skript zu einer späteren Zeit ausführen

at ␣ **-f** ␣ skript ␣ [**-q**warteschlange] ␣ [**-m**] ␣ zeit [␣ datum] [␣ **+**inkrement] ↵

-f ␣ skript
at liest die auszuführenden Kommandos aus dem angegebenen Shell-Skript.

Die übrigen Optionen sind unter *Format 1* beschrieben.

Format 3: Nicht bearbeitete Aufträge ausgeben

at ␣ **-l** [␣ auftragsnummer] ...

-l [␣ auftragsnummer]
at listet den Auftrag mit *auftragsnummer* auf, wenn er noch nicht bearbeitet wurde. *auftragsnummer* ist die Nummer, die auf die Standard-Fehlerausgabe ausgegeben wird, wenn ein Kommandoauftrag mit *at*, *batch* oder *cron* erteilt wird.

auftragsnummer nicht angegeben:
at listet alle Aufträge, die noch nicht bearbeitet wurden, mit ihren Auftragsnummern auf.

Format 4: Aufträge löschen

at ␣ **-r** ␣ auftragsnummer ␣ ...

-r ␣ auftragsnummer
at löscht den Auftrag *auftragsnummer*, den Sie zuvor mit *at* oder *batch* erteilt haben. *auftragsnummer* ist die Nummer, die auf die Standard-Fehlerausgabe ausgegeben wird, wenn ein Kommandoauftrag mit *at* oder *batch* erteilt wird. Sie können mehrere Auftragsnummern, getrennt durch Leerzeichen, angeben. Nur der Systemverwalter hat das Recht, Aufträge von anderen Benutzern zu löschen.

ENDE-STATUS

0 *at* wurde erfolgreich ausgeführt
≠0 Bei der Ausführung von *at* ist ein Fehler aufgetreten

FEHLERMELDUNGEN

Die häufigsten Fehlermeldungen sind:

```
at: bad date specification
```

Sie haben das Datum in einem falschen Format angegeben.

```
at: too late
```

Sie haben für *zeit* "now" angegeben. Da "now" (jetzt) den Zeitpunkt des Aufrufs von *at* angibt, erfolgt eine Ausführungsanweisung immer zu spät.

```
at: you are not authorized to use at. Sorry.
```

Sie dürfen *at* nicht aufrufen (siehe *Vor dem Aufruf beachten*).

DATEIEN

/etc/cron.d/at.allow
: Liste der Benutzerkennungen mit Ausführrecht für *at*. In jeder Zeile steht jeweils eine Benutzerkennung.

/etc/cron.d/at.deny
: Liste der Benutzerkennungen ohne Ausführrecht für *at*. In jeder Zeile steht jeweils eine Benutzerkennung.

/var/spool/cron/atjobs
: Dateiverzeichnis, in dem die noch nicht bearbeiteten *at*-Aufträge in einzelnen Dateien aufgelistet werden. Für jeden *at*-Auftrag gibt es eine eigene Datei mit dem Dateinamen *auftragsnummer.a*. Diese Dateien haben die Zugriffsrechte -r-S--S--- (siehe *chmod*).

/etc/cron.d/queuedefs
: Datei, die Ablaufinformationen enthält.

BEISPIEL

Es soll am 1. April um 13 Uhr an der Datensichtstation tty013 das aktuelle Datum und die Zeichenkette *April, April!* ausgegeben werden:

```
$ at 1pm apr 1 ↵
echo `date`":April, April!" >> /dev/tty013 ↵
END
```

SIEHE AUCH

atq, *atrm*, *batch*, *calendar*, *crontab*, *date*, *kill*, *mail*, *nice*, *ps*, *sh*, *sort*
cron, *environ* [7]
getdate() [19]

atq
Kommandoaufträge, die mit at oder batch erteilt wurden, auflisten (at queue)

atq listet die Kommandoaufträge auf, die mit *at* oder *batch* erteilt wurden und noch nicht abgearbeitet worden sind.
Als nichtprivilegierter Benutzer können Sie nur Ihre eigenen Aufträge abfragen. Als Systemverwalter können Sie sich alle Aufträge auflisten lassen.

atq[_option][_benutzerkennung]...

Keine Option angegeben
atq listet die Aufträge in der Reihenfolge auf, in der sie ausgeführt werden sollen.

option

-c
atq listet die Aufträge in der Reihenfolge auf, in der sie erteilt wurden.

-n
atq gibt nur die Anzahl der Aufträge aus, die in der Warteschlange stehen.

Nur für den Systemverwalter

benutzerkennung
atq gibt nur die Aufträge des angegebenen Benutzers aus.
Sie können mehrere Kennungen angeben.

benutzerkennung nicht angegeben:
Wenn Sie als Systemverwalter *atq* ohne *benutzerkennung* aufrufen, werden sämtliche Aufträge, die in der Warteschlange stehen, aufgelistet.

DATEIEN

/var/spool/cron
Dateiverzeichnis, in dem die noch nicht bearbeiteten Aufträge stehen.

BEISPIEL

```
$ atq
Rank     Execution Date     Owner       Job          Queue    Job Name
1st    Jun 6, 1990 18:00    anna     644709600.a      a        stdin
2nd    Dec 7, 1990 20:30    anna     644711400.a      a        stdin
```

SIEHE AUCH

at, *atrm*, *batch*, *crontab*
cron [7]

atrm
Kommandoaufträge, die mit at oder batch erteilt wurden, löschen (at remove)

atrm löscht Kommandoaufträge, die mit *at* oder *batch* erteilt wurden, aber noch nicht abgearbeitet worden sind.
Als nichtprivilegierter Benutzer können Sie nur Ihre eigenen Aufträge löschen. Als Systemverwalter können Sie alle Aufträge löschen.

atrm[_option]..._argument_...

option

-a
(a - all) *atrm* löscht alle Ihre Aufträge.
Rufen Sie dieses Kommando als Systemverwalter auf, werden sämtliche Aufträge aus der Warteschlange gelöscht.

Bei dieser Option geben Sie kein *argument* an.

-f
(f - force) *atrm* gibt keine Meldungen aus, die das Löschen der Aufträge betreffen.

-i
(i - interactive) *atrm* erwartet für jeden zu löschenden Auftrag eine Bestätigung. Nur wenn Sie jeweils mit *y* bestätigen, löscht *atrm* den Auftrag.

argument
argument ist entweder eine Auftragsnummer oder eine Benutzerkennung. *atrm* löscht die Aufträge mit den angegebenen Nummern sowie alle Aufträge der angegebenen Benutzer.

Als normaler Benutzer dürfen Sie nur Ihre eigenen Aufträge löschen. Aufträge anderer Benutzer darf nur der Systemverwalter löschen.

Mit *atq* können Sie sich die Aufträge mit den zugehörigen Auftragsnummern ausgeben lassen.

DATEIEN

/var/spool/cron
Dateiverzeichnis, in dem die noch nicht bearbeiteten Aufträge stehen.

BEISPIEL

Sie lassen sich zunächst mit *atq* die Aufträge ausgeben, die mit *at* oder *batch* erteilt wurden und noch nicht abgearbeitet worden sind. Anschließend löschen Sie den Auftrag mit der Auftragsnummer 644711400.a.

```
$ atq
Rank     Execution Date     Owner       Job          Queue   Job Name
1st    Jun 6, 1990 18:00    anna     644709600.a      a       stdin
2nd    Dec 7, 1990 20:30    anna     644711400.a      a       stdin

$ atrm 644711400.a
644711400.a removed
```

SIEHE AUCH

at, *atq*, *batch*, *crontab*
cron [7]

awk
Programmierbare Bearbeitung von Textdateien

awk ist ein programmgesteuertes Textbearbeitungssystem.
Sie rufen *awk* zusammen mit einem awk-Programm und den zu bearbeitenden Dateien auf. *awk* führt dann die durch das Programm festgelegte Bearbeitung mit den angegebenen Dateien aus. *awk* verändert die Eingabedateien nicht. Standardmäßig schreibt *awk* das Ergebnis der Bearbeitung auf die Standard-Ausgabe.

Im Vergleich zu Textbearbeitungsprogrammen wie *egrep* und *sed* bietet *awk* folgende Vorteile:

- *awk* arbeitet satzweise. Ein Eingabesatz ist zwar wie bei *egrep* und *sed* standardmäßig eine Zeile, der Benutzer kann diese Einstellung aber ändern und andere Texteinheiten als Satz definieren.
- Jeder Eingabesatz ist in Felder aufgeteilt, die einzeln angesprochen werden können.
- Eine Auswahlbedingung kann eine aus erweiterten regulären Ausdrücken und Vergleichen zusammengesetzte Bedingung sein.
- Der Benutzer kann beliebige Aktionen programmieren. Die *awk*-Sprache ist eine höhere, C-ähnliche Programmiersprache.

Die Beschreibung des *awk* gliedert sich in folgende Abschnitte:

- Verwendungsmöglichkeiten für *awk*
- Struktur eines *awk*-Programms
- Arbeitsweise von *awk*
- Die Eingabedatei (Sätze, Felder, Vordefinierte Variablen)
- Grundelemente der *awk*-Sprache (Kommentare, Konstanten, Variablen)
- Ausdrücke
- Auswahlbedingungen
- Aktionen (Ablaufanweisungen, Funktionen).

awk[-**Fc**][-**v** initialisierung] prog[initialisierung] ... [datei]...

-Fc
Trennzeichen zwischen den Feldern eines Eingabesatzes festlegen.

c
Regulärer Ausdruck zur Bestimmung des Trennzeichens zwischen den Feldern eines Eingabesatzes. Das Trennzeichen gehört nicht zu einem Feld.

-Fc nicht angegeben:
Leer- und Tabulatorzeichen gelten als Feldtrenner.

-v initialisierung
Wertzuweisung der Form

var=wert

awk initialisiert die Variable *var*, die im Programm vorkommt, mit *wert*.

var
Name der Variablen, die initialisiert werden soll.

wert
Wert, mit dem die Variable *var* initialisiert werden soll. *wert* kann genauso definiert werden, wie eine Umgebungsvariable in der Shell.

Die Wertzuweisung über *-v initialisierung* ist identisch mit der Wertzuweisung über *initialisierung* (siehe dort).

prog
Angabe des *awk*-Programms.

prog kann sein:
- 'awk-programm', ein *awk*-Programm in der Kommandozeile
- *-f progdat*, der Name einer Datei, die ein *awk*-Programm enthält.

'awk_programm'
Ein awk-Programm in der Kommandozeile.
Sie sollten das *awk*-Programm immer in Hochkommata '...' einschließen, um Shell-Sonderzeichen vor ungewollter Auswertung durch die Shell zu schützen.
Wenn das Programm länger als eine Zeile ist, dann müssen Sie das Neue-Zeile-Zeichen durch einen Gegenschrägstrich \ entwerten.

Beispiel

Alle Zeilen aus der Datei *eingabe* ausgeben, deren drittes Feld aus dem Zeichen '0' besteht:

```
$ awk '$3 == 0' eingabe
```

-f progdat
Das *awk*-Programm steht in der Datei *progdat*.

initialisierung
Wertzuweisung der Form:

var=wert

awk initialisiert die Variable *var*, die im *awk*-Programm vorkommt, mit *wert*. *initialisierung* und *datei* können in beliebiger Reihenfolge angegeben werden. Die Wertzuweisung erfolgt zu dem Zeitpunkt, zu dem normalerweise die an der Position stehende Datei geöffnet worden wäre.

Ausnahme

Die $-Variablen (siehe *Grundelemente*) können nicht auf diese Weise initialisiert werden.

var
Name der Variablen, die initialisiert werden soll.

wert
Wert, mit dem die Variable *var* initialisiert werden soll. *wert* kann genauso definiert werden wie eine Umgebungsvariable in der Shell.

datei
Name der Textdatei, die bearbeitet werden soll. Sie können mehrere Dateien angeben. Bei mehreren Angaben verarbeitet *awk* die Dateien in der angegebenen Reihenfolge. Wenn Sie für *datei* einen Bindestrich - angeben, liest *awk* von der Standard-Eingabe.

datei nicht angegeben:
awk liest von der Standard-Eingabe. *awk* liest die Eingabe satzweise ein, bearbeitet sie und gibt nach jeder Zeile das Ergebnis für diesen Satz aus. Sie beenden die Eingabe mit END oder CTRL D.

Verwendungsmöglichkeiten für awk

awk ist ein Werkzeug, mit dem Sie Textbearbeitungsaufgaben bequem lösen können. Typische Anwendungen sind:

- Daten aus Dateien heraussuchen
- Dateiinhalte überprüfen
- Berechnungen mit den Daten in einer Datei durchführen
- Format der Eingabedaten ändern.

Dieser Abschnitt zeigt an vier einfachen Beispielen, wie Sie *awk* anwenden können.

Beispiele

Die Datei *artikel* enthält eine Aufstellung von Büroartikeln. Angegeben sind jeweils der Artikelname, die Stückzahl und der Einzelpreis:

```
Bleistift  1500     0.60
Tisch         5   345.00
Lampe        20    79.80
Papier       75     1.00
Diskette   1000     2.40
Umschlag    100     0.20
```

1. Alle Artikel heraussuchen, deren Stückzahl größer als 100 ist:

```
$ awk '$2 > 100 {print}' artikel

Bleistift  1500     0.60
Diskette   1000     2.40
```

   Mit *$2* sprechen Sie das zweite Feld einer Zeile an, das in diesem Beispiel die Stückzahl eines Artikels enthält. Wenn die Stückzahl größer als 100 ist, ist die Bedingung erfüllt und die Funktion *print* wird ausgeführt. Da für *print* keine Argumente angegeben sind, gibt *print* die ganze Zeile aus.

2. Für alle Artikel mit einer Stückzahl größer als 100 den Gesamtpreis berechnen und zusammen mit der Artikelbezeichnung ausgeben:

```
$ awk '$2 > 100 {print $1 "\t" $2*$3}' artikel

Bleistift       900
Diskette        2400
```

   In diesem Beispiel hat die *print*-Funktion drei Argumente. Ausgegeben werden:

$1	Artikelbezeichnung (erstes Feld)
\t	Tabulatorzeichen
$2*$3	Stückzahl (zweites Feld) mal Einzelpreis (drittes Feld)

3. Die Ausgabe mit einer Überschrift versehen:

```
$ awk 'BEGIN     {print "Artikelbezeichnung \tGesamtbetrag"} \
>       $2 > 100 {print $1 "\t\t" $2*$3}' artikel
Artikelbezeichnung      Gesamtbetrag
Bleistift               900
Diskette                2400
```

Dieses Beispiel zeigt die Verwendung des BEGIN-Teils. *awk* führt die Aktion hinter BEGIN nur einmal bei Start des Programms aus. Deshalb wird die Überschrift genau einmal am Anfang ausgegeben.

4. Am Ende die Summe aller Beträge ausgeben.
Dazu wird eine Variable *summe* verwendet, die im BEGIN-Teil mit 0 initialisiert wird. Für jede Zeile wird das Produkt aus zweiter und dritter Spalte aufaddiert:

```
$ awk 'BEGIN     {summe=0; print "Artikelbezeichnung \tGesamtbetrag"} \
>       $2 > 100 {print $1 "\t\t" $2*$3; summe += $2*$3} \
>       END      {print "\nSumme: " summe}' artikel
Artikelbezeichnung      Gesamtbetrag
Bleistift               900
Diskette                2400

Summe: 3300
```

Dieses Beispiel zeigt die Verwendung des END-Teils. *awk* führt die Aktion hinter END nur einmal vor Beendigung des Programms aus. Deshalb wird die Summe aller Beträge genau einmal am Ende ausgegeben.

Struktur eines awk-Programms

Ein *awk*-Programm kann aus einem BEGIN-, Haupt- und END-Teil bestehen, die nach folgendem Schema aufgebaut sind:

```
[ BEGIN {aktion} ]                                          BEGIN-Teil
[ [auswahlbedingung] {aktion}                               Hauptteil
  | auswahlbedingung [{aktion}]
  | funktionsdefinition
  .
  .
  .
                                  ]
[ END {aktion} ]                                            END-Teil
```

auswahlbedingung
: Mit *auswahlbedingung* gibt der Benutzer an, welche Daten aus den Eingabedateien ausgewählt werden sollen (siehe *Auswahlbedingungen*).

aktion
: Mit *aktion* gibt der Benutzer an, wie die ausgewählten Dateien weiter bearbeitet werden sollen (siehe *Aktionen*).

funktionsdefinition
: Mit *funktionsdefinition* hat der Benutzer die Möglichkeit, eigene Funktionen zu definieren (siehe *Funktionen*).

Mindestens einer der drei Teile *auswahlbedingung*, *aktion*, *funktionsdefinition* muß vorhanden sein.
Von einem Paar *auswahlbedingung {aktion}* kann entweder die Auswahlbedingung oder die Aktion fehlen. Fehlt *aktion*, so wird jeweils die durch *auswahlbedingung* erfaßte Zeile ausgegeben. Fehlt *auswahlbedingung*, so wird *aktion* für jede Zeile ausgeführt.
Die Definition einer benutzerdefinierten Funktion kann an beliebiger Stelle im Hauptteil erfolgen.
Es ist erforderlich, daß folgende Teile jeweils am Anfang einer Zeile (nach beliebig vielen Leer- oder Tabulatorzeichen) stehen:
- der BEGIN-Teil
- die Paare *[auswahlbedingung]{aktion}* bzw. *auswahlbedingung [{aktion}]*
- die Funktionsdefinition
- der END-Teil

Arbeitsweise von awk

awk führt das vom Benutzer angegebene *awk*-Programm aus. Dabei geht *awk* im einzelnen wie folgt vor:

1. Anfangsarbeiten
   Wenn Variablen angegeben wurden, so werden als erstes diese Variablen initialisiert. Falls ein *BEGIN*-Teil mit *aktion* vorhanden ist, führt *awk* dann die dort festgelegte Aktion aus. Die im *BEGIN*-Teil angegebene Aktion wird genau einmal ausgeführt und zwar vor der Bearbeitung der ersten Eingabezeile.

2. Dateibearbeitung
   Anschließend verarbeitet *awk* die angegebenen Eingabedateien. *awk* liest die Eingabesätze der Reihe nach ein. Für jeden Eingabesatz prüft *awk* jede Auswahlbedingung ab und zwar in der Reihenfolge, wie sie im *awk*-Programm angegeben sind. Wenn eine Auswahlbedingung zutrifft, wird die zugehörige Aktion ausgeführt.
   Wenn zu einer Aktion keine Auswahlbedingung angegeben ist, führt *awk* die Aktion für jeden Satz aus. Wenn zu einer Auswahlbedingung keine Aktion angegeben ist, ist die Standard-Aktion Ausgabe des Satzes.
   Mehrere Eingabedateien werden in der angegebenen Reihenfolge abgearbeitet.

3. Abschlußarbeiten
   Wenn alle angegebenen Dateien abgearbeitet sind und ein END-Teil vorhanden ist, dann führt *awk* zum Schluß die im END-Teil angegebene Aktion aus. Danach beendet sich *awk*.

Die Eingabedatei

Eine Eingabedatei besteht aus Sätzen, die in Felder eingeteilt sind.

Sätze

Die Sätze sind durch ein Satztrennzeichen getrennt. Die Satztrennzeichen sind nicht Bestandteil des Satzes. Standardmäßig ist ein Satz eine Zeile und das Satztrennzeichen ist das Neue-Zeile-Zeichen. Der Benutzer hat die Möglichkeit, diese Einteilung zu ändern. Dazu gibt es die vordefinierte Variable *RS* (RS - Record Separator), der Sie ein beliebiges Zeichen zuweisen können. Falls als Wert für *RS* eine Zeichenkette angegeben wird, wird nur das erste Zeichen dieser Zeichenkette berücksichtigt. Die Anzahl der aktuell verarbeiteten Sätze wird in der Variablen *NR* (NR - Number of Records) gezählt. Bei mehreren Eingabedateien wird *NR* über alle Dateien hochgezählt. Auf den aktuellen Satz können Sie mit der vordefinierten Variablen *$0* zugreifen. Näheres zu Variablen erfahren Sie im Abschnitt *Grundelemente der awk-Sprache*.

Felder

Jeder Satz ist in Felder unterteilt, die durch ein oder mehrere Feldtrennzeichen getrennt sind. Standardmäßig ist eine beliebig lange Folge von Leer- und Tabulatorzeichen ein Feldtrenner. Der Benutzer hat die Möglichkeit, diese Einteilung zu ändern. Dazu gibt es die vordefinierte Variable *FS* (FS - Field Separator), der Sie entweder beim Aufruf durch Angabe der Option *-F* oder im *awk*-Programm ein beliebiges anderes Zeichen zuweisen können. Der an *FS* zugewiesene Wert wird als erweiterter regulärer Ausdruck (siehe *Tabellen und Verzeichnisse, Reguläre Ausdrücke*) interpretiert.

Beispiele

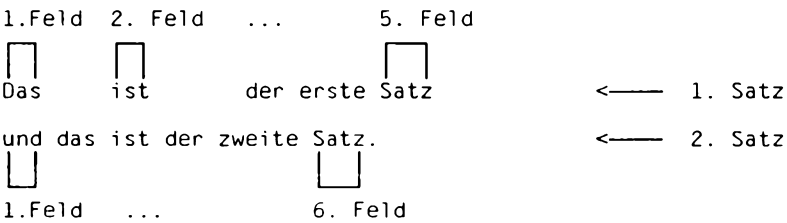
1. Sie wollen die Zeichen *x* und *y* als alternative Feldtrenner definieren.
   Syntax beim *awk*-Aufruf: `-F[xy]`
   Syntax im *awk*-Programm: `FS=[xy]`
2. Sie wollen eine beliebig lange Folge des Zeichens *x* als Feldtrenner definieren.
   Syntax beim *awk*-Aufruf: `-Fx+`
   Syntax im *awk*-Programm: `FS=x+`

Die Standard-Einstellung (beliebig lange Folge von Leer- und Tabulatorzeichen) kann also mit dem folgenden regulären Ausdruck dargestellt werden: `[␣\t]+` (␣ steht hier für das Leerzeichen, \t für das Tabulatorzeichen).
Beachten Sie, daß ein Neue-Zeile-Zeichen immer als Feldtrenner interpretiert wird, egal welchen Wert *FS* hat!

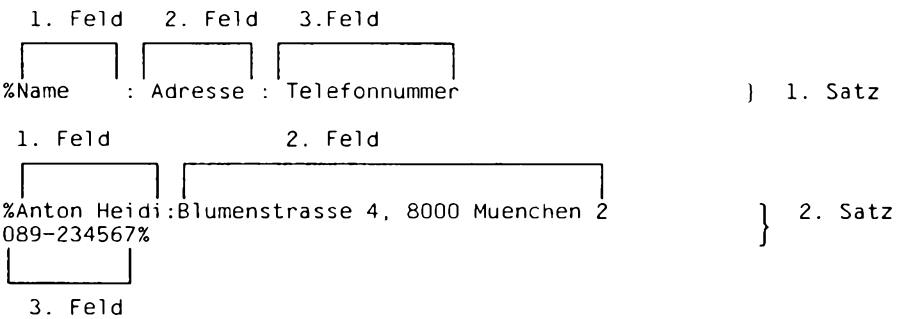
Die Anzahl der Felder des aktuellen Satzes wird in der Variablen *NF* (NF - Number of Fields) gespeichert. Auf die einzelnen Felder des aktuellen Satzes können Sie mit den vordefinierten Variablen *$1*, *$2* bis *$NF* zugreifen. Näheres zu Variablen erfahren Sie im Abschnitt *Grundelemente der awk-Sprache*.

Beispiele

1. Standard-Einteilung



2. Nicht-Standard-Einteilung: RS="%"; FS=":";



Regeln für Satz- und Feldtrennzeichen

- Standard-Einstellungen für Satztrenner
  - Standardmäßig ist das Neue-Zeile-Zeichen das Satztrennzeichen.
  - Wenn *RS* die leere Zeichenkette ist (*RS* = ""), besteht die Datei aus einem einzigen Satz. Falls mehrere Dateien angegeben werden, besteht jede Datei aus einem einzigen Satz (*NR* hat am Ende die Anzahl der Dateien als Wert).
- Standard-Einstellungen für Feldtrenner
  - Wenn der Satztrenner das Neue-Zeile-Zeichen ist, dann gelten standardmäßig Leer- und Tabulatorzeichen als Feldtrenner.
  - Wenn der Satztrenner nicht das Neue-Zeile-Zeichen ist, dann gilt das Neue-Zeile-Zeichen **immer** als Feldtrenner, unabhängig davon, welches Zeichen als Feldtrenner definiert ist (siehe *Felder, Beispiel 2*).
  - Wenn Sie *FS* explizit das Leerzeichen zuweisen, durch Aufruf von *awk* mit -F" " oder mit der Zuweisung FS=" ", dann gelten Leer- **und** Tabulatorzeichen als Feldtrenner.
  - Wenn Sie dagegen *FS* explizit das Tabulatorzeichen zuweisen (FS="\t"), dann gilt nur noch das Tabulatorzeichen als Feldtrenner, das Leerzeichen nicht mehr.
- Führende Feldtrenner und Folgen von Feldtrennern
  - Für die Feldtrenner Leer-, Tabulator- und Neue Zeile-Zeichen gilt:
    1. Führende Feldtrenner werden ignoriert.
    2. Beliebige Folgen von Feldtrennern werden als ein Trennzeichen gezählt. (siehe *Beispiel 9*).
  - Bei jedem anderen Feldtrenner werden führende Feldtrenner gezählt. Bei einer Folge von Feldtrennern wird jedes Zeichen einzeln gezählt. Zwei aufeinanderfolgende Feldtrenner ergeben daher ein leeres Feld. (siehe *Beispiel 10*).
- Einteilungen verändern
  Sie können *RS* im *awk*-Programm verändern, wenn Sie für eine Datei verschiedene Satzeinteilungen benötigen. Der neue Satztrenner gilt, sobald die Zuweisung an *RS* ausgeführt wurde. Ebenso können Sie *FS* im *awk*-Programm verändern, wenn Sie für eine Datei verschiedene Feldeinteilungen benötigen. Der neue Feldtrenner gilt, sobald die Zuweisung an *FS* ausgeführt wurde.

Vordefinierte Variablen für die Eingabedatei

Die folgende Tabelle enthält alle vordefinierten *awk*-Variablen, die die Eingabedatei betreffen. In der zweiten Spalte ist angegeben, mit welchem Wert *awk* diese Variable standardmäßig belegt.

Variable	Wert, den *awk* in der Variablen speichert
FILENAME	Name der aktuellen Eingabedatei, - bei Standard-Eingabe
FS	Feldtrenner für die Eingabe (Standard: beliebig lange Folge von Leer- und Tabulatorzeichen)
NF	Anzahl der Felder des aktuellen Satzes
NR	Laufende Nummer des aktuellen Satzes im Input
FNR	Laufende Nummer des aktuellen Satzes in der aktuellen Datei
RS	Satztrenner für die Eingabe (Standard: Neue-Zeile-Zeichen)
$0	Aktueller Satz
$1	Erstes Feld des aktuellen Satzes
$2	Zweites Feld des aktuellen Satzes
...	
$NF	Letztes Feld des aktuellen Satzes

Sie können die Variablen im *awk*-Programm verändern. Die Eingabedatei wird dadurch aber nicht verändert. Näheres zu Variablen erfahren Sie im nächsten Abschnitt *Grundelemente der awk-Sprache*.

Grundelemente der awk-Sprache

In diesem Abschnitt sind die Grundelemente der *awk*-Sprache zusammengestellt. Die Grundelemente benötigen Sie bei der Formulierung von Auswahlbedingungen und Aktionen.

Kommentar

Sie können ein *awk*-Programm wie eine Shell-Prozedur kommentieren. Kommentar beginnt mit dem Zeichen # und geht bis zum Ende derselben Zeile.

Konstanten

Es gibt zwei Arten von Konstanten:

```
zahl
zeichenkette
```

zahl

Eine Zahl (numerische Konstante)
ist eine Ganzzahl oder eine Gleitkommazahl mit oder ohne Vorzeichen. *awk* überprüft das Format nicht. Wenn eine Zahl ungültige Zeichen enthält, versucht *awk* einen gültigen Teil herauszufiltern und ignoriert den Rest.

ganzzahl

Eine Ganzzahl ist eine Folge aus den Ziffern 0 bis 9.

Beispiel

```
-43
```

gleitkommazahl

Eine Gleitkommazahl besteht aus Mantisse mit oder ohne Exponent.
Die Mantisse besteht aus einer Ganzzahl mit oder ohne Nachkommateil.
Der Nachkommateil besteht aus einem Dezimalpunkt und einer Ganzzahl.

zeichenkette

Eine Zeichenkette (alphanumerische Konstante)
ist eine Folge von Zeichen, eingeschlossen in Anführungszeichen "...". Fehlen die Anführungszeichen, dann interpretiert *awk* die Zeichenkette als Variablennamen, Zahl oder Operator.

zeichen

Ein Einzelzeichen wird auch in Anführungszeichen "..." eingeschlossen, damit *awk* das Zeichen nicht als Variablennamen interpretiert. Ein Zeichen ist ein darstellbares Zeichen aus dem aktuell gültigen Zeichensatz (siehe *Tabellen und Verzeichnisse, Zeichensatz ISO 646*) oder eines der folgenden Sonderzeichen, die wie in C dargestellt werden:

\"	für	"
\\	für	\
\n	für	Neue-Zeile-Zeichen
\t	für	Tabulatorzeichen
\b	für	Rücksetzzeichen (backspace)
\r	für	Wagenrücklauf (carriage return)
\f	für	Seitenvorschub

Variablen

awk ermöglicht die Verwendung von einfachen Variablen und Arrays, um Werte abzuspeichern.
Es gibt vordefinierte und benutzerdefinierte Variablen.

Variablennname

Der Name einer benutzerdefinierten Variablen fängt entweder mit einem Unterstrich _, einem Großbuchstaben oder einem Kleinbuchstaben an und besteht nur aus Unterstrichen, Groß- und Kleinbuchstaben sowie Ziffern.

Datentyp

Variablen haben keinen Datentyp. Sie können derselben Variablen sowohl eine Zahl als auch eine Zeichenkette zuweisen. In einem eindeutig numerischen Kontext werden Variablen als numerische Variablen behandelt, ansonsten gelten sie standardmäßig als alphanumerisch.

Beispiel

```
x = "Mueller";     # die Variable x enthält die Zeichenkette Mueller
x = "3"+4     ;    # die Variable x hat den Wert 7
```

Definition

Variablen werden nicht explizit definiert. Benutzerdefinierte Variablen sind mit der ersten Verwendung automatisch definiert.

Initialisierung

Die vordefinierten Variablen werden von *awk* wie vorgesehen belegt. Benutzerdefinierte Variablen werden von *awk* standardmäßig je nach Kontext mit der leeren Zeichenkette bzw. mit Null initialisiert. Bei Aufruf von *awk* können Sie andere Initialisierungen angeben.

Ausnahmen

- Für i>NF ist $i nicht unbedingt die leere Zeichenkette.
- $-Variablen können nicht bei Aufruf initialisiert werden.

Vordefinierte Variablen

awk kennt die in der Tabelle aufgeführten vordefinierten Variablen. In der Tabelle ist angegeben, welche Werte *awk* standardmäßig in diesen Variablen speichert. Der Benutzer kann den Variablen neue Werte zuweisen.

Variable	Wert, den awk in der Variablen speichert
ARGC	Anzahl der Elemente im Array *ARGV*
ARGV	Array, das die Argumente der Kommandozeile enthält (ausgenommen Optionen und das Argument *prog*), numeriert von 0 bis ARGC-1
ENVIRON	Array mit den Werten der Umgebungsvariablen, die Indices sind die Namen der Variablen
FILENAME	Name der aktuellen Eingabedatei, - bei Standard-Eingabe
FS	Feldtrenner für die Eingabe (Standard: beliebig lange Folge von Leer- und Tabulatorzeichen)
NF	Anzahl der Felder des aktuellen Satzes
NR	Laufende Nummer des aktuellen Satzes im Input
FNR	Laufende Nummer des aktuellen Satzes in der aktuellen Datei
OFS	Feldtrenner für die Ausgabe (Standard: Leerzeichen)
ORS	Satztrenner für die Ausgabe (Standard: Neue-Zeile-Zeichen)
OFMT	Ausgabeformat für Gleitkommazahlen (siehe *awk-Funktionen, Formatierte Ausgabe*) (Standard: %.6g, höchstens 6 Stellen hinter dem Komma)
RS	Satztrenner für die Eingabe (Standard: Neue-Zeile-Zeichen)
RLENGTH	Länge der Zeichenkette, die die *match*-Funktion als passend erkannt hat
RSTART	Anfangsposition der Zeichenkette, die die *match*-Funktion als passend erkannt hat. Die Numerierung fängt mit 1 an. Dieser Wert entspricht immer dem Return-Wert der *match*-Funktion.
SUBSEP	Subscript-Zeichenketten-Trenner für mehrdimensionale Arrays. Standard-Einstellung ist \034.
$0	Aktueller Satz
$*n*	*n*-tes Feld des aktuellen Satzes
$NF	letztes Feld des aktuellen Satzes

Wie wirkt sich die Änderung von vordefinierten Variablen aus?

Beispiel

Durch die Zuweisung

```
$1 = "neu";
```

wird *$1* die Zeichenkette *neu* zugewiesen. Das erste Feld des aktuellen Eingabesatzes bleibt aber unverändert.

Das gilt auch für folgende *awk*-Einstellungen, die die Eingabedatei betreffen:

1. Die aktuelle Eingabedatei ändert sich nicht, wenn Sie FILENAME einen neuen Namen zuweisen.
2. Wenn Sie an eine Variable $i mit i>NF einen Wert zuweisen, bekommt *NF* den Wert *i* zugewiesen.
3. Wenn Sie *NR* einen neuen Wert zuweisen, wird nur die Zeilennummer verändert, aber die Einstellung, welcher Satz der aktuelle ist, bleibt unverändert.

   Beispiel

   Der Inhalt von *$0* bleibt unverändert, auch wenn *NR* verändert wird.

   ```
   {print NR, $0; NR=NR+34; print NR, $0}
   ```

   Die Ausgabe sieht dann etwa so aus:

   ```
   10 Dies ist die zehnte Zeile
   44 Dies ist die zehnte Zeile
   ```

Vorsicht

Wenn Sie einer Variablen einen neuen Wert zuweisen, wird der alte Inhalt gelöscht. Wenn Sie z.B. *NF* verändern, ist die Information über die Feld-Anzahl des aktuellen Satzes verloren.

Besonderheit bei $-Variablen:
Sie können die Nummer einer $-Variablen als Konstante angeben oder durch einen Ausdruck, der bei Auswertung die Nummer ergibt.

Beispiel

Mit:

```
$(NF-1)
```

sprechen Sie das vorletzte Feld an.

Array

Ein Array ist ein Feld von Konstanten.

Ein Array-Element wird angesprochen mit:

```
array_name[index]
```

array_name
Variablenname.

index
einfache Variable.
Der Index kann numerisch oder alphanumerisch sein. Sie können für *index* daher eine Zahl, eine Zeichenkette oder einen Ausdruck angeben, der bei Auswertung den Indexwert ergibt.

awk bietet bezüglich Arrays zwei besondere Möglichkeiten:

- Arrays werden dynamisch angelegt:
  Arrays werden wie einfache Variablen nicht deklariert, insbesondere muß auch keine Dimension festgelegt werden. Bei Bedarf wird automatisch ein neues Array-Element angelegt.
- Arrays können assoziativ durchlaufen werden:
  Sie können einen alphanumerischen Index verwenden, um die Array-Elemente anzusprechen.
  Zum Durchlaufen aller Elemente eines assoziativen Arrays gibt es die spezielle Laufanweisung:

  for(index in array) anweisung

  index durchläuft in **unbestimmter** Reihenfolge die bisher vorhandenen Indexwerte. Für jedes Array-Element wird *anweisung* einmal ausgeführt (siehe Ablaufanweisung *for*).

Beispiel

In der Datei *ausgaben* sind Ausgaben erfaßt. Für jede Ausgabe sind Tag, Monat, Betrag und eine Kurzbeschreibung angegeben. Die einzelnen Angaben sind durch Doppelpunkt : getrennt, z.B.:

```
01:Januar:   40.78:Buerobedarf
05:Januar: 6789.00:Laser-Drucker
23:Maerz:   240.32:Lampen
11:Januar:  478.00:Stuehle
01:Februar:  45.00:Literatur
```

Durch Verwendung eines Arrays können Sie aus dieser Datei sehr einfach die Gesamtausgaben für jeden Monat berechnen. Das Beispielprogramm verwendet das Array *mausgaben* und die Monatsnamen als alphanumerischen Index. Für jede Zeile werden die Ausgaben im dritten Feld (*$3*) zu den Ausgaben des Monats addiert, der in dem zweiten Feld (*$2*) steht.

```
$ awk 'BEGIN {FS=":"}                    \
>      {mausgaben[$2] += $3;}  \
>      END {for (i in mausgaben) print "Gesamtausgaben",\
>           i, mausgaben[i]}' ausgaben

Gesamtausgaben Februar 45
Gesamtausgaben Maerz 240.32
Gesamtausgaben Januar 7307.78
```

Ausdrücke

Ein Ausdruck kann sein:

- konstante
- variable
- funktionsaufruf
- un_op ausdruck
- ausdruck bin_op ausdruck
- (ausdruck)
- ausdruck ? ausdruck : ausdruck

konstante
: numerische oder alphanumerische Konstante (siehe *Grundelemente*).

variable
: Variable (siehe *Grundelemente*).

funktionsaufruf
: Aufruf einer vordefinierten Funktion (siehe *Funktionen*).

ausdruck
: Ausdruck.

un_op
: unitärer Operator (siehe *Operator-Tabelle*).

bin_op
: binärer Operator (siehe *Operator-Tabelle*).

Ausdrücke werden ausgewertet und liefern einen Wert. Sie können in Auswahlbedingungen und Aktionen vorkommen.

awk-Operatoren

awk kennt alle C-Operatoren und zusätzlich die Operatoren für Mustervergleich und Konkatenation von Zeichenketten.
In der folgenden Tabelle sind alle *awk*-Operatoren nach steigender Priorität aufgeführt. Operatoren in einer Zeile haben die gleiche Priorität.

Operator	Bedeutung
=	Zuweisung
+= -= *= /= %=	zusammengesetzte Zuweisung wie in C
||	logisches ODER
&&	logisches UND
!	logisches NICHT
> >= < <= != ==	Vergleich
~ !~	Mustervergleich
hintereinanderschreiben	Konkatenation
+ - * / %	plus, minus, multiplizieren, dividieren, modulo
^ **	Exponent
++ --	Inkrement, Dekrement

Auswertung von Ausdrücken

Für die Operanden ist kein Datentyp vorgeschrieben. Sie können numerische und alphanumerische Konstanten beliebig mischen. *awk* bestimmt aus dem Kontext, ob eine numerische oder alphanumerische Operation ausgeführt wird.
Beachten Sie, daß es wie in C keine speziellen Wahrheitswerte gibt. Bei *awk* gilt wie bei C Null als falsch und ein Wert ungleich Null als wahr. Dies bedeutet, als Argument einer logischen Operation wird jeder Wert $\neq 0$ als wahr erkannt. Das Ergebnis einer einer solchen Operation wird, wenn es wahr ist, durch 1 dargestellt.

Beispiel

```
(2&&2)+3=4
```

Auswahlbedingungen

Mit den Auswahlbedingungen gibt der Benutzer an, welche Daten aus den Eingabedateien ausgewählt werden sollen. Eine Auswahlbedingung kann sein:

- /regulärer_ausdruck/
- vergleich
- mustervergleich
- bereichsangabe
- zusammengesetzte-auswahlbedingung

/regulärer_ausdruck/
Regulärer Ausdruck.
awk unterstützt erweiterte reguläre Ausdrücke. (siehe *Tabellen und Verzeichnisse, Reguläre Ausdrücke*). Ein regulärer Ausdruck wird in Schrägstriche /.../ eingeschlossen.

Beispiel

Regulärer Ausdruck, der für Folgen aus a, b oder c steht:

```
/[abc]+/
```

vergleich
Ein Vergleich ist ein Ausdruck (siehe *Ausdrücke*) mit Vergleichsoperatoren. Die Vergleichsoperatoren und ihre Bedeutung sind:

a > b	*a* größer als *b*?
a >= b	*a* größer als oder gleich *b*?
a < b	*a* keiner als *b*?
a <= b	*a* kleiner als oder gleich *b*?
a == b	*a* gleich *b*?
a != b	*a* ungleich *b*?

Die Operanden *a* und *b* können beliebige Ausdrücke sein. Die Operanden-Ausdrücke werden zuerst ausgewertet, dann wird der Vergleich mit den berechneten Ergebnissen ausgeführt. Wenn beide Operanden numerisch sind, wird ein numerischer Vergleich durchgeführt, ansonsten ein alphanumerischer.

mustervergleich

Ein Mustervergleich ist ein Ausdruck (siehe *Ausdrücke*) mit Mustervergleichsoperatoren. Bei einem Mustervergleich wird eine Zeichenkette mit einem regulären Ausdruck, genannt Muster, verglichen. Die Mustervergleichsoperatoren und ihre Bedeutung sind:

zk ~ m	Zeichenkette *zk* paßt zu Muster *m*
zk !~ m	Zeichenkette *zk* paßt nicht zu Muster *m*

Mit einem Mustervergleich als Auswahlbedingung können Sie einzelne Felder auswählen.
Zur Schreibweise von *zk* müssen Sie die üblichen Syntaxkonventionen beachten.

Beispiel

Alle Sätze auswählen, deren erstes Feld mit A oder a beginnt:

```
$1 ~ /^[Aa]/
```

Der reguläre Ausdruck ^[Aa] steht für Zeichenketten mit A oder a am Anfang. Das erste Feld des Satzes ($1) muß zu dem regulären Ausdruck passen (~), d.h. am Anfang muß ein A oder a stehen.

bereichsangabe

Die Bereichsangabe hat die Form:

/regulärer_ausdruck/, /regulärer_ausdruck/

Die Bereichsangabe bedeutet, daß die zugehörige Aktion für alle Sätze ausgeführt werden soll, die innerhalb des Bereichs liegen. Anfang und Ende des Bereichs werden durch zwei reguläre Ausdrücke festgelegt. Der Bereich beginnt mit dem Satz, der die erste Zeichenkette enthält, die zum ersten regulären Ausdruck paßt. Der Bereich endet mit dem Satz, der die erste Zeichenkette enthält, die zum zweiten regulären Ausdruck paßt.

Vorsicht

Wenn der Satz, der die erste zum zweiten regulären Ausdruck passende Zeichenkette enthält, in der Datei vor dem Satz steht, der die erste zum ersten regulären Ausdruck passende Zeichenkette enthält, dann ist der Bereich leer.

Beispiel

Sie wollen den Bereich von der ersten Zeile, die mit C beginnt, bis zur ersten Zeile, die mit K beginnt, auswählen und von jeder Zeile in diesem Bereich das erste Feld ausgeben lassen:

```
/^C/, /^K/  {print $1}
```

zusammengesetzte-auswahlbedingung

Mit den logischen Operatoren (siehe *Ausdrücke*) können Auswahlbedingungen negiert und mehrere Auswahlbedingungen zu einer Bedingung zusammengesetzt werden. Die logischen Operatoren und ihre Bedeutung sind:

!aw	Negation der Auswahlbedingung *aw*
aw1 || aw2	Auswahlbedingung *aw1* oder *aw2*. Die Bedingung ist erfüllt, wenn *aw1* oder *aw2* wahr (ungleich 0) ergibt.
aw1 && aw2	Auswahlbedingung *aw1* und *aw2*. Die Bedingung ist erfüllt, wenn *aw1* und *aw2* wahr (ungleich 0) ergeben.
(aw)	Klammern

Die Auswertung einer zusammengesetzten Bedingung erfolgt von links nach rechts.

Beispiel

Sie wollen alle Sätze auswählen, die eine gerade Feld-Anzahl haben und deren erstes Feld mit einem Buchstaben zwischen M (inklusive) und Q (exklusive) beginnt.

```
NF%2==0 && $1 >= "M" && $1 < "Q"
```

Es gibt im allgemeinen mehrere Möglichkeiten, eine Auswahl durch eine Bedingung zu formulieren. Besteht in der aktuell gültigen Sortierreihenfolge der Bereich [M-Q] genau aus den Großbuchstaben M, N, O, P, Q, so erhält man dieselbe Auswahl durch folgenden Mustervergleich:

```
NF%2==0 && $1 ~ /^[MNOP]/
```

Die erste *awk*-Zeile kann in Abhängigkeit von der Sortierreihenfolge des aktuellen Zeichensatzes unterschiedliche Ergebnisse liefern. Die zweite *awk*-Zeile wählt immer nur die Sätze aus, deren erstes Feld mit den Buchstaben M, N, O oder P beginnt.

Aktionen

In den Aktionen gibt der Benutzer an, wie die ausgewählten Daten weiter bearbeitet werden sollen. Die Aktion muß auf derselben Zeile wie die zugehörige Auswahlbedingung beginnen. Ist dies nicht möglich, so ist ein Neue-Zeile-Zeichen mit \ zu entwerten. Leer- und Tabulatorzeichen zwischen Aktion und Auswahlbedingung werden ignoriert.
Eine Aktion besteht aus einer oder mehreren Anweisungen und muß in geschweifte Klammern {...} eingeschlossen sein:

```
{anweisung_[;anweisung]...}
```

Anweisung

Eine Anweisung kann sein:

- ausdruck
- ablaufanweisung

ausdruck
: Der Ausdruck wird ausgewertet, jedoch nur dann weiter verwertet, wenn *ausdruck* eine Zuweisung, ein Inkrement oder ein Dekrement ist (siehe *Ausdrücke*).

ablaufanweisung
: Mit *ablaufanweisung* können Sie den Ablauf des *awk*-Programms steuern (siehe *Ablaufanweisungen*).

Eine Anweisung kann mehrere Zeilen einnehmen. Dabei muß jede Zeile mit einem Gegenschrägstrich \ abgeschlossen werden. Hierdurch wird das Neue-Zeile-Zeichen entwertet.

Mehrere Anweisungen

Mehrere Anweisungen können durch geschweifte Klammern {} zusammengefaßt werden. Anweisungen werden voneinander getrennt durch:

- Strichpunkt ;
- geschweifte Klammer zu}
- Neue-Zeile-Zeichen.

Ablaufanweisungen

Mit den Ablaufanweisungen können Sie den Ablauf des *awk*-Programms steuern. Bei *awk* gibt es folgende Ablaufanweisungen:

Ablaufanweisung	Bedeutung
break	Schleife abbrechen
continue	Rest eines Schleifendurchgangs überspringen
exit	*awk*-Programm beenden
for	Gezählte Wiederholung und ARRAY-Durchlauf
if	Bedingte Anweisung
next	Mit dem nächsten Eingabesatz fortfahren
while	Schleife
do	Schleife
delete array[i]	Element *i* des Arrays *array* löschen
return x	Rücksprung aus einer Funktion mit Wertangabe
return	Rücksprung aus einer Funktion ohne Wertangabe

Die Ablaufanweisungen sind im folgenden alphabetisch beschrieben.

break - Schleife abbrechen

break kann im Schleifenrumpf einer *for*-, *while*- oder *do*-Schleife verwendet werden. *break* bewirkt, daß die Schleife beendet wird.

break

Beispiel

Solange ein Satz mit . beginnt, soll der nächste Satz eingelesen werden. Wenn das 2. Feld des eingelesenen Satzes größer als 1000 ist, soll abgebrochen werden.

```
{ while($1 ~ /^\./)
    {
       getline;
       if($2 > 1000) break;
    }
}
```

continue - Rest eines Schleifendurchlaufs überspringen

continue kann im Schleifenrumpf einer *for-*, *while-* oder *do*-Schleife verwendet werden. *continue* bewirkt, daß der aktuelle Schleifendurchlauf beendet und mit dem nächsten Durchlauf weitergemacht wird.

continue

Beispiel

Es sollen nur gerade Felder ausgegeben werden:

```
{
   i=1;
   while(i++ <= NF)
      {
        if(i%2) continue;
        else print $i
      }
}
```

do - Schleife

Mit der *do*-Schleife (oder *do-while*-Schleife) wird eine Anweisung wiederholt, solange eine Bedingung erfüllt ist. Im Gegensatz zur *while*-Schleife wird die Anweisung auf jeden Fall mindestens einmal ausgeführt.

do anweisung **while** (ausdruck)

anweisung
: Anweisung, die bei jedem Schleifendurchgang ausgeführt wird. Wenn mehrere Anweisungen ausgeführt werden sollen, müssen sie mit geschweiften Klammern { } zusammengefaßt werden.

ausdruck
: Ausdruck (siehe *Ausdrücke*), der die Bedingung angibt.

Beispiel

Die Felder eines Satzes sollen einzeln ausgegeben werden:

```
{ i=0; do {print $(++i)} while (i != NF) }
```

exit - *awk*-Programm beenden

Mit *exit* wird das *awk*-Programm beendet.
Wenn ein END-Teil vorhanden ist, dann führt *awk* noch die dort angegebene Abschluß-Aktion aus, ansonsten wird das Programm sofort beendet.

exit

Beispiel

Wenn das Zeichen Klammeraffe in der Eingabe erscheint, soll das Ergebnis ausgegeben und die Bearbeitung beendet werden:

```
...
/@/ {exit}
...
END {print ergebnis}
```

for - Gezählte Wiederholung

Mit der *for*-Schleife wird eine Anweisung wiederholt, solange eine Bedingung erfüllt ist. Die *for*-Schleife wird üblicherweise für die gezählte Wiederholung mit Laufvariablen verwendet.

for(ausdruck1; ausdruck2; ausdruck3) anweisung

ausdruck1
Ausdruck (siehe *Ausdrücke*).
ausdruck1 wird einmal bei Start der *for*-Anweisung ausgewertet.

Beispiel

```
i=1
```

ausdruck2
Ausdruck (siehe *Ausdrücke*).
ausdruck2 wird vor jedem Schleifendurchgang ausgewertet. *anweisung* wird nur ausgeführt, falls *ausdruck2* ungleich 0 (wahr) ergibt, ansonsten wird die Wiederholung beendet.

Beispiel

```
i<10
```

ausdruck3
Ausdruck (siehe *Ausdrücke*).
ausdruck3 wird nach jedem Schleifendurchgang ausgewertet.

Beispiel

```
i++
```

anweisung
Anweisung, die bei jedem Schleifendurchgang ausgeführt wird. Wenn mehrere Anweisungen ausgeführt werden sollen, müssen sie mit geschweiften Klammern {} zusammengefaßt werden.

Beispiel

Die Felder des aktuellen Satzes sollen in umgekehrter Reihenfolge ausgegeben werden.

```
{for(i=NF; i>0; i--) print $i}
```

for - Array-Durchlauf

Diese Form der *for*-Anweisung ist eine *awk*-Besonderheit zum Durchlaufen eines Arrays.

```
for(index_in_array)_anweisung
```

index
: Variable (siehe *Grundelemente*), die in unbestimmter Reihenfolge alle Elemente des Arrays *array* durchläuft. Der Index kann numerisch oder alphanumerisch sein.

array
: Array, das durchlaufen wird.

anweisung
: Anweisung, die für jedes Array-Element ausgeführt wird. Wenn mehrere Anweisungen ausgeführt werden sollen, müssen sie mit geschweiften Klammern { } zusammengefaßt werden.

Beispiel

Das Array *monate* enthält für jeden Monat die Anzahl von Tagen. Ein Array-Element wird mit dem Monatsnamen indiziert, z.B.
`monate["Januar"]=31.`
Durch folgendes *awk*-Programm wird für jeden Monat der Monatsname und die Anzahl von Tagen ausgegeben.

```
$ awk ' BEGIN { monate["Januar"]=31;     \
>               monate["Februar"]=28;    \
>               monate["Maerz"]=31;      \
>               monate["April"]=30;      \
>               monate["Mai"]=31;        \
>               monate["Juni"]=30;       \
>               monate["Juli"]=31;       \
>               monate["August"]=31 }    \
>       END { for(i in monate) print "Monat",i,"hat",monate[i],"Tage" } '

Monat Mai hat 31 Tage
Monat August hat 31 Tage
Monat Juli hat 31 Tage
Monat April hat 30 Tage
Monat Juni hat 30 Tage
Monat Januar hat 31 Tage
Monat Maerz hat 31 Tage
Monat Februar hat 28 Tage
```

if - Bedingte Anweisung

Bei der *if*-Anweisung wird eine Anweisung abhängig von einer Bedingung ausgeführt.

if(ausdruck)␣anweisung1␣[**else**␣anweisung2]

ausdruck
: Ausdruck (siehe *Ausdrücke*), der die Bedingung angibt. Wenn *ausdruck* ungleich 0 (wahr) ergibt, wird *anweisung1* ausgeführt.

anweisung1
: Anweisung, die ausgeführt wird, falls *ausdruck* wahr ist. Wenn mehrere Anweisungen ausgeführt werden sollen, müssen sie mit geschweiften Klammern { } zusammengefaßt werden.

anweisung2
: Anweisung, die ausgeführt wird, falls *ausdruck* falsch ist. Wenn mehrere Anweisungen ausgeführt werden sollen, müssen sie mit geschweiften Klammern { } zusammengefaßt werden.

Beispiel

Wenn Feld 1 größer als 2 ist, dann sollen die Felder 2 und 3 ausgegeben werden, ansonsten die Felder 4 und 5:

```
{ if($1 > 2) print $2, $3; else print $4, $5 }
```

next - Mit dem nächsten Eingabesatz fortfahren

awk unterbricht die Bearbeitung des aktuellen Satzes; die Anweisungen, die *next* folgen, werden nicht ausgeführt. Dann liest *awk* den nächsten Eingabesatz ein. *NR*, *NF*, *FNR*, *$0* und *$1* bis *$NF* werden neu gesetzt.

Unterschied zur *getline*-Funktion:
getline macht den nächsten Satz zum aktuellen Satz; die Anweisungen, die *getline* folgen, werden mit den Werten des nächsten Satzes für die $-Variablen und für *NR*, *NF* und *FNR* ausgeführt.

next

Beispiel

Sätze, die mit . beginnen, sollen ignoriert werden:

```
{ if ($1 ~/^\./) next }
```

while - Schleife

Mit der *while*-Schleife wird eine Anweisung wiederholt, solange eine Bedingung erfüllt ist.

while(ausdruck) anweisung

ausdruck
Ausdruck (siehe *Ausdrücke*), der die Bedingung angibt.

anweisung
Anweisung, die bei jedem Schleifendurchgang ausgeführt wird. Wenn mehrere Anweisungen ausgeführt werden sollen, müssen sie mit geschweiften Klammern { } zusammengefaßt werden.

Beispiel

Alle Eingabefelder werden ausgegeben; jedes Feld wird in eine eigene Ausgabezeile geschrieben:

```
{ i = 1;
  while (i <= NF) {
      print $i
      i++
  }
}
```

Funktionen

awk stellt eine Reihe vordefinierter Funktionen zur Verfügung, bietet aber auch die Möglichkeit, eigene Funktionen zu definieren:

function name(arg,...) {anweisungen}

Vor *{anweisungen}* darf ein Neue-Zeile-Zeichen stehen. Leerzeilen innerhalb der geschweiften Klammern *{...}* sind ebenfalls erlaubt. Eine Funktionsdefinition steht im Hauptteil eines *awk*-Programms gleichberechtigt neben *auswahlbedingung {aktion}*.

Funktionsaufrufe dürfen innerhalb eines Aktionsteils an beliebiger Stelle in einem Ausdruck stehen, selbst vor der Funktionsdefinition. Beim Aufruf darf zwischen dem Funktionsnamen und der öffnenden runden Klammer kein Leerzeichen stehen.
Funktionsaufrufe können geschachtelt werden, Funktionen dürfen rekursiv aufgerufen werden.

Bei den meisten Funktionen müssen Sie die Argumente nicht in Klammern einschließen. Klammern sind aber empfehlenswert, weil sie die Lesbarkeit verbessern. Übergeben Sie ein Array als Argument, dann wird ein Verweis auf das Array übergeben (call by reference) - Sie können aus der Funktion die Elemente des Arrays verändern. Bei skalaren Argumenten wird zur Übergabe der Wert der Variable kopiert (call by value) - Sie können den Wert der Variablen aus der Funktion heraus nicht ändern. Argumente haben einen lokal auf die Funktion beschränkten Geltungsbereich, während alle anderen Variablen immer einen globalen haben. Benötigen Sie lokale Variablen in einer Funktion, dann definieren Sie diese am Ende der Argumentliste in der Funktionsdefinition. Jede Variable der Argumentliste, für die kein aktuelles Argument existiert, ist eine lokale Variable, die mit dem Wert *0* vorbelegt ist.

Wie in C kann es Funktionen geben, die ein Ergebnis liefern (z.B. *exp*) und solche mit prozeduralem Charakter (z.B. Ausgabefunktionen).
Die Anweisung *return* kann mit oder ohne Wertangabe benutzt werden, oder ganz wegfallen - dann ist der Rückgabewert undefiniert, falls darauf zugegriffen werden sollte.

Beispiel

Die Funktion *suche* sucht in dem Array *allenamen* nach der Zeichenkette *wer* und gibt den Index oder *-1* zurück. Dabei wird das 3. Argument *lauf* als lokale Variable verwendet.

```
   ...
{ print $1, suche($1, allenamen) }
   ...
function suche(wer, allenamen, lauf)
{
   for (lauf=0; allenamen[lauf]; lauf++)
      if (index(allenamen[lauf], wer) == 1
          && length(allenamen[lauf]) == length(wer))
             return lauf
   return -1
}
```

Vordefinierte Funktionen

Funktion	Bedeutung
Eingabefunktion	
getline	Satz einlesen
Ausgabefunktionen	
print([arg,...])	Standard-Ausgabefunktion
printf(format [arg,...])	Formatierte Ausgabe
Arithmetische Funktionen	
atan2(y,x)	Arcustangens von *y/x*
cos(x)	Cosinus
exp(x)	Exponentialfunktion
int(x)	Ganzzahliger Anteil
log(x)	Natürlicher Logarithmus
rand()	Liefert eine Zufallszahl
sin(x)	Sinus
sqrt(x)	Quadratwurzel
srand([x])	Setzt den Anfangs-Berechnungswert für *rand()*
Zeichenketten-Funktionen	
gsub(re,repl[,in])	Globale Substitutionsfunktion
index(zk1,zk2)	Erstes Vorkommen einer Teilzeichenkette
length([zk])	Länge einer Zeichenkette
match(zk,re)	Prüft, ob *zk* zum regulären Ausdruck *re* paßt
split(zk,array,[sep])	Aufteilen einer Zeichenkette
sprintf(format,e1,e2,...)	Formierte Ausgabe in eine Variable
sub(re, repl[,in])	Substitutionsfunktion
substr(zk,m,[n])	Teilzeichenkette bestimmen
Allgemeine Funktionen	
close(expr)	Datei oder Pipe schließen
system(expr)	Shell-Kommando aufrufen

Die Funktionen sind im folgenden alphabetisch beschrieben. Für jede Funktion ist angegeben, für welche Argumente sie vorgesehen ist. Sie können als Argument eine Konstante angeben oder einen Ausdruck (siehe *Ausdrücke*). *awk* wertet die Argument-Ausdrücke zuerst aus und wendet dann die Funktion auf die berechneten Ergebnisse an.

atan2 - Arcustangens

atan2 berechnet den Arcustangens des Quotienten zweier Zahlen: *atan2(y,x)* liefert den Arcustangens von y/x.

atan2(y,x)

y,x
Zahlen, für deren Quotient der Arcustangens berechnet werden soll.

close - Datei oder Pipe schließen

close schließt die angegebene Datei bzw. Pipe.

close(expr)

expr
Name der Datei oder der Pipe, die geschlossen werden soll (siehe bei der Beschreibung der Funktionen *print* und *printf* den Abschnitt über Ausgabeumlenkung).

cos - Cosinus

cos berechnet den Cosinus einer Zahl.

cos(x)

x
Zahl, für die der Cosinus berechnet werden soll.

exp - Exponentialfunktion

exp berechnet e hoch x.

exp(x)

x
Zahl, für die e^x berechnet werden soll.

getline - Einen Satz einlesen

awk liest einen Satz ein (siehe *next*, *Ablaufanweisung*).
getline hat mehrere Formate. Die Formate von *getline* haben folgende Rückgabewerte:

1 bei fehlerfreiem Lesevorgang
0 bei Dateiende
-1 bei einem Fehler

getline

awk liest in $0 den nächsten Eingabesatz aus der Eingabedatei ein. *NR*, *NF*, *FNR*, *$0* und *$1* bis *$NF* werden neu gesetzt.

Beispiel

Wenn ein Satz %%% enthält, wird der nächste Satz eingelesen, d.h. Eingabesätze, die %%% enthalten, werden ignoriert.

```
/%%%/ {getline}
```

getline < datei

awk liest in $0 einen Satz aus der Datei *datei* ein. *NF*, *$0* und *$1* bis *$NF* werden neu gesetzt.

datei
Name der Datei, aus der gelesen werden soll.

getline var

awk liest in die Variable *var* den nächsten Eingabesatz aus der Eingabedatei ein. *NR* und *FNR* werden neu gesetzt.

var
Variable, in die der nächste Satz eingelesen werden soll.

getline var < datei

awk liest in die Variable *var* einen Satz aus der Datei *datei* ein. *NR*, *NF*, *FNR*, *$0* und *$1* bis *$NF* werden nicht verändert.

var
Variable, in die der Satz eingelesen werden soll.

datei
Name der Datei, aus der gelesen werden soll.

kommando | **getline** [var]

Die Ausgabe des Kommandos *kommando* wird nach *getline* umgelenkt. Mit jedem *getline*-Aufruf in diesem Format liest *awk* in $0 bzw. *var* jeweils die nächste Zeile aus der Ausgabe von *kommando* ein.
Ist *var* angegeben, werden *NR*, *NF*, *FNR*, *$0* und *$1* bis *$NF* nicht verändert. Ist *var* nicht angegeben, werden *NF*, *$0* und *$1* bis *$NF* neu gesetzt.

Dieses Konstrukt wirkt wie der Aufruf der C-Funktion *popen()* mit Modus *r*.

var
Variable, in die der Satz eingelesen werden soll.

var nicht angegeben:
Der Satz wird in $0 eingelesen.

kommando
Name des Kommandos, aus dessen Ausgabe gelesen werden soll.

gsub - Globale Substitutionsfunktion

gsub ersetzt alle Zeichenketten in $0 bzw. in *in*, die zu dem erweiterten regulären Ausdruck *re* passen, durch die Zeichenkette *repl*.
gsub liefert die Anzahl der Ersetzungen zurück.

gsub(re,repl[,in])

re
Erweiterter regulärer Ausdruck, der als Muster für die Ersetzung dienen soll.

repl
Zeichenkette, die die zu *re* passenden Zeichenketten ersetzen soll.

in
Zeichenkette, in der die Ersetzung stattfinden soll.

in nicht angegeben:
Die Zeichenketten werden in $0 ersetzt.

index - Teilzeichenkette suchen

index sucht eine Teilzeichenkette in einer Zeichenkette. Wenn die Teilzeichenkette vorkommt, gibt *index* die Anfangsposition (in Zeichen, numeriert ab 1) des ersten Vorkommens zurück. Wenn die Teilzeichenkette nicht vorkommt, gibt *index* 0 zurück.

index(zk1,zk2)

zk1
Zeichenkette, in der *index* die Teilzeichenkette sucht.

zk2
Teilzeichenkette, die *index* sucht.

Beispiel

```
index("ToTo-LoTo","To") ist 1.
```

int - Ganzzahliger Anteil

int gibt den ganzzahligen Anteil zu einer Zahl zurück.

int(x)

x
Zahl, deren ganzzahliger Anteil bestimmt werden soll.

length - Länge bestimmen

length bestimmt die Länge einer Zeichenkette.

length[(zk)]

zk
length bestimmt die Länge der Zeichenkette *zk*.

zk nicht angegeben:
length bestimmt die Länge des aktuellen Eingabesatzes $0.

log - Logarithmus

log berechnet den natürlichen Logarithmus zur Basis *e*.

log(x)

x
Zahl, für die der natürliche Logarithmus berechnet werden soll.

match - Prüffunktion für reguläre Ausdrücke

match prüft, ob eine Zeichenkette in *zk* zu dem erweiterten regulären Ausdruck *re* paßt. *match* liefert die Stelle in *zk* (in Zeichen, numeriert ab 1) zurück, an der die zu *re* passende Zeichenkette anfängt; wenn keine Zeichenkette in *zk* paßt, liefert *match* 0 zurück. Die Variable *RSTART* wird auf den Return-Wert von *match* gesetzt. Die Variable *RLENGTH* wird auf die Länge der passenden Zeichenkette gesetzt bzw. auf -1, falls keine Zeichenkette paßt.

match(zk,re)

zk
: Zeichenkette, in der die Mustersuche stattfinden soll.

re
: Erweiterter regulärer Ausdruck.

print - Standard-Ausgabefunktion

print ist die Standard-Ausgabefunktion. *print* gibt entweder den aktuellen Satz oder die angegebenen Argumente aus und schließt die Ausgabe mit dem Ausgabesatztrenner *ORS* (ORS - Output Record Separator) ab. Nähere Beschreibung des Ausgabeformats siehe *Ausgabeformat*.

print(arg1[[,]arg2]...)[umlenkung]

Kein Argument angegeben:
: *print* gibt den aktuellen Eingabesatz auf die Standard-Ausgabe aus.

arg1arg2
: Argumente, die ausgegeben werden sollen. *print* wertet die Argument-Ausdrücke aus und hängt die Ergebnisse in der Reihenfolge der Argumente hintereinander.

arg1,arg2
: Argumente, die ausgegeben werden sollen. *print* gibt die ausgewerteten Argument-Ausdrücke in der angegebenen Reihenfolge aus, trennt sie aber durch den Ausgabe-Feldtrenner *OFS* (OFS - Output Field Separator).

umlenkung

Sie können die Ausgabe in eine Datei umlenken oder an ein Programm weiterreichen. Maximal können Sie 10 Ausgabedateien verwenden.

umlenkung kann sein:
>, >>, Name eines Programms

>datei

Die Ausgabe wird in die Datei *datei* geschrieben. Der alte Inhalt von *datei* wird beim ersten *print*-Aufruf gelöscht. Alle weiteren *print*- oder *printf*-Ausgaben auf *datei* in demselben *awk*-Programm werden angehängt. *datei* bleibt bis zum Ende des *awk*-Programms geöffnet, falls sie nicht explizit geschlossen wird.

>>datei

Die Ausgabe wird an den bisherigen Inhalt der Datei *datei* angehängt. *datei* bleibt bis zum Ende des *awk*-Programms geöffnet, falls sie nicht explizit geschlossen wird.

|prog

Die Ausgabe wird über eine Pipe an das Programm *prog* geschickt.

Sie dürfen innerhalb eines *awk*-Programms nur eine Pipe zu *prog* öffnen. Auf diese Pipe können mehrere *print*- oder *printf*-Ausgaben erfolgen.
Dieses Konstrukt wirkt wie der Aufruf der C-Funktion *popen()* mit Modus *w*.
Die Pipe bleibt bis zum Ende des *awk*-Programms geöffnet, falls sie nicht explizit geschlossen wird.

Den Dateinamen bzw. Programmnamen können Sie direkt angeben, eingeschlossen in "...", oder durch eine Variable, die den Dateinamen enthält.

Vorsicht

Wenn Sie auf die Eingabedatei umlenken, wird die Eingabedatei ohne Warnung zerstört!

Ausgabeformat

print gibt ganze Zahlen dezimal aus und Zeichenketten in ihrer vollen Länge. Ansonsten ist das Ausgabeformat durch folgende vordefinierte Variablen festgelegt:

OFS - Ausgabe-Feldtrenner
OFS ist standardmäßig ein Leerzeichen. Sie können *OFS* ein beliebiges Zeichen zuweisen und damit den Ausgabe-Feldtrenner ändern.

ORS - Ausgabe-Satztrenner
ORS ist standardmäßig das Neue-Zeile-Zeichen. Sie können *ORS* ein beliebiges Zeichen zuweisen und damit den Ausgabe-Satztrenner ändern.

OFMT - Gleitkomma-Ausgabeformat
(OFMT - output format) *OFMT* enthält das Ausgabeformat für Gleitkommawerte. *OFMT* ist standardmäßig "%.6g". Das bedeutet, daß Gleitkommazahlen mit maximal 6 Stellen hinter dem Dezimalpunkt ausgegeben werden. Sie können *OFMT* ein anderes *printf*-Format für Gleitkommazahlen zuweisen (siehe *Funktionen, Formatierte Ausgabe*).

Beispiele

1. Erstes und zweites Feld, getrennt durch Leerzeichen, ausgeben:

```
{print $1,$2}
```

2. Erstes und zweites Feld ohne Ausgabe-Feldtrenner aneinanderhängen:

```
{print $1$2}
```

oder

```
{print $1 $2}
```

printf - Formatierte Ausgabe

printf ist die Ausgabefunktion für formatierte Ausgabe. Sie können das Ausgabeformat wie bei der Standard-C-Funktion *printf()* angeben.

printf(format,arg,...)[umlenkung]

format
Zeichenkette, die das Ausgabeformat enthält. Das Ausgabeformat besteht aus Zeichen und Formatangaben. Die darstellbaren Zeichen werden unverändert ausgegeben. Die im Abschnitt Grundelemente angegebenen Sonderzeichen werden gleich umgesetzt. So wird z.B. durch \n auf den Anfang der nächten Zeile positioniert.
Eine Formatangabe wird mit dem Prozent-Zeichen % eingeleitet. Die wichtigsten Formatelemente sind in der folgenden Tabelle zusammengestellt.

Formatelement	Bedeutung
%c	Einzelnes Zeichen
%d	Ganzzahl dezimal
%e	Gleitkommazahl in Exponent-Darstellung, z.B. 5.234e+2
%f	Gleitkommazahl, z.B. 52.34
%g	%e oder %f, je nachdem, welche Darstellung kürzer ist
%o	Ganzzahl oktal (Basis 8)
%s	Zeichenkette
%u	Ganzzahl dezimal ohne Vorzeichen
%x	Ganzzahl hexadezimal (Basis 16)

arg
Argumente, die ausgegeben werden sollen.
printf wertet die Argument-Ausdrücke aus, ordnet sie den Formatangaben in *format* in der angegebenen Reihenfolge zu und gibt sie entsprechend formatiert aus.

- Wenn Formatangabe und Argument nicht zusammenpassen (Formatangabe numerisch, Argument alphanumerisch), wird 0 ausgegeben.
- Wenn mehr Argumente vorhanden sind als Formatangaben, werden die überzähligen Argumente ignoriert, d.h. nicht ausgegeben.
- Wenn mehr Formatangaben vorhanden sind als Argumente, wird eine Fehlermeldung ausgegeben.

umlenkung
Bezüglich Umlenkung gilt dasselbe wie bei *print*.

umlenkung nicht angegeben:
printf gibt auf die Standard-Ausgabe aus.

Beispiel

Feld 1 wird als Dezimalzahl mit mindestens 2 Stellen ausgegeben, danach, getrennt durch **, Feld 2 als Zeichenkette mit mindestens 5 Zeichen und danach Neue Zeile:

```
{ printf("%2d**%5s\n", $1,$2) }
```

rand - Zufallszahl liefern

rand liefert eine Zufallszahl *r* zurück, für die gilt: 0 < = *r* < 1.

rand

Siehe auch *srand*.

sin - Sinus

sin liefert den Sinus einer Zahl.

sin(x)

x
Zahl, deren Sinus berechnet werden soll.

sprintf - Formatierte Ausgabe in Zeichenkette

Bei *sprintf* ist die Formatierung wie bei *printf*. Es erfolgt aber keine Ausgabe. Stattdessen gibt *sprintf* die formatierte Ausgabe als Ergebnis-Zeichenkette zurück. Diese Zeichenkette können Sie z.B. einer Variablen zuweisen.

sprintf(format,arg,...)

format
Zeichenkette, die das Ausgabeformat enthält (siehe *Formatierte Ausgabe*).

arg
Argumente, die ausgegeben werden sollen (siehe *Formatierte Ausgabe*).

Beispiel

Das folgende *awk*-Programmstück erzeugt diesselbe Ausgabe wie das Beispiel bei *printf*.

```
{ x = sprintf("%2d**%5s\n", $1,$2); print x }
```

split - Zeichenkette aufteilen

split teilt eine Zeichenkette in Teilzeichenketten auf und speichert die Teilzeichenkette als Elemente eines Arrays ab. Die Array-Elemente werden beginnend mit 1 aufsteigend indiziert.
split liefert die Anzahl der Elemente des Arrays zurück.

split(zk,array[,sep])

zk
Zeichenkette, die aufgeteilt werden soll.

array
Name des Ergebnis-Arrays.

sep
Erweiterter regulärer Ausdruck, der die Zeichen bestimmt, die in *zk* als Trennzeichen zwischen den Teilzeichenketten gelten sollen.

sep nicht angegeben:
FS gilt als Trennzeichen.

Beispiel

```
{
   s=split("januar:februar:maerz", monate, ":");
   for(i=1; i<s; i++) print monate[i];
}
```

erzeugt die Ausgabe:

```
januar
februar
maerz
```

sqrt - Quadratwurzel berechnen

sqrt berechnet die Quadratwurzel zu einer Zahl.

sqrt(x)

x
Zahl, deren Quadratwurzel berechnet werden soll.

srand - Anfangs-Berechnungswert für die Funktion rand setzen

srand setzt den Anfangs-Berechnungswert für die Funktion *rand* auf die Zahl *x* bzw. auf die aktuelle Uhrzeit, falls kein Argument angegeben ist.

srand([x])

x
Zahl, die als Anfangs-Berechnungswert für *rand* dienen soll.

sub - Globale Substitutionsfunktion

sub ersetzt die erste Zeichenkette in $0 bzw. in *in*, die zu dem erweiterten regulären Ausdruck *re* paßt, durch die Zeichenkette *repl*.
sub liefert die Anzahl der Ersetzungen zurück.

sub(re,repl[,in])

re
Erweiterter regulärer Ausdruck, der als Muster für die Ersetzung dienen soll.

repl
Zeichenkette, die die zu *re* passenden Zeichenketten ersetzen soll.

in
Zeichenkette, in der die Ersetzung stattfinden soll.

in nicht angegeben:
Die Zeichenketten werden in $0 ersetzt.

substr - Teilzeichenkette bestimmen

substr liefert eine Teilzeichenkette einer Zeichenkette.

substr(zk,m[,n])

zk
: Zeichenkette, aus der eine Teilzeichenkette ausgewählt werden soll.

m
: Position in *zk*, bei der die Teilzeichenkette beginnt. Die Zeichenpositionen werden beginnend mit 1 von links nach rechts aufsteigend durchnumeriert.

n
: Maximale Länge der Teilzeichenkette.

  n nicht angegeben:
  Die Teilzeichenkette geht bis zum Ende von *zk*.

Beispiel

```
{
   x = substr("060789",3,2); print "Monat = "x
}
```

erzeugt die Ausgabe:

```
Monat = 07
```

system - Shell-Kommando ausführen

system führt das angegebene Shell-Kommando aus und liefert den Ende-Status des Kommandos zurück.

system(kommando)

kommando
: Name des Shell-Kommandos, das ausgeführt werden soll.

FEHLERMELDUNGEN

Wenn das *awk*-Programm fehlerhaft ist, gibt *awk* Fehlermeldungen aus und beendet sich sofort. Die Fehlermeldungen enthalten die Fehlerursache, falls *awk* diese erkennt, sowie die Zeile des *awk*-Programms, in der *awk* den Fehler vermutet.
Typische Fehlermeldungen sind:

```
awk: syntax error at source line xxx
```

In der Zeile xxx des *awk*-Programms liegt ein Syntaxfehler vor.

```
awk: illegal statement source line number xxx
```

In der Zeile xxx des *awk*-Programms steht eine falsche Anweisung.

BEISPIELE

1. Sie wollen alle Eingabezeilen auswählen, bei denen Feld 3 größer ist als Feld 5:

   ```
   $ awk '$3 > $5' datei
   ```

   Da keine Aktion angegeben ist, gibt *awk* standardmäßig die ausgewählten Zeilen aus.

2. Sie wollen jede 10. Zeile der Datei *datei* ausgeben:

   ```
   $ awk '(NR % 10) == 0' datei
   ```

3. Sie wollen für jede Zeile das vorletzte und letzte Feld, getrennt durch Doppelpunkt, ausgeben:

   ```
   $ awk 'BEGIN {OFS=":"}  \
   >       {print $(NF-1), $NF}' datei
   ```

   Bei Zeilen, die aus einem einzigen Feld bestehen, wird die ganze Zeile zweimal, getrennt durch Doppelpunkt, ausgegeben (zuerst $0, dann $1).

4. Sie wollen die Werte des ersten Feldes von allen Zeilen addieren. Zum Schluß sollen Summe und Mittelwert ausgegeben werden.

   ```
   $ awk '{s += $1} \
   >       END {print "Summe: ", s, "Mittelwert: ", s/NR}'\
   >       datei
   ```

5. Präprozessor-*if*-Anweisung heraussuchen, d.h. einen Bereich auswählen, dessen erste Zeile mit #if und dessen letzte Zeile mit #endif beginnt:

```
$ awk '/^#if/, /^#endif/' datei
```

6. Sie wollen alle Zeilen ausgeben, deren erstes Feld sich von dem ersten Feld der vorhergehenden Zeile unterscheidet:

```
$ awk '$1 != vorher { print; vorher = $1 }' datei
```

7. *datei* enthält eine Tabelle mit Daten über Jugendliche. In dem zweiten Feld steht entweder *Schueler*, *Student*, *Lehrling* oder *Sonstige*. Für eine Statistik sollen Schüler und Studenten gezählt werden:

```
$ awk '$2 ~ /Schueler/ {haeuf["Schueler"]++} \
>      $2 ~ /Student/ {haeuf["Student"]++} \
>      END {print "Schueler:" haeuf["Schueler"]; \
>           print "Student:" haeuf["Student"]} ' datei
```

8. In der Datei *inhalt* steht ein mit Dezimalklassifikation gegliedertes Inhaltsverzeichnis eines Textes. Die Liste hat folgendes Format:

```
1. Vorwort
2. Einleitung
3. Das Schachspiel
3.1. Geschichte
3.2. Regeln
3.2.1 Aufstellen der Figuren
.
.
.
4. Das Damespiel
4.1. Geschichte
.
.
.
8. Register
```

Mit folgendem *awk*-Programm bringen Sie diese Liste in ein übersichtlicheres Format.

```
$ awk '{$1=$1"          ";     \
>      $1=substr($1,1,6);  \
>      print $0}' inhalt  >> inh.form
```

Die Aufbereitung der Ausgabezeilen erfolgt in folgenden Schritten: Zuerst werden an das erste Feld sechs Leerzeichen angehängt (*$1 = $1"␣␣␣␣␣␣"*). Dann wird das erste Feld auf Länge sechs gekürzt. Damit ist in jeder Zeile das erste Feld 6 Zeichen lang und Feld 2 beginnt immer auf Spalte sieben. Sie erhalten folgende Ausgabe in der Datei *inh.form*:

```
1.     Vorwort
2.     Einleitung
3.     Das Schachspiel
3.1.   Geschichte
3.2.   Regeln
3.2.1 Aufstellen der Figuren
.
.
.
4.     Das Damespiel
4.1.   Geschichte
.
.
.
8.     Register
```

9. Das folgende *awk*-Programm in der Datei *prog* gibt für jeden Satz die Feldanzahl und die Felder aus. Der Satztrenner wird auf das Dollarzeichen $ umdefiniert. Feldtrenner sind daher Leer-, Tabulator- und Neue-Zeile-Zeichen:

```
BEGIN { RS="$"; printf "Satz\tAnz" }
      { printf ("\n%4d\t%3d\t", NR, NF);
        for(i=1;i<=NF; i++) printf "%s:", $i }
END {print"\n"}
```

Die Datei *text* enthält folgenden Text:

```
erster Satz$  zweiter   Satz     $
$
vierter     und  letzter
Satz$
```

Der Aufruf:

```
$ awk -f prog text
```

liefert:

```
Satz    Anz
   1      2      erster:Satz:
   2      2      zweiter:Satz:
   3      0
   4      4      vierter:und:letzter:Satz:
   5      0
```

10. Sie ändern die Datei *text* wie folgt:

```
&&
erster&&Satz$zweiter Satz$$vierter
und&
letzter

Satz&
```

und rufen *awk* auf, dieses Mal mit der Option *-F*, um den Feldtrenner auf & zu ändern.

```
$ awk -F"&" -f prog text
```

Die Ausgabe ergibt:

```
Satz    Anz
   1      6    :::erster::Satz:
   2      1    zweiter Satz:
   3      0
   4      8    vierter:und::letzter::Satz:::
```

Dieses Besipiel verdeutlicht die Feldeinteilung bei einem Nicht-Standard-Feldtrenner. Die erste Zeile (&&) der Datei Text ist Teil des ersten Satzes und ergibt z.B. 3 Felder, weil bei einer Folge von Trennern jeder Trenner zählt und Neue Zeile implizit auch als Trenner gilt (2 & + 1 Neue Zeile = 3).

SIEHE AUCH

egrep, *fgrep*, *grep*, *sed*
lex [19]

Kernighan, B.W.; Pike, R.
The UNIX Programming Environment [36]

Tare, R.S.:
UNIX Utilities [48]

Bach, F. et al.
UNIX: Handbuch zur Programmentwicklung [30]

Aho, A.; Kernighan, B.W.; Weinberger, P.:
The AWK Programming Language [28]

banner
Zeichen in großer Darstellung ausgeben

banner schreibt Zeichenketten wie im Beispiel gezeigt in groß gestalteter Darstellung auf die Standard-Ausgabe, wobei jede Zeichenkette in eine eigene Zeile gesetzt wird. Für die Ausgabe wird der US-ASCII-Zeichensatz (siehe *Tabellen und Verzeichnisse, Zeichensatz ISO 646*) verwendet.

banner zeichenkette ...

zeichenkette
zeichenkette besteht aus beliebigen alphanumerischen Zeichen. Mehrere Zeichenketten sind durch ein oder mehrere Leerzeichen voneinander getrennt. Leerzeichen innerhalb einer Zeichenkette und Sonderzeichen der Shell (siehe *Tabellen und Verzeichnisse, Sonderzeichen der Bourne-Shell sh*) müssen Sie einzeln mit einem Gegenschrägstrich \ entwerten oder in Hochkommas '...' oder Anführungszeichen "..." einschließen, um sie vor der Auswertung durch die Shell zu schützen. In der Ausgabe wird je Zeile eine Zeichenkette von maximal 10 Zeichen dargestellt. Überzählige Zeichen werden ignoriert. Zwischen Klein- und Großbuchstaben in der Zeichenkette wird unterschieden. Die Kleinbuchstaben werden um eine Zeile kleiner ausgegeben.

BEISPIEL

Ausgabe der Zeichenkette *Dies ist ein Banner!* in großer Darstellung:

```
$ banner Dies ist ein Banner!
```

```
######
#     #     #    ######   ####
#     #     #    #       #
#     #     #    #####    ####
#     #     #    #           #
#     #     #    #       #    #
######      #    ######   ####

    #     ####    #####
    #    #            #
    #     ####        #
    #         #       #
    #    #    #       #
    #     ####        #

######      #     #    #
#           #     ##   #
#####       #     # #  #
#           #     #  # #
#           #     #   ##
######      #     #    #

######                                          ###
#     #    ##     #    #  #    #  ###### #####   ###
#     #   #  #    ##   #  ##   #  #      #    #  ###
######   #    #   # #  #  # #  #  #####  #    #   #
#     #  ######   #  # #  #  # #  #      #####
#     #  #    #   #   ##  #   ##  #      #   #   ###
######   #    #   #    #  #    #  ###### #    #  ###
```

SIEHE AUCH

echo

basename
Dateinamen vom Pfad trennen

Mit *basename* können Sie

- aus dem Pfadnamen einer Datei den einfachen Dateinamen dieser Datei erzeugen,
- aus Dateinamen beliebige Endungen entfernen.

basename entfernt aus einer beim Aufruf angegebenen Zeichenkette alle Zeichen bis einschließlich des letzten vorkommenden Schrägstrichs /. Den Rest gibt *basename* auf die Standard-Ausgabe aus. Sie können damit den einfachen Dateinamen einer Datei vom Pfad-Präfix trennen. Wenn Sie beim Aufruf von *basename* zusätzlich die Endung der Zeichenkette angeben, entfernt *basename* auch diese Endung. *basename* findet nützliche Anwendung in Shell-Prozeduren zwischen Gegenhochkommas `...` (siehe *sh*).

basename zeichenkette[endung]

zeichenkette
zeichenkette ist eine beliebige Zeichenkette.
basename entfernt aus *zeichenkette* alle Zeichen bis einschließlich des letzten Schrägstrichs /. Den Rest gibt *basename* auf die Standard-Ausgabe aus. Zeichenketten, die das Zeichen / nicht enthalten, werden unverändert ausgegeben.

endung
endung ist eine beliebige Zeichenkette.
Wenn *endung* mit dem Ende von *zeichenkette* übereinstimmt, wird *zeichenkette* ohne *endung* ausgegeben.

BEISPIELE

1. Aus */home/katharina/programm* soll der Name *prog* erzeugt werden:

```
$ basename /home/katharina/programm ramm
prog
```

2. Die folgende Shell-Prozedur übersetzt ein C-Quellprogramm. *basename* erzeugt den Namen des übersetzten Programms aus dem Dateinamen, der beim Aufruf der Shell-Prozedur übergeben wird. Das übersetzte Programm wird in einer ausführbaren Datei im aktuellen Dateiverzeichnis abgelegt. Die Shell-Prozedur steht in der Datei *uebersetze*.

   Inhalt von *uebersetze*:

```
cc -o `basename $1 .c` $1
```

   Wenn Sie *uebersetze* wie folgt aufrufen:

```
$ uebersetze /home/anna/cprogs/tab.c
```

   wird dem Kommando *cc* (Steuerprogramm zum Übersetzen und Binden von C-Programmen, siehe *cc* [19]) im Stellungsparameter $1 (siehe *sh*) der Name der C-Quelldatei übergeben. Die Shell ersetzt den Operanden für die Option *-o* von *cc* durch das Ergebnis des Aufrufs von *basename*. Der Name der ausführbaren Datei ist *tab*.

SIEHE AUCH

dirname, *ed*, *sh*

batch
Kommandos zu einer späteren Zeit ausführen

batch liest Kommandos von der Standard-Eingabe, setzt sie in eine Warteschlange und führt sie aus, sobald die Systembelastung es zuläßt.

Wenn Sie die Standard-Ausgabe und Standard-Fehlerausgabe der auszuführenden Kommandos nicht umgelenkt haben, wird Ihnen die Ausgabe mit *mail* geschickt. Die Umgebungsvariablen, das aktuelle Dateiverzeichnis, die für neue Dateien gültigen Zugriffsrechte (siehe *umask*) und die maximal zulässige Dateigröße (siehe *ulimit*) bleiben erhalten. Offene Dateien und Prioritäten werden nicht vererbt. Das Kommando *trap* (eingebautes Shell-Kommando zum Abfangen von Signalen) wird aufgehoben.

batch schreibt die Auftragsnummer und die berechnete Ausführungszeit auf die Standard-Fehlerausgabe.

Aufträge, die mit *batch* erteilt werden, werden auch dann abgearbeitet, wenn der Auftraggeber sich vom System abgemeldet hat.

batch verhält sich genau wie *at -qb* ohne zusätzliche Option.

Vor dem Aufruf beachten

Wenn die Datei */etc/cron.d/at.allow* existiert, dann dürfen Sie das Kommando *batch* nur dann aufrufen, wenn Ihre Benutzerkennung in dieser Datei steht.
Wenn die Datei */etc/cron.d/at.allow* nicht existiert, dann dürfen Sie das Kommando *batch* nur dann aufrufen, wenn Ihre Benutzerkennung *nicht* in der Datei */etc/cron.d/at.deny* steht.
Wenn weder */etc/cron.d/at.allow* noch */etc/cron.d/at.deny* existieren, dann darf nur der Systemverwalter *batch* aufrufen.

Existiert z.B. nur die leere deny-Datei, so dürfen alle Benutzer *batch* aufrufen.

Die allow/deny-Dateien darf nur der Systemverwalter anlegen und ändern. Sie enthalten pro Zeile eine Benutzerkennung.

batch ↵
kommando ... ↵
END

kommando
: Beliebiges Kommando oder Shell-Prozedur. Sie können *kommando* mehrmals, jeweils durch Strichpunkt ; oder Neue-Zeile-Zeichen getrennt, angeben. Eine so entstandene Kommandoliste läuft unter einer Auftragsnummer.

FEHLERMELDUNG

```
at: you are not authorized to use at. Sorry.
```

Sie dürfen *batch* nicht aufrufen. Siehe *Vor dem Aufruf beachten*.

DATEIEN

/etc/cron.d/at.allow
: Liste der Benutzerkennungen mit Ausführrecht für *batch*. In jeder Zeile steht jeweils eine Benutzerkennung.

/etc/cron.d/at.deny
: Liste der Benutzerkennungen ohne Ausführrecht für *batch*. In jeder Zeile steht jeweils eine Benutzerkennung.

/var/spool/cron/atjobs
: Dateiverzeichnis, in dem die noch nicht bearbeiteten *batch*-Aufträge in einzelnen Dateien aufgelistet werden. Für jeden *batch*-Auftrag gibt es eine eigene Datei mit dem Dateinamen *auftragsnummer.b*. Diese Dateien haben die Zugriffsrechte -r-S--S--- (siehe *chmod*).

/etc/cron.d/queuedefs
: Enthält Ablaufinformationen.

BEISPIEL

Im folgenden Beispiel wird die Standard-Eingabe umgelenkt, so daß *batch* seinen Auftrag aus der Datei *auftraege* liest:

```
$ batch < auftraege
job 604763316.b at Wed Mar  1 14:48:36 1989
```

batch führt die Aufträge, die in der Datei *auftraege* stehen, der Reihe nach als Hintergrundprozesse aus. Nach Beendigung können Sie sich das Ergebnis mit *mail* auf dem Bildschirm ausgeben lassen.

SIEHE AUCH

at, *crontab*, *mail*

bc
Arithmetische Sprache

Mit *bc* können Sie rechnen. *bc* ist ein interaktives Programm für eine C-ähnliche Eingabe-Sprache; *bc* ist eigentlich ein Präprozessor für *dc*; *bc* ruft den Tischrechner *dc* automatisch auf und gibt die Ergebnisse mit beliebiger Genauigkeit auf die Standard-Ausgabe aus.

bc[-c][-d][-l][datei]...

-c
-d
(c - compile only) *bc* erzeugt nur Anweisungen für den Tischrechner *dc* und gibt diese auf die Standard-Ausgabe aus. *bc* ruft *dc* jedoch nicht auf. Die Option *-d* hat die gleiche Wirkung wie die Option *-c*.

-l
-l steht für */usr/lib/lib.b*. In dieser Bibliothek sind *bc*-Programme für verschiedene mathematische Funktionen enthalten.
Sie müssen die Option *-l* angeben, wenn Sie eine der folgenden Funktionen verwenden:

s(x)	Sinus
c(x)	Cosinus
e(x)	Exponentialfunktion mit Basis e
l(x)	natürlicher Logarithmus
a(x)	Arcustangens
j(n,x)	Bessel-Funktion n-ter Ordnung

datei
Name der Datei mit einem *bc*-Programm. Sie können mehrere Dateien angeben. Nachdem alle Anweisungen aus allen Dateien abgearbeitet sind, liest *bc* von der Standard-Eingabe. Sie können dann weitere Anweisungen eingeben.

datei nicht angegeben:
bc liest von der Standard-Eingabe.

ELEMENTE VON BC-PROGRAMMEN

Ein *bc*-Programm besteht aus

- Definitionen
- Anweisungen
- Kommentaren

Bei der Beschreibung des Aufbaus von *bc*-Programmen werden folgende Bezeichnungen verwendet:

L (L - letter) steht für einen der Buchstaben a-z
E (E - expression) steht für einen Ausdruck
S (S - statement) steht für eine Anweisung

Kommentare

Kommentare werden wie in C-Programmen in /*...*/ eingeschlossen.

Anweisungen

Anweisungen in *bc*-Programmen können sein:

- Ausdrücke (siehe *Ausdrücke*)
  Das Ergebnis einer Anweisung, die ein Ausdruck ist, wird ausgegeben, es sei denn, der Hauptoperator ist ein Zuweisungsoperator.
- Blöcke: {S; ...; S}
- Auswahlanweisung: if (E) S
  Wenn der Ausdruck E wahr ist, d.h. einen Wert ungleich 0 hat, dann wird die Anweisung S ausgeführt.
- Wiederholungsanweisungen:
  - while (E) S
    Der Ausdruck E wird ausgewertet und wenn sein Wert ungleich 0 ist, dann wird die Anweisung S ausgeführt. Danach wird E erneut ausgewertet und S ggf. wiederum ausgeführt. Dies wiederholt sich, solange der Wert von E ungleich 0 ist.
  - for (E; E; E) S
    Zuerst wird der erste Ausdruck ausgewertet. Danach wird der zweite Ausdruck ausgewertet und falls dessen Wert ungleich 0 ist, wird die Anweisung S ausgeführt. Schließlich wird der dritte Ausdruck ausgewertet. Anschließend wird wieder der zweite Ausdruck ausgewertet und gegebenenfalls Anweisung S ausgeführt usw. Anders als in C-Programmen muß eine for-Anweisung immer drei Ausdrücke enthalten.

- Sprunganweisung: break
  Die Anweisung break darf nur innerhalb einer Wiederholungsanweisung benutzt werden. Sie sorgt für den Abbruch der nächstgelegenen while- oder for-Anweisung. Das Programm wird mit der Anweisung fortgesetzt, die auf die abgebrochene Wiederholungsanweisung folgt.
- Abbruchanweisung: quit
  Die Anweisung quit beendet die Ausführung des *bc*-Programms. quit wird sofort beim Einlesen interpretiert, nicht erst bei der Ausführung des *bc*-Programms.

  Beispiel

  Das folgende *bc*-Programm beendet sich sofort, ohne daß der Wert von a ausgegeben wird:

```
a=5
if (a>10) quit
(a)
```

Anweisungen können Sie durch einen Strichpunkt oder ein Neue-Zeile-Zeichen voneinander trennen.

Ausdrücke

Ausdrücke bestehen aus Operanden und Operatoren.

Operanden sind Namen oder beliebig lange Zahlen, wahlweise mit Vorzeichen und Dezimalpunkt.

Namen

L	einfache Variablen
L	Funktionsnamen
L[E]	Feldelemente
ibase	Basis für eingelesene Zahlen, Standard (keine Angabe): 10
obase	Basis für ausgebene Zahlen, Standard (keine Angabe): 10
scale	Anzahl der Nachkommastellen, Standard (keine Angabe): 0

Wenn Felder als Argumente einer Funktion verwendet oder als auto-Variablen definiert werden, so müssen Sie hinter den Namen des Feldes leere eckige Klammern schreiben.

Eine einfache Variable, ein Feld und eine Funktion können denselben Namen haben.

Alle Variablen sind global im ganzen *bc*-Programm.

Weitere Operanden

(E)	Ergebnis von E
sqrt(E)	Quadratwurzel von E
length(E)	Zahl der signifikanten Dezimalziffern von E
scale(E)	Zahl der Nachkommastellen von E
L(E, ...,E)	

Operatoren

+ - * /	Addition, Subtraktion, Multiplikation, Division
^	Potenzieren
%	Rest bei ganzzahliger Division (Modulo)
++ --	Inkrement- und Dekrement-Operator, die sowohl in Präfix- wie in Postfix-Notation auf Namen angewendet werden können
< <= == >= > !=	Vergleichsoperatoren (echt kleiner, kleiner gleich, gleich, größer gleich, echt größer, ungleich)
=	Zuweisungsoperator
=@	zusammengesetzte Zuweisungsoperatoren; dabei bedeutet a=@b dasselbe wie a=a@b. Für @ können Sie einen der Operatoren + - * / ^ oder % angeben.

Die logischen Operatoren *&&* und || stehen in *bc nicht* zur Verfügung.

Definition von Funktionen

```
define L (L, ...,L) {
          auto L, ...,L
          S; ...;S
          return (E)
  }
```

Beispiel

```
   define p(x) {
         auto q
         q = p * p
         return (q)
     }
```

Indem die Ergebnisparameter einer Funktion als auto vereinbart werden, wird ihr Gültigkeitsbereich auf diese Funktion beschränkt. Alle Funktionsargumente werden als Werte übergeben.

Funktionen in der mathematischen Bibliothek /usr/lib/lib.b

In */usr/lib/lib.b* sind Definitionen der folgenden mathematischen Funktionen enthalten. Sie stehen Ihnen zur Verfügung, wenn Sie *bc* mit der Option *-l* aufrufen:

```
s(x)    Sinus
c(x)    Cosinus
e(x)    Exponentialfunktion mit Basis e
l(x)    natürlicher Logarithmus
a(x)    Arcustangens
j(n,x)  Bessel-Funktion n-ter Ordnung
```

Wenn Sie Schreibrecht für die Datei */usr/lib/lib.b* haben, können Sie die Definitionen weiterer Funktionen hinzufügen oder vorhandene Definitionen abändern oder löschen.

Basis für Ein- und Ausgaben festlegen

Mit *ibase* und *obase* legen Sie fest, in welchem Zahlensystem eingegebene bzw. ausgegebene Werte interpretiert werden. Dabei gelten folgende Regeln:

1. Wenn Sie *ibase* oder *obase* nicht explizit einen Wert zuweisen, werden eingegebene Zahlen im Dezimalsystem interpretiert und Ergebnisse im Dezimalsystem ausgegeben.
2. Wenn Sie mit der Anweisung *ibase = n* eine Basis für Eingaben festgelegt haben, müssen Sie die gewünschte Basis für Ausgaben in der Anweisung *obase = m* schon in der Eingabe-Basis *n* darstellen.

   Beispiel

   Basis für Eingaben soll 2, Basis für Ausgaben soll 16 sein:

   ```
   $ bc
   ibase=2
   obase=10000
   10100000/1010
   10
   ```

Nachkommastellen

Der Variablen *scale* können Sie die Anzahl der Nachkommastellen zuweisen, die die errechneten Ausdrücke haben sollen.

Beispiel

```
scale=8
```

bewirkt, daß Ergebnisse mit 8 Nachkommastellen ausgegeben werden.

DATEIEN

/usr/lib/lib.b
Mathematische Bibliothek

/usr/bin/dc
Tischrechner-Programm

BEISPIELE

1. Zahlen addieren, subtrahieren, multiplizieren, dividieren. Nicht-ganzzahlige Ergebnisse sollen mit zwei Nachkommastellen ausgegeben werden:

```
$ bc
scale=2
3+7
10
8-15
-7
7*6
42
3/5
.60
quit
$
```

2. Definition einer Funktion, die angenähert den Wert der Exponentialfunktion berechnet.

```
scale=20
define e(x){
      auto a, b, c, i, s
      a = 1
      b = 1
      s = 1
      for(i=1;1==1;i++){
              a = a*x
              b = b*i
              c = a/b
              if(c==0) return (s)
              s = s+c
        }
  }
```

Das Abbruchkriterium der for-Schleife ist im Schleifenrumpf enthalten (*if(c = =0)*) und wird nicht, wie sonst üblich, durch den zweiten Ausdruck der for-Anweisung gegeben. In einem C-Programm würde man den zweiten Ausdruck der for-Anweisung in einem solchen Fall einfach weglassen. Die for-Anweisung in einem *bc*-Programm muß jedoch immer drei Ausdrücke enthalten. Daher wird hier der Ausdruck *1 = = 1* eingefügt, der immer wahr ist.

3. Ausgeben der angenäherten Werte der Exponentialfunktion für die ersten 10 natürlichen Zahlen.

```
for(i=1;i<=10;i++) e(i)
```

4. Die Quadrate der ersten vier natürlichen Zahlen ausgeben:

```
$ bc
for(i=1;i<5;i++) {i*i}
1
4
9
16
quit
```

5. Ausgeben der Eingabe für *dc* beim Errechnen der Quadratzahlen von 1 bis 4.

```
$ bc -c
for(i=1;i<5;i++) {i*i}
[lili*ps.lidl+sis.li 5>0]s0
ldsis.li 5>0
```

SIEHE AUCH

dc

bdiff
Große Dateien vergleichen (big diff)

bdiff vergleicht zwei Dateien wie *diff*, kann jedoch Dateien vergleichen, die für *diff* zu groß sind.

bdiff gibt (genau wie *diff*) aus:

- die Zeilen, in denen sich die angegebenen Dateien unterscheiden,
- *ed*-Kommandos, mit denen man aus *datei1 datei2* erzeugen kann (siehe *ed*).

bdiff datei1 datei2 [n] [**-s**]

Die Reihenfolge der Argumente und Optionen ist nicht beliebig!

datei1 datei2
: *datei1* und *datei2* sind die Namen der Dateien, die Sie vergleichen wollen. Wenn Sie statt eines Dateinamens das Zeichen - angeben, dann liest *bdiff* von der Standard-Eingabe.

n
: Mit *n* können sie bestimmen, wieviele Zeilen die Abschnitte haben sollen, in die *bdiff* die Dateien aufteilt.
n ist eine ganze Zahl größer als 0.

  n nicht angegeben:
  n = 3500

-s
: Fehlermeldungen von *bdiff* werden teilweise unterdrückt. Fehlermeldungen von *diff* werden jedoch nicht unterdrückt. Diese Option wird vorwiegend in Shell-Prozeduren verwendet.

Vorsicht

Wenn Sie sowohl einen Wert für *n* als auch die Option *-s* angeben, dann müssen Sie die richtige Reihenfolge beachten: *n* muß vor *-s* stehen!

Arbeitsweise

Zeilen am Anfang der Dateien, die sich nicht unterscheiden, ignoriert *bdiff*. Ab der ersten unterschiedlichen Zeile teilt *bdiff* die angegebenen Dateien in Abschnitte zu *n* Zeilen auf. Der Standard-Wert für *n* ist 3500, kann aber beim Aufruf von *bdiff* geändert werden (siehe Parameter *n*). *bdiff* ruft *diff* auf. *diff* vergleicht dann die Abschnitte der Dateien. Sie müssen deshalb den Wert für *n* so wählen, daß *diff* keine Schwierigkeiten hat, Abschnitte mit *n* Zeilen zu vergleichen.

bdiff kann nur die Unterschiede zwischen sich entsprechenden Abschnitten in den beiden Dateien feststellen. *bdiff* vergleicht also z.B. nicht den 4. Abschnitt in *datei1* mit dem 5. Abschnitt in *datei2*. Deshalb gibt *bdiff* die Unterschiede oft umständlicher an, als man es sich wünschen würde.

Die Zeilennumerierung ist trotz der Aufteilung in Abschnitte von Dateianfang bis -ende durchlaufend, d.h., es sieht so aus, als seien die Dateien als Ganzes verarbeitet worden.

DATEIEN

/tmp/bd?????
Zwischendateien

BEISPIEL

Das Beispiel soll zeigen, wie sich die Aufteilung in Abschnitte auswirkt. Dazu muß zunächst eine Datei *abc1* mit 26 Zeilen angelegt werden. Fortlaufend steht in jeder dieser Zeilen in Spalte 1 ein Kleinbuchstabe des Alphabets; in Zeile 1 steht *a*, in Zeile 26 steht *z*. Diese Datei wird in die Datei *abc2* kopiert, und vor die erste Zeile der Datei *abc2* wird eine Zeile mit der Ziffer *0* in der ersten Spalte eingefügt. Der Vergleich beider Dateien erfolgt in Abschnitten von 5 Zeilen:

```
$ bdiff abc1 abc2 5
0a1
> 0
5d5
< e
5a6
> e
10d10
< j
10a11
> j
15d15
< o
15a16
> o
20d20
< t
20a21
> t
25d25
< y
25a26
> y
```

Zunächst werden die ersten 5 Zeilen beider Dateien verglichen. Damit die ersten 5 Zeilen der Datei *abc1* mit den ersten 5 Zeilen der Datei *abc2* übereinstimmen, muß folgendes geschehen:

- Vor die erste Zeile von *abc1* muß die erste Zeile (Ziffer *0*) von *abc2* eingefügt werden. *bdiff* gibt das *ed*-Kommando aus, mit dem Sie dies tun können:
  0a1
  (Tatsächlich fügt dieses *ed*-Kommando Zeile 1 aus *abc2* hinter Zeile 0 von *abc1* ein.)
- Die fünfte Zeile von *abc1* enthält den Buchstaben *e* und muß gelöscht werden, weil *abc2* innerhalb der ersten 5 Zeilen keine Zeile mit dem Buchstaben *e* enthält:
  5d5

Danach werden die Zeilen 6 - 10 beider Dateien verglichen. Vor Zeile 6 (Buchstabe *f*) von *abc1* muß Zeile 6 (Buchstabe *e*) von *abc2* eingefügt werden; Zeile 10 von *abc1* muß gelöscht werden. Damit sind beide Abschnitte gleich.

Weil *bdiff* abschnittsweise vergleicht, ist die Ausgabe von *bdiff* umständlicher als die Ausgabe von *diff*. Denn es hätte ja genügt, nur die erste Zeile von *abc2* vor die erste Zeile von *abc1* einzufügen.

SIEHE AUCH

cmp, *comm*, *diff*, *diff3*, *ed*

bfs
Große Dateien durchsuchen (big file scanner)

bfs ist ähnlich wie *ed*; man kann jedoch mit *bfs* nichts an der bearbeiteten Datei ändern. Dateien werden von *bfs* nur gelesen, wobei die Dateien viel größer sein können als bei *ed*.

Die Dateien können bis zu 1024 KByte groß sein. Die maximale Anzahl der Zeilen ist 32 K, die der Zeichen pro Zeile 512 (einschließlich Neue-Zeile-Zeichen).

bfs ist in der Regel effizienter als *ed*, da die zu durchsuchende Datei nicht in einen Puffer kopiert wird. *bfs* ist hilfreich, um Abschnitte einer großen Datei zu identifizieren, die durch *csplit* aufgeteilt werden soll, damit sich die einzelen Teile leichter editieren lassen.

bfs-Kommandos können interaktiv eingegeben werden oder in einer Datei stehen, aus der sie von *bfs* gelesen werden (siehe *Zusätzliche bfs-Kommandos, xf*).

bfs[␣-]␣datei

-
Unterdrückt die standardmäßige Ausgabe
- der Größe einer Datei
- der Fehlermeldungen

datei
Name der Datei, die Sie mit *bfs* durchsuchen möchten.

Ausgabe

bfs gibt die Größe der durchsuchten Datei sowie die Größe einer mit *w* geschriebenen Datei aus.

Wie auch bei *ed* erscheint nach der Eingabe von *P* ↵ als Bereitzeichen das Zeichen Stern * am Bildschirm. Falls im *-Modus ein Fehler auftritt, gibt *bfs* eine Fehlermeldung aus. Wenn Sie ein weiteres Mal *P* ↵ eingegeben, so wird die Anzeige des Bereitzeichens * und der Fehlermeldungen wieder ausgeschaltet.

Adressen

Adressen können Sie im gleichen Format angeben wie bei *ed*. Zusätzlich haben Sie folgende Möglichkeiten:

- Reguläre Ausdrücke können außer in /... / und ?...? auch in >...> und <...< eingeschlossen werden. > bedeutet hier eine Suche in Vorwärtsrichtung bis zum Dateiende, < eine Suche in Rückwärtsrichtung bis zum Dateianfang. Anders als bei / bzw. ? wird aber zur Fortsetzung der Suche nicht von Dateiende zum Dateianfang bzw. umgekehrt gesprungen.
- Zur Markierung der Zeilen sind nur die Buchstaben *a* bis *z* zulässig. Alle 26 Markierungen werden gespeichert.

Kommandos

Es gibt Kommandos, die sich ähnlich wie *ed*-Kommandos verhalten, sowie zusätzliche *bfs*-Kommandos. Beide Arten werden im folgenden getrennt alphabetisch aufgelistet.

ed-ähnliche Kommandos

Die folgenden Kommandos funktionieren ähnlich wie bei *ed*. Die vorhandenen kleinen Unterschiede werden jeweils genannt. Die genaue Beschreibung der Kommandos finden Sie bei *ed*.

e datei
: Es muß eine Datei angegeben werden; *datei* ist nicht optional. Falls Sie keine Datei angeben, wird die Datei . (das aktuelle Dateiverzeichnis) von *bfs* bearbeitet.

f
: Gibt den Namen der durchsuchten Datei aus.
Es gibt keinen gespeicherten Dateinamen.

p
: Das Kommando *1,10* hat die gleiche Bedeutung wie das Kommando *1,10p*: die ersten 10 Zeilen werden ausgegeben.

P
: Falls ein Fehler auftritt werden Fehlermeldungen ausgegeben.

q

w
: *w* ist unabhängig von einer Ausgabeumleitung, -abkürzung oder -komprimierung (siehe unten *xo*, *xt*, *xc*).

=

!

↵
leeres Kommando ohne Wirkung

Zusätzliche bfs-Kommandos

[bereich]**xb**/regulärer Ausdruck/label

bereich nicht angegeben:
Es wird nur ab der aktuellen Zeile gesucht.

Wird im angegebenen Bereich eine Zeichenfolge gefunden, die zu *regulärer Ausdruck* paßt, so wird diese Zeile zur aktuellen Zeile und es erfolgt ein Sprung (entweder in Vorwärts- oder in Rückwärtsrichtung) zu *label*. Ein *label* kann mit dem Doppelpunkt-Kommando *:label* (s.u.) gesetzt werden. Das *label* befindet sich in der *bfs*-Kommandodatei (siehe *xf*).

In den folgenden Situationen ist ein Sprung nicht möglich:

- Eine der Adressen von *bereich* liegt nicht zwischen *1* bis *$* (erste Zeile bis letzte Zeile der Datei).
- Die zweite Adresse liegt vor der ersten.
- Der reguläre Ausdruck paßt nicht zumindest zu einer Zeile im angegebenen Bereich (einschließlich der ersten und letzten Zeile).

Bei ungültigen Adressen gibt *xb* als einziges Kommando keine Fehlermeldungen aus, sondern es tut nichts. An diesem Verhalten erkennen Sie, daß die angegebene Adresse ungültig ist. Deshalb können Sie mit diesem Kommando vor der Ausführung anderer Kommandos prüfen, ob Adressen gültig sind.

Das Kommando

```
xb/^/label
```

bewirkt einen unbedingten Sprung, d.h., es muß keine Bedingung erfüllt sein.

Das Kommando *xb* ist nur zulässig, wenn es *nicht* von einer Datensichtstation gelesen wird. Wenn es aus einer Pipe gelesen wird, so ist nur ein Sprung in Vorwärtsrichtung möglich.

xbnlabel
xbzlabel
Bei diesen beiden Kommandos wird der zuletzt gespeicherte Ende-Status eines UNIX-Kommandos (!*kommando*) geprüft:

- *xbz* springt zum gesetzten Label, falls der Ende-Status 0 (wahr) ist.
- *xbn* springt zum gesetzten Label, falls der Ende-Status ungleich 0 (falsch) ist.

Andernfalls sind *xbn* und *xbz* wirkungslos.

xc[schalter]
Komprimierung der Ausgabe von *p* oder des *leeren Kommandos* einschalten oder ausschalten. In einer komprimierten Ausgabe werden aufeinanderfolgende Tabulator- und Leerzeichen bis auf ein Leerzeichen gestrichen und Leerzeilen entfallen.

schalter kann sein

0 Die Ausgabe wird nicht komprimiert (Standard).

1 Die Ausgabe wird komprimiert.

schalter nicht angegeben:
Ein gesetzter Schalter wird umgekehrt.

xfdatei
bfs liest aus der angegebenen *datei* zusätzliche Kommandos. Wenn das Dateiende erreicht ist, ein Unterbrechungssignal empfangen wurde oder ein Fehler aufgetreten ist, dann liest *bfs* das nächste Kommando in der Datei, die das *xf*-Kommando enthält.

Es können bis zu 10 *xf*-Kommandos ineinander verschachtelt werden.

xn
Die aktuellen Markierungen werden ausgegeben. Eine Zeile wird mit dem Kommando *k* markiert (siehe *ed*).

xo[datei]
Die nachfolgenden Ausgaben des Kommandos *p* und des leeren Kommandos werden in die angegebene *datei* umgeleitet. Falls die Datei noch nicht existiert, wird sie neu angelegt und erhält folgende Zugriffsrechte:

```
rw-------
```

Die Zugriffsrechte einer neu erstellten Datei hängen von *umask* ab.

Falls die Datei bereits existiert, wird sie auf die Länge 0 verkürzt und gegebenenfalls überschrieben; ihre Zugriffsrechte werden nicht verändert. Schreibgeschützte Dateien werden nicht überschrieben.

datei nicht angegeben:
Die Ausgabe wird auf die Standard-Ausgabe umgeleitet. Falls Sie die Ausgabe nacheinander in zwei verschiedene Dateien umleiten wollen, müssen Sie dazwischen die Ausgabe auf die Standard-Ausgabe umleiten.

xtn

Die Ausgabe des Kommandos *p* und des leeren Kommandos wird auf maximal *n* Zeichen gekürzt.

xtn nicht angegeben:
Die Ausgabe wird auf 255 Zeichen begrenzt.

xvziffer[␣][wert]

Mit *xv* können Sie einer Variablen einen Wert zuweisen. Den Variablennamen geben Sie durch die *ziffer* an, die auf das Kommando *xv* folgt. *ziffer* kann eine Zahl zwischen 0 und 9 sein.

Sowohl die Kommandoeingabe `xv5100`
als auch die Kommandoeingabe `xv5 100`
weisen der Variablen *5* den Wert *100* zu. Die Kommandoeingabe `xv61,100p`
weist der Variablen *6* den Wert *1,100p* zu.

Eine Variable wird adressiert, indem ihrem Namen ein % vorangestellt wird. So werden, wenn den Variablen *5* und *6* die obengenannten Werte zugewiesen wurden, mit jeder der folgenden Kommandoeingaben

```
1,%5p
1,%5
%6
```

die ersten 100 Zeilen ausgegeben.

Mit der Kommandoeingabe

```
g/%5/p
```

suchen Sie die Zeichenkette *100* global und jede Zeile, in der diese Zeichen enthalten sind, wird ausgegeben. Wenn Sie dem Prozent-Zeichen % einen Gegenschrägstrich \ voranstellen \%, dann wird die Sonderbedeutung von % aufgehoben.

Mit der Kommandoeingabe

```
g/".*\%[cds]/p
```

suchen Sie global nach einer Zeichenkette, die Anführungszeichen enthält, dann beliebig viele Zeichen gefolgt von einem Prozent-Zeichen vor einem der Buchstaben *c, d* oder *s*. Gefundene Zeichenketten werden ausgegeben. Damit suchen Sie *printf*-Anweisungen, wie sie das Kommando *awk* oder die Programmiersprache *C* verwendet.

Mit *xv* können Sie auch dafür sorgen, daß die erste Ausgabezeile eines SINIX-Kommandos in einer Variablen abgelegt wird. Dazu muß das erste Zeichen von *wert* ein Ausrufezeichen ! sein.

So kann etwa der Wert der Variablen *6* um eins erhöht werden, indem das Kommando *expr* verwendet wird:

```
xv6!expr %6 + 1
```

Ein etwas komplizierteres Beispiel:

```
.w tom
xv5!cat tom
!rm tom
!echo "%5"
```

Die aktuelle Zeile wird in die Datei *tom* geschrieben, dann mit *cat* aus *tom* gelesen und in der Variablen *5* abgelegt. Anschließend wird die Datei *tom* gelöscht. Die gespeicherte Zeile wird von *echo* ausgegeben (siehe *Tabellen und Verzeichnisse, Sonderzeichen der Bourne-Shell sh* und *sh, Sonderzeichen der Shell*).

Wenn Sie dem Ausrufezeichen ! einen Gegenschrägstrich \ voranstellen, wird die Sonderbedeutung von ! zu Beginn von *wert* aufgehoben.

```
xv7\!date
```

legt die Zeichenkette *!date* in der Variablen *7* ab.

wert nicht angegeben:
Die Variable enthält die Nummer der aktuellen Zeile.

:label

Ein *label* ist eine Zeichenkette, die als Sprungziel für die Kommandos *xb*, *xbn* und *xbz* dient. Sie können ein *label* in eine Kommandodatei (siehe *xf*) schreiben.

Sie schließen *label* mit einem Neue Zeile-Zeichen ab. Leerzeichen zwischen dem Doppelpunkt : und dem Beginn von *label* werden ignoriert. Mit dem Doppelpunkt-Kommando können Sie auch Kommentare in eine Kommandodatei einfügen, da Labels nicht unbedingt Sprungziele sein müssen.

FEHLERMELDUNGEN

Ein Fragezeichen ? wird bei fehlerhaften Kommandos ausgegeben, wenn sich *bfs* nicht im *-Modus befindet (siehe *ed-ähnliche Kommandos*, Kommando *P*).

Selbsterklärende Fehlermeldungen werden ausgegeben, wenn sich *bfs* im *-Modus befindet.

BEISPIEL

Wirkung eines Labels

Legen Sie eine Datei *tiere* an, mit folgenden fünf Zeilen (zur besseren Übersicht enthält die Datei Zeilennummern):

```
1 Hund
2 Katze
3 Maus
4 Pferd
5 Kuh
```

Zusätzlich wird eine *bfs*-Kommando-Datei *kmdo1* benötigt, aus der später Kommandos gelesen werden. Zuerst soll nach einer Zeile gesucht werden, in der das Wort *Maus* steht. Falls *Maus* nicht gefunden wird, sollen alle Zeilen ausgegeben werden, falls *Maus* gefunden wird, soll die Zeile ausgegeben werden, in der *Maus* steht. Dann soll *bfs* beendet werden. Dazu muß die Datei *kmdo1* folgenden Inhalt haben:

```
1,$xb/Maus/gefunden
1,$p
xb/^/fortsetzung
:gefunden
.p
:fortsetzung
q
```

Falls *Maus* nicht gefunden wird, muß die Anweisung, die aktuelle Zeile auszugeben, übersprungen werden, da sonst die letzte Zeile zweimal ausgegeben würde: vom Kommando *1,$p* (Zeile 2) und vom Kommando *.p* (Zeile 5). Deshalb findet mit *xb/^/fortsetzung* ein unbedingter Sprung über die Zeile 5 statt.

Nachdem beide Dateien erstellt sind, werden sie wie folgt verwendet:

```
$ bfs tiere
36
xfkmdo1
3 Maus
$
```

Als Alternative können Sie entweder in *kmdo1* oder in *tiere* *Maus* in *Laus* ändern. Dann werden alle Zeilen von *tiere* ausgegeben.

Wirkung von xbz und xbn

Zunächst soll eine Datei fünfmal auf die Zeichenkette *Maus* durchsucht werden. Dazu muß die *bfs*-Kommandodatei *kmdo2* folgenden Inhalt haben:

```
xv55
:1
/Maus/
xv5!expr %5 - 1
![ %5 != 0 ]
xbz 1
q
```

Der Ziffern-Variablen *5* wird der Wert *5* zugewiesen. Dann wird die erste Zeichenkette *Maus* gesucht und ausgegeben. Im nächsten Schritt wird der Wert in der Ziffern-Variable *5* um 1 erniedrigt. Dann wird geprüft, ob der Wert von *5* ungleich 0 ist. Falls ja, wird zum Label *1* gesprungen. Falls der Wert von *5* gleich 0 ist, wird *bfs* beendet.

Es wird wieder die Datei *tiere* durchsucht.

```
$ bfs tiere
36
xfkmdo2
3 Maus
3 Maus
3 Maus
3 Maus
3 Maus
$
```

Die Zeichenkette *Maus* wird fünf mal ausgegeben, da mit /.../ über das Dateiende hinaus und wieder am Dateianfang beginnend gesucht wird.

Falls *xbn* verwendet wird, muß, bei gleichem Ergebnis, *kmdo2* so aussehen:

```
xv55
:1
/Maus/
xv5!expr %5 - 1
![ %5 = 0 ]
xbn 1
q
```

SIEHE AUCH

csplit, *ed*, *umask*

cal
Kalender ausgeben (calendar)

cal gibt einen Kalender auf die Standard-Ausgabe aus.

cal[[_monat]_jahr]

Kein Argument angegeben
cal gibt den Kalender für den aktuellen Monat aus.

monat
cal gibt den Kalender für den angegebenen Monat aus.
Für *monat* können Sie Werte von 1 bis 12 angeben.
Wenn Sie einen Wert für *monat* angeben, dann müssen Sie auch einen Wert für *jahr* angeben.

monat nicht angegeben:
cal gibt den Kalender für das ganze Jahr aus.

jahr
cal gibt den Kalender für das angegebene Jahr aus.
Für *jahr* können Sie Werte von 1 bis 9999 angeben.

Beachten Sie: Wenn Sie z.B. *cal 88* eingeben, dann erhalten Sie den Kalender für das Jahr 88 und nicht für das Jahr 1988.

ENDE-STATUS

Der Ende-Status ist 0 bei Erfolg und ungleich 0, falls die angegebenen Werte für *jahr* bzw. *monat* außerhalb des zulässigen Bereichs liegen.

FEHLERMELDUNG

```
Bad argument
```

Die Werte, die Sie für *jahr* bzw. *monat* angegeben haben, liegen außerhalb des zulässigen Bereichs.

BEISPIEL

Ausgabe des Kalenders für Mai 1993:

```
$ cal 5 1993
    May 1993
 S  M Tu  W Th  F  S
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
```

calendar
Terminkalender

Mit *calendar* können Sie sich bequem an Ihre Termine erinnern lassen. Sie schreiben Ihre Termine hierfür in eine Datei mit dem Namen *calendar*. Wenn Sie dann das Kommando *calendar* aufrufen, werden alle Zeilen aus der Datei *calendar*, die an beliebiger Stelle das Datum des aktuellen oder des darauffolgenden Tages enthalten, auf die Standard-Ausgabe ausgegeben.

Sie müssen das Kommando *calendar* von dem Dateiverzeichnis aus aufrufen, in dem die Datei *calendar* steht. Wenn Sie nach jedem Login automatisch an Ihre Termine erinnert werden möchten, legen Sie die Datei *calendar* in Ihrem Home-Dateiverzeichnis an und schreiben in Ihre Datei *.profile* das Kommando *calendar*. Dann werden nach jedem Login Ihre am aktuellen oder darauffolgenden Tag anfallenden Termine auf die Standard-Ausgabe ausgegeben.
Einen regelmäßigen Aufruf von *calendar* erreichen Sie auch mit Hilfe der Kommandos *at* und *crontab*.

Zulässig sind Datumsangaben im Format *8/31*, *Aug.31* und *august31*, wobei die englischen Monatsnamen zu verwenden sind. Ist der aktuelle Tag ein Freitag, so gelten der folgende Samstag, Sonntag und Montag als auf den Freitag folgender Tag, also als *Morgen*.
Feiertage werden von *calendar* jedoch *nicht* berücksichtigt. Ist der aktuelle Tag der Tag vor einem Feiertag, so gilt nur der Feiertag als *Morgen*.

calendar[-]

-

Steht die Datei *calendar* in Ihrem Home-Dateiverzeichnis, so können Sie mit der Option - das Kommando *calendar* von irgendeinem Ihrer Dateiverzeichnisse aus aufrufen.
Ihre Termine werden dann nicht auf die Standard-Ausgabe ausgegeben, sondern über *mail* zugestellt.

Nur für den Systemverwalter

Der Systemverwalter bewirkt mit dem Aufruf von *calendar* mit der Option -, daß jedem Benutzer, der in seinem Home-Dateiverzeichnis eine Datei *calendar* hat, die Termine mit *mail* zugestellt werden. Diese Aufgabe übernimmt in der Regel das UNIX-System, das die Benutzer täglich über Termine informiert, die in der Datei *calendar* vermerkt sind.

DATEIEN

/usr/lib/calprog
Datei zur Bestimmung des aktuellen Datums sowie des Datums des folgenden Tages

/etc/passwd
Datei, die alle eingerichteten Benutzerkennungen enthält

*/tmp/cal**
Temporärdatei

BEISPIEL

Herr Aschenbach schreibt in seine Datei */home/aschenbach/calendar* die folgenden Termine:

```
10/09 Zahnarzt 13 Uhr
10/10 Herrn Blitz um 17 Uhr 30 vom Flughafen abholen
10/25 Geburtstag Onkel Erwin
```

In seine Datei */home/aschenbach/.profile* schreibt er den Kommandoaufruf:

```
calendar
```

Am 8. Oktober erscheint nach dem Login auf dem Bildschirm der Text:

```
10/09 Zahnarzt 13 Uhr
```

Am 9. Oktober erscheint nach dem Login auf dem Bildschirm der Text:

```
10/09 Zahnarzt 13 Uhr
10/10 Herrn Blitz um 17 Uhr 30 vom Flughafen abholen
```

Am 10.Oktober erscheint nach dem Login auf dem Bildschirm der Text:

```
10/10 Herrn Blitz um 17 Uhr 30 vom Flughafen abholen
```

Am 24. und 25. Oktober erscheint nach dem Login auf dem Bildschirm der Text:

```
10/25 Geburtstag Onkel Erwin
```

SIEHE AUCH

at, *date*, *crontab*, *mail*
cron, *environ* [7]

cancel
Druckaufträge löschen

Mit *cancel* können Sie Druckaufträge abbrechen oder löschen, die Sie mit den Kommandos *lp* oder *lpr* erteilt haben.

Mit *lpr -ca* können Sie ebenfalls Druckaufträge abbrechen oder löschen.

Mit *lpstat* können Sie sich Informationen über Druckaufträge ausgeben lassen.

cancel[_auftragsnr]...[_drucker]...

Sie müssen für mindestens einen der Operanden einen Wert angeben.

auftragsnr
: Für *auftragsnr* geben Sie die Auftragsnummer eines Druckauftrags an. Diese Nummer erhalten Sie auf die Standard-Ausgabe, nachdem Sie mit *lp* den Druckauftrag erteilt haben. Sie können mehrere Auftragsnummern angeben, jeweils getrennt durch ein Leerzeichen. Die Auftragsnummern aller anstehenden Druckaufträge können Sie mit dem Kommando *lpstat* abfragen. *cancel* gibt eine Meldung über gelöschte bzw. abgebrochene Druckaufträge auf die Standard-Ausgabe aus.

drucker
: Für *drucker* geben Sie den Namen eines Druckers oder einer Druckergruppe an. Sie können mehrere Namen angeben. Ein auf *drucker* laufender Druckauftrag wird abgebrochen, und der Drucker wird frei für den nächsten Auftrag. *cancel* gibt eine Meldung über abgebrochene Druckaufträge auf die Standard-Ausgabe aus.
Die Kommandos *lpstat* und *lpr -q* geben aus, auf welchen Druckern gerade Aufträge laufen.

FEHLERMELDUNGEN

```
cancel: printer "G005" was not busy
```

Sie haben beim *cancel*-Aufruf für *drucker* den Namen einer Druckergruppe angegeben, in diesem Fall G005, auf der kein Druckauftrag lief. Durch die Angabe des Druckergruppennamens können Sie nur einen Druckauftrag abbrechen, der gerade auf einem Drukker dieser Druckergruppe läuft. Sie können aber nicht Druckaufträge löschen, die zwar für diese Druckergruppe gestellt wurden, aber noch auf ihre Ausführung warten. Um das zu erreichen, müssen Sie die Auftragsnummer angeben.

```
cancel: request "G002-3" non existent
```

Sie haben beim *cancel*-Aufruf eine Auftragsnummer angegeben, in diesem Fall G002-3, für die es keinen Druckauftrag gibt. Einen Überblick über alle laufenden und anstehenden Druckaufträge erhalten Sie mit den Kommandos *lpstat* oder *lpr -q*.

```
cancel: "XY" is not a request id or a printer
```

Sie haben beim *cancel*-Aufruf ein Argument XY angegeben, das weder eine Auftragsnummer noch der Name einer Druckergruppe ist.

BEISPIELE

In Ihrem System gibt es zwei Druckergruppen, G001 und G002, zu denen jeweils ein Drucker gehört.

1. Der auf der Druckergruppe G001 laufende Auftrag soll abgebrochen werden:

   ```
   $ cancel G001
   request "G001-4" cancelled
   ```

   Der abgebrochene Druckauftrag hatte die Auftragsnummer G001-4.

2. Die Druckaufträge mit den Auftragsnummern G002-3 und G002-7 sollen gelöscht werden:

   ```
   $ cancel G002-3 G002-7
   request "G002-3" cancelled
   request "G002-7" cancelled
   ```

SIEHE AUCH

lp, *lpr*, *lpstat*

Handbuch SINIX-SPOOL V3.1 [15]

cat
Dateien aneinanderfügen und ausgeben (concatenate)

Das Kommando *cat* liest Dateien sequentiell und gibt diese auf der Standard-Ausgabe aus. Die Reihenfolge der Zeichen innerhalb der Dateien und deren Format bleiben dabei unverändert.
Werden beim Aufruf von *cat* mehrere Dateien angegeben, so werden diese in der gleichen Reihenfolge nacheinander ausgegeben.
Wenn Sie beim Aufruf keine Datei angeben, so liest *cat* von der Standard-Eingabe.

cat[␣**-s**][␣**-u**][␣**-v**[**et**]][␣datei]...

Keine Option angegeben
Die Ausgabe erfolgt gepuffert in Blöcken von *BUFSIZ* byte. Der Wert von *BUFSIZ* ist abhängig von der Maschine, auf der Sie arbeiten. Er wird in der Datei */usr/include/stdio.h* definiert und kann 512, 1024, 4096 oder 8192 byte betragen.
Wenn beim Aufruf angegebene Dateien nicht existieren, gibt *cat* eine entsprechende Meldung aus.

-s
Meldungen über nicht vorhandene Dateien werden unterdrückt.

-u
Byteweise Ausgabe ohne Zwischenspeicherung (ungepuffert).

-v
Nicht druckbare Zeichen (mit Ausnahme der Zeichen: Tabulator, Neue-Zeile, Seitenvorschub) werden auf die Standard-Ausgabe ausgegeben und zwar in folgender Form:

- Steuerzeichen:
  ^x (Control x)
  Welchen Wert x im Zeichensatz ISO 646 annimmt, können Sie der Tabelle in *Tabellen und Verzeichnisse, Zeichensatz ISO 646* entnehmen.
- DEL-Zeichen (oktal 0177):^?
- Nicht-ASCII-Zeichen (bei denen das achte Bit gesetzt ist): **M**-x, wobei x das durch die 7 übrigen Bits definierte Zeichen ist.

-e
Wirkt nur zusammen mit **-v**
Am Ende jeder Zeile wird vor dem Neue-Zeile-Zeichen das Dollarzeichen $ ausgegeben.

-t
Wirkt nur zusammen mit **-v**
Tabulatorzeichen werden ausgegeben in der Form: ^I und Seitenvorschübe in der Form: ^L

datei
Name der Datei, die ausgegeben werden soll. Sie können mehrere Dateien angeben.
Wenn Sie für *datei* einen Bindestrich - angeben, liest *cat* von der Standard-Eingabe.

datei nicht angegeben:
cat liest von der Standard-Eingabe.

Vorsicht

Wenn Sie die Ausgabe von *cat* auf eine Datei umlenken, von der gelesen wird, dann hat das den Verlust des originalen Dateiinhalts zur Folge. Der Inhalt von *datei1* geht im folgenden Beispiel verloren:

cat datei1 datei2 datei3 > datei1

FEHLERMELDUNGEN

`cat >aus_datei aus_datei: cannot create`

Sie haben kein Schreibrecht für die Ausgabedatei *aus_datei* oder für das Dateiverzeichnis, in dem *aus_datei* enthalten ist.

`cat ein_datei cat: cannot open ein_datei`

Sie haben kein Leserecht für die Eingabedatei *ein_datei*.

BEISPIELE

1. Aneinanderhängen und Umleiten der Ausgabe von zwei Dateien:

```
$ echo Montag Dienstag Mittwoch >datei1
$ echo Donnerstag Freitag Samstag >datei2
$ cat datei1 datei2 > datei3
$ cat datei3
Montag Dienstag Mittwoch
Donnerstag Freitag Samstag
```

2. Ausgabe des Inhalts von *datei1*

```
$ cat datei1
Jeder weiß, was so ein Mai-
käfer für ein Vogel sei.
```

In *datei2* schreiben Sie nun zwei Zeilen Text:

```
$ cat > datei2
In den Bäumen hin und her ↵
kriecht und fliegt und krabbelt er. ↵
END
```

Nun bringen Sie den Inhalt von *datei1* und *datei2*, ergänzt um zwei weitere Zeilen, die Sie von der Standard-Eingabe eingeben, in *datei3*. Anschließend lassen Sie den Inhalt von *datei3* ausgeben:

```
$ cat datei1 datei2 - > datei3
Max und Moritz immer munter ↵
schütteln sie vom Baum herunter. ↵
END
$ cat datei3
Jeder weiß, was so ein Mai-
käfer für ein Vogel sei.
In den Bäumen hin und her
kriecht und fliegt und krabbelt er.
Max und Moritz immer munter
schütteln sie vom Baum herunter.
```

SIEHE AUCH

cp, *pg*, *pr*

cd
Aktuelles Dateiverzeichnis wechseln (change working directory)

Das in die Bourne-Shell *sh* eingebaute Kommando *cd* macht das angegebene Dateiverzeichnis zu Ihrem aktuellen Dateiverzeichnis.

In einer eingeschränkten Shell wird das Kommando *cd* abgewiesen (siehe *sh*).

cd[_dateiverzeichnis]

dateiverzeichnis
Name des Dateiverzeichnisses, das Ihr aktuelles Dateiverzeichnis werden soll. Für dieses Dateiverzeichnis brauchen Sie Ausführrecht. Wenn Sie für *dateiverzeichnis* einen relativen oder absoluten Pfadnamen angeben, brauchen Sie Ausführrecht für alle Dateiverzeichnisse, aus denen sich dieser Pfadname zusammensetzt.

Das angegebene Dateiverzeichnis wird ohne Zugriff auf die Umgebungsvariable *CDPATH* (siehe *sh*) gesucht, falls der Name mit folgenden Zeichen beginnt:

/ bedeutet, daß die Suche im Dateiverzeichnis / (root) beginnt.
./ bedeutet, daß die Suche im aktuellen Dateiverzeichnis beginnt.
../ bedeutet, daß die Suche im übergeordneten Dateiverzeichnis beginnt.

Beginnt der Name des angegebenen Dateiverzeichnisses mit keinem dieser Zeichen, so wertet *cd* den Inhalt der Umgebungsvariablen *CDPATH* aus:

- Ist die Variable *CDPATH* nicht definiert oder leer, so wird das angegebene Dateiverzeichnis relativ zum aktuellen Dateiverzeichnis gesucht.
- Ist der Variablen *CDPATH* ein Wert zugewiesen, so wird das angegebene Dateiverzeichnis der Reihe nach in den Dateiverzeichnissen gesucht, deren Pfad in der Variablen *CDPATH* enthalten ist. Wenn *cd* das Dateiverzeichnis gefunden hat, schreibt es vor dem Wechsel den absoluten Pfadnamen dieses Dateiverzeichnisses auf die Standard-Ausgabe.

dateiverzeichnis nicht angegeben:
Das Kommando *cd* macht Ihr HOME-Dateiverzeichnis zum aktuellen Dateiverzeichnis. Das HOME-Dateiverzeichnis ist identisch mit dem Login-Dateiverzeichnis, falls Sie der Shell-Variablen HOME keinen anderen Pfadnamen zugewiesen haben.

FEHLERMELDUNGEN

datei: does not exist

Das angegebene Dateiverzeichnis existiert nicht. Das können Sie mit *ls -l* prüfen.

datei: not a directory

Die angegebene Datei ist kein Dateiverzeichnis. Das können Sie mit *ls -l* prüfen.

datei: permission denied

Sie haben für das angegebene Dateiverzeichnis kein Ausführrecht.
Wenn Sie für *dateiverzeichnis* einen relativen oder absoluten Pfadnamen angegeben haben, haben Sie kein Ausführrecht für eines der Dateiverzeichnisse, aus denen sich dieser Pfadname zusammensetzt.

cd: restricted

Die aktuelle Shell ist eingeschränkt, deshalb wird *cd* abgewiesen.

UMGEBUNGSVARIABLEN

HOME
: enthält den absoluten Pfadnamen Ihres HOME-Dateiverzeichnisses.

CDPATH
: Sie können dieser Variablen die absoluten Pfadnamen der Dateiverzeichnisse zuweisen, die *cd* durchsuchen soll.
Standardmäßig ist diese Variable nicht definiert.

BEISPIELE

1. Mit der folgenden Eingabe wird das Unterdateiverzeichnis *termine* zum aktuellen Dateiverzeichnis:

```
$ cd termine
$ pwd

/home/rosa/termine
```

2. Die Benutzerin *rosa* hat die Umgebungsvariable *CDPATH* neu definiert. Sie will in ihr Unterdateiverzeichnis *usr* wechseln, landet aber mit den folgenden Angaben im Dateiverzeichnis */usr*:

```
$ echo $CDPATH
/:/home/rosa/termine:.
$ pwd
/home/rosa
$ ls -l
drwx--x--x 2 rosa        144 Feb 28 12:32 usr
drwx--x--x 2 rosa        192 Feb 28 11:51 termine
-rw------- 1 rosa      11734 Mar  7 16:22 probe
.
.
.
$ cd usr
$ pwd

/usr
```

   Das Dateiverzeichnis *usr* wird zuerst in den Dateiverzeichnissen gesucht, deren Pfadnamen der Variablen *CDPATH* zugewiesen sind. Hier enthält *CDPATH* als ersten Pfadnamen / für das Root-Dateiverzeichnis. Das aktuelle Dateiverzeichnis durchsucht *cd* als letztes.

   Mit der folgenden Eingabe kann die Benutzerin *rosa* verhindern, daß *cd* die Umgebungsvariable *CDPATH* auswertet:

```
$ cd ./usr
$ pwd

/home/rosa/usr
```

SIEHE AUCH

pwd, *sh*
chdir() [19]

ced
Bildschirmorientierter Editor

Im Gegensatz zu einem zeilenorientieren Editor können Sie bei einem bildschirmorientierten Editor die Schreibmarke mit den Pfeiltasten ↑, ↓, →, ←, ↖ und ↵ frei über den Bildschirm bewegen, um Text an beliebiger Stelle einzugeben oder zu ändern. Der *ced* ist menügeführt und bietet verschiedene Bearbeitungsmodi. Je nach Modus können Sie:

- Text eingeben
- Kommandos eingeben für unterschiedliche Bearbeitungsfunktionen, z.B. Kopieren, Löschen usw.

Aufbau dieser Beschreibung

Die Beschreibung des *ced* gliedert sich in vier Teile:

- Kommandoübersicht
- Arbeiten mit dem ced
  - Bildschirm
  - Fenster
  - Bedienbereich
  - Schreibmarke bewegen
  - Modusauswahl
- Beschreibung der Modi
- Fehlermeldungen

Hier ist die Standard-Version von *ced* beschrieben, in der die Menüfunktionen und Meldungstexte in englischer Sprache ausgegeben werden (siehe *INTERNATIONALE UMGEBUNG*).

ced[␣option][␣datei]

option

-f␣zeilennummer

Der *ced* positioniert die Schreibmarke auf die mit *zeilennummer* angegebene Zeile.

-s␣suchzeichenkette

Der *ced* positioniert die Schreibmarke auf die erste Zeile, die *suchzeichenkette* enthält. Ist *suchzeichenkette* nicht in der Datei vorhanden, zeigt der *ced* den Dateianfang und positioniert die Schreibmarke auf die erste Zeile.

datei

Name der Datei, die Sie bearbeiten wollen. Gibt es im aktuellen Dateiverzeichnis keine Datei dieses Namens, legt *ced* diese neu an.

datei nicht angegeben:
ced öffnet die Datei, die Sie zuletzt an Ihrer Datensichtstation mit dem *ced* bearbeitet haben und positioniert die Schreibmarke dahin, wo sie zuletzt stand. *ced* holt die hierfür erforderlichen Informationen aus der Datei */var/ced/CEttyname.benutzerkennung*.

KOMMANDOÜBERSICHT

Im folgenden finden Sie eine tabellarische Übersicht der wichtigsten Funktionstasten, Menüpunkte und Kommandos des *ced* nach Funktionsgruppen geordnet. Eine ausführliche Beschreibung finden Sie im Abschnitt *ARBEITEN MIT DEM CED*.

ced laden	
ced ↵	*ced* mit der zuletzt editierten Datei aufrufen
ced *datei* ↵	*ced* mit der zu editierenden *datei* laden

ced beenden	
MENU **qy**	*ced* verlassen, Änderungen speichern in aktuelle Datei
MENU **qn**	*ced* verlassen, Änderungen nicht speichern
CTRL @	*ced* verlassen (Notbremse), Änderungen nicht speichern

Allgemeine Bearbeitung	
MENU **e**	neuen Text eingeben
MENU **i**	neuen Text in alten einfügen

MENU **a**	Abspeichern von Text
F9	Sichern in aktuelle Datei
MENU **a** ↵	Sichern in aktuelle Datei Bedienbereich: *datei*
MENU **a** ↵	Sichern in angegebene *datei*
MENU **qy**	*ced* verlassen und sichern in aktuelle Datei

MENU **d**	Datei wechseln
MENU **d** ↵	wechseln in die letzte Datei Bedienbereich: *datei*
MENU **d** ↵	wechseln in die angegebene Datei
F10	Datei wechseln

MENU **c**	Kommandos ausführen Bedienbereich: [bereich] kommandobereich *n* Zeilen ab Schreibmarke *n* Abschnitte an Schreibmarke - kein Eingabebereich, nichts löschen Standard: Schreibmarke bis Abschnittende
MENU **!**	Shell aufrufen
MENU **p**	Tasten programmieren
MENU **x**	Tastenprogrammierung anzeigen
MENU **t**	Textmarkierung

Spezielle Textbearbeitung

MENU **l**	eine Zeile bearbeiten
MENU **lb**	Einfügen einer Leerzeile
MENU **ld**	Löschen und Speichern einer Zeile
MENU **li**	Zurückholen des zwischengespeicherten Blocks
MENU **lj**	Verbinden zweier Zeilen
MENU **lr**	Rest der Zeile löschen
MENU **ls**	Teilen einer Zeile
MENU **b**	Zeilenbereich bearbeiten
MENU **bb**	Einfügen von Leerzeilen
MENU **bc**	Kopieren des markierten Blocks
MENU **bd**	Löschen und speichern des markierten Blocks
MENU **bi**	Zurückholen des zwischengespeicherten Blocks
MENU **bm**	Markieren einer Zeile
MENU **r**	Rechteck bearbeiten
MENU **rd**	Löschen eines Rechtecks
MENU **rh**	Horizontal einfügen
MENU **rm**	Markieren einer Ecke
MENU **ro**	Überschreibend kopieren
MENU **rv**	Vertikal einfügen

Positionieren

START	Dateianfang
↖ ↖ ↑	erste Zeile der Datei
↖ ↖ ↓	letzte Zeile der Datei
⇧	nach oben um 1/2 Fenster
⇩	nach unten um 1/2 Fenster
F15	nach unten um 1 Fenster
F16	nach oben um 1 Fenster

MENU **w**	Fenster positionieren (gezielt)
	auf eine bestimmte Zeile oder Spalte Bedienbereich: *n* (Zahl)
↵ ↑ ↓	Zeile
← →	Spalte
	relativ zur aktuellen Position Bedienbereich: +*n* oder -*n*
↵ ↓	*n* Zeilen nach unten
↑	*n* Zeilen nach oben
↓	*n* Zeilen nach unten
←	*n* Spalten nach links
→	*n* Spalten nach rechts

MENU **s**	Suchen nach Zeichenketten
	Bedienbereich: *suchzeichenkette*
↓, ↵ oder F13	vorwärts
↑ oder F14	rückwärts

Schreibmarke positionieren

↑	Eine Zeile nach oben
↖ ↑	erste Zeile des Fensters
↖ ↖ ↑	erste Zeile der Datei

↓	Eine Zeile nach unten
↖ ↓	letzte Zeile des Fensters
↖ ↖ ↓	letzte Zeile der Datei

←	Ein Zeichen nach links
⇤	Tabulatorposition nach links (Tabulatorpositionen sind die Spalten 1, 9, 17 usw.)
↖ ←	linker Rand des Fensters
↖ ↖ ←	Zeilenanfang
↖ ⇤	Zeilenanfang

→	Ein Zeichen nach rechts
⇥	Tabulatorposition nach rechts (Tabulatorpositionen sind die Spalten 1, 9, 17 usw.)
↖ →	rechter Rand des Fensters
↖ ↖ →	Zeilenende
↖ ⇥	Zeilenende

Tasten

CHAR	Zeichen ein- oder ausfügen
LINE	Zeilen ein- oder ausfügen
⌫	ein Zeichen links der Schreibmarke löschen
CTRL **r**	Bildschirm neu ausgeben
CTRL @	*ced* verlassen (Notbremse)
	Bedienbereich
⌫	ein Zeichen links der Schreibmarke löschen
CTRL **x**	ganze Eingabe löschen

Standard-Funktionen

START	Dateianfang
F9	Datei sichern
F10	Datei wechseln
F11	Zeile markieren (2 Marken)
F12	Zeile kopieren
F13	suchen vorwärts
F14	suchen rückwärts
F15	eine Seite vor
F16	eine Seite zurück
F17	zurückholen des gelöschten Textteils

ARBEITEN MIT DEM CED

ced arbeitet mit einer Kopie der Datei, die Sie bearbeiten wollen. *ced* übernimmt die Änderungen erst dann in die Datei selbst, wenn Sie

- den *ced* verlassen und dabei *y* (yes) angeben (siehe Modus *q*)
- die Änderungen mit Modus *a* abspeichern. Dabei können Sie auch einen anderen Dateinamen wählen.

Bildschirm

```
** CED Text Editor V1.3        Line:   1  Column:   1  Name: datei
```

```
You wish to enter text?
Please enter your text
```

Die Kopfzeile enthält:
- die Bezeichnung *ced*-Version
- die aktuelle Zeilen- (Line) und Spaltenposition (Column) der Schreibmarke
- den Dateinamen der aktuell im Fenster gezeigten Datei

Fenster

Das Fenster zeigt einen Ausschnitt von 19 Zeilen aus der bearbeiteten Datei. Leerzeilen nach dem Ende einer Datei sind durch eine Tilde ~ am Zeilenanfang gekennzeichnet. An der Schreibmarkenposition in der Kopfzeile erkennen Sie, welchen Abschnitt der Datei Sie gerade bearbeiten.

Das Fenster können Sie wie folgt verschieben:

⇧ nach oben um 9 Zeilen (1/2 Fenster),

⇩ nach unten um 9 Zeilen (1/2 Fenster),

F15 nach unten um 19 Zeilen (1 Fenster),

F16 nach oben um 19 Zeilen (1 Fenster).

Mit Modus *w* (position the window) können Sie das Fenster beliebig nach oben, unten, rechts und links verschieben.

Das Fenster verschiebt sich automatisch, wenn Sie mit der Schreibmarke an den Rand des gezeigten Bereiches kommen.

Geht eine Zeile über den rechten Rand hinaus, so wird dies durch ein Fortsetzungszeichen > in der letzten Spalte angezeigt.

Bedienbereich

Im Bedienbereich zeigt der *ced* an, welchen Modus Sie gerade gewählt haben und welche Funktionen Ihnen in diesem Modus zur Verfügung stehen.

Nach dem Aufruf befindet sich der *ced* immer im Modus *e* (enter new text) für neuen Text eingeben.

In einigen Modi können Sie im Bedienbereich Text eingeben, z.B. den Dateinamen im Modus *d* (switch documents) oder die zu suchende Zeichenkette im Modus *s* (search for something).

Sie korrigieren eine Eingabe im Bedienbereich wie folgt:

⌫ einzelne Zeichen löschen

CTRL X die ganze Eingabe löschen

Schreibmarke bewegen

↑ nach oben

↓ nach unten

→ nach rechts

← nach links

⇥ Tabulator nach rechts, Tabulatorpositionen sind die Spalten 1, 9, 17, 25 usw.

⇤ Tabulator nach links

↖ Mit dieser Taste positionieren Sie auf Bereichsgrenzen

↖ → rechter Rand des Fensters

↖ ← linker Rand des Fensters

↖ ↑ erste Zeile des Fensters

↖ ↓ letzte Zeile des Fensters (oder Dateiende, wenn im Fenster vorhanden)

↖ ⇥ erste Spalte des um 80 Spalten nach rechts verschobenen Bildschirms

↖ ⇤ erste Spalte des aktuellen Fensters

↖ ↖ → letzte Spalte (512)

↖ ↖ ← erste Spalte (1)

↖ ↖ ↑ erste Zeile der Datei

↖ ↖ ↓ letzte Zeile der Datei

Modus auswählen

Drücken Sie nacheinander die Taste MENU und eine Buchstabentaste. Sie wählen damit eine der folgenden Funktionen:

Taste	Funktion
MENU **a**	Abspeichern von Text
MENU **b**	Zeilenbereich bearbeiten
MENU **c**	Kommando ausführen
MENU **d**	Datei wechseln
MENU **e**	Neuen Text eingeben
MENU **i**	Text einfügen
MENU **l**	Zeile bearbeiten
MENU **p**	Tasten programmieren
MENU **q**	*ced* verlassen
MENU **r**	Rechteck bearbeiten
MENU **s**	Suchen nach Zeichenketten
MENU **t**	Textmarkierung suchen
MENU **w**	Fenster positionieren
MENU **x**	Tastenbelegung anzeigen
MENU **!**	Shell aufrufen
MENU HELP	zeigt die Kurzbeschreibung des *ced* in englischer Sprache.

In der Kurzbeschreibung können Sie vorwärts und rückwärts blättern, wie in einer anderen Datei. Wenn Sie vorher im Modus *s* eine Zeichenkette angeben, können Sie mit F13 bzw. F14 diese Zeichenkette in der Kurzbeschreibung suchen.

Der *ced* verarbeitet Eingaben sofort, ohne daß Sie die Taste ↵ drücken müssen. Ausnahmen sind Eingaben im Bedienbereich.

Abhängig vom Modus haben die Tasten unterschiedliche Wirkung.

Modus rückgängig machen

Wenn Sie sich geirrt haben und den ausgewählten Modus wieder rückgängig machen wollen, drücken Sie nochmal die Taste (MENU). Sie erhalten dann wieder das Ausgangsmenü, aus dem Sie erneut den gewünschten Modus auswählen können.

BESCHREIBUNG DER MODI

(MENU) a - Abspeichern von Text

Im Bedienbereich erscheint der Text:

```
You wish to save your text?
Please enter a file name or press <RETURN>
```

Im Bedienbereich geben Sie den Namen der Datei an, in der der Text abzuspeichern ist, den Sie gerade bearbeiten. Dann drücken Sie die Taste (↵). Geben Sie keinen Dateinamen an, schreibt der *ced* den Text in die aktuelle Datei. Der Dateiname steht in der Kopfzeile. Der ursprüngliche Dateiinhalt wird überschrieben.

Wenn Sie (END) eingeben, erscheint im Bedienbereich der Text:

```
You wish to quit CED?
Save text in document datei ?:  Enter y=yes or n=no
```

Sie können jetzt weiterarbeiten wie bei Modus *q* beschrieben.

(MENU) b - Zeilenbereich bearbeiten (b - line block)

Im Bedienbereich erscheint der Text:

```
You wish to edit a line block?
Please select one of the following functions: (b,c,d,i,m, <MENU>)
```

In diesem Modus können Sie Bereiche von aufeinanderfolgenden Zeilen bearbeiten. Markieren Sie die oberste und unterste Zeile des Bereichs (Funktion *m*).

Folgende Funktionen stehen Ihnen zur Verfügung:

m

(m - mark) Markieren einer Zeile:
Setzen Sie die Schreibmarke auf die gewünschte Zeile und drücken Sie die Taste *m*. *ced* meldet im Bedienbereich: MARK SET (Marke gesetzt). Es gelten immer die beiden zuletzt gesetzten Marken. Löschen eines Bereichs (Funktion *d*) löscht auch die Marken.

b

(b - blank lines) Einfügen von Leerzeilen:
Oberhalb der Schreibmarke werden soviele Leerzeilen eingefügt, wie der zuletzt markierte Bereich umfaßt.

c

(c - copy) Kopieren des markierten Bereichs:
Oberhalb der Schreibmarke wird eine Kopie des markierten Bereichs eingefügt.

d

(d - delete and store) Löschen und Speichern des markierten Bereichs:
Der markierte Bereich wird aus der Datei gelöscht und zwischengespeichert. Jedes weitere Löschen und Speichern überschreibt den zuletzt gespeicherten Bereich!

i

(i - insert) Zurückholen des zwischengespeicherten Bereichs:
Der zuletzt gelöschte und damit zwischengespeicherte Bereich wird oberhalb der Schreibmarke eingefügt.

Wollen Sie nur eine einzige Zeile bearbeiten, können Sie diese durch zweimaliges Drücken der Taste *m* markieren.

Wenn Sie (END) eingeben, erscheint im Bedienbereich der Text:

```
You wish to quit CED?
Save text in document datei ?:  Enter y=yes or n=no
```

Sie können jetzt weiterarbeiten wie bei Modus *q* beschrieben.

(MENU) c - Kommando ausführen (c - command)

Im Bedienbereich erscheint der Text:

```
You wish to execute a command? —>
Please enter a command
```

In diesem Modus können Sie SINIX-Kommandos aufrufen und diesen einen Bereich Ihrer Datei als Eingabe übergeben. Die Ausgabe des ausgeführten Kommandos schreibt *ced* an die Stelle des Eingabebereichs in die Datei. In der Eingabezeile des Menübereichs legen Sie den Eingabebereich und das auszuführende Kommando fest:

[bereich]_kommando

bereich
Eingabebereich für das Kommando. Dieser Bereich der Datei wird an die Standard-Eingabe des Kommandos übergeben und in der Datei gelöscht.

bereich kann sein:
nl, n, -

nl
: *n* Zeilen ab der Schreibmarke.

n
: *n* Abschnitte ab dem Abschnitt, in dem die Schreibmarke steht. Ein Abschnitt ist ein Bereich ohne Leerzeilen und endet mit einer Leerzeile.

-
: leerer Eingabebereich, aus der Datei wird nichts gelöscht. Es werden auch keine Daten an die Eingabe des SINIX-Kommandos übergeben.

bereich nicht angegeben:
Eingabebereich ist der Bereich ab der Schreibmarke bis zum Ende des Abschnitts.

kommando
: Kommando, das ausgeführt werden soll. Eine fehlerhafte Angabe meldet *ced* und löscht den Eingabebereich nicht. Zugelassen sind alle SINIX-Kommandos, außer z.B. *login, su* und solche Kommandos, die die Shell-Umgebung verändern.

  Die Ausgabe des Kommandos wird an Stelle der gelöschten Eingabe eingefügt. Wollen Sie keine Eingabe löschen, müssen Sie einen leeren Eingabebereich angeben (0l oder 0 oder kürzer -).

  SINIX-Kommandos, die keine Dateieingabe erwarten (wie z.B. cat *datei*), sollten Sie keine Datei übergeben (leeren Eingabebereich angeben!), da es sonst zu Störungen kommen kann. Ein entsprechender Bereich ist dann gesondert zu löschen.

  Ist SHELL nicht gesetzt, so können Sie kein Kommando ausführen.

MENU d - Datei wechseln (d - switch documents)

Im Bedienbereich erscheint der Text:

```
You wish to switch documents? —>
Please enter a file name or press <RETURN>
```

Im Bedienbereich geben Sie den Namen einer weiteren Datei (document) an, die Sie bearbeiten möchten. Drücken Sie nun die Taste ↵. Im Fenster sehen Sie den Anfang der neuen Datei. Der *ced* ist jetzt im Modus e (neuen Text eingeben).

Existiert die angegebene Datei nicht, fragt *ced*, ob die Datei angelegt werden soll. Antworten Sie mit y, so wird diese Datei angelegt.

Mehrere Dateien parallel bearbeiten

Haben Sie mehrere Dateien geöffnet, wie oben beschrieben, dann können Sie:

- von einer Datei auf die nächste umschalten:
  Drücken Sie im Modus *d* nur die Taste ↵. *ced* schaltet zwischen den beiden zuletzt bearbeiteten Dateien um.
- Text von einer Datei in eine andere übertragen:
  1) Auf Bereiche, die Sie markiert haben (Modus *b*, *r* oder *l*) können Sie direkt von der zweiten Datei aus zugreifen, wenn Sie in den entsprechenden Modus geschaltet haben.
  2) Zwischengespeicherte Bereiche können Sie in der anderen Datei zurückholen (Modus *b* oder *l*).

Schalten Sie in den Modus *q* (*ced* verlassen), fragt *ced* für die aktuelle Datei, ob sie gesichert werden soll. Anschließend schaltet *ced* auf die nächste offene Datei zurück, in der Sie Änderungen vorgenommen haben. Auf diese Weise können Sie alle während einer *ced*-Sitzung bearbeiteten Dateien abschließen.

MENU e - Neuen Text eingeben (e - enter new text)

Im Bedienbereich erscheint der Text:

```
You wish to enter text?
Please enter your text
```

In diesem Modus können Sie Text eingeben. Sie überschreiben dabei schon vorhandenen Text.
Folgende Tasten (jeweils im Tastenfeld INSERT bzw. DELETE) können Sie dabei verwenden:

CHAR um einzelne Buchstaben einzufügen oder zu löschen.

LINE um Zeilen einzufügen oder zu löschen.

Wenn Sie END eingeben, erscheint im Bedienbereich der Text:

```
You wish to quit CED?
Save text in document datei ?:  Enter y=yes or n=no
```

Sie können jetzt weiterarbeiten wie bei Modus *q* beschrieben.

(MENU) i - Text einfügen (i - insert)

Im Bedienbereich erscheint der Text:

```
You wish to insert something in the text?
Please insert your text
```

In diesem Modus können Sie Text einfügen. Der Text, den Sie eingeben, wird an der aktuellen Schreibmarkenposition eingefügt, der schon vorhandene Text wird ab der Schreibmarkenposition nach rechts verschoben. Wenn nötig, wird der Text auch über den rechten Fensterrand hinaus geschoben. Die maximale Länge einer Zeile beträgt 512 Zeichen. Der *ced* meldet, wenn durch das Einfügen die Zeile zu lang wird.

Wenn Sie (END) eingeben, erscheint im Bedienbereich der Text:

```
You wish to quit CED?
Save text in document datei ?:  Enter y=yes or n=no
```

Sie können jetzt weiterarbeiten wie bei Modus *q* beschrieben.

(MENU) l - Zeile bearbeiten (l - line)

Im Bedienbereich erscheint der Text:

```
You wish to edit a line?
Please select one of the following functions: (b,d,i,j,r,s, <MENU>)
```

Welche Zeile Sie bearbeiten wollen, wählen Sie mit der Schreibmarke.

b

(b - blank line) Einfügen einer Leerzeile:
Eine Leerzeile wird oberhalb der Schreibmarke eingefügt.

d

(d - delete and store) Löschen und Speichern der Zeile:
Die gewählte Zeile wird gelöscht und zwischengespeichert. Jedes weitere Löschen und Speichern überschreibt die zuletzt gespeicherte Zeile.

i

(i - insert) Zurückholen der zwischengespeicherten Zeile:
Die zuletzt zwischengespeicherte Zeile wird oberhalb der Schreibmarke eingefügt.

j

(j - join) Verbinden zweier Zeilen:
Die gewählte Zeile und die darauf folgende Zeile werden zu einer Zeile verbunden.

r

(r - remove) Rest der Zeile löschen:
Der Rest der Zeile ab der Schreibmarkenposition wird gelöscht.

s

(s - split) Teilen der Zeile:
Der Rest der Zeile ab der Schreibmarkenposition wird in eine neue Zeile geschrieben.

Wenn Sie (END) eingeben, erscheint im Bedienbereich der Text:

```
You wish to quit CED?
Save text in document namebrief ?:  Enter y=yes or n=no
```

Sie können jetzt weiterarbeiten wie bei Modus *q* beschrieben.

(MENU) **p - Tasten programmieren** (p - key programming)

Im Bedienbereich erscheint der Text des zuletzt benutzten Menüs.

In diesem Modus können Sie beliebige Zeichenketten, Textbausteine, Schreibmarkenbewegungen sowie Modus-Umschaltungen bestimmten Tasten zuweisen.
Fast alle Tasten können Sie auf diese Weise programmieren. Bevorzugt sollten Sie jedoch dafür die F-Tasten verwenden. Wenn Sie dann eine so definierte Taste (key) drücken, erhält der *ced* die entsprechenden Zeichenketten bzw. Anweisungen.

Makroprogrammierung

Wenn Sie in den Modus *Tasten programmieren* umschalten, erscheint in der Kopfzeile der Text PROTO. Alles, was Sie von nun an ausführen, wird gespeichert: Text, den Sie eingeben, Modus-Umschaltungen usw. Um diese Aktionen einer Taste zuzuordnen, drücken Sie erneut die Tasten (MENU) und (P). Der *ced* meldet:

```
****PRESS KEY TO BE CHANGED
```

Drücken Sie die Taste, der Sie die gespeicherten Aktionen zuordnen wollen. Anschließend schaltet *ced* wieder in den Modus *e* (neuen Text eingeben).

Einschränkungen bei der Tastenbelegung

Sie können alle Tasten belegen, außer MENU und p.
Verwenden Sie vor allem die Tasten F1, F2 usw.
Beachten Sie also:
Die Tasten F9 bis F17 sind vom *ced* mit Standard-Funktionen vorbelegt (siehe unten, *Tasten mit besonderer Bedeutung*), sie sind aber auch mit neuen Zeichenketten belegbar. Neu belegte Tasten verlieren ihre ursprüngliche Funktion, bis Sie sie wieder mit der ursprünglichen Funktion belegen. Die Belegung bleibt auch nach Beendigung der *ced*-Sitzung erhalten.

Beim Verlassen des *ced* werden Ihre Tastenbelegungen in der Datei */var/ced/CEkey.benutzerkennung* automatisch für Ihre Benutzerkennung gespeichert. Wenn Sie den *ced* wieder aufrufen, gelten wieder diejenigen Tastenbelegungen, welche Sie in der letzten *ced*-Sitzung festgelegt hatten.

Löschen der Tastenbelegungen

Im Modus *p* haben alle Tasten ihre ursprüngliche Funktionsbelegung. Möchten Sie die Makro-Belegung einer Taste rückgängig machen, gehen Sie wie folgt vor: Sie geben MENU p (Modus *Tasten programmieren*) ein. Dann tippen Sie die Taste ein, deren Belegung Sie rückgängig machen wollen. Nun ist die ursprüngliche Funktion der Taste für die nächste Belegung gespeichert. Dann geben Sie nochmal MENU p und als zu belegende Taste die Taste ein, deren Belegung Sie rückgängig machen wollen.

MENU **q - ced verlassen** (q - quit CED)

Im Bedienbereich erscheint der Text:

```
You wish to quit CED?
Save text in document datei ?:  Enter y=yes or n=no
```

Im Modus *q* können Sie wählen, ob Sie Ihre Änderungen in der Datei sichern wollen oder nicht. Geben Sie *y* oder *n* ein:

y
(y - yes) Sie verlassen den Editor und die geänderte Datei wird gesichert, d.h., alle Änderungen stehen jetzt in Ihrer Datei.

n
(n - no) Sie verlassen den Editor und die Änderungen werden nicht gespeichert, d.h., Ihre Datei hat den ursprünglichen Inhalt. Leere Dateien werden am Ende der Sitzung gelöscht, sofern sie nicht vor Beginn der Sitzung schon leer waren.

Wenn Sie nach der Eingabe von MENU *q* die *ced*-Sitzung doch nicht beenden, sondern weiterarbeiten wollen, drücken Sie einfach die Taste MENU. Sie erhalten dann wieder das Auswahlmenü und Sie können normal weiterarbeiten.

MENU r - Rechteck bearbeiten (r - rectangle)

Im Bedienbereich erscheint der Text:

```
You wish to edit a rectangle?
Please select one of the following functions: (d,h,m,o,v, <MENU>)
```

Ein Rechteck ist ein Bereich der Datei, den Sie durch Markieren zweier Ecken festlegen (Funktion *m*). Die Ecken liegen sich diagonal gegenüber.

m

(m - mark) Markieren einer Ecke:
Markiert wird die Position, auf der die Schreibmarke steht, wenn Sie Taste *m* drücken. *ced* meldet: MARK SET (Marke gesetzt). Es gelten immer die beiden zuletzt gesetzten Marken. Löschen eines Rechtecks (Funktion *d*) löscht auch die Marken.

d

(d - delete) Löschen eines Rechtecks:
Das markierte Rechteck wird gelöscht. Es wird *nicht* gespeichert! Wenn Sie ein Rechteck verschieben wollen, kopieren Sie es erst (Funktion *h, o* oder *v*) und löschen es anschließend.

h

(h - horizontal) horizontal einfügen:
Das markierte Rechteck wird mit seiner linken oberen Ecke an der Schreibmarkenposition eingefügt. Text in den betroffenen Zeilen ab der Schreibmarkenposition wird nach rechts verschoben.

o

(o - overwrite) überschreibend kopieren:
Das markierte Rechteck wird mit der linken oberen Ecke auf die Schreibmarkenposition kopiert. Vorhandener Text wird dabei überschrieben.

v

(v - vertical) vertikal einfügen:
Das markierte Rechteck wird vor der Schreibmarkenzeile eingefügt. Es steht mit dem linken Rand auf der Spalte, in der die Schreibmarke steht.

Wenn Sie END eingeben, erscheint im Bedienbereich der Text:

```
You wish to quit CED?
Save text in document datei ?:  Enter y=yes or n=no
```

Sie können jetzt weiterarbeiten wie bei Modus *q* beschrieben.

MENU s - Suchen nach Zeichenketten (s - search)

Im Bedienbereich erscheint der Text:

```
You want to search for something? —>
Please enter the search string
```

Im Bedienbereich tragen Sie die Zeichenkette ein, nach der Sie suchen wollen. Den Suchvorgang starten Sie mit einer der folgenden Tasten:

↓ oder ↵	suchen in Richtung Dateiende
↑	suchen in Richtung Dateianfang

Tritt die Zeichenkette bis zum Dateiende bzw. Dateianfang nicht mehr auf, meldet *ced*: `TEXT NOT FOUND` (Text nicht gefunden).

Führende und angehängte Leerzeichen sind signifikant. Im Gegensatz zu anderen Editoren kennt der *ced* keine regulären Ausdrücke.
Der Suchvorgang kann mit der Taste DEL abgebrochen werden.

MENU t - Textmarkierung (t - tags)

Wenn Sie mehrere C-, Pascal-, FORTRAN-, LEX- oder YACC-Quelldateien haben, die zu einem größeren Programmsystem gehören, können Sie mit dem Kommando *ctags* eine Markierungsdatei erzeugen, welche die Namen und genauen Fundorte (Datei und Position) aller zu diesem Programmsystem gehörenden Funktionen enthält.

Im Modus "Textmarkierung" geben Sie in der Bedienzeile den Namen der Funktion an, zu der Sie umschalten möchten. *ced* schaltet in die Datei an die Stelle um, an der sich die Definition befindet. Die Markierungsdatei muß im aktuellen Dateiverzeichnis vorhanden sein und den Namen der angegebenen Funktion enthalten.

MENU w - Fenster positionieren (w - window)

Im Bedienbereich erscheint der Text:

```
You wish to position the window? —>
Please enter line, column or relative position (with prefix)
```

Das Fenster können Sie in diesem Modus auf zwei Arten verschieben:

- auf eine bestimmte Zeile (line) oder Spalte (column):

  Geben Sie eine Zahl ohne Vorzeichen (prefix) im Bedienbereich an. Diese Zahl darf nicht größer sein als die Gesamtanzahl der Zeilen der Datei. Drücken Sie dann eine der folgenden Tasten:

  ↵ oder ⇑ oder ⇓
  Die Schreibmarke wird auf die angegebene Zeile positioniert, egal ob diese Zeile vor oder hinter der aktuellen Zeile liegt.

  ← oder →
  Das Fenster beginnt mit der angegebenen Spalte. Die Schreibmarke steht in der Zeile, in der sie vorher war.

- relativ (relative position) zur aktuellen Position:

  Geben Sie eine Dezimalzahl n mit Vorzeichen + oder - ein und drücken Sie eine der folgenden Tasten:

  ↵ oder
  ⇓ n Zeilen nach unten
  ⇑ n Zeilen nach oben
  ← n Spalten nach links
  → n Spalten nach rechts

  Ob Sie +n oder -n angeben, ist völlig gleichgültig. Die Verschieberichtung richtet sich nur nach den Pfeiltasten.

MENU x - Tastenbelegung anzeigen

Im Bedienbereich erscheint der Text:

```
TO CONTINUE PRESS <MENU>
```

Im Modus *x* können Sie die von Ihnen programmierten Tasten und die diesen zugeordneten Texte ansehen. Auf dem Bildschirm wird Ihnen ohne *ced*-Fenster eine vollständige Liste gezeigt. Auf der linken Seite des ersten Doppelpunktes ist die Taste verzeichnet, auf der rechten Seite die dieser Taste zugeordnete Zeichenkette. Dabei bedeuten:

-r	→	Schreibmarke nach rechts
l-	←	Schreibmarke nach links
UP	↑	Schreibmarke nach oben
DOWN	↓	Schreibmarke nach unten
HOME	↖	Schreibmarke nach links oben
MENU	MENU	gedrückt
RETURN	↵	gedrückt

Die von *ced* vorbelegten Tasten F9 bis F17 werden nicht angezeigt. Zurück in die Menüauswahl gelangen Sie mit MENU.

MENU ! - Shell aufrufen

Unter dem Bedienbereich erscheint der Text:

```
>>>    Exit to command shell, return to CED with the END key.
```

In diesem Modus verlassen (exit) Sie den *ced* vorübergehend. Der *ced* ruft das Programm auf, das Sie in der Shellvariablen *SHELL* angegeben haben. Diese Variable können Sie z.B. in Ihrer Datei *.profile* setzen. Ist *SHELL* nicht gesetzt, so können Sie auch keine Shell aufrufen. an. Sie können so vorübergehend in der Shell arbeiten, ohne daß der Inhalt der editierten Datei verlorengeht. Wenn Sie die Shell mit der Taste END beenden, gelangen Sie wieder in den *ced* zurück.

Tasten mit besonderer Bedeutung

Außer den bereits beschriebenen Tasten haben die folgenden eine besondere Bedeutung:

⌫ einzelne Zeichen löschen, auch im Bedienbereich.

CTRL R Bildschirminhalt neu ausgeben, wenn er durcheinandergeraten ist, z.B. wenn Sie die Meldung eines Hintergrund-Prozesses erhalten haben.

CTRL X Eingabe in der Bedienzeile ganz löschen.

CTRL @ "Notbremse": *ced* beenden, wenn er auf keine Eingabe mehr reagiert. Sollte anschließend immer noch keine normale Eingabe möglich sein, so sprechen Sie den Systemverwalter an.

Vordefinierte Standard-Funktionen

START zurück an den Dateianfang.

F9 Datei sichern.

F10 Dokument wechseln: wechseln in eine zweite Datei, die Sie bereits mit Modus d (Datei wechseln) geöffnet haben und zurück.

F11 Zeile markieren: markiert wird die Schreibmarkenzeile. Sie kann dann mit F12 kopiert werden.

F12 Zeile kopieren: die Zeile, die Sie vorher mit F11 markiert haben, wird oberhalb der Schreibmarkenzeile kopiert.

F13 suchen vorwärts: Sie müssen vorher im Modus s eine Zeichenkette definiert haben, die gesucht werden soll.

F14 suchen rückwärts, sonst wie F13.

F15 eine Seite vorblättern

F16 eine Seite zurückblättern

F17 Zeile zurückholen: die Zeile oder der Zeilenbereich, der zuletzt gelöscht und zwischengespeichert wurde, wird zurückgeholt (abhängig davon, ob Modus *z* oder *b* eingestellt war).

Legen Sie den mitgelieferten beschrifteten Streifen, der die vordefinierten Standard-Funktionen der Tasten anzeigt, auf Ihre Tastatur in den dafür vorgesehenen Bereich.

FEHLERMELDUNGEN

```
ced: no status document /var/ced/CEttyname.benutzerkennung, start not possible
without document name
```

– Sie haben versucht, den *ced* aufzurufen ohne eine Datei anzugeben, die Sie bearbeiten wollen. Die Datei */var/ced/CEttyname.benutzerkennung* enthielt jedoch keine passenden Informationen über die zuletzt bearbeitete Datei, weil diese Datei nicht existiert oder sich nicht mehr im erwarteten Verzeichnis befindet.

```
ERROR IN COMMAND EXECUTION
```

– Sie haben im Modus *c* (Kommando ausführen) ein falsches oder unpassendes Kommando angewendet.

– Die Syntax des verwendeten Kommandos ist falsch oder fehlerhaft.

```
INVALID INPUT
```

– Sie haben im Modus *s* (suchen) auf eine der Pfeiltasten ↖, ← oder → gedrückt. Verwendbar sind in diesem Modus jedoch nur ↑, ↓ und ↵.

– Sie haben im Modus *b* (Zeilenbereich bearbeiten), *l* (Zeile bearbeiten) oder *r* (Rechteck bearbeiten) eine nicht existierende Funktion gewählt.

– Sie haben im Auswahlmenü einen nicht existierenden Modus gewählt.

– Sie haben versucht, die *HELP*-Kurzbeschreibung des *ced* durch mehrmaliges Drükken der Taste END zu verlassen. Verlassen Sie *HELP* mit der Taste MENU.

– Sie haben im Auswahlmenü versucht, mit der Taste END Ihre *ced*-Sitzung zu beenden. Wählen Sie stattdessen Modus *q* (*ced* verlassen).

– Sie wollten den Modus *s* (suchen) oder den Modus *w* (Fenster positionieren) mit der Taste END verlassen. Drücken Sie stattdessen die Taste MENU.

```
invalid key: press <MENU> to continue
```

Sie haben im Modus *x* (Tastenbelegung anzeigen) die Taste END gedrückt, um den Modus zu verlassen. Drücken Sie stattdessen die Taste MENU.

LINE TOO LONG

Die Zeile, die Sie gerade bearbeiten, wird länger als das zulässige Maximum von 512 Zeichen. Dies kann geschehen, wenn Sie

- im Modus *e* (neuen Text eingeben) eine zu lange Zeile eingeben
- im Modus *i* (einfügen) den Text ab der Schreibmarkenposition über die 512. Spalte hinaus nach rechts verschieben
- im Modus *l* (Zeile bearbeiten) die Funktion j gewählt haben und die beiden Zeilen, die Sie verbinden wollen, zusammen länger als 512 Zeichen werden.
- im Modus *r* (Rechteck bearbeiten) die Funktion h gewählt haben und mindestens eine der Zeilen des Rechtecks, das sie horizontal einfügen wollen, zusammen mit der an der entsprechenden Einfügeposition stehenden Zeile mehr als 512 Zeichen ergibt.

NO ALTERNATIVE DOCUMENT AVAILABLE

Sie haben im Modus *d* (Datei wechseln) den Namen der Datei, in die Sie wechseln möchten, nicht angegeben.

NO TEXT

Sie haben im Modus *s* (suchen) auf eine der Funktionstasten F13 (vorwärts suchen) oder F14 (rückwärts suchen) gedrückt, ohne vorher eine Suchzeichenkette zu definieren.

NO TEXT AVAILABLE

Sie haben im Modus *b* (Zeilenbereich bearbeiten) einen Bereich markiert und anschließend versucht, ihn mit der Funktion i oder der Funktionstaste F17 zurückzuholen, ohne ihn jedoch zuvor mit der Funktion d gelöscht und zwischengespeichert zu haben.

THIS KEY CANNOT BE CHANGED

Sie haben im Modus *p* (Tasten programmieren) versucht, entweder die Taste P oder MENU zu programmieren. Verwenden Sie zum Programmieren von Makros nur die alphanumerischen Tasten oder die Funktionstasten F1 bis F8.

DATEIEN

/var/ced/CEkey.benutzername
Datei, in der die Tastenbelegungen des jeweiligen Benutzers gespeichert werden.

/var/ced/CEttyname.benutzerkennung
Datei, aus der *ced* die Informationen entnimmt, welche Datei ein Benutzer zuletzt mit dem *ced* editiert hat und an welcher Stelle die Schreibmarke im Text zuletzt stand.

/var/ced/CEerror.nummer
Datei, in der Fehlermeldungen während der Ausführung eines SINIX-Kommandos zwischengespeichert werden.

/tmp/CEtextanummer
Datei, in der alle Änderungen enthalten sind, die im Lauf einer *ced*-Sitzung an einem Dokument vorgenommen wurden

/tmp/CEtextbnummer
Kopie der Datei, die gerade mit *ced* bearbeitet wird.

INTERNATIONALE UMGEBUNG

Wenn Sie nicht in einer englischen, sondern in einer anderssprachigen Umgebung arbeiten, dann gibt *ced* die Meldungstexte und die Menüfunktionen in dieser Sprache aus. Die Sprache wird durch die NLS-Umgebungsvariable *LANG* definiert.

Weitere Informationen zur internationalen Umgebung finden Sie in *Kapitel 3, Internationale Umgebung - NLS*.

UMGEBUNGSVARIABLEN

CED_TABS
Ist diese Variable auf NO gesetzt, dann werden beim Abspeichern einer Datei die Leerzeichen nicht in Tabulatorzeichen umgewandelt. Tabulatoren am Zeilenanfang, die zu Beginn der Sitzung schon in der Datei vorhanden waren, bleiben jedoch unverändert erhalten.

TMPDIR
Standardmäßig werden temporäre Dateien im Dateiverzeichnis */tmp* abgelegt.
Mit der Variablen *TMPDIR* bestimmen Sie den Pfad, unter dem temporäre Dateien des *ced* angelegt werden, wenn gegebenenfalls der Speicherplatz unter */tmp* nicht ausreichen sollte.

SHELL

Mit dieser Variablen bestimmen Sie, welche Shell beim Aufruf einer Shell bzw. bei Ausführung eines Kommandos verwendet werden soll. Ist diese Variable nicht gesetzt, so kann auch keine Subshell bzw. kein Kommando aufgerufen werden.

TERM

Mit dieser Variablen wird der Typ der Datensichtstation gesetzt. Ist diese Variable nicht gesetzt, so können Sie den *ced* nicht starten.

SIEHE AUCH

ed, *ex*, *sed*, *vi*

chgrp
Gruppennummer einer Datei ändern (change group)

chgrp weist einer Datei oder einem Dateiverzeichnis eine neue Benutzergruppe zu.

Nur der Systemverwalter darf die Benutzergruppe für jede Datei beliebig verändern. Mit der Option *_POSIX_CHOWN_RESTRICTED* des Betriebssystems kann für Benutzer ohne Systemverwalterrechte die Möglichkeit, eine Benutzergruppe zu verändern, eingeschränkt werden. Ist diese Option gesetzt, darf die Gruppe nur für eigene Dateien geändert werden. Der Benutzer muß dann

- in der Datei */etc/group* als Mitglied der beim chgrp-Aufruf angegebenen neuen Gruppe eingetragen sein (siehe *DATEIEN*),
- aktuell zu dieser neuen Gruppe gehören, d.h., er muß vor dem Aufruf von *chgrp* mit dem Kommando *newgrp* in die neue Benutzergruppe wechseln (siehe *newgrp* und *BEISPIELE*).

Wird *chgrp* von einem Benutzer ohne Systemverwalterrechte aufgerufen, werden für die angegeben Dateien alle gesetzten s-Bits (set-user-ID und set-group-ID, siehe *chmod*) zurückgesetzt.

chgrp [-R] [-h] neuegruppe datei ...

-R
(R - rekursiv) *chgrp* ändert die Gruppennummer rekursiv in allen angegebenen Dateiverzeichnissen und deren Unterverzeichnissen. Dabei werden auch symbolische Links durchlaufen.

-h
Ist *datei* ein symbolischer Link, ändert *chgrp* dessen Gruppennummer. Ohne diese Option wird die Gruppe der Datei verändert, auf die der symbolische Link verweist.

neuegruppe
Neuer Gruppenname oder neue Gruppennummer. *neuegruppe* muß in der Datei */etc/group* eingetragen sein.

datei
Name der Datei, die eine neue Benutzergruppe erhalten soll. *datei* kann auch ein Dateiverzeichnis sein. Pro Aufruf können Sie mehrere Namen angeben.

FEHLERMELDUNGEN

datei: Not owner

Sie dürfen die Benutzergruppe der angegebenen Datei nicht ändern, da Sie nicht Eigentümer dieser Datei sind, nicht als Mitglied der angegebenen Gruppe eingetragen sind oder nicht aktuell zu dieser Gruppe gehören. Nur der Systemverwalter darf die Gruppe für alle Dateien beliebig ändern.

chgrp: unknown group: *neuegruppe*

Sie haben für *neuegruppe* einen Gruppennamen angegeben, der nicht in der Datei */etc/group* eingetragen ist.

DATEI

/etc/group
Die Datei */etc/group* enthält alle eingerichteten Benutzergruppen. Jede Zeile dieser Datei besteht aus vier Feldern, die durch Doppelpunkte getrennt sind:

gruppenname:[kennwort]:gruppennummer:benutzer,benutzer

Nur der Systemverwalter darf neue Benutzergruppen einrichten und neue Gruppenmitglieder eintragen.

BEISPIEL

Sie arbeiten unter der Benutzerkennung *berta*. Diese Kennung ist in der Datei */etc/group* als Mitglied der Benutzergruppen *ag* und *prog* eingetragen. Aktuell gehören Sie zu der Benutzergruppe *ag*. Das können Sie daran erkennen, daß bei neu angelegten Dateien für "Gruppe" der Name *ag* eingetragen wird:

```
$ >datei
$ ls -l datei
-rw-------   1 berta    ag          2426   Feb 17 15:48 datei
```

Sie möchten nun für *datei* die Benutzergruppe ändern. Die neue Gruppe soll *prog* sein. Dazu wechseln Sie mit dem Kommando *newgrp* in die Gruppe *prog* und ändern dann mit *chgrp* die Gruppe für *datei*:

```
$ newgrp prog
$ chgrp prog datei
$ ls -l datei
-rw-------   1 berta    prog        2426   Feb 17 15:48 datei
```

SIEHE AUCH

chmod, *chown*, *id*, *newgrp*
chown() [19]
group, *passwd* [7]

chmod
Zugriffsrechte ändern (change mode)

chmod ändert die Zugriffsrechte für Dateien.

Nur der Eigentümer der Datei oder der Systemverwalter darf die Zugriffsrechte ändern. Das s-Bit für die Gruppe darf nur ein Benutzer setzen, dessen aktuelle Gruppennummer mit der Gruppennummer der Datei übereinstimmt, deren Zugriffsrechte verändert werden (siehe *chgrp* und *newgrp*).

chmod [-R] modus datei ...

-R

(R - rekursiv) *chmod* ändert rekursiv die Zugriffsrechte aller Dateien in allen angegebenen Verzeichnissen und deren Unterverzeichnissen.

modus

Mit *modus* geben Sie an, wie Sie die Zugriffsrechte für die angegebenen Dateien ändern wollen. Sie können *modus* auf zwei Arten definieren:

- durch eine symbolische Angabe
- durch eine absolute Angabe

Symbolische Angabe: `[wer]darf[was][,[wer]darf[was]]...`

wer

Mit *wer* geben Sie an, für wen Sie die Zugriffsrechte ändern wollen.
wer kann sein:

u	(u - user) für Eigentümer
g	(g - group) für Gruppe
o	(o - other) für andere Benutzer
a	(a - all) für *ugo*, d.h. für alle Benutzer

oder eine Kombination aus den Buchstaben *u*, *g*, *o*.

wer nicht angegeben:
für *wer* gilt *ugo*, d.h. alle Benutzer.

darf

Mit *darf* geben Sie an, ob Sie die angegebenen Zugriffsrechte erteilen, unverändert lassen oder entziehen wollen.
darf kann sein:

+	Zugriffsrechte neu erteilen
-	Zugriffsrechte entziehen
=	Zugriffsrechte absolut setzen, d.h., es werden genau die angegebenen Zugriffsrechte erteilt, alle anderen werden entzogen.

was

Mit *was* geben Sie an, welche Zugriffsrechte Sie erteilen bzw. entziehen wollen. *was* kann eine Kombination folgender Optionen sein:

r (r - read) für Leserecht

w (w - write) für Schreibrecht

x (x - execute) für Ausführrecht bzw. für Recht zum Durchlaufen von Dateiverzeichnissen.

s für s-Bit (set-user-ID-Bit bzw. set-group-ID-Bit).
Die Angabe *s* beim *chmod*-Aufruf ist nur zusammen mit *u*, *g* oder *ug* wirksam (wird *wer* nicht angegeben, so wird hierfür *ug* angenommen). Ein gesetztes s-Bit ist nur für ausführbare Binärdateien (nicht für Shell-Prozeduren!) wirksam (siehe *Das s-Bit*).

t für Sticky-Bit (t-Bit).
Nur der Systemverwalter kann das Sticky-Bit setzen. Versucht ein nichtprivilegierter Benutzer das Sticky-Bit zu setzen, so wird dies ignoriert. Die Angabe *t* beim *chmod*-Aufruf ist nur zusammen mit *u*, *a* oder ohne Angabe für *wer* wirksam. Ein gesetztes Sticky-Bit ist nur für ausführbare Dateien wirksam (siehe *Das Sticky-Bit*). Werden die Zugriffsrechte einer Datei mit gesetztem Sticky-Bit geändert, so wird das Sticky-Bit automatisch gelöscht.

l (l - lock) für obligatorische Sperre von Dateien, Dateiverzeichnissen oder Datensätzen. Hierbei können Schreib- und Lesezugriff gesperrt werden, solange ein Programm auf *datei* zugreift.
für *datei* darf nicht das l-Bit gesetzt sein, wenn

- für *datei* gleichzeitig das Ausführrecht für Gruppe gesetzt ist,
- für *datei* gleichzeitig das s-Bit gesetzt ist.

Folgende Beispiele sind deshalb *nicht* korrekt und führen zu Fehlermeldungen:

chmod g+x,+l datei
chmod g+s,+l datei

u Die Zugriffsrechte des aktuellen Eigentümers sollen übernommen werden.

g Die Zugriffsrechte der aktuellen Gruppe sollen übernommen werden.

o Die Zugriffsrechte der aktuellen anderen Benutzer sollen übernommen werden.

was nicht angegeben:
Dies ist nur sinnvoll in Kombination mit dem Gleichheitszeichen = ; dem betreffenden *wer* werden dann alle Zugriffsrechte entzogen.

Wie oben angegeben, können Sie mehrere "wer-darf-was"-Angaben, getrennt durch Kommas, aneinanderreihen, z.B.

```
chmod g-w,o-rw datei
```

chmod arbeitet die Zeichenkette, die Sie für *modus* angeben, von links nach rechts ab. So erhält z.B. durch die Angabe a-w,u+w der Eigentümer Schreibrecht, während es allen anderen entzogen wird.

Absolute Angabe

Eine absolute Angabe für *modus* ist eine drei- oder vierstellige Oktalzahl. Die zulässigen Oktalzahlen erhalten Sie, wenn Sie die untenstehenden oktalen Modi im Binärsystem mit dem bitweisen logischen ODER verknüpfen. Dasselbe Ergebnis erhalten Sie, wenn Sie die Modi (im Oktal- oder Dezimalsystem) addieren. Eine führende Null (weder s-Bit noch Sticky-Bit) können Sie weglassen.
Die angegebenen Zugriffsrechte werden erteilt, alle anderen Zugriffsrechte werden entzogen.

4000 s-Bit für Eigentümer
20#0 s-Bit für Gruppe, wenn # gleich 7, 5, 3 oder 1 ist. Eine obligatorische Sperre wird gesetzt, wenn # gleich 6, 4, 2 oder 0 ist. Der Wert von # wird ignoriert, wenn *datei* ein Dateiverzeichnis ist. In diesem Fall dürfen Sie nur die symbolische Angabe verwenden.
1000 Sticky-Bit (t-Bit)
0400 Leserecht für Eigentümer
0200 Schreibrecht für Eigentümer
0100 Ausführrecht (bzw. Recht zum Durchlaufen von Dateiverzeichnissen) für Eigentümer
0040 Leserecht für Gruppe
0020 Schreibrecht für Gruppe
0010 Ausführrecht (bzw. Recht zum Durchlaufen von Dateiverzeichnissen) für Gruppe
0004 Leserecht für andere Benutzer
0002 Schreibrecht für andere Benutzer
0001 Ausführrecht (bzw. Recht zum Durchlaufen von Dateiverzeichnissen) für andere Benutzer

Beispiel

Wenn Sie für den Eigentümer das Lese-, Schreib- und Ausführrecht setzen wollen, und für Gruppe das Lese- und Ausführrecht, dann geben Sie für *modus* 750 an:

400 + 200 + 100 + 40 + 10 = 750

Für die Datei gelten dann die Zugriffsrechte `rwxr-x---`.

datei
Name der Datei, für die Sie die Zugriffsrechte definieren oder ändern wollen. *datei* kann auch ein Dateiverzeichnis sein. Pro Aufruf können Sie mehrere Namen angeben.

Das s-Bit

Ist für ein ausführbares Programm das s-Bit für den Eigentümer gesetzt, dann ist beim Aufruf des Programms die *effektive* Benutzernummer des zugehörigen Prozesses gleich der Benutzernummer des *Eigentümers* der Datei (und nicht gleich der Benutzernummer des Aufrufers). Das bedeutet, daß der Prozeß unter der Benutzernummer des Programm-Eigentümers abläuft. Dadurch kann er auch auf Dateien zugreifen, für die der Programm-Aufrufer direkt kein Zugriffsrecht hat.
Die *reale* Benutzernummer des Prozesses bleibt die des Programm-*Aufrufers*.
Ist das s-Bit für die Gruppe nicht gesetzt, dann bleibt die effektive Gruppennummer gleich der realen Gruppennummer des Prozesses: das ist die Gruppennummer (GID) des Programm-Aufrufers.

Ist das s-Bit für die Gruppe gesetzt, dann ist die *effektive* Gruppennummer des Prozesses gleich der Gruppennummer des Programm-*Eigentümers*. Der Prozeß läuft also unter der Gruppennummer des Programm-Eigentümers ab.
Die *reale* Gruppennummer des Prozesses bleibt die des Programm-*Aufrufers*.

Das s-Bit wirkt nur bei ausführbaren Binärdateien (ausführbaren Programmen), nicht aber bei Shell-Prozeduren. Für Dateien, die eine Shell-Prozedur enthalten, kann das s-Bit zwar mit *chmod* gesetzt werden, es bleibt jedoch wirkungslos.

Bei Dateiänderungen werden die s-Bits aus Sicherheitsgründen zurückgesetzt.

Vorsicht

Setzt der Systemverwalter für ein Programm, das ihm gehört, das s-Bit, erlaubt er allen Benutzern, die das Programm aufrufen dürfen, alle Operationen, die er selbst mit Hilfe dieses Programms ausführen darf. Er darf das s-Bit also nur dann setzen, wenn sichergestellt ist, daß dadurch keine Daten gefährdet werden.

Ein Beispiel für die Anwendung des s-Bits ist das Kommando *passwd* (Dateiverzeichnis */bin*). Als normaler Benutzer haben Sie für die Kennwortdatei */etc/passwd* kein Schreibrecht. Da für das Kommando *passwd* jedoch das s-Bit für den Eigentümer gesetzt ist, können Sie über dieses Kommando ein neues Kennwort in die Datei */etc/passwd* schreiben.

Das Sticky-Bit (t-Bit)

Nur der Systemverwalter kann das Sticky-Bit (t-Bit) setzen. Versucht ein nichtprivilegierter Benutzer das Sticky-Bit zu setzen, so wird dies ignoriert.

Das Sticky-Bit wirkt nur bei Dateiverzeichnissen und ausführbaren Dateien. Für andere Dateien kann es zwar mit *chmod* gesetzt werden, es ist jedoch wirkungslos.

Ist das Sticky-Bit bei ausführbaren Dateien gesetzt, kann beim Programmstart das Einlesen des Programms aus der Programmdatei und das Auslagern in den Swap-Bereich teilweise vermieden werden.

Wenn ein Dateiverzeichnis schreibbar ist, aber das Sticky-Bit gesetzt hat, muß eine der folgenden Bedingungen erfüllt sein, um eine Datei unter diesem Dateiverzeichnis zu löschen, zu verschieben oder einen Verweis auf diese Datei zu erzeugen:

- die Datei muß dem Benutzer gehören
- das Dateiverzeichnis muß dem Benutzer gehören
- der Benutzer muß Schreibberechtigung für die Datei haben
- der Benutzer muß ein privilegierter Benutzer sein

Bei der Ausgabe des Kommandos *ls -l* erscheint ein gesetztes Sticky-Bit an der letzten Stelle der Zugriffsrechte. Ist gleichzeitig das x-Bit für "andere Benutzer" gesetzt, so wird ein *t* ausgegeben, andernfalls ein *T*.

Das l-Bit

Mit Hife der Funktion *lockf()* kann ein Programm eine Datei sperren, solange das Programm auf diese Datei zugreift. Ist für diese Datei das l-Bit gesetzt, bewirkt der Funktionsaufruf eine obligatorische Sperre (mandatory locking) der Datei (siehe *lockf()* [19]).

FEHLERMELDUNGEN

`chmod: ERROR: Invalid mode`

Sie haben für *chmod* einen unzulässigen Modus angegeben.

`chmod: WARNING: Locking not permitted on datei, a group executable file`

Sie wollten eine Datei zur Ausführung sperren, obwohl für diese Datei das Ausführrecht für Gruppe gesetzt ist.

`chmod: WARNING: Execute permission requiered for set-ID on execution for datei`

Sie wollten eine Datei zur Ausführung sperren, obwohl für diese Datei das s-Bit gesetzt ist.

BEISPIELE

Die folgenden Beispiele beziehen sich alle auf eine Datei mit den Zugriffsrechten `rw-------`. Die ersten beiden Spalten der Tabelle enthalten Angaben für *modus*, die letzte Spalte enthält das Ergebnis des jeweiligen *chmod*-Aufrufs.

Symbolische Angabe	Absolute Angabe	Ergebnis
u-w -w	400 400	r-------- r--------
go+r go=r	644 644	rw-r--r-- rw-r--r--
go+rw =rw	666 666	rw-rw-rw- rw-rw-rw-
+rx	755	rwxr-xr-x
=r	444	r--r--r--
ug=rw,o=r	664	rw-rw-r--
u=rwx,g=rx,o=	750	rwxr-x---
+x,u+s	4711	rws--x--x
+xt	1711	rwx--x--t

Das Sticky-Bit (letztes Beispiel) kann nur der Systemverwalter setzen. Versucht ein nicht-privilegierter Benutzer das Sticky-Bit zu setzen, so wird dies ignoriert.

SIEHE AUCH

chgrp, *ls*, *newgrp*, *umask*
chmod(), *chown()*, *lockf()* [19]
passwd [7]

chown
Eigentümer einer Datei ändern (change owner)

chown weist einer Datei oder einem Dateiverzeichnis einen neuen Eigentümer zu.

Nur der Systemverwalter darf den Eigentümer einer Datei beliebig verändern. Benutzer ohne Systemverwalterrechte dürfen hingegen nur den Eigentümer ihrer eigenen Dateien verändern. Ist die Option *_POSIX_CHOWN_RESTRICTED* des Betriebssystems gesetzt, besteht auch diese Möglichkeit nicht.

Wird *chown* von einem Benutzer ohne Systemverwalterrechte aufgerufen, dann wird das s-Bit für Eigentümer, 4000, zurückgesetzt.

chown [-R] [-h] neuereigentümer datei ...

-R
(R - rekursiv) *chown* ändert den Eigentümer rekursiv in allen angegebenen Dateiverzeichnissen und deren Unterverzeichnissen. Dabei werden auch symbolische Links durchlaufen.

-h
Ist *datei* ein symbolischer Link, ändert *chown* dessen Eigentümer. Ist diese Option nicht gesetzt, wird der Eigentümer der Datei verändert, auf die der symbolische Link verweist.

neuereigentümer
Benutzerkennung oder Benutzernummer des neuen Eigentümers. Geben Sie eine Benutzerkennung an, so muß diese in der Datei */etc/passwd* eingetragen sein.

datei
Name der Datei, die einen neuen Eigentümer erhalten soll. *datei* kann auch ein Dateiverzeichnis sein. Pro Aufruf können Sie mehrere Namen angeben.

FEHLERMELDUNGEN

`chown: datei: Not owner`

Sie dürfen den Eigentümer der Datei *datei* nicht ändern, da Sie nicht Eigentümer von *datei* sind.

`chown: unknown user id neuereigentümer`

Sie haben für *neuereigentümer* eine Benutzerkennung angegeben, die nicht in der Datei */etc/passwd* eingetragen ist.

DATEI

/etc/passwd
Die Datei */etc/passwd* enthält alle eingerichteten Benutzerkennungen. Nur der Systemverwalter darf neue Benutzerkennungen einrichten.

BEISPIEL

Sie arbeiten als Systemverwalter und wollen den Eigentümer einer Datei ändern. Die Datei *text1* der Benutzerin *carla* soll als neuen Eigentümer den Benutzer *markus* erhalten. Dazu geben Sie ein:

```
# ls -l text1
-rw-------   1 carla    ag          2426   Feb 17 15:48 text1
# chown markus text1
# ls -l text1
-rw-------   1 markus   ag          2426   Feb 17 15:48 text1
```

SIEHE AUCH

chgrp, *chmod*
chown() [19]
passwd [7]

clear
Bildschirm löschen

clear löscht den Bildschirm der Datensichtstation. Anschließend gibt die Shell wieder das Bereitzeichen aus.

clear

clear prüft über die Umgebungsvariable *TERM* der Arbeitsumgebung, um welchen Typ von Datensichtstation es sich handelt.
Anhand dieser Information entscheidet *clear*, ob und wie der Bildschirm gelöscht werden kann. Wenn die *TERM*-Variable gesetzt ist, löscht *clear* den Bildschirm.

UMGEBUNGSVARIABLE

TERM
enthält Informationen zum Typ der Datensichtstation

BEISPIEL

```
$ set | grep TERM ↵
TERM=98765

$ clear ↵
$
```

SIEHE AUCH

tput

cmp
Dateien zeichenweise vergleichen (compare)

Mit *cmp* können Sie zwei Dateien zeichenweise vergleichen. Wenn es Unterschiede zwischen den beiden Dateien gibt, dann gibt *cmp* die Unterschiede auf die Standard-Ausgabe aus.
Wenn die beiden Dateien identisch sind, gibt *cmp* nichts aus.

cmp[␣-l][␣-s]␣datei1␣datei2

Keine Option angegeben
Wenn die Dateien identisch sind, gibt *cmp* nichts aus.
Wenn sich die Dateien unterscheiden, gibt *cmp* die Zeichennummer und die Zeilennummer des *ersten* Unterschieds zwischen *datei1* und *datei2* in folgender Form aus:

```
datei1 datei2 differ: char zeichennummer, line zeilennummer
```

-l
Alle Unterschiede werden in folgender Form ausgegeben:

```
zeichennummer   zeichen(datei1)   zeichen(datei2)
```

zeichennummer ist die Position der abweichenden Zeichen ab Dateianfang. Dabei erhält das erste Zeichen einer Datei die Nummer 1, Leerzeichen werden mitgezählt. *zeichennummer* wird dezimal ausgegeben.

zeichen sind die voneinander abweichenden Zeichen in *datei1* und *datei2*. *zeichen* werden oktal ausgegeben. Eine ASCII-Tabelle mit der Aufschlüsselung der oktalen Werte finden Sie in *Tabellen und Verzeichnisse*. Wenn die Dateien identisch sind, wird nichts ausgegeben.

-s
cmp gibt nichts aus. Der Wert des Ende-Status wird zurückgeliefert, aber nicht automatisch am Bildschirm ausgegeben.

datei1 datei2
Namen der Dateien, die Sie vergleichen wollen.

- Wenn Sie für *datei1* den Bindestrich - angeben, dann liest *cmp* von der Standard-Eingabe und vergleicht Ihre Eingaben mit *datei2*.
- Wenn eine der beiden Dateien endet, ohne daß *cmp* einen Unterschied feststellen konnte, dann meldet *cmp* mit folgendem Text, daß in der kürzeren Datei EOF (Dateiende) erkannt wurde:

  ```
  cmp: EOF on datei
  ```
- Wenn in einer von zwei sonst gleichen Dateien ein Zeichen fehlt, dann meldet *cmp -l* wegen der Verschiebung der Positionen alle folgenden Zeichen als unterschiedlich.
- Dem ersten Zeichen einer Datei ist als *zeichennummer* die 1 und nicht die 0 zugeordnet.
- Leerzeichen und Neue-Zeile-Zeichen zählen bei den *zeichennummern* mit.

ENDE-STATUS

0 Dateien sind identisch
1 Dateien sind unterschiedlich
2 Auf Datei kann nicht zugegriffen werden, oder Argument fehlt.

FEHLERMELDUNG

`cmp: cannot open` *datei*

Sie haben für eine Datei kein Leserecht oder eine der angegebenen Dateien existiert nicht.

BEISPIELE

1. Vergleich zweier Dateien mit Ausgabe der unterschiedlichen Zeichen und ihrer Position *zeichennummer*.

```
$ echo 1 2 3 4 5 7 8 a >dat1
$ echo 1 2 3 4 5 6 9 a >dat2
$ cmp -l dat1 dat2
11  67  66
13  70  71
$
```

2. Die Shell-Prozedur *lösche.gl* vergleicht zwei Dateien und löscht bei Gleichheit eine der beiden.

```
$ cat >lösche gl ↵
if cmp -s $1 $2 ↵
then ↵
rm $2 ↵
fi ↵
END
$
```

Beim Aufruf der Prozedur mit

```
$ lösche gl dat1 dat2
```

übergeben Sie *dat1* und *dat2* als Stellungsparameter an die Prozedur. *cmp* liefert mit der Option *-s* den Ende-Status zurück. Wenn sein Wert gleich 0 ist, wird *dat2* gelöscht, sonst nicht.

SIEHE AUCH

comm, diff

col
Filter für umgekehrte Zeilenvorschübe

col liest von der Standard-Eingabe und schreibt das Ergebnis auf die Standard-Ausgabe. Es verarbeitet Zeilen, die sich aufgrund von ganzen oder halben Zeilenvorschüben rückwärts oder halben Zeilenvorschüben vorwärts überlagern. Als Eingabe akzeptiert *col* halbzeilige Bewegungen, gibt sie normalerweise aber nicht auf die Standard-Ausgabe weiter. Stattdessen wird der Text, der zwischen zwei Zeilen erscheinen würde, hinter bzw. vor die nächste ganze Zeilengrenze geschoben. Zeilenvorschübe rückwärts von der ersten Zeile aus werden ignoriert. Ein oder mehrere Leerzeichen (oktal 40) ersetzt *col* in der Ausgabe durch Tabulatorzeichen, wenn dadurch die Position des folgenden Nicht-Leerzeichens erhalten bleibt. Die Ausgabezeit wird dadurch möglichst kurz gehalten. Standardmäßig liegt immer nach
8 Zeichen eine Tabulatorposition. Die ASCII-Steuerzeichen SO und SI (siehe *Tabellen und Verzeichnisse, Zeichensatz ISO 646*) werden von *col* als Anfang und Ende eines Textes in einem alternativen Zeichensatz interpretiert. Der Zeichensatz, zu dem jedes Eingabezeichen gehört, wird gespeichert. Bei der Ausgabe werden die Zeichen SI und SO generiert; dadurch ist sichergestellt, daß jedes Zeichen dem korrekten Zeichensatz entnommen wird.

col eignet sich speziell zur Filterung einer mehrspaltigen Ausgabe, die mit dem Kommando *.rt* von *nroff* erzeugt wurde oder einer Ausgabe, die sich aus der Anwendung des Präprozessors *tbl* ergibt. Das von *col* akzeptierte Eingabeformat stimmt mit der Ausgabe überein, die *nroff* mit der Option *-T37* oder *-Tlp* erzeugt. Verwenden Sie die Option *-T37* (und die Option *-f* von *col*), wenn Sie auf ein Gerät ausgeben, daß halbe Zeilenvorschübe interpretieren kann. Sonst wählen Sie *-Tlp*.

col ignoriert lokale vertikale Bewegungen, die über die erste Zeile des Dokuments hinausgehen. Die erste Zeile darf deshalb keine Überschriften enthalten.

col kann nicht mehr als 128 Zeilen rückwärts gehen und nicht mehr als 800 Zeichen pro Zeile verarbeiten.

Nicht druckbare Zeichen in der Eingabe werden bis auf folgende Ausnahmen ignoriert:

Nicht druckbare Zeichen, die *col* interpretiert:	ASCII-Wert (oktal)	so einzugeben
Leerzeichen	040	Leerzeichen-Taste
Rücksetzzeichen (Backspace)	010	⌫
Tabulator	011	⇥
Wagenrücklauf	015	CTRL M od. entspr. Taste
Neue-Zeile-Zeichen	012	CTRL J od. entspr. Taste
SI	017	CTRL O
SO	016	CTRL N
VT	013	CTRL K
Zeilenvorschub rückwärts	033 067	ESC 7
halber Zeilenvorschub rückwärts	033 070	ESC 8
halber Zeilenvorschub vorwärts	033 071	ESC 9

Die Taste CTRL müssen Sie gleichzeitig mit der folgenden Taste, die Taste ESC und die nachfolgende Taste müssen Sie nacheinander drücken.

col entfernt normalerweise alle eingegebenen Escape-Sequenzen, die es nicht kennt, es sei denn, die Option *-p* ist gesetzt.

col[_option]...

option

-b

col nimmt an, daß auf einem Gerät ausgegeben werden soll, das nicht zu Rückwärtsschritten (b - backspacing) fähig ist. Falls mehrere Zeichen an derselben Position erscheinen sollen, gibt *col* nur das zuletzt gelesene aus.

-f

col gibt halbe Zeilenvorschübe vorwärts aus, nicht aber halbe oder ganze Zeilenvorschübe rückwärts.

-p

col akzeptiert auch andere Escape-Sequenzen als die oben beschriebenen. Verwenden Sie *-p* nur, wenn Sie die Folgen genau kennen!

-x

col wandelt ein oder mehrere Leerzeichen in der Eingabe für die Ausgabe nicht in Tabulatorzeichen um.

FEHLERMELDUNG

```
col: bad option
```

Sie haben eine ungültige Option angegeben.

BEISPIEL

In der Datei *texte* stehen folgende Zeichen:

text1 [ESC] 9text2 [ESC] 9text3 [ESC] 9text4 [ESC] 8 [ESC] 8text5

Mit *col* erhalten Sie folgende Ausgabe auf dem Bildschirm:

```
$ cat texte|col
text1
     text2text3     text5
               text4
```

Der Inhalt einer Datei mit solchen Zeichen kann so auch im normalen Vorwärtslauf ausgegeben werden.

SIEHE AUCH

ascii [7]

Tabellen und Verzeichnisse, Zeichensatz ISO 646

comm
Gleiche Zeilen in zwei sortierten Dateien suchen (common lines)

comm vergleicht zwei Dateien, deren Zeilen nach der aktuell gültigen Sortierreihenfolge sortiert sind. Zum Sortieren können Sie das Kommando *sort* (siehe *sort*) verwenden.

comm[_option]_datei1_datei2

Keine Option angegeben

comm gibt drei Spalten aus, die folgende Bedeutung haben:

Spalte 1	Spalte 2	Spalte 3
Zeilen, die nur in *datei1* stehen	Zeilen, die nur in *datei2* stehen	Zeilen, die in beiden Dateien stehen

option

-1
Spalte 1 wird nicht ausgegeben.

-2
Spalte 2 wird nicht ausgegeben.

-3
Spalte 3 wird nicht ausgegeben.

-12
comm gibt alle Zeilen aus, die beiden Dateien gemeinsam sind.

-23
comm gibt alle Zeilen aus, die nur in *datei1* stehen.

-13
comm gibt alle Zeilen aus, die nur in *datei2* stehen.

-123
comm gibt nichts aus.

datei1 datei2
Namen der beiden sortierten Dateien, die Sie vergleichen wollen.
Das Kommando *comm* arbeitet nur korrekt, wenn die beiden zu vergleichenden Dateien sortiert vorliegen. Wenn Sie für einen der beiden Namen den Bindestrich - eingeben, liest *comm* von der Standard-Eingabe.

BEISPIEL

In der Datei *buch* sind Buchtitel und die dazugehörenden Autoren erfaßt. In jeder Zeile steht ein Buchtitel und dahinter, durch ein Leerzeichen getrennt, der jeweilige Autor. Sie möchten die Datei *buch* nach mehreren Autoren durchsuchen, die Sie in der Datei *autoren1* erfaßt haben. Die Dateien *buch* und *autoren1* haben folgenden Inhalt:

```
buch                                                    autoren1

"Mann_aus_Apulien" Stern                                Duras
"Wir_sind_noch_einmal_davongekommen" Wilder             Goethe
"Tod_in_Venedig" Mann                                   Joyce
"Ulysses" Joyce                                         Mann
                                                        Schiller
                                                        Wilder
```

Sie gehen in folgenden Schritten vor:
- Herausfiltern der Autoren aus *buch* mit *awk*.
- Sortieren der Autoren aus *buch* mit *sort*.
- Umlenken der Ausgabe von *sort* in die neue Datei *autoren2*.
- Vergleichen der beiden Dateien *autoren1* und *autoren2* mit *comm -2*.

```
$ awk '{printf"%s\n",$2}' buch | sort > autoren2
```

Die Datei *autoren2* hat folgenden Inhalt:

```
Joyce
Mann
Stern
Wilder
```

```
$ comm -2 autoren1 autoren2

Duras
Goethe
        Joyce
        Mann
Schiller
        Wilder
```

Ausgegeben werden in der ersten Spalte alle Autoren, die nur in *autoren1* stehen. Darauf folgt der Inhalt der dritten Spalte, das heißt die Namen der Autoren, die in beiden Dateien stehen.

SIEHE AUCH

cmp, *diff*, *sort*, *uniq*

compress
Dateien komprimieren

compress komprimiert Dateien mittels adaptiver Lempel-Ziv-Codierung: Zeichenketten, die sich im Text wiederholen, werden durch eindeutige Codes von 9 bis maximal 16 bit Länge abgekürzt.

Eigentümer, Zugriffsrechte und Zugriffs- oder Änderungsdatum der angegebenen Dateien werden nicht verändert. Jede angegebene Datei wird ersetzt durch eine Datei gleichen Namens mit *.Z*-Suffix.

Der Umfang der Komprimierung hängt ab von der Größe der Eingabedatei, vom Wert für *maxbits* (siehe unten, Option *-b*) sowie von der Verteilung gleicher Zeichenketten. Dateien, die nur Text oder Quellcode enthalten, werden in der Regel um 50-60% komprimiert. Die verwendete Lempel-Ziv-Codierung erreicht im allgemeinen eine bessere Komprimierung als die Codierung nach Huffman (siehe *pack*) und verbraucht auch weniger Rechenzeit.

Eine Komprimierung wird nicht durchgeführt, wenn

- die zu komprimierende Datei keine einfache Datei ist
- auf die zu komprimierende Datei Verweise bestehen
- die zu komprimierende Datei bereits ein *.Z*-Suffix hat
- die anzulegende *.Z*-Datei bereits existiert und *compress* im Hintergrund (*/usr/bin/sh*) abläuft
- die Komprimierung keine Platzersparnis erwarten läßt

Mit *uncompress* können Sie den Originalzustand einer komprimierten Datei wiederherstellen.

Mit *zcat* können Sie komprimierte Dateien im Originalzustand auf die Standard-Ausgabe ausgeben. Die komprimierte Datei wird dabei nicht verändert.

compress[␣option]...[␣datei]...

Keine Option angegeben
: Die angegebenen Dateien werden komprimiert, wenn dadurch Speicherplatz gespart werden kann.

option

-c
: Die Ausgabe von *compress* wird nur auf die Standard-Ausgabe geschrieben. Es wird keine Datei geändert oder angelegt.

-f

(f - force) Die Komprimierung wird erzwungen, auch wenn dadurch kein Speicherplatz gespart wird oder die anzulegende *.Z*-Datei bereits existiert. Diese Datei wird überschrieben.

-f nicht angegeben:
compress fragt nach, ob eine existierende *.Z*-Datei überschrieben werden soll oder nicht. Diese Nachfrage erfolgt jedoch nicht, wenn *compress* im Hintergrund abläuft (wenn Sie unter der normalen Shell */usr/bin/sh* arbeitet).
Bei Shells mit Job-Control, wie z.B. der *ksh*, wartet der Hintergrundprozeß auf Antwort. Dieser Prozeß kann temporär in den Vordergrund geholt werden um die gewünschte Antwort zu erhalten.

-v

(v - verbose) Die prozentuale Platzersparnis für jede komprimierte Datei wird angezeigt:

```
datei Compression: xx.xx% -- replaced with datei.Z
```

-bmaxbits

Das Limit für den Abkürzungs-Code gleicher Zeichenketten wird auf *maxbits* bits festgelegt. Der Wert für *maxbits* muß zwischen 9 und 16 liegen. Eine Herabsetzung des Wertes bewirkt, daß die Dateien weniger komprimiert werden.

Der Parameter *maxbits* wird verschlüsselt in der komprimierten Datei abgelegt, zusammen mit einer Dateikennung (magic number), die sicherstellt, daß weder eine mehrfache Komprimierung noch eine zufällige Dekomprimierung erlaubt ist.

-b nicht angegeben:
Für *maxbits* wird 16 angenommen.

Vorsicht

Komprimierte Dateien sind nur kompatibel zwischen Maschinen mit großem Prozeßdatenbereich. Für Datenübertragungen auf Architekturen mit kleinerem Prozeßdatenbereich (64 Kbyte oder weniger) sollte für die Komprimierung die Option *-b12* angegeben werden.

datei

Name der Datei, die komprimiert werden soll. Sie können mehrere Dateien angeben. *datei* darf kein Dateiverzeichnis sein, und es dürfen keine Verweise auf *datei* bestehen.
Die komprimierte Datei erhält den Namen *datei.Z*, *datei* wird nach erfolgreicher Komprimierung gelöscht. *datei.Z* hat dieselben Zugriffsrechte, dasselbe Zugriffs- bzw. Änderungsdatum und denselben Eigentümer wie *datei*. Der Name von *datei* darf höchstens 12 Zeichen lang sein, damit die Namenserweiterung auf *datei.Z* noch möglich ist. Ist der Name länger, so wird *datei* nicht komprimiert.

datei nicht angegeben:
Die Daten der Standard-Eingabe werden in komprimierter Form auf die Standard-Ausgabe geschrieben.

ENDE-STATUS

0 Komprimierung erfolgreich
1 Fehler
2 Datei wurde nicht komprimiert, da die Komprimierung die Datei vergrößert hätte.

FEHLERMELDUNGEN

datei: filename too long to tack on .Z

Der Name der zu komprimierenden Datei ist länger als 12 Zeichen. *compress* komprimiert nicht.

datei -- not a regular file: unchanged

Die angegebene Datei ist keine einfache Datei. *compress* komprimiert nicht.

datei: -- has xx other links: unchanged

Auf die angegebene Datei bestehen *xx* Verweise. *compress* komprimiert nicht.

datei unchanged

Es kann keine Einsparung durch die Komprimierung erzielt werden. *compress* komprimiert nicht. Wollen Sie dennoch komprimieren, so geben Sie die Option *-f* an.

BEISPIEL

Die Datei *filme*, die in unkomprimiertem Zustand 4862 Byte belegt, wird komprimiert.

```
$ ls -l
  total 10
  -rw-------  1  felix    gruppe1    4862 Aug 19 09:27 filme

$ compress -v filme
compress: filme: 50.78% Compression -- replaced with filme.Z

$ ls -l
  total 6
  -rw-------  1  felix    gruppe1    2393 Aug 19 09:27 filme.Z
```

SIEHE AUCH

uncompress, *zcat*, *pack*, *unpack*, *pcat*

copy
Dateien gruppenweise kopieren (copy groups of files)

Das *copy* Kommando kopiert Inhalte von einem oder mehreren Dateiverzeichnissen in ein anderes Dateiverzeichnis oder es kopiert Benutzer- und Gerätedateien.

copy[_-option..]_quelle..._ziel

Keine Option angegeben

Ist *quelle* kein Dateiverzeichnis, entspricht *copy* dem Kommando *cp*.
Ist *quelle* ein Dateiverzeichnis wird jede Datei aus dem Quell-Dateiverzeichnis kopiert. Untergeordnete Dateiverzeichnisse werden nicht berücksichtigt.
Ein nicht vorhandenes Ziel (Datei oder Dateiverzeichnis) wird neu erstellt und erhält die gleichen Attribute wie die Quelle. Im Gegensatz zu *cp* werden bei *copy* die Zugriffsrechte von der jeweiligen umask-Einstellung (siehe *umask*) eventuell weiter eingeschränkt.
Ein bereits vorhandenes Ziel wird beim Kopieren überschrieben.

option

Mehrere Angaben zum Operanden *option* dürfen in beliebiger Reihenfolge stehen.

-a

copy fragt vor dem Kopieren jeder einzelnen Datei, ob diese Datei wirklich kopiert werden soll. Nur wenn Sie jeweils mit dem Buchstaben *y* bestätigen, wird das *copy*-Kommando ausgeführt.
Die Option *-a* setzt automatisch auch die Option *-ad*.

-ad

copy fragt vor dem Kopieren eines untergeordneten Dateiverzeichnisses, ob dieses wirklich kopiert werden soll. Nur wenn Sie jeweils mit dem Buchstaben *y* bestätigen, wird das *copy*-Kommando ausgeführt.

-l

(l - link) Bei Dateien werden nach Möglichkeit lediglich Verweise gesetzt und keine Kopien erstellt.
Bei Dateiverzeichnissen und Gerätedateien hat der Schalter keine Bedeutung.
Sind als Quelle mehrere Dateien angegeben, dann ist als Ziel ein existierendes Dateiverzeichnis anzugeben.

-m

(m - modification) Wenn der Schalter gesetzt ist, wird für die kopierte Datei die letzte Änderungs- und Zugriffszeit der Quelldatei übernommen. Standard (keine Angabe): Bei Ziel wird der Zeitpunkt des Kopierens als letzte Dateiänderung eingetragen.

-n

(n - new) Die Zieldatei muß eine neue Datei sein. Wenn die Zieldatei schon vorhanden ist, wird das copy-Kommando auf diese Datei nicht ausgeführt. Bei Gerätedateien ist dieser Schalter Voraussetzung. Ist die Quelle ein Dateiverzeichnis, ist er bedeutungslos; er betrifft aber die im Verzeichnis eingetragenen Dateien.

-o

(o - owner) Nur der Systemverwalter darf diesen Schalter angeben. Wenn der Schalter gesetzt ist, werden Eigentümer und Gruppenzugehörigkeit sowie die letzte Änderungs- und Zugriffszeit der Quelldatei übernommen.
Ist der Schalter nicht gesetzt, wird der Kommandoaufrufende zum Eigentümer der kopierten Datei.

-r

(r - recursive) Jedes untergeordnete Dateiverzeichnis wird kopiert. Mit diesem Schalter können (rekursiv) ganze Dateisysteme kopiert werden.

-v

(v - verbose) Protokollierung des Ablaufs von *copy* am Eingabegerät.

quelle

quelle kann entweder eine Datei, ein Dateiverzeichnis oder eine Gerätedatei sein. *quelle* muß vorhanden sein.
Erlaubt sind mehrere Angaben zum Operanden.

ziel

ziel ist entweder eine Datei oder ein Dateiverzeichnis. *ziel* und *quelle* dürfen nicht identisch sein. Beachten Sie, daß *copy* keine Fehlermeldung ausgibt, wenn *quelle* und *ziel* identisch sind!

BEISPIEL

Sie wollen den Inhalt des Dateiverzeichnisses *TOOLS* nach *TOOLS.SIK* kopieren. *TOOLS.SIK* existiert nicht, wird aber durch das *copy*-Kommando angelegt.

```
$ copy TOOLS TOOLS.SIK
```

SIEHE AUCH

cp, *ln*

cp
Dateien kopieren (copy)

cp kopiert Dateien. Kopieren heißt: die Datei ist nachher physisch noch einmal vorhanden.

cp hat zwei Formate. Das Kommando kopiert

- entweder eine Datei in eine Datei mit anderem Namen (Format 1)
- oder eine oder mehrere Dateien in ein anderes Dateiverzeichnis, wobei die Kopien dieselben einfachen Dateinamen haben (Format 2).

cp[␣**-i**][␣**-p**]␣datei␣dateikopie — Format 1
cp[␣**-i**][␣**-p**][␣**-r**]␣datei␣...␣dateiverzeichnis — Format 2

Format 1: Eine Datei kopieren

cp[␣**-i**][␣**-p**]␣datei␣dateikopie

-i
(i - interaktiv). Wenn *dateikopie* bereits existiert, erwartet *cp* eine Bestätigung, daß diese Datei überschrieben werden darf. Geben Sie *y* ein, wird kopiert. Jede andere Eingabe verhindert ein Überschreiben.

-p
cp dupliziert auch das Änderungsdatum von *datei*.

datei
Dateiname des Originals.

dateikopie
Dateiname der Kopie.
Wenn es noch keine Datei mit dem Namen *dateikopie* gibt, wird sie neu angelegt.

Ist die Option *-p* nicht gesetzt, so erhält die Kopie die gleichen Zugriffsrechte wie das Original sowie die Benutzer- und Gruppennummer des Benutzers, der *cp* aufgerufen hat. Das Änderungsdatum wird jedoch nicht dupliziert, d.h., daß das aktuelle Datum gesetzt wird.

Vorsicht

Wenn es bereits eine Datei mit dem Namen *dateikopie* gibt, wird der Inhalt dieser Datei ohne Rückfrage überschrieben, falls sie nicht die Option *-i* angegeben haben; Zugriffsrechte, Eigentümer und Gruppe bleiben aber unverändert.

Wenn *dateikopie* ein Verweis auf eine Datei ist, bleiben alle Verweise erhalten. Der Inhalt der Datei *dateikopie* wird mit dem Inhalt von *datei* überschrieben.

Format 2: Dateien in ein anderes Dateiverzeichnis kopieren

cp[␣**-i**][␣**-p**][␣**-r**]␣datei␣...␣dateiverzeichnis

-i

(i - interaktiv). Wenn *datei* in *dateiverzeichnis* bereits existiert, erwartet *cp* jeweils eine Bestätigung, daß diese Datei überschrieben werden darf. Geben Sie *y* ein, wird kopiert. Jede andere Eingabe verhindert ein Überschreiben.

-p

cp dupliziert auch das Änderungsdatum von *datei*.

-r

(r - rekursiv). Ist *datei* ein Dateiverzeichnis, kopiert *cp* rekursiv dessen Inhalt und den Inhalt aller Unterverzeichnisse.

datei

Dateiname des Originals. Sie können mehrere Namen angeben und so auf einmal mehrere Dateien kopieren. Die Kopien erhalten jeweils denselben einfachen Dateinamen wie die Originale.
Ist *datei* ein Dateiverzeichnis, muß sich das Zielverzeichnis in demselben physikalischen Dateisystem befinden. Quell- und Zielverzeichnis brauchen jedoch kein gemeinsames Elternverzeichnis zu besitzen.

Vorsicht

Wenn es in *dateiverzeichnis* bereits eine Datei gibt, die denselben einfachen Dateinamen hat wie die Originaldatei, so wird der Inhalt der Datei ohne Rückfrage überschrieben, falls Sie nicht die Option *-i* angegeben haben.

dateiverzeichnis

Name des Dateiverzeichnisses, in das die Kopien eingetragen werden sollen. Es darf nicht das Dateiverzeichnis sein, in dem die Originale stehen.

Ist die Option *-p* nicht gesetzt, so erhalten die Kopien die gleichen Zugriffsrechte wie die Originale sowie die Benutzer- und Gruppennummer des Benutzers, der *cp* aufgerufen hat. Das Änderungsdatum wird jedoch nicht dupliziert, d.h., daß das aktuelle Datum gesetzt wird.

FEHLERMELDUNGEN

`cp: cannot access` *datei*

datei existiert nicht.

`cp: cannot open` *datei*

Sie haben kein Leserecht für *datei*.

`cp: cannot create` *datei*

Sie haben kein Schreibrecht für das Dateiverzeichnis, in dem *datei* angelegt werden soll, bzw. dieses Dateiverzeichnis existiert nicht.

`cp: <`*dvz*`> directory`

dvz ist ein Dateiverzeichnis und kann nicht kopiert werden (Format 1), bzw. Sie haben **-r** nicht gesetzt (Format 2).

BEISPIELE

1. Die Datei *fachliteratur* soll kopiert werden, bevor man sie verändert. Die Kopie soll *fl* heißen und im gleichen Dateiverzeichnis stehen wie *fachliteratur*.

```
$ cp fachliteratur fl
```

2. Alle Dateien aus dem aktuellen Dateiverzeichnis, deren Namen mit *dat* beginnen, sollen ins Dateiverzeichnis */home/do/sicher* kopiert werden.

```
$ cp dat* /home/do/sicher
$ ls -l /home/do/sicher
total 4
-rw------  1  dober  grl        37 Nov 11 11:11  datei1
-rw------  1  dober  grl        97 Apr 01 13:24  datei2
-rw------  1  dober  grl       116 Dec 31 12:13  datei3
-rw------  1  dober  grl       381 Feb 16 08:08  datei4
```

SIEHE AUCH

chmod, *cpio*, *ln*, *mv*, *rm*

cpio
Dateien und Dateiverzeichnisse ein- und auslagern (copy in and out)

Das Kommando *cpio* hat drei Funktionen:
- Es kopiert eine oder mehrere Dateien in eine Archiv-Datei (Format 1)
- Es holt Dateien aus einem zuvor mit *cpio* erstellten Archiv (Format 2)
- Es kopiert Dateien in ein Dateiverzeichnis (Format 3)

cpio␣**-o**[zusatzoption...][einzeloption...] Format 1
cpio␣**-i**[zusatzoption...][einzeloption...][␣muster]... Format 2
cpio␣**-p**[zusatzoption...][einzeloption]␣dvz Format 3

Vorsicht

Es kann zu Verlusten bei der Bandbearbeitung führen, wenn Sie eine Gerätedatei verwenden, die einen Gertätetreiber anspricht, der puffert. *cpio* fordert dann nicht wie sonst am Bandende ein Folgeband an. Sie vermeiden das auf jeden Fall, wenn Sie eine Gerätedatei mit direktem Zugriff (raw device) verwenden (siehe *Tabellen und Verzeichnisse, Gerätedateien für Datenträger*).

Bei der Darstellung der drei Formate werden nur die Hauptoptionen *-o*, *-i* und *-p* beschrieben. Die übrigen Optionen sind im Abschnitt *Zusatz- und Einzeloptionen* erläutert. Bei den jeweiligen Formaten ist angegeben, welche Zusatz- und Einzeloptionen Sie verwenden können.

Format 1: Dateien auslagern

cpio␣**-o**[zusatzoption...][einzeloption...]

zusatzoption
aABcLvV

einzeloption
-C␣puffergröße
-H␣header
-O␣archiv[␣**-M**␣mitteilung]
-R␣benutzernummer

-o
(o - out) *cpio* liest von der Standard-Eingabe eine Reihe von Pfadnamen einfacher Dateien und gibt diese Dateien zusammen mit Status-Informationen in einem speziellen Archiv-Format auf die Standard-Ausgabe aus. Dabei wird die Ausgabe auf ein Vielfaches von 512 byte aufgefüllt. Die Anzahl der Blöcke von 512 byte gibt *cpio* auf die Standard-Fehlerausgabe aus.

Format 2: Dateien einlagern

cpio␣**-i**[zusatzoption...][einzeloption...][␣muster]...

zusatzoption
bBcdfkmrsStuvV6

einzeloption
-C␣puffergröße
-E␣datei
-H␣header
-I␣archiv[␣**-M**␣mitteilung]

-i

(i - in) *cpio* liest von der Standard-Eingabe, die zuvor mit *cpio -o* erzeugt worden sein muß, und holt aus der Eingabe diejenigen Dateien, deren Namen zu *muster* passen. Diese Dateien werden abhängig von den Optionen, die Sie angeben, erzeugt und in den aktuellen Dateiverzeichnis-Baum kopiert. Die so eingelagerten Dateien erhalten die gleichen Zugriffsrechte wie die mit *cpio -o* ausgelagerten Dateien. Benutzer- und Gruppennummer sind diejenigen des Benutzers, der *cpio -i* aufruft. Nur wenn der Systemverwalter *cpio -i* aufruft, erhalten die eingelagerten Dateien dieselbe Benutzer- bzw. Gruppennummer, wie sie die mit *cpio -o* ausgelagerten Dateien hatten. Gerätedateien dürfen nur vom Systemverwalter extrahiert werden.

Die Anzahl der gelesenen Blöcke von 512 byte gibt *cpio* auf die Standard-Fehlerausgabe aus.

muster

Mit *muster* legen Sie fest, welche Dateien aus der Eingabe erzeugt werden. Zur Angabe von *muster* können Sie alle Shell-Sonderzeichen zur Generierung von Dateinamen verwenden (siehe *Tabellen und Verzeichnisse, Sonderzeichen der Bourne-Shell sh*). *muster* darf maximal 1024 Zeichen lang sein.

muster nicht angegeben:
wirkt wie die Angabe eines Sterns *, d.h., alle Dateien werden erzeugt, deren Name nicht mit einem Punkt beginnt.

Format 3: Dateien in ein Dateiverzeichnis kopieren

cpio␣-p[zusatzoption...][einzeloption]␣dvz

zusatzoption
adlLmuvV

einzeloption
-R␣benutzernummer

-p
(p - pass) *cpio* liest von der Standard-Eingabe eine Reihe von Pfadnamen einfacher Dateien, erzeugt die entsprechenden Dateien und kopiert sie abhängig von den Optionen, die Sie angeben, in das Dateiverzeichnis *dvz*. Die Anzahl der kopierten Blöcke von 512 byte gibt *cpio* auf die Standard-Fehlerausgabe aus.

dvz
Name des Zieldateiverzeichnisses, in das die Dateien kopiert werden. Er darf maximal 1024 Zeichen lang sein.

Zusatz- und Einzeloptionen

Die Zusatzoptionen müssen ohne Leerzeichen direkt auf die Hauptoptionen folgen, z.B. *cpio -pdl*.

Geben Sie die Einzeloptionen nach den Zusatzoptionen an. Den Einzeloptionen müssen Sie einen Bindestrich voranstellen. Auf die Einzeloptionen muß das dazugehörige Argument folgen, z.B. *cpio -oBcv -O /dev/rmt/c0s0*.

a
(a - access time) Nach dem Kopieren wird das Zugriffsdatum für die Eingabedateien neu gesetzt.
Wenn auch die Option *l* gesetzt ist, dann wird das Datum für die Dateien, auf die es Verweise gibt, nicht neu gesetzt.
Die Optionen *a* und *m* dürfen Sie nicht gemeinsam verwenden, sie schließen sich gegenseitig aus.

A
(A - append) Verwenden Sie diese Option nur zusammen mit der Einzeloption *-O␣archiv*. Die angegebenen Dateien werden an ein bestehendes Archiv angefügt. Das Archiv muß für diese Operation in einer Datei, auf Diskette(n) oder in einer logischen Platte einer Festplatte stehen. Magnetband- und Kassettenlaufwerke werden nicht unterstützt.

b
(b - bytes) Verwenden Sie diese Option nur mit der Hauptoption *-i*. Die Reihenfolge der Byte in jedem Wort wird umgedreht. Ein Wort entspricht 4 byte.

B

(B - block) Bei der Ein- und Ausgabe wird mit einer Blockgröße von 5120 byte gearbeitet.
Wenn die Zusatzoption *-B* oder die Einzeloption *-C* nicht angegeben sind, wird standardmäßig mit einer Puffergröße von 512 byte gearbeitet. Bei der Ausgabe der abgelegten oder eingelesenen Blöcke geht *cpio* prinzipiell von einer Blockgröße von 512 byte aus. Diese Option ist nur bei Gerätedateien mit direktem Zugriff (raw devices) sinnvoll.
Sie dürfen diese Option nicht zusammen mit der Hauptoption *-p* verwenden.

c

(c - compatible) *cpio* schreibt oder liest die Informationen im Header (Dateivorspann) aus Gründen der Portabilität im US-ASCII-Format (siehe *Tabellen und Verzeichnisse, Zeichensatz ISO 646*).
Diese Zusatzoption (oder die Einzeloption *-H*) sollten Sie immer dann verwenden, wenn Sie Information zwischen Rechnern unterschiedlichen Typs austauschen.
Hinweis:
Ein mit *-c* in Version 5.40 angelegtes Archiv kann nicht von einem Rechner, auf dem eine frühere SINIX-Version installiert ist, eingelesen werden. Verwenden Sie die Einzeloption *-H_odc*, wenn Sie Ihr Archiv auch auf einem Rechner mit einer früheren Version einlesen wollen.
Ein mit *-H_odc* gebildeter Header entspricht einem mit *c* auf einer früheren Version gebildeten Header.
Die Optionen *c*, *6* und *-H* schließen sich gegenseitig aus und dürfen nicht zusammen verwendet werden.

-C_puffergröße

Bei der Ein- und Ausgabe wird mit der Blockgröße *puffergröße* in byte gearbeitet. *puffergröße* muß eine positive ganze Zahl sein. Wenn Sie mit Disketten arbeiten, muß *puffergröße* ein Vielfaches von 512 (Sektorengröße einer Diskette) sein. Standardmäßig wird mit einer Puffergröße von 512 byte gearbeitet, wenn die Einzeloption *-C* oder die Zusatzoption *B* nicht angegeben ist. Diese Option ist nur bei Gerätedateien mit direktem Zugriff (raw devices) sinnvoll.
Sie dürfen diese Option nicht mit der Hauptoption *-p* verwenden.

d

(d - directory) Dateiverzeichnisse werden automatisch angelegt, wenn es notwendig ist.

-E_datei

(E - extract) Bei dieser Einzeloption können Sie den Namen einer Datei angeben, die aus dem Archiv extrahiert werden soll. Sie können nur einen Dateinamen angeben, dürfen aber zusätzlich auch noch ein Muster angeben.

f

Es werden nur Dateien eingelesen, auf die das angegebene *muster* nicht paßt.

-H␣header

(H - header) *cpio* schreibt oder liest Informationen im Dateivorspann (header) im Format *header*. Verwenden Sie prinzipiell entweder *-H* oder die Option *c*, wenn Sie auf Maschinen verschiedenen Typs lesen oder schreiben.

Folgende Formate sind zulässig:

crc	ASCII-Header mit expandierten Gerätenummern und zusätzlicher Prüfsumme pro Datei
ustar	Header und Format im IEEE/P1003 Data Interchange Standard
tar	Header und Format im *tar*-Format
odc	ASCII-Header mit kurzen Gerätenummern

Wenn Sie Daten mit einem Rechner austauschen wollen, auf dem eine frühere Version als SINIX V5.40 installiert ist, verwenden Sie die Einzeloption *-H␣odc*. Ein mit *-H␣odc* in SINIX V5.40 gebildeter Header entspricht einem mit *c* auf früheren SINIX Versionen erzeugten Header. Pfadangaben sind in binären Headern (Voreinstellung) oder in Headern, die durch *-H␣odc* erzeugt wurden, auf maximal 256 Zeichen begrenzt, sonst sind maximal 1024 Zeichen erlaubt.

-I␣archiv

(I - input) Verwenden sie diese Option nur zusammen mit der Hauptoption *-i*.
cpio liest die Eingabe aus *archiv*. Ist
archiv eine Gerätedatei mit direktem Zugriff (raw device), dann können Sie das Band nach vollständigem Lesen auswechseln und anschließend mit ↵ mit dem Folgeband fortfahren.

k

(k - corrupted) Verwenden Sie diese Option nur mit der Hauptoption *-i*.
cpio versucht weiterzuarbeiten, wenn beschädigte Datei-Köpfe oder Ein-/Ausgabe-Fehlermeldungen auftreten. Wollen Sie Informationen von einem teilweise beschädigten Datenträger kopieren, dann können Sie mit Hilfe dieser Option die fehlerfreien Dateien extrahieren. Bearbeitet *cpio* Archive, die weitere Archive im cpio-Format enthalten und tritt ein Fehler auf, so wird meist vorzeitig abgebrochen. *cpio* sucht dann nach dem nächsten korrekten Datei-Kopf, der zu einem kleineren Archiv gehören kann, und beendet sich, wenn es auf das Ende-Zeichen des Archivs stößt.

l

(l - link) Nur mit der Option *-p* verwenden.
Die Dateien werden nicht kopiert, sondern es wird ein Verweis für sie eingetragen. Wenn möglich sollten Sie bei größeren Dateimengen mit Verweisen arbeiten, um Speicherplatz zu sparen, statt Dateien in ein anderes Verzeichnis zu kopieren.

L

(L - symbolic links) Ist diese Option gesetzt, dann verfolgt *cpio* symbolische Verweise (links). Wenn Sie auf einer früheren Version einen symbolischen Verweis kopieren, wird eine normale Datei erzeugt, die den Pfadnamen der vom symbolischen Verweis bezeichneten Datei trägt.
Wenn Sie *cpio* mit der Zusatzoption *L* zusammen mit dem Kommando *find* anwenden, müssen Sie bei *find* den Ausdruck *follow* angeben.

m

(m - modification time) Das alte Änderungsdatum der Datei bleibt erhalten. Diese Option ist für Dateiverzeichnisse unwirksam. Die Option *m* und die Option *a* dürfen Sie nicht gemeinsam verwenden, sie schließen sich gegenseitig sich aus.

-M_mitteilung

(M - message) Wenn Sie diese Option mit *-O* oder *-I* und einer Gerätedatei mit direktem Zugriff (raw device) verwenden, können Sie hier eine Mitteilung definieren, die ausgegeben wird, wenn das Bandende erreicht ist. Geben Sie in *mitteilung* %d an, wird dafür die laufende Nummer des Folgebandes eingesetzt. Sie können auch mehrere Wörter für *mitteilung* angeben, müssen diese allerdings in Anführungszeichen "" einschließen.

-O_archiv

(O - output) Verwenden Sie diese Option nur mit der Hauptoption *-o*.
cpio leitet die Ausgabe nach *archiv* um. Ist *archiv* eine Gerätedatei mit direktem Zugriff (raw device), können Sie bei Bandende ein Folgeband einlegen und mit der Taste ↵ fortfahren.

r

(r - rename) Die Dateien können beim Kopieren interaktiv umbenannt werden. *cpio* fragt bei jeder Datei nach einem neuen Namen. Wenn Sie mit einer Leerzeile antworten (die Taste ↵ drücken), wird die Datei nicht umbenannt.

Die Option *r* funktioniert nicht bei Magnetbändern, Magnetbandkassetten und Exabytekassetten.

-R_benutzernummer

(R - reassign) Diese Option darf nur der Systemverwalter verwenden. *cpio* ersetzt Benutzerkennung und Gruppennummer entsprechend *benutzernummer*.
benutzernummer muß in */etc/passwd* eingetragen sein.

s

(s - swap) Verwenden Sie diese Option nur zusammen mit der Hauptoption *-i*.
In jedem Halbwort werden die Byte vertauscht. Ein Wort entspricht 4 byte.

S

(S - swap) Verwenden Sie diese Option nur mit der Hauptoption *-i*. In jedem Wort werden die Halbworte vertauscht. (Ein Wort entspricht 4 byte.)

t

(t - table of contents) Es wird ein Inhaltsverzeichnis des Datenträgers, von dem die Dateien eingelesen werden sollen ausgegeben. Es werden keine Dateien kopiert.

u

(u - unconditional) Eine vorhandene Datei wird durch eine neu einzulesende Datei mit demselben Pfadnamen ersetzt, auch wenn die vorhandene Datei ein neueres Zugriffsdatum als die neu einzulesende Datei hat.

u nicht angegeben:
Vorhandene Dateien werden nur dann durch neu einzulesende Dateien mit denselben Pfadnamen ersetzt, wenn die neu einzulesenden Dateien neueren Datums als die vorhandenen sind.

v

(v - verbose) Die Namen archivierter bzw. eingelesener Dateien werden ausgegeben. In Kombination mit der Option *t* werden zusätzlich detaillierte Informationen über die betroffenen Dateien ausgegeben.

V

(V - verbose) Für jede gelesene bzw. geschriebene Datei wird ein Punkt ausgegeben, um dem Benutzer ohne nähere Informationen lediglich anzuzeigen, daß *cpio* arbeitet.

6

(6 - 6. Edition) Verwenden Sie diese Option nur mit der Hauptoption *-i*.
cpio arbeitet eine Archivdatei im Format des UNIX-Systems Edition 6 ab. Die Optionen *c*, *-H* und *6* schließen einander aus.

FEHLERMELDUNGEN

```
Error old format can' t support expanded types on <datei>,
Value too long for defined data type
```

Sie haben die Option *-c* nicht angegeben. *cpio* verwaltet aus Kompatibilitätsgründen die Indexnummer (inode) einer zu kopierenden Datei so, daß Indexnummern über 64 Kbyte nur bearbeitet werden können, wenn Sie die Option *-c* angeben.

BEISPIELE

1. Alle Dateien im aktuellen Dateiverzeichnis, deren Namen mit *.f* enden, sollen in eine Archiv-Datei *f.progs* ausgelagert werden:

```
$ find . *.f -print | cpio -o >f.progs
56 blocks
```

find *.*f* durchsucht das aktuelle Verzeichnis nach Dateinamen, die mit *.f* enden und schreibt diese auf Standard-Ausgabe, die durch das Pipe-Zeichen | mit der Standard-Eingabe von *cpio* verbunden ist. *cpio -o* liest die Dateinamen und kopiert die entsprechenden Dateien zusammen mit Status-Informationen auf die Standard-Ausgabe. Durch das Größerzeichen > wird die Standard-Ausgabe in die Datei *f.progs* umgelenkt. Nach der Ausführung gibt *cpio* die Anzahl der kopierten Blöcke auf die Standard-Fehlerausgabe aus.

2. Der Inhalt eines Dateiverzeichnisses mit allen Unterverzeichnissen wird auf eine Magenetbandkassette kopiert:

```
$ find . -print | cpio -oBcv -O /dev/rmt/c0s0
```

find durchsucht das aktuelle Verzeichnis und alle Verzeichnisse, die darunter liegen und gibt die relativen Pfadnamen aller gefundenen Dateien auf die Standard-Ausgabe aus. Durch das Pipe-Zeichen | ist die Standard-Ausgabe von *find* mit der Standard-Eingabe von *cpio* verbunden. *cpio* liest die Namen und kopiert die Dateien auf die Magnetbandkassette, wobei die Verzeichnisstruktur beibehalten wird. Während des Kopiervorgangs werden die Namen der Dateien, die ausgelagert werden, auf die Standard-Ausgabe ausgegeben (Option *v*). Option *B* wird benutzt, um die Puffergröße zu vergrößern; Option *c*, um zu garantieren, daß das Band auch auf Rechnern anderen Typs eingelesen werden kann. Die Option *-c* sollten Sie aus Gründen der Kompatibilität zu anderen Systemen möglichst immer verwenden.

3. Es soll eine Kopie des Dateiverzeichnisses *verz.alt* mit allen Unterdateiverzeichnissen erstellt werden. Die Kopie soll den Namen *verz.neu* enthalten:

```
$ cd verz.alt
$ find . -print | cpio -pdl verz.neu
```

Durch den *cd*-Aufruf wechseln Sie in das Dateiverzeichnis *verz.alt*. *find* durchsucht dann dieses Dateiverzeichnis und alle unter diesem liegenden Dateiverzeichnisse und gibt die Namen aller gefundenen Dateien auf die Standard-Ausgabe aus. Durch das Pipe-Zeichen | ist die Standard-Ausgabe von *find* mit der Standard-Eingabe von *cpio* verbunden. *cpio* liest die Namen und kopiert die Dateien in das Dateiverzeichnis *verz.neu* (*-p verz.neu*). Wenn es erforderlich ist, werden Dateiverzeichnisse angelegt (*-d*), d.h., rekursiv werden auch alle Unterdateiverzeichnisse von *verz.alt* kopiert. Da die Option *-l* angegeben ist, werden Dateien wenn möglich nicht kopiert, sondern es werden Verweise eingerichtet.

SIEHE AUCH

cat, *echo*, *find*, *ls*, *mt*, *tar*, *tapecntl*
ar [19]

crontab
Kommandos regelmäßig zu bestimmten Zeitpunkten ausführen

Mit *crontab* führen Sie Kommandos, Shell-Prozeduren oder ausführbare Programme regelmäßig zu bestimmten Zeitpunkten aus.

Sie können mit *crontab*:

- Kommandoaufträge erteilen (Format 1)
- erteilte Aufträge ausgeben lassen (Format 2)
- erteilte Aufträge löschen (Format 3)
- erteilte Aufträge editieren (Format 4).

Vor dem Aufruf beachten

Wenn die Datei */etc/cron.d/cron.allow* existiert, dann dürfen Sie das Kommando *crontab* nur dann aufrufen, wenn Ihre Benutzerkennung in dieser Datei steht.
Wenn die Datei */etc/cron.d/cron.allow* nicht existiert, dann dürfen Sie das Kommando *crontab* nur dann aufrufen, wenn Ihre Benutzerkennung *nicht* in der Datei */etc/cron.d/cron.deny* steht.
Wenn weder */etc/cron.d/cron.allow* noch */etc/cron.d/cron.deny* existieren, dann darf nur der Systemverwalter *crontab* aufrufen.

Existiert z.B. nur die leere deny-Datei, so dürfen alle Benutzer *crontab* aufrufen.

Die allow/deny-Dateien darf nur der Systemverwalter anlegen und ändern. Sie enthalten pro Zeile eine Benutzerkennung.

Wenn ein privilegierter Benutzer die *crontab*-Datei eines anderen Benutzers verändert, ist das Verhalten nicht vorhersagbar. Der privilegierte Benutzer sollte sich zuerst mit *su* und der Benutzerkennung des anderen Benutzers die Rechte verschaffen, mit denen er wirksame Änderungen an dessen *crontab*-Datei vornehmen kann.

Vorsicht

Wenn Sie *crontab* versehentlich ohne Argumente aufrufen, dürfen Sie das Kommando *nicht* mit CTRL D abbrechen! Denn in diesem Fall würden sämtliche Einträge in Ihrer *crontab*-Datei gelöscht. Benutzen Sie stattdessen DEL.

crontab[␣datei]	Format 1
crontab␣**-l**[␣benutzerkennung]	Format 2
crontab␣**-r**[␣benutzerkennung]	Format 3
crontab␣**-e**[␣benutzerkennung]	Format 4

Format 1: Kommandoaufträge erteilen

crontab[_datei]

Wenn Sie mit *crontab* einen Kommandoauftrag erteilen wollen, geben Sie an:

- das Kommando (bzw. die Shell-Prozedur/das Programm), das ausgeführt werden soll, und
- den Zeitpunkt, zu dem das Kommando ausgeführt werden soll (z.B. jeden Freitag oder jeden 15. Januar).

Rufen Sie *crontab* im obigen Format auf, dann schreibt *crontab* diese Angaben in Ihre *crontab*-Datei */var/spool/cron/crontabs/benutzerkennung*. Der Prozeß */etc/cron* überprüft jede Minute, ob Kommandos in den jeweiligen *crontab*-Datei ausgeführt werden sollen, und veranlaßt ggf. die Ausführung.

Um *kommando* auszuführen, ruft *crontab* von Ihrem Home-Verzeichnis aus eine neue Shell (*sh*) auf. *crontab* richtet zu jeder Shell eine Standardumgebung ein, mit den Variablen *HOME*, *LOGNAME*, *SHELL* (/bin/sh) und *PATH* (:/bin:/usr/bin:/usr/lbin). Wenn Sie Ihre *.profile*-Datei ausführen lassen wollen, müssen Sie dies in Ihrer *crontab*-Datei explizit angeben.

Werden in der *crontab*-Datei die Standard-Ausgabe und Standard-Fehlerausgabe der Kommandos nicht umgelenkt, dann erhalten Sie sowohl die Standard-Ausgabe als auch die Standard-Fehlerausgabe über *mail*.

Pro Benutzerkennung gibt es höchstens eine *crontab*-Datei. Existiert die *crontab*-Datei */var/spool/cron/crontabs/benutzerkennung* noch nicht, so wird sie beim *crontab*-Aufruf neu angelegt. Existiert sie bereits, so wird sie überschrieben. Wenn Sie also mit *crontab* neue Kommandoaufträge erteilen wollen, dann verwenden Sie die Option *-e*.

datei
: Name der Datei, die die auszuführenden Kommandos und die Zeitpunkte der Ausführung enthält.
datei muß dem unten beschriebenen Aufbau entsprechen (siehe *Aufbau einer crontab-Datei*).

  datei nicht angegeben:
  crontab liest den Eingabetext von der Standard-Eingabe. Der Text, den Sie in diesem Fall eingeben, muß dem unten beschriebenen Aufbau entsprechen (siehe *Aufbau einer crontab-Datei*).

Format 2: Erteilte Aufträge ausgeben

crontab␣-l[␣benutzerkennung]

-l

(l - list) *crontab* gibt den Inhalt Ihrer *crontab*-Datei */var/spool/cron/crontabs/benutzerkennung* aus.
Ist diese Datei nicht vorhanden, erhalten Sie eine Fehlermeldung (siehe *Fehlermeldungen*).

benutzerkennung

Ist *benutzerkennung* angegeben, wird die *crontab*-Datei des zugehörigen Benutzers gezeigt. Die Angabe von *benutzerkennung* ist jedoch nur dem Systemverwalter erlaubt.

Format 3: Erteilte Aufträge löschen

crontab␣-r[␣benutzerkennung]

-r

(r - remove) *crontab* löscht Ihre *crontab*-Datei */var/spool/cron/crontabs/benutzerkennung*.

benutzerkennung

Ist *benutzerkennung* angegeben, wird die *crontab*-Datei des zugehörigen Benutzers gelöscht. Die Angabe von *benutzerkennung* ist jedoch nur dem Systemverwalter erlaubt.

Format 4: Erteilte Aufträge editieren

crontab␣-e[␣benutzerkennung]

-e

(e - edit) Mit dieser Option können Sie eine Kopie Ihrer *crontab*-Datei oder, falls diese nicht existiert, eine leere Datei editieren und als aktuelle *crontab*-Datei speichern. Die Umgebungsvariable *VISUAL* legt fest, welcher Editor aufgerufen wird. Ist *VISUAL* nicht definiert, so wird die Umgebungsvariable *EDITOR* überprüft. Ist auch *EDITOR* nicht definiert, so wird *ed* aufgerufen.

benutzerkennung

Ist *benutzerkennung* angegeben, wird die *crontab*-Datei des zugehörigen Benutzers bearbeitet. Die Angabe von *benutzerkennung* ist jedoch nur dem Systemverwalter erlaubt.

Aufbau einer crontab-Datei

Eine *crontab*-Datei bzw. der Eingabetext, den Sie beim *crontab*-Aufruf übergeben (siehe *Format 1*), muß folgendermaßen aufgebaut sein:

Der Text besteht

- aus Zeilen, die einen Kommandoauftrag enthalten
- evt. aus Kommentarzeilen.

Leerzeilen sind nicht erlaubt.

Zeilen mit einem Kommandoauftrag

Die Zeilen, die einen Kommandoauftrag enthalten, bestehen jeweils aus sechs Feldern, die durch Leer- oder Tabulatorzeichen voneinander getrennt sind. Die Felder enthalten folgende Angaben:

1	2	3	4	5	6
Minute	Stunde	Tag im Monat	Monat	Wochentag	Kommando

Die Felder 1 bis 5 legen den Zeitpunkt fest, zu denen das in Feld 6 angegebene Kommando ausgeführt werden soll.

Minute	Mögliche Angaben: 0-59 oder *muster*
Stunde	Mögliche Angaben: 0-23 oder *muster*
Tag im Monat	Mögliche Angaben: 1-31 oder *muster*
Monat	Mögliche Angaben: 1-12 oder *muster*
Wochentag	Mögliche Angaben: 0-6 (0=Sonntag) oder *muster*
Kommando	Name des Kommandos, das zu dem angegebenen Zeitpunkt ausgeführt werden soll. Mit Hilfe des Prozentzeichens % stellen Sie dem Kommando die Standard-Eingabe zur Verfügung. Die Shell interpretiert jedes Prozentzeichen im Kommandofeld, das nicht durch einen Gegenschrägstrich \ entwertet ist, als Neue-Zeile-Zeichen. Sie führt den Inhalt des Kommandofeldes nur bis zum ersten nicht entwerteten Prozentzeichen oder Neue-Zeile-Zeichen aus; der Rest des Feldes wird dem Kommando als Standard-Eingabe übergeben (siehe *Beispiel 3*).

muster kann sein:

- ein Stern * (steht für alle zulässigen Werte)
- ein Bereich *zahl-zahl*
- eine Liste von zulässigen Zahlen oder Bereichen; die Elemente der Liste sind durch Kommas getrennt.
  Beispiel: 1,3,5 oder 1-3,5

Angabe des Tages

Für die Angabe des Tages stehen zwei Felder zur Verfügung: das 3. Feld (Tag im Monat) und das 5. Feld (Wochentag).

Wenn Sie das 3. Feld und zugleich das 5. Feld mit einer Zahl, einem Bereich oder einer Liste besetzen, sind beide Angaben unabhängig voneinander gültig.

Zum Beispiel bewirkt der Eintrag

0 0 1,15 3 1 kommando

in der *crontab*-Datei, daß *kommando* sowohl am 1. und 15. März als auch an jedem Montag im März aufgerufen wird.

Wenn Sie für die Angabe des Tages nur ein Feld besetzen wollen, müssen Sie in das andere Feld einen Stern schreiben.
Zum Beispiel bewirkt der Eintrag

0 0 * 3 1 kommando

in der *crontab*-Datei, daß *kommando* nur an jedem Montag im März aufgerufen wird.

Kommentarzeilen

Eine Kommentarzeile enthält in Spalte 1 das Nummernzeichen #. Darauf kann beliebiger Text folgen. Kommentarzeilen können Sie verwenden, um die Kommandoaufträge zu erläutern oder um die Aufträge zur besseren Übersicht in Abschnitte zu gliedern. Beachten Sie, daß der Eingabetext, den Sie an *crontab* übergeben, beliebig viele Kommentarzeilen, aber keine Leerzeilen enthalten darf.

ENDE-STATUS

0 *crontab* wurde erfolgreich ausgeführt
≠0 Bei der Ausführung von *crontab* ist ein Fehler aufgetreten

FEHLERMELDUNGEN

```
crontab: you are not authorized to use cron. Sorry.
```

Sie sind nicht berechtigt, *crontab* aufzurufen. Siehe *Vor dem Aufruf beachten*.

Format 1

```
crontab: can't open your crontab file.
```

Sie haben *crontab* mit dem Namen einer Datei aufgerufen, die nicht existiert oder auf die nicht zugegriffen werden kann.

```
crontab: error in previous line; ...
```

Hat eine Zeile des Eingabetextes, den Sie an *crontab* übergeben haben, nicht den richtigen Aufbau, dann gibt *crontab* diese Zeile aus, gefolgt von obiger Fehlermeldung. Anstelle der drei Punkte steht die genauere Beschreibung des Fehlers, z.B.

```
unexpected character found in line
```

Ungültiges Zeichen in der Zeile gefunden

```
number out of bounds
```

Eine angegebene Zahl liegt außerhalb der zulässigen Grenzen.

Format 2

```
crontab: can't open your crontab file.
```

Ihre *crontab*-Datei */var/spool/cron/crontabs/benutzerkennung* existiert nicht. Sie können sie also auch nicht mit *crontab -l* lesen.

DATEIEN

/etc/cron.d/cron.allow
: Liste der Benutzerkennungen mit Ausführrecht für *crontab*. In jeder Zeile steht jeweils eine Benutzerkennung.

/etc/cron.d/cron.deny
: Liste der Benutzerkennungen ohne Ausführrecht für *crontab*. In jeder Zeile steht jeweils eine Benutzerkennung.

/var/spool/cron/crontabs/benutzerkennung
: *crontab*-Datei des Benutzers *benutzerkennung*.

UMGEBUNGSVARIABLEN

VISUAL
Die Umgebungsvariable *VISUAL* legt bei Verwendung der Option *-e* fest, welcher Editor benutzt wird.

EDITOR
Falls VISUAL nicht gesetzt ist, legt *EDITOR* bei der Verwendung der Option *-e* fest, welcher Editor benutzt wird.

BEISPIELE

1. Der Rechner soll Sie jedes Jahr am 15. Mai an den Geburtstag von Tante Emma erinnern. Dazu legen Sie eine Datei *geburtstag* mit folgendem Inhalt an:

```
0 0 15 5 * echo Tante Emma hat heute Geburtstag !!!
```

   Beachten Sie, daß Sie nur für den Tag im Monat eine Angabe machen wollen, nicht für den Wochentag. Deshalb müssen Sie in das Feld für den Wochentag einen Stern schreiben.

   Nun übergeben Sie Ihre *crontab*-Datei *geburtstag*:

```
$ crontab geburtstag
```

   Sie erhalten dann jeden 15. Mai um 0 Uhr die betreffende Nachricht über *mail*. Das funktioniert natürlich nur dann, wenn der Rechner in dieser Nacht nicht abgeschaltet ist.

2. Sie möchten eine Kollegin und sich selbst jeden Montag um 13 Uhr 30 an die Skigymnastik am Abend erinnern. Die Nachricht soll auf den Bildschirm ausgegeben werden. Sie sitzen am Bildschirm tty007, Ihre Kollegin am Bildschirm tty014. Sie können in den Eingabetext Kommentarzeilen einfügen, damit Sie ihn später mit *crontab -l* besser lesen können. Ihre *crontab*-Datei könnte dann z.B. so aussehen:

```
# *    = alle zulaessigen Werte
# n-n = Bereich
# n,n = Liste
# Wochentag: 0 = Sonntag
#
# Min   Std   Tag(Mon)   Mon   Tag(Wo)   Kommando
#
  30    13       *         *     1       echo Heute abend Skigymnastik ! >/dev/tty007
  30    13       *         *     1       echo Heute abend Skigymnastik ! >/dev/tty014
```

   Ihre Kollegin und Sie selbst erhalten dann jeden Montag um 13 Uhr 30 die Meldung "Heute abend Skigymnastik !" auf den Bildschirm (und nicht in den Postkasten).

3. Sie möchten der Benutzerin *doro*, die am Rechner *brummbox* arbeitet, jeden Montag mit *mail* die folgende Nachricht schicken:

```
Guten Morgen,
schoene Woche...
```

   Dazu schreiben Sie den folgenden Eintrag in Ihre *crontab*-Datei:

```
0 0 * * 1 mail doro@brummbox %Guten Morgen, %schoene Woche...
```

SIEHE AUCH

at, *atq*, *atrm*, *mail*, *sh*, *su*, *vi*
cron [7]

crypt
Text verschlüsseln und entschlüsseln

crypt ver- und entschlüsselt den Inhalt von Dateien. Es liest von der Standard-Eingabe und schreibt auf die Standard-Ausgabe.
Dateien, die ohne Verwendung der Option *-c* verschlüsselt wurden, sind kompatibel zu Dateien, die mit den Editoren *ed*, *edit*, *ex* oder *vi* im Verschlüsselungsmodus erstellt oder bearbeitet wurden.

crypt[_-c][_schlüssel]

-c
(c - compatible crypt) Ist die Option *-c* gesetzt, dann können
- Dateien, die auf Rechnern der NSC-Linie verschlüsselt wurden, auf Rechnern der Intel-Linie entschlüsselt werden
- Dateien, die auf Rechnern der Intel-Linie verschlüsselt wurden, auf Rechnern der NSC-Linie entschlüsselt werden

schlüssel
Als Schlüssel können Sie selbst eine Zeichenkette (1 - 8 Zeichen) bestimmen, mit der Sie den Modus zur Ver- und Entschlüsselung festlegen.

schlüssel nicht angegeben:
Sie geben den gewünschten Verschlüsselungs-Mechanismus über die Datensichtstation ein. Die Eingabe erscheint nicht auf dem Bildschirm.

Sicherheitskriterien

Die Datensicherheit von verschlüsselten Dateien beruht grundsätzlich auf folgenden drei Faktoren:

- der zugrundeliegende Verschlüsselungs-Mechanismus darf nicht erkennbar sein.
- die zur Kodierung eingegebene Tastenfolge darf keiner direkten Suche zugänglich sein.
- "Schleichwege", auf denen dennoch die Kodierung oder sogar der Klartext erkennbar ist, müssen eine Seltenheit bleiben.

Vorsicht

Wenn Sie zwei oder mehrere Dateien, die mit demselben Schlüssel kodiert wurden, zu einer Datei zusammenfassen, und anschließend versuchen, diese zu dekodieren, wird nur der Inhalt der ursprünglich ersten Datei richtig entschlüsselt.

Wenn Sie Ihren Schlüssel als Argument des Kommandos *crypt* übergeben, können andere Benutzer des Systems möglicherweise Ihren Schlüsseltext erkennen, indem sie das *ps* oder ähnlich funktionierende Kommandos ausführen. Die Auswahl des Schlüsseltextes und die Kodier-Sicherheit sind die Schwachstellen von *crypt*.

Wenn Sie die Ausgabe nach *nroff* pipen, den Schlüssel aber nicht in die Kommandozeile geschrieben haben, dürfen Sie *crypt* nicht durch eine Pipe mit dem Kommando *pg* verbinden, da Kommandos wie *pg* die Bildschirm-Voreinstellungen verändern.

DATEI

/dev/tty
wenn das Schüsselwort über die Standard-Eingabe angegeben wurde.

BEISPIEL

Die Datei *Omibrief* enthält folgende geheime Botschaft:

```
Liebe Großmutter,
heute möchte ich eine Geheimbotschaft an Dich schicken, damit der böse
Wolf nicht erfährt, daß ich morgen mit Wein und Kuchen zu
Dir in den Wald komme.
Gruß Dein Rotkäppi
```

Sie muß nun unbedingt verschlüsselt werden! Hierzu verwendet Rotkäppi die Tastenfolge *abcd*.

Verschlüsseln und umlenken in die *Geheimschrift-Datei*:

```
$ crypt abcd < Omibrief > Geheimschrift-Datei ↵
```

```
$ cat Geheimschrift-Datei ↵
```

Die *Geheimschrift-Datei* enthält nur unleserliche Sonderzeichen und Buchstabenfolgen!

Großmutter kann die geheimnisvolle Datei nur entschlüsseln, weil ihr der Jäger im Auftrag von Rotkäppi das Schlüsselwort, also die Tastenfolge *abcd*, mitgeteilt hat:

Entschlüsseln und ausgeben auf dem Bildschirm:

```
$ crypt < Geheimschrift-Datei
Schlüssel eingeben: abcd ↵
Liebe Großmutter,
heute möchte ich eine Geheimbotschaft an Dich schicken, damit der böse
Wolf nicht erfährt, daß ich morgen mit Wein und Kuchen zu
Dir in den Wald komme.
Gruß Dein Rotkäppi
```

Wenn Rotkäppi die Botschaft auf einem Rechner der NSC-Linie verschlüsselt hat, Großmutter aber einen Rechner der Intel-Linie besitzt, dann gibt Großmutter folgendes ein:

```
$ crypt -c < Geheimschrift-Datei
```

SIEHE AUCH

ed, *edit*, *ex*, *makekey*, *pg*, *ps*, *stty*, *vi*
nroff [13]

csh
C-Shell

Die C-Shell *csh* ist wie die Standard-(Bourne-)Shell *sh* ein Kommando-Interpreter und bietet ebenso wie *sh* auch die wesentlichen Elemente einer Programmiersprache. Bei der C-Shell ist die Syntax dieser Programmiersprache der Syntax der Programmiersprache C sehr ähnlich (daher auch der Name).

Wie die Bourne-Shell kennt auch die C-Shell Ein-/Ausgabe-Umlenkung und Variablen, ersetzt an gewünschten Stellen Kommandos durch ihre Ausgabe, ersetzt Muster durch passende Dateinamen und anderes mehr. Im interaktiven Gebrauch bietet die C-Shell mehr Möglichkeiten als die Bourne-Shell, z.B.

- Vervollständigung von Dateinamen und Benutzerkennungen
- History-Ersetzung
- Alias-Namen für Komandos
- Auftragssteuerung (Job Control).

Darüberhinaus hat die C-Shell eine Reihe von eingebauten Kommandos, die in der Bourne-Shell nicht enthalten sind.

csh[_option]...[_datei[_argument]...]

option

-b
(b - break) Argumente, die nach dieser Option stehen, werden nicht als Optionen für *csh* interpretiert. Damit können Sie beim *csh*-Aufruf Optionen an eine Shell-Prozedur übergeben, ohne daß sie als Optionen für *csh* interpretiert werden.

-c
Zusammen mit dieser Option müssen Sie für *datei* den Namen einer Shell-Prozedur angeben. Alle auf *datei* folgenden *argumente* werden als Argumente für *datei* angesehen und in der Variablen *argv* gespeichert. Es wird eine Subshell erzeugt, die die Shell-Prozedur ausführt.

-e
(e - exit) *csh* wird sofort beendet, wenn ein Kommando nicht ordnungsgemäß beendet wird oder einen Ende-Status ungleich 0 zurückgibt.

-f
(f - fast start) Beim Start wird weder die Datei *.cshrc* noch die Datei *.login* gelesen.

-f nicht angegeben:
Beim Start wird die Datei *.cshrc* gelesen; wird *csh* als Login-Shell gestartet, wird außerdem die Datei *.login* gelesen.

-i

(i - interactive) *csh* gibt das Bereitzeichen für Kommando-Eingaben aus, selbst wenn die Standard-Eingabe anscheinend keine Datensichtstation ist.

-n

(n - no execution) *csh* liest und interpretiert alle Kommandos, führt sie aber nicht aus.
Diese Option ist nützlich, um C-Shell-Prozeduren auf Syntax-Fehler zu überprüfen.

-s

(s - standard input) *csh* liest die Kommandos von der Standard-Eingabe.

-t

csh liest nur eine einzige Kommandozeile und führt sie aus. Ist die Kommandozeile, die Sie eingeben wollen, länger als eine Eingabezeile, so können Sie das Neue-Zeile-Zeichen mit einem Gegenschrägstrich \ entwerten.

-v

(v - verbose) Die vordefinierte Variable *verbose* wird gesetzt. Das bewirkt, daß *csh* jede Kommandoeingabe auf die Standard-Fehlerausgabe schreibt, und zwar nach der History-Ersetzung, vor anderen Ersetzungen und vor der Kommando-Ausführung.

-V

Die Variable *verbose* wird gesetzt, bevor die Datei *.cshrc* gelesen wird.

-x

Die vordefinierte Variable *echo* wird gesetzt. Das bewirkt, daß *csh* jede Kommandoeingabe auf die Standard-Fehlerausgabe schreibt, und zwar nach allen Ersetzungen, direkt vor der Kommandoausführung.

-X

Die Variable *echo* wird gesetzt, bevor die Datei *.cshrc* gelesen wird.

datei[␣argument]...

datei[␣*argument*]... können Sie nur dann angeben, wenn Sie keine der Optionen *-i*, *-s*, *-t* angeben.
datei ist der Name des Kommandos oder der Shell-Prozedur, die ausgeführt werden soll. *datei* wird als 0-tes Argument übergeben.

argument

Argument für das Kommando bzw. die Shell-Prozedur *datei*. Sie können mehrere Argumente angeben. Die Argumente werden der Variablen *argv* zugewiesen.

datei[␣*argument*]... nicht angegeben:
csh liest die Kommandos von der Standard-Eingabe.

FUNKTIONSWEISE UND GEBRAUCH DER C-SHELL

Dieser Abschnitt ist in folgende Unterabschnitte gegliedert:

- Starten und Beenden der C-Shell
  - Die Dateien *.cshrc, .login* und *.logout*
- Interaktiver Ablauf
  - Bereitzeichen der C-Shell
  - Wie bearbeitet die C-Shell die Kommandozeile?
- Nicht-interaktiver Ablauf
- Vervollständigung von Dateinamen und Benutzerkennungen
- Lexikalische Analyse
  - Aufteilung der Kommandozeile in Worte
- Parsen der Kommando-Eingabezeile
  - Einfache Kommandos
  - Pipelines
  - Pipelines verknüpfen
  - Pipelines klammern
  - Kommandos bzw. Pipelines, die im Hintergrund ablaufen
- History-Ersetzung
- Alias-Namen für Komandos
- Umlenkung der Ein-/Ausgabe
- Variablen-Ersetzung
  - Variablen ausgeben, definieren und löschen
  - Variablen, die die Shell benutzt
  - Auf den Wert einer Variablen zugreifen
  - Entwertungszeichen und Variablen-Ersetzung
  - Zeitpunkt der Variablen-Ersetzung
  - Shell-Parameter
- Kommandos durch ihre Ausgabe ersetzen
- Muster durch passende Dateinamen ersetzen
- Ausdrücke und Operatoren
  - Klammer-Operator
  - Arithmetische Operatoren
  - Logische Operatoren
  - Vergleichsoperatoren
  - Dateieigenschaften abfragen
  - Abfragen, ob ein Kommando erfolgreich ausgeführt wurde

- Ablaufsteuerung
  - Schleifen und Verzweigungen
- Kommandoausführung
  - Wie sucht die C-Shell nach einem Kommando?
  - Wie wird das Kommando ausgeführt?
- Signalbehandlung
- Auftragssteuerung (Job Control)
  - Auftrag und Auftragsnummer
  - Wie können Sie den Status von Aufträgen ändern?
  - Wie bezeichnen Sie einen bestimmten Auftrag?
  - Ein-/Ausgabe bei Hintergrundaufträgen
  - Wie meldet die C-Shell Status-Änderungen von Aufträgen?
- Eingebaute C-Shell-Kommandos
  - Die eingebauten Kommandos *:, alias, bg, cd, chdir, dirs, echo, eval, exec, exit, fg, glob, hashstat, history, jobs, kill, limit, login, logout, nice, nohup, notify, onintr, popd, pushd, rehash, repeat, set, setenv, shift, source, stop, suspend, time, umask, unalias, unhash, unlimit, unset, unsetenv, wait,* % und @.
  - Ablaufanweisungen
- Umgebungsvariablen und vordefinierte Shell-Variablen
- C-Shell-Prozeduren
  - Was ist eine C-Shell-Prozedur?
  - C-Shell-Prozeduren starten
  - Kommentare
  - Welches Programm führt die Prozedur aus?

Starten und Beenden der C-Shell

Wenn die C-Shell gestartet wird, führt sie normalerweise (siehe Option *-f*) die Kommandos aus, die in der Datei *.cshrc* in Ihrem HOME-Dateiverzeichnis stehen. Dazu muß diese Datei lesbar sein, und Sie müssen entweder der Eigentümer der Datei sein, oder Ihre reale Gruppennummer muß mit der Gruppennummer der Datei übereinstimmen. Wird die C-Shell mit einem Namen gestartet, der mit einem Bindestrich - beginnt, läuft sie als Login-Shell ab (z.B. wenn sie von *login* aufgerufen wird). In diesem Fall führt die C-Shell zunächst die Kommandos in der Datei *.cshrc* aus und dann zusätzlich die Kommandos in der Datei *.login* in Ihrem HOME-Dateiverzeichnis. Für die *.login*-Datei benötigen Sie dieselben Rechte wie für die *.cshrc*-Datei. Die *.login*-Datei enthält in der Regel Kommandos, die den Terminaltyp und die Umgebung festlegen.

Wenn eine Login-C-Shell beendet wird, führt sie die Kommandos aus, die in der Datei *.logout* in Ihrem HOME-Dateiverzeichnis stehen. Hierfür benötigen Sie wieder dieselben Rechte wie für die *.cshrc*-Datei.

Interaktiver Ablauf

Nach erfolgreichem Start gibt eine interaktive C-Shell das Bereitzeichen *rechnername%␣* (für normale Benutzer) bzw. *rechnername#␣* (für den Systemverwalter) aus. Das Bereitzeichen können Sie mit Hilfe der Variablen *prompt* ändern.
Nach Ausgabe des Bereitzeichens liest die Shell die Kommandos, die über die Datensichtstation eingegeben werden. Die Shell bearbeitet jede Eingabezeile wie folgt:

1. Die Zeile wird in Worte aufgeteilt (siehe *Lexikalische Analyse*).
2. Diese Wortfolge wird in die History-Liste gesetzt (siehe *History-Ersetzung*).
3. Die Wortfolge wird geparst (siehe *Parsen der Kommando-Eingabezeile*).
4. Alias-Namen werden ersetzt (siehe *Alias-Namen für Kommandos*).
5. History-Befehle werden aufgelöst (siehe *History-Ersetzung*).
6. Variablen werden ersetzt (siehe *Variablen-Ersetzung*).
7. Muster für Dateinamen werden ersetzt (siehe *Muster durch passende Dateinamen ersetzen*).
8. Die Ein-/Ausgabe wird ggf. umgelenkt (siehe *Umlenkung der Ein-/Ausgabe*).
9. Kommandos in Gegenhochkommata `...` werden durch ihre Ausgabe ersetzt (siehe *Kommandos durch ihre Ausgabe ersetzen*).
10. Die Shell führt die Kommandos in der Eingabezeile aus (siehe *Kommandoausführung*).

Nicht-interaktiver Ablauf

Wenn die C-Shell nicht-interaktiv abläuft, gibt sie kein Bereitzeichen aus. Eine nicht-interaktive C-Shell führt entweder ein Kommando aus, das beim *csh*-Aufruf als Argument mitgegeben wurde, oder führt die Kommandos einer Shell-Prozedur aus.

Vervollständigung von Dateinamen und Benutzerkennungen

Wenn die Variable *filec* gesetzt ist, kann eine interaktive C-Shell einen nur teilweise eingegebenen Dateinamen oder eine nur teilweise eingegebene Benutzerkennung vervollständigen:

- Ist ein teilweise eingegebener Dateiname eindeutig und folgt ihm ein ESC-Zeichen, dann fügt die Shell die fehlenden Zeichen an und vervollständigt ihn so zu einem passenden Dateinamen des aktuellen Dateiverzeichnisses.
- Folgt einem teilweise eingegebenen Dateinamen ein EOF-Zeichen (CTRL d bzw. END), listet die Shell alle passenden Dateinamen des aktuellen Dateiverzeichnisses auf. Anschließend gibt sie wieder das Bereitzeichen zusammen mit der unvollständigen Kommandozeile aus, damit Sie einen der aufgelisteten Dateinamen eingeben können.
- Wenn das letzte Wort der Kommandozeile nur teilweise eingegeben ist, wenn es mit einer Tilde ~ beginnt und wenn ihm ein ESC-Zeichen folgt, dann versucht die Shell dieses Wort zu ~*benutzerkennung* zu vervollständigen, wobei *benutzerkennung* eine existierende Benutzerkennung ist (siehe *Muster durch passende Dateinamen ersetzen*).
  Folgt dem teilweise eingegebenen Wort ein EOF-Zeichen (CTRL d bzw. END), listet die Shell alle passenden Benutzerkennungen auf. Anschließend gibt sie wieder das Bereitzeichen zusammen mit der unvollständigen Kommandozeile aus, damit Sie eine der aufgelisteten Kennungen eingeben können.

Treten dabei Fehler auf oder passen zu einer unvollständigen Eingabe mehrere Dateinamen bzw. Benutzerkennungen, so ertönt die Terminalglocke. Dies können Sie verhindern, indem Sie die Variable *nobeep* setzen.

Sie können außerdem Dateinamen mit bestimmten Suffixen ausschließen, indem Sie diese Suffixe in der Variablen *fignore* auflisten. Wenn jedoch der unvollständige Dateiname nur zu einem Dateinamen mit einem solchen Suffix vervollständigt werden kann (es sonst also keine andere Möglichkeit gibt), dann wird dieser Dateiname nicht ignoriert. Der Inhalt der Variablen *fignore* hat keinen Einfluß auf die Liste der Dateinamen, die bei einem EOF-Zeichen ausgegeben wird.

Siehe *Beispiel 1* und *2* am Ende der C-Shell-Beschreibung.

Lexikalische Analyse

Die C-Shell teilt jede Eingabezeile nach den folgenden Regeln in Worte auf:

- Worttrenner sind normalerweise Leer- und Tabulatorzeichen.
- Die Sonderzeichen &, |, ;, <, >, (und) bilden jeweils ein eigenes Wort; treten diese Zeichen als Paar auf (z.B. || oder >> oder |&), bildet ein solches Paar ein Wort. Sind diese Zeichen mit einem vorangestellten Gegenschrägstrich \ entwertet, können sie Teil eines anderen Wortes sein (z.B. ist a\|b ein Wort, a|b sind drei Worte).
- Ein Neue-Zeile-Zeichen, dem ein Gegenschrägstrich vorangestellt ist, hat die gleiche Bedeutung wie ein Leerzeichen.
- Eine Zeichenkette, die in Hochkommata '...', Anführungszeichen "..." oder Gegenhochkommata `...` eingeschlossen ist, bildet ein Teilwort. Sonderzeichen (einschließlich Leer- und Tabulatorzeichen), die in einer solchen Zeichenkette vorkommen, bilden kein eigenes Wort. Ein Neue-Zeile-Zeichen, das innerhalb von Hochkommata '...', Anführungszeichen "..." oder Gegenhochkommata `...` steht und dem ein Gegenschrägstrich vorangestellt ist, wirkt wie ein Neue-Zeile-Zeichen (nicht wie ein Leerzeichen).
- Wenn die Shell nicht von der Datensichtstation liest, leitet das Nummernzeichen # einen Kommentar ein; der Kommentar endet mit der Eingabezeile. Die Shell ignoriert alle Zeichen, die zum Kommentar gehören (siehe *C-Shell-Prozeduren*).
  Ist dem Nummernzeichen ein Gegenschrägstrich \ vorangestellt oder steht es innerhalb von Hochkommata '...', Anführungszeichen "..." oder Gegenhochkommata `...`, verliert es seine Sonderbedeutung als Kommentarzeichen.

Beispiel

Die beiden Kommandos

```
who>/tmp/whos_on
who>>/tmp/whos_on
```

bestehen aus je drei Worten: *who*, > bzw. >> und */tmp/whos_on*. Das Kommando

```
"who am I"
```

besteht aus nur einem Wort; die C-Shell interpretiert dieses Wort *who am I* als Namen eines Programms, das ausgeführt werden soll.

Parsen der Kommando-Eingabezeile

Für die C-Shell ist die Kommando-Eingabezeile folgendermaßen aufgebaut:

- Ein *einfaches* Kommando besteht aus einer Folge von Worten. Das erste Wort (das nicht Teil einer Ein-/Ausgabe-Umlenkung ist) bezeichnet das Kommando, das ausgeführt werden soll. Alle weiteren Worte, die keine Sonderbedeutung für die C-Shell haben, werden dem auszuführenden Kommando als Argumente übergeben.
- Ein einfaches Komando oder eine Folge von einfachen Kommandos, die durch | oder |& voneinander getrennt sind, bilden eine *Pipeline*. Dabei bedeutet

  kommando1 | kommando2
  Die Standard-Ausgabe von *kommando1* wird in die Standard-Eingabe von *kommando2* umgelenkt.

  kommando1 |&kommando2
  Die Standard-Ausgabe und die Standard-Fehler-Ausgabe von *kommando1* werden beide in die Standard-Eingabe von *kommando2* umgelenkt.
- Mehrere Pipelines können durch einen Strichpunkt, durch && oder durch || miteinander verbunden werden. Dabei bedeutet

  pipeline1;pipeline2
  Die beiden Pipelines werden hintereinander ausgeführt.

  pipeline1&&pipeline2
  Zuerst wird *pipeline1* ausgeführt. *pipeline2* wird nur dann ausgeführt, wenn *pipeline1* erfolgreich ausgeführt wurde.

  pipeline1 | |pipeline2
  Zuerst wird *pipeline1* ausgeführt. *pipeline2* wird nur dann ausgeführt, wenn *pipeline1* nicht erfolgreich ausgeführt wurde.
- Eine Pipeline oder eine Folge von Pipelines kann in runde Klammern (...) eingeschlossen werden. Dadurch entsteht ein einfaches Kommando, das wieder Teil einer Pipeline oder Folge von Pipelines sein kann.
- Ist an eine Folge von Pipelines das Zeichen & angefügt, wird sie asynchron, d.h. im Hintergrund, ausgeführt. In diesem Fall wartet die Shell nicht, bis die Folge ausgeführt ist, sondern gibt die Auftragsnummer, die zugehörigen Prozeßnummern und dann sofort das Bereitzeichen aus (siehe *Auftragssteuerung*).

History-Ersetzung

Mit Hilfe der History-Ersetzung können Sie früher eingegebene Kommandozeilen nochmals ausführen lassen oder Worte aus früher eingegebenen Kommandozeilen in der aktuellen Kommandozeile verwenden. Damit können Sie Schreibfehler leicht korrigieren und komplizierte Kommandos schnell wiederholen.
Die C-Shell speichert die eingegebenen Kommandozeilen in der History-Liste. Die Größe dieser Liste wird durch den Wert der Variablen *history* bestimmt. Das zuletzt eingegebene Kommando wird auf jeden Fall in der Liste gespeichert.
Eine früher eingegebene Kommandozeile, die in der History-Liste gespeichert ist, wird *Ereignis* genannt. Jedes Ereignis erhält eine Ereignisnummer. Die Nummern werden in aufsteigender Reihenfolge vergeben.
History-Befehle können Sie überall in einer Kommandozeile eingeben; Sie können History-Befehle jedoch nicht schachteln.
Jeder History-Befehl bezieht sich auf ein bestimmtes Ereignis (siehe *Ereignis-Bezeichner*). Um einen History-Befehl auszuführen, durchsucht die Shell die History-Liste rückwärts vom letzten bis zum ersten Ereignis. Hat sie das gewünschte Ereignis gefunden, wählt sie ggf. bestimmte Worte aus (siehe *Wort-Bezeichner*) und ändert diese ggf. ab (siehe *Parameter*). Die so entstehende Zeichenkette setzt sie anstelle des History-Befehls in die aktuelle Kommandozeile ein. Die Kommandozeile, die sich dadurch ergibt, wird dann in die History-Liste gesetzt und außerdem auf den Bildschirm ausgegeben. Dann erst führt die Shell ggf. weitere Ersetzungen durch (z.B. Variablen-Ersetzungen, Ersetzung von Mustern durch passende Dateinamen usw.) und führt das Kommando aus. Siehe *Beispiele 3* bis *7* am Ende der C-Shell-Beschreibung.

Ein History-Befehl hat das folgende Format:

```
![Ereignis-Bezeichner][:Wort-Bezeichner][:Parameter]...
```

!

startet die History-Ersetzung, es sei denn, dem Ausrufezeichen folgt ein Leerzeichen, Tabulatorzeichen, Neue-Zeile-Zeichen, Gleichheitszeichen = oder eine öffnende runde Klammer (.
Mit der Variablen *histchars* können Sie statt ! ein anderes Zeichen für den Beginn eines History-Befehls definieren. Mit einem vorangestellten Gegenschrägstrich heben Sie die Sonderbedeutung dieses Zeichens auf.
In speziellen Fällen können History-Befehle auch mit einem Dach ^ beginnen, siehe *Kurzschreibweise*.

Ereignis-Bezeichner

Mit einem Ereignis-Bezeichner bezeichnen Sie eine bestimmte Kommandozeile (Ereignis) in der History-Liste. Folgende Ereignis-Bezeichner sind zulässig:

!
bezeichnet das vorhergehende Ereignis.
Beispiel: Der History-Befehl !! wiederholt die vorhergehende Kommandozeile.

n
bezeichnet das Ereignis mit der Nummer *n*.

-n
bezeichnet das aktuelle Ereignis minus *n*.

str
bezeichnet das letzte Ereignis, das mit *str* beginnt.

?str[?]
bezeichnet das letzte Ereignis, das *str* enthält.
Das Trennzeichen ?, das auf *str* folgt, dürfen Sie weglassen, wenn unmittelbar auf *str* ein Neue-Zeile-Zeichen folgt.

Fehlt der Ereignis-Bezeichner, so bezeichnet der History-Befehl

- entweder die zuletzt eingegebene Kommandozeile
- oder, falls unmittelbar vorher bereits ein History-Befehl eingegeben wurde, die Kommandozeile, auf die sich dieser History-Befehl bezog.

Der Ereignis-Bezeichner darf jedoch nur dann fehlen, wenn nach dem Ausrufezeichen, mit dem der History-Befehl beginnt, kein Leer-, Tabulator-, Neue-Zeile- oder Gleichheitszeichen und keine öffnende runde Klammer folgt.

Beispiel

!:1 !$!:3:s/d/dat/ sind gültige History-Befehle.
! ist kein gültiger History-Befehl. Wollen Sie die zuletzt eingegebene Kommandozeile nochmals ausführen, verwenden Sie den History-Befehl !!.

Wort-Bezeichner

Auf einen Ereignis-Bezeichner kann, getrennt durch einen Doppelpunkt, ein Wort-Bezeichner folgen. Der Doppelpunkt darf fehlen, wenn der Wort-Bezeichner mit ^, $, *, - oder % beginnt.
Mit einem Wort-Bezeichner wählen Sie ein oder mehrere Worte in der Kommandozeile aus. Folgende Wort-Bezeichner sind zulässig:

#
die ganze bisher eingegebene Kommandozeile

0
das erste Wort (d.h. der Kommandoname)

n
das *n*-te Argument

^
das erste Argument (entspricht 1)

$
das letzte Argument

%
das Wort, das beim letzten *?s*-Suchlauf gefunden wurde

n-m
n-tes bis *m*-tes Argument einschließlich

-m
Abkürzung für 0-*m*

n*
Abkürzung für *n*-$

n-
wie *n**, aber ohne das letzte Argument

*
alle Argumente oder nichts, wenn das Ereignis nur ein einziges Wort enthält

Parameter

Auf den Ereignis-Bezeichner (bzw. den Wort-Bezeichner, falls vorhanden) können ein oder mehrere Parameter folgen. Die Parameter sind voneinander und vom Ereignis- bzw. Wort-Bezeichner durch einen Doppelpunkt zu trennen (Ausnahme: Parameter *g*).

Vorsicht

Parameter können bei der Ersetzung von Kommandos durch ihre Ausgabe nicht auf die Ausgabe angewendet werden.

Wenn Sie Parameter im Zusammenhang mit Variablen-Ersetzung verwenden, gibt es zwei Einschränkungen: Es sind nicht alle Parameter verfügbar, und pro Ersetzung ist nur ein einziger Parameter erlaubt.

Folgende Parameter sind verfügbar:

h

(h - head) löscht die letzte Komponente des Pfadnamens.

t

(t - tail) löscht alle führenden Komponenten des Pfadnamens, so daß nur die letzte Komponente übrigbleibt.

r

(r - root) löscht ein Suffix der Form *.xxx* im Dateinamen.

e

löscht den ganzen Pfadnamen außer das Suffix.

s/l/r[/]

(s - substitute) ersetzt die Zeichenkette *l* durch *r*. Ist der Parameter *g* nicht angegeben, wird die Änderung nur auf die erste zu *l* passende Zeichenkette angewandt. Paßt keine Zeichenkette, wird ein Fehler gemeldet.
Als Trennzeichen können Sie statt des Schrägstrichs / auch jedes andere Zeichen verwenden; das Zeichen wir dadurch als Trennzeichen definiert, daß es unmittelbar auf *s* folgt. Wollen Sie das Trennzeichen auch in den Zeichenketten *l* oder *r* verwenden, müssen Sie es mit einem Gegenschrägstrich \ entwerten.
Ein kommerzielles Und & in der Ersetzungszeichenkette *r* ist ein Sonderzeichen: Es wird durch die Zeichenkette *l* ersetzt. Mit einem Gegenschrägstrich \ können Sie das Sonderzeichen & entwerten.
Ist die Zeichenkette *l* leer, wird für *l* entweder die Zeichenkette *l* des letzten Kommandos *s* oder die Zeichenkette *str* des letzten Kommandos *!?str* genommen.
Das Trennzeichen, das auf *r* folgt, dürfen Sie weglassen, wenn unmittelbar auf *r* ein Neue-Zeile-Zeichen folgt.

&
wiederholt die letzte Ersetzung.

g
(g - global) Diesen Parameter geben Sie unmittelbar vor einem der Parameter *h, t, r, e, s, &* an (z.B. *gr* oder *gs* oder *g&*). Die Änderung wird dann auf die erste passende Zeichenkette in *jedem* Wort angewandt. (Vorsicht: Sie wird *nicht* auf alle passenden Zeichenketten angewandt!)

p
(p - print) Das durch die History-Ersetzung entstehende Kommando wird in die History-Liste gesetzt und auf den Bildschirm ausgegeben, aber nicht ausgeführt.

q
(q - quote) Die ersetzten Worte werden entwertet, so daß weitere Ersetzungen verhindert werden.

x
wie *q*, nur gelten Leer-, Tabulator- und Neue-Zeile-Zeichen als Worttrenner.

Klammern {...}

Mit geschweiften Klammern {...} grenzen Sie einen History-Befehl von nachfolgenden Zeichen, die nicht zu dem Befehl gehören, ab:

!{...}zeichenkette

Siehe *Beispiel 6* am Ende der C-Shell-Beschreibung.

Kurzschreibweise

^l^r[^]
Dies ist die Kurzschreibweise für den History-Befehl !:s^*l*^*r*[^] (ersetze in der vorherigen Kommandozeile bzw. in der Kommandozeile, die der letzte History-Befehl bezeichnet hat, die Zeichenkette *l* durch die Zeichenkette *r*). Siehe *Beispiel 7* am Ende der C-Shell-Beschreibung.
Mit der Variablen *histchars* können Sie statt ^ ein anderes Zeichen für diese Kurzschreibweise definieren.

Alias-Namen für Kommandos

Mit Hilfe von Alias-Namen können Sie oft benutzte Kommandofolgen abgekürzt eingeben.
Einen Alias-Namen definieren Sie mit dem eingebauten C-Shell-Kommando *alias*:

alias␣alias-name␣alias-definition

Die C-Shell trägt den Alias-Namen und die zugehörige Alias-Definition in eine Liste ein. Mit dem Kommando *alias* (ohne Argumente) können Sie sich die Liste ausgeben lassen. Mit *unalias* löschen Sie Einträge aus der Liste. (Siehe *Beispiel 8* am Ende der C-Shell-Beschreibung.)

Argumente können Sie in der Alias-Definition folgendermaßen angeben:

- Konstante Argumente geben Sie direkt an.
  Beispiel: `alias cpr cc -c prog.c`
  Der Aufruf *cpr* ergibt dann *cc -c prog.c*.

- Für Argumente, die erst in der Kommando-Eingabezeile angegeben werden sollen, geben Sie in der Alias-Definition Platzhalter in Form von History-Befehlen an. Dabei müssen Sie jedem History-Befehl einen Gegenschrägstrich \ voranstellen, damit die Shell den Befehl nicht schon bei der Ausführung des *alias*-Kommandos, sondern erst bei der Auflösung des Alias-Namens ersetzt.
  Beispiel: `alias gpw grep \!^ /etc/passwd`
  Der Aufruf *gpw anna* ergibt dann *grep anna /etc/passwd*.

  Sollen solche variablen Argumente nicht innerhalb der Alias-Definition, sondern danach eingesetzt werden, brauchen Sie keinen History-Befehl anzugeben.
  Beispiel: `alias lld ls -ld`
  Der Aufruf *lld dvz* ergibt dann *ls -ld dvz*.

Die Shell löst Alias-Namen in Kommandozeilen folgendermaßen auf: Nachdem sie die Kommandozeile in einzelne Kommandos zerlegt hat, prüft sie in jedem Kommando das erste Wort. Ist dieses Wort ein gültiger Alias-Name, liest die Shell die zugehörige Alias-Definition. Enthält die Alias-Definition History-Befehle, ersetzt die Shell diese Befehle, wobei sie die ursprüngliche Kommandozeile mit dem Alias-Namen als zuletzt eingegebenes Kommando interpretiert. Enthält die Alias-Definition keine History-Befehle, fügt die Shell die Argumentliste der ursprünglichen Kommandozeile an die Alias-Definition an. Siehe *Beispiel 8* und *9* am Ende der C-Shell-Beschreibung.

Alias-Einträge können Sie schachteln, d.h. eine Alias-Definition kann einen anderen Alias-Namen enthalten. Solche Schachtelungen werden ausgewertet, bevor History-Ersetzungen ausgeführt werden. Dies ist nützlich bei Pipelines wie z.B.

```
alias lm 'ls -l \!* | more'
```

(Siehe *Beispiel 10* am Ende der C-Shell-Beschreibung.)

Enthält die Alias-Definition zum Alias-Namen *name1* einen anderen Alias-Namen *name2*, so darf in der Definition zu *name2* der Alias-Name *name1* nicht vorkommen, da dies bei der Alias-Ersetzung eine Schleife ergeben würde. Die Shell entdeckt solche Schleifen und meldet einen Fehler.

Eine Alias-Definition darf auch den "eigenen" Alias-Namen enthalten, aber nur als erstes Wort. Dies ergibt keine Schleife, da die Shell in diesem Fall nach der ersten Alias-Ersetzung aufhört und nicht nach weiteren Alias-Namen sucht (siehe *Beispiel 11* am Ende der C-Shell-Beschreibung).

Umlenkung der Ein-/Ausgabe

Mit den folgenden Ausdrücken können Sie die Standard-Eingabe, Standard-Ausgabe und Standard-Fehlerausgabe umlenken. Für Dateinamen, die auf diese Ausdrücke folgen, führt die Shell die Variablen-Ersetzung, die Ersetzung von Kommandos durch ihre Ausgabe sowie die Ersetzung von Mustern durch passende Dateinamen getrennt vom Rest des Kommandos durch. Für die Zeichenkette *wort* (siehe < <*wort*) führt sie diese Ersetzungen nicht durch.
Wenn Sie die Standard-Ausgabe und die Standard-Fehlerausgabe getrennt umlenken wollen, müssen Sie wie folgt eine Subshell aufrufen:

```
(kommando > ausgabedatei) >& fehlerdatei
```

<
Die Standard-Eingabe wird umgelenkt.

< <wort
Die Standard-Eingabe wird bis zu einer Zeile, die genau aus *wort* besteht, gelesen. Die gelesenen Zeilen werden in einer Temporärdatei abgelegt. Ist *wort* nicht mit Gegenschrägstrich oder Anführungszeichen entwertet, bearbeitet die Shell dann diese Zeilen: Sie ersetzt Variablen durch ihren Wert und Kommandos, die in Gegenhochkommata eingeschlossen sind, durch ihre Ausgabe. Anschließend erhält das Kommando, für das die Standard-Eingabe umgelenkt wurde, die Temporärdatei als Standard-Eingabe.

> >! >& >&!
Die Standard-Ausgabe wird in eine Datei umgelenkt. Existiert die Datei noch nicht, wird sie angelegt. Existiert sie bereits, wird ihr Inhalt überschrieben; der bisherige Inhalt ist damit verloren.
Ist die Variable *noclobber* gesetzt, verhindert sie, daß existierende Dateien gelöscht werden. Die Variable verhindert außerdem die Umlenkung auf eine Datensichtstation oder in den "Papierkorb" */dev/null*, es sei denn, Sie verwenden die Ausdrücke mit Ausrufezeichen. Mit den &-Ausdrücken wird sowohl die Standard-Ausgabe als auch die Standard-Fehlerausgabe in die Datei umgelenkt.

>> >>! >>& >>&!
Diese Ausdrücke wirken wie die obigen mit einfachem >, nur wird die Standard-Ausgabe an die Datei angehängt. Existiert die Datei bereits, wird ihr Inhalt also nicht überschrieben.
Ist die Variable *noclobber* gesetzt, meldet die Shell einen Fehler, wenn die Datei noch nicht existiert, es sei denn, Sie verwenden die Ausdrücke mit Ausrufezeichen. Mit den &-Ausdrücken wird sowohl die Standard-Ausgabe als auch die Standard-Fehlerausgabe an die Datei angehängt.

Variablen-Ersetzung

Variablen der C-Shell bestehen aus einem Namen und einem Wert.
Der Name einer Variablen ist eine Zeichenkette, die aus Buchstaben, Ziffern und Unterstrich _ bestehen kann. Der Name kann bis zu 20 Zeichen enthalten; das erste Zeichen muß entweder ein Buchstabe oder ein Unterstrich sein.
Der Wert einer Variablen ist eine Liste von null, ein oder mehreren Worten, die durch Leer- oder Tabulatorzeichen voneinander getrennt sind. Wenn Sie bei der Variablendefinition die Wortliste in runde Klammern eingeschlossen haben, können Sie mit $*var*[*index*] bzw. ${*var*[*index*]} auf die einzelnen Listenelemente zugreifen (siehe *Shell-Parameter* und *Beispiel 12* am Ende der C-Shell-Beschreibung).

Die C-Shell unterscheidet zwischen Umgebungsvariablen und Shell-Variablen: Umgebungsvariablen werden automatisch exportiert, d.h. aufgerufenen Prozessen bekanntgemacht, Shell-Variablen nicht (siehe *Umgebungsvariablen und vordefinierte Shell-Variablen*). Was im vorliegenden Abschnitt *Variablen-Ersetzung* gesagt wird, gilt für beide Arten von Variablen, sofern nicht explizit zwischen beiden unterschieden wird.

Variablen ausgeben, definieren und löschen

Mit dem eingebauten C-Shell-Kommando *set* können Sie sich bereits definierte Variablen ausgeben lassen und neue Variablen definieren. Mit dem Kommando *unset* löschen Sie Variablen.

Variablen, die die C-Shell benutzt

Einige Variablen sind vordefiniert und werden von der C-Shell verwaltet und benutzt. Zum Beispiel enthält die Variable *argv* die Argumentliste der Shell.
Bei einigen Variablen, die die Shell benutzt, ist der Wert, den sie haben, bedeutungslos; von Bedeutung ist nur, ob sie definiert sind oder nicht. Eine solche Variable ist z.B. die Variable *verbose*: Ist ihr irgendein Wert zugewiesen (es kann auch ein leerer Wert sein), wird jedes eingegebene Kommando nach der History-Ersetzung auf den Bildschirm ausgegeben.
Die vordefinierten Variablen sind in *Umgebungsvariablen und vordefinierte Shell-Variablen* beschrieben.

Auf den Wert einer Variablen zugreifen

Den Wert einer Variablen erhalten Sie, indem Sie dem Variablen-Namen ein Dollarzeichen $ voranstellen:

$name

Der Ausdruck *$name* ist ein Shell-Parameter. Die Shell ersetzt einen Parameter in der Kommandozeile durch den Wert, den die entsprechende Variable aktuell hat. Sie können auch bestimmte Worte des Variablenwerts auswählen oder sich andere Informationen über die Variable ausgeben lassen (siehe *Shell-Parameter*).
Folgt auf den Namen eines Shell-Parameters ein Buchstabe, eine Ziffer oder ein Unterstrich _, interpretiert die Shell dieses Zeichen als Teil des Namens. Mit geschweiften Klammern können Sie den Shell-Parameter von anderen Zeichen des Eingabewortes abgrenzen.

Beispiel

Mit dem Kommando

```
echo $datei
```

können Sie sich den Wert der Variablen *datei* (z.B. */usr/tabellen/liste*) ausgeben lassen. Mit folgendem Kommando hängen Sie an den Dateinamen die Endung 00 an:

```
mv $datei ${datei}00
```

Ohne die geschweiften Klammern würde die Shell nach einer Variablen *datei00* suchen, die vielleicht gar nicht definiert ist.

Sprechen Sie eine Variable an, die nicht definiert ist, meldet die Shell normalerweise einen Fehler.

Mit numerischen Variablenwerten können Sie wie mit Zahlen rechnen. Führen Sie numerische Operationen aus, wird ein leerer Variablenwert als 0 interpretiert. Besteht der Wert einer Variablen aus mehreren Worten, so wird bei solchen Operationen nur das erste Wort berücksichtigt, die übrigen Worte werden ignoriert.

Entwertungszeichen und Variablen-Ersetzung

Das Dollarzeichen $ können Sie entwerten, indem Sie ihm einen Gegenschrägstrich \ voranstellen. Die Variable wird dann nicht ersetzt, es sei denn, der zugehörige Shell-Parameter steht innerhalb von Anführungszeichen "...": Innerhalb von Anführungszeichen werden Variablen immer ersetzt. Innerhalb von Hochkommata '...' werden Variablen nicht ersetzt. Das Dollarzeichen wird auch dann entwertet, wenn ihm ein Leer-, Tabulator- oder Neue-Zeile-Zeichen folgt.

Ist ein Shell-Parameter in Anführungszeichen "..." eingeschlossen, dann wird er durch den Wert ersetzt. Enthält der Wert Sonderzeichen, so werden diese so behandelt, als seien sie in Anführungszeichen eingeschlossen. Besteht der Wert aus mehreren Worten, erhält man eine Zeichenkette, die Leerzeichen enthält. Z.B. entspricht die Angabe "*$name*" der Angabe "*Anna Mueller*", wenn *Anna Mueller* der Wert der Variablen *name* ist.
Anders ist es, wenn bei der History-Ersetzung der History-Parameter *q* angegeben wurde: Besteht der Variablenwert aus mehreren Worten, dann wird der Shell-Parameter nicht durch eine einzige Zeichenkette ersetzt, sondern durch eine Liste von Worten, die durch Leerzeichen voneinander getrennt sind und die jedes für sich entwertet sind, um Kommando- und Muster-Ersetzung zu verhindern.

Zeitpunkt der Variablen-Ersetzung

Variablen werden normalerweise durch ihren Wert ersetzt, nachdem die Eingabezeile analysiert, Alias-Namen aufgelöst und Ein-/Ausgabe-Umlenkungen bearbeitet sind. Anders verhält es sich, wenn

- Variablen bei Ein-/Ausgabe-Umlenkungen im Dateinamen oder bei der Umlenkung *<<wort* in den Eingabezeilen vorkommen: Hier werden die Variablen durch ihren Wert ersetzt, während die Umlenkung stattfindet.
- Variablen in Kommandos vorkommen, die in Gegenhochkommata `...` eingeschlossen sind (siehe *Kommandos durch ihre Ausgabe ersetzen*).

Erst nachdem die Variablen durch ihren Wert ersetzt sind, ersetzt die Shell Kommandos in Gegenhochkommata `...` durch ihre Ausgabe und Muster durch passende Dateinamen. In folgenden Fällen führt die Shell diese Ersetzungen nicht durch:

- wenn das Kommando bzw. das Muster innerhalb von Anführungszeichen "..." steht
- wenn die Variable *noglob* gesetzt ist (verhindert nur die Ersetzung von Mustern durch passende Dateinamen)
- wenn bei der History-Ersetzung für den betreffenden Text der History-Parameter *q* angegeben wurde.

Shell-Parameter

Ein Shell-Parameter wird durch das Dollarzeichen $ eingeleitet. Die Shell ersetzt einen Parameter in der Kommandozeile durch den Wert, den die entsprechende Variable aktuell hat.

Im folgenden sind die Shell-Parameter beschrieben, die die C-Shell kennt.

$var
${var}
Diese Shell-Parameter werden durch den Wert der Variablen *var* ersetzt. Besteht der Wert aus mehreren Worten, ist das Ergebnis eine Liste von Worten, die durch je ein Leerzeichen voneinander getrennt sind.
Ist *var* eine Umgebungsvariable (siehe *Umgebungsvariablen und vordefinierte Shell-Variablen*), wird ihr Wert zurückgegeben; History-Parameter dürfen Sie auf Umgebungsvariablen nicht anwenden.

$var[index]
${var[index]}
Diese Shell-Parameter wählen bestimmte Worte aus dem Variablenwert aus. Der Variablenwert muß bei der Variablendefinition in runde Klammern eingeschlossen worden sein (siehe das eingebaute C-Shell-Kommando *set*).
index kann entweder selbst eine Zahl, ein Zahlenbereich *zahl-zahl* oder ein Stern * sein oder nach Auswertung eine Zahl, einen Zahlenbereich oder einen Stern ergeben. Die Worte des Variablenwerts werden, bei 1 beginnend, durchnumeriert; der Stern bezeichnet alle Worte. Gibt *index* eine Zahl an, die größer als die Anzahl der Worte ist, aus denen der Variablenwert besteht, meldet die Shell einen Fehler. Zahlenbereiche können Sie abgekürzt angeben:

-zahl	ist die Abkürzung für 1-*zahl*
zahl-	ist die Abkürzung für *zahl-anzahl*, wobei *anzahl* die Anzahl der Worte ist, aus denen der Variablenwert besteht.

Liegt das zweite Argument einer Bereichsangabe im zulässigen Intervall 1 bis $*#var* oder fehlt es, dann ist es kein Fehler, wenn der Bereich leer ist.

$#name
${#name}
Diese Shell-Parameter liefern die Anzahl der Worte, aus denen der Variablenwert besteht.

$0
Der Shell-Parameter *$0* liefert den Namen der Datei, aus der die Kommandoeingabe gelesen wird. Ist der Name dieser Datei unbekannt, liefert *$0* einen Fehler.

$n
${n}
äquivalent zu *$argv[n]* (siehe die vordefinierte Shell-Variable *argv*); *n* ist eine Zahl größer gleich 1.

$*
äquivalent zu *$argv[*]* (siehe die vordefinierte Shell-Variable *argv*).

$?var
${?var}
liefert die Zeichenkette 1, falls die Variable *var* gesetzt ist, sonst 0.

$?0
liefert 1, falls der Name der aktuellen Eingabedatei bekannt ist, sonst 0.

$$
liefert die Prozeßnummer der Vater-Shell.

$<
liefert eine Zeile von der Standard-Eingabe, ohne daß nachfolgende Auswertungen durchgeführt werden. Damit können Sie von einer C-Shell-Prozedur aus von der Tastatur lesen.

Auf folgende der oben angeführten Shell-Parameter können Sie bei der History-Ersetzung die History-Parameter *e, h, q, r, t* sowie *gh, gt* und *gr* anwenden. Bei Shell-Parametern mit geschweiften Klammern {...} müssen Sie die History-Parameter innerhalb der Klammern schreiben.

$*var* bzw. ${*var*}
$*var*[*index*] bzw. ${*var*[*index*]}
$*#name* bzw. ${*#name*}
$0
$*n* bzw. ${*n*}
$*

Kommandos durch ihre Ausgabe ersetzen

Ein Kommando, das in Gegenhochkommata `...` eingeschlossen ist, wird von einer Subshell ausgeführt. Die Standard-Ausgabe des Kommandos wird dann in Worte zerlegt; Worttrenner sind Leer-, Tabulator- und Neue-Zeile-Zeichen, leere Worte werden ignoriert. Innerhalb von Anführungszeichen "..." wirkt nur das Neue-Zeile-Zeichen als Worttrenner, Leer- und Tabulatorzeichen bleiben erhalten.
Der entstehende Text wird an die Stelle der Zeichenkette in Gegenhochkommata gesetzt.
Das letzte Neue-Zeile-Zeichen wird in jedem Fall ignoriert. Damit ist es möglich, daß für das Kommando in Gegenhochkommata nur ein Wortteil eingesetzt wird, auch wenn die Standard-Ausgabe des Kommandos eine vollständige Zeile geliefert hat.

csh wendet die Ersetzung von Kommandos in Gegenhochkommata `...` durch ihre Ausgabe und die Ersetzung von Mustern durch passende Dateinamen getrennt auf die Argumente von eingebauten Kommandos an. Dabei werden Teilausdrücke, die nicht ausgewertet wurden, auch nicht verändert; nur auf ausgewertete Ausdrücke werden die Ersetzungen angewandt.
Bei nicht-eingebauten Kommandos führt *csh* die Ersetzung von Dateinamen-Mustern im Kommandonamen getrennt von der Ersetzung von Dateinamen-Mustern in der Argumentliste durch. Die Ersetzung findet in einer Subshell statt und erst, nachdem Ein-/Ausgabe-Umlenkungen bearbeitet worden sind.

Muster durch passende Dateinamen ersetzen

Worte,

- die eines der Zeichen * ? [{ enthalten oder
- die mit einer Tilde ~ beginnen,

betrachtet die Shell als Muster. Diese Muster wertet sie folgendermaßen aus (siehe auch *Beispiel 13* am Ende der C-Shell-Beschreibung):

- Muster, die * ? [enthalten, ersetzt die Shell durch eine alphabetisch sortierte Liste von Dateinamen, die zu dem Muster passen. Die Dateinamen gehören zu existierenden Dateien. Die Shell meldet einen Fehler, wenn zu einem solchen Muster kein Dateiname paßt.
- Das Muster {...} dient als Abkürzung für eine Liste von Zeichenketten. Die Zeichenketten müssen *nicht* Namen von existierenden Dateien sein. Die Liste wird nicht sortiert.
- Die Muster ~ und ~*benutzerkennung* sind eine Abkürzung für ein HOME-Dateiverzeichnis.

Die Sonderzeichen haben folgende Bedeutung:

*****
paßt zu null, einem oder mehreren beliebigen Zeichen.

?
paßt zu einem beliebigen Zeichen.

[s]
paßt zu einem der Zeichen, die in der Zeichenkette *s* enthalten sind.

[c1-c2]
paßt zu einem der Zeichen, die im Zeichenbereich *c1-c2* gemäß der ASCII-Sortierreihenfolge enthalten sind.

{s1,s2,...}
paßt zu einer der Zeichenketten (oder einem der Dateinamen-Muster) *s1, s2* usw. Anders als bei den Mustern *, ?, [*s*] und [*c1-c2*] wird das Ergebnis der Auswertung von {*s1,s2,...*} nicht sortiert.

~
Dies ist eine Abkürzung für Ihr HOME-Dateiverzeichnis, so wie es in der Variablen *home* festgelegt ist.

~benutzerkennung
Dies ist eine Abkürzung für das HOME-Dateiverzeichnis des angegebenen Benutzers, so wie es in der Kennwortdatei */etc/passwd* festgelegt ist.

Stehen die oben angegebenen Sonderzeichen innerhalb von Hochkommata '...' oder Anführungszeichen "..." oder ist ihnen ein Gegenschrägstrich \ vorangestellt, dann haben die Zeichen keine Sonderbedeutung, sondern stehen für sich selbst.

Ein Pfadname mit einer Komponente, die mit einem Punkt beginnt (z.B. */usr/anna/.cshrc*), paßt nur dann zu einem Muster, wenn der Punkt im Muster explizit als Punkt angegeben ist. Schrägstriche müssen im Muster ebenso explizit angegeben sein.

Eine einzelne öffnende { oder schließende } geschweifte Klammer oder ein leeres geschweiftes Klammernpaar {} wird nicht ersetzt, sondern bleibt unverändert.

Weitere Informationen zur Ersetzung von Dateinamen-Mustern finden Sie bei *Kommandos durch ihre Ausgabe ersetzen.*

Ausdrücke und Operatoren

Eine Reihe von eingebauten C-Shell-Kommandos verarbeiten Ausdrücke, deren Operatoren den Operatoren der Programmiersprache C ähnlich sind und denselben Vorrangregeln genügen. Solche Ausdrücke werden vor allem bei den Kommandos @, *exit*, *if* und *while* verwendet; oft dienen sie dazu, den Ablauf von Kommandos zu steuern (siehe *Ablaufsteuerung*).

Die einzelnen Komponenten eines Ausdrucks werden durch Leer- oder Tabulatorzeichen getrennt. Für fehlende Werte wird 0 eingesetzt. Das Ergebnis eines Ausdrucks ist immer eine Zeichenkette; diese Zeichenkette kann auch aus einer Dezimalzahl bestehen.

Im folgenden sind die Operatoren beschrieben. Sie sind nach ihrem Vorrang in absteigender Reihenfolge geordnet; Operatoren in derselben Zeile haben gleichen Vorrang.

(...)
Zusammenfassung zu einer Gruppe

!
Logische Negation

*** / %**
Multiplikation, Division, Modulo-Operator (Rest bei ganzzahliger Division).
Diese Operatoren sind rechts-assoziativ; dies kann zu unerwarteten Ergebnissen führen! Im Zweifelsfall sollten Sie die Ausdrücke klammern.

+ -
Addition, Subtraktion.
Auch diese Operatoren sind rechts-assoziativ.

<< >>
Bitweise Verschiebung nach links bzw. rechts

< > <= >=
Kleiner, größer, kleiner oder gleich, größer oder gleich

== != =~ !~
Gleichheit (==), Ungleichheit (!=), Mustervergleich für die Ersetzung durch passende Dateinamen (=~ und !~). Diese Operatoren vergleichen ihre Argumente als Zeichenketten.
Die Operatoren =~ und !~ bedeuten: Die Zeichenkette auf der linken Seite paßt bzw. paßt nicht zu dem Dateinamen-Muster auf der rechten Seite. Diese beiden Operatoren machen in Shell-Prozeduren *switch*-Anweisungen für derartige Mustervergleiche überflüssig.

&
Bitweises UND (AND)

^
Bitweises EXKLUSIV-ODER (XOR)

|
Bitweises INKLUSIV-ODER (OR)

&&
Logisches UND (AND)

||
Logisches ODER (OR)

Die Operatoren == != =~ !~ vergleichen ihre Argumente als Zeichenketten; die übrigen Operatoren arbeiten mit Zahlen.

Darüberhinaus gibt es die folgenden Abfragen zu Dateien:

-r datei
liefert "wahr" (d.h. 1) zurück, falls Sie Leserecht für *datei* besitzen; andernfalls wird "falsch" (d.h. 0) zurückgeliefert.

-w datei
wahr, wenn Sie Schreibrecht für *datei* besitzen

-x datei
wahr, wenn Sie Ausführrecht (bzw. bei Dateiverzeichnissen das Recht zum Durchlaufen) für *datei* besitzen

-e datei
wahr, wenn *datei* existiert

-o datei
wahr, wenn Ihnen die Datei gehört

-z datei
wahr, wenn *datei* die Länge 0 hat (d.h. leer ist)

-f datei
wahr, wenn *datei* eine einfache Datei ist

-d datei
wahr, wenn *datei* ein Dateiverzeichnis ist.

Existiert *datei* nicht oder kann auf sie nicht zugegriffen werden, liefern alle diese Abfragen "falsch" zurück.

Zusätzlich gibt es eine Abfrage zur erfolgreichen Beendigung eines Kommandos:

{␣kommando␣**}**
Läuft *kommando* erfolgreich ab, liefert der Ausdruck "wahr" (d.h. 1) zurück; andernfalls wird "falsch" (d.h. 0) zurückgeliefert.
Beachten Sie, daß Kommandos bei erfolgreichem Ablauf normalerweise 0 als Ende-Status zurückliefern und bei Auftreten von Problemen irgendeinen anderen Wert.
Wollen Sie den Status bei Beendigung eines Kommandos direkt abprüfen, so verwenden Sie statt dieses Ausdrucks lieber die Variable *status*.

Ablaufsteuerung

Die Shell enthält eine Reihe von eingebauten Kommandos, mit denen Sie den Ablauf von Kommandos in Shell-Prozeduren und, in eingeschränktem Maße, auch vom Terminal aus steuern können. Diese Kommandos veranlassen die Shell entweder, Schleifen auszuführen (z.B. *foreach*, *while*), oder unter bestimmten Bedingungen einen Teil der Eingabe zu überspringen (Verzweigung, z.B. *switch*, *if*). Siehe *Eingebaute C-Shell-Kommandos*, *Ablaufanweisungen*.

Ist die Eingabe für die Shell nicht an beliebiger Stelle abrufbar und liest die Shell eine Schleife, wird diese Eingabe gepuffert. Die Shell kann bestimmte Positionen im Puffer anspringen und die Eingabe wiederholt lesen, so wie es die Schleife vorsieht. Soweit es die Pufferung erlaubt, können auch *goto*-Kommandos, die rückwärts springen, erfolgreich ausgeführt werden.

Kommandoausführung

Ist das Kommando in die C-Shell eingebaut, führt sie es direkt aus. Andernfalls sucht die Shell nach einer Datei, die den angegebenen Namen hat und Ausführrecht besitzt. Enthält der Kommandoname einen Schrägstrich /, interpretiert die Shell den Namen als Pfadnamen und sucht an der entsprechenden Stelle. Enthält der Kommandoname keinen Schrägstrich, durchsucht die Shell normalerweise ihre Hash-Tabelle, um die Datei zu finden. Die Hash-Tabelle ist eine interne Tabelle der Shell. Sie enthält die Pfadnamen aller ausführbaren Dateien der in der Variablen *path* aufgelisteten Dateiverzeichnisse. Das aktuelle Dateiverzeichnis bildet hierbei eine Ausnahme: Die dort enthaltenen Dateien stehen nicht in der Hash-Tabelle, sondern werden gesondert geprüft. Steht die Datei weder in der Hash-Tabelle noch im aktuellen Dateiverzeichnis, gibt die Shell eine Meldung aus, daß die Datei nicht gefunden wurde.
Eine interaktive C-Shell legt die Hash-Tabelle normalerweise in folgenden Fällen neu an:

- nachdem sie die Datei *.cshrc* gelesen hat
- nachdem die Variable *path* neu gesetzt worden ist
- wenn Sie das eingebaute C-Shell-Kommando *rehash* aufrufen.

Das Kommando *rehash* sollten Sie immer dann aufrufen, wenn sich der Inhalt der Dateiverzeichnisse, die in *path* eingetragen sind, geändert hat (z.B. wenn ein neues Kommando hinzugekommen ist).
Die Hash-Tabelle dient dazu, die Suche nach der auszuführenden Datei zu beschleunigen. Mit den *csh*-Optionen *-c* und *-t* oder mit dem eingebauten C-Shell-Kommando *unhash* können Sie die Verwendung der Hash-Tabelle verhindern. In diesem Fall fragt die Shell die Variable *path* ab und sucht in allen dort angegebenen Dateiverzeichnissen nach der auszuführenden Datei.

Spezialfall

Wenn das auszuführende Kommando eine Shell-Prozedur ist, der angegebene Name der Prozedur keinen Schrägstrich enthält und das Wort *shell* ein Alias-Name ist, dann wird das Ergebnis dieser spät stattfindenden Alias-Ersetzung ohne Änderungen der Kommandozeile vorangestellt. Das erste Wort der Alias-Definition sollte ein vollständiger Pfadname sein. Das System versucht, dieses erste Wort als Kommando auszuführen. Enthält die Alias-Definition weitere Worte, so werden diese zusammen mit dem Text der Eingabezeile als Argumente behandelt.

Besitzt die gefundene Datei Ausführrechte, erzeugt die Shell mit *fork()* einen neuen Prozeß und übergibt ihn (über den Systemaufruf *execve()*) zusammen mit den Argumenten an den Systemkern. Der Systemkern versucht dann, den neuen Prozeß mit dem gewünschten Programm zu überlagern.
Ist die Datei eine ausführbare Binärdatei (im Format *a.out*, siehe *a.out* [7], [19]), dann gelingt die Überlagerung, und der Kern beginnt, den neuen Prozeß auszuführen.

Ist die Datei eine Textdatei (Shell-Prozedur) und beginnt ihre erste Zeile mit #!, wird das darauffolgende Wort als Pfadname einer Shell (oder eines anderen Kommandos) interpretiert, die die Shell-Prozedur ausführen soll. Weitere Worte, die in der ersten Zeile stehen, gelten als Optionen für diese Shell. Der Kern ruft die angegebene Shell auf (überlagert sie) und gibt ihr den Namen der Shell-Prozedur als Argument mit.
Ist die Datei eine Shell-Prozedur und beginnt ihre erste Zeile nicht mit #!, dann kann der Systemkern die Datei nicht überlagern, d.h. der Systemaufruf *execve()* scheitert. In diesem Fall erzeugt *csh* eine neue Shell, und zwar

- eine C-Shell *csh*, falls die erste Zeile der Datei mit einem Nummernzeichen # beginnt
- eine Bourne-Shell *sh* andernfalls;

diese neu erzeugte Shell führt dann die Datei aus.

Signalbehandlung

Normalerweise ignoriert die C-Shell das Signal QUIT. Hintergrundprozesse sind immun gegen Signale, die von der Tastatur aus gegeben werden, auch gegen Hangup-Signale (HUP). Andere Signale haben die Werte, die die C-Shell aus ihrer Umgebung geerbt hat.
Mit dem eingebauten C-Shell-Kommando *onintr* können Sie festlegen, wie die Shell Unterbrechungs- und Beendigungssignale innerhalb von Shell-Prozeduren behandelt. Login-Shells fangen das Signal TERM ab. Ist die Shell keine Login-Shell, wird dieses Signal an Sohnprozesse weitergegeben. In keinem Fall sind Unterbrechungen möglich, während eine Login-Shell die Datei *.logout* liest.

Auftragssteuerung (Job Control)

Die C-Shell ordnet jeder Kommandofolge bzw. Pipeline einen Auftrag zu, um Kommandos zu verfolgen, die im Hintergrund ablaufen oder mit einem TSTP-Signal (Terminal-Stop-Signal, üblicherweise CTRL z) gestopt wurden. Jeder Auftrag erhält eine Auftragsnummer. Wird eine Kommandofolge bzw. Pipeline mit Hilfe des Zeichens & im Hintergrund gestartet, gibt die Shell eine Zeile aus, die die Auftragsnummer in eckigen Klammern sowie die Liste der zugehörigen Prozeßnummern enthält, z.B.

[1] 1234

Im Beispiel hat der Auftrag die Auftragsnummer 1 erhalten, und es ist ihm der Top-Level-Prozeß mit der Prozeßnummer 1234 zugeordnet.

Mit dem eingebauten C-Shell-Kommando *jobs* können Sie sich die momentan gültige Liste der Aufträge ausgeben lassen. Der Auftrag, der zuletzt gestoppt wurde, bzw. (falls kein Auftrag gestopt wurde) der Auftrag, der zuletzt in den Hintergrund geschickt wurde, ist der *aktuelle* Auftrag. Er ist in der Liste mit einem Pluszeichen + markiert. Der vorhergehende Auftrag ist mit einem Minuszeichen - markiert; er nimmt die Stelle des aktuellen Auftrags ein, wenn dieser beendet oder in den Vordergrund geholt wird.

Um einen laufenden Auftrag zu stoppen, müssen Sie ihm ein Stop-Signal schicken (üblicherweise mit CTRL z). Sobald die Shell gemeldet hat, daß der Auftrag gestoppt ist, und das Bereitzeichen ausgegeben hat, können Sie den Status dieses Auftrags verändern. Hierfür stehen Ihnen die eingebauten C-Shell-Kommandos *bg, fg, kill, stop* und *%* zur Verfügung. Sie können den Auftrag z.B. mit *bg* in den Hintergrund schicken, dann einige andere Kommandos ablaufen lassen und anschließend den Auftrag mit *fg* wieder in den Vordergrund holen.

Vorsicht

Wenn ein Kommando nach einem Stop wieder gestartet wird, gibt die Shell das Dateiverzeichnis aus, in dem das Kommando gestartet wurde, falls dies nicht das aktuelle Dateiverzeichnis ist. Die Ausgabe der Shell kann jedoch falsch sein, da der Auftrag die Dateiverzeichnisse geändert haben könnte.

Eingebaute Shell-Kommandos können nicht gestoppt und wieder gestartet werden.

Kommandofolgen der Form

kommando1; kommando2; kommando3

sind problematisch: Stoppen Sie *kommando2*, führt die Shell *kommando3* niemals aus. Dies ist besonders dann zu beachten, wenn die Kommandofolge Ergebnis einer Alias-Ersetzung ist. Sie können das Problem vermeiden, indem Sie die Folge in runde Klammern (...) einschließen, so daß sie in einer Subshell ausgeführt wird.

Beim Aufruf der Kommandos *bg, fg, kill, stop* und % können Sie die Aufträge folgendermaßen angeben:

% %+ %%
der aktuelle Auftrag

%-
der letzte, vor dem aktuellen Auftrag erteilte Auftrag

%auftragsnummer
der Auftrag mit der angegebenen Auftragsnummer.
Beispiel: Das Kommando `kill -9 %3` bricht den Auftrag Nummer 3 ab.

%zeichenkette
der Auftrag, dessen zugehörige Kommandozeile mit der angegebenen Zeichenkette beginnt. Die Kommandozeile muß durch *zeichenkette* eindeutig bestimmt sein.
Beispiel: Das Kommando `fg %vi` holt einen gestoppten *vi*-Auftrag in den Vordergrund.

%?zeichenkette
der Auftrag, dessen zugehörige Kommandozeile die angegebene Zeichenkette enthält. Die Kommandozeile muß durch *zeichenkette* eindeutig bestimmt sein.

Aufträge, die im Hintergrund ablaufen, stoppen, sobald sie versuchen, von der Datensichtstation zu lesen. Hintergrundaufträge können normalerweise Ausgabe produzieren; dies können Sie jedoch verhindern, indem Sie das Kommando *stty tostop* eingeben: Die Aufträge stoppen dann, sobald sie versuchen, etwas auszugeben.

Wie meldet die C-Shell Status-Änderungen von Aufträgen?

Wenn die C-Shell interaktiv abläuft, verfolgt sie den Status jedes Auftrags und informiert Sie, sobald ein Auftrag beendet oder blockiert wird. Normalerweise gibt sie die entsprechende Meldung erst unmittelbar vor einem Bereitzeichen aus; damit werden Sie bei Ihrer Eingabe nicht gestört. Ist die Variable *notify* gesetzt, gibt die Shell Meldungen über Status-Änderungen sofort aus, anstatt auf ein anstehendes Bereitzeichen zu warten. Dies können Sie auch mit dem eingebauten C-Shell-Kommando *notify* erreichen: Mit *notify* teilen Sie der Shell mit, daß sie Status-Änderungen von bestimmten Aufträgen sofort anzeigen soll (siehe *Eingebaute C-Shell-Kommandos, notify*).

Eingebaute C-Shell-Kommandos

Eingebaute C-Shell-Kommandos werden innerhalb der C-Shell ausgeführt. Steht ein eingebautes Kommando in einer Pipeline und dort nicht an letzter Stelle, wird es in einer Subshell ausgeführt.

Die C-Shell enthält die folgenden eingebauten Kommandos:

:
Leeres Kommando. Dieses Kommando wird interpretiert, löst aber keine Aktion aus.

alias[␣name[␣def]]
weist dem Alias-Namen *name* die Definition *def* zu. *def* ist eine Liste von Worten, die Sonderzeichen für die History-Ersetzung enthalten dürfen. Diese History-Sonderzeichen müssen Sie mit vorangestelltem Gegenschrägstrich entwerten. Für *name* dürfen Sie weder *alias* noch *unalias* angeben.

def nicht angegeben:
alias gibt den Alias-Namen *name* zusammen mit seiner Definition aus.

name[*def*] nicht angegeben:
alias gibt alle Alias-Einträge aus.

Siehe *Alias-Namen für Kommandos*.

bg[␣auftrag]...
(bg - background) *bg* läßt die angegebenen Aufträge im Hintergrund laufen. Die Aufträge können Sie angeben, wie in *Auftragssteuerung (Job Control)* beschrieben.

auftrag nicht angegeben:
bg läßt den aktuellen Auftrag im Hintergrund laufen (siehe *Auftragssteuerung*).

cd[␣dvz]
chdir[␣dvz]
dvz wird zum aktuellen Dateiverzeichnis.
Ist *dvz* ein relativer Pfadname, der nicht im aktuellen Dateiverzeichnis steht, sucht *csh* in den Dateiverzeichnissen, die in der Variablen *cdpath* angegeben sind. Ist *dvz* der Name einer Shell-Variablen, deren Wert *wert* mit einem Schrägstrich / beginnt, wird *wert* zum aktuellen Dateiverzeichnis.

dvz nicht angegeben:
Das Home-Dateiverzeichnis wird zum aktuellen Dateiverzeichnis.

dirs[␣**-l**]
Der Dateiverzeichnis-Stack wird ausgegeben. Die Liste ist von links nach rechts geordnet, wobei das oberste Dateiverzeichnis des Stack (das ist das aktuelle Dateiverzeichnis) ganz links in der Liste steht.

-l
Die Dateiverzeichnisse werden in voller Länge, d.h. nicht abgekürzt, ausgegeben. Haben Sie ein Dateiverzeichnis mit Tilde ~ angegeben (siehe *Muster durch passende Dateinamen ersetzen*), wird es in der Liste vollständig ausgeschrieben.

echo[␣**-n**]␣liste
Die Worte, aus denen *liste* besteht, werden, jeweils durch ein Leerzeichen getrennt, auf die Standard-Ausgabe der C-Shell geschrieben. Ist die Option *-n* nicht angegeben, wird die Ausgabe mit einem Neue-Zeile-Zeichen abgeschlossen.

-n
Am Ende der Wortliste wird kein Neue-Zeile-Zeichen ausgegeben.

eval␣argument␣...
Die C-Shell führt die in der Argumentliste angegebenen Kommandos aus. Dabei werden die Argumente zweimal von der Shell bearbeitet:

- das erste Mal, wenn die Shell die *eval*-Kommandozeile bearbeitet.
- das zweite Mal, wenn die Shell die bearbeiteten Argumente als Kommando ausführt.

Siehe das in die Bourne-Shell *sh* eingebaute Kommando *eval*.

exec␣kommando
Die aktuelle Shell beendet sich; stattdessen wird das angegebene Kommando ausgeführt.

exit[␣(ausdruck)]
Die Shell wird mit dem Wert von *ausdruck* beendet (siehe *Ausdrücke und Operatoren*).

(ausdruck) nicht angegeben:
Die Shell wird mit dem Wert der Variablen *status* beendet.

fg[␣auftrag]...
(fg - foreground) *fg* holt die angegebenen Aufträge in den Vordergrund. Die Aufträge können Sie angeben, wie in *Auftragssteuerung (Job Control)* beschrieben.

auftrag nicht angegeben:
fg holt den aktuellen Auftrag in den Vordergrund (siehe *Auftragssteuerung*).

glob␣wortliste
Muster in *wortliste* werden durch passende Dateinamen ersetzt. Anschließend wird die Worliste ausgegeben. *glob* arbeitet wie *echo*, nur interpretiert *glob* Gegenschrägstriche \ nicht als Entwertung für Sonderzeichen. In der Ausgabe werden die Worte durch NULL-Zeichen voneinander getrennt.

hashstat
hashstat gibt eine Statistik-Zeile aus, die angibt, wie effektiv die interne Hash-Tabelle bei der Suche der Kommandos im Dateibaum gewesen ist und wie viele *exec*-Aufrufe dabei vermieden werden konnten. Bei der Kommandosuche wird für jedes in der *path*-Variablen angegebene Dateiverzeichnis,

- für das die Hash-Funktion einen möglichen Treffer angibt oder
- das nicht mit einem Schrägstrich / beginnt,

versucht, ein *exec* auszuführen. (Siehe *Kommandoausführung*.)

history[␣**-hr**][␣n]
Die History-Liste wird ausgegeben. Ist für *n* eine Zahl angegeben, werden nur die letzten *n* Ereignisse ausgegeben.

-h
Die History-Liste wird ohne die führenden Ereignis-Nummern ausgegeben. Dies ist nützlich, um Dateien für das Kommando *source -h* zu erzeugen (siehe das eingebaute C-Shell-Kommando *source*).

-r
Die History-Liste wird in umgekehrter Reihenfolge ausgegeben, d.h. zuerst erscheint das zuletzt eingegebene Ereignis.

jobs[␣**-l**]
Die aktiven Aufträge unter Auftragskontrolle werden ausgegeben.

-l
Zusätzlich werden die Prozeßnummern ausgegeben.

kill[␣-signal][␣pid]...[␣auftrag]...
kill[␣**-l**]

Das angegebene Signal bzw. das Signal TERM (Signal zur Beendigung) wird an die angegebenen Prozesse oder Aufträge gesendet. Ist das gesendete Signal TERM oder HUP (Hangup), wird zusätzlich das Signal CONT (Continue) an die Prozesse/Aufträge gesendet.
Sie müssen mindestens einen Prozeß oder Auftrag angeben, es darf nicht beides fehlen.

signal
Signalnummer oder Signalname.

pid
Prozeßnummer.

auftrag
Die Aufträge können Sie angeben, wie in *Auftragssteuerung (Job Control)* beschrieben.

-l
Die Namen der Signale, die Sie mit *kill* senden können, werden aufgelistet.

limit[␣**-h**][␣ressource[␣max]]

Mit *limit* legen Sie für den aktuellen Prozeß und alle von ihm erzeugten Prozesse einen Grenzwert *max* für die angegebene Ressource fest.
Fehlt *max*, wird der aktuelle Grenzwert für *ressource* ausgegeben. Fehlt *ressource*, werden alle Grenzwerte ausgegeben.

ressource
Für *ressource* können Sie angeben:

cputime	Maximale CPU-Zeit, die jeder Prozeß verbrauchen darf
filesize	Maximale Größe, die eine Datei haben darf
datasize	Maximale Größe des Datenbereichs (einschließlich Stack), die für den Prozeß erlaubt ist
stacksize	Maximale Stackgröße für den Prozeß
coredumpsize	Maximale Größe für eine Core-Dump-Datei

max
Für *max* können Sie eine Zahl, wahlweise gefolgt von einem Buchstaben für die Maßeinheit angeben oder, bei *cputime*, eine Zeitangabe in Minuten und Sekunden:

ressource	*max*	Bedeutung von *max*
cputime	n	*n* Sekunden
	nm	*n* Minuten
	mm:ss	*mm* Minuten und *nn* Sekunden
	nh	*n* Stunden
file-, data-, stack- und *coredumpsize*	n	*n* Kilobyte
	nk	*n* Kilobyte
	nm	*n* Megabyte

-h
Es werden harte Grenzwerte anstelle der aktuellen Grenzwerte benutzt. Harte Grenzwerte legen für die aktuellen Grenzwerte eine Obergrenze fest. Nur der Systemverwalter darf harte Grenzwerte erhöhen.

login[␣benutzername]
login[␣**-p**]
beendet eine Login-Shell und ruft und das Kommando *login* auf. Die Datei *.logout* wird nicht ausgeführt. Fehlt *benutzername*, fordert *login* Sie auf, einen Benutzernamen einzugeben.

-p
Die aktuelle Umgebung (Variablen) bleibt erhalten.

logout
beendet eine Login-Shell.

nice[␣±n][␣kommando]
Der Prioritätswert für das angegebene Kommando bzw. für die Shell wird um *n* erhöht bzw. erniedrigt.
Je höher der Prioritätswert, desto niedriger ist die Priorität des Prozesses und desto langsamer läuft er ab. Der Bereich für die Prioritätswerte geht von -20 bis 20. Liegt der angegebene Wert außerhalb dieses Bereichs, wird für den Prioritätswert 20 bzw. -20 genommen.

+n
Der Prioritätswert wird um *n* erhöht.

-n
Der Prioritätswert wird um *n* erniedrigt. Dieses Argument darf nur der Systemverwalter angeben.

±*n* nicht angegeben:
Der Prioritätswert wird auf 4 gesetzt.

kommando
Kommando, für das der Prioritätswert gesetzt werden soll. Das Kommando wird mit dem gesetzten Wert in einer Subshell ausgeführt.
Für *kommando* gelten dieselben Einschränkungen wie für das Kommando der Ablaufanweisung *if (ausdruck) kommando*.

kommando nicht angegeben:
nice setzt den Prioritätswert für die aktuelle Shell.

nohup[␣kommando]
Das angegebene Kommando wird ausgeführt, wobei HUP-Signale ignoriert werden. *kommando* läuft immer in einer Subshell ab. Für *kommando* gelten dieselben Einschränkungen wie für das Kommando der Ablaufanweisung *if (ausdruck) kommando*. Ist kein Kommando angegeben, werden HUP-Signale für den ganzen Rest der Shell-Prozedur ignoriert.
Wird ein Prozeß mit & im Hintergrund gestartet, läuft er immer mit *nohup* ab.

notify[␣auftrag]...
Die C-Shell teilt Ihnen Status-Änderungen der angegebenen Aufträge sofort mit, nicht erst unmittelbar vor Ausgabe des Bereitzeichens. Die Aufträge können Sie angeben, wie in *Auftragssteuerung (Job Control)* beschrieben.

auftrag nicht angegeben:
Die C-Shell teilt Ihnen Status-Änderungen des aktuellen Auftrags sofort mit. Der aktuelle Auftrag ist der Auftrag, der zuletzt gestoppt wurde, oder (falls kein Auftrag gestoppt wurde) der Auftrag, der zuletzt in den Hintergrund geschickt wurde.

onintr[␣-]
onintr[␣marke]
Die Aktionen, mit denen die Shell auf Interrupts reagiert, werden gesteuert.

Kein Argument angegeben
onintr setzt die Standard-Aktion ein, mit der die Shell auf Interrupts reagiert. Die Standard-Aktion besteht darin, daß die Shell Shell-Prozeduren beendet und auf Kommando-Eingaben von der Datensichtstation wartet.

-
Die Shell ignoriert alle Interrupts.

marke
Sobald die Shell einen Interrupt erhält oder sich ein Sohnprozeß beendet, weil er einen Interrupt erhielt, führt die Shell das Kommando *goto marke* aus.

popd[␣+n]
popd entfernt das oberste (0-te) Element des Dateiverzeichnis-Stacks und wechselt mit *cd* in das Dateiverzeichnis, das nun oben auf dem Stack liegt.
Die Elemente des Dateiverzeichnis-Stacks sind, bei 0 beginnend, von der Spitze aus durchnumeriert.

+n
popd entfernt das *n*-te Stack-Element.

pushd[␣dateiverzeichnis]
pushd[␣+n]
legt ein Dateiverzeichnis auf dem Dateiverzeichnis-Stack ab.

Kein Argument angegeben
Die obersten zwei Stack-Elemente werden vertauscht, und Sie wechseln mit *cd* in das Dateiverzeichnis, das nun oben auf dem Stack liegt.

dateiverzeichnis
Das aktuelle Dateiverzeichnis wird auf dem Stack abgelegt, und Sie wechseln mit *cd* in *dateiverzeichnis*.

+n
Das *n*-te Stack-Element wird oben auf dem Stack abgelegt, und Sie wechseln mit *cd* in das betreffende Dateiverzeichnis.

rehash
Die interne Hash-Tabelle für die Suche nach Kommandos im Dateibaum wird aus der Variablen *path* neu erzeugt. Siehe *Kommandoausführung*.

repeat n kommando
Das angegebene Kommando wird *n*-mal ausgeführt.
Für *kommando* gelten dieselben Einschränkungen wie für das Kommando der Ablaufanweisung *if (ausdruck) kommando*.

set[var[=wert]]
set var**[**n**]**=wort
Kein Argument angegeben
set gibt die Werte aller Shell-Variablen aus. Werte, die aus mehreren Worten bestehen, werden als geklammerte Liste ausgegeben.

var
set var weist der Shell-Variablen *var* einen leeren Wert (Null-Wert) zu.

var=wert
set var=wert weist der Shell-Variablen *var* den Wert *wert* zu. Vor und nach dem Gleichheitszeichen dürfen Leer- oder Tabulatorzeichen stehen.
wert kann sein:

wort	Einzelnes Wort (kann auch eine Zeichenkette in Anführungszeichen "..." oder Hochkommata '...' sein)
(wortliste)	Geklammerte Liste von Worten, die durch je ein Leerzeichen voneinander getrennt sind. *var* erhält damit einen Wert, der aus mehreren Worten besteht.

Enthält *wert* Muster für Dateinamen oder Kommandos in Gegenhochkommata `...`, so werden diese durch die passenden Dateinamen bzw. die Kommandoausgabe ersetzt, bevor der Wert der Variablen zugewiesen wird.

var**[**n**]**=wort
set var[*n*]=*wort* weist einer Shell-Variablen *var*, deren Wert aus mehreren Worten besteht, *wort* als *n*-te Wert-Komponente zu. Die Variable *var* und ihre *n*-te Wert-Komponente müssen bereits existieren.
Die eckigen Klammern müssen Sie angeben (d.h. sie bezeichnen hier nicht ein optionales Syntaxelement). Vor und nach dem Gleichheitszeichen dürfen Leer- oder Tabulatorzeichen stehen.
Wie bei *set var=wert* werden vor der Zuweisung Dateinamen-Muster und Kommandos in `...` ersetzt.

setenv[␣VAR[␣wort]]

Kein Argument angegeben

setenv gibt die Werte aller Umgebungsvariablen aus.

VAR

setenv VAR weist der Umgebungsvariablen *VAR* einen leeren Wert (Null-Wert) zu.
Üblicherweise werden Umgebungsvariablen mit Großbuchstaben geschrieben.

VAR␣wort

set VAR wert weist der Umgebungsvariablen *VAR* den Wert *wort* zu. *wort* ist ein einzelnes Wort (das kann auch eine Zeichenkette in Anführungszeichen "..." sein). Die am häufigsten benutzten Umgebungsvariablen *USER, TERM* und *PATH* erhalten ihren Wert automatisch von den C-Shell-Variablen *user, term* und *path* und umgekehrt; diese Variablen brauchen Sie also nicht mit *setenv* zu setzen. Darüberhinaus erhält die Umgebungsvariable PWD automatisch den Wert der C-Shell-Variablen *cwd*, wenn sich dieser ändert.

shift[␣var]

shift verschiebt die Komponenten des Wertes der Variablen *var* bzw. *argv* nach links; die erste Komponente entfällt.
Ist die Variable *var* bzw. *argv* nicht gesetzt oder hat sie einen leeren Wert, wird dies als Fehler interpretiert.

source[␣**-h**]␣datei

source führt Shell-Prozeduren in der aktuellen Shell aus. Es wird also keine Subshell erzeugt. *source* liest die Kommandos in *datei* und führt sie aus. Sie dürfen mehrere *source*-Kommandos schachteln; ist die Schachtelungstiefe jedoch zu groß, könnten nicht genug Dateideskriptoren zur Verfügung stehen. Tritt in einer der Shell-Prozeduren ein Fehler auf, werden alle geschachtelten *source*-Kommandos beendet.

-h

Die Kommandos aus *datei* werden auf die History-Liste gesetzt, jedoch nicht ausgeführt.

stop␣auftrag␣...

stoppt die angegebenen Hintergrund-Aufträge. Die Aufträge können Sie angeben, wie in *Auftragssteuerung (Job Control)* beschrieben.

suspend
stoppt die aktuelle Shell, so als ob ihr mit CTRL z ein Stop-Signal gesendet worden wäre. Dies wird meistens benutzt, um Shells zu stoppen, die mit *su* gestartet wurden.

time[␣kommando]
time führt das angegebene Kommando aus und gibt aus, wieviel Zeit das Kommando verbraucht hat.

kommando nicht angegeben:
time gibt aus, wieviel Zeit die aktuelle C-Shell und ihre Sohnprozesse verbraucht haben.

umask[␣wert]
umask gibt die aktuell gültige Schutzbit-Maske aus bzw. ändert sie. Die Schutzbit-Maske legt fest, welche Zugriffsrechte die Dateien und Dateiverzeichnisse erhalten, die Sie ab jetzt in der aktuellen Shell oder einer ihrer Subshells neu anlegen. Siehe das in die Bourne-Shell *sh* eingebaute Kommando *umask*.

wert
3-stellige Oktalzahl, die die Schutzbit-Maske angibt. Um aus der Schutzbit-Maske *wert* die Zugriffsrechte für neu angelegte Dateien zu errechnen, wird *wert* über XOR mit der Oktalzahl 666, für Dateiverzeichnisse mit 777 verknüpft.
Übliche Werte für die Schutzbit-Maske sind 002 oder 022. Dies ergibt die folgenden Zugriffsrechte:

```
002 ergibt  für Dateien:             rw-rw-r--
            für Dateiverzeichnisse:  rwxrwxr-x
022 ergibt  für Dateien:             rw-r--r--
            für Dateiverzeichnisse:  rwxr-xr-x
```

unalias␣muster
löscht alle Alias-Einträge, deren Alias-Name zu *muster* paßt, aus der Alias-Liste. *muster* ist ein Wort, das Sonderzeichen zur Dateinamen-Ersetzung enthalten kann (siehe *Muster durch passende Dateinamen ersetzen*).

unhash
verhindert, daß *csh* die interne Hash-Tabelle benutzt (siehe *Kommandoausführung*).

unlimit[␣**-h**][␣ressource]
hebt den Grenzwert für die angegebene Ressource auf.

ressource
Siehe *limit*.

ressource nicht angegeben:
Der Grenzwert für alle Ressourcen wird aufgehoben.

-h
Es werden die harten Grenzwerte aufgehoben. Dies darf nur der Systemverwalter.

unset␣muster
löscht alle Shell-Variablen, die zu *muster* passen. *muster* ist ein Wort, das Sonderzeichen zur Dateinamen-Ersetzung enthalten kann (siehe *Muster durch passende Dateinamen ersetzen*).
Das Kommando *unset* * löscht alle Shell-Variablen; dies hätte aber unerwünschte Seiteneffekte.

unsetenv␣var
löscht die Umgebungsvariable *var* aus der Umgebung. Für *var* können Sie kein Muster wie bei *unset* angeben.

wait
wartet, bis alle Hintergrundaufträge beendet sind, und gibt dann erst das Bereitzeichen aus. Ein Interrupt unterbricht das *wait*.

%[zk][&]
%zk holt den Auftrag, der durch *%zk* bestimmt ist, in den Vordergrund.
%zk& läßt den Auftrag, der durch *%zk* bestimmt ist, im Hintergrund weiterlaufen.
zk kann sein: +, %, -, *auftragsnummer*, *zeichenkette*, *?zeichenkette* (siehe *Auftragssteuerung (Job Control)*).

zk nicht angegeben:
Der aktuelle Auftrag wird in den Vordergrund geholt bzw. im Hintergrund fortgesetzt.

Das Kommando %[*zk*] hat dieselbe Wirkung wie das Kommando *fg* %[*zk*].
Das Kommando %[*zk*]& hat dieselbe Wirkung wie das Kommando *bg* %[*zk*].

@[␣var=ausdruck]
@[␣var**[n]**=ausdruck]
Kein Argument angegeben
@ gibt die Werte aller Shell-Variablen aus.

var=ausdruck
var[n]=ausdruck
@ *var=ausdruck* weist der Shell-Variablen *var* den Wert zu, den *ausdruck* liefert (siehe *Ausdrücke und Operatoren*).

@ *var[n]=ausdruck* weist einer Shell-Variablen *var*, deren Wert aus mehreren Worten besteht, den Wert von *ausdruckt* als *n*-te Wert-Komponente zu. Die Variable *var* und ihre *n*-te Wert-Komponente müssen bereits existieren.
Die eckigen Klammern müssen Sie angeben (d.h. sie bezeichnen hier nicht ein optionales Syntaxelement).

Enthält *ausdruck* die Zeichen <, >, & oder |, dann müssen diese Zeichen oder der ganze Ausdruck in runde Klammern (...) eingeschlossen werden.

Wie in C können Sie die Operatoren *=, += usw. verwenden. Zwischen dem Variablennamen und dem Zuweisungsoperator können Sie ein Leerzeichen einfügen.
Außerdem können Sie die Postfix-Operatoren ++ und -- verwenden, um den Wert von *var* um 1 zu erhöhen bzw. zu erniedrigen.

Ablaufanweisungen

Ablaufanweisungen sind bei der C-Shell eingebaute Kommandos. Die C-Shell kennt die folgenden Ablaufanweisungen:

break
foreach- oder *while*-Schleife abbrechen

breaksw
switch-Anweisung abbrechen

continue
foreach- oder *while*-Schleife mit dem nächsten Schleifendurchlauf fortsetzen

foreach ... end
Liste in einer Schleife abarbeiten

goto
an markierte Stelle springen

if
Bedingung abfragen und Kommando ausführen

if ... then ... else if ... then ... else ... endif
Bedingungen abfragen und zugehörige Kommandos ausführen

switch ... case ... breaksw ... default ... breaksw ... endsw
abfragen und verzweigen

while ... end
Schleife mit Abbruchbedingung

break
break verwenden Sie innerhalb einer *foreach*- oder *while*-Schleife, um die Schleife abzubrechen. *csh* setzt die Ausführung hinter dem zugehörigen *end* fort. Stehen hinter *break* weitere Kommandos, so werden diese ausgeführt. Damit können Sie geschachtelte Schleifen nacheinander abbrechen, indem Sie die *break*-Kommandos hintereinander in dieselbe Zeile schreiben.

breaksw
breaksw verwenden Sie innerhalb einer *switch*-Anweisung, um die Abarbeitung der Anweisung abzubrechen. *csh* setzt die Ausführung hinter dem zugehörigen *endsw* fort.

continue
continue verwenden Sie innerhalb einer *foreach*- oder *while*-Schleife; *csh* bricht den aktuellen Schleifendurchlauf ab und setzt die Ausführung mit dem nächsten Schleifendurchlauf fort.

foreach var (wortliste)
...
end
Der Variablen *var* werden der Reihe nach die Worte in *wortliste* als Wert zugewiesen. Nach jeder Zuweisung werden die Kommandos, die zwischen *foreach* und *end* stehen, ausgeführt.
Die Anweisungen *foreach var (wortliste)* und *end* müssen jeweils allein in einer eigenen Zeile stehen.

Zwischen der *foreach*- und der *end*-Zeile können die Kommandos *break* und *continue* stehen (siehe oben).

Liest die Shell die *foreach*-Anweisung von einer Datensichtstation (d.h. nicht aus einer Shell-Prozedur), so gibt sie ein Fragezeichen aus, bevor sie die Kommandos in der *foreach*-Schleife ausführt. Auf das Fragezeichen hin können Sie Kommandos eingeben. Erst wenn Sie *end* eingeben, führt die Shell die Kommandos in der *foreach*-Schleife aus. Die Kommandos, die Sie auf das Fragezeichen hin eingeben, werden nicht in die History-Liste gesetzt.

goto marke

Enthält *marke* Sonderzeichen zur Dateinamen-Ersetzung oder Zeichenketten in Gegenhochkommata `...`, so werden diese zuerst ersetzt (siehe *Muster durch passende Dateinamen ersetzen* und *Kommandos durch ihre Ausgabe ersetzen*). Anschließend geht die Shell in ihrer Eingabe soweit wie möglich zurück und sucht nach einer Zeile der Form

marke:

(vor *marke:* dürfen Leer- oder Tabulatorzeichen stehen). Die Shell setzt die Ausführung nach dieser Zeile fort. Es ist ein Fehler, wenn die Zeile *marke:* innerhalb einer *while*- oder *foreach*-Schleife steht.

if (ausdruck) kommando

Hat *ausdruck* den Wert "wahr", wird *kommando* ausgeführt. *kommando* ist ein einfaches Kommando, d.h. keine Kommandofolge, Pipeline oder Kommandofolge in runden Klammern (...). Die Variablenersetzung in *kommando* findet gleichzeitig mit der Variablenersetzung in *ausdruck* statt.

Enthält *kommando* Sonderzeichen zur Ein-/Ausgabe-Umlenkung, so findet die Umlenkung auch dann statt, wenn *ausdruck* falsch ist.

if (ausdruck1) **then**
...
else if (ausdruck2) **then**
...
else
...
endif

Hat *ausdruck1* den Wert "wahr", werden die Kommandos bis zum ersten *else* ausgeführt. Anschließend wird die Abarbeitung nach *endif* fortgesetzt.
Hat *ausdruck1* den Wert "falsch", wird *ausdruck2* ausgewertet.
Hat *ausdruck2* den Wert "wahr", werden die Kommandos zwischen *else if* und dem zweiten *else* ausgeführt. Anschließend wird die Abarbeitung nach *endif* fortgesetzt.
Hat *ausdruck2* den Wert "falsch", werden die Komandos zwischen dem zweiten *else* und *endif* ausgeführt.

Es sind beliebig viele *else if*-Anweisungen erlaubt, aber nur eine *else*-Anweisung. Die *endif*-Anweisung ist nur einmal erforderlich, darf aber nicht fehlen.

Die Anweisung *if (ausdruckX)* muß entweder allein in einer eigenen Zeile stehen (evtl. gefolgt von einem *then*) oder nach einem *else*. Die Schlüsselwörter *else* und *endif* müssen als erstes Wort in der Zeile stehen (d.h. davor dürfen nur Leer- und Tabulatorzeichen stehen).

switch (zeichenkette)
case marke:
...
breaksw
...
default:
...
breaksw
endsw

Enthält *zeichenkette* Sonderzeichen zur Dateinamen-Ersetzung oder Zeichenketten in Gegenhochkommata `...`, so werden diese zuerst ersetzt (siehe *Muster durch passende Dateinamen ersetzen* und *Kommandos durch ihre Ausgabe ersetzen*). Die *case*-Marken können die Sonderzeichen *, ? und [...] zur Dateinamen-Ersetzung enthalten. Variablen in den *case*-Marken werden ersetzt.
Anschließend wird *zeichenkette* der Reihe nach mit allen *case*-Marken verglichen. Paßt *zeichenkette* zu einer *case*-Marke, führt die Shell die Kommandos nach dieser Marke aus. Erreicht die Shell eine *default*-Marke und hat bis dahin noch keine *case*-Marke gepaßt, führt die Shell die Kommandos nach der *default*-Marke aus. Paßt keine Marke und ist keine *default*-Marke vorhanden, setzt die Shell die Abarbeitung nach *endsw* fort.

Das Kommando *breaksw* bricht die *switch*-Anweisung ab und setzt die Abarbeitung nach *endsw* fort.
Paßt *zeichenkette* zu einer Marke und steht zwischen dieser Marke und der nächsten Marke kein *breaksw*, so werden auch die Kommandos der nächsten Marke ausgeführt usw. (genau wie in der Programmiersprache C).

Die Schlüsselwörter *switch*, *case*, *default* und *endsw* müssen jeweils als erstes Wort in der Zeile stehen. Die *default*-Marke sollte nach allen *case*-Marken stehen.

while (ausdruck)
...
end

Solange *ausdruck* den Wert "wahr" hat, werden die Kommandos zwischen der *while*-Anweisung und dem zugehörigen *end* ausgeführt.
Die Anweisungen *while (ausdruck)* und *end* müssen jeweils allein in einer eigenen Zeile stehen.

Zwischen der *while*- und der *end*-Zeile können die Kommandos *break* und *continue* stehen (siehe oben).

Liest die Shell die *while*-Anweisung von einer Datensichtstation (d.h. nicht aus einer Shell-Prozedur), so gibt sie ein Fragezeichen aus, bevor sie die Kommandos in der *while*-Schleife ausführt. Auf das Fragezeichen hin können Sie Kommandos eingeben. Erst wenn Sie *end* eingeben, führt die Shell die Kommandos in der *while*-Schleife aus. Die Kommandos, die Sie auf das Fragezeichen hin eingeben, werden nicht in die History-Liste gesetzt.

Umgebungsvariablen und vordefinierte Shell-Variablen

Anders als die Standard-Shell *sh* unterscheidet die C-Shell zwischen Umgebungsvariablen und Shell-Variablen: Umgebungsvariablen werden automatisch an aufgerufene Prozesse exportiert, Shell-Variablen nicht. Beide Arten von Variablen werden bei der Variablen-Ersetzung gleich behandelt.
Bei der Initialisierung setzt die C-Shell die Shell-Variablen *argv, cwd, home, path, prompt, shell* und *status*. Außerdem kopiert die C-Shell die Umgebungsvariable *USER* in die Shell-Variable *user*, *TERM* in *term* und *HOME* in *home*; werden diese Shell-Variablen neu gesetzt, kopiert die C-Shell sie in die entsprechenden Umgebungsvariablen zurück. Mit *PATH* und *path* verfährt die C-Shell genauso; Sie müssen *path* nur einmal in der Datei *.cshrc* oder *.login* setzen. Die Umgebungsvariable *PWD* erhält ihren Wert aus der Shell-Variablen *cwd*; ändert sich der Wert von *cwd*, wird auch *PWD* neu gesetzt.
Mit dem Kommando *set path* eliminieren Sie aus dem Wert von *path* mehrfach vorhandene Pfadnamen. Dies kann entstehen, wenn in einer Shell-Prozedur oder in der Datei *.cshrc* ein Kommando wie z.B. *set path=(/usr/local /usr/hosts $path)* vorkommt, das sicherstellt, daß die genannten Dateiverzeichnisse in die Pfadnamen-Liste aufgenommen werden.

Die folgenden Shell-Variablen haben eine feste Bedeutung:

argv
Argumentliste. Sie enthält die Liste der Kommandozeilen-Argumente, die an die aktuelle Shell übergeben werden. Die Variable *argv* legt die Werte der Stellungsparameter *$1, $2* usw. fest: *$argv*[1] entspricht $1 usw.

cdpath
enthält eine Liste von Dateiverzeichnissen. Die Kommandos *cd, chdir* und *popd* durchsuchen diese Dateiverzeichnisse, wenn das Dateiverzeichnis, das diese Kommandos als Argument erhalten, kein Unterverzeichnis des aktuellen Dateiverzeichnisses ist.

cwd
enthält den vollen Pfadnamen des aktuellen Dateiverzeichnisses.

echo
Ist diese Variable gesetzt, schreibt *csh* jede Kommandoeingabe auf die Standard-Ausgabe, und zwar nach allen Ersetzungen, direkt vor der Kommandoausführung. Siehe die *csh*-Optionen *-x* und *-X*.

fignore
Liste von Dateinamen-Suffixen, die bei der Vervollständigung von Dateinamen ignoriert werden. Üblicherweise besteht diese Liste aus dem Suffix *.o*. (Siehe *Vervollständigung von Dateinamen und Benutzerkennungen*)

filec

Wenn die Variable *filec* gesetzt ist, kann eine interaktive C-Shell einen nur teilweise eingegebenen Dateinamen oder eine nur teilweise eingegebene Benutzerkennung vervollständigen. In diesem Fall haben die Zeichen ESC und EOF (CTRL d) in einer Eingabezeile, die von der Datensichtstation eingegeben wird, eine Sonderbedeutung:

ESC Beginnt die vor ESC stehende Zeichenkette mit einer Tilde ~, ergänzt die Shell diese Zeichenkette zu ~*benutzerkennung*, wobei *benutzerkennung* eine existierende Benutzerkennung ist. Kommen hierfür mehrere Kennungen in Frage, ertönt die Terminalglocke.
Beginnt die vor ESC stehende Zeichenkette nicht mit einer Tilde ~, ergänzt die Shell diese Zeichenkette zu einem Dateinamen des aktuellen Dateiverzeichnisses. Kommen hierfür mehrere Dateinamen in Frage, ertönt die Terminalglocke.

EOF Die Shell listet alle Benutzerkennungen bzw. Dateinamen des aktuellen Dateiverzeichnisses auf, die mit der vor EOF stehenden Zeichenkette beginnen. Anschließend gibt die Shell die unvollständige Kommandozeile erneut aus, damit Sie eine der aufgelisteten Kennungen bzw. einen der aufgelisteten Dateinamen eingeben können.

(Siehe *Vervollständigung von Dateinamen und Benutzerkennungen*)

hardpaths

Ist diese Variable gesetzt, werden die Pfadnamen im Dateiverzeichnis-Stack so aufgelöst, daß sie keine symbolischen Verweise enthalten. Diese Variable ist besonders nützlich, da die C-Shell Probleme mit symbolischen Verweisen haben kann.

histchars

enthält eine Zeichenkette aus zwei Zeichen. Das erste Zeichen ist das Zeichen, mit dem ein History-Befehl beginnt (normalerweise das Ausrufezeichen). Das zweite Zeichen ist das Sonderzeichen, das für den History-Befehl ^*l*^*r*[^] verwendet wird (normalerweise das Dach ^, siehe *History-Ersetzung, Kurzschreibweise*).

history

Anzahl der Zeilen, die in der History-Liste gespeichert werden. Diese Zahl darf beliebig groß sein; sie wird nur durch den Speicherplatz begrenzt, den die C-Shell zur Verfügung stellen kann.
Ist die Variable nicht gesetzt, wird nur das zuletzt eingegebene Kommando gespeichert.

home

enthält das HOME-Dateiverzeichnis.
Ist in einer Kommando-Eingabezeile ein HOME-Dateiverzeichnis mit Hilfe einer Tilde ~ angegeben, fragt die Shell die Variable *home* ab, um das richtige Dateiverzeichnis zu ermitteln (siehe *Muster durch passende Dateinamen ersetzen*).

ignoreeof
Ist diese Variable gesetzt, ignoriert die Shell EOF-Zeichen, die von einer Datensichtstation eingegeben werden. Damit können Sie verhindern, daß die Shell durch irrtümliche Eingabe von CTRL d beendet wird.

mail
Liste der Dateien, die die C-Shell bzgl. des Empfangs von *mail*-Nachrichten überprüft. Ist das erste Wort des Wertes von *mail* eine Zahl *n*, überprüft die Shell die in der Liste angegebenen Dateien alle *n* Sekunden. Ist keine Zahl angegeben, überprüft die Shell die *mail*-Dateien alle 5 Minuten.

nobeep
Ist diese Variable gesetzt, so läutet die Terminalglocke nicht, wenn zu einer unvollständigen Eingabe mehrere Datei- bzw. Benutzernamen passen (siehe *Vervollständigung von Dateinamen und Benutzerkennungen*).

noclobber
Ist diese Variable gesetzt, ist nur eine eingeschränkte Ausgabe-Umlenkung möglich: Mit > kann die Ausgabe nur in neue, noch nicht existierende Dateien umgelenkt werden; mit >> kann die Ausgabe nur in bereits existierende Dateien umgelenkt werden. Siehe *Umlenkung der Ein-/Ausgabe*.

noglob
verhindert die Ersetzung von Mustern durch passende Dateinamen. Dies ist in Shell-Prozeduren nützlich, wenn die gewünschten Dateinamen bereits zu Verfügung stehen und keine weiteren Ersetzungen stattfinden sollen.

nonomatch
Paßt zu einem Dateinamen-Muster kein Dateiname, interpretiert die Shell dies nicht als Fehler, sondern gibt das eingegebene Muster aus. Muster mit falscher Syntax bewirken weiterhin einen Fehler.

notify
Ist diese Variable gesetzt, teilt Ihnen die Shell Status-Änderungen von Aufträgen sofort mit, nicht erst unmittelbar vor Ausgabe des Bereitzeichens (siehe *Auftragssteuerung*).

path
Liste der Dateiverzeichnisse, die bei der Ausführung eines Kommandos nach der zugehörigen Datei durchsucht werden (siehe *Kommandoausführung*). Ist der Wert von *path* die leere Zeichenkette, wird nur das aktuelle Dateiverzeichnis durchsucht. Üblicherweise hat *path* den Wert

(. /usr/ucb /usr/bin)

Ist *path* nicht gesetzt, führt die Shell nur Kommandos mit absolutem Pfadnamen aus.

path wird mit dem Wert der Umgebungsvariablen *PATH* initialisiert; ändert *path* seinen Wert, so kopiert die Shell diesen Wert in die Umgebungsvariable *PATH* zurück. Eine interaktive C-Shell setzt die in *path* enthaltenen Dateiverzeichnisse normalerweise in folgenden Fällen in die Hash-Tabelle:

- nachdem sie die Datei *.cshrc* gelesen hat
- nachdem die Variable *path* neu gesetzt worden ist
- wenn Sie das eingebaute C-Shell-Kommando *rehash* aufrufen.

Das Kommando *rehash* sollten Sie immer dann aufrufen, wenn sich der Inhalt der Dateiverzeichnisse, die in *path* eingetragen sind, geändert hat (z.B. wenn ein neues Kommando hinzugekommen ist). Siehe *Kommandoausführung*.

prompt

Bereitzeichenkette, mit der eine interaktive C-Shell zur Eingabe auffordert.
Ein Ausrufezeichen ! im Wert von *prompt* wird bei der Ausgabe der Bereitzeichenkette durch die aktuelle Ereignisnummer ersetzt (siehe *History-Ersetzung*). Beachten Sie, daß Sie beim entsprechenden *set*-Aufruf das Ausrufezeichen mit einem vorangestellten Gegenschrägstrich \ entwerten müssen, damit die Shell keine History-Ersetzung durchführt (siehe *Beispiele*, *History-Ersetzung*).
Als Standard-Bereitzeichenkette gibt eine interaktive Shell *rechnername%␣* (für normale Benutzer) bzw. *rechnername#␣* (für den Systemverwalter) aus.
Nicht-interaktive C-Shells setzen die Variable *prompt* nicht. Wollen Sie Alias-Namen und andere Kommandos, die nur interaktiv sinnvoll sind, in die Datei *.cshrc* schreiben, können Sie vorher die folgende *if*-Anweisung einfügen, um die Start-Zeit für nicht-interaktive C-Shells zu verkürzen:

```
if ($?prompt == 0) exit
```

(Ist die Variable *prompt* nicht gesetzt, d.h. ist die Shell nicht-interaktiv, wird *exit* ausgeführt.)

savehist

Anzahl der Zeilen der History-Liste, die in der Datei *~/.history* gespeichert werden, wenn Sie sich ausloggen.
Große Werte von *savehist* verlängern die Start-Zeit der C-Shell.

shell

Datei, die das C-Shell-Programm enthält.
Diese Variable wird benutzt, wenn mit *fork()* eine Shell erzeugt werden soll, um eine Datei zu interpretieren, die Ausführrecht (x-Bits) besitzt, aber die das System nicht ausführen kann. Siehe *Kommandoausführung*.

status

Ende-Status des zuletzt ausgeführten Kommandos. Wurde das Kommando nicht ordnungsgemäß beendet, wird zum Wert von *status* 0200 addiert. Eingebaute Kommandos liefern den Ende-Status 1 bei Mißerfolg und 0 sonst.

time

Ist diese Variable gesetzt, gibt die Shell nach Beendigung jedes Kommandos eine Übersicht über die Ressourcen aus, die das Kommando verbraucht hat.
Der Wert von *time* kann leer sein oder aus einer oder zwei Komponenten bestehen.

- Ist der Wert leer, hat die Übersicht die Form
  %U %S %E %P %X %D %I %O %F %W (siehe unten).
- Für die erste Wert-Komponente können Sie eine Zahl *n* angeben; die Shell gibt dann die Übersicht nur für solche Kommandos aus, die mindestens *n* Sekunden CPU-Zeit verbraucht haben.
- Die zweite Wert-Komponente ist eine Zeichenkette, die das Ausgabeformat der Übersicht festlegt (siehe die Formatzeichenkette bei *printf()*). Die Zeichenkette kann sowohl normale Zeichen als auch Feldbezeichner enthalten. Normale Zeichen werden unverändert ausgegeben. Feldbezeichner sind von der Form *%buchstabe*. Sie werden bei der Ausgabe durch ihren Wert ersetzt. Folgende Feldbezeichner können Sie angeben:

  %D Getrennt benutzter Datenbereich (unshared data space) in Kilobyte, der durchschnittlich belegt wurde
  %E Zeit, die während der Ausführung vergangen ist (real time)
  %F Page faults
  %I Anzahl der Blockleseoperationen von der Platte
  %K Getrennt benutzter Stackbereich (unshared stack space) in Kilobyte, der durchschnittlich belegt wurde
  %M Gesamtgröße des physikalischen Speichers (real memory), der während der Ausführung des Prozesses belegt wurde
  %O Anzahl der Blockschreibeoperationen von der Platte
  %P Gesamte verbrauchte CPU-Zeit (%U plus %S) in Prozent der %E-Zeit
  %S CPU-Zeit in Sekunden, die der Systemkern für den Benutzerprozeß verbraucht hat (Zeit, die im Systemmodus verbracht wurde)
  %U CPU-Zeit in Sekunden, die der Benutzerprozeß selbst verbraucht hat (Zeit, die im Benutzermodus verbracht wurde)
  %W Anzahl der Auslager-Operationen (swaps)
  %X Gemeinsam genutzter Speicher (shared memory) in Kilobyte, der durchschnittlich belegt wurde

verbose

Ist diese Variable gesetzt, gibt die Shell jedes eingegebene Kommando aus, nachdem die History-Ersetzung stattgefunden hat. Siehe die *csh*-Optionen *-v* und *-V*.

C-Shell-Prozeduren

Wie die Bourne-Shell *sh* bietet auch die C-Shell *csh* die wesentlichen Elemente einer Programmiersprache. Ein solches Programm heißt C-Shell-Prozedur.
Eine C-Shell-Prozedur besteht aus einer oder mehreren Kommandozeilen. Insbesondere stehen Ihnen eine Reihe von eingebauten Kommandos zur Verfügung, mit denen Sie den Ablauf von Kommandos in der Shell-Prozedur steuern können. Diese Kommandos veranlassen die Shell entweder, Schleifen auszuführen, oder unter bestimmten Bedingungen einen Teil der Eingabe zu überspringen (siehe *Eingebaute C-Shell-Kommandos, Ablaufanweisungen*).
Üblicherweise schreiben Sie die Shell-Prozedur in eine Datei. Die Datei muß Lese- und Ausführrecht besitzen. Shell-Prozeduren müssen nicht übersetzt werden. Sie starten eine C-Shell-Prozedur normalerweise mit dem Kommando *csh datei*. Daraufhin erzeugt die Shell eine Subshell, die jede Zeile in *datei* liest, interpretiert und ausführt. Sind alle Kommandozeilen ausgeführt, beendet sich die Subshell, und die übergeordnete Shell meldet sich zurück.
C-Shell-Prozeduren können Sie auch mit dem eingebauten C-Shell-Kommando *source* starten; im letzteren Fall wird die Prozedur in der aktuellen Shell ausgeführt, es wird also keine Subshell erzeugt.

Das Nummernzeichen # leitet einen Kommentar ein; der Kommentar endet mit der Eingabezeile. Bei der Abarbeitung der Prozedur ignoriert die Shell alle Zeichen, die zum Kommentar gehören.

Beginnt die erste Zeile einer Shell-Prozedur mit #!, wird das darauffolgende Wort als Pfadname einer Shell (oder eines anderen Kommandos) interpretiert, die die Shell-Prozedur ausführen soll. Weitere Worte, die in der ersten Zeile stehen, gelten als Optionen für diese Shell. Der Kern ruft die angegebene Shell auf (überlagert sie) und gibt ihr den Namen der Shell-Prozedur als Argument mit.
Beginnt die erste Zeile der Prozedur nicht mit #!, erzeugt *csh* eine neue Shell, und zwar

- eine C-Shell *csh*, falls die erste Zeile der Prozedur mit einem Nummernzeichen # beginnt
- eine Bourne-Shell *sh* andernfalls;

diese neu erzeugte Shell führt dann die Prozedur aus.

GRENZWERTE

Für die C-Shell gelten die folgenden Beschränkungen:

- Worte können höchstens 1024 Zeichen lang sein.
- Argumentlisten dürfen höchstens 1 048 576 Zeichen lang sein.
- Kommandos, bei denen Muster durch passende Dateinamen ersetzt werden sollen, dürfen höchstens 1706 Argumente haben.
- Werden Kommandos in Gegenhochkommata `...` durch ihre Ausgabe ersetzt, darf die entstehende Argumentliste höchstens 1 048 576 Zeichen lang sein (siehe oben).
- Die Shell führt pro Zeile höchstens 20 Alias-Ersetzungen durch, um etwaige Schleifen entdecken zu können.

FEHLERMELDUNGEN

```
You have stopped jobs.
```

Sie wollten die C-Shell beenden, hatten jedoch gestoppte Aufträge unter Auftragskontrolle. Ein erneuter Versuch, die Shell zu beenden, wird erfolgreich sein, die gestoppten Aufträge werden beendet.

DATEIEN

~/.cshrc
Datei, die die C-Shell beim Start ausführt.

~/.login
Datei, die eine Login-C-Shell beim Start nach der Datei *.cshrc* ausführt.

~/.logout
Datei, die eine Login-C-Shell beim Logout ausführt.

~/.history
Datei, in der beim Logout die History-Liste gespeichert wird, so daß die Liste beim nächsten Login wieder zur Verfügung steht.

/usr/bin/sh
Standard-Shell. *sh* ist eine Bourne-Shell. Shell-Prozeduren, die nicht mit einem Nummernzeichen # beginnen, werden von *sh* ausgeführt.

*/tmp/sh**
Temporärdatei für << (siehe *Umlenkung der Ein-/Ausgabe*).

/etc/passwd
Kennwortdatei. Sie enthält u.a. die HOME-Dateiverzeichnisse, die die C-Shell bei der Auswertung des Musters *~benutzerkennung* benötigt (siehe *Muster durch passende Dateinamen ersetzen*).

BEISPIELE

Der Kürze halber wird in den Beispielen, wo nichts anderes erwähnt ist, das Bereitzeichen %␣ (statt des Standard-Bereitzeichens *rechnername*%␣) verwendet.

Vervollständigung von Dateinamen und Benutzerkennungen

Für *Beispiel 1* und *2* wird vorausgesetzt, daß die Variable *filec* gesetzt ist.

1. *Dateinamen*

   Das aktuelle Dateiverzeichnis enthalte die Dateien *chaosnet, class, cmd, liste.alt* und *liste.neu*.

   Wenn Sie eingeben:

   ```
   % vi ch (ESC)
   ```

   dann vervollständigt die Shell diese Eingabezeile zu

   ```
   vi chaosnet
   ```

   Geben Sie dagegen ein:

   ```
   % vi l (ESC)
   ```

   dann läutet die Shell die Terminalglocke, da mehrere Dateinamen passen (*liste.alt* und *liste.neu*).

   Wenn Sie statt des Zeichens (ESC) das EOF-Zeichen eingeben, listet die Shell alle passenden Dateinamen auf:

   ```
   % vi l (END)
   liste.alt liste.neu
   % vi l
   ```

2. *Benutzerkennungen*

   Die Eingabezeile

   ```
   % cd ~ro (ESC)
   ```

   wird vervollständigt zu

   ```
   cd ~root
   ```

   falls *ro* eine eindeutige Abkürzung ist. Damit wechseln Sie in das HOME-Dateiverzeichnis von *root*.

History-Ersetzung

Für die Beispiele dieses Abschnitts wird vorausgesetzt, daß das Bereitzeichen für die C-Shell folgendermaßen definiert ist:

```
set prompt="\!%␣"
```

Die C-Shell gibt dann als Bereitzeichen die aktuelle Ereignisnummer, gefolgt von Prozent- und Leerzeichen aus (siehe die Shell-Variable *prompt*).
Außerdem muß die Variable *history* auf einen genügend großen Wert (z.B. 10) gesetzt sein.

3. *Ereignis-Bezeichner*

   Mit der nachstehenden Kommandofolge erzeugen Sie zuerst mit dem Editor *vi* die C-Quelldatei *prog.c* und übersetzen sie anschließend mit dem C-Compiler *cc*. Da Sie im Programm einen Schreibfehler gemacht haben, meldet *cc* einen Syntaxfehler. Sie editieren daraufhin die Datei erneut, korrigieren den Fehler und übersetzen das Programm wieder mit *cc*.

```
1% vi prog.c
 ...
2% cc -O prog.c -o form
 ...
3% vi prog.c
 ...
4% cc -O prog.c -o form
5%
```

   Mit Hilfe der History-Ersetzung können Sie sich die zweimalige Eingabe desselben Kommandos ersparen:

```
1% vi prog.c
 ...
2% cc -O prog.c -o form
 ...
3% !v        oder        3% !vi
vi prog.c
 ...
4% !c        oder        4% !cc        oder        4% !?form
cc -O prog.c -o form
5%
```

   Im Ereignis 3 wurde der History-Befehl *!v* bzw. *!vi* eingegeben. Dabei ist *v* bzw. *vi* der Ereignis-Bezeichner; er bezeichnet das zuletzt eingegebene Kommando, das mit *v* (*vi*) beginnt - das ist das Kommando *vi prog.c*. Die C-Shell schreibt die bezeichnete Kommandozeile auf den Bildschirm und führt sie anschließend aus. Entsprechend wird durch den History-Befehl *!c* bzw. *!cc* (Ereignis 4) das Kommando *cc -O prog.c -o prog* ausgeführt. Dasselbe leistet hier der History-Befehl *!?form*: Er bezeichnet das zuletzt eingegebene Kommando, das (an beliebiger Stelle) die Zeichenkette *form* enthält.

Statt mit Zeichenketten können Sie die Kommandos auch mit Ereignisnummern ansprechen. Wissen Sie nicht, welche Nummer dem gewünschten Kommando in der History-Liste zugeordnet ist, können Sie sich die Liste mit dem eingebauten C-Shell-Kommando *history* ausgeben lassen. Die Ereignisnummern können Sie entweder absolut oder relativ zum aktuellen Ereignis angeben:

```
1% vi prog.c
 ...
2% cc -O prog.c -o form
 ...
3% history
   1 vi prog.c
   2 cc -O prog.c -o form
   3 history
4% !1        oder        4% !-3
vi prog.c
 ...
5% !2        oder        5% !-3
cc -O prog.c -o form
6%
```

4. *Wort-Bezeichner*

Zuletzt eingegebenes Kommando	Eingabezeile mit History-Befehlen	Eingabezeile nach der History-Ersetzung
grep ^anna /etc/passwd	grep !*	grep ^anna /etc/passwd
	grep !:1 datei grep !^ datei	grep ^anna datei
	grep ^berta !:2 grep ^berta !$	grep ^berta /etc/passwd

5. *Parameter*

Zuletzt eingegebenes Kommando	Eingabezeile mit History-Befehlen	Eingabezeile nach der History-Ersetzung
ls -l /home/anna/plan	ls -ld !$:h ls -ld !$:t	ls -ld /home/anna ls -ld plan
vi prog.c	cc !$ -o !$:r cc !$ -o !$:s/.c/	cc prog.c -o prog
cc prog.c -o prog	!:gs/prog/liste !:s/prog/liste	cc liste.c -o liste cc liste.c -o prog
cat d1 d2 d3	cat !:1:s/d/dat/ !:3:&	cat dat1 dat3

Bei komplizierten History-Befehlen sollten Sie zuerst mit dem Parameter *p* prüfen, ob die History-Ersetzung die gewünschte Kommandozeile liefert, und erst dann das Kommando ausführen. Das würde im obigen Beispiel so aussehen:

```
1% cat d1 d2 d3
 ...
2% cat !:1:s/d/dat/ !:3:&:p
cat dat1 dat3
3% !!
cat dat1 dat3
 ...
4%
```

6. *Klammern*

Zuletzt eingegebenes Kommando	Eingabezeile mit History-Befehlen	Eingabezeile nach der History-Ersetzung
`ls -ld ~paul`	`!la`	das letzte Kommando, das mit la beginnt
	`!{l}a`	`ls -ld ~paula`
	`!!a`	
	`!{!}a`	

7. *Kurzschreibweise*

Zuletzt eingegebenes Kommando	Eingabezeile mit History-Befehlen	Eingabezeile nach der History-Ersetzung
`grep ^berta /etc/passwd`	`^tr^rt`	`grep ^berta /etc/passwd`
	`!:s^tr^rt`	
	`!:s/tr/rt`	

Alias-Namen

8. *Einfaches Beispiel*

```
% alias lld ls -ld
% alias
...
lld     (ls -ld)
...
```

Wenn Sie eingeben:

```
% lld /usr
```

dann liest die Shell die zum Alias-Namen *lld* gehörige Alias-Definition *ls -ld* und fügt das Argument */usr* an. Damit ergibt sich das Kommando

```
ls -ld /usr
```

Mit dem Kommando

```
% unalias lld
```

löschen Sie den Alias-Eintrag *lld ls -ld* aus der Alias-Liste. Mit

```
% unalias *
```

löschen Sie die gesamte Alias-Liste.

9. *History-Befehle in Alias-Definitionen*

```
% alias gpw grep \!^ /etc/passwd
```

Wegen des Gegenschrägstrichs wertet die Shell den History-Befehl bei der Ausführung des *alias*-Kommandos noch nicht aus:

```
% alias
gpw     (grep !^ /etc/passwd)
```

Wenn Sie nun eingeben:

```
% gpw anna
```

dann liest die Shell die zum Alias-Namen *gpw* gehörige Alias-Definition *grep* !^ */etc/passwd* und ersetzt den History-Befehl !^. Bei dieser History-Ersetzung interpretiert die Shell das Kommando *gpw anna* als letztes Ereignis, d.h. so, als ob Sie nacheinander die Kommandos

```
gpw anna
grep !^ /etc/passwd
```

eingegeben hätten; der History-Befehl !^ bezeichnet also das Argument *anna*, und das entstehende Kommando lautet:

```
grep anna /etc/passwd
```

Hierzu noch zwei komplexere Beispiele:

```
% alias vcc 'vi \!^; cc \!^ -o \!^:r'
% alias
vcc     vi !^; cc !^ -o !^:r
```

Der Aufruf

```
% vcc prog.c
```

ergibt die Kommandofolge

```
vi prog.c; cc prog.c -o prog
```

Beim *alias*-Aufruf müssen Sie die Alias-Definition in Hochkommata '...' einschließen, da die Shell sonst den Strichpunkt als Abschluß des Kommandos *alias* interpretiert statt als Bestandteil der Alias-Definition.

```
% alias lm 'ls -l \!* | more'
```

Der Aufruf

```
% lm dvz1 dvz2
```

ergibt die Pipeline

```
ls -l dvz1 dvz2 | more
```

10. *Geschachtelte Alias-Definition*

```
% alias clm 'cd \!^; lm'
% alias lm 'ls -l | more'
```

Der Aufruf

```
% clm dvz
```

ergibt die Kommandofolge

```
cd dvz; ls -l | more
```

11. *Alias-Definition, die den zugehörigen Alias-Namen enthält*

```
% alias ls ls -l
```

Der Aufruf

```
% ls dvz
```

ergibt das Kommando

```
ls -l dvz
```

Variablen-Ersetzung

12. Das folgende Beispiel zeigt, daß C-Shell-Variablen als Felder (arrays) wie in C betrachtet werden können. Sie können sie mit einer Wortliste initialisieren, die einzelnen Elemente mit Index ansprechen und ihnen Werte zuweisen.

```
% set farbe=(rot gelb gruen)
% echo $farbe
rot gelb gruen
% echo $#farbe
3
% echo $farbe[3] $farbe[1] $farbe[2-3]
gruen rot gelb gruen
% set farbe[2]=blau
% echo $farbe
rot blau gruen
```

Zunächst wird der Shell-Variablen *farbe* die Wortliste *rot gelb gruen* als Wert zugewiesen. Das zweite Kommando gibt den Wert von *farbe* aus. *echo $#farbe* liefert die Anzahl der Worte, aus denen der Wert von *farbe* besteht. Die drei letzten Kommandos zeigen, wie Sie einzelne Wert-Komponenten ansprechen und verändern können.

Muster durch passende Dateinamen ersetzen

13. Das aktuelle Dateiverzeichnis enthalte die Dateien *liste*, *plan*, *plan1*, *plan2*, *plan3*, *planung*.
Das Unterdateiverzeichnis *prog* enthalte die Dateien *ls.c*, *newls.c*, *oldls.c* und *pgm.c*.

Muster	wird ersetzt durch
plan*	plan plan1 plan2 plan3 planung
plan?	plan1 plan2 plan3
plan[12]	plan1 plan2
plan[1-3]	plan1 plan2 plan3
prog/*ls*	prog/ls.c prog/newls.c prog/oldls.c
prog/{oldls,ls,main}.c	prog/oldls.c prog/ls.c prog/main.c
./{plan?,liste}	./plan1 ./plan2 ./plan3 ./liste

Beachten Sie in obiger Tabelle die Reihenfolge: Bei der Auswertung der geschweiften Klammern {...} werden die Zeichenketten nicht sortiert; bei der Auswertung der anderen Sonderzeichen werden die passenden Dateinamen alphabetisch sortiert. Die vorletzte Zeile der Tabelle macht zudem deutlich, daß die Shell die geschweiften Klammern direkt auswertet, ohne die entstehenden Zeichenketten mit den Namen der existierenden Dateien zu vergleichen; so enthält die Liste, durch die das Muster ersetzt wird, auch den Pfadnamen *prog/main.c*, obwohl eine solche Datei gar nicht existiert.

Wenn Ihr HOME-Dateiverzeichnis */home/anna* ist und das HOME-Dateiverzeichnis des Benutzers Kurt */home/kurt*, dann werden die Muster ~ bzw. ~*kurt* folgendermaßen ausgewertet:

Muster	wird ersetzt durch
~/version5/prog	/home/anna/version5/prog
~kurt/prog	/home/kurt/prog
~kurt/prog/*.c	/home/kurt/prog/cls.c /home/kurt/prog/ls.c

SIEHE AUCH

login, *sh*
a.out, *access()*, *exec()*, *fork()*, *pipe()* [19]
ascii, *environ*, *terminfo* [7]

csplit
Datei nach bestimmten Kriterien unterteilen (context split)

csplit teilt den Inhalt einer Datei oder den Eingabetext, den *csplit* von der Standard-Eingabe liest, in kleinere Abschnitte auf. Die Abschnitte, oder nur einige dieser Abschnitte, schreibt *csplit* in je eine Ausgabedatei. Die ursprüngliche Datei bleibt erhalten.
Mit Hilfe von Argumenten geben Sie an, wie *csplit* die Datei aufteilen und für welche Abschnitte *csplit* Ausgabedateien erstellen soll.
Pro Aufruf erstellt *csplit* höchstens 100 Ausgabedateien.

csplit[option]... datei argument ...

Keine Option angegeben
Die Ausgabedateien heißen *xx00, xx01* usw.
Auf die Standard-Ausgabe gibt *csplit* für jede erstellte Ausgabedatei die Anzahl der Zeichen aus, die in dieser Datei stehen.
Tritt ein Fehler auf, so entfernt *csplit* alle bereits erstellten Dateien.

option

-fname
Die Ausgabedateien heißen *name00, name01* usw.

-f nicht angegeben:
Die Ausgabedateien heißen *xx00, xx01* usw.

-k
Tritt ein Fehler auf, so bleiben die bereits erstellten Dateien erhalten.

-s
Die Ausgabe der Zeichenanzahl wird unterdrückt.

datei
Name der Eingabedatei.
Wenn Sie für *datei* einen Bindestrich - eingeben, liest *csplit* von der Standard-Eingabe.

argument

Sie können mehrere Argumente angeben. Jedes Argument verweist auf eine Zeile der Eingabedatei. Diese Zeilen bilden die Trennlinien zwischen den Abschnitten, in die *csplit* die Datei aufteilt: Jede solche Zeile ist die erste Zeile eines Abschnitts. Geben Sie n Argumente an, dann teilt *csplit* die Eingabedatei in $n+1$ Abschnitte auf. Die Abschnitte enthalten also jeweils folgende Zeilen:

Abschnitt	Inhalt
0	alle Zeilen vom Beginn der Eingabedatei bis (ausschließlich) zu der Zeile, auf die das 1. Argument verweist
1	alle Zeilen von der Zeile, auf die das 1. Argument verweist, bis (ausschließlich) zu der Zeile, auf die das 2. Argument verweist
.	.
.	.
n	alle Zeilen von der Zeile, auf die das *n*-te Argument verweist, bis zum Ende der Eingabedatei

Normalerweise schreibt *csplit* jeden Abschnitt in eine Ausgabedatei. Ausnahme: siehe das Argument *%regulärer_ausdruck%*[*+zahl*][*-zahl*]. Der letzte Abschnitt (Abschnitt n) wird auf jeden Fall in eine Ausgabedatei geschrieben.

csplit arbeitet die Argumente der Reihe nach ab. Zu Beginn ist die erste Zeile der Eingabedatei die aktuelle Zeile. Ist ein Argument abgearbeitet, so wird die Zeile, auf die dieses Argument verweist, zur aktuellen Zeile. Die Zeile, auf die das darauffolgende Argument verweist, muß sich in dem Bereich zwischen der aktuellen Zeile (ausschließlich) und dem Ende der Eingabedatei befinden. Z.B. muß die Zeile, auf die das zweite Argument verweist, in der Eingabedatei nach der Zeile stehen, auf die das erste Argument verweist.

argument kann sein:

/regulärer_ausdruck/[+zahl][-zahl]

Ein Argument der Form */regulärer_ausdruck/* verweist, ausgehend von der aktuellen Zeile, auf die nächste Zeile, die zu dem regulären Ausdruck paßt. *csplit* schreibt den Abschnitt von der aktuellen Zeile bis (ausschließlich) zu der Zeile, die zu dem regulären Ausdruck paßt, in eine Ausgabedatei. Dann wird die zum regulären Ausdruck passende Zeile zur aktuellen Zeile.

Zusammen mit *+zahl* oder *-zahl* verschiebt sich die Abschnittsgrenze um *zahl* Zeilen hinter (+) oder vor (-) die Zeile, die zu dem regulären Ausdruck paßt. Entsprechend wird die Zeile, die *zahl* Zeilen hinter (+) bzw. vor (-) der zum regulären Ausdruck passenden Zeile steht, zur aktuellen Zeile.

Zulässig sind einfache reguläre Ausdrücke (siehe *Tabellen und Verzeichnisse, Reguläre Ausdrücke*). Enthält das Argument Leerzeichen oder Sonderzeichen der Shell (siehe *Tabellen und Verzeichnisse, Sonderzeichen der Bourne-Shell sh*), dann müssen Sie diese Zeichen mit einem Gegenschrägstrich \ entwerten oder das Argument in Hochkommata '...' einschließen. Der reguläre Ausdruck darf keine Neue-Zeile-Zeichen enthalten.

%regulärer_ausdruck%[+zahl][-zahl]
Ein Argument der Form *%regulärer_ausdruck%* verweist, ausgehend von der aktuellen Zeile, auf die nächste Zeile, die zu dem regulären Ausdruck paßt. Die zum regulären Ausdruck passende Zeile wird zur aktuellen Zeile. *csplit* erstellt für den betreffenden Abschnitt **keine** Ausgabedatei.

Zusammen mit *+zahl* oder *-zahl* wird die Zeile, die *zahl* Zeilen hinter (+) bzw. vor (-) der zum regulären Ausdruck passenden Zeile steht, zur aktuellen Zeile.

Zulässig sind einfache reguläre Ausdrücke (siehe *Tabellen und Verzeichnisse, Reguläre Ausdrücke*). Enthält das Argument Leerzeichen oder Sonderzeichen der Shell (siehe *Tabellen und Verzeichnisse, Sonderzeichen der Bourne-Shell sh*), dann müssen Sie diese Zeichen mit einem Gegenschrägstrich \ entwerten oder das Argument in Hochkommata '...' einschließen. Der reguläre Ausdruck darf keine Neue-Zeile-Zeichen enthalten.

nr
Dieses Argument verweist auf die Zeile mit der Nummer *nr*. *csplit* schreibt den Abschnitt von der aktuellen Zeile bis (ausschließlich) zur *nr*-ten Zeile in eine Ausgabedatei. Dann wird die *nr*-te Zeile zur aktuellen Zeile.

{n}
Dieses Argument steht abkürzend für *n* Argumente der oben beschriebenen Formen und bedeutet: "Vorhergehendes Argument *n* **weitere** Male verwenden". *n* ist eine ganze Zahl größer gleich 1.

Das Argument *{n}* kann, getrennt durch Leerzeichen, auf jedes der obengenannten Argumente folgen.
Folgt es auf ein Argument der Form */regulärer_ausdruck/[+zahl][-zahl]* oder *%regulärer_ausdruck%[+zahl][-zahl]*, so wird dieses Argument *n* **weitere** Male verwendet.

Beispiel

'/reg_ausdruck/' {2}

ist eine Abkürzung für

'/reg_ausdruck/' '/reg_ausdruck/' '/reg_ausdruck/'

Folgt *{n}* auf ein Argument der Form *nr*, so wird die Datei ab der *nr*-ten Zeile *n*-mal alle *nr* Zeilen unterteilt.

Beispiel

100 {2}

ist eine Abkürzung für

100 200 300

FEHLERMELDUNGEN

`argument - out of range`

Die Zeile, auf die das Argument *argument* verweist, befindet sich außerhalb des zulässigen Bereichs. Der zulässige Bereich erstreckt sich von der aktuellen Zeile (ausschließlich) bis zum Dateiende.

`100 file limit reached at arg ...`

Sie haben so viele Argumente angeben, daß *csplit* mehr als 100 Ausgabedateien anlegen müßte.

BEISPIELE

1. Die Datei *buch* enthält einen Text, der in drei Kapitel gegliedert ist. Vor dem ersten Kapitel steht ein Vorwort, nach dem letzten ein Anhang. Jedes Kapitel beginnt mit der Überschrift "KAPITEL ..."; der Anhang hat die Überschrift "ANHANG".
   Sie wollen nun das Vorwort, die einzelnen Kapitel und den Anhang jeweils in eine eigene Datei schreiben. Die Ausgabedateien sollen *kap..* heißen.

```
$ csplit -fkap buch '/KAPITEL/' '/KAPITEL/' '/KAPITEL/' '/ANHANG/'
1636
15124
32743
20344
2576

$ ls
buch
kap00
kap01
kap02
kap03
kap04
```

   Die Datei *kap00* enthält das Vorwort; sie besteht aus 1636 Zeichen. Der Anhang steht in der Datei *kap04*.
   Der folgende, kürzere *csplit*-Aufruf hat die gleiche Wirkung:

```
$ csplit -fkap buch '/KAPITEL/' {2} '/ANHANG/'
.
.
```

   Nun können Sie die Abschnitte getrennt editieren. Danach setzen Sie sie mit *cat* wieder zusammen:

```
$ cat kap0[0-4] > buch
```

2. Eine Eingabedatei *datei* soll bei jeder hundertsten Zeile unterteilt werden. Dazu geben Sie ein:

```
$ csplit datei 100 {98}
```

Das Argument {98} steht für 98 Argumente: 200 300 ... 9900.
Enthält *datei* 9900 oder mehr Zeilen, dann erstellt *csplit* 100 Ausgabedateien. Die erste Ausgabedatei *xx00* enthält Zeile 1 bis 99 (einschließlich), die letzte Ausgabedatei *xx99* enthält den Rest von *datei* ab Zeile 9900.
Enthält *datei* weniger als 9900 Zeilen, dann gibt *csplit* die Fehlermeldung "{98} - out of range" aus und bricht ab. Geben Sie beim Aufruf die Option *-k* an, dann bleiben die bereits erstellten Dateien erhalten:

```
$ csplit -k datei 100 {98}
```

Enthält *datei* z.B. nur 9830 Zeilen, dann ist *xx98* die zuletzt erstellte Ausgabedatei und enthält die Zeilen 9800 bis 9830.

3. Die Datei *prog.c* enthält ein C-Quellprogramm. Das Programm enthält die Funktion *main* sowie höchstens 20 weitere Funktionen. Jede Funktion endet gemäß den C-Konventionen mit einer schließenden geschweiften Klammer am Anfang der Zeile (1.Spalte). Innerhalb einer Funktion stehen schließende geschweifte Klammern nicht in der 1.Spalte einer Zeile.
Jede Funktion soll in eine eigene Datei geschrieben werden. Dazu geben Sie ein:

```
$ csplit -k prog.c '%main(% '/^}/+1' {19}
```

Enthält das Programm außer der Funktion *main* genau 20 weitere Funktionen, dann teilt *csplit* die Datei in 22 Abschnitte auf.
Abschnitt 0 enthält alle Zeilen vom Beginn der Datei bis ausschließlich zum Beginn der Funktion *main*. Dieser Abschnitt wird in **keine** Ausgabedatei geschrieben (Argument *%main(%*).
Abschnitt 1 enthält die Funktion *main* und wird in die Ausgabedatei *xx00* geschrieben (Argument /^}/+1).
Entsprechend werden sukzessive die Funktionen 1 bis 19 in Ausgabedateien geschrieben (Argument {19}). Zum Schluß wird Abschnitt 22, der den Rest der Eingabedatei enthält (das ist gerade die 20. Funktion), in die Ausgabedatei *xx20* geschrieben.

Enthält das Programm weniger Funktionen, so bricht *csplit* bei der letzten Funktion mit der Fehlermeldung "{19} - out of range" ab. Da jedoch Option *-k* gesetzt ist, bleiben die erstellten Ausgabedateien erhalten.

SIEHE AUCH

ed, *sh*, *split*

ct
getty-Prozeß für ferne Datensichtstation erzeugen

ct wählt die Telefonnummer eines Modems, das an einen fernen Rechner angeschlossen ist und erzeugt einen getty-Prozeß für diese Datensichtstation.

ct[_option]_telnr...

option

-h
ct belegt die aktuelle Leitung. Der ankommende Anruf kann über diese Leitung also nicht sofort beantwortet werden. *ct* wartet die Beendigung des angegebenen Prozesses ab, bevor es zur Datensichtstation des Benutzer zurückkehrt.

-h nicht angegeben:
ct hält die aktuelle Leitung frei, so daß hereinkommende Anrufe sofort bearbeitet werden können.

-sspeed
Legt die Übertragungsleistung von *ct* fest. *speed* gibt die Übertragungsgeschwindigkeit in Baud an.

-sspeed nicht angegeben:
Die Übertragungsgeschwindigkeit beträgt standardmäßig 1200 Baud.

-v (verbose)
ct schreibt Fehlermeldungen auf die Standard-Fehlerausgabe

-wn
Wenn gerade keine freie Leitung verfügbar ist, fragt *ct*, ob und wie lange es auf eine freie Leitung warten soll, bevor es auf die Kommando-Ebene zurückkehrt. Im Minutentakt wählt *ct* die angegebene Telefonnummer, bis es das Modem erreicht oder das mit *n* spezifizierte Zeitlimit erreicht ist. *n* gibt an, wieviele Minuten *ct* höchstens auf eine freie Leitung warten soll.

-xn
ct schreibt ein detailliertes Protokoll der Programmausführung auf die Standard-Fehlerausgabe. *n* ist eine Zahl zwischen 0 und 9, die die Testebene angibt. *-x9* beinhaltet die meiste Information.

telnr

Telefonnummer des am fernen Rechner angeschlossenen Modems. Die maximale Länge von *telnr* beträgt 31 Zeichen. Als Zeichen dürfen Sie nur die Ziffern 0 bis 9 bzw. die Sonderzeichen - / = / * / # verwenden. Die Telefonnummer interpretiert bei der Herstellung einer Verbindung

- Gleichheitszeichen (=) als *Wiederholung des Wähltons*
- Minuszeichen (-) als (zeitliche) Verzögerungen
- Die Sonderzeichen (*) und (#) haben keine Sonderbedeutung, sondern gehören zur Telefonnummer.

Sie können mehrere Telefonnummern angeben, die *ct* der Reihe nach solange bearbeitet, bis eines der angewählten Modems antwortet.

Arbeitsweise

ct durchläuft alle in der Datei */usr/uucp/Devices* angeführten Leitungen, bis es eine verfügbare Leitung mit passenden Attributen findet oder bis diese Datei zu Ende ist.

Nachdem sich der Benutzer an der fernen Datensichtstation abgemeldet hat, können je nach getty-Typ (*getty* oder *uugetty*) zwei verschiedene Situationen auftreten: beim Typ *getty* wird der Benutzer gefragt, ob er wieder eine Verbindung aufnehmen will ("Reconnect?"). Beginnt die Antwort mit dem Buchstaben *n*, wird die Leitung freigegeben, ansonsten wird der Dämon *ttymon* erneut gestartet und das Bereitzeichen "login:" ausgegeben. Beim Typ *uugetty* ist bereits ein neuer Dämon *ttymon* gestartet und das Bereitzeichen"login:" wird ausgegeben. Zur korrekten Abmeldung geben Sie CTRL D ein.

Vorsicht

Wenn Sie eine bi-direktionale Leitung mit nur einer Anschlußnummer verwenden, müssen Sie für das *ttymon*-Programm die Optionen *-r* und *-b* angeben (siehe *ttymon*, *Referenzhandbuch für Systemverwalter* [7])

ct funktioniert nicht mit einer DATAKIT Multiplex-Schnittstelle.

DATEIEN

/etc/uucp/Devices

enthält Leitungen mit Attributen

SIEHE AUCH

cu, *login*, *uucp*
ttymon [7]

ctags
Markierungs-Datei erstellen (create a tags file)

ctags hilft Ihnen beim Finden von Funktions-, Makro- und Datentypdefinitionen in C-, Pascal-, FORTRAN-, YACC- und LEX-Quellprogrammen. *ctags* erstellt zu angegebenen Quellprogrammen eine Markierungsdatei, die standardmäßig den Namen *tags* bekommt.

Jede Zeile der Markierungsdatei informiert über eine Funktion bzw. ein Makro oder einen Datentyp (nur bei Option *-t*), der mit *typedef* vereinbart ist, und enthält die folgenden, durch Leerzeichen oder Tabulatorzeichen getrennten Angaben:

- den Namen der Funktion bzw. des Makros bzw. des Datentyps
- den Namen der Datei, in der die Funktion bzw. das Makro bzw. der Datentyp definiert sind
- bei einer Funktion:
  ein Textmuster, das Sie zum Suchen der Funktionsdefinition mit einem Editor verwenden können
  bei einem Makro:
  ein Textmuster, das Sie zum Suchen der Makrodefinition mit einem Editor verwenden können
  bei einem Datentyp:
  die Zeilennummer der typedef-Vereinbarung

Der Funktionsname *main* erhält in C-Programmen eine Sonderbehandlung:
ctags verwendet statt *main* den einfachen Namen der Datei, in der *main* definiert ist, und zwar ohne die Endungen *.c* oder *.h*. Diesem verkürzten Dateinamen wird der Buchstabe *M* vorangestellt. Z.B. wird die Funktion *main*, die in einer Datei */home/jannis/help/xyz.c* steht, in der Markierungdatei mit *Mxyz* bezeichnet.

Mit den Angaben in der Markierungsdatei können Editoren, wie *vi*, *ex* oder *ced*, schnell und einfach auf Funktions-, Makro- und Datentypdefinitionen positionieren.

Vorsicht

Der Algorithmus zum Erkennen von Funktionen, Unterfunktionen und Prozeduren in FORTRAN und Pascal ist sehr einfach gehalten. Blockstrukturen werden nicht berücksichtigt.

ctags erkennt keine *#ifdef*-Konstruktionen.

ctags[␣option]...[␣datei]...

Keine Option angegeben
Die Markierungsdatei erhält den Namen *tags*.
Die Textmuster stehen zwischen Schrägstrichen /, die bei Verwendung eines Editors zum Vorwärtssuchen geeignet sind. C-Programme werden nach der Definition von C-Routinen und C-Makros durchsucht.

option

-a
(a - append) *ctags* hängt die Ausgabe an das Ende der Markierungsdatei *tags* (oder *tagsdatei*, siehe Option *-f*) an. Die Markierungsdatei wird also nicht überschrieben, falls sie bereits existiert.

-B
(B - backwards) Die Textmuster stehen zwischen Fragezeichen ? und sind damit zur Rückwärtssuche geeignet.

-F
(F - forward) Die Textmuster stehen zwischen Schrägstrichen / und sind damit zur Vorwärtssuche geeignet. Diese Option ist standardmäßig gesetzt.

-f␣tagsdatei
(f - file) Die Markierungsdatei erhält den Namen *tagsdatei*.

-t
(t - typedef) *ctags* erzeugt eine Markierungsdatei, in der neben Funktions- und Makrodefinitionen auch Datentypdefinitionen (*typedef*-Definitionen) aufgeführt sind.

-t nicht angegeben:
ctags gibt nur Makro- und Funktionsdefinitionen aus.

-u
(u - update) *ctags* bringt die Informationen in der Markierungsdatei auf den neuesten Stand. D.h., für alle angegebenen Dateien werden die alten Angaben gelöscht und die aktuellen in die Markierungsdatei geschrieben. *ctags* arbeitet verhältnismäßig langsam, wenn Sie diese Option verwenden. Es ist deshalb in der Regel sinnvoller, die Markierungsdatei neu zu erstellen.

-v
(v - verbose) *ctags* schreibt auf die Standard-Ausgabe eine lexikographisch geordnete Liste, die zu jedem Funktionsnamen die Datei und die Seitennummer angibt, in der die Funktion definiert wird. Bei der Berechnung der Seitennummer wird von 64 Zeilen pro Seite ausgegangen.

-w

(w - warning) *ctags* unterdrückt Warnungen.

-x

(x - index) *ctags* schreibt nicht in eine Markierungsdatei. Stattdessen gibt *ctags* auf die Standard-Ausgabe ein Verzeichnis mit Informationen über die angegebenen Dateien aus. Eine Ausgabezeile enthält folgende Angaben:

- den Namen der Funktion bzw. des Datentyps
- die Zeilennummer der Funktions- bzw. Datentypdefinition
- den Namen der Datei, in der diese Definition steht
- den Text der Definitionszeile.

main erhält keine Sonderbehandlung. Bei Namensgleichheit von Funktionen erfolgen Warnungen.

Zusammen mit Option *-t*:
ctags gibt nur die letzte Zeile der Datentypdefinitionen aus.

datei

Name der Datei, die *ctags* durchsuchen soll.
Pro Aufruf können Sie mehrere Dateinamen angeben. In diesem Fall schreibt *ctags* die Informationen in eine Markierungsdatei.

Abhängig von der Endung des Dateinamens vermutet *ctags* folgendes Quellprogramm:

.c und .h	C-Quellprogramm
.y	YACC-Quellprogramm
.l	LEX-Quellprogramm
sonstige	Pascal- oder FORTRAN-Quellprogramm, falls die Definition einer Funktion, eines Unterprogramms oder einer Prozedur in der Datei steht, die Pascal- bzw. FORTRAN-spezifisch ist. C-Quellprogramm, falls keine solche Definition vorhanden ist.

FEHLERMELDUNG

`Duplicate entry in files` *datei1* `and` *datei2*`: ...`

Namensgleichheit eines Objekts in *datei1* und *datei2*.

DATEI

tags

Markierungsdatei, in die *ctags* standardmäßig schreibt

BEISPIEL

In der Datei *programm.c* steht folgendes C-Programm:

```
#define ENDE 20

main()
{
    int i;
    for (i=0; i<ENDE; i++)
        ausgabe(i);
}

ausgabe(i)
int i;
{
    printf("Das Quadrat von %d lautet %d\n", i, quadrat(i));
}

int quadrat(i)
int i;
{
    return(i*i);
}
```

```
$ ctags -x programm.c
  ausgabe        10 programm.c     ausgabe(i)
  main            3 programm.c     main()
  quadrat        17 programm.c     int quadrat(i)
```

SIEHE AUCH

ex, *vi*

cu
Verbindung zu einem anderen UNIX-System aufbauen (call unix)

cu stellt Verbindung zu einem anderen UNIX-System, zu einer Datensichtstation oder einem Rechner mit einem anderen Betriebssystem als UNIX her. *cu* läuft in zwei Phasen ab: In der ersten Phase, der Verbindungsphase, wird die Verbindung aufgebaut. In der zweiten Phase, der Kommunikationsphase, werden Daten vom und zum fernen System übertragen.

cu[␣optionen][␣ziel]

Verbindungsphase

cu[␣**-s**speed][␣**-c**typ][␣**-l**gerätedatei][␣**-b**n][␣**-dehnot**]

cu baut eine Verbindung auf die gleiche Art und Weise wie *uucp* auf, d.h. unter Verwendung der *uucp*-Steuerdateien */etc/uucp/Devices* und */etc/uucp/Systems*. Dadurch kann *cu* die Verbindung über verschiedenartige Medien aufbauen. Mögliche Medien sind Telefonleitungen, direkte Verbindungen und lokale Rechnernetze (local area networks, LAN). Die Datei */etc/uucp/Devices* enthält eine Liste der Medien, die auf Ihrem System verfügbar sind. Die Datei */etc/uucp/Systems* enthält Informationen für die Verbindung mit fernen Systemen, sie ist jedoch aus Sicherheitsgründen nicht allgemein lesbar.

Mit den Optionen *-s*, *-c* und *-l* beeinflussen Sie die Auswahl des Übertragungsmediums, mit den übrigen Optionen die Einstellungen der Verbindung.

option

-sspeed
: *speed* ist die Übertragungsgeschwindigkeit in Baud. Mögliche Werte sind 300, 1200, 2400, 4800 und 9600. Der Standardwert hängt von der Reihenfolge der Einträge in der Datei */etc/uucp/Devices* ab. Die meisten Modems arbeiten mit Übertragungsgeschwindigkeiten von 300, 1200 oder 2400 Baud. Bei direkten Verbindungen können Sie auch Übertragungsgeschwindigkeiten über 2400 Baud verwenden.

-ctyp

Das Typenfeld ist das erste Feld in der Datei */etc/uucp/Devices*. Mit der Option *-c* wird *cu* gezwungen, nur solche Einträge im Typenfeld zu verwenden, die mit *typ* übereinstimmen. *typ* ist normalerweise der Name eines lokalen Rechnernetzes (LAN).

Ist *typ* der Name eines lokalen Rechnernetzess (LAN), betrachtet *cu ziel* als Name eines Systems. Falls keine Verbindung zu einem System mit diesem Namen aufgebaut werden kann, wird *ziel* als LAN-Adresse interpretiert.

Verwenden Sie *-c* nicht zusammen mit *-l*.

-lgerätedatei

Mit *-l* können Sie eine bestimmte Verbindungsleitung festlegen. *gerätedatei* ist der Name einer Gerätedatei, die für die Verbindung verwendet werden soll. Damit entfällt die Suche nach der ersten verfügbaren Leitung mit der richtigen Geschwindigkeit.

Wenn Sie *-l* ohne die Option *-s* verwenden, wird die Übertragungsgeschwindigkeit aus der Datei */etc/uucp/Devices* ermittelt. Es wird der Eintrag verwendet, bei dem das zweite Feld mit *gerätedatei* übereinstimmt. Das zweite Feld kennzeichnet in dieser Datei jeweils die Übertragungsleitung.
Wenn Sie *-l* und *-s* zusammen verwenden, überprüft *cu* anhand der Datei */etc/uucp/Devices*, ob die bei *-s* angegebene Übertragungsgeschwindigkeit verfügbar ist. Falls ja, wird eine Verbindung mit dieser Geschwindigkeit aufgebaut. Anderenfalls wird eine Fehlermeldung ausgegeben und keine Verbindung aufgebaut.

Üblicherweise kennzeichnet *gerätedatei* eine direkte Verbindung durch eine serielle Leitung (z.B. */dev/term/ab*). In diesem Fall ist die Angabe einer Telefonnummer nicht nötig. Die angegebene Gerätedatei muß sich nicht im Verzeichnis */dev* befinden. Wenn die angegebene Gerätedatei eine automatische Wähleinheit bezeichnet, muß eine Telefonnummer angegeben werden. Falls der Parameter *ziel* zusammen mit dieser Option angegeben wird, muß er eine Telefonnummer bezeichnen.

-l sollten Sie nicht zusammen mit *-c* verwenden.

-bn

Auf der Leitung werden *n* Bits übertragen. *n* hat entweder den Wert 7 oder 8. Damit kann eine Verbindung zwischen Systemen mit verschiedenen Zeichengrößen hergestellt werden.

-b nicht angegeben:

Als Größe eines übertragenen Zeichens wird der gleiche Wert wie bei der lokalen Datensichtstation verwendet.

-d

cu gibt Diagnosemeldungen zur Erleichterung der Fehlersuche aus.

-e

(e - even parity) Für Daten, die zum fernen System übertragen werden, wird gerade Parität erzeugt.

-h

Die Übertragung findet im Duplex-Modus statt. Bei dieser Option wird das lokale Kommando *echo* emuliert. Damit können Verbindungen zu anderen Rechnersystemen aufgebaut werden, bei denen Datensichtstationen im Halb-Duplex-Modus betrieben werden.

-n

Die Telefonnummer wird interaktiv abgefragt. Anstatt die Telefonnummer von der Kommandozeile zu übernehmen, wird der Benutzer aufgefordert, die Nummer einzugeben.

-o

(o - odd parity) Für Daten, die zum fernen System übertragen werden, wird ungerade Parität erzeugt.

-t

Damit wird eine Datensichtstation angewählt, die auf automatische Antwort eingestellt ist. Wagenrücklauf-Zeichen werden durch die Kombination Wagenrücklauf-Zeichen - Zeilenvorschub ersetzt.

ziel

Mit dem Parameter *ziel* teilen Sie *cu* mit, zu welchem System es eine Verbindung aufbauen soll. *ziel* kann sein:

- eine Telefonnummer: Eine Telefonnummer darf aus den Ziffern *0* bis *9*, den Zeichen Stern * und Nummernzeichen sowie den Sonderzeichen = und - bestehen. Das Gleichheitszeichen bezeichnet den zweiten Wahlton, das Minuszeichen bezeichnet eine Verzögerung von vier Sekunden.
- ein Systemname: Als Namen eines Systems können Sie den Namen jedes Systems, das mit *uucp* angesprochen werden kann, angeben.
- eine Rechneradresse: Die Adresse eines Rechners im lokalen Netz. Das Das Format der LAN-Adressen ist in der Dokumentation zu ihrem lokalen Rechnernetz beschrieben.

Wenn Sie die Optionen *-c* und *-l* nicht angeben, benutzt *cu* den Parameter *ziel*, um festzustellen, welches Übertragungsmedium verwendet werden soll. Folgende Fälle sind möglich:

- *ziel* ist eine Telefonnummer. *cu* nimmt dann an, daß Sie eine Telefonleitung benutzen wollen und arbeitet mit einer automatischen Wähleinheit (automatic call unit - ACU).
- *ziel* ist keine Telefonnummer. *cu* nimmt an, daß *ziel* ein Systemname ist. *cu* benutzt dann den Vebindungsmechanismus von *uucp* und verwendet die Dateien *etc/uucp/Systems* und */etc/uucp/Device*, um die beste verfügbare Verbindung aufzubauen. *cu* wählt automatisch die richtige Übertragungsgeschwindigkeit für das ausgewählte Medium. In diesem Fall dürfen Sie die Option *-s* nicht angeben.

ziel nicht angegeben:

Dies ist nur möglich, wenn Sie mit der Option *-l* den Namen einer Gerätedatei für eine direkte Verbindung angegeben haben.

Kommunikationsphase

Nachdem die Verbindung aufgebaut wurde, laufen zwei Prozesse von *cu*: Der Übertragungsprozeß und der Empfangsprozeß.
Der Übertragungsprozeß liest Daten von der Standard-Eingabe und überträgt sie zum fernen System. Der Empfangs-Prozeß übernimmt Daten vom fernen System und gibt sie auf die Standard-Ausgabe aus. Sie können die Ausgabe jedoch auch in lokale Dateien umleiten. Zeilen, die mit dem Tilde-Zeichen beginnen, haben ein spezielle Bedeutung (siehe unten). Normalerweise wird automatisch ein DC3/DC1-Protokoll (START/STOP - Kontrolle) zur Steuerung der vom fernen System empfangenen Daten verwendet, um ein Überlaufen des Puffers zu verhindern.

cu überprüft die übertragenen Daten nicht. Datenfelder mit speziellen *cu*-Steuerzeichen werden unter Umständen nicht korrekt übertragen. Je nach Verbindung kann es erforderlich sein, die Verbindung mit dem kommando ~. (siehe unten) zu beenden, auch wenn *stty 0* benutzt wurde.

Mit Verbindungen zwischen mehreren Systemen können Sie folgendermaßen arbeiten: Wenn mit *cu* eine Verbindung zum System Y und auf dem System Y eine Verbindung zum System Z aufgebaut wird, können *cu*-Kommandos auf dem System Y ausgeführt werden, indem sie mit ~~ eingeleitet werden. Mit *uname* erfahren Sie den Namen des jeweils ausführenden Systems. Auf den Systemen Z, Y, und X kann *uname* zum Beispiel folgendermaßen ausgeführt werden:

```
uname
Z
~[X]!uname
X
~~[Y]!uname
Y
```

Ein führendes ~ Zeichen bewirkt, daß das Kommando auf dem lokalen System ausgeführt wird. Zwei führende ~ Zeichen bewirken, daß das Kommando auf dem nächsten System in der Kette ausgeführt wird.

Sie können den Übertragungsprozeß mit folgenden Befehlen steuern:

~.
Die Verbindung wird abgebaut.

~!
Auf dem lokalen System wird eine Subshell aufgerufen.

~!kommando
Das Kommando wird auf dem lokalen System mit *sh -c* ausgeführt, d.h., es wird über eine Subshell ausgeführt.

~$kommando
Das Kommando wird auf dem lokalen System ausgeführt, die Ausgabe des Kommandos wird jedoch zum fernen System übertragen.

~**%cd**[dateiverzeichnis]
Auf dem lokalen System wird in das angegebene Dateiverzeichnis gewechselt. Mit *~!cd* können Sie das Dateiverzeichnis nicht wechseln, da dieser Aufruf in einer eigenen Subshell ausgeführt wird und daher keine Wirkung für den *cu*-Prozeß hat.

~**%take**␣quelle[␣ziel]
Die Datei *quelle* auf dem fernen System wird auf die Datei *ziel* auf dem lokalen System kopiert.

ziel nicht angegeben:

quelle wird auch als Name der Zieldatei verwendet.

Für das Kommando *~%take* werden auf dem fernen System die Kommandos *echo* und *cat* benötigt. Außerdem muß auf dem fernen System der *tabs*-Modus eingestellt sein (siehe *stty*), wenn Tabulatorzeichen nicht in Leerzeichen expandiert werden sollen. Nicht druckbare Zeichen werden von *~%take* nicht zuverlässig übertragen.
~%take kann nicht für Verbindungen über mehrere Systeme benutzt werden. Dateien müssen jeweils über einzelne Verbindungen übertragen werden.

~**%put**␣quelle[␣ziel]
Die Datei *quelle* auf dem lokalen System wird auf die Datei *ziel* auf dem fernen System kopiert.

ziel nicht angegeben:

quelle wird auch als Name der Zieldatei verwendet.

Für das Kommando *~%put* werden auf dem fernen System die Kommandos *stty* und *cat* benötigt. Außerdem müssen die Steuerzeichen *erase* und *kill* auf dem fernen System auf die gleichen Werte wie auf dem lokalen System eingestellt sein (*stty*). Wo es nötig ist, werden Gegenschrägstriche eingefügt. Nicht druckbare Zeichen werden von *~%put* nicht zuverlässig übertragen.
~%put kann nicht für Verbindungen über mehrere Systeme benutzt werden. Dateien müssen jeweils über einzelne Verbindungen übertragen werden.

~~␣zeile
Die Zeile *~␣zeile* wird zum fernen System übertragen.

~**%break**
Ein *BREAK*-Signal wird zum fernen System geschickt.

~**%b**
wie *~%break*

~**%debug**
Die Option *-d* zur Fehlersuche wird aus-/eingeschaltet.

~**%d**
wie *~%debug*

~t
Die Werte der Struktur-Variablen von *termio* (siehe *termio* [7]) werden für die benutzte Datensichtstation ausgegeben. Dies kann bei der Fehlersuche nützlich sein.

~l
Die Werte Struktur-Variablen von *termio* werden für die Verbindungsleitung zum fernen System ausgegeben. Dies kann bei der Fehlersuche nützlich sein.

~%ifc
Es wird zwischen DC3/DC1-Protokoll (START/STOP-Protokoll) zur Steuerung der eingehenden Daten und Datenfluß ohne Steuerung umgeschaltet. Dies ist nützlich, wenn das ferne System nicht korrekt auf die Zeichen DC3 und DC1 reagiert.

~%nostop
wie *~%ifc*

~%ofc
Die Flußkontrolle für ausgehende Daten wird ein- bzw. ausgeschaltet, je nach augenblicklicher Einstellung. Im eingeschalteten Zustand ist eine Flußkontrolle der ausgehenden Daten durch das ferne System möglich.

~%noostop
wie *ofc*

~%divert
Umleitungen, die nicht durch *%take* spezifiziert wurden, werden zugelassen/nicht zugelassen.

~%old
Alte Syntax für erhaltene Umleitungen ist erlaubt/nicht erlaubt.

Hinweis:
Bei dem Kommando *put* wird die Übertragung von *cu* künstlich verlangsamt, damit ein Datenverlust unwahrscheinlich ist. Dateien, die mit dem Kommando *take* oder *put* übertragen werden, müssen mit einem <LF> enden. Anderenfalls bleibt das Kommando hängen. Diese Situation kann normalerweise mit CTRL D beendet werden.

DATEIEN

/etc/uucp/Sysfiles
/etc/uucp/Systems
/etc/uucp/Devices
*/var/spool/locks/**

BEISPIELE

1. Sie wollen ein System mit der Telefonnummer 9 1 201 555 1234 anwählen, wobei die Übertragungsgeschwindigkeit 1200 Baud verwendet werden soll. Nach der Ziffer 9 wird ein Wählton erwartet:

   ```
   cu -s1200 9=12015551234
   ```

2. Sie wollen sich auf einem System in einem Datakit VCS Netzwerk anmelden, das jedoch von Ihrem Systemverwalter nicht definiert wurde, d.h., es gibt in der Datei */etc/uucp/Systems* keinen Eintrag dafür:

   ```
   cu -cDK adresse
   ```

   Dabei ist *DK* der Name des lokalen Netzwerks und *adresse* ist die Datakit-Adresse in der Form */area/exchange/machine*.

3. Sie wollen sich auf einem System anmelden, das über eine direkte Leitung angeschlossen ist, XX steht für einen beliebigen Devicenamen:

   ```
   cu -l /dev/term/XX
   ```

   oder

   ```
   cu -l term/XX
   ```

4. Sie wollen ein System über eine bestimmte Leitung anwählen, die mit einer automatischen Wähleinheit ausgestattet ist:

   ```
   cu -l culXX 9=12015551234
   ```

5. Sie wollen eine Verbindung zu dem System *name* aufbauen:

   ```
   cu name
   ```

6. Das ferne System muß nicht unbedingt ein UNIX-System sein. Sie können z.B. auch zu einem PostScript-Drucker im Dialog-Modus eine Verbindung herstellen und dann interaktiv PostScript-Anweisungen eingeben. Der PostScript-Drucker sei mit der Gerätedatei */dev/lpPS* verbunden und auf eine Übertragungsgeschwindigkeit von 9600 Baud sowie gerade Parität eingestellt. Die Druckerverwaltung muß für diesen Drukker deaktiviert sein.

   Mit dem folgenden Kommando wird die Verbindung aufgebaut:

   ```
   cu -l /dev/lpPS -s9600 -e
   ```

   Anschließend können Sie entweder direkt PostScript-Anweisungen eingeben oder mit dem Kommando *$cat datei* eine Datei mit PostScript-Anweisungen zum Drucker schicken. Beachten Sie, daß Sie bei vielen PostScript-Druckern erst das Schlüsselwort *executive* eingeben müssen, bevor Sie im Dialog-Modus arbeiten können. Dieses Schlüsselwort wird nicht angezeigt, d.h. Sie müssen "blind" tippen.

SIEHE AUCH

cat, *ct*, *echo*, *stty*, *uucp*, *uname*, *uuname*

Referenzhandbuch für Systemverwalter [7]

cut
Felder oder Spalten aus den Zeilen einer Datei herausschneiden

cut liest einen Eingabetext zeilenweise aus Dateien oder von der Standard-Eingabe und schneidet aus den Zeilen bestimmte Spalten (Format 1) oder Felder (Format 2) heraus. Die herausgeschnittenen Spalten bzw. Felder gibt *cut* auf die Standard-Ausgabe aus.

cut␣**-c**liste[␣datei]... Format 1
cut␣**-f**liste[␣**-d**zeichen][␣**-s**][␣datei]... Format 2

Format 1: Spalten herausschneiden

cut␣**-c**liste[␣datei]...

-cliste
(c - column) *cut* schneidet aus jeder Eingabezeile die in *liste* angegebenen Spalten heraus und gibt sie auf die Standard-Ausgabe aus. Eine Spalte ist genau ein Zeichen breit.

liste ist eine Liste von Zahlen oder Zahlenbereichen. Die Elemente der Liste müssen durch Kommas getrennt und in aufsteigender Reihenfolge angeordnet sein. Ein Bereich *n1-n2* bezieht sich auf alle Zahlen von *n1* bis *n2* einschließlich.

Bei Bereichsangaben sind folgende Kurzformen zulässig:
-n für 1-*n*
n- für *n*-"letzte Spalte"

Beispiel

1,3,5	*cut* schneidet jeweils die 1., 3. und 5. Spalte heraus.
1-3,5	*cut* schneidet jeweils die 1., 2., 3. und 5. Spalte heraus.
-3,5	ist Kurzform für 1-3,5
3-	ist Kurzform für 3-"letzte Spalte"

datei
Name der Eingabedatei.
Sie können mehrere Dateien angeben.

datei nicht angegeben:
cut liest von der Standard-Eingabe.

Format 2: Felder herausschneiden

cut␣**-f**liste[␣**-d**zeichen][␣**-s**][␣datei]...

-fliste
(f - field) *cut* schneidet aus jeder Eingabezeile die in *liste* angegebenen Felder heraus und gibt sie auf die Standard-Ausgabe aus.
Ein Feld besteht aus den Zeichen, die zwischen zwei Feldtrennzeichen stehen. Zwei aufeinanderfolgende Feldtrennzeichen begrenzen ein leeres Feld. Standard-Feldtrennzeichen ist das Tabulatorzeichen. Mit der Option *-d* können Sie das Feldtrennzeichen umdefinieren.
Die ausgegebenen Felder sind durch je ein Feldtrennzeichen voneinander getrennt.
Eingabezeilen ohne Feldtrennzeichen werden normalerweise vollständig ausgegeben (Ausnahme: Option *-s*). Dies ist bei Tabellentiteln nützlich.

liste hat die unter *Format 1* beschriebene Form.

-dzeichen
Feldtrennzeichen ist das Zeichen *zeichen*.
Ist *zeichen* das Leerzeichen oder ein Shell-Sonderzeichen (siehe *Tabellen und Verzeichnisse, Sonderzeichen der Bourne-Shell sh*), dann müssen Sie *zeichen* in Hochkommata einschließen: -d'*zeichen*'.

-d nicht angegeben:
Feldtrennzeichen ist das Tabulatorzeichen.

-s
Zeilen ohne Feldtrennzeichen werden nicht ausgegeben.

-s nicht angegeben:
Zeilen ohne Feldtrennzeichen werden vollständig ausgegeben.

datei
Name der Eingabedatei.
Sie können mehrere Dateien angeben.

datei nicht angegeben:
cut liest von der Standard-Eingabe.

FEHLERMELDUNGEN

`ERROR: line too long`

Eine Zeile kann maximal 1023 Zeichen oder Felder lang sein. Möglicherweise kann auch das Neue-Zeile-Zeichen fehlen.

`ERROR: bad list for c/f option`

Falsch angegebene *liste* oder fehlende Option -c oder *-f*. Es wird kein Fehler gemeldet, wenn die Zeile weniger Felder hat, als *liste* fordert.

`ERROR: no fields`

liste ist leer.

`ERROR: no delimiter`

Das Feldtrennzeichen zur Option *-d* fehlt.

`ERROR: cannot handle multiple adjacent backspaces`

Angrenzende Rückschritt-Zeichen können nicht richtig abgearbeitet werden.

`WARNING: cannot open` *datei*

Entweder existiert die Datei *datei* nicht oder sie kann nicht gelesen werden. Haben Sie mehrere Dateien angegeben, dann wird mit deren Bearbeitung fortgefahren.

BEISPIELE

1. Die ersten 72 Zeichen jeder Eingabezeile ausgeben:

   ```
   $ cut -c1-72 datei
   ```

2. Aus den Zeilen der Datei */etc/passwd* sollen jeweils das erste (Benutzerkennung) und dritte Feld (UID) herausgeschnitten und ausgegeben werden. Die Felder in dieser Datei sind durch einen Doppelpunkt getrennt.

   ```
   $ cut -f1,3 -d: /etc/passwd
   ```

3. Aus der Datei *rechnung* eines Versandhauses sollen die Namen der Kunden, deren Rechnung im Mai 88 fällig ist, zusammen mit dem Rechnungsbetrag herausgefiltert werden.
   Die Datei hat folgenden Inhalt:

```
Becker          Muenchen        10.000          13.05.88
Brauer          Nuernberg        7.000          01.07.88
Drechsler       Hamburg          8.000          07.05.88
Ebersbusch      Stuttgart          450          20.06.88
```

   Die Felder der Tabelle sind durch genau ein Tabulatorzeichen voneinander getrennt und mit Leerzeichen aufgefüllt.

```
$ grep '\.05\.88' rechnung | cut -f1,3 > namen
$ cat namen
Becker          10.000
Drechsler        8.000
```

   Erläuterung: *grep* schreibt alle Zeilen der Datei, die die Zeichenkette .05.88 enthalten, auf die Standard-Ausgabe. Diese Zeilen erhält *cut* als Eingabe und schneidet aus ihnen das erste und dritte Feld heraus. Die Ausgabe von *cut* wird in die Datei *namen* geschrieben.

   Wollen Sie den Kundennamen in der Datei *namen* zusätzlich das Datum voranstellen und dann das Ergebnis nach dem Datum sortieren, dann geben Sie ein:

```
$ grep '\.05\.88' rechnung | cut -f4 > datum
$ paste datum namen | sort
07.05.88        Drechsler        8.000
13.05.88        Becker          10.000
```

   Erläuterung: Mit der ersten Kommandozeile schreiben Sie das Datum, das zu den ausgewählten Kundennamen gehört, in die Datei *datum*. Der *paste*-Aufruf fügt die Zeilen der Dateien *datum* und *namen* horizontal, getrennt durch ein Tabulatorzeichen, zusammen. *sort* sortiert die Ausgabezeilen aufsteigend nach dem Datum.

SIEHE AUCH

awk, *grep*, *paste*, *sh*

date
Datum und Uhrzeit ausgeben oder Systemuhr stellen

date schreibt das aktuelle Datum und die Uhrzeit auf die Standard-Ausgabe (Format 1). Das Format, in dem *date* Datum und Uhrzeit ausgibt, hängt vom Wert der Umgebungsvariablen *LC_TIME* bzw., wenn diese leer oder nicht definiert ist, vom Wert der Umgebungsvariablen *LANG* ab. Sind *LC_TIME* und *LANG* beide leer bzw. nicht definiert, ist die entsprechende Datenbasis nicht vorhanden oder hat eine der NLS-Umgebungsvariablen einen ungültigen Wert, dann verhält sich *date*, als wäre das System nicht internationalisiert, d.h., es gibt Datum und Uhrzeit im amerikanischen Format aus.

Als Systemverwalter können Sie mit *date* außerdem die Systemuhr stellen (Format 2). Dies sollten Sie aber nur im Einbenutzer-Betrieb tun, wenn alle Dateisysteme abgehängt sind. Andernfalls kann es zu Inkonsistenzen in den Dateisystemen kommen.

Arbeitsweise

Das System arbeitet mit der Weltzeit UTC (Universal Time Coordinated). *date* wandelt die UTC in die Ortszeit um und umgekehrt. Ist die Umgebungsvariable *TZ* definiert und die Option -u nicht gesetzt, so wird sie zur Bestimmung der Zeitzone bzw. zur Umrechnung der UTC in die Ortszeit verwendet (siehe *sh*, *Standard-Variablen der Shell*).

date[␣**-u**][␣+format] Format 1
date[␣**-a**[␣-]␣sss.fff][␣**-u**][␣neues_datum] Format 2

Format 1: Datum und Uhrzeit ausgeben lassen

date[␣**-u**][␣+format]

Kein Argument angegeben
date gibt das aktuelle Datum und die Uhrzeit in der aktuell gültigen internationalisierten Umgebung aus.

-u
date gibt das aktuelle Datum und die Uhrzeit in Greenwich-Zeit aus.
Wenn die Option *-u* angegeben ist, dann wird die Angabe *+format* ignoriert.

+format

Mit dem Argument *format* bestimmen Sie das Ausgabeformat von *date*. Enthält *format* Leer- bzw. Tabulatorzeichen oder sonstige Sonderzeichen der Shell, die die Shell nicht interpretieren soll, dann schließen Sie *format* in Hochkommas ein: +'format'.

format ist dem Format des ersten Arguments der C-Funktion bzw. der *awk*-Funktion *printf()* ähnlich (siehe *awk*, *printf* und C-Funktion *printf()* [19]).

format erlaubt es, mit Hilfe von Feldbezeichnern die Ausgabe von *date* zu formatieren.
Feldbezeichner sind von der Form *%buchstabe*. Sie werden bei der Ausgabe durch ihren Wert ersetzt. Alle Zeichen, die nicht Teil eines Feldbezeichners sind, werden unverändert ausgegeben. Am Schluß der Ausgabe wird auf jeden Fall ein Neue-Zeile-Zeichen ausgegeben.

In der folgenden Übersicht sind die möglichen Feldbezeichner aufgelistet. Zwischen den Feldbezeichnern %h und %b besteht kein Unterschied; aus Kompatibilitätsgründen wurden jedoch beide Bezeichner beibehalten.

%n Neue-Zeile-Zeichen
%t Tabulatorzeichen

%c Datum und Uhrzeit im Default-Format der aktuell gültigen internationalisierten Umgebung
%C Datum und Uhrzeit im Standard-Format der aktuell gültigen internationalisierten Umgebung

%D Datum im Format %m/%d/%y
%x Datum im Format der aktuell gültigen internationalisierten Umgebung

%y Jahr (2stellig, die letzten beiden Ziffern der Jahreszahl, 00-99)
%Y Jahr (4stellig, alle Ziffern der Jahreszahl)

%m Monat (01 bis 12)
%h Monat (in Buchstaben, abgekürzt)
im Format der aktuell gültigen internationalisierten Umgebung
%b wie %h
%B Monat (in Buchstaben, ausgeschrieben)
im Format der aktuell gültigen internationalisierten Umgebung

%W Woche im Jahr (00 bis 53, Montag ist der erste Tag in der Woche)
%U Woche im Jahr (00 bis 53, Sonntag ist der erste Tag in der Woche)

%j Tag im Jahr (001 bis 366)
%d Tag im Monat (01 bis 31)
%e Tag im Monat (1 bis 31), bei einstelligen Datumsangaben wird ein Blank vorangestellt

%a Wochentag (in Buchstaben, abgekürzt) im Format der aktuell gültigen internationalisierten Umgebung
%A Wochentag (in Buchstaben, ausgeschrieben) im Format der aktuell gültigen internationalisierten Umgebung
%w Wochentag (0 bis 6, Sonntag = 0)

%R Uhrzeit im Format %H:%M
%T Uhrzeit im Format %H:%M:%S
%X Uhrzeit im Format der aktuell gültigen internationalisierten Umgebung
%r Uhrzeit in 12-Stunden-Notation: %I:%M:%S %p

%H Stunde (00 bis 23)
%I Stunde (01 bis 12)
%p Ante-meridiem- bzw. Post-meridiem-Affix im Format der aktuell gültigen internationalisierten Umgebung

%M Minute (00 bis 59)

%S Sekunde (00 bis 61)

%Z Name der Zeitzone oder keine Ausgabe, falls keine Zeitzone vorhanden ist (abhängig von der Umgebungsvariablen *TZ*, siehe *sh*, *Standard-Variablen der Shell*)

Format 2: Systemuhr stellen

date[␣**-a**[␣-]␣sss.fff][␣**-u**][␣neues␣datum]

Dieses Format ist nur für den Systemverwalter bestimmt.

Mit diesem Format können Sie entweder die Uhr nachstellen oder das Datum neu setzen.

-a[␣-]␣sss.fff
Die Uhr des Systems wird *sss* Sekunden und *fff* Sekundenbruchteile nachgestellt. Die Uhr kann vor oder zurück (-) gestellt werden. Die Korrektur wird durchgeführt, indem die Systemuhr so lange beschleunigt oder verlangsamt wird, bis die angegebene Differenz ausgeglichen ist.

-u
Datum und Uhrzeit werden in Greenwich-Zeit gesetzt (siehe auch Format 1, -u).

neues_datum
[mmdd]HHMM | mmddHHMM[cc]yy *date* setzt das Datum und die Uhrzeit des Systems auf den angegebenen Wert. Für *[mmdd]HHMM* oder *mmddHHMM[cc]yy* geben Sie an:

mm	dd	HH	MM	cc	yy
Monat (01 - 12)	Tag (01 - 31)	Stunde (00 - 23)	Minute (00 - 59)	Jahrhundert - 1	Jahr (letzte 2 Ziffern)

mmdd nicht angegeben:
date bezieht die angegebene Zeit *HHMM* auf den aktuellen Tag.

cc nicht angegeben:
date bezieht das angegebene Datum *mmddHHMMyy* auf das aktuelle Jahrhundert.

Die Systemuhr sollten Sie nur im Einbenutzer-Betrieb stellen, wenn alle Dateisysteme abgehängt sind. Andernfalls kann es zu Inkonsistenzen in den Dateisystemen kommen.

FEHLERMELDUNGEN

Format 2

```
date: no permission.
```

Sie wollten als normaler Benutzer die Systemuhr stellen. Dies darf jedoch nur der Systemverwalter.

```
date: bad conversion.
```

Sie wollten die Systemuhr stellen, haben *date* jedoch im falschen Format aufgerufen.

UMGEBUNGSVARIABLEN

CFTIME
beeinflußt das Format, in dem das Datum ausgegeben wird. Für *CFTIME* wird ein Ausgabeformat in der Form definiert, wie unter *Format1, +format* beschrieben (siehe *Beispiele, Format 1*).

LANG
beeinflußt den Zeichensatz und die Sprache der Meldungstexte. Ist die Umgebungsvariable *LC_TIME* leer oder nicht definiert, dann beeinflußt *LANG* zusätzlich die Sprache und das Format, in dem das Datum und die Uhrzeit bzw. der Wert der Datums- und Zeitkonstanten ausgegeben werden.

LC_TIME
beeinflußt die Sprache und das Format, in dem das Datum und die Uhrzeit bzw. der Wert der Datums- und Zeitkonstanten ausgegeben werden.

TZ
Die Umgebungsvariable *TZ* enthält, falls sie definiert ist, Informationen über die Zeitzonen. *date* benutzt die Variable *TZ* zur Bestimmung der Zeitzone bzw. zur Umrechnung der Weltzeit UTC (Universal Time Coordinated) in die Ortszeit und umgekehrt. Der Wert von *TZ* besteht aus

- der Standard-Zeitzone,
- der Differenz zur UTC in Stunden und
- evtl. der Sommerzeitzone mit Zeitangaben zur Umstellung von Standard-Zeit auf Sommerzeit und zurück

Eine detaillierte Beschreibung zu *TZ* finden Sie bei *sh*, *Standard-Variablen der Shell*.

BEISPIELE

Datum und Uhrzeit ausgeben lassen (Format 1)

Wenn Sie am 15. Juli 1991 um 17 Uhr MDT *date* ohne Argument aufrufen, erhalten Sie, falls die Systemuhr die richtige Zeit angibt, die Ausgabe

```
Mon Jul 15 17:00:00 MDT 1991
```

Mit der Kommandozeile

```
$ date '+DATE: %m/%d/%y%nTIME: %H:%M:%S'
```

erzeugen Sie folgende Ausgabe:

```
DATE: 07/15/91
TIME: 17:00:00
```

Sie haben die Umgebungsvariable *CFTIME* definiert und lassen sich den Wert von *CFTIME* mit *echo* ausgeben:

```
$ echo $CFTIME

%d %B %y
```

Wenn Sie am 15. Juli 1991 um 17 Uhr MET *date* ohne Argument aufrufen, erhalten Sie nun, falls die Systemuhr die richtige Zeit angibt, die Ausgabe

```
15 July 91
```

Systemuhr stellen (Format 2)

Sie arbeiten unter der Systemverwalter-Kennung *root*. Sie möchten das Datum und die Uhrzeit auf den 17. März 1989, 16 Uhr 30 setzen. Dazu gehen Sie in den Einbenutzer-Betrieb und hängen alle Dateisysteme ab. Dann geben Sie ein:

```
# date 0317163089
```

Rufen Sie *date* im Jahr 1989 auf, dann können Sie die Jahresangabe weglassen.

SIEHE AUCH

cal
sysadm [7]
strftime, *environ*, *ctime()*, *printf()* [19]

Leitfaden für Systemverwalter, Einstellen der Systemuhr [6]

dc
Tischrechner (desk calculator)

Mit *dc* können Sie im Dialog mit SINIX die Funktionen eines Tischrechners nutzen.

dc[_datei]

datei
Name der Datei, in der die Rechenoperationen stehen, die ausgeführt werden sollen. *dc* liest *datei* bis zu ihrem Ende, dann von der Standard-Eingabe.

datei nicht angegeben:
dc liest von der Standard-Eingabe

ARBEITSWEISE VON DC

dc arbeitet nach dem Prinzip eines Stapelspeichers (stack). Ein Stapelspeicher ist ein linearer Speicher, auf den nur nach dem Prinzip LIFO (Last In First Out) zugegriffen werden kann. D.h., der zuletzt in den Stapelspeicher geladene Wert ist der oberste, nur er ist für Operationen zugänglich. Wenn tieferliegende Werte abgearbeitet werden sollen, müssen die darüberliegenden gelöscht oder in einem anderen Speicher zwischengespeichert werden. Dieser Speicher ist wieder ein Stapelspeicher oder ein Register.

Ein Register ist ein Speicher in der CPU, auf den schnell zugegriffen werden kann.

Der Speicher, mit dem *dc* arbeitet, wird im folgenden in der Regel nur mit "Stapel" bezeichnet. Nur dort, wo es erforderlich ist, diesen Arbeitsspeicher von anderen Speichern zu unterscheiden (wiederum Stapel oder Register), wird er als "Hauptstapel" bezeichnet.

dc-Eingaben erfolgen in der umgekehrten polnischen Notation (Postfix-Notation). Wenn Sie z.B. 3 und 4 addieren möchten, geben Sie ein:

```
3 4 +
```

Standardmäßig arbeitet *dc* mit ganzen Dezimalzahlen. Sie können jedoch auch eine andere Basis für Ein- und Ausgaben festlegen (siehe *Syntax von dc-Programmen, i bzw. o*) und die Zahl der Nachkommastellen verändern (siehe *Syntax von dc-Programmen, k*).

dc wird von *bc* aufgerufen (siehe *bc*). *bc* ist ein Präprozessor für *dc*, der die Eingabe der Operatoren und Operanden in der normalen arithmetischen Notation ermöglicht und eine C-ähnliche Syntax zur Realisierung von Funktionen bietet. Außerdem können *bc*-Programme Anweisungen zur Ablaufsteuerung enthalten.

SYNTAX VON DC-PROGRAMMEN

c
Der gesamte Stapelinhalt wird gelöscht.

d
Der oberste Wert im Stapel wird dupliziert (ein weiteres Mal in den Stapel geladen).

f
Alle im Stapel enthaltenen Werte werden ausgegeben, bleiben jedoch im Stapel erhalten.

i
Der oberste Wert im Stapel wird als Basis für weitere Eingaben verwendet; der Wert wird dabei aus dem Stapel entfernt.
Der voreingestellte Wert für die Basis, in der Eingaben interpretiert werden, ist 10.

I
Die aktuell geltende Basis für Eingaben wird in den Stapel geladen.

k
Der oberste Wert im Stapel wird aus dem Stapel entfernt und als Skalierungsfaktor benutzt. D.h., bei allen folgenden Ausgaben wird eine diesem Wert entsprechende Anzahl von Nachkommastellen ausgegeben, und diese Anzahl wird bei Multiplikationen, Division und Exponentiationen beibehalten.

lx
Der Inhalt des Registers *x* wird in den Stapel geladen. Das Register *x* bleibt dabei unverändert. Alle Register enthalten am Anfang den Wert 0.

Lx
Der oberste Wert wird vom Stapel *x* gelöscht und in den Haupt-Stapelspeicher übertragen.

o
Der oberste Wert im Stapel wird als Basis für weitere Ausgaben verwendet; der Wert wird dabei aus dem Stapel gelöscht.
Der voreingestellte Wert für die Basis, in der Zahlen ausgegeben werden, ist 10.

O
Die aktuell geltende Basis für Ausgaben wird in den Stapel geladen.

p
Der oberste Wert im Stapel wird ausgegeben, jedoch nicht aus dem Stapel entfernt.

P
Der oberste Wert im Stapel wird als ASCII-Zeichenkette interpretiert und ausgegeben, wobei der Wert aus dem Stapel entfernt wird.

q

Das Programm wird beendet. Wenn eine Schleife ausgeführt wird, so wird die Rekursionsstufe um zwei verringert.

Q

Das Programm wird beendet. Der oberste Wert im Stapel wird aus dem Stapel entfernt und die Ausführungsstufe der Schleife um diesen Wert verringert.

sx

Der oberste Wert des Stapels wird gelöscht und in das Register *x* geladen, d.h. als oberstes Element im Register *x* abgelegt. *x* kann ein beliebiges Zeichen sein.

Sx

Der oberste Wert des Hauptstapels wird gelöscht und in den Stapel *x* geladen, d.h. als oberstes Element im Stapel *x* abgelegt. *x* kann ein beliebiges Zeichen sein.

v

Der oberste Wert im Stapel wird durch seine Quadratwurzel ersetzt. Beim Wurzelziehen werden auch alle Nachkommastellen berücksichtigt (auch wenn der geltende Skalierungsfaktor kleiner ist als die Anzahl der vorhandenen Nachkommastellen, siehe *k*).

x

Der oberste Wert im Stapel wird als Folge von Anweisungen an das *dc*-Kommando interpretiert und ausgeführt.

X

Der zuletzt geladene Wert wird durch die aktuell geltende Zahl der Nachkommastellen ersetzt.

z

Die Anzahl der Elemente des Stapels wird in den Stapel geladen.

Z

Die oberste Zahl im Stapel wird durch ihre Länge ersetzt.

zahl

zahl wird in den Stapel geladen. *zahl* ist eine Zeichenkette, die aus den Ziffern 0-9 sowie Punkten bestehen darf und der ein Unterstrich _ vorangestellt sein kann.
Der erste auftretende Punkt wird als Dezimalpunkt interpretiert, alle anderen Punkte werden ignoriert.
Ein vorangestellter Unterstrich _ wird als negatives Vorzeichen interpretiert.

!

Der Rest der Zeile nach dem Ausrufezeichen wird als SINIX-Kommando interpretiert und ausgeführt.

+
-
/
*****
%
^

Die beiden obersten Zahlen im Stapel werden mit dem angegebenen Operator verknüpft. Dabei bedeutet:

+	addieren
−	subtrahieren
/	dividieren
*	multiplizieren
%	Kongruenzklassen bilden (Rest beim Teilen)
^	potenzieren

Die beiden Zahlen werden aus dem Stapel gelöscht. Das Ergebnis der ausgeführten Operation wird anstelle der beiden Zahlen in den Stapel geladen.

;:

bc verwendet die Zeichen ; und : für Feldoperationen.

<x
>x
=x

Die beiden obersten Werte werden aus dem Stapel geholt und miteinander verglichen. Die Relation ist erfüllt, wenn der oberste Wert im Stapel kleiner bzw. größer ist als der zweitoberste Wert, bzw. gleich dem zweitobersten Wert ist. Wenn die angegebene Relation erfüllt ist, dann wird Register *x* ausgewertet.

?

Eine Eingabezeile wird vom Eingabegerät (das ist normalerweise die Datensichtstation) geholt und ausgeführt.

[...]

Die in eckigen Klammern eingeschlossene ASCII-Zeichenkette wird in den Stapel geladen.

FEHLERMELDUNGEN

`x is unimplemented`

x ist eine Oktalzahl, die keine gültige Anweisung für *dc* ist.

`stack empty`

Stapelinhalt reicht für auszuführende Operation nicht aus.

`Out of space`

Kein freier Platz mehr im Stapel (zu viele Ziffern).

`Out of headers`

Zu viele Zahlen.

`Out of pushdown`

Zu viele Werte im Stapel.

`Nesting Depth`

Zu große Schachtelungstiefe bei angeforderten Operationen.

`exp not an integer`

Exponent beim Potenzieren ist keine ganze Zahl.

BEISPIELE

1. In diesem Beispiel werden die Zahlen 0-9 ausgegeben:

```
[lip1+sili10>a]sa
0si lax
0
1
2
3
4
5
6
7
8
9
```

dc arbeitet diese Eingabe folgendermaßen ab: Zunächst wird die Zeichenkette *lip1+sili10>a* als oberster Wert in den Stapel geladen ([ ]).
Dann wird die Zeichenkette im Stapel gelöscht und im Register *a* gespeichert (*sa*). Der Befehl *0si* bewirkt, daß die Zahl 0 im Register *i* gespeichert wird. Daraufhin wird die Zeichenkette *lip1+sili10>a* im Stapel abgelegt, bleibt aber zusätzlich im Register *a* gespeichert. Der Befehl *x* bewirkt, daß nun die Zeichenkette *lip1+sili10>a* als Folge von *dc*-Befehlen interpretiert und entsprechend ausgewertet wird.

Der Ablauf sieht demnach folgendermaßen aus: Die Zahl 0 (Inhalt des Registers *i*) wird in den Stapel geladen und ausgegeben usw.

2. In diesem Beispiel werden die ersten zehn Werte von n! ausgegeben:

```
[la1+dsa*pla10>y]sy
0sa1
lyx
1
2
6
24
120
720
5040
40320
362880
3628800
```

SIEHE AUCH

bc

dd
Dateien kopieren und konvertieren

Mit *dd* können Sie Dateien kopieren und dabei konvertieren, z.B von EBCDIC nach ASCII oder umgekehrt. Da das Kommando *dd* in beliebigen Block-Größen lesen und schreiben kann, eignet es sich gut für die Ein- und Ausgabe über im Raw-Modus betriebene Geräte (raw devices). Allerdings ist darauf zu achten, daß raw devices nur mit Blockgrößen beschrieben werden können, die für das betreffende Gerät spezifisch sind (z.B. 512 byte für Diskettenlaufwerke). Wenn die zu konvertierende Dateigröße nicht ein Vielfaches der spezifischen Blockgröße ist, müssen Sie die Option *conv=sync* angeben. Nach beendeter Bearbeitung gibt *dd* die Zahl der ganzen Blöcke und der Teil-Blöcke, die eingelesen bzw. ausgegeben wurden, auf die Standard-Fehlerausgabe aus.

Vorsicht

Verwenden Sie *dd* nie zum Kopieren von Dateien zwischen Dateisystemen mit unterschiedlicher Blockgröße.

Wenn Sie ein blockorientiertes Gerät (block device) zum Kopieren der Datei verwenden, dann wird der letzte Block möglicherweise durch Null-Byte bis zur Blockgröße aufgefüllt.

dd[_option]...

option

Option	Kurzbeschreibung
if=eingabedatei	Name der Eingabe-Datei
of=ausgabedatei	Name der Ausgabe-Datei
ibs=n	Größe der Eingabe-Blöcke in byte
obs=n	Größe der Ausgabe-Blöcke in byte
bs=n	Größe der Ein-und Ausgabe-Blöcke in byte
cbs=n	Größe des Konvertierungspuffers in byte
files=n	Anzahl der Dateien, die kopiert und aneinandergfügt werden
skip=n	Überspringen der ersten n *B*löcke der Eingabe-Datei
iseek=n	Überspringen der ersten n *B*löcke der Eingabe-Datei
oseek=n	Kopie der Eingabe-Datei ab *n*+1 Blöcken der Ausgabe-Datei
seek=n	Kopie der Eingabe-Datei ab *n*+1 Blöcken der Ausgabe-Datei
count=n	Die ersten *n* Blöcke der Eingabe-Datei kopieren/konvertieren
conv=ascii	Konvertierung von EBCDIC nach ASCII
ebcdic	Konvertierung von ASCII nach EBCDIC
ibm	Konvertierung von ASCII nach IBM-spezifischen EBCDIC
block	Konvertierung von Sätzen, die mit Neue-Zeile-Zeichen abgeschlossen sind in Sätze mit fester Satzlänge
unblock	Konvertierung von Sätzen mit fester Satzlänge in Sätze, die mit Neue-Zeile-Zeichen abgeschlossen sind
lcase	Umwandlung von Großbuchstaben in Kleinbuchstaben
ucase	Umwandlung von Kleinbuchstaben in Großbuchstaben
swab	Vertauschen zweier aufeinanderfolgender Byte
noerror	Bearbeitung trotz Fehler fortsetzen
sync	Auffüllen der Eingabe-Blöcke auf die in *ibs=n* festgelegte Anzahl Zeichen
conv=...,...	Angabe mehrere Werte für *conv*

Keine Option angegeben
dd liest von der Standard-Eingabe und gibt auf die Standard-Ausgabe aus. *dd* führt keine Konvertierungen durch. Die Größe der Eingabe-Blöcke und der Ausgabe-Blöcke beträgt jeweils 512 byte.

Wenn eine Option Größenangaben verlangt, können Sie diese auf folgende Weise machen:

*n***b** bedeutet *n* mal 512
*n***k** bedeutet *n* mal 1024
*n***w** bedeutet *n* mal 2
*n***x***m* bedeutet *n* mal *m*

if=eingabedatei
(if - input file) Für *eingabedatei* geben Sie den Namen der Eingabe-Datei an, aus der *dd* lesen soll.

if=eingabedatei nicht angegeben:
dd liest von der Standard-Eingabe.

of=ausgabedatei
(of - output file) Für *ausgabedatei* geben Sie den Namen der Ausgabe-Datei an, in die *dd* schreiben soll.

of=ausgabedatei nicht angegeben:
dd schreibt auf die Standard-Ausgabe.

ibs=n
(ibs - input block size) Für *n* geben Sie die Größe der Eingabe-Blöcke in byte an.

ibs=n nicht angegeben:
Die Größe der Eingabe-Blöcke ist 512 byte.

obs=n
(obs - oputput block size) Für *n* geben Sie die Größe der Ausgabe-Blöcke in byte an.

obs=n nicht angegeben:
Die Größe der Ausgabe-Blöcke ist 512 byte.

bs=n
(bs - block size) Für *n* geben Sie die Größe der Eingabe- und Ausgabe-Blöcke in byte an. Die Angabe von *bs* hebt die Angaben für *ibs* und *obs* auf.

cbs=n
(cbs - conversion buffer size) Für *n* geben Sie die Größe des Konvertierungspuffers in byte an. Diese Angabe ist nur in folgenden Fällen wirksam:
- bei der Konvertierung von EBCDIC in ASCII und umgekehrt
- bei der Konvertierung von und zu fester Satzlänge.

(siehe *conv=ascii*, *conv=ebcdic* *conv=ibm* und *conv=block*, *conv=unblock*).

files=n
Für *n* geben Sie eine Anzahl von Dateien an. Diese werden eingelesen, kopiert und aneinandergefügt. Danach beendet *dd* die Bearbeitung.
Die Verwendung dieser Option ist nur dann sinnvoll, wenn die Eingabe von einem Magnetband oder einem ähnlichem Gerät (z.B. Streamerband) gelesen wird.

skip=n
Die ersten *n* Blöcke der Eingabe-Datei werden übersprungen und die Bearbeitung beginnt erst beim Block *n*+1.
Diese Option ist nur für Magnetbandgeräte geeignet, für die *iseek* nicht definiert ist.

iseek=n
Die ersten *n* Blöcke der Eingabe-Datei werden übersprungen und die Bearbeitung beginnt erst beim Block *n*+1. Diese Option ist speziell für die Bearbeitung von Plattendateien gedacht, da *skip* hier zu langsam ist. Diese Option ist nur für Geräte zulässig, die den *lseek*-Systemaufruf unterstützen (z.B. Plattengeräte).

oseek=n
Die ggf. konvertierte Kopie der Eingabe-Datei beginnt erst beim *n*+1-ten Block der Ausgabe-Datei. Die ersten *n* Blöcke der Ausgabe-Datei bestehen aus Null-Bytes. Diese Option ist nur für Geräte zulässig, die den *lseek*-Systemaufruf unterstützen (z.B. Plattengeräte).

seek=n
seek hat die gleiche Funktion wie *oseek*.

count=n
Es werden nur die ersten *n* Blöcke der Eingabe-Datei kopiert und, falls angegeben, konvertiert.

conv=ascii
Konvertierung von EBCDIC nach ASCII.
Die durch *cbs=n* festgelegte Anzahl von Zeichen wird jeweils in den Konvertierungspuffer kopiert und eventuell angegebene Zeichenumwandlungen werden ausgeführt. Leerzeichen am Zeilenende werden gelöscht und ein Neue-Zeile-Zeichen wird eingefügt. Damit wird die Größe der Ausgabeblöcke der Datei an den durch *cbs=n* festgelegten Wert angepaßt. Haben Sie die *cbs*-Option nicht oder mit *cbs=0* angegeben, wird die Eingabe-Datei nur konvertiert und am Zeilenende werden keine Leerzeichen gelöscht.

conv=ebcdic
Konvertierung von ASCII nach EBCDIC.
Die ASCII-Zeichen werden in den Konvertierungspuffer gelesen und nach EBCDIC konvertiert. Die Ausgabeblockgröße wird durch Hinzufügen von Leerzeichen an den durch *cbs=n* festgelegten Wert angepaßt. Haben Sie die Option *cbs* nicht oder mit *cbs=0* angegeben, wird die Eingabe-Datei nur konvertiert, ihre Blockstruktur wird jedoch nicht verändert.

conv=ibm
Konvertierung von ASCII nach EBCDIC wie oben beschrieben, wobei eine IBM-spezifische EBCDIC-Tabelle verwendet wird.

conv=block
Durch Neue-Zeile-Zeichen abgeschlossene ASCII-Sätze werden in Sätze mit fester Satzlänge konvertiert. Die ASCII-Zeichen werden in den Konvertierungspuffer gelesen. Die Satzlänge oder Ausgabeblockgröße wird durch Hinzufügen von Leerzeichen an den durch *cbs=n* festgelegten Wert angepaßt. Haben sie die Option *cbs* nicht oder mit *cbs=0* angegeben, wird die Eingabe-Datei nur konvertiert.

conv=unblock
ASCII-Sätze mit fester Satzlänge werden in durch Neue-Zeile-Zeichen abgeschlossene Sätze konvertiert. Die Satzlänge wird durch den mit *cbs=n* festgelegten Wert bestimmt. Diese Anzahl Zeichen wird jedesmal in den Konvertierungspuffer geladen, am Zeilenende überschüssige Leerzeichen werden gelöscht und ein Neue-Zeile-Zeichen wird eingefügt, bevor die Zeichen ausgegeben werden. Haben Sie die Option *cbs* nicht oder mit *cbs=0* angegeben, wird die Eingabe-Datei nur konvertiert .

conv=lcase
Großbuchstaben werden in die entsprechenden Kleinbuchstaben verwandelt.

conv=ucase
Kleinbuchstaben werden in die entsprechenden Großbuchstaben verwandelt.

conv=swab
Jeweils zwei aufeinanderfolgende Byte werden vertauscht.

conv=noerror
Die Bearbeitung wird beim Auftreten eines Fehlers nicht beendet.

conv=sync
Alle Eingabe-Blöcke werden auf die durch *ibs=n* festgelegte Anzahl von Zeichen aufgefüllt.

conv=...,...
Sie können für *conv* mehrere Werte, durch Kommas voneinander getrennt, angeben und so mehrere Konvertierungen herbeiführen.

BEISPIEL

Der Inhalt eines EBCDIC-Magnetbandes mit einer Blockstruktur von 10 mal 80 byte wird in die ASCII-Datei *xy* geschrieben:

```
$ dd if=/dev/rmt32 of=xy ibs=800 cbs=80 conv=ascii
```

SIEHE AUCH

cp

deroff
Herausfiltern von nroff-, troff-, tbl- und eqn-Anweisungen

deroff entfernt alle Zeilen mit nroff- und troff-Anweisungen, alle Funktionen mit einem Gegenschrägstrich \, alle Makroaufrufe, die eqn-Anweisungen (zwischen den Zeilen .EQ und .EN oder zwischen Begrenzern), sowie Tabellenbeschreibungen. Die Anweisungen werden durch Leerzeichen ersetzt. Der Dateiinhalt wird in der modifizierten Form auf die Standard-Ausgabe geschrieben.

deroff folgt Ketten einzufügender Dateien (Anweisungen .so und .nx); falls eine Datei bereits eingefügt wurde, wird ein .so ignoriert und ein .nx beendet die Ausführung.

deroff[␣option]...[␣datei]...

Keine Option angegeben
: *deroff* filtert aus dem eingelesenen Text die oben genannten Formatieranweisungen aus und schreibt den Rest auf die Standard-Ausgabe.

option

-m[m|l]
: (m - macros) *-mm* bewirkt, daß die Makros so interpretiert werden, daß nur Fließtext ausgegeben wird. Aus Makrozeilen wird kein Text ausgegeben.
-ml erzwingt die Option *-mm* und bewirkt, daß zusätzlich die Listen gelöscht werden, die mit *mm* Makros assoziiert sind.

-w
: Die Ausgabe besteht aus einer Wortliste, die pro Zeile ein Wort enthält, alle anderen Zeichen werden ignoriert. Im übrigen folgt die Ausgabe dem Original mit den oben angegebenen Veränderungen.
Im Text ist für *deroff* ein Wort eine beliebige Zeichenkette, die mindestens aus zwei Buchstaben besteht, und sich aus Buchstaben, Ziffern, kommerziellen Und-Zeichen & und Apostrophen ' zusammensetzt. In einem Makroaufruf dagegen ist ein Wort eine Zeichenkette, die mit mindestens zwei Buchstaben anfängt, und mindestens drei Buchstaben enthält.
Begrenzungszeichen sind alle Zeichen außer Buchstaben, Ziffern, Apostrophen ' und kommerziellen Und-Zeichen &. Dem Wort folgende Apostrophe und kommerziellen Und-Zeichen werden entfernt.

datei
: Name der Datei, die gefiltert werden soll. Es können mehrere Dateien angegeben werden.

datei nicht angegeben:
deroff liest von der Standard-Eingabe.

BEISPIEL

Der Text der troff-Datei text.der wird zuerst ungefiltert mit *cat* und danach gefiltert mit *deroff* ausgegeben:

```
$ cat text.der
'\"macro stdmacro
.if n .pH g1.basename @(#)basename      40.5 of 10/10/89
.\" Copyright
.nr X
.if \nX=0 .ds x} basename 1 "User Environment Utilities" "\&"
.if \nX=1 .ds x} basename 1 "User Environment Utilities"
.if \nX=2 .ds x} basename 1 "" "\&"
.if \nX=3 .ds x} basename "" "" "\&"
.TH \*(x}
.if t .ds ' \h@.05m@\s+4\v@.333m@\'\v@-.333m@\s-4\h@.05m@

$ deroff text.der
macro stdmacro

 Copyright

     basename  User Environment Utilities
     basename  User Environment Utilities
     basename
     basename
```

destroy
Dateien physikalisch löschen

destroy überschreibt Dateien mit beliebigen Zeichen und löscht anschließend den Eintrag im Dateiverzeichnis. Mit *destroy* können Sie nur einfache Dateien überschreiben und löschen.

destroy[_option]..._datei_...

Keine Option angegeben
Die Eingabedateien werden vor dem Löschen mit binären Nullen überschrieben. *destroy* überschreibt und löscht nur Dateien, auf die nur ein einziger Verweis besteht.

option

-cX
(c - character) Die Eingabedateien werden vor dem Löschen statt mit binären Nullen mit dem Zeichen überschrieben, das Sie für *X* eingeben.

-i
(i - interactive mode) *destroy* fragt für jede Datei, ob sie gelöscht werden soll.

-l
(l - linked files) *destroy* überschreibt und löscht auch Dateien, auf die mehrere Verweise bestehen.
Die Dateien, die auf die durch *destroy* gelöschte Datei verwiesen haben, werden durch binäre Nullen (oder, falls die Option *-cX* angegeben wurde, durch das Zeichen *X*) überschreiben.

-l nicht angegeben:
destroy überschreibt und löscht nur Dateien, auf die nur ein einziger Verweis besteht.

-s
(s - silent mode) Fehlermeldungen werden unterdrückt.

datei
Name der Datei, die Sie mit *destroy* löschen möchten. *destroy* löscht nur einfache Dateien. Pro Aufruf können Sie mehrere Dateinamen angeben.

FEHLERMELDUNGEN

`destroy: cannot overwrite` *datei* `(Permission denied)`

Sie haben für *datei* kein Schreibrecht, deshalb kann *destroy* die Datei nicht überschreiben.

`destroy: cannot destroy directory` *dvz*

Sie haben *destroy* mit dem Namen des Dateiverzeichnisses *dvz* aufgerufen, *destroy* löscht jedoch nur einfache Dateien.

`destroy: file` *datei* `has n links -- not destroyed`

Auf die Eingabedatei *datei* bestehen *n* Verweise (n > 1); *destroy* löscht Dateien mit mehreren Verweisen jedoch nur dann, wenn Sie beim Aufruf Option *-l* angeben.

`destroy: cannot stat` *datei* `(No such file or directory)`

Die Datei *datei* existiert nicht.

BEISPIEL

Das aktuelle Dateiverzeichnis enthält die Datei *datei*. Mit *ln* können Sie einen zweiten Verweis auf die Datei definieren:

```
$ ln datei kopie
$ ls -l
-rw-------   2 hugo   projekt   14  Jan 16 15:04 datei
-rw-------   2 hugo   projekt   14  Jan 16 15:04 kopie
$ cat datei
Guten Morgen!
$ cat kopie
Guten Morgen!
```

Die Datei ist unter beiden Namen ansprechbar, sie ist physikalisch aber nur einmal vorhanden.

```
$ destroy datei
destroy: file datei has 2 links -- not destroyed
```

destroy überschreibt und löscht die Datei nicht, da mehrere Verweise auf sie bestehen.

```
$ destroy -l -cX datei
```

destroy überschreibt den Dateiinhalt, da Sie Option *-l* angegeben haben. Aufgrund der Option *-cX* wird der Dateiinhalt mit *X* überschrieben. Anschließend löscht *destroy* den Verweis *datei*:

```
$ ls -l
-rw-------   1 hugo   projekt   14  Jan 16 15:04 kopie
$ cat kopie
XXXXXXXXXXXXXX
```

SIEHE AUCH

ln, *rm*
secure und *wcheck* [8]

df
Anzahl der freien und belegten Plattenblöcke und I-Nodes ausgeben (disk free)

Mit *df* können Sie sich ausgeben lassen, wieviel freie Plattenblöcke und I-Nodes im Superblock noch für montierte bzw. nicht montierte lokale Dateisysteme sowie für Ressourcen zur Verfügung stehen. Eine Ressource ist ein Dateisystem eines anderen Rechners, das über RFS/NFS montiert ist und Benutzern im Netzwerk zur Verfügung gestellt wird.

df[_option]...[_datei]...

Keine Option angegeben
df gibt die Anzahl der freien Blöcke und I-Nodes aller montierten lokalen Dateisysteme und Ressourcen aus.

option

-F_DStyp
Das Dateisystem, mit dem gearbeitet werden soll, ist vom Typ *DStyp*. Die Angabe dieser Option ist nur notwendig, wenn das Dateisystem nicht montiert ist.

DStyp kann sein:

s5	Original UNIX System V Dateisystem
ufs	Berkeley-Dateisystem
bfs	Boot-Dateisystem (nur */stand*)
proc	Pseudo-Dateisystem zum Zugriff auf Prozeßdaten
fdfs	Pseudo-Dateisystem zum Zugriff auf Dateikennzahlen
nfs	an fernem Rechner montiertes Dateisystem (network file system, Berkeley)

-n[_mountdvz]
df gibt für *mountdvz* den zugehörigen Dateisystem-Typ aus. *mountdvz* ist der Name des Dateiverzeichnisses, an das das entsprechende Dateisystem montiert ist. Sie können mehrere Verzeichnisse angeben.
Diese Option kann nicht mit s5_option oder mit *-o* benutzt werden. Kombinieren Sie *-n* mit anderen Optionen, so erhalten Sie zusätzlich zur Ausgabe eine Fehlermeldung.

mountdvz nicht angegeben:
Sie erhalten eine Liste aller montierten Dateisysteme mit den zugehörigen Dateisystem-Typen.

-b

Der freie Speicherplatz (in Kbyte) wird ausgegeben. Diese Option kann nicht mit *-o* benutzt werden.

-e

Die Anzahl der freien I-Nodes wird ausgegeben. Diese Option kann nicht mit *-o* benutzt werden.

-g

df gibt für alle montierten Dateisysteme oder für *DStyp* bzw. *datei* die gesamte *statvfs*-Struktur aus. Sie erhalten eine vierzeilige Ausgabe mit folgenden Informationen:

`dvz`	Name des Montier-Dateiverzeichnisses (die Zeichenkette *dvz* erscheint nicht in der Ausgabe)
`gerät`	entsprechende Gerätedatei (die Zeichenkette *gerät* erscheint nicht in der Ausgabe)
`block size`	bevorzugte Blockgröße des Dateisystems,
`frag size`	grundlegende Blockgröße des Dateisystems
`total blocks`	gesamte Anzahl der Blöcke in Einheiten von *frag size*
`free blocks`	gesamte Anzahl der freien Blöcke
`available`	Anzahl der freien Blöcke für Nicht-Systemverwalter
`total files`	gesamte Anzahl der I-Nodes
`free files`	Anzahl der freien I-Nodes
`filesys id`	Nummer des Dateisystems
`name`	Dateisystem-spezifische Bezeichnung (die Zeichenkette *name* erscheint nicht in der Ausgabe)
`fstype`	Typ des Dateisystems
`flag`	Flags des Dateisystems
`filename length`	maximale Länge von Dateinamen

Diese Option kann nicht mit s5_option oder mit *-o* benutzt werden und setzt die Optionen *-b, -e, -k, -n, -t* außer Kraft.

-k

Mit dieser Option erhalten Sie spaltenweise folgende Informationen über alle montierten oder die angegebenen Dateisysteme:

`filesystem`	Gerätedatei des Dateisystems
`kbytes`	Gesamt-Speicherplatz in Kbyte
`used`	belegter Speicherplatz in Kbyte
`avail`	noch nicht belegter Speicherplatz in Kbyte
`capacity`	belegter Speicherplatz in Prozent
`mounted on`	Name des Dateiverzeichnisses, an das das Dateisystem montiert ist

Die Ausgabe für *capacity* basiert für normale Benutzer auf einem Grundwert von 100 % und für Systemverwalter auf einem Grundwert von 110 %.

-k kann nicht mit *-o* benutzt werden und setzt *s5_option* außer Kraft.

-l

df gibt für alle lokal montierten oder die angegebenen Dateisysteme die Anzahl der freien Blöcke und I-Nodes aus. *-l* ist die standardmäßig voreingestellte Option. Diese Option kann nicht mit *s5_option* oder mit *-o* benutzt werden.

-t

df gibt pro montiertem oder angegebenem Dateisystem die Anzahl freier Blöcke und Dateien sowie die Gesamtanzahl der zur Verfügung stehenden Blöcke und Dateien aus.
Diese Option kann nicht mit *-o* benutzt werden.

-v

Mit dieser Option erhalten Sie spaltenweise folgende Informationen über alle montierten oder die angegebenen Dateisysteme:

`Mount Dir`	Name des Dateiverzeichnisses, an das das Dateisystem montiert ist
`Filesystem`	Gerätedatei des Dateisystems
`blocks`	Gesamt-Speicherplatz in 512-byte-Blöcken
`used`	belegter Speicherplatz in 512-byte-Blöcken
`free`	noch nicht belegter Speicherplatz in 512-byte-Blöcken
`%used`	belegter Speicherplatz in Prozent

Die Ausgabe für *%used* basiert für normale Benutzer auf einem Grundwert von 100 % und für Systemverwalter auf einem Grundwert von 110 %.

Diese Option kann nicht mit *-o* benutzt werden.

-V

df ergänzt eine unvollständig eingegebene *df*-Kommandozeile und gibt das ergänzte Kommando auf die Standard-Ausgabe aus. Das Kommando *df* wird jedoch nicht ausgeführt. *df* orientiert sich für die Ergänzung an den entsprechenden Einträgen in den Dateien */etc/mnttab* und */etc/vfstab*.

Ergänzt wird die Option *-F␣DStyp* und/oder die zugehörige Gerätedatei, je nachdem, ob Sie eine der beiden Komponenten bereits beim Kommandoaufruf angegeben haben.
Wenn Sie *df␣-V* ohne andere Optionen und Argumente eingeben, dann erhalten Sie eine Liste von *df*-Kommandozeilen, die Ihnen alle Typen der an Ihrem Rechner montierten Dateisysteme und die entsprechenden Gerätedateien anzeigt.

s5␣option

Ist *DStyp* vom Typ *s5*, dann können Sie eine *s5*-spezifische Option angeben:

-f

Mit dieser Option wird nicht der Zähler im Superblock überprüft, sondern die Anzahl der freien Blöcke aus der "Liste der freien Blöcke" entnommen.
-f wird durch die Option *-k* außer Kraft gesetzt.

-o␣ufs␣option

Ist *DStyp* vom Typ *ufs*, dann können Sie eine *ufs*-spezifische Option angeben:

i

Sie erhalten spaltenweise folgende Informationen:

`Filesystem`	Name der Gerätedatei
`iused`	Anzahl der belegten I-Nodes
`ifree`	Anzahl der freien I-Nodes
`%iused`	belegte I-Nodes in Prozent
`Mounted on`	Name des Dateiverzeichnisses, an das das Dateisystem montiert ist

-o␣i ist nur mit der Option *-F* kombinierbar.

datei

datei ist entweder der Name des Dateiverzeichnisses, an das *DStyp* montiert ist (siehe *-F*) oder der Name der entsprechenden Gerätedatei. Sie können mehrere Namen angeben.

datei nicht angegeben:
df bezieht sich auf alle lokal und fern montierten Dateisysteme oder auf alle Dateisysteme vom Typ *DStyp*.

FEHLERMELDUNGEN

Hinweis

Bei einigen der folgenden Fehlermeldungen wird zusätzlich die korrekte Kommando-Syntax ausgegeben. Diese Syntax weicht von einigen Elementen der obigen Beschreibung wie folgt ab:

Syntaxelement der Fehlermeldung	entspricht in der obigen Beschreibung:
current_options	s5_option
specific_options	ufs_option
directory | special	datei
generic options	*-begkltvV*

`df: FStype cannot be identified`

Bei *df␣-V* wurde der Name einer Gerätedatei oder eines Dateiverzeichnisses unvollständig oder falsch angegeben.

`ufs usage: df [-F ufs] [generic options] [-o i] [directory|special]`

Zur Option *-o* wurde für *ufs_option* ein ungültiges Argument angegeben. *ufs_option* kann nur *i* sein.

`df: operation not applicable for FStype DStyp`

Bei *df␣-F␣DStyp␣-o␣i* wurde für *DStyp* ein ungültiger Dateisystem-Typ angegeben. Im Zusammenhang mit der Option *-o* kann nur der Typ *ufs* angegeben werden.

`df: cannot stat` *datei*

datei (Gerätedatei oder Dateiverzeichnis) existiert nicht oder ist ungültig.

`df: cannot stat xx`

Die Option *-n* wurde mit einer Option *xx* kombiniert. *-n* sollte nur allein benutzt werden.

DATEIEN

*/dev/dsk/**
/proc
/dev/fd
Gerätedateien der Dateisysteme

/etc/mnttab
Tabelle der montierten Dateisysteme

/etc/vfstab
Liste der Standardparameter für jedes Dateisystem

*/sbin/**
Kommandos und Programme zur Systemverwaltung

*/etc/fs/DStyp/**
Kommandos für die Dateisystem-Typen

BEISPIELE

1. Für alle Benutzerverzeichnisse sollen die zugehörigen Dateisystem-Typen ausgegeben werden:

```
$ df -n /h*
/home    : ufs
/home2   : ufs
/home3   : ufs
```

2. Für das Dateisystem */stand* soll die gesamte *statvfs*-Struktur ausgegeben werden:

```
$ df -g /stand
/stand          (/dev/dsk/c0d0s10):     512 block size     512 frag size
  10710 total blocks  7577 free blocks   7577 available      104 total file
    100 free files      10 filsys id                          /stand
    bfs fstype  0x00000000 flag           14 filename length
```

SIEHE AUCH

du

df [7], *statvfs()* [19]

Netzwerke - Leitfaden für Benutzer und Verwalter, Beschreibung [9]

diff
Dateien zeilenweise vergleichen (differential)

Mit *diff* können Sie zwei Dateien zeilenweise vergleichen. *diff* gibt aus:

- die Zeilen, in denen sich die Dateien unterscheiden
- *ed*-ähnliche Kommandos, mit denen man aus *datei1 datei2* erzeugen kann.

Sie können mit *diff* Unterschiede zwischen Dateien feststellen und, wenn erwünscht, auch beseitigen.

diff[_option]_datei1_datei2

Keine Option angegeben

Wenn die verglichenen Dateien gleich sind, gibt *diff* nichts aus. Wenn die verglichenen Dateien Unterschiede aufweisen, gibt *diff* aus:

1. Zeile[nbereich] aus *datei1* — *ed*-Kommando — Zeile[nbereich] aus *datei2*
2. Zeilen, die nur in *datei1* stehen
3. Zeilen, die nur in *datei2* stehen

Die Ausgabe hat folgendes Format:

```
                a
1. n1[,n2]      d n1[,n2]
                c

2. < text einer Zeile aus datei1
          .
          .
          .
          - - -
3. > text einer Zeile aus datei2
          .
          .
          .
```

a, d und *c* sind *ed*-ähnliche Kommandos. Sie bedeuten:

a (append) anfügen
d (delete) löschen
c (change) ändern

Die Ausgabe lesen Sie folgendermaßen:
Die *ed*-Kommandos *a, d* und *c* mit den davorstehenden Zeilen(bereichs)angaben zeigen, wie *datei1* in *datei2* umzuwandeln ist.

Wenn Sie *a* durch *d* und *d* durch *a* ersetzen und die rechts stehenden Zeilen-(bereichs)angaben verwenden, sehen Sie, wie *datei2* in *datei1* umzuwandeln ist. Zeilen aus *datei1* sind mit < gekennzeichnet, aus datei2 mit >.

option

-b

diff berücksichtigt weder Leerzeichen und Tabulatorzeichen am Zeilenende, noch unterschiedlich lange Folgen von Leerzeichen innerhalb von Zeilen an derselben Stelle. Leerzeichen am Zeilenanfang werden als Differenz ausgegeben, ebenso Leerzeilen.

-i

diff berücksichtigt Groß- und Kleinschreibung nicht, z.B. wird 'A' nicht von 'a' unterschieden.

-t

diff expandiert Tabulatorzeichen in der Ausgabe. Bei Option *-c* oder normaler Ausgabe fügt *diff* gelegentlich am Zeilenanfang Zeichen ein, die die Einrückung der ursprünglichen Zeilen verfälschen und dadurch die Interpretation der Ausgabe erschweren. Diese Option erhält die ursprüngliche Einrückung aufrecht.

-w

diff berücksichtigt keine Leer- und Tabulatorzeichen und unterscheidet nicht zwischen verschieden langen Folgen von Leer- und Tabulatorzeichen. Z.B. wird die Zeichenkette
'if␣(␣a␣=␣=␣b␣)' als identisch mit '(a==b)' angesehen.

Die folgenden Optionen schließen sich gegenseitig aus:

-c

diff erzeugt eine dreiteilige Liste, wobei das Ausgabeformat leicht abgeändert ist. Zuerst werden Namen und Entstehungsdatum von *datei1* und *datei2* angezeigt. Dann wird der gefilterte Inhalt von *datei1* ausgegeben. Es werden die Kontrastzeilen ausgegeben, wobei alle Zeilen, die nicht in *datei2* vorkommen, mit einem Minuszeichen - versehen sind. Bei der darauffolgenden Ausgabe von *datei2* sind alle Zeilen, die nicht in *datei1* vorkommen, mit einem Pluszeichen + markiert. Zeilen, die sich in *datei1* und *datei2* unterscheiden, werden jeweils mit einem Ausrufezeichen ! gekennzeichnet.

-C␣zahl

diff gibt im gleichen Format aus wie bei der Option *-c*, zeigt jedoch außer den sich unterscheidenden Stellen auch jeweils *zahl* Zeilen davor und *zahl* Zeilen dahinter an.

-e

nicht mit Option *-h*, *-l* oder *-s* zu verwenden!
diff erzeugt ein *ed*-Skript und gibt es aus. Das *ed*-Skript enthält die Kommandos *a,d* und *c*, sowie die zugehörigen Textzeilen, mit denen der Editor *ed datei1* in *datei2* umwandeln kann. Hierfür muß das *ed*-Skript an den Editor *ed* als Eingabe übergeben werden. Vorher müssen allerdings die Anweisungen *w* und *q* an das Ende des *ed*-Skripts geschrieben werden. Außerdem müssen Sie an den Anfang das Kommando *e datei1* setzen (siehe *ed*).

Die folgenden Optionen beeinflussen die Arbeitsweise von *diff*:

-f

nicht mit Option *-h* zu verwenden!
diff erzeugt ein ähnliches Skript wie bei Option *-e*, nur in der umgekehrten Richtung. Dieses Skript eignet sich aber **nicht** als Eingabe für *ed*.

Die unter *-e* und *-f* erstellten ed-Skripts sind möglicherweise nicht korrekt, wenn die verglichenen Zeilen nur aus einem einzelnen Punkt . bestehen.

-h

nicht mit Option *-e*, *-f*, *-c*, *-C*, *-n* und *-D* zu verwenden!
diff arbeitet schneller, und die Dateien können beliebig lang sein. Allerdings ist das Ergebnis bei Option *-h* nicht zuverlässig!

-n

diff erzeugt, ähnlich wie bei der Option *-e*, ein Skript. Dort stehen die *ed*-Kommandos jedoch in umgekehrter Reihenfolge. Außerdem steht hinter jedem *Insert*- oder *Delete*-Kommando die Anzahl der zu ändernden Zeilen.

-D␣zeichenkette
datei1 und *datei2* sollten in diesem Fall C-Quellprogramme oder C-Quellprogrammteile enthalten. *diff* erzeugt ein C-Quellprogramm aus *datei1* und *datei2*, in das es Angaben für den C-Präprozessor einfügt. Wird dieses Programm übersetzt, dann entsteht eine übersetzte *datei1*, wenn *zeichenkette* nicht definiert ist. Es ergibt sich eine übersetzte *datei2*, wenn *zeichenkette* definiert ist.

Die folgenden Optionen werden zum Vergleich von Dateiverzeichnissen verwendet:

-l

diff gibt im langen Format aus. Vor der Ausführung von *diff* wird jede Textdatei in Seiten zerteilt, indem sie durch eine Pipeline zu *pr* gesendet wird. Weitere Unterschiede werden gesammelt und am Stück ausgegeben, nachdem alle textspezifischen Unterschiede angezeigt sind.

-r

diff arbeitet rekursiv alle gemeinsamen Unterverzeichnisse ab.

-s
diff zeigt an, welche Dateien übereinstimmen. Standardmäßig unterbleibt dies.

-S_name
diff bearbeitet ein Verzeichnis erst ab der Datei *name*.

datei1 datei2
Namen der Dateien, die *diff* vergleichen soll. Ist *datei1* ein Dateiverzeichnis, dann wird aus diesem Verzeichnis die Datei *datei2* für den Vergleich mit *datei2* herangezogen. Ist *datei2* ein Dateiverzeichnis, dann wird mit *datei1* aus diesem Verzeichnis verglichen.

ENDE-STATUS

0 Dateien sind identisch
1 Dateien sind nicht identisch
2 Eingabefehler

FEHLERMELDUNGEN

`diff: two filename arguments required`

Sie haben eine falsche Anzahl Dateien angeben. Es können nur zwei Dateien verglichen werden.

`diff: No such file or directory`

Eine der angegebenen Dateien existiert nicht.

`diff: files too big, try -h`

Sie müssen die Option *-h* angeben, da die zu vergleichenden Dateien zu groß sind.

`diff: Permission denied`

Sie haben kein Leserecht für eine der angegebenen Dateien.

BEISPIEL

1. Die Dateien datei1 und datei2 haben folgenden Inhalt:

```
datei1                  datei2

Amsel Drossel           Amsel und Drossel
Fink und Star           Fink Star
und die ganze           Vogelschar
Vogelschar
```

Mit dem Aufruf von *diff* können Sie genau feststellen, in welchen Zeilen sich die beiden Dateien unterscheiden:

```
$ diff datei1 datei2
1,3c1,2
<Amsel Drossel
<Fink und Star
<und die ganze
- - -
>Amsel und Drossel
>Fink Star
```

Das bedeutet: Um aus *datei1* *datei2* zu erzeugen, müssen die Zeilen 1 bis 3 aus *datei1* (1,3) ersetzt werden (c) durch die Zeilen 1 bis 2 (1,2) aus *datei2*. Was in diesen Zeilen jeweils steht, ist an den mit < oder > beginnenden Ausgabezeilen zu sehen: mit < beginnen die Zeilen, die nur in *datei1*, mit > die Zeilen, die nur in *datei2* stehen.

2. Dateien vergleichen und ein *ed*-Skript erstellen:

```
Inhalt von datei1:      Inhalt von datei2:

heute ist Montag        heute ist Dienstag
es ist kalt             es ist Herbst
                        es ist kalt
```

Nach folgendem Aufruf gibt *diff* die *ed*-Kommandos aus, mit denen *ed datei1* in *datei2* umwandeln kann. Um das Resultat dieses Aufruf als Eingabe für den *ed* benutzen zu können, müssen Sie noch die Anweisungen *w* und *q* anfügen (siehe *ed*).

```
$ diff -e datei1 datei2
1c
heute ist Dienstag
es ist Herbst
.
$
```

SIEHE AUCH

bdiff, cmp, comm, diff3, ed, pr

diff3
Drei Dateien zeilenweise vergleichen (differential)

diff3 vergleicht drei Dateien und schreibt auf die Standard-Ausgabe, in welchen Zeilenbereichen sich die Dateien unterscheiden und wie sie ineinander überführt werden können.

diff3[_option]_datei1_datei2_datei3

Keine Option angegeben

Für nicht übereinstimmende Zeilenbereiche gibt *diff3* eine der folgenden Zeichenketten aus:

====	alle 3 Dateien unterscheiden sich
====1	*datei1* unterscheidet sich von *datei2* und *datei3*
====2	*datei2* unterscheidet sich von *datei1* und *datei3*
====3	*datei3* unterscheidet sich von *datei1* und *datei2*

Unter dieser Angabe stehen jeweils die Änderungen, die nötig wären, um einen bestimmten Zeilenbereich einer Datei einem bestimmten Zeilenbereich einer anderen Datei anzugleichen. Diese Änderungen werden in einem der zwei folgenden Formate ausgegeben:

Text ändern:

i:n1[,n2]**c**
[text]

Um die Unterschiede zu beseitigen, muß in *datei i* (i = 1,2,3) die Zeile *n1* bzw. der Zeilenbereich *n1* bis *n2* geändert werden (c - change). *text* ist der zu ändernde Zeilenbereich. Unterscheiden sich zwei Dateien in dem betreffenden Zeilenbereich nicht, wird *text* für die Datei mit der niedrigeren Nummer *i* nicht ausgegeben.

Text anfügen:

i:na
[text]

Um die Unterschiede zu beseitigen, muß in *datei i* (i = 1,2,3) nach Zeile *n* Text angefügt werden (a - append). *text* ist der anzufügende Zeilenbereich. Unterscheiden sich zwei Dateien in dem betreffenden Zeilenbereich nicht, wird *text* für die Datei mit der niedrigeren Nummer *i* nicht ausgegeben.

option

-e

diff3 gibt ein *ed*-Skript aus, das in *datei1* alle Unterschiede zwischen *datei2* und *datei3* einarbeitet, die ohne Angabe einer Option mit `====` oder `====3` gekennzeichnet würden (siehe *ed*).

Vorsicht

Bei Textzeilen, die nur aus einem einzelnen Punkt . bestehen, wird die Option *-e* ignoriert.

Beispiele

Inhalt von	*datei1*	*datei2*	*datei3*
	aber	doch	aber
	morgen	heute	heute

Der Aufruf von *diff3 -e* erzeugt folgende Ausgabe:

```
$ diff3 -e datei1 datei2 datei3
1,2c
aber
heute
.
w
q
```

Inhalt von	*datei1*	*datei2*	*datei3*
	aber	aber	aber
		heute	heute
			morgen

Der Aufruf von *diff3 -e* erzeugt folgende Ausgabe:

```
$ diff3 -e datei1 datei2 datei3
1a
heute
morgen
.
w
q
```

-E

diff3 gibt ein *ed*-Skript aus, das in *datei1* alle Unterschiede zwischen *datei2* und *datei3* einarbeitet, die ohne Angabe einer Option mit = = = = oder = = = =3 gekennzeichnet würden (siehe *ed*).
Dabei erzeugt *diff3* für die Unterschiede, die ohne Angabe einer Option mit = = = = gekennzeichnet würden, eine erweiterte Ausgabe im *ed*-Skript. Diese Unterschiede werden entsprechend der Datei aus der sie kommen in Zeilen mit < < < < < < oder > > > > > > eingeschlossen.

-x

diff3 gibt ein *ed*-Skript aus, das in *datei1* alle Unterschiede zwischen *datei2* und *datei3* einarbeitet, die ohne Angabe einer Option mit = = = = gekennzeichnet würden (siehe *ed*).

-X

diff3 gibt ein erweitertes *ed*-Skript aus, das in *datei1* alle Unterschiede zwischen *datei2* und *datei3* einarbeitet, die ohne Angabe einer Option mit = = = = gekennzeichnet würden (siehe *ed*).
Dabei werden diese Unterschiede entsprechend der Datei aus der sie kommen in Zeilen mit < < < < < < oder > > > > > > eingeschlossen.

-3

diff3 gibt ein *ed*-Skript aus, das in *datei1* alle Unterschiede zwischen *datei2* und *datei3* einarbeitet, die ohne Angabe einer Option mit = = = =3 gekennzeichnet würden (siehe *ed*).

datei1␣datei2␣datei3

Namen der drei Dateien, die Sie vergleichen möchten. Die Dateien dürfen eine Länge von maximal 64 Kbyte haben.

DATEIEN

*/tmp/d3a**

Datei, die *diff* anlegt. Sie enthält die Unterschiede zwischen *datei1* und *datei3*. *diff* wird von *diff3* aufgerufen.

*/tmp/d3b**

Datei, die *diff* anlegt. Sie enthält die Unterschiede zwischen *datei2* und *datei3*.

/usr/lib/diff3prog

Programm, das die Dateien */tmp/d3a** und */tmp/d3b** vergleicht. *diff3* ruft dieses Programm auf.

BEISPIELE

Sie haben drei Versionen *adr1*, *adr2*, *adr3* einer Adressendatei mit folgendem Inhalt:

```
Inhalt von adr1:          Inhalt von adr2:          Inhalt von adr3:

Herr Witzig               Herr Witzig               Herr Witzig
Lachstr. 00               Lachstr. 00               Lachstr. 00
0000 Kicherlingen 00      0000 Kicherlingen 00      0000 Kicherlingen 00
Tel.: 000 000             Tel.: 000 000             Tel.: 000 000

Herr Traurig              Herr Traurig              Herr Traurig
Traenenstr. 77            Traenenstr. 77            Traenenstr. 77
7777 Weinhausen 77        7777 Weinhausen 77        7777 Weinhausen 77
Tel.: 777 777             Tel.: 444 444             Tel.: 444 444

                                                    Herr Gruebel
                                                    Runzelgasse 33
                                                    3333 Faltenberg 33
                                                    Tel.: 333 333
```

1. Aufruf von *diff3* ohne Option:

```
$ diff3 adr1 adr2 adr3
====
1:9c
  Tel.: 777 777
2:9c
  Tel.: 444 444
3:9,14c
  Tel.: 444 444

  Herr Gruebel
  Runzelgasse 33
  3333 Faltenberg 33
  Tel.: 333 333
```

2. Aufruf von *diff3* mit Option *-e*:

```
$ diff3 -e adr1 adr2 adr3
9c
Tel.: 444 444

Herr Gruebel
Runzelgasse 33
3333 Faltenberg 33
Tel.: 333 333
.
w
q
```

Wenn Sie die Ausgabe von *diff3 -e* in eine Datei *script* umlenken, können Sie mit der folgenden Kommandozeile die Änderungen in *adr1* einarbeiten:

```
$ (cat script; echo '1,$p') | ed - adr1
```

3. Aufruf von *diff3* mit Option *-3*:

```
$ diff3 -3 adr1 adr2 adr3
```

Dieser Aufruf erzeugt keine Ausgabe, da *adr1* und *adr2* verschieden sind. *diff3 -3* gibt nur dann ein *ed*-Skript aus, wenn es Unterschiede der Art ====3 gibt; in diesem Beispiel gibt es aber nur Unterschiede der Art ====.

SIEHE AUCH

diff, *ed*

dircmp
Dateiverzeichnisse vergleichen (directory comparison)

dircmp vergleicht zwei Dateiverzeichnisse und gibt aus, ob sich in den beiden Dateiverzeichnissen *dvz1* und *dvz2* Dateien mit gleichem Namen befinden und ob Dateien mit gleichem Namen auch den gleichen Inhalt haben.

dircmp[␣-d][␣-s][␣-wn]␣dvz1␣dvz2

Keine Option angegeben
dircmp gibt zuerst die Namen der Dateien aus, die nur in einem der beiden Dateiverzeichnisse *dvz1 dvz2* stehen: links die Dateinamen aus *dvz1*, rechts die Dateinamen aus *dvz2*. Die Ausgabe hat folgendes Format:

```
Nov  16  10:42  1987  dvz1 only dvz2 only Page 1
./dateiname1              ./dateiname2
        .                         .
        .                         .
        .                         .
        .                         .
```

Die daran anschließenden Ausgabezeilen geben Auskunft darüber, ob die Dateien mit gleichem Namen in beiden Dateiverzeichnissen den gleichen Inhalt haben. Wenn ja, stehen sie in einer mit *same* (gleich), wenn nein, in einer mit *different* (verschieden) beginnenden Zeile. Die Ausgabe hat folgendes Format:

```
directory .
different ./dateiname in dvz1 und dvz2
        .
        .
        .
same      ./dateiname in dvz1 und dvz2
```

-d
dircmp führt einen Vergleich des Inhalts der Dateien mit gleichem Namen aus den Dateiverzeichnissen *dvz1* und *dvz2* durch und gibt das Ergebnis aus wie bei dem Kommando *diff*. Das Ergebnis dieses Vergleichs wird im Anschluß an die Ausgabe des Vergleichs des Inhalts von *dvz1* und *dvz2* ausgegeben.

-s
dircmp gibt nicht aus, welche Dateien mit gleichem Namen auch den gleichen Inhalt haben.

-wn
Die Seitenbreite wird auf *n* Zeichen geändert. Standardmäßig werden 72 Zeichen pro Zeile ausgegeben.

dvz1 dvz2
Namen der Dateiverzeichnisse, die Sie vergleichen möchten.

BEISPIEL

Vergleich der beiden Dateiverzeichnisse *A* und *B*, die folgenden Inhalt haben:

Dateiverzeichnis	Dateinamen	Inhalt der Dateien
A	a	aaaaaaa
	b	bbbb
	c	cccc
	d	dddd
B	a	aaaaaaaaa
	b	BBBB
	C	cccc
	d	dddd

Aufruf von *dircmp* mit Option *-d*:

```
$ dircmp -d A B
Nov  16  10:42  1987  A only and B only Page 1

./c                     ./C


Nov  16  10:42  1987  Comparison of A B Page 1

directory .
different ./a
different ./b
same      ./d


Nov  16  10:42  1987  difference of ./a in A and B Page 1

1c1
< aaaaaaa
---
> aaaaaaaaa


Nov  16  10:42  1987  difference of ./b in A and B Page 1

1c1
< bbbb
---
> BBBB
```

SIEHE AUCH

cmp, *diff*

dirname
Pfad-Präfix vom Dateinamen trennen

Mit *dirname* können Sie das Pfad-Präfix vom einfachen Dateinamen trennen. *dirname* entfernt aus einer Zeichenkette alle Zeichen ab dem letzten Schrägstrich / einschließlich und gibt den Rest auf die Standard-Ausgabe aus. *dirname* ist nützlich in Shell-Prozeduren zwischen Gegenhochkommata `...` (siehe *Tabellen und Verzeichnisse, Sonderzeichen der Bourne-Shell sh*).

dirname_zeichenkette

zeichenkette
: *zeichenkette* ist eine beliebige Zeichenkette.
dirname löscht alle Zeichen ab dem letzten / einschließlich dieses Schrägstrichs und schreibt den Rest auf die Standard-Ausgabe. Zeichenketten, die keinen Schrägstrich / enthalten, werden also unverändert ausgegeben.

BEISPIEL

Im folgenden Beispiel wird der Variablen *NAME* als Wert das Pfad-Präfix */usr/src/cmd* zugewiesen:

```
NAME=`dirname /usr/src/cmd/xyz.c`
```

SIEHE AUCH

basename, *ed*, *sh*

doscat
Dateien einer MS-DOS-Diskette ausgeben

doscat gibt eine oder mehrere Dateien einer MS-DOS-Diskette auf die Standard-Ausgabe des SINIX-Systems aus.

doscat[␣**-m**][␣**-r**]␣**a**:dospfad...

Keine Option angegeben
MS-DOS-Textdateien enthalten für jeden Zeilenvorschub die Zeichenfolge CR-NL, SINIX-Dateien nur das Zeichen NL. Die Zeichenfolge CR-NL wird in NL umgesetzt. Das DOS-Dateiendezeichen SUB("1A") wird eliminiert. Aus einer MS-DOS-Textdatei wird eine SINIX-Textdatei. Bei Binärdateien entfällt eine Zeichenumsetzung.

-m
Die Zeichenumsetzung wird erzwungen; d.h., auch bei Binärdateien werden die oben beschriebenen Zeichenumwandlungen vorgenommen.

-r
doscat überträgt Dateien "roh"; d.h., alle Zeichen bleiben unverändert (auch bei Textdateien).

a:
A:
a: oder *A:* bezeichnet nach DOS-Konvention das Laufwerk, über das die MS-DOS-Diskette angesprochen werden kann. Dieser Name muß vor jedem DOS-Pfadnamen angegeben werden. *a:* bzw. *A:* greift auf das erste Diskettenlaufwerk zu. Existiert ein zweites Laufwerk, so kann auf dieses mit *b:* bzw. *B:* zugegriffen werden.

dospfad
gibt den Pfadnamen der gewünschten Datei im MS-DOS-Dateisystem an.
Die Verwendung von Sonderzeichen für Dateinamen (*,?,[ ]) ist erlaubt; der damit angegebene Pfadname muß jedoch eindeutig sein.

BEISPIELE

1. Welchen Inhalt hat die MS-DOS-Datei *BUECHER*?

```
$ doscat a:buecher
Hier werden alle Buecher aufgefuehrt, die in der Firma angeschafft wurden.

Autor          Titel                          Anschaffungs-   Preis
                                              datum
-------------------------------------------------------------------
$
```

2. Sie wollen den Inhalt der MS-DOS-Dateien *BENUTZER/MEIER/SPENDEN* und *SPIELE/FISCH* lesen.

```
$ doscat a:benutzer/meier/spenden a:spiele/fisch
SPENDEN seit 01.01.1986
FISCH-SPIEL:
Wenn dieser Bildschirm ein Aquarium waere,
koennte man darin
einen kleinen Fisch schwimmen lassen!
$
```

doscat hat hintereinander den Inhalt der Dateien ausgegeben.

doscp
Dateien zwischen SINIX und MS-DOS kopieren

doscp kopiert eine oder mehrere SINIX-Dateien auf eine MS-DOS-Diskette bzw. MS-DOS-Dateien von der MS-DOS-Diskette nach SINIX.

Bei dem Kommando *doscp* sind folgende 4 Formate zu unterscheiden:

- eine SINIX-Datei nach MS-DOS kopieren (Format 1)
- mehrere SINIX-Dateien in ein MS-DOS-Dateiverzeichnis kopieren (Format 2)
- eine MS-DOS-Datei nach SINIX kopieren (Format 3)
- mehrere MS-DOS-Dateien in ein SINIX-Dateiverzeichnis kopieren (Format 4)

doscp[␣**-m**][␣**-r**]␣sidatei␣**a:**dospfad	Format 1
doscp[␣**-m**][␣**-r**]␣sidatei...␣**a:**dosverzeichnis	Format 2
doscp[␣**-m**][␣**-r**]␣**a:**dosdatei␣sipfad	Format 3
doscp[␣**-m**][␣**-r**]␣**a:**dosdatei...␣slverzeichnis	Format 4

Keine Option angegeben
MS-DOS-Textdateien enthalten für jeden Zeilenvorschub die Zeichenfolge CR-NL, SINIX-Dateien nur das Zeichen NL. Beim Kopieren von MS-DOS nach SINIX wird die Zeichenfolge CR-NL in NL umgesetzt. Das DOS-Dateiendezeichen SUB("1A") wird eliminiert.
Beim Kopieren von SINIX nach MS-DOS wird die Zeichenfolge NL in CR-NL umgesetzt. Das DOS-Dateiendezeichen SUB("1A") wird eingefügt.
Aus einer MS-DOS-Textdatei wird eine SINIX-Textdatei oder umgekehrt. Bei Binärdateien entfällt eine Zeichenumsetzung.

Hinweis
Wenn Sie sicher gehen wollen, daß eine Zeichenumsetzung erzwungen bzw. ausgeschlossen wird, ist es besser, die Optionen *-m* bzw. *-r* anzugeben.

-m
Die Zeichenumsetzung wird erzwungen; d.h., auch bei Binärdateien werden die oben beschriebenen Zeichenumwandlungen vorgenommen.

-r
doscp überträgt Dateien "roh"; d.h. alle Zeichen bleiben unverändert (auch bei Textdateien).

sidatei
Name der SINIX-Datei

a:
A:

a: oder *A:* bezeichnet nach DOS-Konvention das Laufwerk, über das die MS-DOS-Diskette angesprochen werden kann. Dieser Name muß vor jedem DOS-Pfadnamen angegeben werden. *a:* bzw. *A:* greift auf das erste Diskettenlaufwerk zu. Existiert ein zweites Laufwerk, so kann auf dieses mit *b:* bzw. *B:* zugegriffen werden.

dospfad
Pfadname des MS-DOS-Dateiverzeichnisses der MS-DOS-Diskette, in das die SINIX-Datei kopiert werden soll. Die Verwendung von Sonderzeichen für Dateinamen (*,?,[ ]) ist erlaubt; der damit angegebene Pfadname muß jedoch eindeutig sein.

Wenn bereits eine MS-DOS-Datei mit diesem Namen vorhanden ist, kann man durch *dospfad* der SINIX-Datei im MS-DOS-System einen anderen Namen geben.

dospfad nicht angegeben:
doscp kopiert die SINIX-Datei in das oberste Dateiverzeichnis der MS-DOS-Diskette.

dosverzeichnis
Name des MS-DOS-Dateiverzeichnisses, in das die SINIX-Datei oder -Dateien kopiert werden sollen. Die Verwendung von Sonderzeichen für Dateinamen (*,?,[ ]) ist erlaubt; das damit angegebene Dateiverzeichnis muß jedoch eindeutig sein.

dosdatei
Name bzw. Pfadname der MS-DOS-Datei auf der MS-DOS-Diskette. Die Verwendung von Sonderzeichen für Dateinamen (*,?,[ ]) ist erlaubt.

sipfad
Name des SINIX-Dateiverzeichnisses oder der SINIX-Datei

siverzeichnis
Name des SINIX-Dateiverzeichnisses

Vorsicht

Für die Wahl von MS-DOS-Dateinamen sind bestimmte Konventionen zu beachten, damit die Daten auch auf einem DOS-System gelesen werden können. MS-DOS-Dateinamen bestehen aus einer Basis von höchstens 8 Zeichen, an die ein Punkt, gefolgt von einem Suffix von höchstens 3 Zeichen gehängt werden kann. Eine Basis von mehr als 8 Zeichen wird auf 8 Zeichen verkürzt, ein Suffix von mehr als 3 Zeichen auf 3 Zeichen. Bestimmte Suffixe besitzen in DOS eine Sonderbedeutung. Es können nicht alle Sonderzeichen wie bei SINIX verwendet werden. Im Fehlerfall werden Sie über eine entsprechende Meldung informiert. DOS unterscheidet nicht zwischen Groß- und Kleinbuchstaben. Wenn Sie z.B. Namen mit Kleinbuchstaben angeben, werden diese auf der Diskette in Großbuchstaben umgewandelt. Beim nächsten Lesen der Diskette erscheinen die Namen dann in Großbuchstaben.

BEISPIELE

1. Die SINIX-Datei *termine* soll in das Dateiverzeichnis *benutzer/meier* auf die MS-DOS-Diskette kopiert werden.

```
$ doscp termine a:benutzer/meier
```

2. Die SINIX-Datei *termine* soll auf eine MS-DOS-Diskette kopiert und gleichzeitig in die Datei *april* umbenannt werden.

```
$ doscp termine a:april
```

3. Sie wollen alle SINIX-Dateien, deren Namen mit *max* beginnen, auf die MS-DOS-Diskette kopieren.

```
$ doscp max* a:
```

   Durch das Kommando *dosdir* oder *dosls* überprüfen Sie anschließend, ob sich die Dateien auf der MS-DOS-Diskette befinden.

```
$ dosdir a:
MAX      1        712   1-20-86   5:09p
MAX      2       8075   1-20-86   5:09p
MAX      3      18727   1-20-86   5:09p
MAX      4       3417   1-20-86   5:10p
```

4. Sie wollen die MS-DOS-Datei *BUECHER* in das SINIX-Dateiverzeichnis */usr/gast* kopieren und gleichzeitig in *bibliothek* umbenennen.

```
$ doscp a:BUECHER bibliothek
```

5. Sie wollen die Dateien *SPIELE/WURM* und *BENUTZER/APRIL* von der MS-DOS-Diskette in Ihr aktuelles SINIX-Dateiverzeichnis kopieren.

```
$ doscp a:spiele/wurm a:benutzer/april
```

6. Sie wollen die MS-DOS-Dateien *BUSCH/MAX* und *BUSCH/MORITZ* sowie die Dateien *MAERCHEN/HAENSEL* und *MAERCHEN/GRETEL* in das SINIX-Dateiverzeichnis *usr/wer/erzaehlungen* kopieren.

```
$ doscp a:busch/max a:busch/moritz a:maerchen/haensel a:maerchen/gretel
/usr/wer/erzaehlungen
```

dosdir
Inhalt von Dateiverzeichnissen einer MS-DOS-Diskette im MS-DOS-Format ausgeben

dosdir gibt den Inhalt eines oder mehrerer Dateiverzeichnisse einer MS-DOS-Diskette im MS-DOS-Format sowie den noch verfügbaren freien Speicherplatz aus.

dosdir_a:dospfad...

a:
A:
a: oder *A:* bezeichnet nach DOS-Konvention das Laufwerk, über das die MS-DOS-Diskette angesprochen werden kann. Dieser Name muß vor jedem DOS-Pfadnamen angegeben werden. *a:* bzw. *A:* greift auf das erste Diskettenlaufwerk zu. Existiert ein zweites Laufwerk, so kann auf dieses mit *b:* bzw. *B:* zugegriffen werden.

dospfad
gibt den Pfadnamen des gewünschten Dateiverzeichnisses der MS-DOS-Diskette an. Die Verwendung von Sonderzeichen für Dateinamen (*, ?, [ ]) ist erlaubt.

dospfad nicht angegeben:
dosdir gibt das oberste Dateiverzeichnis der MS-DOS-Diskette aus.

BEISPIELE

1. Von der MS-DOS-Diskette den Inhalt des obersten Dateiverzeichnisses ausgeben:

```
$ dosdir a:
SPIELE       <DIR>    530   1-15-86    2:58p
BIN          <DIR>   1292   1-15-86    2:58p
BENUTZER     <DIR>    478   1-15-86    2:58p
BUCHFRNG     <DIR>   2249   1-15-86    2:58p
PROGRAMM     <DIR>   8463   1-15-86    2:58p
       5 File(s)    161472 Bytes free
```

2. Von der MS-DOS-Diskette den Inhalt eines untergeordneten Dateiverzeichnisses ausgeben:

```
$ dosdir a:spiele
.            <DIR>      728  1-15-86   2:58p
..           <DIR>     5153  1-15-86   2:58p
FISCH              1    438  1-15-86   3:09p
KNIFFEL            1    917  1-15-86   3:09p
RECHNEN            1   2384  1-15-86   3:09p
SPIRALE            1   6725  1-15-86   3:09p
WURM               1    839  1-15-86   3:09p
ENTERPR  MSD       1   1277  1-15-86   3:20p
MASTERM  MSD       1   1082  1-15-86   3:20p
    9 File(s)        78278 Bytes free
```

3. Von der MS-DOS-Diskette den Inhalt mehrerer Dateiverzeichnisse ausgeben:

```
$ dosdir a:spiele a:benutzer
/SPIELE:
     .            <DIR>          ...
     ..           <DIR>          ...
     FISCH              1        ...
     KNIFFEL            1        ...
     RECHNEN            1        ...
     SPIRALE            1        ...
     WURM               1        ...
     ENTERPR  MSD       1        ...
     MASTERM  MSD       1        ...
/BENUTZER:
     .            <DIR>          ...
     ..           <DIR>          ...
     MEIER        <DIR>          ...
```

dosfilt
Dateien mit sprachabhängigen Sonderzeichen für MS-DOS lesbar machen

dosfilt wandelt SINIX-Dateien mit sprachabhängigen Sonderzeichen im 7-bit-Format (wie z.B. ä, ö, ü, ß) bzw. mit Daten im 8-bit-Zeichensatz ISO 8859-1 derart um, daß sie für MS-DOS lesbar und bearbeitbar werden (IBM-Zeichensatz). NL wird in die Zeichenfolge CR-NL umgesetzt und das DOS-Dateiendezeichen SUB ("1A") angehängt. *dosfilt* schreibt standardmäßig auf die Standard-Ausgabe.

dosfilt -sprache[-o ausdatei][datei...]

sprache
Sprache bzw. Zeichensatz der SINIX-Datei, die in den IBM-Zeichensatz umgewandelt werden soll.
Sie können unter folgenden Sprachen bzw. Zeichensätzen wählen:

- **a** – dänisch
- **b** – belgisch
- **d** – deutsch
- **e** – spanisch
- **f** – französisch
- **g** – britisch
- **i** – italienisch
- **n** – norwegisch
- **s** – schwedisch
- **z** – schweizerisch
- **x** – Mit dieser Option setzen Sie die Zeichen einer umzuwandelnden Datei aus dem Zeichensatz ISO 8859-1 in die Zeichen des IBM-Zeichensatzes um.

-o ausdatei
Name der Ausgabedatei

datei
Name der umzuwandelnden Datei

datei nicht angegeben:
dosfilt liest von Standard-Eingabe

Um MS-DOS-Dateien in SINIX-Dateien umzuwandeln, benutzen Sie das Kommando *sinfilt*.

Vorsicht

dosfilt -x setzt die angegebenen Dateien vom Zeichensatz ISO 8859-1 in den IBM-Zeichensatz um. Die beiden Zeichensätze besitzen allerdings nicht den gleichen Zeichenvorrat. Daher ist folgendes zu beachten:

- 52 Zeichen (inklusive aller ASCII-Zeichen und deutschen Umlaute) können direkt 1 : 1 übertragen werden.
- Einie Reihe nationaler Sonderzeichen des Zeichensatzes ISO 8859-1 (insgesamt 44 Zeichen) können nicht direkt in identische Zeichen des IBM-Zeichensatzes umgewandelt werden, da diese dort nicht enthalten sind. Sie werden durch ähnliche Zeichen ersetzt.
- Sämtliche Semigraphikzeichen, viele mathematische und griechische Zeichen des IBM-Zeichensatzes (insgesamt 63 Zeichen mit Hexwerten größer als 7f) sind nicht im Zeichensatz ISO 8859-1 enthalten und können deshalb nicht aus dem ISO-Zeichensatz erzeugt werden.

BEISPIEL

Umwandeln der SINIX-Datei *termine* im Zeichensatz ISO 8859-1 in die MS-DOS-Datei *TERMINE* im IBM-Zeichensatz:

```
$ dosfilt -x -o TERMINE termine
```

SIEHE AUCH

sinfilt

dosformat
MS-DOS-Diskette formatieren

dosformat formatiert eine Diskette für Rechner mit dem Betriebssystem MS-DOS. Die möglichen Formate sind weiter unten angegeben.

dosformat[_option]_gerätedatei

Keine Option angegeben
dosformat formatiert die Diskette in dem Format, das durch *gerätedatei* bestimmt wird.

option

-f
unterdrückt die Abfrage, ob eine Diskette im Laufwerk eingelegt ist, und beginnt gleich mit der Formatierung.

-q
unterdrückt Informationen, die normalerweise während des Formatiervorgangs am Bildschirm ausgegeben werden. Die Abfrage, ob eine Diskette im Laufwerk eingelegt ist, wird nicht unterdrückt.

-v
dosformat fragt nach einem Diskettennamen (volume-label) zur näheren Bezeichnung der Diskette. Die Länge des Diskettennamens kann maximal 11 Zeichen betragen. Der Name wird bei *dosdir* ausgegeben.

gerätedatei
Hier müssen Sie einen der in der Tabelle (siehe unten) aufgeführten Gerätedateinamen angeben. Dieser legt das Disketten-Format fest.

Sie können folgende Disketten-Formate erzeugen:

Format-bezeichnung	Spuren	Sektoren	Brutto-Speicher-kapazität	Netto-Speicher-kapazität	Gerätedateinamen (bei Formatierung anzugeben)	Disketten-grösse
	80	9	720 KB	713 KB	/dev/rdsk/f03d9t	3 1/2 Zoll
	80	18	1440 KB	1423,5 KB	/dev/rdsk/f03ht	3 1/2 Zoll
PC-D	80	9	720 KB	713 KB	/dev/rdsk/f15qt	5 1/4 Zoll
PC-D-2/3+AT	80	15	1200 KB	1185,5 KB	/dev/rdsk/f15ht	5 1/4 Zoll
XT	40	8	320 KB	315 KB	/dev/rdsk/f15d8t	5 1/4 Zoll
XT	40	9	360 KB	354 KB	/dev/rdsk/f15d9t	5 1/4 Zoll

Einseitig formatierte Disketten können weder erzeugt, gelesen, noch beschrieben werden.

Die Netto-Speicherkapazität gibt den tatsächlich nutzbaren Speicherbereich der Diskette an. Der Rest enthält Verwaltungsinformationen über die Diskette. Da die Dateien immer in Blöcken von 512 byte / 1 KB oder 2 KB (je nach Format) auf der Diskette gespeichert werden, kann bei der Kopie von vielen kleinen Dateien die Netto-Speicherkapazität auch erheblich unter den in der Tabelle angegebenen Werten liegen.

Die so formatierte Diskette können Sie mit den DOS-Kommandos am SINIX-Computer bearbeiten und am MS-DOS-Rechner verwenden.

Vorsicht

dosformat arbeitet ohne Rücksicht auf eventuelle Inhalte der Diskette. Haben Sie also bereits Dateien auf der Diskette gespeichert, werden diese durch das Formatieren überschrieben.

Wenn Sie eine Diskette, die in einem XT-Disketten-Laufwerk mit 40 Spuren formatiert wurde, an einem SINIX-Computer erneut mit 40 Spuren formatieren und danach beschreiben, können beim Lesen an einem XT-Rechner Lesefehler auftreten, da die beiden Systeme unterschiedliche Spurbreiten verwenden. Deshalb sollten Sie nur unformatierte bzw. AT- oder SINIX-formatierte Disketten für diese Zwecke verwenden.

BEISPIEL

Mit diesem Kommando formatieren Sie eine 3 1/2 Zoll-Diskette im ersten Laufwerk mit 80 Spuren und 18 Sektoren pro Spur. Die Diskette hat dann eine Speicherkapazität von 1440 KB.

```
$ dosformat /dev/rdsk/f03ht
```

dosls
Inhalt von Dateiverzeichnissen einer MS-DOS-Diskette im SINIX-Format ausgeben

dosls gibt den Inhalt eines oder mehrerer Dateiverzeichnisse einer MS-DOS-Diskette im SINIX-Format aus.

dosls a:dospfad...

a:
A:
a: oder *A:* bezeichnet nach DOS-Konvention das Laufwerk, über das die MS-DOS-Diskette angesprochen werden kann. Dieser Name muß vor jedem DOS-Pfadnamen angegeben werden. *a:* bzw. *A:* greift auf das erste Diskettenlaufwerk zu. Existiert ein zweites Laufwerk, so kann auf dieses mit *b:* bzw. *B:* zugegriffen werden.

dospfad
gibt den Pfadnamen des gewünschten Dateiverzeichnisses der MS-DOS-Diskette an. Die Verwendung von Sonderzeichen für Dateinamen (*,?,[ ]) ist erlaubt.

dospfad nicht angegeben:
dosls gibt das oberste Dateiverzeichnis der MS-DOS-Diskette an.

BEISPIELE

1. Den Inhalt des obersten Dateiverzeichnisses der MS-DOS-Diskette ausgeben.

```
$ dosls a:
SPIELE
BIN
BENUTZER
BUCHFRNG
PROGRAMM
```

2. Von der MS-DOS-Diskette den Inhalt eines untergeordneten Dateiverzeichnisses ausgeben:

```
$ dosls a:benutzer
.
..
MEIER
```

3. Von der MS-DOS-Diskette den Inhalt mehrerer Dateiverzeichnisse ausgeben:

```
$ dosls a:benutzer a:spiele
/SPIELE:
      .
      ..
      FISCH
      KNIFFEL
      RECHNEN
      SPIRALE
      WURM
      ENTERPR  MSD
      MASTERM  MSD
/BENUTZER:
      .
      ..
      MEIER
```

dosmkdir
Dateiverzeichnisse auf einer MS-DOS-Diskette einrichten

dosmkdir richtet ein oder mehrere Dateiverzeichnisse auf einer MS-DOS-Diskette ein.

dosmkdir_a:dospfad...

a:
A:
a: oder *A:* bezeichnet nach DOS-Konvention das Laufwerk, über das die MS-DOS-Diskette angesprochen werden kann. Dieser Name muß vor jedem DOS-Pfadnamen angegeben werden. *a:* bzw. *A:* greift auf das erste Diskettenlaufwerk zu. Existiert ein zweites Laufwerk, so kann auf dieses mit *b:* bzw. *B:* zugegriffen werden.

dospfad
gibt den Pfadnamen des neuen Dateiverzeichnisses im MS-DOS-Dateisystem an.

Für die Wahl von MS-DOS-Dateiverzeichnis-Namen sind bestimmte Konventionen zu beachten, damit die Daten auch auf einem DOS-System gelesen werden können. Dateiverzeichnis-Namen mit einer Basis von mehr als 8 Zeichen werden auf 8 Zeichen verkürzt, Suffixe von mehr als 3 Zeichen auf 3 Zeichen. Der Name eines MS-DOS-Dateiverzeichnisses besteht also aus einer Basis mit höchstens 8 Zeichen Länge und einem optionalen Suffix mit höchstens 3 Zeichen Länge. Das Suffix wird durch einen Punkt von der Basis getrennt. Es können nicht alle Sonderzeichen wie bei SINIX verwendet werden. Im Fehlerfall werden Sie über eine entsprechende Meldung informiert. DOS unterscheidet nicht zwischen Groß- und Kleinbuchstaben. Wenn Sie z.B. Namen mit Kleinbuchstaben angeben, werden diese auf der Diskette in Großbuchstaben umgewandelt. Beim nächsten Lesen der Diskette erscheinen die Namen dann in Großbuchstaben.

dosmkdir richtet pro Dateiverzeichnis folgende Verweise ein:

"." für das Dateiverzeichnis selbst und

".." für das übergeordnete Dateiverzeichnis.

BEISPIELE

1. Das Dateiverzeichnis *adressen* ist einzurichten:

```
$ dosmkdir a:adressen

$ dosls a:
SPIELE
BIN
BENUTZER
BUCHFRNG
PROGRAMM
ADRESSEN
```

dosls zeigt, daß *dosmkdir* ausgeführt wurde.

2. Mehrere Dateiverzeichnisse einrichten:

Dem Dateiverzeichnis *ADRESSEN*, das im 1.Beispiel eingerichtet wurde, sollen weitere Dateiverzeichnisse zugeordnet werden. Die Dateiverzeichnisse sollen *MUENCHEN* und *BAYERN* heißen. Ferner soll im Dateiverzeichnis *BENUTZER/MEIER* das Dateiverzeichnis *SPENDEN* eingerichtet werden.

Sie geben folgendes Kommando ein:

```
$ dosmkdir a:adressen/muenchen a:adressen/bayern a:benutzer/meier/spenden

$ dosdir a:adressen a:benutzer/meier
/ADRESSEN:
     .            <DIR>
     ..           <DIR>
     MUENCHEN     <DIR>
     BAYERN       <DIR>
/BENUTZER/MEIER:
     .            <DIR>
     ..           <DIR>
     SPENDEN      <DIR>
```

dosdir zeigt, daß das Kommando dosmkdir ausgeführt wurde und die neuen Dateiverzeichnisse eingerichtet sind.

dosrm
MS-DOS-Dateien löschen

dosrm löscht eine oder mehrere MS-DOS-Dateien auf der MS-DOS-Diskette. Die Dateien müssen dabei mit vollem Pfadnamen angegeben werden. Beachten Sie, daß kein Schutz gegen ein unberechtigtes Löschen der Datei besteht. Sie haben auch keine Möglichkeit, eine bereits gelöschte Datei zurückzuholen.

dosrm_a:dosdatei...

a:
A:
a: oder *A:* bezeichnet nach DOS-Konvention das Laufwerk über das die MS-DOS-Diskette angesprochen werden kann. Dieser Name muß vor jedem DOS-Pfadnamen angegeben werden. *a:* bzw. *A:* greift auf das erste Diskettenlaufwerk zu. Existiert ein zweites Laufwerk, so kann auf dieses mit *b:* bzw. *B:* zugegriffen werden.

dosdatei
Name bzw. Pfadname der MS-DOS-Datei im MS-DOS-Dateisystem. Die Verwendung von Sonderzeichen für Dateinamen (*,?,[ ]) ist erlaubt.

BEISPIELE

1. Sie wollen die MS-DOS-Datei *PFERDE* löschen.

```
$ dosrm a:pferde
```

2. Sie wollen die MS-DOS-Datei mit dem Pfadnamen *KOERPER/HERZ* auf der MS-DOS-Diskette löschen.

```
$ dosrm a:koerper/herz
```

3. Sie wollen die MS-DOS-Dateien *POST*, *KASSE* und *PFAND* auf der MS-DOS-Diskette löschen.

```
$ dosrm a:post a:kasse a:pfand
```

4. Sie wollen die MS-DOS-Dateien mit den Pfadnamen *BLUMEN/CLIVIE*, *TIERE/SAEUGER/IGEL* sowie *GESTEINE/GRANIT* löschen.

```
$ dosrm a:blumen/clivie a:tiere/saeuger/igel a:gesteine/granit
```

dosrmdir
Dateiverzeichnisse im MS-DOS-Dateisystem löschen

dosrmdir löscht eines oder mehrere Dateiverzeichnisse des MS-DOS-Dateisystems auf der MS-DOS-Diskette. Die Dateiverzeichnisse müssen leer sein.

dosrmdir [-a:]dosdir...

a:
A:
a: oder *A:* bezeichnet nach DOS-Konvention das Laufwerk, über das die MS-DOS-Diskette angesprochen werden kann. Dieser Name muß vor jedem DOS-Pfadnamen angegeben werden. *a:* bzw. *A:* greift auf das erste Diskettenlaufwerk zu. Existiert ein zweites Laufwerk, so kann auf dieses mit *b:* bzw. *B:* zugegriffen werden.

dosdir
gibt den Pfadnamen des Dateiverzeichnisses an, das Sie im MS-DOS-Dateisystem löschen wollen. Die Verwendung von Sonderzeichen für Dateinamen (*,?,[ ]) ist erlaubt.

BEISPIELE

1. Ein einzelnes Dateiverzeichnis löschen:

   Auf Ihrer MS-DOS-Diskette sind im Dateiverzeichnis *BENUTZER* folgende Dateiverzeichnisse eingerichtet:

   ```
   $ dosls a:benutzer
   .
   ..
   MEIER
   KUHN
   ```

   Das Dateiverzeichnis *BENUTZER/KUHN* ist zu löschen.

   ```
   $ dosrmdir a:benutzer/kuhn
   ```

2. Mehrere Dateiverzeichnisse löschen:

   Das Dateiverzeichnis *BENUTZER* enthält die Angaben

```
$ dosdir a:benutzer
.            <DIR>   476   1-16-86  10:43a
..           <DIR>   263   1-16-86  10:43a
MEIER        <DIR>  2745   1-16-86  10:43a
KUHN         <DIR>   834   1-16-86  11:18a
LOBER        <DIR>  1476   1-16-86  11:19a
       5 File(s)    ... Bytes free
```

   Das Dateiverzeichnis *ADRESSEN/SONSTIGE* enthält die Angaben

```
$ dosdir a:adressen/sonstige
.            <DIR>   273   1-16-86  11:17a
..           <DIR>   185   1-16-86  11:17a
PLZ1         <DIR>   947   1-16-86  12:39p
PLZ7         <DIR>   204   1-16-86  12:39p
       4 File(s)    ... Bytes free
```

   Aus dem Dateiverzeichnis *BENUTZER* soll das Dateiverzeichnis *LOBER* und aus dem Dateiverzeichnis *ADRESSEN/SONSTIGE* das Dateiverzeichnis *PLZ* gelöscht werden. Anschließend lassen Sie sich durch *dosls* den Inhalt der beiden Dateiverzeichnisse noch einmal ausgeben.

```
$ dosrmdir a:benutzer/lober a:adressen/sonstige/plz1
```

```
$ dosls a:benutzer a:adressen/sonstige
/BENUTZER:
     .
     ..
     MEIER
     KUHN
/ADRESSEN/SONSTIGE:
     .
     ..
     PLZ7
```

   dosls zeigt, daß die Dateiverzeichnisse gelöscht sind.

du
Belegten Speicherplatz ausgeben (display used blocks)

du gibt die Speicherplatz-Belegung durch Dateiverzeichnisse, Unter-Dateiverzeichnisse und einfache Dateien mit der Anzahl der belegten Blöcke von 512 byte aus.

du[_option]...[_datei]...

Keine Option angegeben

Wenn *datei* ein Dateiverzeichnis ist, listet *du* den Speicherplatz auf, den das Dateiverzeichnis und alle seine Unterverzeichnisse belegen. Der von den einfachen Dateien im angegebenen Dateiverzeichnis belegte Speicherplatz ist enthalten, wird aber nicht einzeln aufgelistet.
Wenn *datei* kein Dateiverzeichnis ist, listet *du* nichts auf.

option

-a

Wenn *datei* ein Dateiverzeichnis ist, listet *du* den belegten Speicherplatz für alle Dateien dieses Dateiverzeichnisses einzeln auf. Wenn *datei* kein Dateiverzeichnis ist, listet *du* den von *datei* belegten Speicherplatz auf.

-r

du gibt eine Fehlermeldung aus, wenn *datei* ein Dateiverzeichnis ist, für das Sie kein Leserecht haben oder eine Datei, die nicht geöffnet werden kann.

-r nicht angegeben:
du gibt in den o.g. Fällen keine Fehlermeldung aus.

-s

du gibt nur die Gesamtsumme des Speicherplatzes aus, den der Teil-Dateibaum oder die Datei belegt.

datei

Name der Datei bzw. des Dateiverzeichnisses, für die/das die Speicherplatz-Belegung ausgegeben werden soll. Eine Datei, auf die zwei oder mehr Verweise vorhanden sind, wird nur einmal gezählt. Eine Datei mit nicht verwendeten Blöcken (z.B. nur Block 1 und 100 geschrieben) führt zu falschen Ergebnissen. Ist *datei* eine einfache Datei, gibt *du* ohne Optionen nichts aus.

datei nicht angegeben:
Der vom aktuellen Dateiverzeichnis und allen Unterverzeichnissen belegte Speicherplatz wird ausgegeben.

BEISPIELE

1. Auflisten der Anzahl Speicherblöcke von 512 byte, die von den Unter-Dateiverzeichnisses des aktuellen Dateiverzeichnisses belegt werden und deren Namen mit *DIR* beginnen. Der von einfachen Dateien belegte Speicherplatz ist enthalten, wird aber nicht einzeln aufgelistet.

```
$ du DIR*
6       DIR-1
136     DIR-2/DVZ-1
140     DIR-2
5       DIR-3
54      DIR-4
52      DIR-5
```

2. Auflisten der Anzahl Speicherblöcke von 512 byte, die von den Unter-Dateiverzeichnisses des aktuellen Dateiverzeichnisses belegt werden und deren Namen mit *DIR* beginnen. Der von einfachen Dateien belegte Speicherplatz wird mit Option *-a* einzeln aufgelistet.

```
$ du -a DIR*
1       DIR-1/datei1
1       DIR-1/datei2
1       DIR-1/datei3
6       DIR-1
0       DIR-2/datei4
1       DIR-2/datei5
34      DIR-2/DVZ-1/datei6
99      DIR-2/DVZ-1/datei7
136     DIR-2/DVZ-1
140     DIR-2
1       DIR-3/datei8
1       DIR-3/datei9
5       DIR-3
50      DIR-4/datei10
1       DIR-4/datei10.bak
54      DIR-4
50      DIR-5/datei11
52      DIR-5
```

SIEHE AUCH

df

dumpmsg
Meldungstext-Datei aus Meldungskatalog-Datei erzeugen (dump message)

dumpmsg liest einen formatierten Meldungskatalog aus einer Datei oder von der Standard-Eingabe und erzeugt daraus lesbaren Meldungsquelltext. Diesen Meldungsquelltext schreibt *dumpmsg* entweder in eine Datei oder auf die Standard-Ausgabe. *dumpmsg* gibt nur die Meldungen selbst aus, nicht die zugehörigen Begrenzungszeichen (z.B. Anführungszeichen).

dumpmsg[_-C]_meldungskatalog_meldungstextdatei

-C
Zeichen, die nicht im ASCII-Zeichensatz enthalten sind, werden oktal in der Form \xxx ausgegeben.

meldungskatalog
Name der Datei, die den formatierten Meldungskatalog enthält.
Wenn Sie für *meldungskatalog* einen Bindestrich - angeben, liest *dumpmsg* von der Standard-Eingabe.

meldungstextdatei
Name der Datei, in die *dumpmsg* den Meldungsquelltext schreiben soll.
Wenn Sie für *meldungstextdatei* einen Bindestrich - angeben, schreibt *dumpmsg* auf die Standard-Ausgabe.

INTERNATIONALE UMGEBUNG

Wenn Sie nicht in einer englischen, sondern in einer anderssprachigen Umgebung arbeiten, dann gibt *dumpmsg* die Meldungstexte in dieser Sprache aus. Die Sprache wird durch die NLS-Umgebungsvariable *LANG* definiert.

Hat eine der Variablen *LANG, LC_CTYPE, LC_TIME, LC_COLLATE, LC_NUMERIC* und *LC_MONETARY* einen ungültigen Wert oder ist die Variable *LANG* nicht definiert bzw. leer, dann verhält sich *dumpmsg*, als wäre das System nicht internationalisiert, d.h., die Meldungstexte werden auf Englisch ausgegeben.

dumpmsg ist 8-bit-transparent.

BEISPIEL

Durch den Aufruf des Kommandos *exctract* wird die Meldungstextdatei *hallo.msf* erzeugt. Aus der Meldungstextdatei *hallo.msf* generiert anschließend das Kommando *gencat* den binär kodierten Meldungskatalog *hallo.cat*. Danach werden durch Aufruf von *dumpmsg* alle Meldungen aus diesem Katalog in lesbarer Form auf der Standard-Ausgabe (symbolisiert durch den Bindestrich -) ausgegeben:

```
$ cat hallo.c
main()
{
  printf("Hello world!");
}

$ extract hallo.c
...

$ gencat hallo.cat hallo.msf
$ dumpmsg hallo.cat -
$set 1

1 Hello world!
```

SIEHE AUCH

gencat, *extract*

echo
Aufruf-Argumente ausgeben

Das in die Bourne-Shell *sh* eingebaute Kommando *echo* schreibt seine Aufruf-Argumente auf die Standard-Ausgabe und beendet sich. Vor der Ausgabe passiert folgendes:

1. Wie bei jedem anderen Kommando, interpretiert die Shell die Kommando-Zeile und übergibt an *echo* die aufbereiteten Aufruf-Argumente (siehe *sh*). Argument-Trenner sind für die Shell Leer- und Tabulatorzeichen.
2. *echo* interpretiert noch verbliebene Sonderzeichen, die die Ausgabe steuern (siehe Beschreibung zu *argument*).
3. *echo* gibt die bearbeiteten Argumente wie folgt aus:
   - Die einzelnen Argumente werden durch ein Leerzeichen voneinander getrennt, auch wenn Sie beim Aufruf mehrere Argument-Trennzeichen zwischen den Argumenten eingegeben haben.
   - Nach dem letzten Argument wird ein Neue-Zeile-Zeichen ausgegeben.

Sie können mit *echo*

- den Inhalt von Shell-Variablen abfragen,
- Meldungen in Shell-Prozeduren erzeugen und so die Funktionsweise testen,
- testen, wie die Shell einen Kommando-Aufruf interpretiert, ohne daß das Kommando ausgeführt wird,
- Daten in eine Pipe schicken und testen, wie die Pipe diese Eingabe verarbeitet.

Neben dem eingebauten Kommando *echo* gibt es auch */usr/bin/echo*. Für die Ausführung von */usr/bin/echo* erzeugt die Shell einen neuen Prozeß. Sonst verhält sich */usr/bin/echo* wie *echo*.

echo[␣argument]...

argument
beliebige Zeichenkette, die jeweils durch Leer- oder Tabulatorzeichen begrenzt wird. Das letzte Argument wird durch ein Kommando-Trennzeichen abgeschlossen.

Sie können beliebig viele Argumente angeben, jeweils getrennt durch mindestens ein Tabulator- bzw. ein Leerzeichen.

Wie bei jedem anderen Kommando auch wird die Zeichenkette zunächst von der Shell interpretiert:

- Enthält die Zeichenkette die Zeichen Stern * bzw. Fragezeichen ? bzw. [...], so ersetzt die Shell diese Zeichenkette durch alle passenden Dateinamen im aktuellen Dateiverzeichnis. Paßt kein Dateiname, übergibt die Shell diese Zeichenkette unverändert an *echo*.
- 'zeichenkette'
  Die Hochkommas entwerten alle Sonderzeichen der Shell. Die Shell übergibt die Zeichenkette ohne Hochkommas als ein Argument an *echo*. Alle zwischen den Hochkommas eingegebenen Leer-, Tabulator- und Neue-Zeile-Zeichen bleiben erhalten.
- `zeichenkette`
  Die Shell führt *zeichenkette* als SINIX-Kommando aus und übergibt an *echo* die Ausgabe dieses Kommandos. Dabei interpretiert die Shell in der Ausgabe die Zeichen als Argument-Trennzeichen, die der Umgebungsvariablen IFS zugewiesen sind. Standardmäßig sind IFS das Leer-, das Tabulator- und das Neue-Zeile-Zeichen zugewiesen.
- "zeichenkette"
  Die Anführungszeichen entwerten alle Sonderzeichen der Shell bis auf Gegenschrägstrich \, Gegenhochkommas `...` und Dollarzeichen $. Die Shell übergibt die bearbeitete Zeichenkette als ein Argument an *echo*. Alle zwischen den Anführungszeichen eingegebenen Leer-, Tabulator- und Neue-Zeile-Zeichen bleiben erhalten.

Das eingebaute Kommando *echo* interpretiert zusätzlich die nachfolgend beschriebenen Steuerzeichen. Da der Gegenschrägstrich für die Shell eine Sonderbedeutung hat, muß er entwertet werden:

- Durch einen vorangestellten Gegenschrägstrich \
  Das gilt auch, wenn das Argument, das dieses Steuerzeichen enthält, in Anführungszeichen eingeschlossen ist.
- Durch Hochkommas '...'
  Ist das Argument, das dieses Steuerzeichen enthält, in Hochkommas eingeschlossen, so ist der Gegenschrägstrich bereits entwertet.

Beispiel

Eingabe	Ausgabe
`hallo\\bi`	`halli`
`hallo'\'bi`	`halli`
`'hallo\bi'`	`halli`
`"hallo\\bi"`	`halli`

Die folgenden Steuerzeichen beeinflussen die Ausgabe von *echo*, falls der Gegenschrägstrich entsprechend entwertet ist:

\b
(b - backspace) *echo* setzt die Schreibmarke um eine Spalte zurück. Ein nach dem Steuerzeichen eingegebenes Zeichen überschreibt das Zeichen über der Schreibmarke.
\b entspricht ⌫ bzw. CTRL H

\c
echo gibt nur aus, was bis zu diesem Steuerzeichen eingegeben wurde, und schließt die Ausgabe nicht mit einem Neue-Zeile-Zeichen ab.

\f
(f - form feed) Seitenvorschub; bei Ausgabe auf den Bildschirm wird dieses Steuerzeichen nicht umgesetzt.
echo schreibt das, was nach dem Steuerzeichen eingegeben wurde, auf die nächste Seite, z.B. wenn die Ausgabe von *echo* an den Drucker geschickt wird.
\f entspricht CTRL L

\n
(n - new line) Zeilenvorschub; *echo* schreibt das, was nach dem Steuerzeichen eingegeben wurde, in die nächste Zeile.
\n entspricht ↵ bzw. CTRL J

\r

(r - carriage return) Wagenrücklauf; *echo* setzt die Schreibmarke auf Spalte 1 der aktuellen Zeile. Die Zeichen, die nach dem Steuerzeichen eingegeben wurden, überschreiben die Zeichen, die *echo* auf den entsprechenden Spalten bereits ausgegeben hat.
\r entspricht CTRL M

\t

Steuerzeichen für Tabulator; *echo* setzt die Schreibmarke auf den nächsten Tabulator. Die Zeichen, die nach *\t* eingegeben wurden, schreibt *echo* ab dieser Spalte weiter.

Die Tabulator-Positionen sind abhängig von der verwendeten Datensichtstation. Für eine Siemens-Datensichtstation 97801 sind Tabulatoren auf folgende Spalten gesetzt:

1 9 17 25 33 41 49 57 65 73 79

\t entspricht ⇥ bzw. CTRL I

\v

Steuerzeichen für Vertikal-Tabulator; *echo* setzt die Schreibmarke auf den nächsten Vertikal-Tabulator. Normalerweise sind Vertikal-Tabulatoren nicht eingestellt (geräteabhängig).

Dieses Steuerzeichen entspricht CTRL K

\0n

n muß eine ein-, zwei- oder dreistellige Oktalzahl sein. Das Kommando *echo* gibt das entsprechende Zeichen aus (siehe *Tabellen und Verzeichnisse, Zeichensatz ISO 646*). Auf diese Weise können Sie beispielsweise Zeichen erzeugen, deren interner Code einen Wert größer 7f (hexadezimal) hat, auch wenn Sie keine 8-Bit-Datensichtstaion haben.
Für nicht darstellbare Zeichen wird ein Schmierzeichen ausgegeben (geräteabhängig).

Vorsicht

Wenn Sie nach einem oktal beschriebenen Zeichen eine Ziffer ausgeben wollen, dann müssen Sie die Oktalzahl dreistellig angeben.

Beispiel

Kommandos	Ergebnis	
`echo '\0337' | od -xc`	`0adf`	(hex)
	`337 \n`	(ascii)
`echo '\00337' | od -xc`	`371b 000a`	(hex)
	`033 7 \n`	(ascii)

Sollen diese Steuerzeichen weder von der Shell noch von *echo* interpretiert werden, müssen Sie sie wie folgt eingeben:

Eingabe	Ausgabe
`\\`	`das Zeichen \`
`\\\\b`	`die Zeichenkette \b`
`'\\b'`	`die Zeichenkette \b`
`"\\\b"`	`die Zeichenkette \b`

Für die übrigen Zeichenketten gilt dies entsprechend.

argument nicht angegeben:
echo gibt nur eine Leerzeile aus.

Die folgende Option steht allgemein für Benutzer des Kommandointerpreters *csh* (C-Shell) zur Verfügung. Arbeiten Sie jedoch mit *sh*, dann muß *PATH* vor dem Eintrag */usr/bin* den Eintrag */usr/ucb* enthalten, wenn Sie die Option verwenden wollen.

-n

Die Ausgabe von *echo* wird nicht mit einem Neue-Zeile-Zeichen abgeschlossen.

Vorsicht

Diese Option wird möglicherweise in Zukunft nicht mehr unterstützt.

BEISPIELE

1. Der folgende Bildschirm-Dialog soll zeigen, wann Leerzeichen, Tabulatorzeichen und Neue-Zeile-Zeichen erhalten bleiben:

```
$ echo Heute ist        Montag.
Heute ist Montag.
```

Hier erhält *echo* drei Argumente und gibt sie aus, jeweils getrennt durch ein Leerzeichen.

```
$ echo "Heute ist        Montag."
Heute ist        Montag.
```

Zwischen den Anführungszeichen bleiben alle eingegebenen Leerzeichen erhalten.

```
$ echo `ls p*`
pers pp ppp probe punkt pwd pwddr
$ echo "`ls p*`"
pers
pp
ppp
probe
punkt
pwd
pwddr
```

Ohne Anführungszeichen entfernt die Shell alle Argument-Trenner bis auf ein Leerzeichen aus der Ausgabe des Kommandos *ls* und übergibt sie an *echo*.
Die Anführungszeichen sorgen dafür, daß alle Leer-, Tabulator- und Neue-Zeile-Zeichen erhalten bleiben.

2. Die Benutzerin *anna* will wissen, welcher Wert der Variablen HOME zugewiesen ist:

```
$ echo $HOME

/home/anna
```

3. Die folgende Zeile in einer Shell-Prozedur bewirkt, daß auf die Standard-Fehlerausgabe eine Fehlermeldung ausgegeben wird:

```
echo Die Datei $1 ist nicht vorhanden >&2
```

4. Die folgende Zeile in einer Shell-Prozedur bewirkt, daß vor der Fehlermeldung ein akustisches Signal ertönt:

```
echo \\07Die Datei $1 ist nicht vorhanden >&2
```

5. Vor die Ausgabe des Kommandos *date* soll eine konstante Zeichenkette gestellt werden:

```
$ echo "Heutiges Datum mit Uhrzeit:  \\c"; date
Heutiges Datum mit Uhrzeit:  Wed Mar  8 17:22:05 MDT 1989
```

   Das Steuerzeichen \c bewirkt, daß *echo* kein Neue-Zeile-Zeichen ausgibt. Trotz der Anführungszeichen muß der Gegenschrägstrich entwertet werden.

SIEHE AUCH

csh, ksh, sh

ed
Zeilenorientierter Editor im Dialogbetrieb

ed ist ein interaktiver zeilenorientierter Editor. Mit Hilfe von *ed-Skripts* (siehe *Arbeiten mit ed-Skripts*) können Sie bequem mehrere Dateien mit derselben Kommandofolge bearbeiten. *ed* verarbeitet die Ausgabe des Kommandos *diff -e* (siehe *diff*).

ed[_-s][_-p_zeichenkette][_-x][_-C][_datei]

-s

Die Option -s unterdrückt die folgenden standardmäßigen Ausgaben:

- Anzahl der verarbeiteten Zeichen bei den *ed*-Kommandos
  e (edit - editieren)
  r (read - lesen)
  w (write - schreiben)
- Fragezeichen ?, das vor versehentlichem Löschen des Pufferinhalts warnt, bei den *ed*-Kommandos
  e (edit - editieren)
  q (quit - verlassen)
- Ausrufezeichen ! als Bereitzeichen von *ed* nach einem !-Kommando.

Die Option *-s* entspricht der alten Option -, die weiterhin unterstützt wird.

-p_zeichenkette

Mit *zeichenkette* können Sie die Bereitzeichenkette definieren, die *ed* im Kommandomodus ausgibt. Für *zeichenkette* können Sie ein oder mehrere Zeichen angeben.

-p_zeichenkette nicht angegeben:
ed gibt keine Bereitzeichenkette aus.

-x

Verschlüsselungsoption: *ed* führt beim Aufruf das *ed*-Kommando *X* aus und fragt Sie nach einem Schlüssel. Dieser Schlüssel wird dann zum Textent- und Textverschlüsseln gemäß dem Algorithmus des Kommandos *crypt* verwendet. *ed* klärt ab, ob der einzulesende Text verschlüsselt ist oder nicht. Die temporäre Pufferdatei wird ebenfalls mit einer Transformation des eingegebenen Schlüssels verschlüsselt (siehe *crypt*).

-C

Verschlüsselungsoption: wie *-x*, nur daß *ed* das *ed*-Kommando *C* ausführt. Das Kommando *C* entspricht dem Kommando *X* mit dem Unterschied: es wird angenommen, daß jeder zu lesende Text verschlüsselt ist (siehe *.crypt*).

datei

Name der Datei, die Sie bearbeiten möchten. *ed* kopiert die Datei in den internen Puffer und speichert *datei* als aktuellen Dateinamen.

datei nicht angegeben:
Sie beginnen in einem leeren Puffer zu arbeiten und bestimmen erst beim Schreiben des Pufferinhalts mit dem Kommando *w␣datei* (write - schreiben) in eine Datei deren Namen.

ED-PUFFER

Beim Aufruf von *ed* wird ein Puffer eröffnet.

Wenn Sie keine Datei angegeben haben, ist der Puffer leer. Sie füllen ihn während Ihrer Editorsitzung mit Text.

Wenn Sie eine Datei angegeben haben, wird eine Kopie dieser Datei in den Puffer eingelesen. Den Text im Puffer bearbeiten Sie während Ihrer Editorsitzung.

Bevor Sie den Editor wieder verlassen, müssen Sie entscheiden, ob Sie den neu erstellten oder geänderten Pufferinhalt sichern, d.h. in eine Datei schreiben möchten.

Möchten Sie den Pufferinhalt sichern, schreiben Sie den Pufferinhalt mit dem Kommando *w*[*␣datei*] (write - schreiben) in die angegebene Datei (Standard: die beim Aufruf von *ed* angegebene) zurück und verlassen dann den Editor mit einem der Kommandos *q* (quit - verlassen), *Q* (Quit - verlassen) oder mit der Taste END.

Möchten Sie den Pufferinhalt nicht sichern, können Sie mit *Q* oder zweimal *q* den Editor verlassen, ohne vorher den Pufferinhalt mit *w* zurückzuschreiben. Nach der ersten Eingabe von *q* gibt *ed* als Warnung vor versehentlichem Löschen des Pufferinhalts ein ? aus. Gelöscht wird der Pufferinhalt erst, wenn Sie dann ein zweites Mal *q* eingeben. Statt des zweiten *q* können Sie auch *Q* oder END eingeben.

ARBEITSMODI

Der *ed* bietet Ihnen zwei Arbeitsmodi: den Kommandomodus und den Eingabemodus. Nach dem Aufruf mit *ed[_datei]* ↵ befindet sich *ed* im Kommandomodus. Im Kommandomodus geben Sie i.a. ein Kommando in einer Zeile an und schließen es mit der Taste ↵ ab.

Den Eingabemodus schalten Sie ein mit einem der Kommandos
a (append - anfügen)
i (insert - einfügen)
c (change - verändern)
(siehe unten, *ed-Kommandos*).

Im Eingabemodus werden alle folgenden Eingabezeichen, auch nicht-druckbare Zeichen (wie z.B. die Tastencodes der Schreibmarkentasten), in die Arbeitskopie im Puffer geschrieben. Im Eingabemodus erkennt *ed* keine Kommandos. Sie können für den Kommandomodus ein Bereitzeichen definieren (siehe Option *-p* und *ed*-Kommando *P*) und erkennen dann sofort, in welchem Modus Sie gerade arbeiten. Verlassen können Sie den Eingabemodus mit einem Punkt in der ersten Spalte . ↵ oder mit der Taste DEL. Die Taste DEL bewirkt generell, daß *ed* alle Eingaben seit dem letzten ↵ ignoriert und ein ? als Warnung ausgibt.

KOMMANDOSTRUKTUR

Für *ed* existiert zu jedem Zeitpunkt eine *aktuelle Zeile*. Die aktuelle Zeile ist in der Regel die zuletzt durch ein Kommando bearbeitete Zeile. Auf diese aktuelle Zeile beziehen sich die Kommandos immer dann, wenn Sie vor den Kommandos keine anderen Adressen angeben.

Die meisten *ed*-Kommandos haben folgende Struktur:

```
[bereich]ed-kommando[parameter...] ↵
```

bereich
Mit *bereich* wählen Sie die Zeilen im Puffer aus, auf die das *ed*-Kommando angewendet werden soll. Für *bereich* können Sie eine oder zwei Adressen angeben:

bereich = adresse
Die durch *adresse* bezeichnete Zeile gilt als ausgewählt.

bereich = adresse1,adresse2
adresse1,adresse2 kennzeichnen den Bereich zwischen den angegebenen Intervallgrenzen einschließlich. Die Suche nach beiden Adressen beginnt in der aktuellen Zeile. Verändert wird die aktuelle Zeile erst bei der Ausführung von Kommandos.
adresse2 muß sich auf eine Zeile im Puffer beziehen, die hinter der mit *adresse1* bezeichneten Zeile liegt, sonst meldet *ed* einen Fehler.

bereich = adresse1;adresse2
adresse1;adresse2 kennzeichnen den Bereich zwischen den angegebenen Intervallgrenzen einschließlich. Die Suche nach *adresse1* beginnt in der aktuellen Zeile. Die mit *adresse1* bezeichnete Zeile wird dann zur neuen aktuellen Zeile, dann erst wird *adresse2* ermittelt. Sie können so die Zeile festlegen, bei der ein Suchvorgang in Vor- und Rückwärtsrichtung beginnen soll (siehe *Adressen*, Punkte *5* und *6*).
adresse2 muß sich auf eine Zeile im Puffer beziehen, die hinter der mit *adresse1* bezeichneten Zeile liegt, sonst meldet *ed* einen Fehler.

bereich nicht angegeben:
ed nimmt die zum jeweiligen Kommando gehörige Standard-Adresse an; diese ist bei jedem *ed*-Kommando beschrieben.

Benötigt *ed* keine Adresse und haben Sie trotzdem eine angegeben, meldet *ed* einen Fehler.

Haben Sie mehr Adressen angegeben als das Kommando erlaubt, nimmt *ed* die letzten.

ADRESSEN

Adressen konstruieren Sie folgendermaßen :

Adresse		Bedeutung
1.	.	aktuelle Zeile
2.	$	letzte Zeile
3.	n	n-te Zeile
4.	'x	die mit dem Buchstaben *x* markierte Zeile. *x* muß ein Kleinbuchstabe sein (siehe *k*).

5. /rA/

   Ein einfacher regulärer Ausdruck *rA* in /.../ eingeschlossen (siehe *Tabellen und Verzeichnisse, Reguläre Ausdrücke*), adressiert die erste Zeile, die eine zu dem regulären Ausdruck passende Zeichenkette enthält.
   Es wird dabei von der aktuellen Zeile aus vorwärts gesucht. Findet *ed* keine passende Zeile, setzt *ed* die Suche am Dateianfang fort und sucht wieder bis zur aktuellen Zeile. Enthält der reguläre Ausdruck *rA* selbst eines der Begrenzungszeichen / oder ?, muß es mit \ entwertet werden.

   Die Zeichen \n in einem regulären Ausdruck erzielen bei Neue-Zeile-Zeichen im durchsuchten Text *keinen* Treffer!
   Ein regulärer Ausdruck kann in *bereich* nur einmal vorkommen, z.B. adressiert /rA1/,/rA2/ nur die erste zu /rA2/ passende Zeile.

6. //

   Ein leerer regulärer Ausdruck // adressiert die Zeile, die zu dem zuletzt angegebenen regulären Ausdruck paßt.

7. ?rA?

   Wie /rA/, aber es wird von der aktuellen Zeile aus rückwärts in Richtung Dateianfang gesucht. Enthält der reguläre Ausdruck *rA* selbst eines der Begrenzungszeichen / oder ?, muß es mit \ entwertet werden.

8. adr[+]n

   n-te Zeilen nach der durch *adr* bezeichneten Zeile.

   adr[-]n

   n-te Zeilen vor der durch *adr* bezeichneten Zeile.

9. +n

   n Zeilen nach der aktuellen Zeile.

   -n

   n Zeilen vor der aktuellen Zeile.

10. [adr]+...
    [adr]-...

    Eine Zeile nach (+) bzw. vor (-) der durch *adr* bezeichneten Zeile. Jedes Vorkommen von + erhöht die Adreßangabe um 1, - erniedrigt die Adreßangabe um 1. Die Adreßangabe + + adressiert also die zweite Zeile nach der aktuellen Zeile.

11. ,

    Ein Komma , ist gleichbedeutend mit dem Adressenpaar 1,$, wenn ein Kommando folgt, sonst wird die letzte Zeile ausgegeben.

    ;

    Ein Strichpunkt ; ist gleichbedeutend mit dem Adressenpaar .,$, wenn ein Kommando folgt, sonst wird die letzte Zeile ausgegeben.

KOMMANDOS

Die folgende Liste enthält eine systematische Übersicht aller *ed*-Kommandos, die Sie im Kommandomodus eingeben können. Die daran anschließende ausführliche Kommandobeschreibung ist alphabetisch geordnet.

Übersicht der ed-Kommandos

```
Eingabemodus einschalten
   a        append     anfügen
   c        change     löschen und ersetzen
   i        insert     einfügen
Bereitzeichen im Kommandomodus ausgeben
   P        prompt     Bereitzeichen * ausgeben
Kommandos rückgängig machen
   u        undo       rückgängig machen
Kommandos abbrechen
(DEL)       ---        abbrechen
Fehler erläutern
   h        help       letzte Fehlermeldung erläutern
   H        Help       Hilfemodus ein-/ausschalten. Bei jeder
                       Fehleranzeige in Form eines Fragezeichens ?
                       wird bei eingeschaltetem Hilfemodus eine
                       Fehlermeldung ausgegeben
                       (siehe FEHLERMELDUNGEN).
Text ändern
   a        append     anfügen
   c        change     löschen und ersetzen
   d        delete     löschen
   i        insert     einfügen
Zeilen ausgeben
   p        print      ausgeben
   l        list       ausgeben mit nicht-druckbaren Zeichen
                       als Oktalzahlen
   adresse  ---        adressierte Zeile ausgeben
   adresse
     +zahl  ---        adressierte Zeile ausgeben
   n        number     Zeilen numerieren und ausgeben
(↵)         ---        Zeile hinter aktueller Zeile ausgeben
Zeilennummern ausgeben
   adresse= ---        Nummer der adressierten Zeile ausgeben
   n        number     Zeilen numerieren und ausgeben
Zeilenbereiche verschieben
   t        transfer   kopieren von Zeilen
   m        move       verschieben von Zeilen

Textmuster suchen und ersetzen
   s        substitute suchen und ersetzen
Zeilen verbinden
   j        join       aufeinanderfolgende Zeilen verbinden
Zeilen markieren
   k        mark       Zeilen markieren
```

Ausgewählte Zeilen mit Kommandos bearbeiten		
g	global	Kommandoliste global für alle Zeilen ausführen, die zu /*rA*/ passen
G	Global	interaktiv Kommandoliste global für alle Zeilen ausführen, die zu /*rA*/ passen
v	vice-versa	wie *g*, aber für alle Zeilen, die nicht zu /*rA*/ passen
V	Vice-Versa	wie *G*, aber für alle Zeilen, die nicht zu /*rA*/ passen
Aktuellen Dateinamen ändern		
f	file-name	aktuellen Dateinamen ändern/ausgeben
Shell-Kommandos ausführen		
!	---	Shell-Kommando aufrufen
Dateien in Puffer einlesen		
e	edit	Pufferinhalt löschen und neu einlesen
E	Edit	Pufferinhalt ohne Warnung löschen und neu einlesen
r	read	Datei in Puffer einlesen
Dateiinhalt ver- und entschlüsseln		
C	crypt	Schlüsseleingabe für Leseoperationen
X	crypt	Schlüsseleingabe für Lesen und Sichern
Pufferinhalt sichern		
w	write	Pufferinhalt in Datei schreiben
W	write	Pufferinhalt an Datei anfügen
Editor verlassen		
q	quit	*ed* verlassen
Q	Quit	*ed* verlassen ohne Warnung
END	---	*ed* verlassen

BESCHREIBUNG DER ED-KOMMANDOS

Die eckigen Klammern [] sind nicht einzugeben! Sie zeigen an, daß die dazwischen eingeschlossene Adreßangabe fakultativ ist.
In der Regel darf in einer Zeile nur ein Kommando stehen. Jedoch können Sie an die Kommandos (mit Ausnahme von *e, f, r* und *w*) als Suffix *l, n* oder *p* anhängen, wenn die im folgenden unter den Kommandos *l, n* und *p* beschriebenen Funktionen ausgeführt werden sollen.

[adresse]**a**
text
.

(a - append) liest den eingegebenen *text* und fügt ihn hinter der mit *adresse* adressierten Zeile an. Die aktuelle Zeile ist nun entweder die letzte Zeile des eingefügten Textes oder, falls Sie keinen Text eingegeben haben, die adressierte Zeile. Die Adresse 0 ist bei diesem Kommando erlaubt; der Text wird dann vor die erste Zeile des Puffers eingefügt. Über die Datensichtstation können Sie maximal 256 Zeichen je Zeile einschließlich Neue-Zeile-Zeichen eingeben.

adresse nicht angegeben:
adresse = .

[bereich]**c**
text
.

(c - change) löscht den angegebenen *bereich* und ersetzt ihn durch den eingegebenen *text*. Die aktuelle Zeile ist nun entweder die letzte Zeile des eingegebenen Textes oder, falls Sie keinen Text eingegeben haben, die Zeile vor den gelöschten Zeilen.

bereich nicht angegeben:
bereich = .,.

C

Verschlüsselungskommando:
C entspricht dem Kommando *X* mit dem Unterschied: es wird angenommen, daß jeder mit den Kommandos *e* und *r* zu lesende Text verschlüsselt ist
(siehe Option *-C*).

[bereich]**d**

(d - delete) löscht den angegebenen *bereich*. Die Zeile hinter der letzten gelöschten Zeile wird zur aktuellen Zeile. Standen die gelöschten Zeilen am Ende des Puffers, wird die neue letzte Zeile die aktuelle Zeile.

bereich nicht angegeben:
bereich = .,.

e[␣datei]
(e - edit) löscht den gesamten Puffer und liest eine Kopie des Inhalts von *datei* ein. Ist der Pufferinhalt seit dem letzten Sichern oder Überschreiben verändert und nicht mit *w* gesichert worden, löscht *ed* den Pufferinhalt nicht sofort, sondern warnt Sie mit einem ? vor versehentlichem Löschen. Geben Sie daraufhin ein weiteres Mal *e* ein, wird der Pufferinhalt ohne weitere Warnung überschrieben. Die Anzahl der eingelesenen Bytes wird ausgegeben, wenn Sie *ed* nicht mit der Option -s aufgerufen haben. Die aktuelle Zeile ist die letzte Zeile des Puffers. Der Name *datei* wird gespeichert, so daß er später bei den Kommandos *e, r* und *w* als aktueller Dateiname verwendet werden kann. Wenn Sie für *datei* ein Ausrufezeichen ! angeben, wird der Rest der Zeile als Shell-Kommando interpretiert. Das Ergebnis dieses Shell-Kommandos wird in den Puffer eingelesen. Ein mit ! eingeleitetes Shell-Kommando wird nicht als Dateiname gespeichert.

datei nicht angegeben:
datei = aktueller Dateiname

E[␣datei]
(E - edit) verhält sich wie *edit*, außer, daß es den Puffer ohne warnendes ? überschreibt, auch wenn der Inhalt des Puffers verändert und nicht gerettet wurde.

datei nicht angegeben:
datei = aktueller Dateiname

f[␣datei]
(f - file) setzt den aktuellen Dateinamen auf *datei*. Den aktuellen Dateinamen verwenden die Kommandos *e, E, r* und *w*.

datei nicht angegeben:
ed gibt den aktuellen Dateinamen aus.

[bereich]␣**g**/rA/kommandoliste
(g - global) markiert im ersten Schritt alle Zeilen, die eine Zeichenkette enthalten, die zu *rA* paßt. *rA* ist ein einfacher regulärer Ausdruck (siehe *Tabellen und Verzeichnisse, Reguläre Ausdrücke*). Dann wird die nächste markierte Zeile zur aktuellen Zeile und *kommandoliste* darauf angewandt.

Ein einzelnes Kommando oder das erste einer *kommandoliste* müssen Sie in dieselbe Zeile schreiben wie das Kommmando *g*. Alle Zeilen einer *kommandoliste* außer der letzten müssen Sie mit \ ↵, die letzte mit ↵ abschließen.

Die Kommandos *a, i* und *c* mit der dazugehörigen Eingabe *text* sind zugelassen. Auch Zeilen in *text* müssen Sie mit \ ↵ abschließen. Den die Eingabe normalerweise abschließenden Punkt . können Sie in der letzten Zeile der *kommandoliste* weglassen.

Eine leere *kommandoliste* entspricht dem Kommando *p*.

Die Kommandos *g,G,v* und *V* sind in der *kommandoliste* nicht zugelassen.

Nicht mit dem Kommando ! zu kombinieren!

bereich nicht angegeben:
bereich = 1,$

[bereich]**G/rA/**

(G - global) *G* ist die interaktive Variante des Kommandos *g*. Zuerst wird jede Zeile markiert, die zu *rA* paßt. *rA* ist ein einfacher regulärer Ausdruck (siehe *Tabellen und Verzeichnisse, Reguläre Ausdrücke*). Die erste der markierten Zeilen wird ausgegeben. Sie wird gleichzeitig zur aktuellen Zeile.

Sie haben nun die Möglichkeit, ein Kommando anzugeben, das ausgeführt werden soll. Hierbei darf es sich nicht um *a, c, i, g, G, v* oder *V* handeln. Nach der Ausführung dieses Kommandos wird die nächste markierte Zeile ausgegeben, usw.

Ein Neue-Zeile-Zeichen hat dieselbe Wirkung wie ein leeres Kommando. Wenn Sie das kommerzielle Und & eingeben, wird das Kommando wiederholt, das seit dem letzten Aufrufen von *G* als letztes ausgeführt wurde.

Mit Kommandos, die unter *G* aufgerufen werden, können beliebige Zeilen im Puffer adressiert und bearbeitet werden. *G* können Sie mit der Taste DEL abbrechen.

bereich nicht angegeben:
bereich = 1,$

h

(h - help) gibt eine kurze Fehlermeldung aus, die den Grund für das letzte Fragezeichen ? auf dem Bildschirm erklärt. Mögliche Fehlermeldungen siehe *FEHLERMELDUNGEN*.

H

(H - help) bewirkt, daß *ed* in einen Modus übergeht, in dem bei auftretenden Fehlern anstelle von ? erläuternde Fehlermeldungen ausgegeben werden. Auch das zuletzt ausgegebene ? wird, falls vorhanden, erläutert. Diesen Modus können Sie durch ein zweites Aufrufen von *H* ausschalten.
Standardmäßig ist der Modus *H* ausgeschaltet. Mögliche Fehlermeldungen siehe *FEHLERMELDUNGEN*.

[adresse]**i**
text
.

(i - insert) fügt *text* vor der mit *adresse* adressierten Zeile ein. Die aktuelle Zeile ist nun entweder die letzte Zeile des eingefügten Textes oder, falls Sie keinen Text eingegeben haben, die adressierte Zeile. Dieses Kommando unterscheidet sich vom Kommando *append* nur durch die Positionierung des eingegebenen Textes. Adresse 0 ist nicht zulässig. Über eine Datensichtstation dürfen Sie maximal 256 Zeichen einschließlich des Neue-Zeile-Zeichens je Zeile eingeben.

adresse nicht angegeben:
adresse = .

[bereich]**j**

(j - join) verbindet alle in *bereich* liegenden Zeilen zu einer einzigen Zeile und löscht dabei die entsprechenden Neue-Zeile-Zeichen. Wenn Sie als Adresse nur eine Zeile angeben, geschieht nichts. Die neue Zeile wird zur aktuellen Zeile.

bereich nicht angegeben:
bereich = .,.

[adresse]**k***x*

(k - mark) markiert die mit *adresse* adressierte Zeile mit dem für *x* angegebenen Buchstaben, wobei *x* ein Kleinbuchstabe sein muß. Die Markierung wird aber nicht ausgegeben. Adressieren können Sie die markierte Zeile mit Hochkomma x ('x). Die aktuelle Zeile bleibt unverändert.

adresse nicht angegeben:
adresse = .

[bereich]**l**

(l - list) gibt im Gegensatz zu *p* den angegebenen *bereich* wie folgt aus: Einige nicht-druckbare Zeichen werden durch Mnemoniks, z.B Tabulatorzeichen durch das Zeichen >, die übrigen nicht-druckbaren Zeichen werden als Oktalzahlen ausgegeben; überlange Zeilen werden mehrzeilig ausgegeben, am Ende mit dem Zeilenfortsetzungszeichen \ versehen. Das Kommando *l* können Sie an jedes Kommando anhängen, mit Ausnahme von *e, f, r* und *w*. Beachten Sie, daß durch die Darstellung des Neue-Zeile-Zeichens als \012 die optische Zeilenstruktur zerstört wird.

bereich nicht angegeben:
bereich = .,.

[bereich]**m**adresse

(m - move) verschiebt den *bereich* hinter die mit *adresse* adressierte Zeile. Die letzte der verschobenen Zeilen wird die aktuelle Zeile. Wenn Sie für *adresse* den Wert 0 angeben, wird *bereich* ganz an den Anfang der Datei gesetzt. Liegt *adresse* innerhalb von *bereich*, gibt *ed* eine Fehlermeldung aus.

bereich nicht angegeben:
bereich = .,.

[bereich]**n**

(n - number) gibt die mit *bereich* adressierten Zeilen aus, wobei an den Anfang jeder Zeile die Zeilennummer und ein Tabulatorzeichen gesetzt werden. Die zuletzt ausgegebene Zeile wird zur neuen Zeile. Das Kommando *n* können Sie an jedes Kommando anhängen, mit Ausnahme von *e, f, r* und *w*.

bereich nicht angegeben:
bereich = .,.

[bereich]**p**

(p - print) gibt die mit *bereich* adressierten Zeilen aus. Nicht-druckbare Zeichen werden unverändert ausgegeben. Überlange Zeilen laufen ohne besondere Kennzeichnung auf der nächsten Zeile weiter, d.h. man kann sie nicht als solche erkennen. Die zuletzt ausgegebene Zeile wird zur aktuellen Zeile. Das Kommando *p* können Sie an jedes Kommando anhängen, mit Ausnahme von *e, f, r* und *w*. Mit *dp* wird also die aktuelle Zeile gelöscht und die neue aktuelle Zeile ausgegeben.

bereich nicht angegeben:
bereich = .,.

P

(P - prompt) bewirkt, daß *ed* im Kommandomodus als Bereitzeichen den Stern * oder die Bereitzeichenkette (siehe Option *-p*) ausgibt. Diesen Modus können Sie durch einen zweiten Aufruf von *P* wieder ausschalten. Standardmäßig ist dieser Modus ausgeschaltet.

q

(q - quit) beendet den *ed*. Ist der Pufferinhalt seit dem letzten Sichern oder Überschreiben verändert und nicht mit *w* gesichert worden, gibt *ed* als Warnung vor versehentlichem Löschen ein ? aus und wartet auf weitere Eingabe. Wenn Sie dann END, *Q* oder zum zweitenmal *q* eingeben, beenden Sie den *ed* ohne weitere Warnung und ohne den Pufferinhalt gesichert zu haben.

Vorsicht

Wenn Sie nach dem ersten *q* noch Aktionen durchführen, die den Pufferinhalt nicht verändern, beenden Sie mit dem zweiten *q* auch den *ed* ohne weitere Warnung und ohne Sicherung des Pufferinhalts.

Q

(Q - quit) beendet den *ed* sofort ohne Warnung, auch wenn Sie den Pufferinhalt seit dem letzten Sichern oder Überschreiben verändert und nicht mit *w* gesichert haben.

[adresse]**r**[␣datei]

(r - read) liest *datei* und fügt ihren Inhalt hinter der mit *adresse* adressierten Zeile ein.
Die Adresse 0 ist für dieses Kommando erlaubt.
Sie bewirkt, daß *datei* an den Anfang des Puffers geschrieben wird. Nach erfolgreichem Lesen wird die Anzahl der gelesenen Bytes ausgegeben, wenn Sie *ed* nicht mit der Option *-s* aufgerufen haben. Die aktuelle Zeile ist die letzte eingelesene Zeile. Der aktuelle Dateiname wird nicht auf *datei* gesetzt, es sei denn, Sie haben *ed* ohne Dateinamen aufgerufen und *datei* ist der erste seit dem Aufruf angesprochene Dateiname. Wenn Sie für *datei* ein Ausrufezeichen ! angeben, dann wird der Rest der Zeile als Shell-Kommando interpretiert, ausgeführt und die Ausgabe davon eingelesen. Ein solches Kommando wird nicht als aktueller Dateiname gespeichert.

adresse nicht angegeben:
adresse = $

datei nicht angegeben:
datei = aktueller Dateiname

[bereich]**s**/rA/Ersetzungszeichenkette/[g][n]

(s - substitute) durchsucht jede Zeile in *bereich* nach Zeichenketten, die zu *rA* passen. *rA* ist ein einfacher internationalisierter regulärer Ausdruck (siehe *Tabellen und Verzeichnisse, Reguläre Ausdrücke*). In *jeder* so gefundenen Zeile wird eine zu *rA* passende Zeichenkette durch *Ersetzungszeichenkette* ersetzt: ohne *g* und *n* wird das erste, mit *g* jedes und mit *n* das n-te Auftreten einer passenden Zeichenkette in der Zeile ersetzt. *n* ist eine positive ganze Zahl von 1 bis 512. Falls keine passende Zeichenkette gefunden wurde, meldet *ed* ein ? als Fehler. Als Trennzeichen zwischen dem Kommando *s*, dem regulären Ausdruck *rA* und *Ersetzungszeichenkette* können Sie nicht nur den Schrägstrich / verwenden, sondern auch jedes andere beliebige Zeichen außer: Leerzeichen und Neue-Zeile-Zeichen. Das Zeichen wird dadurch als Trennzeichen definiert, daß es unmittelbar auf *s* folgt. Zur aktuellen Zeile wird die Zeile, auf der die letzte Ersetzung stattgefunden hat.

Sonderzeichen in der Ersetzungszeichenkette

Sonderzeichen in der Zeichenkette	Bedeutung
&	wird durch die Zeichenkette ersetzt, die in der aktuellen Zeile zu dem regulären Ausdruck *rA* paßt.
\m	wird durch die Zeichenkette ersetzt, die zum *m*-ten in \(...\) eingeschlossenen regulären Unterausdruck von *rA* paßt, wobei *m* eine einstellige Dezimalzahl ist. Bei durch Klammern ineinander geschachtelten Unterausdrücken wird *m* durch Zählen des Auftretens von \ von links nach rechts bestimmt.
%	wird durch die Zeichenkette ersetzt, mit der beim zuletzt abgelaufenen *s*-Kommando ersetzt wurde, wenn *Ersetzungszeichenkette* nur aus dem Prozent-Zeichen % besteht.

Die Sonderbedeutung dieser Zeichen können Sie aufheben, indem Sie Ihnen jeweils einen Gegenschrägstrich \ voranstellen.

Zeile trennen in Ersetzungszeichenkette

Wenn Sie in *Ersetzungszeichenkette* ein Neue-Zeile-Zeichen haben möchten, müssen Sie es mit \ davor entwerten. Sie geben also \ ↵ ein. Ein solches Ersetzungskommando darf nicht Teil von *kommandoliste* der Kommandos *g* und *v* sein.

bereich nicht angegeben:
bereich = .,.

[bereich]ta

kopiert *bereich* hinter die angegebene Zeile *a*. Als Adresse für *a* ist 0 zugelassen. Zur aktuellen Zeile wird die letzte der kopierten Zeilen.

bereich nicht angegeben:
bereich = .,.

u

(u - undo) macht das letzte Kommando rückgängig, mit dem der Inhalt des Puffers geändert wurde. Rückgängig machen können Sie die Kommandos *a, c, d, g, i, j, m, r, s, t, v, G* und *V*.

[bereich]**v/rA/kommandoliste**
(v - vice-versa) bearbeitet alle Zeilen mit *kommandoliste*, die *keine* Zeichenkette enthalten, die zu *rA* paßt. *rA* ist ein einfacher regulärer Ausdruck (siehe *Tabellen und Verzeichnisse, Reguläre Ausdrücke*). Bis auf diese Zeilenauswahl funktioniert *v* wie das Kommando *g*.

Nicht mit dem Kommando ! zu kombinieren!

bereich nicht angegeben:
bereich = 1,$

[bereich]**V/rA/**
(V - vice-versa) ist die interaktive Variante des Kommandos *v*. Sie können mit *V* alle Zeilen bearbeiten, die *keine* Zeichenkette enthalten, die zu *rA* paßt. *rA* ist ein einfacher internationalisierter regulärer Ausdruck (siehe *Tabellen und Verzeichnisse, Reguläre Ausdrücke*). Bis auf diese Zeilenauswahl funktioniert *V* wie das Kommando *G*.

bereich nicht angegeben:
bereich = 1,$

[bereich]**w**[␣datei]
(w - write) schreibt *bereich* in die *datei*. Der aktuelle Dateiname ändert sich nicht, falls er bereits gesetzt ist. Ist er nicht gesetzt, wird *datei* zum neuen aktuellen Dateinamen. Der alte Inhalt von *datei* wird dabei überschrieben. Existiert *datei* noch nicht, so wird sie angelegt. Die aktuelle Zeile bleibt unverändert. Nach erfolgreichem Schreiben wird die Anzahl der geschriebenen Bytes ausgegeben, vorausgesetzt, Sie haben den *ed* nicht mit der Option *-s* aufgerufen.

Wenn Sie für *datei* statt eines Dateinamens das Ausrufezeichen ! angeben, wird der Rest der Zeile als Shell-Kommando interpretiert und ausgeführt. *bereich* ist dann die Standard-Eingabe für das Shell-Kommando. Ein solches Kommando wird nicht als aktueller Dateiname gespeichert.

bereich nicht angegeben:
bereich = 1,$

datei = aktueller Dateiname

[bereich]**W**[␣datei]
(W - write) fügt *bereich* an die Datei *datei* an, sonst verhält es sich wie das Kommando *w*, das *datei* aber überschreibt. Existiert *datei* noch nicht, wird sie angelegt.

x

Verschlüsselungsoption: *ed* fragt Sie nach einem Schlüssel, der bei folgenden *e*-, *r*- und *w*-Kommandos zum Ent- und Verschlüsseln von Text durch den Algorithmus des Kommandos *crypt* verwendet wird. Ein leerer Schlüssel schaltet diesen Mechanismus wieder ab. *ed* klärt ab, ob der einzulesende Text (*e* und *r*) verschlüsselt ist oder nicht. Die temporäre Pufferdatei wird ebenfalls mit einer Transformation des eingegebenen Schlüssels verschlüsselt (siehe Option *-x*).

adresse

Die mit *adresse* adressierte Zeile wird ausgegeben.

[adresse]+zahl

Dieses Kommando gibt die Zeile aus, die *zahl* Zeilen hinter der mit *adresse* adressierten Zeile liegt.

adresse nicht angegeben:
adresse = .

[adresse]=

Die Nummer von *adresse* wird ausgegeben. Die aktuelle Zeile wird dadurch nicht verändert.

adresse nicht angegeben:
adresse = $

!kommando

Der Rest der Zeile hinter dem ! wird als Shell-Kommando interpretiert und ausgeführt. Ist in *kommando* ein nicht entwertetes Prozent-Zeichen % enthalten, wird es durch den gespeicherten Dateinamen ersetzt. !! wiederholt das letzte *kommando*. In beiden Fällen wird die expandierte Kommandozeile ausgegeben. Nach Beendigung des *kommandos* ist *ed* wieder aktiv. Die aktuelle Zeile wird nicht verändert. Nicht zu kombinieren mit den Kommandos *g* und *v*!

↵

Geben Sie im Kommandomodus die Taste ↵ alleine ein, wird die Zeile hinter der aktuellen Zeile ausgegeben. Dies ist mit der Eingabe .+1p gleichbedeutend. Sie können so im Puffer von einer Zeile zur nächsten springen.

Die Taste ↵ müssen Sie drücken, um im Kommandomodus die Eingabe eines Kommandos oder um im Eingabemodus die Eingabe einer Textzeile abzuschließen.

DEL

Mit der Taste DEL können Sie laufende *ed*-Kommados unterbrechen oder die Eingabe einer Zeile abbrechen. *ed* meldet sich anschließend mit einem ? zurück.

END

Die Wirkung der Taste END ist dieselbe wie beim Kommando *q*.

Steht das Zeichen, das einen regulären Ausdruck oder eine Ersetzungszeichenkette abschließt (z.B. ein /), unmittelbar vor einem Neue-Zeile-Zeichen, dann können Sie dieses Zeichen weglassen. Die adressierte Zeile wird dann ausgegeben. Die folgenden Beispielpaare haben die gleiche Funktion:

```
s/s1/s2       s/s1/s2/p
g/s1          g/s1/p
?s1           ?s1?
```

ARBEITEN MIT ED-SCRIPTS

ed liest Kommandos und einzufügenden Text von der Standard-Eingabe. Deshalb können Sie die Eingabe auch umlenken, so daß *ed* aus einer Datei liest. Mit

```
$ ed -s datei < ed_scriptdatei > ausgabe
```

wird die Datei *datei* editiert und mit den *ed*-Kommandos bearbeitet, die in der Datei *ed_scriptdatei* stehen. Option -s unterdrückt die standardmäßige Ausgabe der Meldungstexte auf den Bildschrim.

Das Arbeiten mit *ed-scripts* hat den Vorteil, daß Sie bestimmte Kommandofolgen jederzeit reproduzieren und beliebig oft verwenden können. Außerdem können Sie so diesen Vorgang auch als Hintergrundprozeß ablaufen lassen und selbst ungestört am Bildschirm weiterarbeiten:

```
$ ed datei < ed_scriptdatei&
```

Arbeiten Sie mit einem fehlerhaften *ed*-Skript, dann beendet *ed* beim ersten Fehler.

ENDE-STATUS

0 bei Erfolg
>0 bei fehlerhaftem Aufruf des *ed* oder bei Abbruch einer Prozedur aufgrund fehlerhafter *ed*-Kommandos.

FEHLERMELDUNGEN

Wenn Sie beim Aufruf von *ed* zwischen Option *p* und *zeichenkette* kein Leerzeichen eingeben oder *zeichenkette* vergessen:

```
ed: -p arg missing
```

ed: -p argument fehlt

Wenn Ihnen bei *ed*-Kommandos Fehler unterlaufen:

```
?
```

Syntaxfehler im Kommando

```
?datei
```

Datei *datei* ist nicht vorhanden oder kann nicht gelesen werden.

Nähere Informationen bekommen Sie mit den Kommandos *h* und *H*. Die häufigsten Fehlermeldungen sind:

```
line out of range
```

Zeile außerhalb des gültigen Zeilenbereichs

```
warning: expecting 'w'
```

Warnung: 'w' wird erwartet

```
no space after command
```

kein Leerzeichen hinter dem Kommando

```
unknown command
```

unbekanntes Kommando

```
bad range
```

ungültige Bereichsangabe

```
cannot open input file
```

Eingabedatei kann nicht geöffnet werden

`illegal or missing delimiter`

unzulässiges oder fehlendes Trennzeichen

`illegal suffix`

unzulässiges Suffix

`illegal or missing filename`

unzulässiger oder fehlender Dateiname

`no match`

keine passende Zeichenkette gefunden

DATEIEN

ed.hup
In dieser Datei werden die Daten gesichert, wenn *ed* das Signal SIGHUP empfängt (siehe *kill()*).

/var/tmp
Wenn dieses Dateiverzeichnis existiert, dann wird es als Dateiverzeichnis für die Speicherung der Temporärdatei verwendet.

$TMPDIR
Wenn diese Variable gesetzt und nicht leer ist, dann wird ihr Wert anstelle von */var/tmp* als Dateiverzeichnis für die Speicherung der Temporärdatei verwendet.

/tmp
Wenn die Variable *TMPDIR* nicht auf ein existierendes Dateiverzeichnis gesetzt ist und das Verzeichnis */var/tmp* nicht existiert, dann wird */tmp* zur Speicherung der Temporärdatei verwendet.

BEISPIELE

1. Beispiel für ein *ed-script*:

   In einer Datei sollen die ersten drei Zeilen durch eine Zeile mit dem Text "Adressen" und überall soll das Wort "Stachus" durch "Karlsplatz" ersetzt werden.

   Inhalt der Datei *edscript*:

```
1,3c
Adressen
.
1,$s/Stachus/Karlsplatz/g
w
```

   Bearbeitung einer Datei mit den Kommandos aus *edscript*:

```
$ ed datei < edscript
```

   Wenn *ed* seine Kommandos nicht von der Tastatur, sondern aus einer Datei liest, wird der Editor nach dem ersten für *ed* unverständlichen Kommando verlassen.

2. Beispiel für ein *here-script* (siehe *sh*):

   In beliebigen Dateien, die beim Aufruf der Prozedur *xy* als Argumente übergeben werden, sollen die ersten drei Zeilen durch eine Zeile mit dem Text "Adressen" und überall soll das Wort "Stachus" durch "Karlsplatz" ersetzt werden.

   Inhalt der Prozedurdatei *xy*:

```
for i in $*
do
ed $i << scrend
1,3c
Adressen
.
1,\$s/Stachus/Karlsplatz/g
w
scrend
done
```

   Bearbeitung der Dateien *text1*, *text2* und *text3* mit der Prozedur *xy*:

```
$ sh xy text1 text2 text3
```

   Die Zeichenkette << *scrend* hinter dem *ed*-Aufruf bewirkt, daß die Shell den Text bis zur Zeichenkette *scrend* an *ed* als Eingabe übergibt. Die zweite Zeichenkette *scrend* muß ohne führende Leerzeichen als einziges Wort in einer Zeile stehen. Bei der Adreßangabe 1,$ muß die Sonderbedeutung von $ mit \ aufgehoben werden, da die Shell sonst das nachfolgende *s* als Name einer Shell-Variablen interpretieren würde.

3. Im folgenden Beispiel werden einige *ed*-Kommandos vorgeführt und erläutert:

```
$ ed                    - Aufruf
p                       - Bereitzeichen * im Kommandomodus ausgeben
*a                      - Anfügen an aktuelle Zeile, hier Dateianfang
zeile1                  - Eingabetext
zeile2
zeile3
.                       - Eingabemodus beenden
*1,$p                   - Zeile 1 bis letzte Zeile ausgeben
zeile1
zeile2
zeile3
*p                      - Aktuelle (= zuletzt bearbeitete) Zeile ausgeben
zeile3
*n                      - Aktuelle Zeile mit Zeilennummer ausgeben
3       zeile3
*1,$n                   - Zeile 1 bis letzte Zeile mit Zeilennummern ausgeben
1       zeile1
2       zeile2
3       zeile3
*2c                     - Zeile 2 ersetzen durch nachfolgende Eingabe
                          mit Tabulatorzeichen
zeile2 ⇥ zeile2 ⇥ zeile2
noch eine zeile2
.                       - Eingabemodus beenden
*p                      - Aktuelle Zeile ausgeben
noch eine zeile2
*1,$n                   - Zeile 1 bis letzte Zeile mit Zeilennummern ausgeben
1       zeile1
2       zeile2  zeile2  zeile2
3       noch eine zeile2
4       zeile3
*4s/3/4                 - In Zeile 4 Zeichen 3 suchen und ersetzen durch 4
zeile4
*n                      - Aktuelle Zeile mit Zeilennummer ausgeben
4       zeile4
*1,$s/z/Z/g             - Von Zeile 1 bis letzte Zeile alle Zeichen z
                          suchen und durch Z ersetzen
*1,$p                   - Zeile 1 bis letzte Zeile ausgeben
Zeile1
Zeile2  Zeile2  Zeile2
noch eine Zeile2
Zeile4
*2l                     - Zeile 2 mit nicht druckbaren Zeichen als
                          Mnemoniks ausgeben.
Zeile2>Zeile2>Zeile2
*
*2r datei               - Hinter Zeile 2 Inhalt von datei in Puffer einlesen
57                        und Anzahl der eingelesenen Zeichen ausgeben.
                          datei muß hierfür existieren!
```

```
*1,$n                - Zeile 1 bis letzte Zeile mit Zeilennummern ausgeben
1       Zeile1
2       Zeile2  Zeile2  Zeile2  - Diese und die zwei folgenden Zeilen
3       zeile1 aus datei          kommen aus datei
4       zeile2 aus datei
5       letzte zeile aus datei
6       noch eine Zeile2
7       Zeile4

*6,7c                - Zeile 6 bis 7 ersetzen durch nachfolgende Eingabe
allerletzte Zeile
.                    - Eingabemodus beenden
*1,$n                - Zeile 1 bis letzte Zeile mit Zeilennummern ausgeben
1       Zeile1
2       Zeile2  Zeile2  Zeile2
3       zeile1 aus datei
4       zeile2 aus datei
5       letzte zeile aus datei
6       allerletzte Zeile
*q                   - Versuch, Editor mit q zu verlassen
?                    - Warnung
*h                   - Fragezeichen erklären
                     - Erklärung: erwartetes Kommando w
warning:expecting 'w'
                       zum Sichern des Pufferinhalts wurde nicht
                       eingegeben
*w bsp               - Sichern des Pufferinhalts in der Datei bsp
103                  - Ausgabe der Anzahl der gesicherten Zeichen
*q                   - Editor verlassen
$                    - Bereitzeichen der Shell
```

SIEHE AUCH

ced, *crypt*, *edit*, *ex*, *grep*, *sed*, *sh*, *stty*, *umask*, *vi*, *fspec*, *regexp* [7]

Tabellen und Verzeichnisse, Reguläre Ausdrücke

edit
Einfach zu bedienender Editor (Variante von ex)

edit ist eine einfach zu bedienende Variante des Editors *ex*. Benutzen Sie ihn, wenn Sie Anfänger sind oder wenn Sie nur gelegentlich editieren. Die folgende kurze Einführung soll Ihnen helfen, mit *edit* zu arbeiten. *edit* ist ein zeilen- und kommandoorientierter Editor. Wenn Sie bildschirmorientiert arbeiten wollen, arbeiten Sie besser mit *vi* oder *ced*.

edit[_-r][_-x][_-C]_datei

-r
(r - restore) *datei* wird nach einem Abbruch des Editors oder einem Absturz wiederhergestellt.

-x
Verschlüsselungsoption: *edit* verschlüsselt die Datei beim Schreiben und benötigt einen Schlüssel zum Lesen der Datei. *edit* klärt ab, ob der einzulesende Text verschlüsselt ist oder nicht (siehe *crypt*).

-C
Verschlüsselungsoption: wie *-x*, nur daß *edit* annimmt, daß alle Dateien verschlüsselt sind.

datei
Name der Datei, die Sie mit dem Editor bearbeiten bzw. erstellen möchten.

ARBEITEN MIT EDIT

Vorhandene Datei editieren

Wenn Sie eine bereits vorhandene Datei bearbeiten möchten, beginnen Sie mit dem Kommandoaufruf:

```
$ edit datei
```

edit erstellt dann eine Kopie der Datei, die Sie bearbeiten möchten, kopiert die Datei in einen Puffer und gibt die Anzahl der Zeilen und Zeichen in *datei* aus.

Neue Datei erstellen

Wählen Sie einen Namen für die neue Datei und rufen Sie auf:

```
$ edit datei_neu
```

Dieser Aufruf führt zu der Meldung: "datei_neu" [New file]. Es ist nun ein leerer Puffer eröffnet, in den Sie mit dem Kommando *append* (anfügen) Text schreiben können.

edit-Kommandos eingeben

Nach dem Aufruf fordert *edit* Sie mit einem Doppelpunkt : zur Eingabe von Kommandos auf.

Wenn Sie eine vorhandene Datei editieren, gibt es bereits Zeilen im Puffer, auf die Sie *edit*-Kommandos anwenden können. Mit der Eingabe

`zeile kommando` ↵

bewirken Sie, daß *kommando* auf *zeile*, mit

`zeile,zeile kommando` ↵,

daß *kommando* auf den Zeilenbereich *zeile,zeile* angewendet wird.

Wenn Sie eine neue Datei erstellen, ist der Puffer nach dem Aufruf von *edit* noch leer. Die einzig hier sinnvollen Kommandos sind *append*, *insert* und *quit*.

Aktuelle Zeile

Die meisten Kommandos werden auf eine voreingestellte *aktuelle Zeile* angewendet, es sei denn, Sie geben explizit eine(n) Zeile(nbereich) an. Wenn Sie z.B.

`:print` ↵ (alle Kommandos werden mit ↵ abgeschlossen)

eingeben, wird diese aktuelle Zeile ausgegeben. Wenn Sie

`:delete`

eingeben, wird die aktuelle Zeile gelöscht und die neue aktuelle Zeile ausgegeben. Zu Beginn einer Editorsitzung setzt *edit* die aktuelle Zeile auf die letzte Zeile in der Datei. Im allgemeinen wird die letzte durch ein Kommando angesprochene Zeile zur aktuellen Zeile.

Anfügen - append

Wenn Sie eine neue Datei erstellen oder in einer schon vorhandenen Datei neue Zeilen hinzufügen möchten, verwenden Sie das Kommando *append* oder *a*.
edit liest anschließend Text von Ihrer Tastatur solange ein, bis Sie eine Zeile eingeben, die nur aus einem Punkt in der ersten Spalte besteht:

```
:append
text
.
:
```

In einer neuen Datei wird der Text an den Anfang, in einer schon vorhandenen Datei hinter die aktuelle Zeile geschrieben. Die letzte Zeile, die Sie vor dem Punkt eingeben, wird dann die neue aktuelle Zeile.

Einfügen - insert

Das Kommando *insert* oder *i* funktioniert wie *append*, außer daß der neue Text *vor* die aktuelle Zeile geschrieben wird:

```
:insert
text
.
:
```

Zeilennummern

edit numeriert die Zeilen im *edit*-Puffer. Die erste Zeile hat die Nummer 1. Wenn Sie das Kommando

```
:1
```

eingeben, gibt *edit* diese erste Zeile auf dem Bildschirm aus. Diese erste Zeile ist nun auch die neue aktuelle Zeile, da sie zuletzt durch ein Kommando bearbeitet wurde. Wenn Sie anschließend

```
:delete
```

eingeben, löscht *edit* die aktuelle, also die erste Zeile und die bisherige Zeile 2 wird zu Zeile 1. *edit* gibt dann die neue aktuelle Zeile (die neue Zeile 1) aus, so daß Sie sich wieder orientieren können.

Ersetzen - substitute

Sie können mit dem Kommando *substitute* oder *s* Teile des Textes in der aktuellen Zeile ersetzen. Wenn Sie

```
:s/alt/neu/
```

eingeben, wird in der aktuellen Zeile die erste Zeichenkette *alt* durch die Zeichenkette *neu* ersetzt. Wenn die Zeichenketten *alt* oder *neu* selbst einen Schrägstrich enthalten, müssen Sie ihn mit einem Gegenschrägstrich \ davor entwerten.

Datei abfragen - file

Mit dem Kommando *file* oder *f* können Sie abfragen, wie viele Zeilen im *edit*-Puffer sind und ob Sie den Puffer geändert haben.

```
:file
```

Wenn Sie den Puffer geändert haben, wird ausgegeben: "[Modified]"

Puffer in die Datei zurückschreiben - write

Im Puffer neu erstellten bzw. geänderten Text können Sie mit dem Kommando *write* oder *w* sichern, d.h. in die Datei schreiben, die Sie beim Aufruf des Editors mit *edit datei* angegeben haben. Der alte Inhalt der Datei wird dabei überschrieben.

```
:w
```

Editor verlassen - quit

Nach der Eingabe des Kommandos *write* können Sie den Editor mit dem Kommando *quit* oder *q* verlassen:

```
:quit
```

Wenn Sie in der *edit*-Sitzung die Datei nicht geändert haben, brauchen Sie den *edit*-Puffer nicht mit *write* zurückschreiben (allerdings schadet es auch nichts). Wenn Sie den Pufferinhalt geändert, aber nicht mit *write* gesichert haben und den Editor verlassen wollen, gibt *edit* eine Warnung aus und wartet auf ein weiteres Kommando:

"No write since last change" "Keine Sicherung seit der letzten Veränderung"

Sie können jetzt den Pufferinhalt noch sichern, indem Sie *write* eingeben. Wenn Sie dies nicht wollen, geben Sie ein:

```
:q!
```

Der Pufferinhalt ist dann unwiderruflich gelöscht. Sie kehren zurück in die Shell.

Sie sind jetzt so weit, daß Sie alle Änderungen, die Sie machen wollen, durchführen können. Wenn Sie häufiger mit *edit* arbeiten möchten, sollten Sie allerdings noch einige weitere Möglichkeiten kennenlernen, die im folgenden beschrieben werden.

Ändern - change

Mit dem Kommando *change* oder *c* können Sie die aktuelle Zeile ersetzen durch eine Folge von Zeilen, die Sie eingeben. Wie bei *append* schließen Sie den neuen Text ab mit einer Zeile, in der nur ein Punkt in der ersten Spalte steht:

```
:change
text
.
:
```

Mehrere Zeilen können Sie ändern, indem Sie vor dem Kommando den Zeilenbereich durch Anfangs- und Endzeile angeben, z.B. Zeile 3 bis Zeile 5 einschließlich:

```
:3,5change
text
.
:
```

Die 3 alten Zeilen 3,4,5 werden ersetzt durch 1 neue Zeile mit dem Inhalt *text*.

Ausgeben - print

Sie können auf die gleiche Weise auch mehrere Zeilen mit dem Kommando *print* ausgeben. Wenn Sie z.B.

```
:1,23print
```

eingeben, werden die ersten 23 Zeilen der Datei ausgegeben.

Rückgängig machen - undo

Das Kommando *undo* macht die Wirkung der letzten Änderung wieder rückgängig. Wenn Sie z.B. mit *substitute* eine Ersetzung gemacht haben, die Sie nicht beabsichtigt haben, geben Sie ein:

```
:undo
```

und der alte Inhalt wird wiederhergestellt. Sie können mit *undo* auch *undo* selbst wieder rückgängig machen. *edit* gibt eine Warnung aus, wenn Kommandos mehr als eine Zeile im *edit*-Puffer betreffen.

Die Kommandos *write* und *quit* können nicht rückgängig gemacht werden.

Zeilen ausgeben

Die nächste Zeile wird ausgegeben, wenn Sie

:↵

eingeben. Ein halber Bildschirm von Zeilen wird ausgegeben, wenn Sie eingeben:

: CTRL D (gleichzeitig gedrückt)

Einen Ausschnitt von einigen Zeilen vor bis nach der aktuellen Zeile erhalten Sie durch Eingabe von:

```
:z.
```

Zur aktuellen Zeile wird nun die zuletzt ausgegebene. Sie können auf die Position vor dem Kommando *z.* zurückgehen, indem Sie eingeben:

```
:''
```

Sie können andere Ausschnitte wählen, indem Sie hinter *z* andere Zeichen anfügen:

```
:z-
```

gibt einen Bildschirm aus (24 Zeilen). Die letzte ausgegebene Zeile wird zur aktuellen Zeile.

```
:z+
```

gibt den nächsten Bildschirm aus.
Wenn Sie weniger sehen wollen, z.B. nur 13 Zeilen, geben Sie ein:

```
:z.13
```

Die alte aktuelle Zeile ist die siebte Zeile auf dem Bildschirm, also die mittlere Zeile. Wenn Sie eine gerade Zahl angeben, gibt *edit* eine Zeile weniger aus, so daß die aktuelle Zeile wieder genau in der Mitte erscheint. Die Methode, eine Anzahl anzugeben, funktioniert auch bei anderen Kommandos. Sie können z.B. die nächsten fünf Zeilen ab der aktuellen Zeile löschen, indem Sie eingeben:

```
:delete 5
```

Suchen von Textmustern

Um Stellen im Text zu suchen, können Sie Zeilennummern verwenden, wenn Sie sie zufällig wissen. Diese Methode ist nicht sehr praktisch, da sich die Nummern durch Einfügen und Löschen von Zeilen ändern. Sie können in der Datei vorwärts nach einem bestimmten Textmuster suchen, indem Sie eingeben:

```
:/textmuster/
```

Wird dabei das Dateiende erreicht, bevor der gesuchte Text gefunden ist, wird die Suche vom Dateianfang bis zu der Zeile fortgesetzt, an der sie begonnen wurde. Rückwärts suchen Sie, indem Sie eingeben:

```
:?textmuster?
```

Wenn *textmuster* selbst eines der Begrenzungszeichen / oder ? enthält, müssen Sie es mit einem Gegenschrägstrich \ davor entwerten. Nützlich sind in diesem Zusammenhang zwei Spezialfälle des Suchkommandos, mit denen Sie nach *textmuster* nur am Zeilenanfang oder am Zeilenende suchen können:

`:/^textmuster/`	sucht *textmuster* am Zeilenanfang und
`:/textmuster$/`	sucht *textmuster* am Zeilenende.

Das zweite / bzw. ? kann in diesen Kommandos weggelassen werden.

Spezielle Zeilen

Die aktuelle Zeile können Sie explizit mit einem Punkt , die letzte Zeile mit einem Dollar-Zeichen bezeichnen.
Diese Bezeichnungen sind vor allem in Bereichsangaben nützlich. Mit

```
:.,$print
```

wird z.B. der Rest der Datei ausgegeben. Der Aufruf

```
:$delete
```

löscht die letzte Zeile einer Datei, unabhängig davon, welche Zeile gerade die aktuelle Zeile ist.

Sie können auch Positionen relativ zu diesen Bezeichnungen angeben,
z.B. ist

`:$-5`	die 5. Zeile vor der letzten,
`:.+20`	die 20. Zeile nach der aktuellen Zeile.

Aktuelle Position

Die aktuelle Zeilennummer können Sie abfragen mit:

```
:.=
```

oder mit

```
=
```

Bereichsangaben

Wenn Sie Bereiche ändern, kopieren, verschieben oder sich ausgeben lassen möchten, können Sie vor den Kommandos Bereichsangaben machen, z.B. löschen Sie mit

```
:10,20delete
```

den Bereich von Zeile 10 bis 20 einschließlich.

Verschieben

Bereiche können Sie wie folgt verschieben: Bereich feststellen (mit Hilfe des Kommandos =), z.B. Bereich:

```
10,20
```

Für diesen Bereich rufen Sie auf:

```
:10,20delete a
```

Damit wird der Bereich von Zeile 10 bis 20 gelöscht und gleichzeitig in einen Puffer mit dem Namen *a* (*a*-Puffer) geschrieben. *edit* hat 26 dieser Hilfspuffer. Sie heißen *a,b,...z*.

Sie können diesen Text später wieder aus dem *a*-Puffer holen.

```
:put a
```

holt den Inhalt des *a*-Puffers und schreibt ihn hinter die aktuelle Zeile. Der Inhalt eines Puffers bleibt nur innerhalb einer Editor-Sitzung erhalten und zwar solange, bis er mit anderen Zeichen überschrieben wird.

Datei wechseln

Wenn Sie dabei auch noch die Datei wechseln wollen, geben Sie nach dem Zwischenspeichern ein:

```
:edit andere_datei
```

Dadurch wechselt *edit* die Datei auf *andere_datei* und Sie können wie oben beschrieben fortfahren.

Kopieren

Wenn Sie innerhalb einer Datei kopieren, z.B. den Bereich 10,20 an das Dateiende, geben Sie ein:

```
:10,20move $
```

Sie brauchen in diesem Fall keinen Hilfspuffer.
Wenn Sie in eine andere Datei kopieren, benutzen Sie statt des Kommandos *delete* das Kommando *yank*. Alles weitere geht genauso wie beim Verschieben.

KOMMANDOÜBERSICHT

Die folgende Liste enthält die *edit*-Kommandos in alphabetischer Reihenfolge.

a	append	anfügen
c	change	löschen und ersetzen
d	delete	Zeile(nbereich) löschen oder löschen und in einen benannten Puffer schreiben
f	file	Anzahl der Zeilen im Puffer ausgeben
i	insert	einfügen
m	move	Zeilenbereich verschieben
n1[,n2]	---	Zeile mit Nummer n1 oder Zeilenbereich von n1 bis n2 ausgeben
p	print	ausgeben
put	put	Inhalt eines bestimmten (benannten) Puffers in Dateikopie schreiben
q	quit	*edit* verlassen
q!	quit	*edit* ohne Warnung verlassen
s	substitute	suchen und ersetzen
u	undo	Kommando rückgängig machen
w	write	Pufferinhalt in Datei schreiben
y	yank	Zeile(nbereich) in (benannten) Puffer schreiben
z	---	umgebende Zeilen ausgeben
.=	---	aktuelle Zeilennummer ausgeben
↵	---	zur nächsten Zeile im Puffer springen
END oder CTRL-D	---	halben Bildschirm mit Zeilen füllen
/text[/]	---	nach *text* suchen
/^text[/]	---	nach *text* am Zeilenanfang suchen
/text$[/]	---	nach *text* am Zeilenende suchen
?text[?]	---	rückwärts nach *text* suchen
?^text[?]	---	rückwärts nach *text* am Zeilenanfang suchen
?text$[?]	---	rückwärts nach *text* am Zeilenende suchen

ENDE-STATUS

immer 0

FEHLERMELDUNGEN

`At end-of-file`

Das Dateiende ist erreicht.

`Badly formed address`

Sie haben die Adresse(n) in einer unzulässigen Form angegeben, z.B. 5.7 statt 5,7.

`File is read only`

Sie haben für die Datei nur das Leserecht (siehe *chmod*) und dürfen deshalb den Puffer nicht in die Datei zurückschreiben.

`First address exceeds second`

Sie haben einen ungültigen Zeilenbereich angegeben: die zweite Adresse liegt vor der ersten.

`Not an editor command`

Sie haben ein ungültiges *edit*-Kommando angegeben.

`Nothing in register b`

Sie haben versucht, den Inhalt eines leeren benannten Puffers zu verwenden.

`Not that many lines in the buffer`

Sie haben eine zu hohe Adresse angegeben: im *edit*-Puffer befinden sich nicht so viele Zeilen.

`No write since last change (:quit! overrides)`

Warnung: Sie haben den Pufferinhalt seit der letzten Veränderung nicht mit *write* gesichert.

BEISPIEL

Im folgenden Beispiel werden einige *edit*-Kommandos vorgeführt und erläutert.

```
$ edit bueroartikel          - Aufruf von edit mit neuem Dateinamen
"bueroartikel" [New file]    - Meldungstext von edit
:i                           - Vor aktueller Zeile einfügen
No lines in the buffer       - Keine Zeilen im Puffer
:a                           - Nach aktueller Zeile (= am Dateianfang)
                               einfügen
Bleistift                    - Eingabetext
Radiergummi
Ringbuch
.                            - Eingabe beenden
:p                           - Aktuelle Zeile ausgeben
Ringbuch                     - Aktuelle (= letzte bearbeitete) Zeile
:1,$p                        - Erste bis letzte Zeile ausgeben
Bleistift
Radiergummi
Ringbuch
:3d                          - Dritte Zeile löschen
Radiergummi                  - Neue aktuelle Zeile (= letzte) wird
                               ausgegeben
:1,$p                        - Erste bis letzte Zeile ausgeben
Bleistift
Radiergummi
:2i                          - Vor der zweiten Zeile einfügen
Filzstift
Spitzer
.                            - Eingabe beenden
:1,$p                        - Erste bis letzte Zeile ausgeben
Bleistift
Filzstift
Spitzer
Radiergummi

:1,2delete a                 - Zeile 1 bis 2 löschen und in
                               Puffer a schreiben
2 lines deleted              - Bestätigung
Spitzer                      - Neue aktuelle (= neue erste) Zeile
                               wird ausgegeben
:w                           - Sichern des Inhalts des Editorpuffers
                               durch Zurückschreiben
                               in die beim Aufruf angegebene Datei
"bueroartikel" [New file] 2 lines, 20 characters
:1,$p                        - Erste bis letzte Zeile ausgeben
Spitzer
Radiergummi
:edit stifte                 - Neue Datei stifte editieren
"stifte" No such file or directory   - Bestätigung einer neuen Datei
:put a                       - Puffer a hineinschreiben
2 lines puted                - Bestätigung
Filzstift                    - Aktuelle (= letzte) Zeile von stifte
                               wird ausgegeben
```

```
:1,$p                          - Erste bis letzte Zeile ausgeben
Bleistift
Filzstift
:w                             - Zurückschreiben des Inhalts des Editorpuffers
                                 in die zuletzt editierte Datei,
                                 hier stifte
"stifte" [New file] 2 lines, 20 characters
:a                             - An aktuelle (= letzte) Zeile anfügen
Kugelschreiber                 - Eingabetext
.                              - Eingabe beenden
:q                             - Versuch, Editor zu verlassen
No write since last change (:quit! overrides)
                               - Warnung, daß der Inhalts des Editorpuffers
                                 seit der letzten Veränderung
                                 nicht mit w gesichert wurde
:w                             - Zurückschreiben des Inhalts
                                 des Editorpuffers in die zuletzt
                                 editierte Datei, hier stifte
"stifte" 3 lines, 35 characters      - Bestätigung
:q                             - Editor verlassen
$                              - Bereitzeichen der Shell
```

SIEHE AUCH

ced, *ed*, *ex*, *vedit*, *vi*

egrep
Muster suchen (expression grep)

egrep liest Zeilen aus einer oder mehreren Dateien oder von der Standard-Eingabe und vergleicht die Zeilen mit den angegebenen Mustern. Ist mittels Optionen nichts anderes angegeben, schreibt *egrep* alle Zeilen, die zu einem der Muster passen, auf die Standard-Ausgabe.
Als Muster können Sie erweiterte reguläre Ausdrücke angeben (siehe *Tabellen und Verzeichnisse, Reguläre Ausdrücke*).
Geben Sie mehrere Eingabedateien an, dann wird jeder Ausgabezeile der Name der betreffenden Datei vorangestellt.

egrep[␣option]...[␣musterliste][␣datei]...

Die Muster, mit denen *egrep* die Eingabezeilen vergleichen soll, geben Sie entweder über *musterliste* oder über die Option *-e musterliste* oder über die Option *-f musterdatei* an. Eines dieser drei Argumente müssen Sie angeben, mehrere zusammen sind nicht erlaubt.

Keine Option angegeben
: *egrep* gibt alle Zeilen aus, die zu mindestens einem der in *musterliste* angegebenen Muster passen. Geben Sie mehrere Eingabedateien an, dann wird jeder Ausgabezeile der Name der Datei vorangestellt, aus der die Zeile gelesen wurde.

option

-b
: (b - block) Jeder Ausgabezeile wird die Nummer des Blockes vorangestellt, in dem sie enthalten ist.
Die Blöcke, aus denen eine Datei besteht, sind je 512 byte groß und werden, mit 0 beginnend, durchnumeriert.
Option *-b* kann hilfreich sein, wenn die Nummern von Blöcken nach dem Kontext ermittelt werden sollen (siehe *od*, Argument *offset*).

-c
: (c - count) *egrep* gibt nur die Anzahl der gefundenen Zeilen aus (das sind die Zeilen, die *egrep* ohne die Option *-c* ausgeben würde, siehe *Beispiel 4*); die Zeilen selbst werden nicht ausgegeben.

-e␣musterliste
: (e - expression) Diese Option brauchen Sie, wenn der erste Ausdruck in *musterliste* mit einem Bindestrich - beginnt. Zusammen mit *-e* wird eine solche Musterliste nicht als Option interpretiert, sondern als Liste von Mustern, mit denen *egrep* die Eingabezeilen vergleichen soll.

-f musterdatei
(f - file) *egrep* liest die Musterliste aus der Datei *musterdatei*. Jede Zeile von *musterdatei* wird als ein erweiterter regulärer Ausdruck interpretiert.

-i
oder **-y**
(i - ignore) *egrep* unterscheidet beim Vergleich nicht zwischen Groß- und Kleinbuchstaben.

-h
(h - hidden) Beim Durchsuchen mehrerer Eingabedateien unterläßt *egrep* die Voranstellung der Dateinamen vor jeder Ausgabezeile.

-l
(l - list) *egrep* gibt nur die Namen der Dateien aus, die mindestens eine der gefundenen Zeilen enthalten. (Das sind die Zeilen, die *grep* ohne die Option *-l* ausgeben würde, siehe *Beispiel 5*). Jeder Dateiname wird nur einmal ausgegeben. Die Zeilen selbst gibt *egrep* nicht aus.

-n
(n - number lines)
Jeder Ausgabezeile wird die Zeilennummer aus der betreffenden Eingabedatei vorangestellt, wobei von 1 an numeriert wird. Liest *egrep* von der Standard-Eingabe, bezieht sich die Zeilennummer auf die Standard-Eingabe.

-v
(v - vice versa)
egrep gibt alle Zeilen aus, die zu *keinem* der angegebenen Muster passen.

Zusammen mit Option *-c*:
egrep gibt nur die Anzahl solcher Zeilen aus.

Zusammen mit Option *-l*:
egrep gibt nur die Namen der Dateien aus, die solche Zeilen enthalten.

musterliste
Liste von erweiterten regulären Ausdrücken, mit denen *egrep* die Eingabezeilen vergleichen soll (siehe *Tabellen und Verzeichnisse, Reguläre Ausdrücke*). Die regulären Ausdrücke trennen Sie durch Neue-Zeile-Zeichen. Ein Neue-Zeile-Zeichen in *musterliste* wird wie ein senkrechter Strich | (Zeichen für die Alternative) in einem erweiterten regulären Ausdruck interpretiert.
Reguläre Ausdrücke der Form (r|s) können Sie auch ohne Klammern angeben: r|s (siehe *Beispiel 1*).
Enthält *musterliste* Neue-Zeile-Zeichen oder sonstige Zeichen, die für die Shell eine Sonderbedeutung haben, dann schließen Sie *musterliste* in Hochkommas ein: *'musterliste'*.

Beginnt der erste Ausdruck in *musterliste* mit einem Bindestrich -, dann müssen Sie *musterliste* mit Option *-e* angeben, da sonst *musterliste* selbst als Option interpretiert wird.

datei
Name der Datei, die *egrep* durchsuchen soll. Pro Aufruf können Sie mehrere Dateinamen angeben.

datei nicht angegeben:
egrep liest die Eingabezeilen von der Standard-Eingabe.

grep, fgrep und egrep

Die Kommandos *grep*, *fgrep* und *egrep* sind von der Oberfläche her weitgehend identisch. Im folgenden sind die wichtigsten Unterschiede zwischen diesen Kommandos aufgeführt.

grep verarbeitet einfache reguläre Ausdrücke. Pro Aufruf können Sie nur einen einzigen regulären Ausdruck angeben.

fgrep verarbeitet nur Zeichenketten. Pro Aufruf können Sie jedoch mehrere Zeichenketten angeben: Die Zeichenketten geben Sie entweder direkt in der Aufrufzeile, getrennt durch Neue-Zeile-Zeichen, an oder Sie übergeben sie in einer Datei.
fgrep kann effizient sehr viele Zeichenketten suchen: *fgrep* sucht jede einzelne Zeile nach allen Zeichenketten ab.

egrep verarbeitet erweiterte reguläre Ausdrücke. Diese umfassen u.a. die einfachen regulären Ausdrücke bis auf eine Ausnahme: Der einfache reguläre Ausdruck \(*regausdruck*\) hat bei erweiterten regulären Ausdrücken keine Sonderbedeutung und wird deshalb auch nicht von *egrep* verarbeitet.
Pro Aufruf können Sie mehrere reguläre Ausdrücke, durch Neue-Zeile-Zeichen getrennt, angeben. *egrep* interpretiert diese Neue-Zeile-Zeichen wie einen senkrechten Strich (Zeichen für die Alternative, siehe *Tabellen und Verzeichnisse, Reguläre Ausdrücke*)
Die regulären Ausdrücke geben Sie entweder direkt in der Aufrufzeile an, oder Sie übergeben sie in einer Datei.

ENDE-STATUS

0 Zeilen gefunden
1 keine Zeile gefunden
2 Syntaxfehler oder "Datei kann nicht geöffnet werden". Dieser Ende-Status gilt auch dann, wenn in anderen Eingabedateien Zeilen gefunden wurden.

BEISPIELE

Grundlage für alle Beispiele sind die Dateien *kunden1* und *kunden2*. Sie haben folgenden Inhalt:

kunden1:

```
080685     999.98  20 LE  Art.  038   Fa. Holzinger
120387    1240.25   3 LE  Art.  023   Fa. Wanninger
180588     330.87   1 LE  Art.  332   Fa. Wanninger
```

kunden2:

```
hinterhuber berta, rosenheim, zugspitzstr.1
wanninger herbert, muenchen 5, kirschstr.3
```

1. Zeilen ausgeben, die zu einem Muster passen (keine Option und Option *-i*):

```
$ egrep Wanninger kunden1 kunden2
kunden1:120387    1240.25   3 LE  Art.  023   Fa. Wanninger
kunden1:180588     330.87   1 LE  Art.  332   Fa. Wanninger
```

Wenn Sie auch Zeilen mit kleingeschriebenem *wanninger* ausgeben lassen möchten, geben Sie ein:

```
$ egrep -i wanninger kunden1 kunden2
kunden1:120387    1240.25   3 LE  Art.  023   Fa. Wanninger
kunden1:180588     330.87   1 LE  Art.  332   Fa. Wanninger
kunden2:wanninger herbert, muenchen 5, kirschstr.3
```

Kompliziertere Muster stellen Sie mit Hilfe von regulären Ausdrücken dar, z.B.:

Zeilen ausgeben, die die Zeichenkette *Holzinger* oder *Wanninger* enthalten:

```
$ egrep '(Holz|Wann)inger' kunden1 kunden2
kunden1:080685     999.98  20 LE  Art.  038   Fa. Holzinger
kunden1:120387    1240.25   3 LE  Art.  023   Fa. Wanninger
kunden1:180588     330.87   1 LE  Art.  332   Fa. Wanninger
```

Statt des regulären Ausdrucks (Holz|Wann)inger können Sie auch den folgenden regulären Ausdruck angeben:

```
(Holzinger|Wanninger)
```

Hier können Sie die Klammern weglassen:

```
Holzinger|Wanninger
```

Statt des senkrechten Strichs im letzten Ausdruck Holzinger|Wanninger können Sie auch ein Neue-Zeile-Zeichen angeben (siehe *Beispiel 2*).

2. Mehrere Muster angeben (keine Option und Option *-f*):

```
$ egrep '^1↵
> 1$' kunden1 kunden2
kunden1:120387    1240.25    3 LE  Art.  023   Fa. Wanninger
kunden1:180588     330.87    1 LE  Art.  332   Fa. Wanninger
kunden2:hinterhuber berta, rosenheim, zugspitzstr.1
```

Sie können die beiden Muster auch in eine Datei *namen* schreiben (je Muster eine Zeile) und *egrep* folgendermaßen aufrufen:

```
$ egrep -f namen kunden1 kunden2
```

Dasselbe Ergebnis erhalten Sie, wenn Sie das Neue-Zeile-Zeichen, das die Muster ^1 und 1$ trennt, durch einen senkrechten Strich ersetzen:

```
$ egrep '^1|1$' kunden1 kunden2
```

3. Zeilen ausgeben, die zu *keinem* der angegebenen Muster passen (Option *-v*):

```
$ egrep -v '^1↵
> 1$' kunden1 kunden2
kunden1:080685     999.98  20 LE  Art.  038   Fa. Holzinger
kunden2:wanninger herbert, muenchen 5, kirschstr.3
```

Mit obigem Aufruf erhalten Sie also alle Zeilen, die weder mit 1 beginnen noch mit 1 enden. Dasselbe Ergebnis erhalten Sie mit folgendem Aufruf (siehe *Beispiel 2*):

```
$ egrep -v '^1|1$' kunden1 kunden2
```

4. Anzahl der gefundenen Zeilen ausgeben (Option *-c*):

   Zuerst soll für jede Eingabedatei die Anzahl der Zeilen, die mit einer 1 beginnen, ausgegeben werden.

```
$ egrep -c '^1' kunden1 kunden2
kunden1:2
kunden2:0
```

   Nun soll die Anzahl der Zeilen, die *nicht* mit einer 1 beginnen, ausgegeben werden.

```
$ egrep -c -v '^1' kunden1 kunden2
kunden1:1
kunden2:2
```

5. Nur Dateinamen ausgeben (Option *-l*):

   Zuerst sollen die Namen der Dateien, die Zeilen mit einer 1 am Anfang enthalten, ausgegeben werden.

```
$ egrep -l '^1' kunden1 kunden2
kunden1
```

   Nun sollen die Namen der Dateien, die Zeilen ohne 1 am Anfang enthalten, ausgegeben werden.

```
$ egrep -l -v '^1' kunden1 kunden2
kunden1
kunden2
```

6. Gefundene Zeilen mit Zeilennummer ausgeben (Option *-n*):

```
$ egrep -n -i wanninger kunden1 kunden2
kunden1:2:120387   1240.25   3 LE  Art.  023   Fa. Wanninger
kunden1:3:180588    330.87   1 LE  Art.  332   Fa. Wanninger
kunden2:2:wanninger herbert, muenchen 5, kirschstr.3
```

SIEHE AUCH

ed, fgrep, grep, sed, sh

env
Umgebung bei Ausführung von Kommandos ändern (set environment)

Mit *env* können Sie sich die aktuellen Umgebungsvariablen und ihre Werte ausgeben lassen oder sie für ein Kommando verändern. *env* liest die aktuelle Umgebung ein, ändert sie entsprechend der Angabe *name=wert* und führt das Kommando dann in der veränderten Umgebung aus. Die schon vorhandenen Angaben für *name* und *wert* werden durch die neuen Angaben überschrieben und vor Ausführung des Kommandos der usprünglichen Umgebung hinzugefügt. Die neuen Angaben bilden zusammen mit den unveränderten Umgebungsvariablen die für die Ausführung von *kommando* gültige Umgebung.
Wenn kein Kommando angegeben ist, wird die durch *env* veränderte Umgebung ausgegeben.

env[␣-][␣name=wert]...[␣kommando[␣arg]...]

-
die ursprüngliche Umgebung wird ignoriert; *kommando* wird dann exakt in der angegebenen Umgebung ausgeführt.

name=wert
name ist der Name einer Variablen, die für *kommando* Gültigkeit haben soll.

wert ist der Wert von *name*, der für *kommando* Gültigkeit haben soll.

kommando
Name des Kommandos oder der Shell-Prozedur, die Sie unter der definierten Umgebung ausführen lassen möchten.

arg
Argument, z.B. Stellungs- oder Kennwortparameter, das Sie an *kommando* übergeben können.

BEISPIELE

1. Ausgabe der aktuellen Werte der Umgebungsvariablen:

```
$ env
DRUO1=-ws=GO1
HOME=/home/sysiphus
LANG=En_US.ASCII
LOGNAME=sysiphus
MAIL=/var/mail/sysiphus
PATH=::/usr/bin:/usr/sbin:/opt/bin:
PS1=$
SHELL=/sbin/sh
TERM=97801
TERMCAP=/etc/termcap
TTY=/dev/004
TZ=MET-1MDT
USER=sysiphus
```

2. Ausgabe der geänderten Werte der Umgebungsvariablen:

```
$ env PATH=$HOME/proz
DRUO1=-ws=GO1
HOME=/home/sysiphus
LANG=En_US.ASCII
LOGNAME=sysiphus
MAIL=/var/mail/sysiphus
PATH=/home/sysiphus/proz
PS1=$
SHELL=/sbin/sh
TERM=97801
TERMCAP=/etc/termcap
TTY=/dev/004
TZ=MET-1MDT
USER=sysiphus
```

   Die Umgebungsvariable *PATH* wurde geändert.

3. Ausgabe der geänderten Umgebungsvariablen mit der Option -:

```
$ env - PATH=$HOME/proz
PATH=/home/sysiphus/proz
```

   Die ursprüngliche Umgebung wird ignoriert.

4. Aufruf der Datei *fly*, die sich in */home/sysiphus/SPRUECHE*, also in einem Unterdateiverzeichnis des HOME-Dateiverzeichnisses befindet.

   Inhalt der Datei *fly*:

```
echo "Wenn hinter $1 $1 $2, $2 $1 $1 nach !"
```

   fly wird nun von einer beliebigen Stelle in Ihrem Dateibaum aus aufgerufen, hier mit den Argumenten *Fliegen* und *fliegen*.

```
$ env PATH=$HOME/SPRUECHE fly Fliegen fliegen
Wenn hinter Fliegen Fliegen fliegen,
fliegen Fliegen Fliegen nach!
```

Mit der neuen Variablendefinition für *PATH* legen Sie fest, wo das eingegebene Kommando, in diesem Fall die Datei *fly*, gesucht werden soll: in einem Unterdateiverzeichnis des *HOME*-Dateiverzeichnisses, das Sie zusammen mit dem Wert der Variablen *HOME (\$HOME)* angeben.
Als Argumente übergeben Sie an die Stellungsparameter \$1 und \$2 die Zeichenketten *Fliegen* und *fliegen*.

Der Inhalt der Datei *fly* wird nur deshalb korrekt ausgeführt, weil das Kommando *echo* ein eingebautes *sh*-Kommando ist. Alle SINIX-Kommandos, die in /usr/bin, /usr/sbin oder /opt/bin stehen, können durch die Veränderung der Variablen *PATH* nicht mehr gefunden werden. Damit die SINIX-Kommandos weiterhin ausgeführt werden, muß die Variable *PATH* wie im folgenden Beispiel geändert werden.

5. Aufruf der Datei *loesch*, die sich im Dateiverzeichnis */home/sysiphus/proz* befindet. In dieser Datei steht eine Prozedur, die zwei Dateien vergleicht und die eine löscht, wenn die Dateien gleich sind.

   Inhalt der Datei *loesch*:

   ```
   if cmp -s $1 $2
   then
   rm $2
   fi
   ```

   Aufruf von *loesch* von einer beliebigen Stelle in Ihrem Dateibaum aus mit den Argumenten *dat1* und *dat2*:

   ```
   $ env PATH=$PATH:$HOME/proz loesch dat1 dat2
   ```

   Hier wurde an den ursprünglichen Suchpfad der neue angefügt, damit sowohl die Prozedur *loesch* als auch die in der Prozedur enthaltenen SINIX-Kommandos ausgeführt werden können. Wenn nur der Suchpfad für *loesch* angegeben wird, wird folgender Fehler gemeldet:

   ```
   /home/sysiphus/proz/loesch: cmp: not found
   ```

SIEHE AUCH

sh, set, exec profile, environ [7]

eval
Aufruf-Argumente bearbeiten und als Kommando ausführen (evaluate)

Mit dem in die Bourne-Shell *sh* eingebauten Kommando *eval* können Sie der Shell *sh* die angegebenen Aufruf-Argumente als Kommando übergeben. Die Shell führt dieses Kommando aus.

Auf diese Weise werden die Aufruf-Argumente zweimal von der Shell bearbeitet:

- Das erste Mal, wenn die Shell die *eval*-Kommandozeile bearbeitet.
- Das zweite Mal, wenn die Shell die bearbeiteten Aufruf-Argumente als Kommando ausführt. Die Shell bearbeitet jede Kommando-Zeile vor der Ausführung.

Wann brauchen Sie eval?

Die Shell bearbeitet jede Kommando-Zeile in mehreren Schritten (siehe *sh*, *Wie bearbeitet die Shell die Kommando-Zeile?*). Bei jedem Bearbeitungsschritt interpretiert die Shell bestimmte Sonderzeichen. Die ursprüngliche Kommando-Zeile kann sich durch die Bearbeitungsschritte verändern.

Wenn die Shell beispielsweise eine Variable durch ihren Wert oder ein Kommando durch seine Ausgabe ersetzt, können in der Kommando-Zeile Sonderzeichen auftreten, die die Shell in den restlichen Bearbeitungsschritten nicht mehr interpretiert. Das eingebaute *sh*-Kommando *eval* sorgt dafür, daß die Shell verbliebene Sonderzeichen im zweiten Durchgang interpretiert (siehe *BEISPIELE*).

eval [argument ...]

argument
: beliebige Zeichenkette, die durch Leer- oder Tabulatorzeichen begrenzt wird. Das letzte Argument wird durch ein Kommando-Trennzeichen abgeschlossen.

  Sie können beliebig viele Argumente angeben, jeweils getrennt durch mindestens ein Leer- oder Tabulatorzeichen.

  Die Shell führt diese Argumente als Kommando aus.

BEISPIELE

1. Die folgende Kommando-Zeile wird nur mit *eval* wie gewünscht ausgeführt:

```
$ kom='date;who'
$ $kom
date;who: not found
$ eval $kom
Thu Mar 09 15:46:02 MET 1989
udo      tty004   Mar  9 09:20
```

Die Shell ersetzt die Variable *kom* durch ihren Wert. Nach der Variablen-Ersetzung erkennt die Shell den Strichpunkt ; nicht mehr als Kommando-Trennzeichen. Bei Aufruf mit *eval* interpretiert die Shell den Strichpunkt ; wie gewünscht, weil sie das Aufruf-Argument $kom zweimal bearbeitet.

2. Die Shell-Prozedur *evaltest* soll zeigen, wie das Kommando *eval* von der Shell abgearbeitet wird. Sie hat folgenden Inhalt:

```
set -x
kennung=max
for i in 1 2 3 4
do
  eval gruppe$i=$kennung$i
  eval echo \$gruppe$i
done
```

set -x bewirkt, daß die Shell, die die Shell-Prozedur ausführt, die bearbeiteten Kommandos vor der Ausführung auf die Standard-Fehlerausgabe schreibt.
Wenn Sie diese Shell-Prozedur ausführen, erhalten Sie am Bildschirm folgende Ausgabe, allerdings ohne Zeilennummern:

```
   $ sh evaltest
1  kennung=max
2  + eval gruppe1=max1
3  gruppe1=max1
4  + eval echo $gruppe1
5  + echo max1
6  max1
.
.
.
```

Hier ist nur die Ausgabe nach dem ersten Schleifendurchlauf gezeigt.

Zeile 2 zeigt die Bearbeitung der ersten *eval*-Kommandozeile:
Die Shell hat $i durch den Wert 1 und $kennung durch den Wert max ersetzt. Der Aufruf mit *eval* ist nötig, weil die Shell im ersten Durchgang das Gleichheitszeichen nicht als Zeichen für Zuweisung erkennt. Die Shell führt eine Zuweisung nur aus, wenn der Variablen-Name, also die Zeichenkette vor dem Gleichheitszeichen = mit einem Buchstaben oder Unterstrich _ beginnt und nur Buchstaben, Ziffern und Unterstriche enthält. *gruppe$i* ist wegen des Dollarzeichens $ kein erlaubter Variablen-Name.

Zeile 3 zeigt die Ausführung der Zuweisung:
Bei der zweiten Bearbeitung der Aufruf-Argumente interpretiert die Shell das Gleichheitszeichen wie gewünscht, weil das Argument vor = ein erlaubter Name für eine Shell-Variable ist.

Zeile 4 zeigt die Bearbeitung der zweiten *eval*-Kommandozeile:
Die Shell hat den Stellungsparameter $i durch den Wert 1 ersetzt und das Entwertungszeichen \ vor $ entfernt. Hier ist *eval* nötig, damit die Variable *gruppe1* durch ihren Wert ersetzt.
Zeile 5 zeigt die Bearbeitung des Kommandos *echo*. Hier hat die Shell *$gruppe1* durch den Wert max1 ersetzt. Zeile 6 enthält die Ausgabe des Kommandos *echo*.

SIEHE AUCH

set, *sh*

ex
Zeilenorientierter Editor

ex ist ein zeilenorientierter Texteditor.

ex bietet Ihnen verschiedene Bearbeitungsmodi.

- Im *ex*-Eingabe-Modus können Sie direkt Text eingeben.
- Im *ex*-Kommando-Modus können Sie Kommandos eingeben, um zum Beispiel
  - die Schreibmarke zu positionieren
  - Textmuster mit regulären Ausdrücken zu suchen (und zu ersetzen)
  - in eine andere Datei zu wechseln
  - eine Shell aufzurufen.

Außerdem können Sie vom zeilenorientierten Editor *ex* in den bildschirmorientierten Editor *vi* wechseln.

Aufbau dieser Beschreibung

Nach der Beschreibung des Aufrufs von *ex* auf SINIX-Ebene finden Sie folgende Abschnitte:

Arbeitsweise des *ex*

- Modi des *ex*
- Editor-Puffer sichern und *ex* verlassen
- Voreinstellung
- Aktuelle und sekundäre Datei
- Reguläre Ausdrücke
- Ersetzungszeichenketten
- Puffer
- Fehler- und Signalbehandlung

ex-Kommandos

- Adressen
- Parameter
- Kommandos

ex-Optionen

ex[_option][_datei]...

option

-s

(s - silent) Alle interaktiven Ausgaben des Editors werden unterdrückt. Die Option *-s* setzen Sie, wenn *ex* seine Kommandos aus einem Kommandoskript lesen soll.

Die Option *-s* ersetzt die alte Option -.

-t_markierung

(t - tag) Die Datei mit *markierung* wird von *ex* zum Editieren aufgerufen. *ex* positioniert dann die Zeile mit der Definition der Markierung in der Mitte des Bildschirms. Die Suchzeichenketten für die Definitionen müssen in der Datei *tags* im gleichen Verzeichnis enthalten sein. Diese Option wird z.B. von *C*-Programmieren dazu verwendet, den Editor beim Aufruf auf die Definition einer Funktion oder eines Makros zu positionieren. Die dafür benötigte *tags*-Datei muß dazu vorher mit dem Kommando *ctags* erzeugt worden sein.

-r_datei

(r - recover) Stellt Ihre *ex*-Sitzung von *datei* wieder her, falls das System oder *ex* während der Sitzung abgestürzt ist.
Bei einem Absturz des Systems oder des *ex*-Editors werden die Änderungen, die nur im Editor-Puffer stehen, nicht in die Datei geschrieben. SINIX versucht jedoch den Inhalt des Puffers zu retten, indem eine Kopie des Pufferinhalts angelegt wird, sofern dies möglich ist. *datei* wird mit den Änderungen, die Sie vor dem Absturz gemacht haben, in den *vi*-Puffer geholt. Sie können nun die Datei weiter editieren oder die Änderungen in eine Datei schreiben.

-L

Nach einem Absturz des Systems oder des *ex*-Editors wird eine Liste aller geretteten Dateien ausgegeben.

-R

(R - read-only) *datei* wird nur zum Lesen geöffnet. Damit können Sie ein versehentliches Überschreiben von *datei* verhindern. Jedoch können Sie den Pufferinhalt im Readonly-Modus in eine Datei mit anderem Namen schreiben.

Vorsicht

Die Datei kann im *ex*-Kommando-Modus mit *w!* dennoch überschrieben werden.

-v

(v - vi) Der Editor *vi* wird aufgerufen. Eine ausführliche Beschreibung des *vi* finden Sie in diesem Handbuch (siehe *vi*);

-c_kommando
Positioniert beim Aufruf des *ex* auf eine bestimmte Zeile der zu editierenden Datei oder führt ein *ex*-Kommando aus. Wenn Sie auf eine Zeile positionieren, steht die gewünschte Zeile danach in der Mitte des Bildschirms. Wenn Sie ein *ex*-Kommando ausführen lassen, wird nach Ausführung des *ex*-Kommandos auf die letzte Zeile der Datei positioniert - falls das *ex*-Kommando keine Positionierung bewirkt (z.B. Suchen).

kommando nicht angegeben:
ex positioniert auf das Ende der Datei.

kommando angegeben:
kommando kann entweder eine Zeilenangabe (*n*) sein, ein Suchkommando (*/muster*) oder ein sonstiges Kommando des *ex* ("*ex-kommando*"). Es wird beim Aufruf des *ex* ausgeführt:

n
n ist eine ganze Zahl. *ex* positioniert auf die *n*-te Zeile der Datei.

/muster
ex positioniert auf die Zeile, die *muster* enthält. Falls *muster* Sonderzeichen enthält, müssen Sie die Sonderzeichen entwerten, damit die Shell diese nicht interpretiert.

'ex-kommando'
"ex-kommando"
ex-kommando kann ein beliebiges *ex*-Kommando sein. Das *ex*-Kommando muß in einfachen Hochkommas oder Anführungszeichen eingeschlossen werden, damit es von der Shell nicht interpretiert wird. Falls nicht schon durch das *ex*-Kommando positioniert wird, positioniert der *ex* auf die letzte Zeile der Datei.

Beispiel

Nach dem Aufruf des *ex* mit

```
ex -c /Dienstag termine
```

wird die Datei *termine* geöffnet und die Schreibmarke auf die erste Zeile der Datei positioniert, die das Wort *Dienstag* enthält (vgl. das Beispiel bei *vi*).

-x
Verschlüsselungsoption: *ex* führt beim Aufruf das Kommando *X* auf und fragt Sie nach einem Schlüssel. Dieser Schlüssel wird dann zum Ent- und Verschlüsseln von Text durch den Algorithmus des Kommandos *crypt* verwendet. *ex* klärt ab, ob der einzulesende Text verschlüsselt ist oder nicht. Die temporäre Pufferdatei wird ebenfalls mit einer Transformation des eingegebenen Schlüssels codiert (siehe *crypt*).

-C

Verschlüsselungsoption: wie *-x*, nur daß *ex* das Kommando *C* ausführt. Das Kommando *C* entspricht dem Kommando *X* mit dem Unterschied: es wird angenommen, daß jeder zu lesender Text verschlüsselt ist (siehe *crypt*).

datei

Name der Datei, die Sie editieren möchten. Wenn Sie mehrere Dateien angeben, werden sie in der Reihenfolge bearbeitet, in der Sie diese angegeben haben. Mit dem *ex*-Kommando *n* wechseln Sie in die nächste Datei.

datei nicht angegeben:
ex öffnet einen leeren Editor-Puffer, in den Sie Text schreiben können. Erst beim Zurückschreiben des Pufferinhalts mit *w* (write) in eine Datei oder durch ein *f*-Kommando bestimmen Sie deren Namen.

Vorsicht

Enthält Ihre Datei Nullbytes, dann werden diese von *ex* beim Lesen weggeworfen. Eine von *ex* geschriebene Datei enthält keine Nullbytes.

ARBEITSWEISE

ex arbeitet immer mit einem Editor-Puffer. Bei Beginn einer *ex*-Sitzung wird eine Arbeitskopie der Datei, die Sie bearbeiten, im Editor-Puffer angelegt. Alle Ihre Änderungen werden zunächst im Editor-Puffer vorgenommen. Sie werden erst gesichert, wenn Sie den Pufferinhalt mit *w* (write) in die Originaldatei zurückschreiben. Danach können Sie *ex* mit *q* (quit) verlassen.
Wenn Sie *ex* verlassen wollen, ohne die Änderungen in die Orginaldatei zurückzuschreiben, müssen Sie *q!* eingeben, ohne ein vorheriges *w*!

Ist in den Eingabedateien das ASCII-Zeichen NUL (Nullbyte) (siehe *Tabellen und Verzeichnisse, Zeichensatz ISO 646*) enthalten, so wird es gelöscht; in den Ausgabedateien kann es daher nicht enthalten sein.

Modi des ex

ex bietet Ihnen zwei Modi zur Bearbeitung einer Datei:

- den *ex*-Kommando-Modus und
- den *ex*-Eingabe-Modus.

Zusätzlich können Sie von *ex* in den *vi* wechseln, in dem wieder verschiedene Arbeitsmodi zur Verfügung stehen (ausführlicher siehe *vi*, *Modi des vi*).

Nach dem Aufruf befindet *ex* sich im Kommando-Modus, den Sie am Bereitzeichen Doppelpunkt : am Bildschirm erkennen. Mit den Kommandos *a* (append), *i* (insert) und *c* (change) wechseln Sie in den Eingabe-Modus, in dem Sie Text im Puffer ergänzen und ändern können (siehe *ex*-Kommandos).

Im Eingabe-Modus werden alle folgenden Eingabezeichen, auch verschiedene nichtdruckbare Zeichen, in den Puffer geschrieben. Kommandos werden im Eingabe-Modus nicht als solche interpretiert. Verlassen können Sie den Eingabe-Modus, indem Sie in der ersten Spalte einer neuen Zeile einen Punkt . eingeben und danach ↵ drücken.

Editor-Puffer sichern und ex verlassen

Um den Editor-Puffer zu sichern oder den *ex* zu verlassen, muß sich der *ex* im *ex*-Kommandomodus befinden.

Mit folgendem Kommando sichern Sie den Inhalt Ihres Editor-Puffers in die editierte oder eine angegebene Datei:

w[␣datei]
(w - write) Der Inhalt des Editor-Puffers wird in die angegebene Datei gesichert.

datei nicht angegeben:
Der Inhalt des Editor-Puffers wird in die editierte Datei geschrieben.

Um den *ex* zu verlassen, haben Sie folgende Möglichkeiten:

q
(q - quit) *ex* verlassen. Funktioniert nur, falls noch keine Änderungen am Editor-Puffer vorgenommen wurden oder der geänderte Editor-Puffer in eine Datei gesichert wurde.

q!
(q - quit) *ex* verlassen; am Editor-Puffer vorgenommene Änderungen gehen verloren.

x
ex verlassen und den geänderten Editor-Puffer in die editierte Datei schreiben.

wq[␣datei]
(wq - write and quit) *ex* verlassen und den geänderten Editor-Puffer in die angegebene Datei *datei* schreiben.

datei nicht angegeben:
Der Inhalt des Editor-Puffers wird in die editierte Datei geschrieben.

Vorsicht

ex gibt keine Warnung aus, wenn Sie beim Verlassen des Editors noch Text in benannten Puffern haben, die Sie nicht verwendet haben. Der Text ist unwiederbringlich verloren.

Voreinstellung

Sie können den *ex* in begrenztem Rahmen an Ihre Bedürfnisse und Gewohnheiten anpassen. Dazu müssen Sie das *ex*-Kommando *se* (set) verwenden, mit dem Sie bestimmte Optionen setzen oder verändern können (siehe *ex*-Optionen). Diese Änderungen gelten in der aktuellen Sitzung. Wie Sie diese Änderungen dauerhaft machen können, ist beschrieben bei *vi*, *Voreinstellung des vi*.

Aktuelle und sekundäre Datei

Die Datei, die, bzw. deren Kopie im Augenblick editiert wird, wird als aktuelle Datei bezeichnet. Die sekundäre Datei ist die Datei, die zuletzt bei einem Editier-Kommando angegeben wurde. Falls die sekundäre Datei zur aktuellen Datei wurde, wird die vorhergehende aktuelle Datei zur sekundären Datei. Die aktuelle Datei können Sie mit dem Prozentzeichen %, die sekundäre Datei mit dem Nummernzeichen # angeben; in Dateinamen wird % durch den Namen der aktuellen Datei und # durch den Namen der sekundären Datei ersetzt.

Beispiel

```
w %.bak
```

sichert die aktuelle Datei in eine Datei mit dem gleichen Namen, aber der Endung *.bak*.

Reguläre Ausdrücke

Die Bedeutung von Sonderzeichen in regulären Ausdrücken hängt bei *ex* davon ab, ob die Option *magic* gesetzt ist (vgl. *ex*-Optionen).

magic gesetzt:

zeichen
Ein einfaches Zeichen steht für sich selbst. Die folgenden Zeichen sind keine einfachen Zeichen, sondern Metazeichen:

^ (Dach) am Anfang eines Musters

$ (Dollar-Zeichen) am Ende eines Musters

* (Stern) überall außer am Anfang eines Musters

. (Punkt) an beliebiger Stelle in einem Muster

[(öffnende eckige Klammer) an beliebiger Stelle in einem Muster

~ (Tilde) an beliebiger Stelle in einem Muster

Metazeichen haben eine besondere Bedeutung und müssen durch einen Gegenschrägstrich \ entwertet werden, wenn Sie ihre Sonderbedeutung verlieren sollen.

^
Am Anfang eines Musters steht ^ für den Zeilenanfang.

$
Am Ende eines Musters steht $ für das Zeilenende.

.
Ein beliebiges Zeichen.

\<
\>
Die Zeichen \< passen zum Beginn eines Wortes. Das Wort muß dabei mit einem Buchstaben, einer Ziffer oder einem Unterstreichungszeichen _ beginnen; vor dem Wort muß entweder der Zeilenanfang stehen oder ein Zeichen, das nicht zu den oben aufgeführten gehört. Die Zeichen \> passen zum Ende eines Wortes.

[zeichenkette]
Ein beliebiges Zeichen aus *zeichenkette*, wobei *zeichenkette* eine nicht leere Folge von Zeichen ist. Innerhalb von *zeichenkette* gibt es folgende Sonderbedeutungen:

- Ein Bindestrich - zwischen zwei Zeichen definiert einen Bereich, zum Beispiel *[a-z]* paßt zu *a*, *z* und allen Zeichen, die in der ASCII-Sortierreihenfolge dazwischen liegen.
- Ein Dach ^ als erstes Zeichen in *zeichenkette* kehrt die Bedeutung von *[zeichenkette]* um: Ein beliebiges Zeichen, das nicht in *zeichenkette* enthalten ist.

Diese Sonderbedeutungen können durch einen Gegenschrägstrich aufghoben werden.

*
Null-, ein- oder mehrmaliges Auftreten des vorausgehenden regulären Ausdrucks.

~
Paßt zu der Ersetzungszeichenkette, die beim letzten *s*-Kommando (substitute, siehe *ex*-Kommandos) verwendet wurde.

\(muster\)
Ein regulärer Ausdruck *muster* kann in durch Gegenschrägstrich \ entwertete runde Klammern eingeschlossen werden. Dies dient nur dazu, den regulären Ausdruck zu identifizieren, wenn er in Ersetzungszeichenketten verwendet werden soll (siehe *Ersetzungszeichenketten*).

rs
Eine Folge *rs* von zwei regulären Ausdrücken *r* und *s* ist wieder ein regulärer Ausdruck. *rs* paßt zu allen Zeichenketten, die aus einer zu *r* passenden Zeichenkette, gefolgt von einer zu *s* passenden Zeichenkette, bestehen.

nomagic gesetzt:

Ist *nomagic* gesetzt, so haben nur die Zeichen
- Dach ^ am Anfang eines Musters
- Dollar $ am Ende eines Musters sowie
- Gegenschrägstrich \

eine Sonderbedeutung.

Die Zeichen
- Punkt .
- Stern *
- öffnende eckige Klammer [
- und Tilde ~

haben nur dann eine Sonderbedeutung, wenn sie mit einem führenden Schrägstrich \ entwertet worden sind.

Ersetzungszeichenketten

Das kommerzielle Und & (Gegenschrägstrich kommerzielles Und \& bei *nomagic*) steht für die Zeichenkette, zu der das Muster paßt und die daher ersetzt werden soll.

Das Zeichen Tilde ~ (\~ bei *nomagic*) wird durch die Ersetzungszeichenkette des letzten *s*-Kommandos (substitute) ersetzt.

Die Zeichenkette *n* (*n* ist eine ganze Zahl) wird durch die Zeichenkette ersetzt, zu der das im *n*-ten Klammernpaar \(...\) enthaltene Muster paßt.

Ist in einer Ersetzungszeichenkette die Zeichenkette *u* oder *l* enthalten, so wird das erste Zeichen in der Ersetzungszeichenkette, das unmittelbar auf *u* (upper) oder *l* (lower) folgt, in einen Großbuchstaben (bei *u*) bzw. Kleinbuchstaben (bei *l*) umgewandelt, falls es sich bei diesem Zeichen um einen Buchstaben handelt. Mit den Zeichen *U* oder *L* werden alle Buchstaben bis zum Ende der Ersetzungszeichenkette bzw. bis zu den Zeichen *e* oder *E* in Groß- bzw. Kleinbuchstaben umgewandelt.

Puffer

Es gibt einen unbenannten und 26 alphabetische Puffer.

In diese Puffer kann Text kopiert werden (*ya* - yank) oder wird gelöschter Text gesichert (*d* - delete), der dann mit *pu* (put) zurückgeholt werden kann.

Die Kommandos *d*, *pu* und *ya* arbeiten mit dem unbenannten Puffer, wenn Sie keinen alphabetischen Puffer angeben. D.h. auch wenn Sie z.B. bei einer Löschoperation keinen Puffer angeben, wird das Ergebnis Ihrer Löschoperation im unbenannten Puffer gesichert.

Sie können in 26 alphabetischen Puffern Textblöcke sichern. Die Puffer werden mit den Kleinbuchstaben a-z oder mit den Großbuchstaben A-Z bezeichnet. Verwenden Sie Kleinbuchstaben, wird der alte Pufferinhalt mit dem neuen überschrieben, der alte somit gelöscht. Verwenden Sie Großbuchstaben, wird der alte Pufferinhalt nicht überschrieben, sondern der neue Text an den alten angehängt.

Fehler- und Signalbehandlung

Tritt während einer *ex*-Sitzung ein Fehler auf, so sendet der *ex* an die Datensichtstation das Zeichen BEL (akustisches Signal; siehe *Tabellen und Verzeichnisse, Zeichensatz ISO 646*) und gibt eine Fehlermeldung aus.

Bei einem Unterbrechungssignal kehrt *ex* zusätzlich in den *ex*-Kommando-Modus zurück. Sie können nun ein *ex*-Kommando eingeben.
Liest *ex* die Eingabe aus einer Datei, so wird *ex* verlassen.

EX-KOMMANDOS

Im Unterschied zu den meisten *vi*-Kommandos, die sofort vom *vi* interpretiert und ausgeführt werden, müssen *ex*-Kommandos mit ⏎ abgeschlossen werden.

Kommandozeilen, die mit Anführungszeichen " beginnen, werden ignoriert. Auf diese Weise können Sie in ein Kommandoskript Kommentarzeilen einfügen.

Adressen

Mit einer Adresse geben Sie eine bestimmte Zeile an. Einige *ex*-Kommandos erwarten eine oder mehrere Adressen, um dann zum Beispiel die angegebene Zeile oder den Bereich zwischen zwei angegeben Zeilen einschließlich zu bearbeiten.

Für *ex* existiert zu jedem Zeitpunkt eine aktuelle Zeile. Sie wird durch einen Punkt . adressiert. Die aktuelle Zeile ist in der Regel die Zeile, die zuletzt durch ein Kommando bearbeitet wurde oder auf die gezielt (z.B. durch Suchen) positioniert wurde.

Die Adressen trennen Sie voneinander durch ein Komma , oder einen Strichpunkt ; . Diese Adressenliste arbeitet *ex* von links nach rechts ab.

- Durch das Trennzeichen Strichpunkt ; wird die zuerst adressierte Zeile zur aktuellen Zeile, erst dann wird die nächste Adresse ausgewertet.
- Beim Trennzeichen Komma , werden alle Adressen von der aktuellen Zeile aus ermittelt. Die aktuelle Zeile ändert sich erst bei der Durchführung eines Kommandos.

Wenn Sie bei einem Kommando mehr Adressen angegeben haben, als das Kommando erwartet, werden alle Adressen außer der letzten (wenn das Kommando eine Zeile als Adresse erwartet) oder den letzten beiden (wenn das Kommando einen Zeilenbereich erwartet) ignoriert. Sind bei einem Kommando zwei Adressen erforderlich, so muß sich die zuerst adressierte Zeile im Puffer vor der danach adressierten Zeile befinden. Wenn Sie keine Adresse angeben, verwendet *ex* standardmäßig die aktuelle Zeile.

adresse

.
  aktuelle Zeile des Puffers.

$
  letzte Zeile des Puffers.

n
  n-te Zeile im Puffer. Die Zeilen sind sequentiell von 1 ab durchnumeriert.

'x
die mit dem Buchstaben *x* markierte Zeile. *x* muß ein Kleinbuchstabe sein (siehe *ex*-Kommandos *ma* bzw. *k*). *'x* ist die Adresse der markierten Zeile. Bevor *ex* ein Positionierkommando durchführt, wird die momentan aktuelle Zeile markiert. Mit 2 Hochkommata " können Sie zu ihr zurückkehren.

/rA/
Ein einfacher regulärer Ausdruck in /.../ eingeschlossen (siehe *Tabellen und Verzeichnisse, Reguläre Ausdrücke*) adressiert die erste Zeile, beginnend mit der aktuellen Zeile, die eine Zeichenkette enthält, die zu dem regulären Ausdruck paßt. Falls notwendig, springt *ex* vom Ende des Puffers an seinen Anfang, um die Suche fortzusetzen (siehe *ex*-Option, *wrapscan*). Soll die zu *rA* passende Zeile nur ausgegeben werden, können Sie den zweiten Schrägstrich / weglassen.

rA nicht angegeben:
das zuletzt angegebene Suchmuster wird verwendet.

?rA?
Wie */rA/*, nur wird rückwärts gesucht.

+n
-n
Beginnt eine Adresse mit Plus- oder Minuszeichen, gefolgt von einer Dezimalzahl *n* (optional), ist die Zeile adressiert, die *n* Zeilen hinter (+) bzw. vor (-) in der aktuellen Zeile liegt. .+3, +3 und + + + haben die gleiche Wirkung.

adr+
adr-
Endet eine Adresse mit einem Plus oder Minus, ist die Zeile adressiert, die eine Zeile hinter (+) bzw. vor (-) in der durch die Adresse bezeichneten Zeile liegt. Mit - wird die Zeile vor der aktuellen Zeile adressiert. Auch am Ende einer Adresse haben + und - kumulativen Effekt. Die Adresse + + adressiert somit die zweite Zeile, d.h. die Zeile, die auf die aktuelle Zeile folgt; z.B. adressiert 3+ + Zeile 5 oder + + ist gleichbedeutend mit .+2.

%
Das Prozent-Zeichen % ist gleichbedeutend mit dem Adressenpaar 1,$, d.h. dem gesamten Puffer.

Parameter

Bei den *ex*-Kommandos werden folgende Parameter benutzt:

zeile = *adresse*
eine bestimmte einzelne Zeile, die Sie in einem der im Abschnitt Adressen angegebenen Formate angeben können.

zeile nicht angegeben:
der Standard-Wert Punkt . gilt. Punkt . ist die aktuelle Zeile.

bereich = *adresse,adresse* bereich = *adresse;adresse*
Angabe eines Bereiches zwischen zwei Zeilen einschließlich. Die Adressen können durch ein Komma , oder einen Strichpunkt ; (siehe oben) voneinander getrennt sein. *bereich* sollten Sie nicht zusammen mit einer Zahl *n* angeben, da sonst die *letzte* Adresse des angegebenen Bereiches zur *Anfangs*adresse eines Bereiches wird, der einschließlich dieser Zeile *n* Zeilen umfaßt. Damit ist auf jeden Fall ein Bereich adressiert, der hinter dem gemeinten *bereich* liegt. In der Syntax der *ex*-Kommandos ist dies berücksichtigt.

bereich nicht angegeben:
Der Standard-Wert .,. gilt.
.,. (nur die aktuelle Zeile)

n
eine positive ganze Zahl. Mit *n* geben Sie die Anzahl der zu bearbeitenden Zeilen an.

n nicht angegeben:
der Standard-Wert 1 gilt.

zusatz
eines oder eine Kombination der *ex*-Kommandos # (siehe *nu*), *p* und *l*. Diese Kommandos geben eine Zeile in einem bestimmten Format aus und werden im Anschluß an das vorangestellte *ex*-Kommandos ausgeführt.

␣
Die Leerzeichen müssen nicht in jedem Fall eingegeben werden. Hier sind sie aus Gründen der Übersichtlichkeit nicht als optional angegeben.

Diese Parameter können mit einer beliebigen Zahl von Plus- oder Minuszeichen kombiniert werden.

Kommandos

Übersicht der ex-Kommandos und ihrer Abkürzungen

CTRL	(scroll)	n Zeilen ausgeben, wobei n=$scroll
ab	abbrev	Abkürzung definieren
a	append	Text-Zeile anfügen
ar	args	Argumentliste der Kommandozeile ausgeben
c	change	Text-Zeile ändern
C	encryption	Text verschlüsseln
co	copy	Text-Bereich kopieren
d	delete	Text-Bereich löschen
e	edit	editierte Datei wechseln
f	file	aktuellen Dateinamen ausgeben
g	global	für alle Zeilen, die zu /rA/ passen, Kommando ausführen
i	insert	Text-Zeile einfügen
j	join	Zeilen verbinden
l	list	Text mit nicht-druckbaren Zeilen ausgeben
map	---	Makros definieren
k	mark	Zeilen markieren
m	move	Text-Bereich verschieben
ma	mark	Zeilen markieren
n	next	nächste Datei editieren
nu	number	Zeilen numerieren und ausgeben
#	number	Zeilen numerieren und ausgeben
pre	preserve	Editor-Puffer retten
p	print	Text-Bereich ausgeben
pu	put	Puffer ausgeben
q	quit	Editor verlassen
r	read	Kopie einer Datei in Puffer einlesen
rec	recover	Datei nach Systemabsturz wiederherstellen
rew	rewind	Argumentliste rücksetzen (vgl. *ar*)
se	set	Optionen ausgeben und setzen
sh	shell	Shell aufrufen
so	source	Kommandos aus Datei lesen
s	substitute	suchen und ersetzen
unab	unabbrev	Definition einer Abkürzung mit *ab* aufheben
u	undo	Kommando rückgängig machen
unm	unmap	Makrodefinition mit *map* aufheben
v	---	wie *g*, aber für alle Zeilen, die nicht zu /rA/ passen
ve	version	Versionsnummer des Editors ausgeben
vi	visual	in *vi* wechseln
w	write	Editor-Puffer in Datei schreiben
x	exit	Editor verlassen mit Sichern des Editor-Puffers
X	encryption	Text verschlüsselen
ya	yank	Zeilen in Puffer kopieren
z	(window)	Text-Bereich ausgeben
!	(escape)	Shell-Kommando ausführen
& s	(resubst)	wiederholen des letzten *substitute*-Kommandos
<	(lshift)	Text-Bereich nach links verschieben
>	(rshift)	Text-Bereich nach rechts verschieben
=	(line no)	Zeilennummer ausgeben

Kein Kommando angegeben
Wenn Sie nur *zeile* oder *bereich* angeben, so wird automatisch *p* (print) ausgeführt. Eine leere Eingabe bewirkt, daß die darauffolgende Zeile ausgegeben wird (wie bei .+1p).

CTRL D
(ASCII EOT) gibt die nächsten *n* Zeilen aus, wobei *n* den Wert der *ex*-Option *scroll* hat.

ab_wort_text
(ab - abbreviate) Nur im *vi* wirksam!
Wenn Sie im *vi*-Eingabe-Modus *wort* als komplettes Wort (d.h. nicht nur als Teil eines Wortes) eingeben, wird es durch die Zeichenkette *text* ersetzt. Falls *text* Sonderzeichen, z.B. ↵, enthalten soll, müssen Sie dem Sonderzeichen CTRL V voranstellen.

zeile_**a**
(a - append) *ex* wechselt in den Eingabe-Modus; der Text wird hinter der angegebenen Zeile eingefügt. Bei *zeile* gleich 0 wird der Text ganz an den Anfang des Puffers gesetzt. Die aktuelle Zeile ist die zuletzt eingegebene bzw., falls keine Eingabe erfolgte, die adressierte Zeile.
Die Eingabe wird mit einem Punkt . in der ersten Spalte beendet.

ar
(ar - arguments)
Die Argumentliste aus der Kommandozeile des *ex*-Aufrufes wird, eingeschlossen in eckigen Klammern [...], ausgegeben.

zeile_**c**_n
bereich_**c**
(c - change) *ex* wechselt in den Eingabe-Modus; es werden *n* Zeilen bzw. die Zeilen im angegebenen Bereich *bereich* durch die eingegebenen Textzeilen ersetzt.
Die letzte der eingegebenen Zeilen wird zur aktuellen Zeile; werden keine Textzeilen eingegeben, so wird der angegebene Bereich gelöscht, d.h. *c* wirkt wie *d* (delete).
Die Eingabe wird mit einem Punkt . in der ersten Spalte beendet.

C
(C - encryption) Das Kommando *C* entspricht *X* mit einem Unterschied: Es wird angenommen, daß jeder mit den Kommandos *e* und *r* zu lesende Text verschlüsselt ist (siehe Option *-C*).

bereich_**co**_zeile_zusatz
(co - copy) Eine Kopie des angegebenen Bereiches *bereich* wird hinter die Zeile *zeile* kopiert; hat *zeile* die Zeilennummer 0, wird *bereich* an den Anfang des Puffers kopiert.

zeile␣**d**␣puffer␣n
bereich␣**d**␣puffer
(d - delete) Die Zeilen im Bereich *bereich* bzw. *n* Zeilen von der angegebenen Zeile *zeile* ab werden gelöscht. Wird der Name eines Puffers *puffer* angegeben, so werden die gelöschten Zeilen in diesem Puffer gesichert. Mit dem *ex*-Kommando *pu* können Sie auf diesen Puffer zugreifen. Die erste auf die gelöschten Zeilen folgende Zeile wird zur aktuellen Zeile; befanden sich die gelöschten Zeilen am Ende des Puffers, so wird die letzte Zeile im Puffer zur aktuellen Zeile.

e[␣+zeile]␣datei
(e - edit) Die Datei *datei* wird editiert. Wurden am aktuellen Editor-Puffer seit dem letzten *w*-Kommando (write) Änderungen vorgenommen, so wird eine Warnung ausgegeben und das Kommando nicht ausgeführt. Sie können jedoch die Ausführung des *e*-Kommandos erzwingen, indem Sie dem *e* ein Ausrufezeichen nachstellen: *e! datei*. Die letzte Zeile des Puffers wird zur aktuellen Zeile. Wird dieses Kommando jedoch von *vi* aus aufgerufen, so wird die erste Zeile im Puffer zur aktuellen Zeile.

+zeile
Die angegebene *zeile* wird zur aktuellen Zeile. *zeile* kann sein:

- eine Zeilennummer *n*
- das Dollarzeichen $ (Dateiende)
- ein regulärer Ausdruck in der Form */rA* oder *?rA*.

f[␣name]
(f - file) Ausgabe des aktuellen Dateinamens sowie Informationen, wie z.B. die Zahl der Zeilen und die Position der aktuellen Zeile,

name angegeben:
name wird zum aktuellen Dateinamen

bereich␣**g**/rA/kommandoliste
(g - global) Zunächst werden alle Zeilen innerhalb des Bereiches *bereich* markiert, zu denen das durch *rA* angegebene Muster paßt. Dann wird das angegebene Kommando *kommando* bzw. werden die in *kommandoliste* enthaltenen Kommandos ausgeführt, wobei die jeweils markierte Zeile zur aktuellen Zeile wird.

Die Kommandos in *kommandoliste* können in mehreren Zeilen stehen, vorausgesetzt, die Neue-Zeile-Zeichen sind mit einem Gegenschrägstrich \ entwertet. Ist *kommandoliste* leer, so werden alle Zeilen ausgegeben. Die Kommandos *a(ppend)*, *c(hange)*, und *i(nsert)* sind erlaubt; am Ende von *kommandoliste* können Sie den abschließenden Punkt weglassen. Das *visual*-Kommando ist ebenfalls erlaubt und liest die Eingabe von der Datensichtstation.

Folgende Kommandos dürfen in *kommandoliste* nicht enthalten sein: *global* und *undo*.

Folgende Optionen dürfen in *kommandoliste* nicht enthalten sein: *autoprint*, *autoindent* und *report*.

Beispiel

```
1,$g/^\*/s//+/
```

Von Zeile 1 bis Dateiende ($) werden in der ersten Spalte ^ alle * durch das Substitute-Kommando (*s*) durch ein Plus-Zeichen (+) ersetzt.

zeile␣**i**

(i - insert) *ex* wechselt in den Eingabe-Modus; der eingegebene Text wird vor der angegebenen Zeile eingefügt. Die letzte der eingegebenen Zeilen wird zur aktuellen Zeile. Erfolgt keine Eingabe, so wird die Zeile vor der adressierten Zeile *zeile* zur aktuellen Zeile. Die Eingabe wird mit einem Punkt . in der ersten Spalte beendet.

zeile␣**j**␣n␣zusatz
bereich␣**j**␣zusatz

(j - join) Faßt die mit *bereich* angegebenen Zeilen bzw. *n* Zeilen von *zeile* ab zu einer Zeile zusammen. Zwischen den ehemaligen Zeilen steht mindestens ein Leerzeichen. Wird eine Zeile durch einen Punkt abgeschlossen, so werden zwei Leerzeichen eingesetzt. Beginnt die anzufügende Zeile mit einer schließenden runden Klammer), wird kein Leerzeichen eingefügt. Zusätzliche Leer- und Tabulatorzeichen am Anfang einer Zeile entfallen.

Wird an das Kommando *j* ein Ausrufezeichen ! angehängt, werden an die Nahtstellen zwischen zwei verbundenen Zeilen keine Leerzeichen gesetzt.

zeile␣**l**␣n␣zusatz
bereich␣**l**␣zusatz

(l - list) Die angegebenen Zeilen werden ausgegeben; Tabulatorzeichen werden durch ^I ersetzt; das Ende jeder Zeile wird durch ein Dollarzeichen $ markiert. Der einzige sinnvolle *zusatz* ist hier #, wenn den Zeilen ihre Zeilennummern vorangestellt werden sollen. Die letzte der ausgegebenen Zeilen wird zur aktuellen Zeile.

map␣x␣kommandos — Format 1
map!␣x␣text — Format 2

Nur im *vi* wirksam!

Format 1: Makrodefinition für *vi*-Kommando-Modus

Definieren von Makros, die im *vi*-Kommando-Modus verwendet werden können. Beim ersten Argument *x* handelt es sich um ein einzelnes Zeichen oder die Zeichenkette *#n*, wobei *n* eine Ziffer ist, mit der die Funktionstaste *n* angegeben wird. Wird im *vi*-Kommando-Modus dieses Zeichen bzw. diese Funktionstaste eingegeben, so verhält *vi* sich so, als ob *kommandos* eingegeben wird. *kommandos* wird dabei als Folge von *vi*-Kommandos interpretiert. Sind in *kommandos* Sonderzeichen, Leerzeichen oder Neue Zeile-Zeichen enthalten, so müssen sie mit einem CTRL V entwertet werden.

Beispiel 1

Sie wollen ein Makro definieren, das in der gesamten Datei das Zeichen Stern * am Zeilenanfang (Spalte 1) durch ein Leerzeichen ␣ ersetzt. Der Name dieses Makros soll "Stern" * sein. Damit Sie dieses Makro stets zur Verfügung haben, machen Sie in Ihrer *.exrc*-Datei (siehe *Voreinstellung des ex*) folgenden Eintrag:

```
:map * :1,$s/^\*/␣/
```

Beispiel 2

Sie möchten die Funktionstaste F1 belegen. Es soll die nächste Zeile gesucht werden, an deren Anfang ein bestimmter regulärer Ausdruck *rA* steht; diese Zeile soll dann automatisch gelöscht werden. Dazu geben Sie (aus dem *vi*-Kommando-Modus) ein:

:map CTRL V F1 /^rA CTRL V ↵ dd ↵

Format 2: Makrodefinition für *vi*-Eingabe-Modus

Definieren von Makros, die im *vi*-Eingabe-Modus verwendet werden können (vgl. Format 1). *Jedes* im *vi*-Eingabe-Modus eingegebene *x* wird durch *text* ersetzt. (Bei dem Kommando *ab* wird *x* nur dann ersetzt, wenn es allein steht.)

Wenn *x* nicht durch *text* ersetzt werden soll, muß *x* ein CTRL V vorangestellt werden.

Format 1 und Format 2:

Ein definiertes Makro können Sie mit:

`unmap x` bzw.

`unmap! x`

wieder aufheben.

zeile␣**ma**␣x
zeile␣**k**␣x
(ma/k - mark) *zeile* wird mit *x* markiert; *x* muß ein einzelner Kleinbuchstabe sein, dem ein Leer- oder Tabulatorzeichen vorangestellt ist. An der Adresse der aktuellen Zeile ändert sich nichts.

bereich␣**m**␣zeile
(m - move) *bereich* wird hinter *zeile* versetzt. Die erste der versetzten Zeilen wird zur aktuellen Zeile.

n[␣datei]...

datei nicht angegeben:
(n - next) Die nächste der in der Kommandozeile aufgeführten Dateien (Argumentliste) wird editiert (vgl. *f* und *ar*). Wurden am aktuellen Editor-Puffer seit dem letzten *w*-Kommando (write) Änderungen vorgenommen, so wird eine Warnung ausgegeben und das Kommando nicht ausgeführt. Sie können jedoch die Ausführung des *n*-Kommandos erzwingen, indem Sie dem *n* ein Ausrufezeichen nachstellen: *n!*.

datei angegeben:
Die bisherige Argumentenliste wird durch die angegebene(n) Datei(en) ersetzt; die erste angegebene Datei wird editiert.

zeile␣**nu**␣n␣zusatz
bereich␣**nu**␣zusatz
zeile␣**#**␣n␣zusatz
bereich␣**#**␣zusatz
(nu - number) Die Zeilen werden, versehen mit den entsprechenden Zeilennummern, ausgegeben. Der einzige hier sinnvolle Zusatz ist *l*. Die letzte der ausgegebenen Zeilen wird zur aktuellen Zeile.

pre
(pre - preserve) Der Inhalt des aktuellen Puffers wird gesichert, als ob es zu einem Absturz des Systems gekommen wäre. *pre* wird in Notfällen verwendet, wenn z.B. *w* (write) nicht ausgeführt werden kann, weil die Platte voll ist, und keine andere Möglichkeit zur Rettung des Pufferinhaltes vorhanden ist.

zeile␣**p**␣n
bereich␣**p**
(p - print) Die angegebenen Zeilen werden ausgegeben, wobei nicht-druckbare Zeichen als Steuerzeichen in der Form ^x ausgegeben werden; DEL wird in Form von ^? ausgegeben. Die zuletzt ausgegebene Zeile wird zur aktuellen Zeile.

zeile␣**pu**␣puffer
(pu - put) Die mit *d* (delete) gelöschten oder mit *y* (yank) in einem Puffer gesicherten Zeilen werden hinter der angegebenen Zeile *zeile* eingefügt. *puffer* ist der Name eines alphabetischen Puffers.

puffer nicht angegeben:
Der Text wird aus dem unbenannten Puffer geholt.

q
(q - quit) Der Editor wird verlassen. Wurde der Puffer seit dem letzten *w*-Kommando (write) verändert, so wird eine Warnung ausgegeben und *q* wird nicht ausgeführt. Sie können jedoch die Ausführung des *q*-Kommandos erzwingen, indem Sie dem *q* ein Ausrufezeichen nachstellen: *q!*. Alle noch nicht (mit *w*) gesicherten Änderungen am Editor-Puffer gehen dann verloren.

zeile␣**r**[␣datei]
(r - read) Eine Kopie von *datei* wird im Editor-Puffer hinter *zeile* eingelesen. Die Zeilennummer von *zeile* kann dabei 0 sein; die Textzeilen werden dann an den Anfang des Puffers gesetzt. Gibt es keine aktuelle Datei (Aufruf von *ex* ohne Dateinamen) wird *datei* zur aktuellen Datei. Die letzte der eingelesenen Zeilen wird zur aktuellen Zeile; im *vi* wird die erste der eingelesenen Zeilen zur aktuellen Zeile.

datei nicht angegeben:
die aktuelle Datei wird eingelesen.

Wird statt *datei* *!sinixkmdo* angegeben, so wird *sinixkmdo* als SINIX-Kommando interpretiert und an die Shell übergeben. Die Ausgabe des SINIX-Kommandos wird in den Puffer eingelesen.

rec␣datei
datei wird nach einem Absturz des Systems oder des Editors wiederhergestellt.

rew
(rew - rewind) Die Argumentliste (siehe *ar*) wird zurückgesetzt, und die erste Datei in der Liste wird editiert. Wurde der Editor-Puffer seit dem letzten *w*-Kommando (write) verändert, so wird eine Warnung ausgegeben und *rew* wird nicht ausgeführt. Sie können jedoch die Ausführung des *rew*-Kommandos erzwingen, indem Sie dem *rew* ein Ausrufezeichen nachstellen: *rew!*. Alle noch nicht (mit *w*) gesicherten Änderungen am Editor-Puffer gehen dann verloren.

se[␣parameter]
(se - set) Mit dem *set*-Kommando können Optionen aufgelistet bzw. gesetzt werden. Es gibt zwei Typen von Optionen: Optionen mit Boole'schen Werten und Optionen mit nicht-Boole'schen Werten. Ein Beispiel für den Boole'schen Typ ist z.B. die Option *number*. Ist sie gesetzt, werden Zeilennummern ausgegeben, ist *nonumber* gesetzt, werden keine Zeilennummern ausgegeben. Ein Beispiel für den nicht-Boole'schen Typ ist *report*. Der Wert dieser Option (Standard = 5) bestimmt die Anzahl der Zeilen, die durch ein Kommando verändert werden muß, damit *ex* und *vi* eine Meldung geben.

parameter

all
die Werte aller Optionen werden ausgegeben.

option**?**
der gegenwärtige Wert der angegebenen Option wird ausgegeben.

option
Nur für Optionen mit Boole'schen Werten: die Option wird gesetzt.

nooption
Nur für Optionen mit Boole'schen Werten: die Option wird nicht gesetzt.

option=wert
Nur für Optionen mit nicht-Boole'schen Werten: der Option wird der Wert *wert* zugewiesen.

parameter nicht angegeben:
set gibt diejenigen *ex*-Optionen aus, deren Werte vom Benutzer festgelegt wurden und daher von den Standard-Werten abweichen.

Nähere Informationen über die *ex*-Optionen finden Sie in *ex-Optionen*.

sh
Ist die Variable *SHELL* gesetzt, wird die dort angegebene Shell gestartet, sonst *sh*. Nach Beendigung der Shell wird die Editorsitzung an derselben Stelle fortgesetzt.

so␣datei
Kommandos werden aus *datei* gelesen und ausgeführt. *so*-Kommandos können ineinander verschachtelt werden.

zeile␣**s**/rA/ersatz/schalter␣n␣zusatz
bereich␣**s**/rA/ersatz/schalter␣zusatz
(s - substitute) Das erste Auftreten des mit *rA* (regulärer Ausdruck) getroffenen Musters in jeder Zeile des angegebenen Bereiches wird durch die Zeichenkette *ersatz* ersetzt (siehe *Reguläre Ausdrücke* und *Ersetzungszeichenketten*). Der angegebene Bereich ist entweder *bereich* oder umfaßt *n* Zeilen von *zeile* ab.

schalter

g
(g - global) *alle* Zeichenketten in der Zeile, zu denen *rA* paßt werden ersetzt.

c
(c - confirm) zunächst wird die Zeile ausgegeben, wobei die zu ersetzende Zeichenkette in der darunterliegenden Zeile durch ein ^ markiert wird. Die Eingabe des Buchstabens *y* bewirkt dann, daß die Substitution durchgeführt wird; bei jeder anderen Eingabe wird das Kommando abgebrochen. Die Zeile, in der zuletzt eine Substitution durchgeführt wurde, wird zur aktuellen Zeile.

una␣wort
(una - unabbreviate) *wort* wird aus der Liste der Abkürzungen gestrichen (vgl. *ab*).

u
(u - undo) Die mit dem letzten Editierkommando vorgenommenen Änderungen werden rückgängig gemacht. *g* (global) und *vi* (visual) werden in diesem Fall als eigenständige Kommandos betrachtet. Kommandos, die die äußere Umgebung verändern (z.B. *w* (write), *e* (edit) und *n* (next)) können nicht rückgängig gemacht werden.

Ein *undo*-Kommando selbst kann mit *u* wieder rückgängig gemacht werden.

Bei *u* gehen alle Markierungen für Zeilen, die geändert und dann wieder zurückgeholt wurden, verloren.

unm␣x
unm!␣x
(unm - unmap) Die mit *map* vorgenommene Definition des Makros *x* wird aufgehoben.

bereich␣**v**/rA/kommandoliste
(v - vice versa) Hat dieselbe Funktion wie *global*, nur wird *kommandoliste* auf alle Zeilen angewandt, zu denen das Muster *rA* nicht paßt.

ve
Die Versionsnummer des Editors wird ausgegeben.

zeile␣**vi**[␣stelle]␣n
Bei der angegebenen *zeile* wird in den *vi* gewechselt. Der optionale Parameter *stelle* kann eines der Zeichen Bindestrich - oder Punkt . sein; wie beim Kommando *z* wird damit die Position von *zeile* auf dem Bildschirm angegeben. Standard: oberer Rand des Bildschirms. Mit *n* wird die Größe des Bildschirms angegeben; standardmäßig hat er die Größe, die über die *ex*-Option *window* definiert ist. Mit *Q* wird der *vi* wieder verlassen und wieder in den *ex* gewechselt. Nähere Information siehe *vi*.

[bereich␣]**w**[**!**][␣datei] Format 1
[bereich␣]**wq**[**!**][␣datei] Format 2

Format 1: In Datei schreiben

(w - write) Der angegebene *bereich* wird in die *datei* geschrieben, wobei die Anzahl der geschriebenen Zeilen und Zeichen ausgegeben wird.

Wird eine (vorhandene) *sekundäre* Datei angegeben, so wird das Kommando nicht ausgeführt, damit die sekundäre Datei nicht versehentlich überschrieben wird. Die Ausführung von *w* kann jedoch erzwungen werden, indem ein Ausrufezeichen ! angefügt wird: *w! #*.

Soll am Ende von *datei* angefügt werden, so muß das Kommando in der Form *w> >* angegeben werden; existiert *datei* nicht, so wird eine Fehlermeldung ausgegeben.

Wird statt *datei* !*sinixkmdo* angegeben, so wird *sinixkmdo* als SINIX-Kommando interpretiert und an die Shell übergeben; der angegebene *bereich* ist dann für das Kommando die Standard-Eingabe.

bereich nicht angegeben:
die ganze aktuelle Datei wird nach *datei* geschrieben.

datei nicht angegeben:
standardmäßig wird die aktuelle Datei verwendet. Wenn weder eine aktuelle Datei vorhanden ist noch eine *datei* angegeben wird, so kann *w* nicht ausgeführt werden.

Format 2: In Datei schreiben und *ex* verlassen

(wq - write and quit) *wq* hat dieselbe Funktion wie *w*, auf das *q* folgt; *wq!* hat dieselbe Funktion wie *w!*, auf das *q* folgt.

x

(exit) Der Editor wird verlassen; wurden seit dem letzten Sichern mit *w* (write) Änderungen vorgenommen, so werden diese zuvor in die Datei geschrieben.

X

(X - encryption) *ex* fragt Sie nach einem Schlüssel, der bei nachfolgenden *e*, *r* und *w*-Kommandos zum Ent- und Verschlüsseln von Text verwendet wird. Dabei wird der Algorithmus des Kommandos *crypt* verwendet. Ein leerer Schlüssel schaltet diesen Mechanismus wieder ab. *ex* klärt ab, ob der einzulesende Text verschlüsselt ist oder nicht. Die temporäre Pufferdatei wird ebenfalls mit einer Transformation des eingegebenen Schlüssels verschlüsselt (siehe Option *-x*).

zeile␣**ya**␣puffer␣n
bereich␣**ya**␣puffer
Der angegebene *bereich* bzw. *n* Zeilen von *zeile* ab werden in den angegebenen *puffer* gesichert.

puffer nicht angegeben:
Wird kein Puffer angegeben, so wird der unbenannte Puffer verwendet.

zeile␣**z**[␣stelle]␣n
(window) *n* Zeilen aus der Umgebung von *zeile* werden ausgegeben.

n nicht angegeben:
n hat den Wert der Variablen *window*.

stelle
Für den optionalen Parameter *stelle* kann ein Bindestrich - oder Punkt . angegeben werden; das Zeichen - bewirkt, daß die *zeile* an den unteren Rand des ausgegebenen Bereiches gesetzt wird; das Zeichen . bewirkt, daß sich die Zeile in der Mitte des ausgegebenen Bereiches befindet. Die letzte der ausgegebenen Zeilen wird zur aktuellen Zeile.

stelle nicht angegeben:
zeile befindet sich am oberen Rand des ausgegebenen Bereiches.

Vorsicht: *ex* gibt die gewünschte Anzahl realer Zeilen aus. Sind Zeilen länger, wie der Bildschirm breit ist, dann kann unter Umständen mehr als ein Bildschirm ausgegeben werden.

!␣kommando
(escape) Die auf das Ausrufezeichen ! folgenden Zeichen werden als Shell-Kommando interpretiert und an den Kommandointerpreter übergeben. Wurden am Puffer seit dem letzten *w*-Kommando (write) Änderungen vorgenommen, so wird eine Warnung ausgegeben. Dann wird das Kommando ausgeführt. Wenn die *Ausgabe des Kommandos* nicht umgelenkt wird, erscheint sie auf dem Bildschirm, ändert aber an der Datei nichts. Allerdings kann das *Kommando selbst* die Datei ändern, z.B. *!cat /etc/profile >> %*. Nach erfolgreicher Ausführung des Kommandos wird ein ! ausgegeben. An der Adresse der aktuellen Zeile ändert sich nichts.

Kommen in *kommando* das Prozentzeichen % und das Nummernzeichen # vor, so werden sie als Dateinamen expandiert (siehe *Aktuelle und sekundäre Datei*).

Ein Ausrufezeichen ! in *kommando* wird durch das vorherige !-Kommando ersetzt. Mit !! wird also das letzte !-Kommando wiederholt. Die so erweiterten Kommandozeilen werden auf dem Bildschirm ausgegeben.

bereich␣!␣kommando
(escape) Bei einem !-Kommando in dieser Form werden die mit *bereich* angegebenen Zeilen an *kommando* als Standard-Eingabe übergeben und durch die Ausgabe von *kommando* ersetzt.

bereich nicht angegeben:
bereich wird nicht durch .,. ersetzt, sondern es gilt die andere Form des !-Kommandos.

"
Kommandozeilen, die mit Anführungszeichen " beginnen, werden ignoriert. Auf diese Weise können Sie in ein Kommandoskript Kommentarzeilen einfügen.

zeile␣&schalter␣n␣zusatz
bereich&schalter␣zusatz
zeile␣sschalter␣n␣zusatz
bereichsschalter␣zusatz
(resubst) Das letzte *s*-Kommando (substitute)
wird wiederholt, als ob *&* durch das letzte *s/rA/ersatz/* ersetzt würde. Dieselbe Funktion hat ein *s*-Kommando, in dem */rA/ersatz/* weggelassen wird.

zeile␣<␣n
bereich␣<
(lshift) Die Zeilen innerhalb des angegebenen Bereichs bzw. *n* Zeilen, die auf *zeile* folgen, werden um *shiftwidth* Positionen nach links verschoben. Leer- und Tabulatorzeichen gehen dabei verloren; die übrigen Zeichen bleiben unverändert erhalten. Die letzte der geänderten Zeilen wird zur aktuellen Zeile.

zeile␣>␣n
bereich␣>
(rshift) Die Zeilen innerhalb des angegebenen Bereichs bzw. *n* Zeilen, die auf *zeile* folgen, werden durch Einfügen von Leer- und Tabulatorzeichen um *shiftwidth* Positionen nach rechts verschoben.

zeile=
(line number) Die Zeilen-Nummer der angegebenen *zeile* wird ausgegeben. Die Position der aktuellen Zeile wird nicht verändert.

zeile nicht angegebnen:
Die Zeilen-Nummer der letzten Zeile wird ausgegeben.

EX-OPTIONEN

Mit den Optionen des *ex* kann das Verhalten der Editoren *ex* und (vor allem) *vi* beeinflußt werden (siehe *vi*, *Voreinstellung des vi*).

Für alle Optionen gibt es Standard-Einstellungen, die beim Aufruf von *ex* oder *vi* wirksam werden. Mit dem *ex*-Kommando *se* (set) können Sie sich alle Optionen ausgeben lassen:

```
set all↵
```

Sie können aber auch mit diesem Kommando eine oder mehrere Optionen nach Ihren Vorstellungen verändern:

```
set showmode scroll=15↵
```

In diesem Fall wird im *vi* der Eingabe-Modus in der Status-Zeile angezeigt und die Anzahl der Zeilen, um die CTRL *D* den Bildschirm rollt, auf *15* gesetzt.

Mit

```
set↵
```

werden Ihnen alle Optionen ausgegeben, die von den Standard-Werten abweichen.

Es gibt zwei Typen von Optionen: Optionen mit Boole'schen Werten und Optionen mit nicht-Boole'schen Werten. Ein Beispiel für den Boole'schen Typ ist z.B. die Option *showmode*. Ist diese Option nicht gesetzt, hat sie den Namen *noshowmode*. Ein Beispiel für den nicht-Boole'schen Typ ist *scroll*.

autoindent, ai
noautoindent, noai (Standard)

Ist die Option *autoindent* gesetzt, so wird im Eingabe-Modus jede Zeile entsprechend der vorhergehenden Zeile eingerückt (es werden Leer- und Tabulatorzeichen eingefügt). Der Beginn der Einrückung wird bestimmt von der Zeile, hinter die angefügt (*a*,) vor die eingefügt (*i*) oder die als erste geändert (*c*) wird. Zusätzliche Einrükkungen können wie gewöhnlich durchgeführt werden, die folgenden Zeilen werden dann automatisch auf die neue Grenze eingerückt.

Soll der Text weniger eingerückt werden, so können sie durch Eingabe von CTRL D den Zeilenanfang um *shiftwidth* Positionen nach links verschieben. Mit einem Dach ^, gefolgt von einem CTRL *D*, wird die Einrückung für die aktuelle Zeile aufgehoben; mit einer Null *0*, gefolgt von einem CTRL D werden alle Einrückungen aufgehoben.

autoprint, ap
noautoprint, noap (Standard)
Nach jedem Kommando, mit dem der Inhalt der Editor-Puffers geändert wurde, wird die aktuelle Zeile ausgegeben. Bei globalen Suchen/Ersetzen-Kommandos (siehe *g*, *s* und *v*) wird *autoprint* unterdrückt.

autowrite, aw
noautowrite, noaw (Standard)
Der Inhalt des Editor-Puffers wird in die aktuelle Datei geschrieben, wenn Änderungen vorgenommen worden sind und eines der Kommandos *e* (edit), *n* (next), *rew* (rewind) oder *!* (escape) aufgerufen wird.

beautify, bf
nobeautify, nobf (Standard)
Alle Steuerzeichen außer dem Tabulatorzeichen, dem Neue Zeile- und dem Formularvorschubzeichen werden aus dem Eingabetext entfernt.

directory=dvz
dir=dvz
Mit *dvz* wird das Dateiverzeichnis angegeben, in welchem der Editor-Puffer angelegt werden soll. Hat der Benutzer für dieses Dateiverzeichnis keine Schreiberlaubnis, so wird der Editor verlassen.

edcompatible, ed
noedcompatible, noed
Ist das *s*-Kommando (substitute) mit den Schaltern *g* und *c* versehen, so werden diese Schalter gespeichert; die nochmalige Angabe der Schalter hebt ihre Wirkung auf.

errorbells, eb
noerrorbells, noeb
Vor Fehlermeldungen, die in der letzten Zeile (in der Regel invers) erscheinen, wird ein Klingelsignal gesendet.

exrc, ex
Diese Option ist aus Sicherheitsgründen nicht wirksam. Sie sollte bewirken, daß *.exrc*-Dateien zulässig sind, die im aktuellen Dateiverzeichnis stehen.

flash, fl (Standard)
noflash, nofl
Wenn in der Beschreibung des Terminals ein Eintrag darüber enthalten ist, wie am Bildschirm ein optisches Signal erzeugt werden kann, so wird der Benutzer auf die dort beschriebene Art auf eine fehlerhafte Eingabe aufmerksam gemacht.

hardtabs=zahl
ht=zahl
Mit dieser Option legen Sie fest, in welchen Schritten auf dem Terminal Tabulatoren definiert sind (zur Optimierung der Bildschirmausgabe).

ignorecase, ic
noignorecase, noic (Standard)
Alle Großbuchstaben im Text werden beim Suchen mit regulären Ausdrücken wie Kleinbuchstaben behandelt. Gleichfalls werden alle Großbuchstaben in regulären Ausdrücken wie Kleinbuchstaben behandelt, außer in Ausdrücken für Zeichenklassen.

lisp
nolisp (Standard)
Die Option *autoindent* wird gesetzt und die *vi*-Kommandos öffnende runde Klammer (, schließende runde Klammer), öffnende geschweifte Klammer {, schließende geschweifte Klammer }, doppelte öffnende eckige Klammern [[und doppelte schließende eckige Klammern]] werden angepaßt.

list
nolist (Standard)
In den ausgegebenen Zeilen werden Tabulatorzeichen durch `^I` ersetzt und das Ende jeder Zeile durch ein $ markiert.

magic (Standard)
nomagic
Ändert die Interpretation der in regulären Ausdrücken und Ersetzungszeichenketten verwendeten Zeichen (siehe *Reguläre Ausdrücke und Ersetzungszeichenketten*).

mesg (Standard)
nomesg
Ein-/Ausschalten der Schreiberlaubnis für andere Benutzer auf das Terminal (z.B durch das Kommando *write*).

modelines (Standard)
nomodelines
Die ersten und die letzten fünf Zeilen einer eingelesenen Datei werden als Editorkommandos betrachtet und ausgeführt, wenn sie von der Form sind:
`ex:`*kommando*`:`

novice
nonovice (Standard)
Es sollen ausführlichere Fehlermeldungen erzeugt werden.

number, nu
nonumber, nonu (Standard)
Die Zeilen werden mit vorangestellten Zeilennummern ausgegeben.

optimize, opt
nooptimize, noopt (Standard)
Sie können das Terminal so umschalten, daß kein automatischer Wagenrücklauf erfolgt (zur Optimierung der Ausgabe).

paragraphs=zeichenkette
para=zeichenkette
Dieser Option wird eine Zeichenkette zugewiesen. Jeweils zwei in dieser Zeichenkette enthaltene Zeichen stehen für den Namen eines *nroff*-Makros, das einen Absatz (Paragraphen) einleitet. Ein Makro wird in einen Text im Format *.XX* eingetragen, wobei der Punkt . das erste Zeichen der Zeile ist.

prompt (Standard)
noprompt
Ist *prompt* gesetzt, so wird im Kommando-Modus das Bereitzeichen Doppelpunkt : angezeigt; andernfalls wird kein Bereitzeichen ausgegeben.

readonly
noreadonly (Standard)
Ist *readonly* gesetzt, so können die zu bearbeitenden Dateien nur gelesen werden. Veränderungen am Text sind nicht möglich. Ist *noreadonly* gesetzt, können Sie in den Dateien Veränderungen vornehmen.

redraw (Standard)
noredraw
Werden Änderungen vorgenommen, so sollen diese sofort auf dem Bildschirm ausgegeben werden. Wegen der großen Menge der ausgegebenen Daten ist dies jedoch nur sinnvoll, wenn die Daten mit einer hohen Geschwindigkeit übertragen werden.

remap (Standard)
noremap
Ist *remap* gesetzt, so werden bei der Makroumsetzung auch Makros berücksichtigt, die durch andere Makros definiert sind. Die Umsetzung wird fortgesetzt, bis das gewünschte Makro erstellt ist. Ist diese *noremap* gesetzt, so wird lediglich eine einstufige Umsetzung durchgeführt.

report=n
Wenn die Zahl der Zeilen, die mit dem letzten Kommando geändert, gelöscht oder kopiert wurde, größer als *n* ist, erscheint in der Status-Zeile eine Meldung.

scroll=n
Der bei *scroll* angegebene Wert *n* steht für die Anzahl von Zeilen, um die das Fenster bei einem CTRL D verschoben werden soll, und die bei einem *z*-Kommando (der zweifache Wert von *scroll*) ausgegeben werden sollen.

sections=zeichenkette
sect=zeichenkette
Dieser Option wird eine Zeichenkette zugewiesen. Jeweils zwei in dieser Zeichenkette enthaltene Zeichen stehen für den Namen eines *nroff*-Makros, das einen Abschnitt (Section) einleitet. Ein Makro wird in einen Text im Format *.XX* eingetragen, wobei der Punkt . das erste Zeichen der Zeile ist.

shiftwidth=n
sw=n
Der bei *sw* angegebene Wert *n* steht für die Schrittweite eines Software-Tabulators um die der Text bei gesetzter Option *autoindent* und den Kommandos Kleinerzeichen < und Größerzeichen > verschoben werden soll.

showmatch, sm
noshowmatch, nosm (Standard)
Wird im *vi* eine schließende runde Klammer) oder eine schließende geschweifte Klammer } eingegeben, so wird die zugehörige (oder { angezeigt, falls sie sich noch auf dem Bildschirm befindet.

showmode, smd
noshowmode, nosmd (Standard)
Wenn *smd* gesetzt ist, wird in der Status-Zeile angezeigt, wenn sich *vi* im *vi*-Eingabe-Modus befindet.

slowopen, slow
noslowopen, noslow (Standard)
Im *vi* soll die Korrektur des Bildschirminhalts während der Eingabe verzögert werden. Dadurch soll der Durchsatz bei nicht intelligenten Datensichtstationen erhöht werden.

tabstop=n
ts=n
n gibt die Software-Tabulatoren an, die vom Editor benutzt werden, um Tabulatoren in der editierten Datei zu expandieren.

taglength=zahl
tl=zahl
Im Zusammenhang mit dem *:tag*-Kommando wird hier bestimmt, wieviele Zeichen für die Suche signifikant sein sollen. Der Wert 0 besagt, daß alle Zeichen berücksichtigt werden.

tags=dateien
Namen der *tag*-Dateien, durch Leerzeichen getrennt.

term=zeichenkette
zeichenkette beschreibt für *vi* den zu verwendenden Terminaltyp. (Standardmäßig wird der Wert der Umgebungsvariable *TERM* verwendet.)

terse
noterse (Standard)
Ist *terse* gesetzt, so werden gekürzte Fehlermeldungen ausgegeben.

timeout
notimeout
Verzögerungszeit setzen/rücksetzen.

ttytype=zeichenkette
zeichenkette beschreibt für *vi* den zu verwendenden Terminaltyp. (Standardmäßig wird der Wert der Umgebungsvariable *TERM* verwendet.)

warn (Standard)
nowarn
Wenn *warn* gesetzt ist, erscheint bei einem *:!*-Kommando die Warnung *No write since last change*, wenn seit der letzten Sicherung mit *w* noch Änderungen am Editor-Puffer erfolgten.

window=n
Die Anzahl der Zeilen in einem Textfenster des *vi*.

wrapmargin=n
wm=n
Nur im *vi* wirksam. Wenn $n > 0$ ist, wird an das letzte Wort in einer Eingabezeile automatisch ein Neue-Zeile-Zeichen angefügt, so daß das Zeilenende um mindestens n Positionen vom rechten Rand des Bildschirms entfernt ist.

Mit $n=0$ (Standard) kann der automatische Zeilenumbruch ausgeschaltet werden.

wrapscan, ws (Standard)
nowrapscan, nows
Ist *wrapscan* gesetzt, so wird das Suchen mit /.../ oder ?...?, wenn das Ende bzw. der Anfang des Editor-Puffers erreicht ist, am Anfang bzw. Ende fortgesetzt. Falls *nows* gesetzt ist, wird das Suchen am Anfang bzw. am Ende der Datei beendet.

writeany, wa
nowriteany, nowa (Standard)
Wenn *nowa* gesetzt ist, wird bei einem Sicherungskommando *w* eine Prüfung auf die Existenz der Datei gemacht. Damit wird verhindert, daß Sie versehentlich eine schon existierende Datei überschreiben. Mit *w!* wird diese Prüfung umgangen. Wenn *wa* gesetzt ist, wird nicht geprüft, ob die Datei schon existiert.

FEHLERMELDUNGEN

Tritt bei *ex* ein Fehler auf, so sendet der Editor an die Datensichtstation das akustische Signal BEL (siehe *Tabellen und Verzeichnisse, Zeichensatz ISO 646*) und gibt eine Fehlermeldung aus. Bei einem Unterbrechungssignal kehrt *ex* auf die Kommandoebene zurück, d.h. Sie können ein neues *ex*-Kommando eingeben. Liest der Editor die Eingabe aus einer Datei, so bricht *ex* bei einem Unterbrechungssignal ab.

Stellt *ex* einen internen Fehler fest, so wird versucht, den Puffer zu retten, wenn die zuletzt vorgenommenen Änderungen noch nicht abgespeichert worden sind. Durch einen erneuten Aufruf von *ex* mit der Option *-r* kann auf die gesicherten Daten zurückgegriffen werden.

DATEIEN

$HOME/.exrc
: Datei mit Voreinstellungen für *ex* und *vi*. Diese Voreinstellungen sind nicht wirksam, wenn durch *EXINIT* Widersprüchliches definiert wird.

UMGEBUNGSVARIABLEN

TERM
: In der Umgebungsvariablen *TERM* muß der Typ der benutzten Datensichtstation festgelegt sein.

EXINIT
: Umgebungsvariable mit Voreinstellungen für *ex* und *vi*. Diese Voreinstellungen sind immer wirksam.

BEISPIEL

Es werden einige *ex*-Kommandos vorgeführt und erläutert:

```
$ ex zahlen                    -  ex aufrufen
"zahlen" 4 lines, 20 characters
:set number                    -  Zeilennummern ausgeben
:1,$p                          -  Zeile 1 bis letzte Zeile ausgeben
     1  eins                   -  Ursprünglicher Inhalt von zahlen
     2  zwei
     3  drei
     4  vier
:1 c 1                         -  Zeile 1 ändern
     1  EINS                   -  Neue Zeile 1
     2  .                      -  Eingabemodus verlassen
:2,3 co 4                      -  Zeile 2 bis 3 hinter Zeile 4
     6  drei                      kopieren
:1,$ g/zwei/s//ZWEI/           -  In der gesamten Datei "zwei"
                                  durch "ZWEI" ersetzen
:2 a                           -  Hinter Zeile 2 neue Zeile einfügen
     3  DREI                   -  Neue Zeile 3
     4  .                      -  Eingabemodus verlassen
:4 i                           -  Vor Zeile 4 neue Zeile einfügen
     4  VIER                   -  Neue Zeile 4
     5  .                      -  Eingabemodus verlassen
:5,$ d a                       -  Zeile 5 bis Dateiende löschen und in Puffer a sichern
     4  VIER
:1,$p                          -  Zeile 1 bis letzte Zeile ausgeben
     1  EINS
     2  ZWEI
     3  DREI
     4  VIER
:wq                            -  Sichern des Pufferinhalts und ex verlassen
"zahlen" 4 lines, 20 characters
```

SIEHE AUCH

vi, ed, edit, vedit

exec
Die aktuelle Shell überlagern (execute)

Das in die Bourne-Shell *sh* eingebaute Kommando *exec* hat zwei Funktionen:

- *exec* überlagert die aktuelle Shell durch ein anderes Programm (Format 1). Die aktuelle Shell ist beendet, wenn dieses Programm gestartet wird. Es entsteht kein neuer Prozeß; das erkennen Sie daran, daß sich die Prozeß-Nummer nicht ändert (siehe *Beispiel 2*).

  Wenn Sie *exec* im Dialog eingeben, ist nach der Ausführung des angegebenen Programms die übergeordnete Shell aktiv bzw. Ihre Sitzung beendet.
  Steht *exec* in einer Shell-Prozedur, so wird diese Prozedur beendet. Kommandos, die in der Prozedur nach dem Aufruf von *exec* stehen, werden nie ausgeführt.

- *exec* lenkt die Standard-Eingabe bzw. die Standard-Ausgabe der Shell um auf eine Datei (Format 2). Nach der Ausführung von *exec* lesen alle eingegebenen Kommandos aus dieser Datei bzw. schreiben in diese Datei, bis die aktuelle Shell beendet ist.

exec programm[umlenkung] Format 1
exec umlenkung Format 2

Format 1: Die Shell durch ein anderes Programm überlagern

exec programm[umlenkung]

programm
beliebiges Kommando, Programm oder Shell-Prozedur, aber kein eingebautes *sh*-Kommando. Für die dazugehörige Datei brauchen Sie Ausführrecht.

Die aktuelle Shell beendet sich; stattdessen wird *programm* ausgeführt. Wenn *programm* ausgeführt ist, meldet sich die übergeordnete Shell zurück oder der Begrüßungsbildschirm wird ausgegeben.

umlenkung
weist dem angegebenen Programm, falls es von der Standard-Eingabe liest bzw. auf die Standard-Ausgabe schreibt, eine Datei für die Eingabe bzw. Ausgabe zu.
Folgende Angaben für *umlenkung* sind möglich:

>datei
Die Standard-Ausgabe des angegebenen Programms wird auf *datei* umgelenkt. In *datei* enthaltene Daten werden vorher gelöscht.

>>datei
Die Standard-Ausgabe des angegebenen Programms wird auf *datei* umgelenkt. In *datei* enthaltene Daten werden nicht gelöscht; die Ausgaben des Programms werden an den bisherigen Inhalt von *datei* angehängt.

2>datei
Die Standard-Fehlerausgabe des angegebenen Programms wird auf *datei* umgelenkt.

<datei
Die Standard-Eingabe des angegebenen Programms wird auf *datei* umgelenkt. Das Programm liest also seine Eingabe aus dieser Datei.

Format 2: Standard-Eingabe bzw. -Ausgabe der Shell umlenken

exec umlenkung

umlenkung
weist Kommandos, die von der Standard-Eingabe lesen bzw. auf die Standard-Ausgabe schreiben, eine Datei für die Eingabe bzw. Ausgabe zu. Diese Umlenkung gilt für alle Kommandos, die die aktuelle Shell nach *exec* ausführt.
Folgende Angaben für *umlenkung* sind möglich:

>datei
Alle Kommandos, die die aktuelle Shell ausführt, schreiben ihre Ausgabe nacheinander in die angegebene Datei. In *datei* enthaltene Daten werden gelöscht.
So werden alle Ausgaben gesammelt.

Wenn Sie *exec* im Dialog eingeben, können Sie die Umlenkung nur rückgängig machen:

- mit der Taste END. Damit beenden Sie die aktuelle Shell.
- mit der Eingabe *exec >/dev/tty*. Damit lenken Sie die Standard-Ausgabe wieder auf den Bildschirm um.

Steht *exec* in einer Shell-Prozedur, so gilt die Umlenkung nur für die Kommandos, die in der Prozedur auf den Aufruf von *exec* folgen. Wollen Sie die Umlenkung innerhalb der Prozedur rückgängig machen, müssen Sie in der Prozedur an der betreffenden Stelle das Kommando *exec >/dev/tty* einfügen. Damit lenken Sie die Standard-Ausgabe der nachfolgenden Kommandos wieder auf den Bildschirm um.

>>datei
Alle Kommandos, die die aktuelle Shell ausführt, schreiben ihre Ausgabe nacheinander in die angegebene Datei. In *datei* enthaltene Daten werden nicht gelöscht; die Ausgaben der Kommandos werden an den bisherigen Inhalt von *datei* angehängt.

2>datei
Die Standard-Fehlerausgabe wird für alle Kommandos, die die aktuelle Shell ausführt, auf *datei* umgelenkt.

<datei
Wenn Sie *exec* im Dialog eingeben, liest die aktuelle Shell die Kommandos, die sie ausführen soll, aus der angegebenen Datei. Nach dem letzten Kommando beendet sich die Shell.

Steht dieses Kommando in einer Shell-Prozedur, so lesen alle Kommandos, die in der Prozedur auf den Aufruf von *exec* folgen, ihre Eingabe aus der angegebenen Datei. Jeder Lesevorgang verändert die Position des Lesezeigers in dieser Datei. Wenn der Lesezeiger auf Dateiende (EOF) steht, erhalten alle noch folgenden Kommandos keine Eingabe.

Mit dem Kommando *exec </dev/tty* lenken Sie die Standard-Eingabe für die nachfolgenden Kommandos wieder auf die Tastatur um.

datei
Wenn Sie *exec* im Dialog eingeben, muß das der Name einer Shell-Prozedur sein. Für diese Shell-Prozedur brauchen Sie nur Leserecht.

Wenn *exec* in einer Shell-Prozedur steht, ist das der Name der Datei, aus der die nachfolgenden Kommandos ihre Eingabe lesen sollen.

BEISPIELE

1. Was passiert bei folgender Eingabe:

```
$ exec date
```

Das Kommando *exec* beendet zuerst die aktuelle Shell und führt dann das Kommando *date* aus. Am Bildschirm wird das aktuelle Datum mit Uhrzeit ausgegeben.

Wenn Sie das Kommando in Ihrer Login-Shell eingegeben haben, wird anschließend der Begrüßungsbildschirm ausgegeben. Sie müssen sich neu am System anmelden.

Wenn Sie das Kommando in einer Subshell eingegeben haben, meldet sich die übergeordnete Shell zurück.

2. Wenn Sie die aktuelle Shell durch ein anderes Programm überlagern, ändert sich die Prozeß-Nummer nicht. Das soll mit den folgenden Dateien gezeigt werden:

   – Die Datei *proz1* hat folgenden Inhalt:

   ```
   echo proz1 hat Prozessnummer: $$
   sh proz2
   ```

   – Die Datei *proz2* hat folgenden Inhalt:

   ```
   echo proz2 hat Prozessnummer: $$
   exec proz3
   ```

   – Die Datei *proz3* - sie muß ausführbar sein - hat folgenden Inhalt:

   ```
   echo proz3 hat Prozessnummer: $$
   ```

   Die Shell-Prozedur *proz1* wird gestartet:

   ```
   $ sh proz1
   proz1 hat Prozessnummer: 2755
   proz2 hat Prozessnummer: 2760
   proz3 hat Prozessnummer: 2760
   ```

   Weil *proz3* von *proz2* mit *exec* aufgerufen wird, laufen beide Shell-Prozeduren unter derselben Prozeß-Nummer. Genauer: Die Shell, die die Shell-Prozedur *proz2* ausführt, wird überlagert von der Shell, die *proz3* ausführt.

3. Mit dem Kommando *exec* wird in der Shell-Prozedur *exctest* die Standard-Eingabe für die nachfolgenden Kommandos umgelenkt auf die Datei */etc/group*. Die Datei *exctest* hat folgenden Inhalt:

   ```
   : Aufruf mit sh exctest
   exec </etc/group
   read zeile
   echo $zeile
   echo
   cat
   ```

   Die Shell-Prozedur *exctest* wird gestartet:

   ```
   $ sh exctest
   root::0:root

   daemon::1:daemon
   sys::2:sys:
   bin::3:bin,admin
   uucp::4:
   .
   .
   .
   ```

   Das eingebaute *sh*-Kommando *read* weist der Variablen *zeile* die erste Zeile der Datei */etc/group* zu. Das Kommando *echo $zeile* gibt den Inhalt dieser Variablen aus.

Das Kommando *cat* liest seine Eingabe ebenfalls aus der Datei */etc/group*. Da der Lesezeiger jetzt auf der zweiten Zeile dieser Datei steht, gibt *cat* den Inhalt erst ab Zeile zwei aus.

Anschließend steht der Lesezeiger auf Dateiende. Weitere Kommandos nach dem Aufruf von *cat* würden deshalb als Eingabe die leere Zeichenkette erhalten.

SIEHE AUCH

sh
exec() [19]

exit
Shell-Prozedur beenden

Das in die Bourne-Shell *sh* eingebaute Kommando *exit* beendet Shell-Prozeduren. Beim Aufruf können Sie mit einer Zahl den Ende-Status der entsprechenden Shell-Prozedur festlegen.

Auch ohne Aufruf von *exit* beenden sich Shell-Prozeduren:

- wenn das letzte Kommando ausgeführt ist, d.h. wenn die ausführende Shell Datei-Ende (EOF) erkennt. Ende-Status ist der Ende-Status des zuletzt ausgeführten Kommandos.
- wenn die ausführende Shell ein Kommando nicht findet oder einen Syntax-Fehler erkennt. Ende-Status ist ein Wert ≠ 0.

Wenn Sie das Kommando *exit* im Dialog eingeben, wird die aktuelle Shell beendet.

exit[_ende_status]

ende_status
: Eine Zahl zwischen 0 und 256. Das ist der Ende-Status, mit dem die Shell-Prozedur beendet wird.
Wenn Sie eine größere Zahl angeben, erhalten Sie als Ende-Status nicht die angegebene Zahl, sondern den entsprechenden Wert modulo 256:
Die angegebene Zahl wird durch 256 geteilt, der ganzzahlige Rest ist dann der entsprechende Wert modulo 256.

  Mit dem Kommando *echo $?* können Sie den Ende-Status abfragen.

  ende_status nicht angegeben:
Ende-Status ist der Ende-Status des Kommandos, das vor dem Aufruf von *exit* ausgeführt wurde.

BEISPIELE

1. Das Kommando *false* läßt sich durch folgende Shell-Prozedur realisieren:

```
: Shell-Version des Kommandos false
: Ende-Status immer 255
exit 255
```

2. Die beiden Shell-Prozeduren *ende* und *abfrage* sollen zeigen, wie der Ende-Status ausgewertet werden kann:

   – Die Datei *ende* hat folgenden Inhalt:

```
: Aufruf mit sh ende datei
if [ -f "$1" ]
then exit 2
elif [ -d "$1" ]
then exit 3
else exit 4
fi
```

   – Die Datei *abfrage* hat folgenden Inhalt:

```
: Aufruf mit sh abfrage datei
sh ende "$1"
case $? in
     2) echo Inhalt der Datei $1:; cat $1;;
     3) echo Inhalt von $1:; ls -l $1|pg;;
     4) echo Die Eingabe $1 hat für diese Prozedur keinen Sinn!
esac
```

   Die Shell-Prozedur *abfrage* wird gestartet:

```
$ sh abfrage .profile
Inhalt der Datei .profile:
HOME=/home1/rosa/src
export HOME
.
.
.
```

   Die Shell-Prozedur *abfrage* ruft die Prozedur *ende* auf. Diese Prozedur liefert als Ende-Status den Wert 2 zurück, da *.profile* eine einfache Datei ist. Anschließend werden in der case-Anweisung die Kommandos ausgeführt, die bei dem entsprechenden Muster 2 angegeben sind.

   Den Inhalt dieser beiden Shell-Prozeduren kann man nicht ohne weiteres in eine einzige Datei schreiben, denn das Kommando *exit* beendet die Prozedur.

Die Shell-Prozedur *endeaktion* zeigt, wie der Ende-Status innerhalb einer Prozedur ausgewertet werden kann:

```
: Aufruf mit sh endeaktion datei
(if [ -f "$1" ]
   then exit 2
   elif [ -d "$1" ]
   then exit 3
   else exit 4
fi)
case $? in
     2) echo Inhalt der Datei $1:; cat $1;;
     3) echo Inhalt von $1:; ls -l $1|pg;;
     4) echo Die Eingabe $1 hat für diese Prozedur keinen Sinn!
esac
```

Die runden Klammern um die if-Anweisung bewirken, daß eine Subshell aufgerufen wird. Diese Subshell beendet sich mit dem Ende-Status, der bei dem betreffenden Kommando *exit* angegeben ist. Anschließend meldet sich die übergeordnete Shell zurück; das ist die Shell, die die Prozedur *endeaktion* bearbeitet. Sie führt die restlichen Kommandos der Prozedur aus.

SIEHE AUCH

false, *sh*
exit() [19]

export
Shell-Variablen exportieren

Das in die Bourne-Shell *sh* eingebaute Kommando *export* exportiert die angegebene Shell-Variable. Jedes Kommando kennt dann den Namen dieser Variablen und kann auf ihren Wert zugreifen.

Wenn Sie *export* ohne Argumente aufrufen, schreibt *export* die Namen aller Shell-Variablen auf die Standard-Ausgabe, die in der aktuellen Shell exportiert wurden.

Wenn Sie die Shell beenden, in der Sie Variablen definiert und exportiert haben, sind diese Variablen wieder gelöscht. Deshalb sollten Sie häufig verwendete Variablen in Ihrer Datei *$HOME/.profile* definieren und exportieren.

Stellungsparameter und Shell-Funktionen können nicht exportiert werden. Einige Standard-Variablen der Shell sind auch in Subshells verfügbar, ohne exportiert worden zu sein. Dazu gehören z.B. *HOME*, *IFS*, *PATH*, *PS1*, *PS2*, usw. Wenn Sie einer dieser Variablen einen anderen Wert zuweisen und der geänderte Wert auch in jeder Subshell gelten soll, dann müssen Sie diese Variable exportieren. Sonst gilt in jeder Subshell wieder der Standard-Wert.

Das eingebaute *sh*-Kommando *set* gibt alle Variablen mit ihrem jeweiligen Wert aus, die in der aktuellen Shell definiert sind, also auch solche, die Sie nicht exportiert haben. Das Kommando *env* gibt die Namen und Werte aller Shell-Variablen aus, die an jedes aufgerufene Kommando und jede Subshell weitergereicht werden. Variablen wie *IFS*, *MAILCHECK*, *PATH*, *PS1* und *PS2* sind in dieser Ausgabe nur enthalten, wenn sie mit *export* exportiert wurden.

export[name]...

name
: Name der Shell-Variablen, die Sie exportieren wollen. Sie können dieser Variablen auch erst nach dem Aufruf von *export* einen Wert zuweisen.
Sie können beliebig viele Shell-Variablen angeben, jeweils getrennt durch ein Leerzeichen.

name nicht angegeben:
export schreibt die Namen aller Shell-Variablen auf die Standard-Ausgabe, die in der aktuellen Shell exportiert wurden. Die Ausgabe hat folgende Form:

```
export name
.
.
.
```

BEISPIEL

Die Shell-Prozedur *editor* soll den Wert der Variablen *EDITOR* ausgeben. Die Prozedur hat folgenden Inhalt:

```
: Aufruf mit sh editor
echo EDITOR: $EDITOR
```

Der folgende Bildschirm-Dialog soll zeigen, warum die Shell-Variable *EDITOR* exportiert werden muß:

```
$ export
export HOME
export PATH
export PS1
$ EDITOR=vi
$ sh editor
EDITOR:
$ export EDITOR
$ export
export HOME
export PATH
export PS1
export EDITOR
$ sh editor
EDITOR: vi
```

Wenn Sie eine Shell-Variable definieren, ist sie nur der aktuellen Shell bekannt. Da jede Shell-Prozedur von einer Subshell ausgeführt wird, muß eine Variable für Shell-Prozeduren exportiert werden.

SIEHE AUCH

env, *set*, *sh*

expr
Ausdrücke auswerten (evaluate expression)

expr interpretiert die in der Kommandozeile angegebenen Argumente als Ausdrücke und wertet sie nacheinander aus. Das Ergebnis der Auswertung wird auf die Standard-Ausgabe ausgegeben.
Mit *expr* können Sie z.B. ganzzahlige Berechnungen durchführen oder Zeichenketten miteinander vergleichen.

expr␣ausdruck␣...

ausdruck
ausdruck besteht entweder nur aus einem Operanden oder aus zwei Operanden, die mit einem Operator verknüpft sind. Als Operanden können Sie beliebige Zeichenketten angeben. Zeichenketten, die nur aus den Ziffern 0 bis 9 bestehen, interpretiert *expr* als ganze Zahlen. Durch ein vorangestelltes Minuszeichen - können Sie ganze Zahlen als negative Zahlen kennzeichnen.

Operanden und Operatoren werden durch Leer- oder Tabulatorzeichen voneinander getrennt. Wenn Sie eine Zeichenkette als Operand angeben wollen, die Leer- oder Tabulatorzeichen enthält, müssen Sie deshalb entweder die ganze Zeichenkette oder die Leer- oder Tabulatorzeichen in Hochkommata '...' oder Anführungszeichen "..." einschließen, damit diese nicht als Trennzeichen interpretiert werden. Sonderzeichen der Shell müssen Sie in Hochkommata '...' oder Anführungszeichen "..." einschließen oder mit einem Gegenschrägstrich \ entwerten (siehe *Tabellen und Verzeichnisse, Sonderzeichen der Bourne-Shell sh*).

Wenn *ausdruck* nur aus einem Operanden besteht, ist das Ergebnis der Auswertung dieser Operand selber.

Wie zwei Operanden miteinander verknüpft werden können, ist im folgenden beschrieben. Dabei sind die Operatoren nach aufsteigendem Vorrang geordnet, Operatoren mit gleichem Vorrang sind in geschweiften Klammern {...} zusammengefaßt.

Das Ergebnis 0 steht für den Wert 0, nicht für eine leere Zeichenkette.

op1␣|␣op2
Wenn *op1* weder die leere Zeichenkette noch 0 ist, ist das Ergebnis *op1*; sonst *op2*.

op1␣&␣op2
Wenn weder *op1* noch *op2* die leere Zeichenkette oder 0 ist, ist das Ergebnis *op1*; sonst 0.

op1␣rel␣op2

rel kann einer der folgenden Vergleichsoperatoren sein:

<	kleiner als
<=	kleiner als oder gleich
=	gleich
>=	größer als oder gleich
>	größer als
!=	ungleich

Wenn die Bedingung erfüllt ist, ist das Ergebnis des Vergleichs 1. Wenn die Bedingung nicht erfüllt ist, ist das Ergebnis des Vergleichs 0. Wenn sowohl *op1* als auch *op2* ganze Zahlen sind, werden sie numerisch miteinander verglichen. Ansonsten werden sie als Zeichenketten lexikalisch miteinander verglichen.

op1␣{+ -}␣op2

Wenn *op1* und *op2* ganze Zahlen sind, ist das Ergebnis die Summe bzw. Differenz der beiden Zahlen. Wenn eines der Argumente keine ganze Zahl ist, gibt *expr* eine Fehlermeldung aus (siehe *FEHLERMELDUNGEN*).

op1␣{* / %}␣op2

Wenn *op1* und *op2* ganze Zahlen sind, ist das Ergebnis das Ergebnis der angegebenen arithmetischen Operation:

*	Multiplikation
/	(ganzzahlige) Division
%	Rest bei (ganzzahliger) Division

Wenn eines der Argumente keine ganze Zahl ist, gibt *expr* eine Fehlermeldung aus (siehe *FEHLERMELDUNGEN*).

op1␣:␣op2

Die Zeichenketten *op1* und *op2* werden miteinander verglichen, beginnend mit dem ersten Zeichen in jeder Zeichenkette und endend mit dem letzten Zeichen in *op2*. *op2* können Sie in Form eines einfachen regulären Ausdrucks (siehe *Tabellen und Verzeichnisse, Reguläre Ausdrücke*) angeben. Wenn *op1* und *op2* vom ersten Zeichen in jeder Zeichenkette bis zum letzten Zeichen in *op2* übereinstimmen, wird normalerweise die Anzahl der übereinstimmenden Zeichen ausgegeben. Wenn Sie für *op2* jedoch das Muster \(...\) verwenden, wird der Teil von *op1* ausgegeben, der zu diesem Muster paßt (siehe *Beispiele 6* und *7*).

(...)

Durch Setzen runder Klammern *(...)* können Sie die Auswertungsreihenfolge verändern. Die in runden Klammern zusammengefaßten Ausdrücke werden zunächst ausgewertet, auch wenn der Vorrang der Operatoren etwas anderes aussagt.

ENDE-STATUS

0 Ausdruck ist weder fehlerhaft noch leer noch 0
1 Ausdruck ist leer oder 0
2 Ausdruck ist fehlerhaft oder Division durch 0

FEHLERMELDUNGEN

`expr: non-numeric argument`

Bei den arithmetischen Operatoren +, -, *, /, % dürfen Sie nur ganzzahlige Operanden angeben.

`expr: division by zero`

Sie haben durch 0 dividiert.

`expr: syntax error`

Sie haben einen Syntaxfehler beim Aufruf von *expr* gemacht, z.B. Operanden und Operatoren nicht durch Leer- oder Tabulatorzeichen getrennt.

`expr: RE error`

Sie haben beim Mustervergleichsoperator : den zweiten Operanden nicht in korrekter Form als einfachen regulären Ausdruck angegeben.

BEISPIELE

1. Einfache Rechnung:

```
$ expr 21 + 9 '*' 2 / 6
24
```

2. Im folgenden Beispiel wird zu der Variablen *a* der Wert 1 addiert.

```
$ echo $a
3
$ a=`expr $a + 1`
$ echo $a
4
```

3. Im folgenden Beispiel werden zwei Umgebungsvariablen verglichen.

```
$ echo $op1 $op2
text1 text2
$ expr $op1 = $op2
0
```

Problematisch wird der Vergleich, wenn der Wert einer Variablen einen Operator für *expr* darstellt. Das Problem kann durch Verbindung der Variablen mit einem unproblematischen Zeichen gelöst werden.

```
$ echo $op1 $op2
= =
$ expr $op1 = $op2
expr: syntax error
$ expr X$op1 = X$op2
1
```

4. Ausgabe der Anzahl der Zeichen in *VAR*:

```
$ echo $VAR
Hallo
$ expr $VAR : '.*'
5
```

5. Vergleich zweier Zeichenketten:

```
$ expr mausoleum : maus
4
```

6. Vergleich zweier Zeichenketten, wobei das Muster als regulärer Ausdruck angegeben wird. Das Zeichen Dach ^ ist ein Sonderzeichen und muß deshalb durch einen Gegenschrägstrich \ entwertet werden.

```
$ expr abc : '[\^d-f]'
1
```

7. In der Variablen *a* ist der absolute Pfadname einer Datei abgelegt, etwa */usr/anna/parnassum/infinitum*. Um den einfachen Dateinamen, also *infinitum*, auszugeben, geben Sie ein:

```
$ expr $a : '.*/\(.*\)'
infinitum
```

8. Wenn in der Variablen *a* entweder der absolute Pfadname oder der einfache Dateiname einer Datei abgelegt ist, erhalten Sie den einfachen Dateinamen auf folgende Weise:

```
$ expr $a : '.*/\(.*\)' \| $a
```

SIEHE AUCH

ed, sh

exstr
Zeichenketten in Quellprogrammen suchen und ersetzen (extract strings)

Mit *exstr* können Sie aus C-Quellprogramm-Dateien Zeichenketten herausziehen (Format 1) und diese durch Aufrufe einer Funktion ersetzen (Format 2), die zur Laufzeit des Programms passende Zeichenketten sucht und dazubindet (siehe *gettxt()* [14]). Als Zeichenketten gelten alle Folgen von einem oder mehreren druckbaren Zeichen, die in Anführungszeichen "..." eingeschlossen sind, also nicht nur Argumente von *printf*.

exstr[␣**-e**]␣datei... Format 1
exstr␣**-r**[␣**-d**]␣datei... Format 2

Format 1: Zeichenketten aus C-Quellprogramm-Dateien herausziehen

exstr[␣**-e**]␣datei...

Keine Option angegeben:
exstr sucht alle Zeichenketten in *datei* und schreibt sie auf die Standard-Ausgabe, wobei jeder Zeichenkette der Name von *datei* und ein Doppelpunkt vorangestellt wird.

-e
exstr gibt zusätzlich zu den gefundenen Zeichenketten Informationen über ihre Position in *datei* aus. Die Ausgabe erfolgt in einer Liste, deren Zeilen aus folgenden Feldern bestehen:

datei:zeile:spalte:meldungsdatei:meldungsnummer:zeichenkette

datei
Name der C-Quellprogramm-Datei

zeile
Zeilennummer der Zeile in *datei*, in der *zeichenkette* steht

spalte
Spaltennummer, in der *zeichenkette* beginnt

meldungsdatei
leeres Feld

meldungsnummer
leeres Feld

zeichenkette
gefundene Zeichenkette

Um die Felder *meldungsdatei* und *meldungsnummer* zu füllen, leiten Sie die Ausgabe in eine Datei um, die sie dann editieren können. In der Datei tragen Sie in die beiden Felder eine Meldungsdatei und eine Meldungsnummer ein:

meldungsdatei
: Name der Meldungsdatei, die die Zeichenketten enthält, die später an die Stelle der ursprünglich in der C-Quellprogramm-Datei enthaltenen Zeichenketten geschrieben werden sollen. *meldungsdatei* muß mit dem Kommando *mkmsgs* an geeigneter Stelle im Dateibaum angelegt werden (siehe *mkmsg*).

meldungsnummer
: Nummer, die *zeichenkette* in *meldungsdatei* hat.

Um Zeichenketten in der Quellprogramm-Datei zu ersetzen, verwenden Sie das zweite Format von *exstr*.

datei
: Name der C-Quellprogramm-Datei, die *exstr* nach Zeichenketten durchsuchen soll. Sie können mehrere Dateien angeben.

Format 2: Zeichenketten in C-Quellprogramm-Dateien ersetzen

exstr␣-r[␣-d]␣datei...

-r
: *exstr* ersetzt jede Zeichenkette in *datei* durch den Funktionsaufruf der Funktion *gettxt()*, die zur Laufzeit des Programms eine entsprechende Zeichenkette sucht und dazubindet.

-d
: Nur zusammen mit Option *-r*! Wenn der Funktionsaufruf *gettxt()* zur Laufzeit des Programms scheitert, wird die ursprüngliche Zeichenkette ausgegeben.

datei
: Name der C-Quellprogramm-Datei, die *exstr* nach Zeichenketten durchsuchen soll. Sie können mehrere Dateien angeben.

Anwendung

exstr ist nützlich für Anwenderprogramme, die in einer internationalen Umgebung ablauffähig sind und Meldungstexte in verschiedenen Landessprachen ausgeben können sollen. *exstr* ersetzt Zeichenketten in C-Quellprogramm-Dateien durch Funktionsaufrufe, die auf Zeichenketten in einer Datenbasis zeigen. Auf welche Datenbasis zugegriffen wird, hängt vom zur Laufzeit des Programms aktuellen Wert der Umgebungsvariablen *LC_MESSAGES* ab (siehe *environ*, [7], [19]).

Arbeitsschritte mit exstr

- Als erstes holen Sie mit *exstr -e datei.c > aus_datei* die Zeichenketten aus einer C-Quellprogramm-Datei *datei.c* heraus und schreiben sie mit ihrer Positionsangabe in eine Datei *aus_datei*.
- Dann überprüfen Sie die Liste der Zeichenketten in *aus_datei* und bestimmen, welche Zeichenketten in eine andere Landessprache übersetzt und durch die Meldungstext-Suchfunktion *gettxt()* ersetzt werden können.
- Die Zeilen mit Zeichenketten, die nicht übersetzt werden können, löschen Sie in *aus_datei*. In den Zeilen mit Zeichenketten, die übersetzt werden können, füllen Sie das Feld *meldungsdatei* mit dem Namen der Meldungsdatei und das Feld *meldungsnummer* mit der Nummer, die die entsprechende Zeichenkette in *meldungsdatei* hat. Die Meldungsdateien, die Sie angeben, müssen mit *mkmsgs* angelegt worden sein und sich im Dateiverzeichnis */usr/lib/locale/Locale/LC_MESSAGES* befinden. Der einfache Name des Dateiverzeichnisses */usr/lib/locale/Locale*, also *Locale*, entspricht der Landessprache, in der die Zeichenketten in den darunter liegenden Meldungsdateien geschrieben sind (siehe *setlocale()*, [19]). Die Meldungsnummern müssen den jeweiligen Nummern entsprechen, die die Zeichenketten in der angegebenen Meldungsdatei haben.
- Die von Ihnen veränderte Datei *aus_datei* übergeben Sie an den Kommandoaufruf *exstr -r* als Eingabe, um von der ursprünglichen C-Quellprogramm-Datei *datei.c* eine neue Fassung *neu_datei.c* zu erstellen, in der Zeichenketten durch Aufrufe der Meldungstext-Suchfunktion *gettxt()* ersetzt sind. Der Aufruf sieht wie folgt aus:

  ```
  exstr -r datei.c < aus_datei > neu_datei.c
  ```

  An die Funktion *gettxt()* werden zwei Argumente übergeben. Das erste Argument wird aus den beiden Feldern *meldungsdatei* und *meldungsnummer* in *aus_datei* gebildet, das zweite Argument besteht aus der Zeichenkette, die im Fehlerfall ausgegeben werden soll, also falls der Aufruf von *gettxt()* zur Laufzeit des Programms scheitert. Das zweite Argument ist die leere Zeichenkette, es sei denn, Sie haben die Option *-d* angegeben (siehe oben).

Einschränkungen

exstr kann nicht alle Arten von Zeichenketten in C-Quellprogramm-Dateien ersetzen, z.B:

- Zeichenketten, die Variablen der Speicherklasse *static* zugewiessen sind, dürfen durch *exstr* nicht ersetzt werden.
- Zeichenketten, die aus Escape-Folgen bestehen, können nicht in eine andere Landesspache übersetzt und sollen folglich nicht durch *exstr* ersetzt werden. Um zu verhindern, daß ungültiger Code produziert wird, müssen Sie also die von *exstr -e* erzeugte Ausgabe überprüfen und Zeilen löschen oder ändern, die durch Aufrufe von *gettxt()* nicht unmittelbar ersetzbare Zeichenketten enthalten.

FEHLERMELDUNGEN

Die wichtigsten Fehlermeldungen sind:

```
ERROR: cannot replace string '%s' in line (%d) of file (%s)
```

Fehler: die Zeichenkette '%s' in Zeile (%d) der Datei (%s) kann nicht ersetzt werden.

```
ERROR: stdin: invalid message number '%s'
```

Fehler: Standard-Eingabe: ungültige Meldungsnummer '%s'

```
ERROR: stdin: invalid message file name '%s'
```

Fehler: Standard-Eingabe: ungültiger Name der Meldungsdatei '%s'

```
ERROR: stdin: badly formed replacement string
```

Fehler: Ersetzungs-Zeichenkette hat ein ungültiges Format

DATEIEN

/usr/lib/locale/Locale/LC_MESSAGES
Dateiverzeichnis mit den Meldungsdateien, wobei der einfache Name des Dateiverzeichnisses */usr/lib/locale/Locale*, also *Locale*, der Landessprache entspricht, in der die Zeichenketten in den darunter befindlichen Dateien geschrieben sind.

UMGEBUNGSVARIABLEN

LC_MESSSAGES
enthält den zu durchsuchenden Sprachraum

BEISPIEL

Das folgende Beispiel zeigt, wie Sie *exstr* anwenden können:

Wir nehmen an, die C-Quellprogramm-Datei *datei.c* enthalte zwei Zeichenketten.

```
main()
{
   printf("Erste Zeichenkette\n");
   printf("Zweite Zeichenkette\n");
}
```

Der Aufruf von *exstr* ohne Option zieht aus der Datei *datei.c* die in "..." eingeschlossenen Zeichenketten heraus und schreibt sie auf die Standard-Ausgabe.

```
$ exstr datei.c
datei.c:Erste Zeichenkette\n
datei.c:Zweite Zeichenkette\n
```

Die Zeichenketten selbst und zusätzliche Informationen über ihre Position in der Datei erhalten Sie mit der Option *-e*. Um die herausgezogenen Zeichenketten zu überprüfen und ggf. zu löschen oder zu ändern, lenken Sie die Standard-Ausgabe in eine Datei *aus_datei* um.

```
$ exstr -e datei.c > aus_datei
$ cat aus_datei
datei.c:3:8:::Erste Zeichenkette\n
datei.c:4:8:::Zweite Zeichenkette\n
```

Jede Ausgabezeile besteht aus sechs durch Doppelpunkt voneinander getrennten Feldern. Im 1. Feld steht der Name der C-Quellprogramm-Datei, im 2. Feld die Zeilennummer der Zeichenkette (hier 3 und 4), im 3. Feld die Spaltennummer des 1. Zeichens der Zeichenkette (hier 8), im 6. Feld die Zeichenkette selbst ohne Anführungsstriche. Im 4. Feld tragen Sie mit Hilfe eines Editors die Meldungsdatei ein, aus der *gettxt()* später die neue Zeichenkette holen soll und im 5. Feld die Nummer, die die Zeichenkette in der Meldungsdatei hat. Angenommen, *UX* ist der Name der Meldungsdatei und die entsprechenden Zeichenketten hätten die Meldungsnummern 1 und 2, dann müßte *aus_datei* wie folgt aussehen, nachdem Sie sie bearbeitet haben:

```
datei.c:3:8:UX:1:Erste Zeichenkette\n
datei.c:4:8:UX:2:Zweite Zeichenkette\n
```

Nun können Sie *exstr -r* aufrufen, um in der C-Quellprogramm-Datei alle Zeichenketten durch den Funktionsaufruf *gettxt()* zu ersetzen. Sie übergeben dabei die von Ihnen geänderte Datei *aus_datei* als Eingabe an *exstr* und lenken die Ausgabe in die Datei *neu_datei.c* um, die dann folgenden neuen Quellcode enthält:

```
$ exstr -r datei.c < aus_datei > neu_datei.c
$ cat neu_datei.c
extern char *gettxt();
main()
{
   printf(gettxt("UX:1",""));
   printf(gettxt("UX:2",""));
}
```

Beim Aufruf mit Option *-d* wird an *gettxt()* als zweites Argument statt der leeren Zeichenkette "" die zu ersetzende Zeichenkette aus der ursprünglichen C-Quellprogramm-Datei übergeben. Zur Laufzeit des Programms wird dann diese Zeichenkette ausgegeben, falls der Aufruf von *gettxt()* scheitert:

```
$ exstr -rd datei.c < aus_datei > neu_datei.c
$ cat neu_datei.c
extern char *gettxt();
main()
{
   printf(gettxt("UX:1","Erste Zeichenkette\n"));
   printf(gettxt("UX:2","Zweite Zeichenkette\n"));
}
```

SIEHE AUCH

gettxt, *mkmsgs*, *printf*, *srchtxt*
gettxt(), *printf()*, *setlocale()* [19]
environ [7] [19]

extract
Zeichenketten in Quellprogrammen interaktiv suchen und ersetzen

Mit *extract* können Sie interaktiv aus Quellprogramm-Dateien Zeichenketten herausziehen und diese durch Aufrufe von Funktionen ersetzen, die zur Laufzeit des Programms aus einem Meldungskatalog passende Zeichenketten holen und dazubinden. Dieser Prozeß wird mit zwei Dateien gesteuert:

- Musterdatei
- Ignoredatei

Die Musterdatei enthält

1. Textmuster, mit denen Zeichenketten aus der angegebenen Quellprogramm-Datei verglichen werden und
2. Regeln, nach denen zu den Mustern passende Zeichenketten ausgewählt und ersetzt werden.

Die Ignoredatei enthält Zeichenketten, die nicht ersetzt werden sollen.

extract[option] ... datei ...

option

-i ignoredatei
Die Datei *ignoredatei* wird als Ignoredatei verwendet.

-i ignoredatei nicht angegeben:
Standardmäßig ist die Datei *ignore* die Ignoredatei. Sie wird zuerst im aktuellen Dateiverzeichnis, dann im *HOME*-Dateiverzeichnis und zuletzt in *usr/lib/nls/extract* gesucht.

-m präfix
Für *präfix* können Sie Zeichen oder Zeichenfolgen angeben, die allen Meldungsnummern in der *nl*-Datei und in der *.msf*-Datei vorangestellt werden sollen.

-m präfix nicht angegeben:
Die Meldungsnummern bekommen keine Zeichen vorangestellt.

-n
Für jede angegebene Quellprogramm-Datei wird eine eigene Meldungstext-Datei erstellt. Die Meldungsnummern beginnen in jeder Meldungsdatei mit 1.

-n nicht angegeben:
Es wird eine Meldungstext-Datei erzeugt. Die Meldungsnummern werden fortlaufend numeriert.

-p␣musterdatei
Die Datei *musterdatei* wird als Musterdatei verwendet.

-p␣musterdatei nicht angegeben:
Standardmäßig ist die Datei *pattern* die Musterdatei. Sie wird zuerst im aktuellen Dateiverzeichnis, dann im *HOME*-Dateiverzeichnis und zuletzt in *usr/lib/nls/extract* gesucht.

-s␣zeichenkette
Für *zeichenkette* können Sie Zeichen angeben, die am Anfang der *.msf*-Datei ausgegeben werden sollen.

-s␣zeichenkette nicht angegeben:
Standardmäßig wird die Zeichenkette *$quote "* ausgegeben oder die *CATHEAD*-Anweisung aus der Musterdatei, sofern sie dort definiert ist.

-u
Es wird eine Meldungsdatei verwendet, die durch einen früheren Lauf von *strextract* erstellt wurde. Für jede angegebene Quellprogramm-Datei wird im aktuellen Dateiverzeichnis nach einer entsprechenden Datei mit dem Suffix *.msg* gesucht. Diese Datei enthält genaue Informationen über alle Zeichenketten, die zu Mustern in der Musterdatei passen, u.a. ihre Position und Zeilennummer in der Datei. *extract* ruft *strextract* auf und interpretiert das Ergebnis.

datei
Name der Quellprogramm-Datei, die *extract* bearbeiten soll. Sie können auch mehrere Dateien angegeben.

Die Musterdatei

Die Musterdatei enthält Textmuster, mit denen die Zeichenketten in der angegebenen Quellprogramm-Datei verglichen werden, und die Regeln, nach denen die Zeichenketten ausgewählt und ersetzt werden. Außerdem können Sie Textmuster angeben für Zeichenketten, die nicht ersetzt werden sollen. Dies ist nötig, um zu verhindern, daß Zeichenketten unzulässigerweise verändert werden und dadurch ungültiger Code produziert wird.

Die Syntax von Textmustern stimmt mit der Syntax von regulären Ausdrücken bei *ed* überein. Im Beispiel wird gezeigt, wie eine Musterdatei für die Anwendung auf C-Quellprogramme aussehen kann.

Die Musterdatei ist in mehrere Abschnitte unterteilt, wobei jeder Abschnitt durch den Eintrag *$SCHLÜSSELWORT* eindeutig identifiziert werden kann.

$SCHLÜSSELWORT kann sein:

$SRCHEAD1
Include-Datei, die am Anfang oder in die erste neue Quellprogramm-Datei eingefügt wird.

$SRCHEAD2
Include-Datei, die an den Beginn der zweiten neuen Quellprogramm-Datei eingefügt wird.

$CATHEAD
Header, der an den Anfang des zu erstellenden Meldungskatalogs eingefügt wird. Sie könnten hier beispielsweise die SCCS-ID, die Definition des aktuellen Zeichensatzes o.ä. einfügen. Unter Umständen wird dies jedoch überschrieben, wenn Sie *extract* mit der Option *-s* aufrufen.

$REWRITE
Regel, nach der ausgewählte passende Zeichenketten ersetzt werden.

$MATCH
Liste der Muster der gesuchten Zeichenketten. Die Syntax der regulären Ausdrücke basiert auf *ed*. Beachten Sie: [^xyz]* bedeutet, daß bis auf xyz jedes Zeichen als passend gilt.

$REJECT
Bestimmte C-Konstrukte sollen nicht berücksichtigt werden.

$ERROR warnung
Für initialisierte Zeichenketten wird eine Warnung ausgegeben.

Die Ignoredatei

Die Ignoredatei enthält wörtlich die Zeichenketten (also keine Muster in der Form regulärer Ausdrücke), die in der neuen Quellprogramm-Datei (der *nl_*Datei) unverändert bleiben. Die Ignoredatei wird, falls vorhanden, zu Beginn des Ablaufs von *extract* gelesen. Sie sorgt dafür, daß bestimmte Zeichenketten in der Eingabedatei ignoriert werden. Während des Laufs können mit dem *ADD*-Kommando (siehe unten) weitere Zeichenketten, die ignoriert werden sollen, in die Datei eingefügt werden. Sie können die Ignoredatei mit einem normalen Editor erstellen und ändern, wobei in jeder Zeile genau eine Zeichenkette stehen darf, die in der Eingabedatei ignoriert werden soll.

Arbeitsweise von extract

Beim Aufruf von *extract* wird eine neue Version der Quellprogramm-Datei erstellt, wobei dem ursprünglichen Dateinamen die Zeichen *nl_* vorangestellt werden. Außerdem wird eine Meldungstext-Datei erstellt. Ihr Name ergibt sich aus dem Namen der ersten angegebenen Datei ohne die Endung *.c* und den Zeichen *.msf*, die als Suffix angehängt werden. Diese Datei kann direkt als Eingabe für *gencat* dienen oder ihr Inhalt kann vor der Übergabe an *gencat* in eine andere Landessprache übersetzt werden.

Während des Laufs von *extract* erscheinen am Bildschirm drei Fenster. Das erste enthält den Quellcode mit der Zeichenkette, die herausgezogen und umgewandelt werden soll. Sie wird auf dem Bildschirm invers dargestellt.

Das zweite Fenster enthält eine Liste der bisher herausgezogenen Zeichenketten mit jeweils einer Meldungsnummer.

Das dritte Fenster enthält eine Liste der verfügbaren Kommandos. Das sind:

EXTRACT
Aus der Quelldatei wird die passende Zeichenkette herausgezogen und in die Katalog-Quelldatei geschrieben. In der Musterdatei wird nach der *REWRITE*-Regel festgelegt, welcher Text den herausgezogenen Text in der *nl_*Datei ersetzen soll.

DUPLICATE
Wenn dieselbe Zeichenkette bereits früher herausgezogen wurde, hebt *extract* diese am Bildschirm in inverser Darstellung hervor.

Wenn Sie dann das Kommando *DUPLICATE* eingeben, verwendet *extract* beim Neuschreiben des Quellprogramms dieselbe Meldungsnummer wie beim letztenmal. Dadurch kann in Meldungskatalogen Speicherplatz gespart werden.

IGNORE
Die Zeichenkette soll hier und bei jedem nachfolgenden Auftreten ignoriert werden.

PASS
Die Zeichenkette soll hier ignoriert werden. Bei jedem nachfolgenden Auftreten derselben Zeichenkette wird sie dem Benutzer jedesmal neu zur Entscheidung angeboten.

ADD
Wie *IGNORE*, aber zusätzlich wird die Zeichenkette an den Inhalt der Ignoredatei angefügt.

COMMENT
Sie können einen Kommentar angeben, der dann in die Meldungstext-Datei (Endung *.msf*) eingetragen wird. Dies ist nützlich für *gencat*.

QUIT
Sie verlassen das Programm, werden aber vorher gefragt, ob Sie es wirklich verlassen wollen. Die Ausgabedateien enthalten das bis zum Zeitpunkt des Verlassens erreichte Ergebnis von *extract*. Es ist nicht möglich, das Kommando *extract* erneut über eine bereits teilweise mit *extract* bearbeitete Quelldatei laufen zu lassen.

HELP
Eine Beschreibung der möglichen Kommandos wird ausgegeben. Die Hilfstexte stehen in */usr/lib/nls/extract/help*.

Das aktuelle Kommando wird am Bildschirm invers dargestellt und durch das Drücken der Returntaste ↵ gestartet. Ein neues Kommando wählen Sie aus, indem Sie den ersten Buchstaben des Kommandos eingeben, z.B. *i* oder *I* für *IGNORE*.

Einschränkungen

- Bei der aktuell gültigen Syntax für die Musterdatei ist es nicht möglich, Zeichenketten in mehrzeiligen Kommentaren zu ignorieren.
- Für alle Klassen von passenden Mustern kann nur eine Zeichenkette zum Ersetzen definiert werden.
- Das Programm arbeitet nicht rekursiv alle Include-Dateien ab. Diese müssen eigens bearbeitet werden.
- Mehrzeilige Zeichenketten werden nicht erkannt.

BEISPIELE

1. Bearbeitung des Quellprogramms *datei1.c* mit *extract*, wobei *neuignore* als Ignoredatei und *c_muster* als Musterdatei verwendet werden sollen.

```
$ extract -i neuignore -p c_muster datei1.c
```

Es werden folgende Dateien erstellt:
Neue Version der Quellprogramm-Datei: *nl_datei1.c*
Meldungstext-Datei: *datei1.msf*

```
$ ls *datei1*
datei1.c
datei1.msf
nl_datei1.c
```

2. Erzeugen des Meldungskatalogs *datei1.cat* für *datei1.c* aus der Meldungstext-Datei *datei1.msf* mit *gencat*.

```
$ gencat datei1.cat datei1.msf
```

3. Kompilieren der mit *extract* neu erstellten Quelldatei *nl_datei1.c*.

```
$ cc nl_datei1.c
```

4. Beispiel einer Musterdatei für ein C-Quellprogramm:

```
# Kommentarzeilen beginnen mit dem Nummernzeichen #, entwerten kann
# man das Nummernzeichen mit einem Gegenschrägstrich davor \#
#
# Zuerst müssen einige Abschnitte in jede neu zu erstellende
# Quelldatei eingebunden werden.
#
# Include-Datei, die an den Anfang der ersten neuen
# Quellprogramm-Datei eingefügt wird.
$SRCHEAD1
\#include <nl_types.h>
nl_catd catd;           /* Definition des Katalog-Deskriptors       */
# Include-Datei, die an den Anfang der zweiten neuen
# Quellprogramm-Datei eingefügt wird.
$SRCHEAD2
\#include <nl_types.h>
extern nl_catd catd;    /* Katalog-Deskriptor                       */
#
# Header, der an den Anfang des zu erstellenden Meldungskatalogs
# eingefügt werden muß. Sie könnten hier beispielsweise die
# SCCS-ID, die Definition des aktuellen Zeichensatzes o.ä.
# einfügen. Unter Umständen wird dies jedoch überschrieben,
# wenn Sie extract mit der Option -s aufrufen.
#
$CATHEAD
\$quote "
\$set 1
#
# Regel, nach der ausgewählte passende Zeichenketten ersetzt
# werden.
$REWRITE
catgets(catd, 1, %n, %t)
# Die %-Deskriptoren werden wie folgt ersetzt:
# %  - a %
# n  - Meldungsnummer (aus der cat-Datei)
# t  - eigentlicher Text
# l  - Länge der Zeichenkette %t
# r  - roher Text ohne Quotier-Zeichen
# N  - ein Neue-Zeile-Zeichen wird eingefügt
# T  - ein Tabulatorzeichen wird eingefügt
#
# Liste der Muster der gesuchten Zeichenketten.
# Die Syntax der regulären Ausdrücke basiert auf ed.
# Beachten Sie: [^xyz]* bedeutet, daß jedes Zeichen als
# passend gilt außer xyz.
```

```
#
$MATCH
#
"[^"]*"
# Bestimmte C-Konstrukte sollen nicht berücksichtigt werden:
$REJECT
# Die leere Zeichenkette
""
# Zeichen zur Ausgabesteuerung
"\\."
# Präprozessor-Anweisungen sollen ignoriert werden
\#.*
# SCCS-ID-Zeilen sollen ignoriert werden
[sS][cC][cC][sS][iI][dD]\[\][       ]*=[    ]*".*"
# Einige Bibliotheksfunktionen, die Dateinamen verwenden, sollen
# ignoriert werden
fopen[  ]*([^,]*,[^)]*)
creat[  ]*([^,]",[^)]*)
#
# Für initialisierte Zeichenketten soll eine Warnung ausgegeben werden.
#
$ERROR initialisierte Zeichenketten können nicht ersetzt werden.
# Die auf ERROR folgende Meldung soll ausgegeben werden, wenn folgende
# Zeichenketten gefunden werden:
^char[^=]*=[     ]*"[^"]*"
char[    ]*[A-Za-z][A-Za-z0-9]*\[[^\]*\][  ]*=[   ]*"[^"]*"
```

SIEHE AUCH

ed, gencat

Kapitel 3, Internationale Umgebung - NLS

factor
Zahl in ihre Primfaktoren zerlegen

Das Kommando *factor* zerlegt Zahlen in Primfaktoren. Sie können *factor* mit oder ohne Argument aufrufen.

- Ohne Angabe eines Arguments (Format 1):
  Sie können dann interaktiv Zahlen eingeben. *factor* zerlegt diese Zahlen in ihre Primfaktoren und gibt diese aus.
- Mit Angabe eines Arguments (Format 2):
  factor zerlegt das Argument in seine Primfaktoren und gibt diese aus.

factor	Format 1
factor␣zahl	Format 2

Format 1: Interaktives Zerlegen

factor

Sie geben interaktiv Zahlen ein. *factor* zerlegt diese Zahlen in ihre Primfaktoren und gibt jeden Primfaktor so oft aus, wie er in der Zahl vorkommt.
Wenn Sie die Zahl 0 oder ein nicht-numerisches Zeichen eingeben, wird die Ausführung von *factor* beendet.

Format 2: Beim Aufruf angegebene Zahl zerlegen

factor␣zahl

zahl
: *zahl* muß eine positive Dezimalzahl $<= 10^{14}$ sein. *factor* zerlegt *zahl* in Primfaktoren. Dann wird zunächst *zahl* und anschließend jeder Primfaktor ausgegeben, so oft wie er in *zahl* vorkommt. Sie können *zahl* auch in der wissenschaftlichen Schreibweise angeben, d.h. etwa 1420 als 1.42e3 oder als 1.42E3.
Wenn Sie für *zahl* die Zahl 0 oder ein nicht-numerisches Zeichen angeben, erfolgt die Ausgabe 0.

FEHLERMELDUNGEN

`Ouch!`

Die interaktiv oder beim Aufruf von *factor* angegebene Zahl liegt nicht im zulässigen Bereich. Sie dürfen nur positive Zahlen angeben, die kleiner oder gleich 10^{14} sind.

`Not an integer!`

Sie haben interaktiv oder beim Aufruf ein nicht-ganzzahliges Argument angegeben.

BEISPIELE

1. Interaktives Zerlegen einer Zahl in Primfaktoren:

```
$ factor ↵
123 ↵
      3
      41
77777777777777 ↵
Ouch!
1234 ↵
      2
      617
0 ↵
$
```

2. Nicht-interaktives Zerlegen einer Zahl in Primfaktoren:

```
$ factor 100 ↵
100
      2
      2
      5
      5
```

false
Ende-Status ungleich 0 zurückgeben

false gibt einen Ende-Status ungleich 0 zurück und tut sonst nichts. Sie können damit in Shell-Prozeduren die Bedingung *falsch* erzeugen.
Analog erzeugen Sie mit dem Kommando *true* die Bedingung *wahr* (Ende-Status gleich 0).

false

ENDE-STATUS

ungleich 0

BEISPIELE

1. Der Ende-Status von *false* ist:

```
$ false
$ echo $?
1
```

2. Mit der folgenden Shell-Prozedur wird eine Endlosschleife gestartet, die Sie z.B. mit der Taste DEL abbrechen können:

```
until false
do
.
.
.
done
```

SIEHE AUCH

sh, *true*

fgrep
Zeichenketten suchen (fast grep)

fgrep liest Zeilen aus einer oder mehreren Dateien oder von der Standard-Eingabe und durchsucht die Zeilen nach Zeichenketten. Ist mittels Optionen nichts anderes angegeben, so schreibt *fgrep* alle Zeilen, die eine der angegebenen Zeichenketten enthalten, auf die Standard-Ausgabe.
Geben Sie mehrere Eingabedateien an, dann wird jeder Ausgabezeile der Name der betreffenden Datei vorangestellt.

fgrep[␣option]...[␣liste][␣datei]...

Die Zeichenketten, nach denen *fgrep* suchen soll, geben Sie entweder über *liste* oder über die Option *-f listendatei* an. Eines dieser beiden Argumente müssen Sie angeben, beide zusammen sind nicht erlaubt.

Keine Option angegeben
: *fgrep* gibt alle Zeilen aus, die mindestens eine der in *liste* angegebenen Zeichenketten enthalten. Geben Sie mehrere Eingabedateien an, dann wird jeder Ausgabezeile der Name der Datei vorangestellt, aus der die Zeile gelesen wurde.

option

-b
: (b - block) Jeder Ausgabezeile wird die Nummer des Blocks vorangestellt, in dem sie enthalten ist.
Die Blöcke, aus denen eine Datei besteht, sind je 512 byte groß und werden, mit 0 beginnend, durchnumeriert.
Option *-b* kann hilfreich sein, wenn die Nummern von Blöcken nach dem Kontext ermittelt werden sollen (siehe z.B. das Kommando *od*, Argument *offset*).

-c
: (c - count) *fgrep* gibt nur die Anzahl der gefundenen Zeilen aus (das sind die Zeilen, die *fgrep* ohne die Option *-c* ausgeben würde, siehe *Beispiel 4*); die Zeilen selbst werden nicht ausgegeben.

-e␣musterliste
: (e - expression) Diese Option brauchen Sie, wenn der erste Ausdruck in *musterliste* mit einem Bindestrich - beginnt. Zusammen mit *-e* wird eine solche Musterliste nicht als Option interpretiert, sondern als Liste von Mustern, mit denen *fgrep* die Eingabezeilen vergleichen soll.

-f␣listendatei
: (f - file) *fgrep* liest die Suchzeichenketten aus der Datei *listendatei*. Jede Zeile von *listendatei* wird als eine Zeichenkette interpretiert.

-i

(i - ignore) *fgrep* unterscheidet beim Vergleich nicht zwischen Groß- und Kleinbuchstaben.

-h

(h - hidden) Beim Durchsuchen mehrerer Eingabedateien unterläßt *fgrep* die Voranstellung der Dateinamen vor jeder Ausgabezeile.

-l

(l - list) *fgrep* gibt nur die Namen der Dateien aus, die mindestens eine der gefundenen Zeilen enthalten (das sind die Zeilen, die *fgrep* ohne die Option *-l* ausgeben würde, siehe *Beispiel 5*). Jeder Dateiname wird nur einmal ausgegeben. Die Zeilen selbst gibt *fgrep* nicht aus.

-n

(n - number lines)
Jeder Ausgabezeile wird die Zeilennummer aus der betreffenden Eingabedatei vorangestellt, wobei von 1 an numeriert wird. Liest *fgrep* von der Standard-Eingabe, bezieht sich die Zeilennummer auf die Standard-Eingabe.

-v

(v - vice versa)
fgrep gibt alle Zeilen aus, die *keine* der angegebenen Zeichenketten enthalten.

Zusammen mit Option *-c*:
fgrep gibt nur die Anzahl solcher Zeilen aus.

Zusammen mit Option *-l*:
fgrep gibt nur die Namen der Dateien aus, die solche Zeilen enthalten.

-x

(x - exact) *fgrep* gibt nur solche Zeilen aus, die eine der angegebenen Zeichenketten und sonst keine weiteren Zeichen enthalten.

Zusammen mit Option *-c*:
fgrep gibt nur die Anzahl solcher Zeilen aus.

Zusammen mit Option *-l*:
fgrep gibt nur die Namen der Dateien aus, die solche Zeilen enthalten.

liste

Liste von Zeichenketten, mit denen *fgrep* die Eingabezeilen vergleichen soll. Die Zeichenketten trennen Sie durch Neue-Zeile-Zeichen. Enthält *liste* Neue-Zeile-Zeichen oder sonstige Zeichen, die für die Shell eine Sonderbedeutung haben, dann schließen Sie *liste* in Hochkommas ein: *'liste'*.

datei
Name der Datei, die *fgrep* durchsuchen soll. Pro Aufruf können Sie mehrere Dateinamen angeben.

datei nicht angegeben:
fgrep liest die Eingabezeilen von der Standard-Eingabe.

grep, fgrep und egrep

Die Kommandos *grep*, *fgrep* und *egrep* sind von der Oberfläche her weitgehend identisch. Im folgenden sind die wichtigsten Unterschiede zwischen diesen Kommandos aufgeführt.

grep verarbeitet einfache reguläre Ausdrücke. Pro Aufruf können Sie nur einen einzigen regulären Ausdruck angeben.

fgrep verarbeitet nur Zeichenketten. Pro Aufruf können Sie jedoch mehrere Zeichenketten angeben: Die Zeichenketten geben Sie entweder direkt in der Aufrufzeile, getrennt durch Neue-Zeile-Zeichen, an oder Sie übergeben sie in einer Datei.
fgrep kann effizient sehr viele Zeichenketten suchen: *fgrep* sucht jede einzelne Zeile nach allen Zeichenketten ab.

egrep verarbeitet erweiterte reguläre Ausdrücke. Diese umfassen u.a. die einfachen regulären Ausdrücke bis auf eine Ausnahme: Der einfache reguläre Ausdruck \(*regausdruck*\) hat bei erweiterten regulären Ausdrücken keine Sonderbedeutung und wird deshalb auch nicht von *egrep* verarbeitet.
Pro Aufruf können Sie mehrere reguläre Ausdrücke, durch Neue-Zeile-Zeichen getrennt, angeben. *egrep* interpretiert diese Neue-Zeile-Zeichen wie einen senkrechten Strich (Zeichen für die Alternative, siehe *Tabellen und Verzeichnisse, Reguläre Ausdrücke*).
Die regulären Ausdrücke geben Sie entweder direkt in der Aufrufzeile an oder Sie übergeben sie in einer Datei.

ENDE-STATUS

0 Zeilen gefunden
1 keine Zeile gefunden
2 Syntaxfehler oder "Datei kann nicht geöffnet werden". Dieser Ende-Status gilt auch dann, wenn in anderen Eingabedateien Zeilen gefunden wurden.

BEISPIELE

Grundlage für die Beispiele sind die Dateien *kunden1* und *kunden2*. Sie haben folgenden Inhalt:

kunden1:

```
080685    999.98  20 LE  Art.  038   Fa. Holzinger
120387   1240.25   3 LE  Art.  023   Fa. Wanninger
180588    330.87   1 LE  Art.  332   Fa. Wanninger
```

kunden2:

```
hinterhuber berta, rosenheim, zugspitzstr.1
wanninger herbert, muenchen 5, kirschstr.3
```

1. Zeilen ausgeben, die eine bestimmte Zeichenkette enthalten (keine Option und Option *-i*):

```
$ fgrep Wanninger kunden1 kunden2
kunden1:120387   1240.25   3 LE  Art.  023   Fa. Wanninger
kunden1:180588    330.87   1 LE  Art.  332   Fa. Wanninger
```

   Wenn Sie auch Zeilen mit kleingeschriebenem *wanninger* ausgeben lassen möchten, geben Sie ein:

```
$ fgrep -i wanninger kunden1 kunden2
kunden1:120387   1240.25   3 LE  Art.  023   Fa. Wanninger
kunden1:180588    330.87   1 LE  Art.  332   Fa. Wanninger
kunden2:wanninger herbert, muenchen 5, kirschstr.3
```

2. Mehrere Zeichenketten suchen (keine Option und Option *-f*):

```
$ fgrep 'Wanninger↵
> Holzinger' kunden1 kunden2
kunden1:080685    999.98  20 LE  Art.  038   Fa. Holzinger
kunden1:120387   1240.25   3 LE  Art.  023   Fa. Wanninger
kunden1:180588    330.87   1 LE  Art.  332   Fa. Wanninger
```

   Sie können die beiden Zeichenketten auch in eine Datei *namen* schreiben (je Zeichenkette eine Zeile) und *fgrep* folgendermaßen aufrufen:

```
$ fgrep -f namen kunden1 kunden2
```

3. Zeilen ausgeben, die bestimmte Zeichenketten nicht enthalten (Option *-v*):

```
$ fgrep -v Wanninger kunden1 kunden2
kunden1:080685    999.98  20 LE  Art.  038   Fa. Holzinger
kunden2:hinterhuber berta, rosenheim, zugspitzstr.1
kunden2:wanninger herbert, muenchen 5, kirschstr.3
```

4. Anzahl der gefundenen Zeilen ausgeben (Option *-c*):

   Zuerst soll für jede Eingabedatei die Anzahl der Zeilen, die die Zeichenkette *Wanninger* enthalten, ausgegeben werden.

```
$ fgrep -c Wanninger kunden1 kunden2
kunden1:2
kunden2:0
```

   Nun soll die Anzahl der Zeilen, die die Zeichenkette *Wanninger* nicht enthalten, ausgegeben werden.

```
$ fgrep -c -v Wanninger kunden1 kunden2
kunden1:1
kunden2:2
```

5. Nur Dateinamen ausgeben (Option *-l*):

   Zuerst sollen die Namen der Dateien, die die Zeichenkette *Wanninger* enthalten, ausgegeben werden.

```
$ fgrep -l Wanninger kunden1 kunden2
kunden1
```

   Nun sollen die Namen der Dateien, die Zeilen ohne die Zeichenkette *Wanninger* enthalten, ausgegeben werden.

```
$ fgrep -l -v Wanninger kunden1 kunden2
kunden1
kunden2
```

6. Gefundene Zeilen mit Zeilennummer ausgeben (Option *-n*):

```
$ fgrep -n -i wanninger kunden1 kunden2
kunden1:2:120387   1240.25   3 LE  Art.  023   Fa. Wanninger
kunden1:3:180588    330.87   1 LE  Art.  332   Fa. Wanninger
kunden2:2:wanninger herbert, muenchen 5, kirschstr.3
```

SIEHE AUCH

ed, *grep*, *egrep*, *sed*, *sh*

file
Art einer Datei bestimmen

file überprüft Dateien, um die Art der darin enthaltenen Informationen zu ermitteln. Auf die Standard-Ausgabe schreibt *file*, von welchem Typ der Dateiinhalt ist. *file* unterscheidet z.B. Dateiverzeichnisse, Gerätedateien, FIFO-Dateien, Bibliotheken, C-Quellprogramme, ausführbare Programme, Shell-Prozeduren und normale Textdateien.

Vorsicht

file führt heuristische Tests durch, um den Typ des Dateiinhaltes zu ermitteln; es liefert daher nicht immer die korrekten Ergebnisse.

file[␣**-h**][␣**-m**␣magicfile][␣**-f**␣fdatei][␣datei␣...]	Format 1
file␣**-c**[␣**-m**␣magicfile]	Format 2

Format 1: Art einer Datei bestimmen

file[␣**-h**][␣**-m**␣magicfile][␣**-f**␣fdatei][␣datei␣...]

-h
(h - hidden) Falls die zu überprüfende Datei ein symbolischer Verweis ist, so wird diesem nicht nachgegangen. *file* gibt in diesem Fall folgende Meldung aus:

datei1: symbolic link to *datei2*

-m␣magicfile
(m - magic) *file* benutzt statt der Systemdatei */etc/magic* die Datei *magicfile*, um die Dateiformatkennungen (magic numbers) der zu überprüfenden Dateien zu bestimmen.

-f␣fdatei
file interpretiert das Argument *fdatei* als Namen einer Datei, die die Namen aller zu überprüfenden Dateien enthält. Wenn Sie diese Option nicht angeben, dann müssen Sie wenigstens eine zu überprüfende Datei *datei* angeben.

datei
Name der Datei, die *file* überprüfen soll. Falls die Datei ein symbolischer Verweis ist, so folgt *file* diesem Verweis und wertet die Ursprungsdatei aus. Pro Aufruf können Sie mehrere Dateinamen angeben. Wenn Sie die Option *-f* nicht angeben, dann müssen Sie wenigstens eine zu überprüfende Datei *datei* angeben.

Ausgabe

file schreibt den Typ des Dateiinhalts auf die Standard-Ausgabe. In der folgenden Tabelle sind die wichtigsten Klassifikationen des *file*-Kommandos aufgelistet.

Ausgabe	Bedeutung
ASCII cpio archive	Bibliothek, erzeugt mit *cpio*, Option *-c*
ascii text	Textdatei
assembler program text	Assembler-Quellprogramm
block special	blockorientierte Gerätedatei
c program text	C-Quellprogramm
character special	zeichenorientierte Gerätedatei
commands text	Shell-Prozedur
compressed data	komprimierte Datei (siehe *compress*)
cpio archive	Bibliothek, erzeugt mit *cpio*
current ar archive	Bibliothek (siehe *ar*)
data	Datei mit Daten (spezielle Fortran- und Assembler-Dateien)
directory	Dateiverzeichnis
ELF 32-bit	ELF-Objektdatei, mögliche Typen:
core file	Speicherauszugsdatei
dynamic lib	dynamische Bibliothek
executable	Datei ausführbar
relocatable	Datei relokierbar
unknown	Type unbekannt
empty file	leere Datei
English text	Datei mit englischem Text
fifo	FIFO-Datei
fortran program text	FORTRAN-Quellprogramm
[nt]roff, tbl, or eqn input text	Datei in *nroff-*, *troff-*, *tbl-* oder *eqn*-Format
packed data	komprimierte Datei (siehe *pack*)
sccs	SCCS-Datei
troff output	*troff*-Ausgabedatei

Ausführbare Programme werden von *file* zusätzlich klassifiziert. Wurde z.B. die Symboltabelle nicht entfernt, so gibt *file* aus: *not stripped* (siehe *strip* [19]).

Arbeitsweise

file führt für jede Eingabedatei eine Reihe von Tests durch und versucht so, den Dateiinhalt zu klassifizieren. Handelt es sich wahrscheinlich um eine Textdatei, dann prüft *file* deren Anfang (die ersten 512 byte) und stellt Vermutungen an über die Sprache des Textes. Die Richtigkeit der Ausgabe ist jedoch nicht gewährleistet.

Enthält die Eingabedatei ein ausführbares Programm, dann erkennt *file* dies und gibt weitere Informationen über den Dateiinhalt aus. Dazu durchsucht *file* die Datei nach sogenannten magic numbers, d.h. nach numerischen Konstanten oder Zeichenkettenkonstanten, die Aufschluß über die Art des Dateiinhaltes geben. Eine Erläuterung zu diesen magic numbers steht in der Datei */etc/magic*.

Format 2: Überprüfung der magicfile

file␣-c[␣-m␣magicfile]

-c
(c - check) Standardmäßig überprüft *file* die Systemdatei */etc/magic* auf Formatfehler. Ist die Option *-m* angegeben, dann wird stattdessen die Datei *magicfile* überprüft.

-m␣magicfile
(m - magicfile) *file* überprüft die Datei *magicfile* auf Formatfehler. Aus Effizienzgründen wird diese Option normalerweise nicht ausgeführt.

DATEI

/etc/magic
enthält Erläuterungen zu den magic numbers.

BEISPIEL

Die Datei *liste* enthält die folgenden Dateinamen:

```
dvz
brief
lib.a
prog.c
prog.s
prog.o
prog
```

Mit folgendem Aufruf erhalten Sie Informationen über die Art des Dateiinhalts:

```
$ file -f liste
dvz:            directory
brief:          ascii text
lib.a:          current ar archive
prog.c:         c program text
prog.s:         assembler program text
prog.o:         ELF 32-bit LSB relocatable 80386 Version 1
prog:           ELF 32-bit LSB executable 80386 Version 1
```

dvz ist also ein Dateiverzeichnis, *brief* enthält normalen ASCII-Text. *lib.a* ist eine Bibliothek, *prog.c* enthält ein C-Quellprogramm, *prog.s* ein Assembler-Quellprogramm (ohne Gewähr!). *prog.o* ist ein Objektmodul, *prog* ein ausführbares Programm im *ELF-Format* (Executable and Linking Format).

Die gleiche Ausgabe erhalten Sie mit dem Aufruf

```
$ file dvz brief lib.a prog.c prog.s prog.o prog
```

SIEHE AUCH

cc, *strip* [19]
filehdr [7]

find
Dateiverzeichnisse durchsuchen

find durchsucht Dateiverzeichnisse und deren Unterdateiverzeichnisse nach Dateien, die angegebene Bedingungen erfüllen. Die gefundenen Dateien können Sie mit einem Kommando bearbeiten und/oder sich ihre Namen ausgeben lassen.
find sucht nicht nur nach einfachen Dateien, sondern auch nach Dateiverzeichnissen, Gerätedateien oder FIFOs.

find dateiverzeichnis ... ausdruck

dateiverzeichnis
Name des Dateiverzeichnisses, das durchsucht werden soll. *find* durchsucht dieses Dateiverzeichnis sowie alle seine Unterdateiverzeichnisse, sofern Sie für die Dateiverzeichnisse Lese- und Ausführrecht besitzen (r- und x-Bit). Sie können mehrere Namen angeben.

ausdruck
Für *ausdruck* geben Sie entweder einen der unten aufgeführten elementaren Ausdrücke an oder mehrere elementare Ausdrücke, die Sie miteinander verknüpfen.

find prüft *ausdruck* für jede Datei, die in einem der angegebenen Dateiverzeichnisse bzw. Unterdateiverzeichnisse steht, von links nach rechts ab. Ein elementarer Ausdruck ist für eine Datei entweder wahr oder falsch. Vom Wahrheitswert des Ausdrucks und dem darauffolgenden Verknüpfungsoperator hängt es ab, ob *find* den nächsten elementaren Ausdruck bearbeitet oder nicht.
Die meisten elementaren Ausdrücke sind Bedingungen, d.h., sie sind wahr, wenn die aktuelle Datei die Bedingung erfüllt. Die Ausdrücke

-print
-exec kommando \;
-ok kommando \;
-cpio gerät

sind keine Bedingungen. Sie bewirken, daß eine Aktion ausgeführt wird, z.B. Ausgeben des Dateinamens oder Löschen der Datei. Wenn *find* einen dieser Ausdrücke bearbeitet, wird die zugehörige Aktion auf jeden Fall ausgeführt, egal, ob der Ausdruck den Wert "wahr" oder "falsch" liefert. Wenn Sie keinen der obigen Ausdrücke angeben, bricht *find* mit einer Fehlermeldung ab.

Innerhalb von *ausdruck* müssen Sie Sonderzeichen der Shell entwerten, so daß die Shell sie nicht interpretiert, sondern an *find* übergibt.

Im folgenden sind die elementaren Ausdrücke beschrieben; weiter unten erfahren Sie, wie Sie elementare Ausdrücke miteinander verknüpfen.

Elementare Ausdrücke als Bedingungen

Wenn Sie Optionen angeben, um Dateien mit bestimmten Eigenschaften herauszufinden (z.B. Dateien, die innerhalb einer bestimmten Zeit verändert wurden), so muß eine solche Option vor der Option *-print* (siehe unten) angegeben werden. Ansonsten werden alle Dateien ausgegeben.

Der Parameter *n* steht für eine Dezimalzahl; *+n* bedeutet mehr als *n*, *-n* bedeutet weniger als *n*, und *n* bedeutet genau *n*.

-name datei
Wahr, wenn die aktuelle Datei den Namen *datei* hat.
Für *datei* können Sie die üblichen Sonderzeichen der Shell zur Dateinamen-Generierung verwenden, müssen dann aber *datei* in Hochkommas einschließen, z.B. find /usr/adam -name 'gruppe.*' (siehe *Tabellen und Verzeichnisse, Sonderzeichen der Bourne-Shell sh*). Sie dürfen nur einfache Dateinamen angeben (nicht erlaubt ist z.B. *text/tmp*).

-perm oktalzahl
Wahr, wenn die aktuelle Datei die Zugriffsrechte *oktalzahl* hat (siehe *chmod*). Geben Sie vor *oktalzahl* ein Minuszeichen an, so werden nur die Bits mit den Zugriffsrechten verglichen, die in *oktalzahl* gesetzt sind, und der Ausdruck ist wahr, wenn sie übereinstimmen.

-type zeichen
Wahr, wenn die aktuelle Datei vom Typ *zeichen* ist, wobei *zeichen* sein kann:

f	für einfache Datei (ordinary file)
d	für Dateiverzeichnis (directory)
b	für blockorientierte Gerätedatei (block special file)
c	für zeichenorientierte Gerätedatei (character special file)
l	für symbolische Verweise (symbolic links)
p	für FIFO (named pipe)

-fstype name
Wahr, wenn das Dateisystem, zu dem die aktuelle Datei gehört, vom Typ *name* ist, wobei *name* sein kann:

s5, *ufs*, *bfs*, *rfs* oder *nfs* (sowie einige Spezial-Dateisysteme wie *proc* oder *fdfs*).

-links [±]n
Wahr, wenn auf die aktuelle Datei (mehr als / weniger als) *n* Verweise bestehen.

-inum [±]n
Wahr, wenn der Index-Eintrag (inode) der aktuellen Datei größer, kleiner bzw. gleich *n* ist. Mit dem Ausdruck *-inum +n -inum -m* finden Sie alle Index-Einträge zwischen *n* und *m*.

-user␣benutzerkennung
Wahr, wenn die aktuelle Datei dem Benutzer *benutzerkennung* gehört. Ist *benutzerkennung* eine Zahl, die nicht in der Datei */etc/passwd* als Kennung eingetragen ist, so wird sie als Benutzernummer interpretiert.

-nouser
Wahr, wenn die aktuelle Datei einer Benutzerkennung gehört, die nicht in der Datei */etc/passwd* als Kennung eingetragen ist.

-group␣gruppenname
Wahr, wenn die aktuelle Datei der Gruppe *gruppenname* gehört. Ist *gruppenname* eine Zahl, die nicht in der Datei */etc/group* als Gruppenname eingetragen ist, so wird sie als Gruppennummer interpretiert.

-nogroup
Wahr, wenn die aktuelle Datei einer Gruppe gehört, die nicht in der Datei */etc/group* als Gruppe eingetragen ist.

-size␣[±]n[**c**]
c nicht angegeben:
Wahr, wenn die aktuelle Datei (mehr/weniger als) *n* Blöcke zu je 512 byte belegt.

c angegeben:
Wahr, wenn die aktuelle Datei (mehr/weniger als) *n* byte belegt.

-atime␣[±]n
(a - access)
Wahr, wenn auf die aktuelle Datei zuletzt vor (mehr als / weniger als) *n* * 24 Stunden zugegriffen wurde. Die Zugriffszeit der Dateiverzeichnisse vom bzw. im durchsuchten Dateiverzeichnis wird von *find* selbst verändert.

-mtime␣[±]n
(m - modification)
Wahr, wenn die aktuelle Datei zuletzt vor (mehr als / weniger als) *n* * 24 Stunden geändert wurde.

-ctime␣[±]n
Wahr, wenn der Index-Eintrag (inode) der aktuellen Datei zuletzt vor (mehr als / weniger als) *n* * 24 Stunden geändert wurde.

-newer␣datei
Wahr, wenn die aktuelle Datei jünger als die angegebene *datei* ist, d.h., wenn sie zu einem späteren Zeitpunkt erstellt oder geändert wurde.

-local
Wahr, wenn sich die aktuelle Datei physikalisch auf dem lokalen System befindet.

-prune
Immer wahr. Wird dieser Ausdruck bearbeitet, werden keine Dateien oder Dateiverzeichnisse durchsucht, die unterhalb der aktuellen Hierarchiestufe liegen.

Elementare Ausdrücke, die Aktionen bewirken

-print
Immer wahr. Wird dieser Ausdruck bearbeitet, gibt *find* den Pfadnamen der aktuellen Datei auf die Standard-Ausgabe aus.

-exec␣kommando␣**\;**
kommando wird ausgeführt, sobald dieser Ausdruck bearbeitet wird. *kommando* müssen Sie mit einem Leerzeichen und einem entwerteten Strichpunkt \; abschließen. Ein Paar geschweifter Klammern {} als Kommandoargument steht für den Namen der aktuellen Datei.

Der Ausdruck *-exec kommando* \; ist wahr, wenn *kommando* den Ende-Status 0 liefert. Der Wahrheitswert ist von Bedeutung, wenn dieser Ausdruck mit weiteren Ausdrücken verknüpft ist.

-ok␣kommando␣**\;**
Arbeitet wie *-exec*, nur fragt *find* jedesmal, wenn der Ausdruck bearbeitet wird, ob das Kommando ausgeführt werden soll. Das Kommando wird nur dann ausgeführt, wenn Sie mit dem Zeichen bzw. der Zeichenkette antworten, die in der aktuell gültigen Umgebung "Ja" bedeutet (siehe Umgebungsvariable *LANG*).

-cpio␣gerät
Immer wahr. Wird dieser Ausdruck bearbeitet, gibt *find* den Inhalt der aktuellen Datei im *cpio*-Format auf das angegebene *gerät* aus (siehe *cpio*).

Diese Option ist überholt. Sie wird in zukünftigen Versionen nicht mehr unterstützt.

Ausdrücke, die die Arbeitsweise von find beeinflussen

-depth
Immer wahr. Wird dieser Ausdruck bearbeitet, dann bearbeitet *find* die Einträge eines Dateiverzeichnisses eher als das Verzeichnis selbst. Dies kann hilfreich sein, wenn *find* in Verbindung mit dem Kommando *cpio* verwendet wird, um Dateien zu übertragen, für deren Verzeichnis Sie kein Schreibrecht besitzen.

-follow
Immer wahr. Wird dieser Ausdruck bearbeitet, dann wird symbolischen Verweisen nachgegangen. Dieser Ausdruck sollte nicht mit der Option *-type l* kombiniert werden.

Wenn Sie *find* zusammen mit dem Kommando *cpio* anwenden, dann beachten Sie folgendes: Wenn Sie *cpio* mit der Option *-L* aufrufen, müssen Sie bei *find* den Ausdruck *-follow* angeben und umgekehrt. Sonst können unerwünschte Effekte auftreten.

-mount
Immer wahr. Wird dieser Ausdruck bearbeitet, dann beschränkt *find* seine Suche auf das Dateisystem, in dem sich das angegebene Dateiverzeichnis befindet.

Ausdrücke verknüpfen

Die oben beschriebenen Ausdrücke können Sie wie folgt miteinander verknüpfen; beachten Sie die Leerzeichen vor und nach den Operatoren!
Die Verknüpfungs-Operatoren sind nach Priorität in absteigender Reihenfolge geordnet.

\(␣ausdruck...␣\)
Klammern fassen mehrere Ausdrücke zusammen. Die Klammern müssen mit einem Gegenschrägstrich \ entwertet werden, da sie eine Sonderbedeutung für die Shell haben.

!␣ausdruck
Negation: wahr, wenn der Ausdruck falsch ist.

ausdruck␣ausdruck
logisches UND: wahr, wenn beide Ausdrücke wahr sind. Ist der erste Ausdruck falsch, dann bearbeitet *find* den zweiten Ausdruck nicht.

Beispiel

Ist der zweite Ausdruck

-exec kommando \;

dann wird das Kommando nur dann ausgeführt, wenn der erste Ausdruck wahr ist.

ausdruck␣**-o**␣ausdruck
logisches ODER: wahr, wenn einer der Ausdrücke oder beide wahr sind. Ist der erste Ausdruck wahr, dann bearbeitet *find* den zweiten Ausdruck nicht.

Beispiel

Ist der zweite Ausdruck

-exec kommando \;

dann wird das Kommando nur dann ausgeführt, wenn der erste Ausdruck falsch ist.

FEHLERMELDUNGEN

Fehlermeldungen, die nicht zum Abbruch führen (Ende-Status 0)

`find: cannot open` *dvz*

Sie haben für das Dateiverzeichnis *dvz* kein Leserecht. Das Dateiverzeichnis kann folglich nicht durchsucht werden. *find* bricht jedoch nach dieser Fehlermeldung nicht ab, sondern durchsucht die Dateiverzeichnisse, für die Sie die erforderlichen Zugriffsrechte (r- und x-Bit) besitzen.

`find: cannot chdir to` *dvz*

Sie haben für das Dateiverzeichnis *dvz* kein "Ausführrecht" (x-Bit). Das Dateiverzeichnis kann folglich nicht durchsucht werden. *find* bricht jedoch nach dieser Fehlermeldung nicht ab, sondern durchsucht die Dateiverzeichnisse, für die Sie die erforderlichen Zugriffsrechte (r- und x-Bit) besitzen.

`find: bad status-- da`*tei* `find: bad status-- dv`*z*

Die Datei *datei* (das Dateiverzeichnis *dvz*) existiert nicht.

Fehlermeldungen, die zum Abbruch führen (Ende-Status ungleich 0)

`find: missing conjunction`

Sie haben beim *find*-Aufruf die elementaren Ausdrücke nicht richtig miteinander verknüpft oder bei einem elementaren Ausdruck mehrere Argumente angegeben.

`find: no action specified`

Sie haben keine Aktion angegeben (*print, exec, ok, cpio*).

`find: incomplete statement`

Sie haben ein Kommando bei *-exec* oder *-ok* nicht mit entwertetem Strichpunkt \; abgeschlossen.

`find: insufficient number of arguments`

Sie haben zu wenige Argumente angegeben.

`find: path-list predicate-list`

Sie haben vergessen, die Pfadliste anzugeben.

DATEIEN

/etc/group
Datei mit Gruppennamen

/etc/passwd
Datei mit Benutzerkennungen

BEISPIELE

1. Arbeitsweise von *find* - ein einfaches Beispiel:

   Die Pfadnamen aller Dateien ausgeben,
   - die im aktuellen Dateiverzeichnis oder einem Unterdateiverzeichnis stehen und
   - deren Name *tmp* ist.

   ```
   $ find . -name tmp -print
   ```

   Hier besteht *ausdruck* aus zwei elementaren Ausdrücken, die mit einem logischen UND verknüpft sind. Für jede Datei prüft *find* zuerst den elementaren Ausdruck *-name tmp* ab. Ist dieser Ausdruck wahr (d.h. hat die aktuelle Datei den Namen *tmp*), dann bearbeitet *find* auch den Ausdruck *-print*, d.h., *find* gibt den Pfadnamen aus. Andernfalls bearbeitet *find* den Ausdruck *-print* nicht, d.h., *find* gibt nichts aus.

   Beachten Sie die Reihenfolge: Wenn Sie stattdessen

   ```
   $ find . -print -name tmp
   ```

   eingeben, dann bearbeitet *find* den Ausdruck *-print* für jede Datei, d.h., *find* gibt die Pfadnamen aller Dateien aus.

2. Arbeitsweise von *find* - ein komplexeres Beispiel:

   Die Pfadnamen aller Dateien ausgeben,
   - die im aktuellen Dateiverzeichnis oder Unterverzeichnissen stehen und
   - deren Name *tmp* ist oder auf *.xx* endet.

   ```
   $ find . \( -name tmp -o -name '*.xx' \) -print
   ```

   Beachten Sie die Leerzeichen zwischen den Argumenten!

Für jede Datei bearbeitet *find* zuerst den Inhalt der Klammern. Dieser besteht aus zwei elementaren Ausdrücken, die mit einem logischen ODER verknüpft sind. *find* prüft zuerst den elementaren Ausdruck *-name tmp* ab. Ist dieser Ausdruck wahr (d.h. hat die aktuelle Datei den Namen *tmp*), dann bearbeitet *find* den Ausdruck *-name '*.xx'* gar nicht mehr: das Ergebnis des geklammerten Ausdrucks ist dann auf jeden Fall wahr. Ist *-name tmp* falsch, prüft *find* den zweiten elementaren Ausdruck *-name '*.xx'* ab; ist er wahr (d.h. endet der Dateiname auf *.xx*), dann ist auch der geklammerte Ausdruck wahr, andernfalls ist der geklammerte Ausdruck falsch.
Ist der geklammerte Ausdruck wahr, dann bearbeitet *find* den Ausdruck *-print*, d.h., *find* gibt den Dateinamen aus (UND-Verknüpfung - siehe *Beispiel 1* !). Ist der geklammerte Ausdruck falsch, dann bearbeitet *find* den Ausdruck *-print* nicht, d.h., *find* gibt nichts aus.

3. Alle Einträge im Dateiverzeichnis */usr/nikolaus* und den Unterverzeichnissen ausgeben, deren Eigentümer nicht *nikolaus* ist.

```
$ find /usr/nikolaus ! -user nikolaus -print
```

4. Alle Dateien löschen,
   - deren Name *tmp* ist oder auf *.xx* endet und
   - auf die mehr als 7 Tage lang nicht zugegriffen wurde.

   Vor dem Löschen soll abgefragt werden, ob die Datei tatsächlich gelöscht werden soll.

```
$ find / \( -name tmp -o -name '*.xx' \) -atime +7 -ok rm {} \;
```

5. Rekursives Drucken der Dateiverzeichnis-Struktur mit Auslassen aller SCCS-Dateiverzeichnisse.

```
$ find . -name SCCS -prune -o -print
```

SIEHE AUCH

chmod, *cpio*, *ln*, *sh*, *test*
stat() [19]
fs [7]

finger
Informationen über Benutzer am lokalen und fernen System ausgeben

finger gibt Informationen über alle aktuell am System angemeldeten Benutzer aus.

finger[␣option]...[␣benutzerkennung] Format 1
finger[␣-l]␣benutzerkennung@rechnername... Format 2

Format 1: Informationen über Benutzer am lokalen System ausgeben

finger[␣option]...[␣benutzerkennung]

Kein Argument angegeben
finger gibt standardmäßig in kurzer Ausgabeform Informationen über alle aktuell am System angemeldeten Benutzer aus.

Die kurze Ausgabeform enthält Informationen über:

- die Benutzerkennung
- den im Kommentarfeld der Datei */etc/passwd* verzeichneten Namen des Benutzers
- den Namen der Datensichtstation. Wenn kein Schreibrecht für die Datensichtstation besteht, wird der Ausgabe ein Stern * vorangestellt.
- den Zeitraum, in dem keine Eingaben gemacht wurden (idle time)
- den Zeitpunkt der Anmeldung am System
- den Namen des Rechners, von dem aus der Benutzer sich eingeloggt hat. Diese Information wird nur dann ausgegeben, wenn sich der Benutzer von einem fernen Rechner eingeloggt hat.

Der Zeitraum ohne Eingaben wird in folgenden Formaten ausgegeben:

n	Minuten
n:n	Stunden, Minuten
dn:n	Tage, Stunden

wobei *n* eine dezimale Ganzzahl ist.

Keine Option angegeben

Wenn Sie eine oder mehrere Benutzerkennungen angeben, gibt *finger* Informationen zu jeder definierten Benutzerkennung in langer Ausgabeform aus. Es spielt dabei keine Rolle, ob die entsprechenden Benutzer am System angemeldet sind oder nicht.

Die lange Ausgabeform umfaßt alle Informationen der kurzen Ausgabeform sowie zusätzliche Informationen über:

- das HOME-Dateiverzeichnis und die Login-Shell des Benutzers
- den Zeitpunkt, zu dem sich der Benutzer angemeldet hat. Ist der Benutzer aktuell nicht angemeldet, so bezieht sich die Information auf den Zeitpunkt der letzten Anmeldung.
- den Zeitpunkt, zu dem der Benutzer zuletzt elektronische Post (mail) erhalten hat sowie den Zeitpunkt, zu dem er sie zuletzt gelesen hat.
- die erste Zeile der Datei *$HOME/.plan*
- die erste Zeile der Datei *$HOME/.project*

option

-b

unterdrückt in der langen Ausgabeform die Ausgabe des *HOME*-Dateiverzeichnisses und der Login-Shell des Benutzers.

-f

unterdrückt in der kurzen Ausgabeform die Ausgabe der Kopfzeile.

-h

unterdrückt in der langen Ausgabeform die Ausgabe der Datei *$HOME/.project*.

-i

erzwingt eine Ausgabeform, die Informationen enthält über:
- die Benutzerkennung
- den Namen der Datensichtstation
- den Zeitpunkt der Anmeldung am System
- den Zeitraum, in dem keine Eingaben gemacht wurden

-l

erzwingt die lange Ausgabenform.

-m

Von den angegebenen Argumenten werden nur definierte Benutzerkennungen erkannt, aber keine Vor- und Nachnamen.

-p

unterdrückt in der langen Ausgabeform die Ausgabe der Datei *$HOME/.plan*.

-q
erzwingt eine Ausgabeform, die Informationen enthält über:
- die Benutzerkennung
- den Namen der Datensichtstation
- den Zeitpunkt der Anmeldung am System

-s
erzwingt die kurze Ausgabeform.

-w
unterdrückt in der kurzen Ausgabeform die Ausgabe des vollen Namens.

benutzerkennung
Name einer am lokalen System definierten Benutzerkennung, Vor- oder Nachname eines Benutzers oder Abrechnungsname.

Format 2: Informationen ausgeben über Benutzer an einem fernen System, das über ein TCP/IP-Netz mit dem lokalen System verbunden ist

finger[␣**-l**]␣benutzerkennung@rechnername...

-l
erzwingt die lange Ausgabeform.

benutzerkennung@rechnername
Für *benutzerkennung* geben Sie eine am fernen Rechner definierte Benutzerkennung ein, für *rechnername* den Namen eines fernen Rechners, der über ein TCP/IP-Netz mit dem lokalen System verbunden ist.

DATEIEN

/var/adm/utmp
Datei mit den Benutzerkennungen der aktuell am System angemeldeten Benutzer

/etc/passwd
Datei mit allen Benutzerkennungen. Im Kommentarfeld der Datei */etc/passwd* sind die Vor- und Nachnamen der Benutzer verzeichnet.

/var/adm/lastlog
Datei, in der der Zeitpunkt der letzten Anmeldung am System von jedem Benutzer gespeichert wird.

$HOME/.plan
Datei mit allen Plänen des jeweiligen Benutzers

$HOME/.project
Datei mit allen Projekten, an denen der jeweilige Benutzer mitarbeitet

BEISPIELE

1. Es sollen Informationen über alle momentan am System angemeldeten Benutzer ausgegeben werden. Da kein Argument angegeben wird, werden die Informationen in kurzer Ausgabeform dargestellt.

```
$ finger
Login       Name              TTY             Idle    When     Where
eddie            ???          term/tty002           Mon  08:02
arthur  Arthur Dent          *pts/0             4   Mon  08:25  sirius
jutta            ???          term/tty004        2   Mon  10:04
```

2. Sie wollen Informationen über den Benutzer *arthur* erhalten. Da eine *benutzerkennung* angegeben ist, werden die Informationen in langer Ausgabeform dargestellt.

```
$ finger arthur
Login name: arthur     (messages off)     In real life: Arthur Dent
Directory: /home2/arthur                  Shell: /sbin/sh
On since Jul 23 08:25 on pts/0             from sirius
4 minutes 18 seconds Idle Time
No unread mail
Project: Kommando-Projekt
Grosse Plaene
```

SIEHE AUCH

passwd, *who*, *whois*

flchk
Labelbereich einer Diskette überprüfen (floppy check)

flchk überprüft die Einträge in den Labelbereich einer eingelegten Diskette. Das Kommando vergleicht den Labelbereich der Diskette mit den Argumenten, die dem Kommando mitgegeben werden. Der Labelbereich befindet sich auf den ersten Spuren einer Diskette. Der Aufbau einer Diskette ist beim Kommando *flinit* beschrieben.

In den Labelbereich kann eingetragen sein:

- der Eigentümer der Diskette
- eine Versionsnummer.

flchk befindet sich im Dateiverzeichnis */usr/sbin*.

Der Eintrag in den Labelbereich einer Diskette kann mit dem Kommando *flinit* vorgenommen und mit *fldisp* ausgegeben werden.

/usr/sbin/flchk[␣**-o**␣eigentümer][␣**-v**␣versionsnummer][␣**-i**]

-o␣eigentümer
flchk vergleicht den angegebenen *eigentümer* mit dem auf der Diskette eingetragenen Eigentümer. Sind beide Eigentümer verschieden, fragt *flchk*, ob der Unterschied ignoriert werden soll. Antworten Sie mit nein, gibt *flchk* die Meldung

```
incorrect floppy inserted
```

aus und endet mit Ende-Status 1.

-v␣versionsnummer
flchk vergleicht die angegebene *versionsnummer* mit der auf der Diskette eingetragenen Versionsnummer. Sind beide Nummern verschieden, fragt *flchk*, ob der Unterschied ignoriert werden soll. Antworten Sie mit nein, gibt *flchk* die Meldung

```
incorrect floppy inserted
```

aus und endet mit Ende-Status 1.

-i
verändert das Verhalten des Kommandos mit den Optionen *-v* und/oder *-o* wie folgt:

Stimmen die angegebene Versionsnummer oder der angegebene Eigentümer nicht mit Einträgen auf der Diskette überein, beendet sich *flchk* ohne eine Abfrage sofort mit Fehlermeldung und Ende-Status 1.

Anwendung des flchk-Kommandos

flchk wird vom Menüsystem bei der Bearbeitung von Disketten benutzt. Mit dem Eintrag im Labelbereich kann ein Programm eine eingelegte Diskette identifizieren. Außerdem kann im Menüsystem nur der im Labelbereich eingetragene Eigentümer auf die Diskette zugreifen. So können Daten vor dem Zugriff anderer Benutzer geschützt werden, soweit dieser Zugriff über das Menüsystem erfolgt.

ENDE-STATUS

0 Der Eintrag oder die Einträge im Labelbereich der eingelegten Diskette stimmen überein mit den Argumenten, die dem Kommando mitgegeben wurden.

1 Ein Eintrag oder beide Einträge im Labelbereich der eingelegten Diskette stimmen nicht überein mit den Werten, die dem Kommando mitgegeben wurden.

BEISPIEL

Der Benutzer möchte überprüfen, ob die richtige Diskette im Laufwerk liegt. Der Besitzer soll *hans* sein, die Versionsnummer *1*.

```
$ flchk -v 1 -o hans
requested: VSN  000001  owner hans
existing:  VSN  000003  owner egon
Should the difference be ignored? (y/n) > n↵
flchk: Incorrect floppy inserted!
```

oder

```
$ flchk -v 1 -o hans -i
requested: VSN  000001  owner hans
existing:  VSN  000003  owner egon
flchk: Incorrect floppy inserted!
```

SIEHE AUCH

fldisp, *flinit*

fldisp
Labelbereich einer Diskette ausgeben (floppy display)

fldisp gibt die Einträge im Labelbereich einer eingelegten Diskette aus. Dieser Bereich befindet sich auf den ersten Spuren einer Diskette. Der Aufbau einer Diskette ist beim Kommando *flinit* beschrieben.

In den Labelbereich kann folgendes eingetragen und mit *fldisp* ausgegeben werden:

- der Eigentümer der Diskette
- eine Versionsnummer.

fldisp befindet sich im Dateiverzeichnis */usr/sbin*.

Einträge in den Labelbereich einer Diskette können mit *flinit* vorgenommen werden.

/usr/sbin/fldisp[_-o][_-v]

Keine Option angegeben
Der Name des Eigentümers und die Versionsnummer werden ausgegeben.

-o
Der Name des Eigentümers wird ausgegeben.

-v
Die Versionsnummer wird ausgegeben.

Anwendung des fldisp-Kommandos

fldisp wird vom Menüsystem bei der Bearbeitung von Disketten benutzt. Mit dem Eintrag im Labelbereich kann ein Programm eine eingelegte Diskette identifizieren. Außerdem kann im Menüsystem nur der im Labelbereich eingetragene Eigentümer auf die Diskette zugreifen. So können Daten vor dem Zugriff anderer Benutzer geschützt werden, soweit dieser Zugriff über das Menüsystem erfolgt.

BEISPIELE

1. Der Benutzer möchte sich den Labelbereich einer Diskette ausgeben lassen.

```
$ fldisp
VSN:   000001
owner: hans
```

   Der Eigentümer der Diskette ist *hans*, die Versionsnummer ist *1*.

2. Der Benutzer möchte wissen, wer Eigentümer einer Diskette ist.

```
$ fldisp -o
hans
```

   Der Eigentümer der Diskette ist *hans*.

SIEHE AUCH

flchk, *flinit*

flinit
in den Labelbereich einer Diskette schreiben (floppy initialisieren)

flinit beschreibt den Labelbereich einer im Laufwerk 0 liegenden Diskette. Der Labelbereich befindet sich am Anfang einer Diskette (Spur 0).

In den Labelbereich kann eingetragen werden:

- der Eigentümer der Diskette
- eine Versionsnummer.

Der Eintrag in den Labelbereich einer Diskette kann mit *fldisp* ausgegeben und mit *flchk* überprüft werden.

flinit befindet sich im Dateiverzeichnis */usr/sbin*.

Vor dem Aufruf beachten

Die Diskette muß in das Diskettenlaufwerk 0 eingelegt werden. Sie darf nicht schreibgeschützt sein. Es werden nur Disketten im Laufwerk 0 unterstützt.

Handelt es sich um eine neue Diskette, so muß sie vorher formatiert werden. Dazu benützen Sie das Kommando
`format /dev/rdsk/f0...t`
je nach gewünschter Diskettenformatierung. Sie können jedes verfügbare Diskettenformat verwenden (siehe *format*). Nur dann wird die Diskette so formatiert, daß Sie mit *flinit* auf den Labelbereich zugreifen können.

/usr/sbin/flinit[**-o** eigentümer][**-v** versionsnummer]

Kein Argument angegeben
: Für *versionsnummer* wird ein leere Zeichenkette eingetragen.
Für *eigentümer* werden 14 Leerzeichen eingetragen.

-o eigentümer
: Hier können Sie für *eigentümer* einen maximal 14 Zeichen langen Eigentümer-Namen angeben.

-o eigentümer nicht angegeben:
: Für *eigentümer* werden 14 Leerzeichen eingetragen.

-v␣versionsnummer
versionsnummer ist eine maximal sechsstellige alphanumerische Zeichenkette, die in den Labelbereich eingetragen wird. Wenn eine Versionsnummer nur aus Ziffern besteht, wird sie rechtsbündig eingetragen und links fehlende Stellen werden mit Nullen aufgefüllt. Enthält die angegebene Versionsnummer auch andere Zeichen, dann wird sie linksbündig eingetragen.

-v␣versionsnummer nicht angegeben:
Für *versionsnummer* wird eine leere Zeichenkette eingetragen.

Der Aufbau einer Diskette

Eine mit *format* formatierte und mit *flinit* beschriebene Diskette ist in zwei Bereiche eingeteilt:

- den Labelbereich, auf den Sie unter dem Gerätedatei-Namen */dev/dsk/f0...t* oder zeichenorientiert unter */dev/rdsk/f0...t* zugreifen können. Der Labelbereich kann Informationen über den Eigentümer der Diskette und über die Version des Diskettenin halts enthalten.

  Vorsicht

  Greifen Sie unter o.g. Gerätedatei-Namen auf die Diskette zu, so können Sie die gesamte Diskette lesen bzw. beschreiben. Der Zugriff bleibt vom Gerätetreiber her nicht auf den Labelbereich begrenzt und es besteht die Gefahr des unkontrollierten Überschreibens von Daten im Anschluß an den Labelbereich.

- den Datenbereich, auf den Sie unter dem Dateinamen */dev/dsk/f0...* (ohne abschließendes t) zugreifen können.
  Der Datenbereich ist der Bereich, in dem Sie Daten speichern.

Anwendung des flinit-Kommandos

flinit wird vom Menüsystem bei der Bearbeitung von Disketten benutzt. Mit dem Eintrag im Labelbereich kann ein Programm eine eingelegte Diskette identifizieren. Außerdem kann im Menüsystem nur der im Labelbereich eingetragene Eigentümer auf die Diskette zugreifen. So können Daten vor dem Zugriff anderer Benutzer geschützt werden, soweit dieser Zugriff über das Menüsystem erfolgt.

Vorsicht

Die Kommandos *cpio*, *tar* u.a. überprüfen den Labelbereich nicht. Sie können also mit *flinit* Ihre Disketten nicht prinzipiell vor unberechtigtem Zugriff schützen.

DATEIEN

/dev/dsk/f0...t
Gerätedatei für den Zugriff auf den Labelbereich einer Diskette.

/dev/dsk/f0...
Gerätedatei für den Zugriff auf den Datenbereich einer Diskette.

BEISPIEL

Ein Benutzer möchte in den Labelbereich den Benutzernamen *hans* und die Versionsnummer *1* eintragen.

```
$ flinit -v 1 -o hans
```

Der Benutzer kann mit dem Kommando *fldisp* überprüfen, ob die Einträge richtig vorgenommen wurden.

Der Eintrag im Labelbereich lautet:

```
VSN:    000001
owner: hans
```

SIEHE AUCH

flchk, *fldisp*, *format*

fmli
FMLI aktivieren (Form and menu language interpreter)

Die Formular- und Menüsprache FMLI stellt für Entwickler eine Umgebung zur Verfügung, um Anwendungen zu schreiben und Anwendungsschnittstellen zu definieren, die auf Formulare und Menüs aufgebaut sind. Mit dem Kommando *fmli* aktivieren Sie den Interpreter für die Formular- und Menüsprache FMLI und eröffnen den oder die Rahmen, die in der angegebenen Datei spezifiziert sind. Eine genaue Beschreibung von FMLI finden Sie im Handbuch *Programmmer's Guide: Character User Interface* [25].

fmli[_option]_datei..

Keine Option angegeben
fmli aktiviert den Formular- und Menü-Sprachen-Interpreter und eröffnet den oder die Rahmen, die in der angegebenen Datei spezifiziert sind.

option

-a_aliasdatei
(a - alias file) Die angegebene *aliasdatei* enthält Synonyme und Abkürzungen, die die Referenzierung von Objekten und Geräten mit überlangen Pfadnamen erleichtern. Das Format der *aliasdatei* unterliegt folgenden Konventionen: Jede Zeile enthält die Syntax *alias=pfadname*. Die Aliasnamen können mit *$alias* angesprochen werden. Dadurch kann in Definitionsdateien einfacher mit Objekten oder Gerätedateien mit langen Pfadnamen gearbeitet werden. Zudem kann ähnlich wie im Kommandointerpreter ein Suchpfad definiert werden.

-c_kommandodatei
(c - command file) In der angegebenen *kommandodatei* können voreingestellte FMLI Kommandos deaktiviert und neue anwendungsspezifische Kommandos definiert werden. Der Inhalt dieser Datei erscheint im FMLI Kommandomenü.

-i_initialisierungsdatei
(i - initialization file) In der *initialisierungsdatei* können Sie eine oder mehrere der folgenden Eigenschaften definieren, die das Gesamtbild der Anwendung bestimmen:

- Einen Einführungsrahmen mit Produktinformation, der nur vorübergehend sichtbar ist.
- Eine Kopfzeile, ihre Position und andere Elemente einer Kopfzeile
- Farbattribute für alle Bildschirmelemente.
- Darstellung der Funktionstasten am Bildschirm.

datei

datei muß mit einem der drei erlaubten Objektnamen *Menu.*, *Form.*, oder *Text.* beginnen und ein festgelegtes Datenformat haben (siehe *Programmer's Guide: Character User Interface* [25].

FEHLERMELDUNGEN

`Initial object must be specified`

Beim Aufruf von *fmli* wurde keine Initialisierungsdatei *datei* angegeben.

`Can't open object` *datei*

Das angegebene Argument *datei* existiert nicht oder ist nicht lesbar. *fmli* beendet sich nach Ausgabe dieser Fehlermeldung.

`I do not recognize that kind of object`

Der Name des angegebenen Arguments *datei* beginnt nicht mit einem der drei erlaubten Objektnamen (*Menu.*, *Form.* oder *Text.*), oder die Datei enthält kein korrektes Datenformat.

UMGEBUNGSVARIABLEN

LOADPFK

Bei Datensichtstationen mit programmierbaren Funktionstasten lädt *fmli* Zeichensequenzen für Funktionstasten in den Speicher der Datensichtstation für programmierbare Funktionstasten. Alle Einstellungen des Benutzers für die programmierbaren Funktionstasten werden dadurch überschrieben. Sie können dies verhindern, indem Sie die Variable *LOADPFK* auf den Wert *NO* setzen.

COLUMNS

Damit kann eine andere Bildschirmbreite angegeben werden, als für eine Datensichtstation des Typs *TERM* definiert ist.

Beispiel

Eine Datensichtstation kann mit 132 Spalten betrieben werden. Wird nun FMLI mit der Kommandozeile

```
COLUMNS=132 fmli datei
```

aufgerufen, so wird die Bildschirmbreite 132 Spalten benutzt.

LINES

Damit kann eine andere Zeilenzahl angegeben werden, als für eine Datensichtstation des Typs *TERM* definiert ist.

BEISPIELE

1. FMLI wird mit einer Datei aufgerufen, die den Namenskonventionen genügt:

```
fmli Menu.start
```

2. FMLI wird mit der Initialisierungsdatei *init.app* aufgerufen:

```
fmli -i init.app Menu.start
```

SIEHE AUCH

Programmer's Guide: Character User Interface [25]

fmt
Einfache Textformatierung (format)

Mit *fmt* können Sie Textdateien oder Text, den *fmt* von der Standard-Eingabe liest, einer einfachen Textformatierung unterziehen. Die Funktionalität von *fmt* umfaßt das Verbinden, Teilen und Auffüllen von Zeilen. Das Ergebnis schreibt *fmt* auf die Standard-Ausgabe.

fmt[_option][_datei]...

Keine Option angegeben
fmt verbindet und teilt die eingelesenen Zeilen so, daß sie bis zu 72 Zeichen lang sind, und gibt die neuen Zeilen auf die Standard-Ausgabe aus.

option

-c
(c - crown margin mode) Die Einrückung der ersten beiden Zeilen eines Textabschnitts bleibt erhalten. Alle nachfolgenden Zeilen innerhalb eines Textabschnitts werden entsprechend dem linken Rand der zweiten Zeile ausgerichtet. Diese Option ist bei mehreren, aufeinanderfolgenden Textabschnitten sinnvoll anwendbar.

-s
(s - split lines only) *fmt* füllt zu kurze Zeilen nicht auf. Mit dieser Option kann verhindert werden, daß Beispiele im Text, wie Quellcodeauszüge o.ä., ungewollt umformatiert werden.

-w_anzahl
(w - width) Die Länge der ausgegebenen Zeilen soll *anzahl* Zeichen sein. Ist diese Option nicht angegeben, produziert *fmt* Ausgabezeilen mit einer Länge von 72 Zeichen.

datei
Name der Textdatei, die neu formatiert werden soll. Sie können mehrere Dateien angeben. Die nach dem Kommando *fmt* angegebenen Optionen gelten dann für alle Dateien. Die Inhalte der Dateien werden verkettet.

datei nicht angegeben:
fmt liest von der Standard-Eingabe.

Arbeitsweise

Einrückungen werden von *fmt* in der Ausgabe beibehalten. Zeilen mit unterschiedlichen Einrückungen werden nicht miteinander verbunden (es sei denn, die Option *-c* ist eingeschaltet).

Leerzeilen und die Wortabstände innerhalb der Zeilen werden von *fmt* unverändert weitergegeben.

Wörter, deren Länge *anzahl* Zeichen überschreitet, werden nicht formatiert. Die maximale Zeilenlänge wird in diesem Fall überschritten.

Zeilen, die mit einem Punkt . beginnen, werden aus Kompatibilitätsgründen zu *nroff* von *fmt* nicht formatiert.

Ebenfalls ausgenommen von der Formatierung sind Zeilen, die mit "From:" anfangen, d.h. Zeilen des *mail*-Vorspanns (siehe *mail*).

BEISPIEL

Die Datei *text* wird zunächst mit dem Kommando *cat* im Originalzustand und dann mit dem Kommando *fmt* auf 40 Spalten Breite formatiert ausgegeben:

```
$ cat text
Es war eine finstere und stuermische Nacht.
Gewitterboeen fegten ueber die
aufgepeitschte See.
Am Horizont erschien ein Piratenschiff. "Hilfe!" schrie
das Dienstmaedchen.

$ fmt -w 40 text
Es war eine finstere und stuermische
Nacht. Gewitterboeen fegten ueber die
aufgepeitschte See.  Am Horizont
erschien ein Piratenschiff. "Hilfe!"
schrie das Dienstmaedchen.
```

SIEHE AUCH

vi
nroff [13]

fmtmsg
Ausgabe formatierter Meldungen (formatted message)

fmtmsg gibt formatierte Meldungen auf die Standard-Fehlerausgabe oder auf die Systemkonsole aus.

Eine formatierte Meldung besteht aus bis zu fünf Standard-Komponenten. Die Klassifikationen und Unterklassen werden nicht als Teil der Meldung ausgegeben, sondern beschreiben zum einen die Ursache und Herkunft der Meldung und steuern zum anderen die Ausgabe der formatierten Meldung.

fmtmsg[option]...meldungstext

Keine Option angegeben
fmtmsg gibt die möglichen Optionen aus.

option

-c klasse
(c - class) Die Herkunft der Meldung wird beschrieben. *klasse* kann sein:

hard	Die Meldung betrifft die Hardware.
soft	Die Meldung betrifft die Software.
firm	Die Meldung betrifft die Firmware.

-u unterklasse
unterklasse ist eine Liste von Schlüsselwörtern, die die Meldung detaillierter beschreiben und die Ausgabe der Meldung steuern. Die Schlüsselwörter müssen durch Kommas voneinander getrennt werden.
Zulässige Schlüsselwörter für *unterklasse* sind:

appl	Die Meldung kommt aus einer Anwendung. Nicht in Kombination mit *util* oder *opsys* verwenden.
util	Die Meldung kommt von einem Dienstprogramm. Nicht in Kombination mit *appl* oder *opsys* verwenden.
opsys	Die Meldung kommt aus dem Systemkern. Nicht in Kombination mit *appl* oder *util* verwenden.
recov	Die Anwendung wird sich von der Ursache der Meldung erholen. Nicht in Kombination mit *nrecov* verwenden.
nrecov	Die Anwendung wird sich nicht von der Ursache der Meldung erholen. Nicht in Kombination mit *recov* verwenden.
print	Die Meldung wird auf die Standard-Fehlerausgabe geschrieben.
console	Die Meldung wird auf die Systemkonsole ausgegeben.

print und *console* können auch zusammen angegeben werden.

-l␣marke
(l - label) Die Herkunft der Meldung wird gekennzeichnet.

-s␣erheblichkeitsgrad
(s - severity) Der Erheblichkeitsgrad des aufgetretenen Fehlers wird beschrieben. Die Schlüsselwörter und Definitionen der Erheblichkeitsgrade sind:

halt	Die Anwendung ist auf einen schwerwiegenden Fehler gestoßen und beendet sich.
error	Die Anwendung ist auf einen Fehler gestoßen.
warn	Die Anwendung ist auf eine ungewöhnliche Situation gestoßen, die Probleme bereiten könnte.
info	Die Anwendung gibt Informationen aus, die keinen Fehler betreffen.

-t␣marke
(t - tag) Eine Zeichenkette, die eine Kennzeichnung für die Meldung enthält.

-a␣aktion
(a - action) Eine Zeichenkette, die die erste Aktion in dem Fehlerbehebungsprozeß beschreibt. Die Zeichenkette muß so geschrieben sein, daß sie als ein einziges Argument interpretiert werden kann. *fmtmsg* stellt in der Ausgabe das Präfix `TO FIX:` voran.

meldungstext
Der Text der Meldung. Er muß so geschrieben sein, daß er als ein einziges Argument interpretiert werden kann.

UMGEBUNGSVARIABLEN

MSGVERB

MSGVERB wird vom Systemverwalter in */etc/profile* für das System gesetzt und kann vom Benutzer in der eigenen Profile-Datei *.profile* neu belegt werden.

Der Inhalt der Variablen *MSGVERB* gibt an, welche Komponenten der Meldung ausgegeben werden, wenn auf die Standard-Fehlerausgabe geschrieben wird. *MSGVERB* enthält eine Liste von optionalen Schlüsselwörtern, die durch Doppelpunkte voneinander getrennt sind. *MSGVERB* wird folgendermaßen gesetzt:

```
MSGVERB=[schlüsselwort[:schlüsselwort[:...]]]
export MSGVERB
```

Die zulässigen Schlüsselworte sind *label, severity, text, action, tag*. Sie können in beliebiger Reihenfolge angegeben werden. Enthält *MSGVERB* das Schlüsselwort für eine Komponente der Meldung und ist das Argument für die Komponente im *fmtmsg*-Aufruf keine leere Zeichenkette, dann wird diese Komponente in der Meldung auf die Standard-Fehlerausgabe ausgegeben. Ist das Schlüsselwort nicht in der Variablen *MSGVERB* enthalten, so erscheint die Komponente auch nicht in der Ausgabe der Meldung.

fmtmsg gibt alle Komponenten der Meldung aus, falls *MSGVERB* nicht definiert ist, falls der Wert der Variablen die leere Zeichenkette ist, falls die Schlüsselwörter im falschen Format angegeben sind oder falls unzulässige Schlüsselwörter enthalten sind.

MSGVERB wird nur für die Meldungen ausgewertet, die für die Standard-Fehlerausgabe bestimmt sind. Meldungen für die Systemkonsole enthalten immer alle Meldungskomponenten.

SEV_LEVEL

kann in Shell-Skripten verwendet werden.

In der Variablen *SEV_LEVEL* (severity level) können, zusätzlich zu den vordefinierten Erheblichkeitsstufen, neue Stufen definiert werden. Die vordefinierten Stufen können nicht geändert werden. Dies sind:

0 (keine Wertung)
1 HALT
2 ERROR
3 WARNING
4 INFO

SEV_LEVEL wird folgendermaßen gesetzt:

```
SEV_LEVEL=[beschreibung[:beschreibung[:...]]]
export SEV_LEVEL
```

beschreibung besteht aus drei Feldern, die durch Kommas voneinander getrennt sind:

beschreibung=schlüsselwort,stufe,text

schlüsselwort kann in der Option *-s* verwendet werden. *stufe* stellt einen positiven, ganzzahligen Wert dar, der an *fmtmsg* weitergegeben wird, wenn das Schlüsselwort *schlüsselwort* verwendet wird. Die Werte 0,1,2,3 oder 4 sind reservierte Werte und können hier nicht angegeben werden.
text ist eine Zeichenkette, die von *fmtmsg* ausgegeben wird, wenn *stufe* benutzt wird.

Ist *SEV_LEVEL* nicht definiert, oder enthält die Variable die leere Zeichenkette, so sind nur die vordefinierten Erheblichkeitsstufen verfügbar. Enthält die Belegung der Variablen Syntaxfehler, so wird der fehlerhafte Teil ignoriert.

ENDE-STATUS

0 Erfolgreiche Ausführung aller Funktionen.
1 Der Kommandoaufruf enthält einen Syntaxfehler, eine unzulässige Option oder ein ungültiges Argument zu einer Option.
2 Die Funktion wurde nur mit teilweisem Erfolg durchgeführt, die Meldung wurde aber nicht auf die Standard-Fehlerausgabe ausgegeben.
4 Die Funktion wurde nur mit teilweisem Erfolg durchgeführt, die Meldung wurde aber nicht auf die Systemkonsole ausgegeben.
32 Keine der angeforderten Funktionen wurde erfolgreich ausgeführt.

BEISPIEL

Die folgende Shellprozedur soll das Kommando *ls -l* unter dem kürzeren Namen *ll* erreichbar machen. Die Standardfehlermeldungen des Kommandos *ls* werden durch den Zusatz *2>/dev/null* abgefangen. Stattdessen wird ein eventuell aufgetretener Fehler am Rückgabewert $? des Kommandos erkannt und es wird, falls nötig, eine eigene Fehlermeldung mit dem Kommando *fmtmsg* erzeugt:

```
$ cat ll

ls -l $* 2>/dev/null
case $? in
     0) echo "Das war das DVZ `pwd`/$*";;
     *) fmtmsg -u print -l "ll" -s error -t "(evt. falsches DVZ)" -a \
       "Versuchen Sie es mit einem anderen DVZ." "Ein Fehler ist aufgetreten!"
esac

$ ll krimskrams
ll: ERROR: Ein Fehler ist aufgetreten!
   TO FIX: Versuchen Sie es mit einem anderen DVZ.  (evt. falsches DVZ)

$ ll kram

total 4
-rw-------   1 kmdo     kmdoag       690 Mar 18 14:00 dat.Z
-rw-r--r--   1 kmdo     kmdoag        63 Mar 13 09:29 startwerte
Das war das DVZ /home/kmdo/hans/kram
```

SIEHE AUCH

addseverity(), fmtmsg() [19]

fold
Lange Zeilen zerlegen

fold zerlegt die Zeilen der Textdateien oder der Standard-Eingabe so, daß sie nicht länger sind als die voreingestellte oder angegebene Zeilenlänge. Standardmäßig sind 80 Zeichen pro Zeile vorgegeben.

Vorsicht

Enthalten die Eingabezeilen Unterstreichungen, so kann dies zu fehlerhaftem Verhalten von *fold* führen.

fold[_option]..[_datei]...

Keine Option angegeben
Die eingelesenen Zeilen zerlegt *fold* in Zeilen mit maximal 80 Zeichen (Voreinstellung).

option

-w_anzahl
(w - width) Die maximale Zeilenlänge soll *anzahl* Zeichen betragen.

Falls Tabulatoren in den Eingabezeilen vorkommen, sollte *anzahl* ein Vielfaches von 8 sein.

datei
Name der Textdatei, die von *fold* bearbeitet werden soll. Sie können mehrere Dateien angeben. Die angegebenen Optionen gelten dann für alle Dateien.

datei nicht angegeben:
fold liest von der Standard-Eingabe.

FEHLERMELDUNG

```
Bad number for fold
```

Sie erhalten diese Fehlermeldung, wenn sie für *anzahl* einen ungültigen Wert angeben.

BEISPIEL

Sie geben den Inhalt der Datei *notiz* auf die Standard-Ausgabe aus.

Unter Verwendung des Kommandos *cat* erhalten Sie folgendes Ergebnis:

```
$ cat

Notizen:
Zum Projekt Z noch Angebot der Firma Leist & Gutmann einholen. Vergleich mit
den anderen Angeboten muss im Mai abgeschlossen sein.
```

Verwenden Sie jedoch das Kommand *fold*, haben Sie die Möglichkeit, die Länge der ausgegebenen Zeilen neu festzusetzen.

In folgendem Beispiel werden die Zeilen so zerlegt, daß die Ausgabe eine maximale Breite von 40 Zeichen hat:

```
$ fold -w 40 notiz

Notizen:
Zum Projekt Z noch Angebot der Firma Lei
st & Gutmann einholen. Vergleich mit
den anderen Angeboten muss im Mai abgesc
hlossen sein.
```

SIEHE AUCH

pr

format
Disketten formatieren

format formatiert Disketten vor der Benutzung auf einem SINIX-System. Sie verwenden *format* von der Kommandozeile aus.

Sie können mit *format* Disketten nur für UNIX- bzw. SINIX-Systeme formatieren. Wenn Sie Disketten für das Betriebssystem MS-DOS formatieren wollen, müssen Sie das Kommando *dosformat* verwenden.

Sie können unter SINIX nur fehlerfreie Disketten verwenden.

Disketten mit geringer Schreibdichte, also 48 tpi (tracks per inch, Spuren pro Zoll) sollten nicht auf einem Laufwerk mit hoher Schreibdichte (96 tpi) formatiert werden. Disketten, die auf einem Laufwerk mit hoher Schreibdichte beschrieben wurden, müssen auf einem Laufwerk mit hoher Schreibdichte gelesen werden. Eine Diskette mit geringer Schreibdichte, die auf einem Laufwerk mit hoher Schreibdichte beschrieben wurde, kann auf einem Laufwerk mit geringer Schreibdichte unter Umständen nicht mehr gelesen werden.

Sie können 3,5-Zoll-Disketten mit hoher Schreibdichte auch für doppelte Schreibdichte formatieren. Wenn Sie beim späteren Beschreiben und Lesen so formatierter Disketten Fehler vermeiden wollen, schließen Sie den (nur bei HD-Disketten vorhandenen) zweiten Schieber in der Diskettenumhüllung. Sollte die Diskette an dieser Stelle nicht über einen Schieber verfügen, schließen Sie zu diesem Zweck die Öffnung mit einem Stück Klebeband.

format[␣option]␣gerätedatei

-f␣zahl
(f - first) Bestimmt die erste Spur der Diskette, die formatiert werden soll.

-l␣zahl
(l - last) Bestimmt die letzte Spur der Diskette, die formatiert werden soll.

-i
(i - interleave) Interleave-Faktor, mit dem die Diskette formatiert wird.
Verwenden Sie einen Rechner vom Typ MX 300 der Intel-Linie, können Sie den Interleave-Faktor nicht verändern. Der Standardwert ist mit 1 vorgegeben.

-V
(V - verify) Mit dieser Option legen Sie fest, daß einige zufällig ausgewählte Sektoren der Diskette zur Probe beschrieben und gelesen werden.

-E

(E - exhaustive verify) Mit dieser Option legen Sie fest, daß alle Sektoren der Diskette zur Probe beschrieben und gelesen werden.

gerätedatei

gibt an, welches Medium formatiert werden soll.

Die folgende Liste gibt Ihnen Aufschluß über die wichtigsten Gerätedateien und welche Formate diese unterstützen:

Gerätedatei	unterstütztes Diskettenformat
/dev/rdsk/f15ht	5,25 Zoll, 160 Spuren, 15 Sektoren/Spur, 1200 kb
/dev/rdsk/f15d9t	5,25 Zoll, 80 Spuren, 9 Sektoren/Spur, 360 kb
/dev/rdsk/f15qt	5,25 Zoll, 160 Spuren, 9 Sektoren/Spur, 720 kb
/dev/rdsk/f03ht	3,5 Zoll, 160 Spuren, 18 Sektoren/Spur, 1440 kb

DATEIEN

*/dev/rdsk/f**
Gerätedateien
/dev/rfd[0-n]
Gerätedateien

BEISPIELE

Um mittels *format* eine 3,5-Zoll-Diskette mit hoher Schreibdichte für die Benutzung vorzubereiten, legen Sie die Diskette in das entsprechende Laufwerk des Rechners und geben ein:

```
$ format /dev/rdsk/f03ht
```

Während der Formatierung zeigt *format* an, wieviele Spuren der Diskette bereits bearbeitet wurden. Für jeweils zehn formatierte Spuren erscheint auf der Ausgabezeile ein Punkt.

Möchten Sie diesen Formatiervorgang im Hintergrund ablaufen lassen und auch durch die Ausgabe der Punkte für bereits formatierte Sektoren nicht gestört werden, geben Sie ein:

```
$ format /dev/rdsk/f03ht 2> /dev/null &
```

SIEHE AUCH

fd [7]

ftp
Programm zur Dateiübertragung (file transfer program)

Die Beschreibung ist in folgende Abschnitte unterteilt:

- Einführung
- Syntax und Beschreibung der Optionen
- *ftp*-Kommandos
  - Funktionale Übersicht
  - Alphabetische Beschreibung
- Konventionen für lokale Dateinamen
- Abbrechen einer Datenübertragung
- Parameter zur Dateiübertragung
- Die Datei *.netrc*
- Dateien
- Hinweis zu den Fehlermeldungen
- Beispiele

Einführung

Das Kommando *ftp* ist die Benutzer-Schnittstelle zum ARPANET Standard-File-Transfer-Protokoll FTP. *ftp* überträgt Dateien von und zu Rechnern innerhalb eines Netzwerkes. Dabei ist es nicht nötig, daß auf dem fernen Rechner auch ein SINIX- oder UNIX-Betriebssystem läuft.

Eine Verbindung zum fernen Rechner können Sie auf zwei Arten herstellen:

- durch Angabe des fernen Rechners beim *ftp*-Aufruf
- durch Angabe des fernen Rechners beim *ftp*-Kommando *open*

Der Rechner, an dem Sie *ftp* aufrufen, bleibt immer der lokale Rechner.

Wenn auf dem fernen Rechner SINIX oder UNIX (V5.4) läuft, so wird jede *ftp*-Sitzung am fernen Rechner in der Datei */var/adm/wtmp* protokolliert, falls diese angelegt ist.

Syntax und Beschreibung der Optionen

ftp[_-dgintv][ferner_rechner]

Keine Option und kein Argument angegeben
ftp startet nur den lokalen *ftp*-Kommando-Interpreter. In diesem Zustand können Sie alle *ftp*-Kommandos benutzen, die weder direkt die Dateiübertragung betreffen noch Arbeiten an einem fernen Rechner ausführen, also *!, ?, bell, case, cr, debug, glob, hash, help, lcd, macdef, nmap, ntrans, runique, sendport, status, sunique* und *verbose*.

Keine Option angegeben
ftp baut eine Verbindung zum FTP-Dämon des angegebenen Rechners auf.

-d
(d - debug) schaltet den Testhilfe-Modus ein. *ftp* gibt jedes Kommando am Bildschirm aus, das zum fernen Rechner gesendet wird (siehe auch *ftp*-Kommando *debug*).

Hinweis
Bitte beachten Sie im Sinne der Systemsicherheit folgenden Hinweis zur Verwendung dieser Option:
Wenn Sie sich mit dem Kommando *user* am fernen Rechner anmelden, fragt der Rechner anschließend Ihr Kennwort ab. Haben Sie *ftp* mit *-d* aufgerufen, so wird nach Ihrer Eingabe das Kennwort am Bildschirm angezeigt.

-g
(g - glob) unterbindet Dateinamen-Expansion (siehe auch *ftp*-Kommando *glob*).

-i
(i - interactive) schaltet die Funktion *prompt* aus. D.h., *ftp* verhindert beim Übertragen oder Löschen mehrerer Dateien mit *mget, mput* oder *mdelete*, daß Sie bei jeder Datei aufgefordert werden, die Ausführung des Kommandos zu bestätigen. Sie können die Funktion *prompt* während der Sitzung mit dem *ftp*-Kommando *prompt* wieder einschalten.

-n

(n - no auto-login) verhindert die automatische Abfrage von Benutzerkennung und Kennwort beim Verbindungsaufbau. Benutzen Sie für eine anschließende Anmeldung am fernen Rechner das *ftp*-Kommando *user*.

-n nicht angegeben:
ftp durchsucht die Datei *.netrc* im Home-Dateiverzeichnis am lokalen Rechner. In dieser Datei kann eine Benutzerkennung für den fernen Rechner eingetragen sein. Wenn kein derartiger Eintrag vorhanden ist oder die Datei *.netrc* nicht existiert, verlangt *ftp*, daß Sie sich mit Benutzerkennung und Kennwort anmelden. Haben Sie auf dem fernen Rechner die gleiche Benutzerkennung wie auf dem lokalen Rechner, dann drükken Sie bei der entsprechenden Aufforderung nur ↵ und geben Sie anschließend Ihr Kennwort ein.

Zu Inhalt und Format der Datei *.netrc* siehe Abschnitt *Die Datei .netrc*.

-t

(t - trace) ermöglicht Datenpaket-Überwachung (derzeit nicht implementiert).

-v

(v - verbose) zeigt alle Antworten des fernen Rechners an und protokolliert den Verlauf der Dateiübertragung am Bildschirm. Diese Anzeigefunktion ist standardmäßig eingeschaltet, wenn *ftp* interaktiv läuft (siehe auch *ftp*-Kommando *verbose*).

ferner_rechner
ist der Name eines fernen Rechners.

ferner_rechner nicht angegeben:
ftp startet nur seinen Kommando-Interpreter. In diesem Zustand können Sie alle *ftp*-Kommandos benutzen, die weder die Dateiübertragung betreffen, noch Arbeiten am fernen Rechner ausführen, also z.B. *!, bell, glob, hash, lcd, macdef, status*.

Mit dem *ftp*-Kommando *open* können Sie jetzt eine Verbindung zu einem Rechner aufbauen.

ftp-Kommandos

Im folgenden Abschnitt erhalten Sie zunächst eine Übersicht über alle *ftp*-Kommandos, sortiert nach ihren Funktionen. Dabei kann es vorkommen, daß einige Kommandos mehrmals aufgeführt werden.
Im Anschluß an die Übersicht werden alle *ftp*-Kommandos in alphabetischer Reihenfolge beschrieben.

Einige *ftp*-Kommandos haben Ein-/Ausschaltfunktion.
Mit *status* können Sie sich die aktuellen Einstellungen an Ihrem Rechner anzeigen lassen.

Alle Ein-/Ausschalt-Kommandos haben folgende voreingestellte Einstellungen (Standards). Dabei bedeutet *on*, daß die Funktion eingeschaltet, und *off*, daß sie ausgeschaltet ist.

Kommando	on	off
bell		x
case		x
cr	x	
debug		x
glob	x	
hash		x
nmap		x
ntrans		x
prompt	x	
runique		x
sendport	x	
sunique		x
verbose	x	

Funktionale Übersicht

In der folgenden Übersicht bedeuten die Angaben in runden Klammern, daß

- die Funktion des Kommandos standardmäßig eingeschaltet (on) oder ausgeschaltet (off) ist
- der angegebene Wert standardmäßig gesetzt ist

Alle *ftp*-Kommandos sind bis zur Eindeutigkeit abkürzbar.

Verbindung zu fernem Rechner aufbauen

open	Verbindung zu einem fernen Rechner aufbauen
proxy	Gleichzeitige Verbindung zu zwei fernen FTP-Dämonen steuern

ftp-Sitzung beenden, vorübergehend verlassen, Verbindung abbauen

bye	*ftp*-Sitzung beenden
quit	*ftp*-Sitzung beenden
!	Am lokalen Rechner Subshell aufrufen oder Shell-Kommando ausführen
close	Verbindung zum fernen FTP-Dämon abbauen
disconnect	Verbindung zum fernen FTP-Dämon abbauen

Am fernen Rechner anmelden

user	Als Benutzer am fernen Rechner anmelden
account	Als Ressource-Benutzer anmelden

Hilfsinformationen ausgeben

verbose	Antworten des FTP-Dämons anzeigen, Übertragung protokollieren (on)
help	Hilfestellung zu den *ftp*-Kommandos ausgeben
?	Hilfestellung zu den *ftp*-Kommandos ausgeben
remotehelp	Hilfestellung zu den *ftp*-Kommandos des fernen Rechners ausgeben
status	Aktuelle Einstellungen der *ftp*-Kommandos ausgeben
debug	Testhilfe-Modus ein- oder ausschalten (off)
hash	Ausgabe des Nummernzeichnes bei Datenblock-Übertragung ein- oder ausschalten (off)
trace	Datenpaket-Überwachung ein- oder ausschalten

Dateien zum fernen Rechner übertragen

put	Einzelne Datei zum fernen Rechner übertragen
send	Einzelne Datei zum fernen Rechner übertragen
mput	Mehrere Dateien zum fernen Rechner übertragen
append	Eine Datei an eine andere am fernen Rechner anhängen

Dateien zum lokalen Rechner übertragen

get	Einzelne Datei zum lokalen Rechner übertragen
recv	Einzelne Datei zum lokalen Rechner übertragen
mget	Mehrere Dateien zum lokalen Rechner übertragen

Dateiverzeichnis einrichten, ausgeben, wechseln, löschen

mkdir	Fernes Dateiverzeichnis einrichten
pwd	Aktuelles Dateiverzeichnis am fernen Rechner ausgeben
ls	Inhalt eines fernen Dateiverzeichnisses ausgeben
dir	Inhalt eines fernen Dateiverzeichnisses ausführlich ausgeben
mls	Liste ferner Dateien in lokale Datei schreiben
mdir	Ausführliche Liste ferner Dateien in lokale Datei schreiben
cd	Fernes Dateiverzeichnis wechseln
cdup	In das dem aktuellen übergeordnete, ferne Dateiverzeichnis wechseln
lcd	Lokales Dateiverzeichnis wechseln
rmdir	Fernes Dateiverzeichnis löschen

Dateien anhängen, löschen

append	Lokale Datei an ferne Datei anhängen
delete	Ferne Datei löschen
mdelete	Mehrere ferne Dateien löschen

Dateinamen bearbeiten

rename	Ferne Datei umbenennen
glob	Dateinamen-Expansion ein- oder ausschalten (on)
case	Umwandlung von Groß- in Kleinschreibung der Dateinamen beim Übertragen mit *mget* ein- oder ausschalten (off).
runique	Abspeichern von Dateien auf dem lokalen Rechner unter einheitlichem Namen ein- oder ausschalten (off)
sunique	Abspeichern von Dateien auf dem fernen Rechner unter einheitlichem Namen ein- oder ausschalten (off)
nmap	Dateinamen-Expansion beim Übertragen von und zu Nicht-UNIX-Rechnern ein- oder ausschalten (off)
ntrans	Übersetzung der Einzelzeichen von Dateinamen beim Übertragen von und zu Nicht-UNIX-Rechnern ein- oder ausschalten (off)

Dateiübertragungs-Parameter setzen

form	Übertragungsformat setzen (*non-print*)
mode	Übertragungsmodus setzen (*stream*)
struct	Übertragungsstruktur setzen (*file*)
type	Übertragungstyp setzen (*ascii*)
ascii	Übertragungstyp auf ASCII-Code *ascii* setzen
binary	Übertragungstyp auf Binärcode *image* setzen
tenex	Übertragungstyp zum Übertragen mit TENEX-Rechnern setzen

Besondere Kommandos zum Übertragen von und zu Nicht-UNIX-Rechnern

nmap	Dateinamen-Expansion ein- oder ausschalten (off)
ntrans	Übersetzung der Einzelzeichen von Dateinamen ein- oder ausschalten (off)
cr	Ausgabe des Wagenrücklauf-Zeichens CR beim Übertragentyp *ascii* ein- oder ausschalten (on)
sendport	Verwendung von PORT-Kommandos ein- oder ausschalten (on)
tenex	Übertragungstyp zum Übertragen mit TENEX-Rechnern setzen

Sonstiges

!	Am lokalen Rechner Shell-Kommando ausführen oder Subshell aufrufen
$	Ein zuvor mit *macdef* definiertes Makro ausführen
bell	Akustisches Kommando-Ende-Zeichen ein- oder ausschalten (off)
case	Umwandlung von Groß- in Kleinschreibung der Dateinamen beim Übertragen mit *mget* ein- oder ausschalten (off).
cr	Ausgabe des Wagenrücklauf-Zeichens CR beim Übertragentyp *ascii* ein- oder ausschalten (on)
glob	Dateinamen-Expansion ein- oder ausschalten (on)
hash	Ausgabe des Nummernzeichnes bei Datenblock-Übertragung ein- oder ausschalten (off).
macdef	Makro definieren
nmap	Dateinamen-Expansion beim Übertragen mit Nicht-UNIX-Rechnern, die andere Namenskonventionen haben, ein- oder ausschalten (off)
ntrans	Übersetzung der Einzelzeichen von Dateinamen beim Übertragen mit Nicht-UNIX-Rechnern, die andere Namenskonventionen haben, ein- oder ausschalten (off)
prompt	Dialog-Aufforderung beim Übertragen mehrerer Dateien ein- oder ausschalten (on)
proxy	Gleichzeitige Verbindung zu zwei fernen FTP-Dämonen steuern
quote	FTP-Dämon testen
reset	Kommando-Antwort-Sequenz mit dem fernen FTP-Dämon resynchronisieren
sendport	Verwendung von PORT-Kommandos ein- oder ausschalten (on)
status	Aktuelle Einstellungen der *ftp*-Kommandos ausgeben
trace	Datenpaket-Überwachung einstellen
verbose	Antworten des FTP-Dämons anzeigen, Übertragung protokollieren (on)

Alphabetische Beschreibung

Alle *ftp*-Kommandos sind bis zur Eindeutigkeit abkürzbar.

![sh_kommando]
Führt das Shell-Kommando *sh_kommando* auf dem lokalen Rechner aus.

sh_kommando nicht angegeben:
Ihre *ftp*-Sitzung wird unterbrochen und eine Subshell aufgerufen.
Die Subshell beenden Sie mit END.

$makro-name[_argument]...
Führt das Makro *makro-name* aus, das zuvor mit dem *ftp*-Kommando *macdef* definiert wurde. Angegebene Argumente werden nicht expandiert.

?[_ftp_kommando]...
Gibt die Bedeutung des angegebenen *ftp*-Kommandos aus. Sie können mehrere Kommandos angeben.
Die gleiche Funktion hat *help*.

ftp_kommando nicht angegeben:
Sie erhalten eine Liste aller am lokalen Rechner bekannten *ftp*-Kommandos.

account[_kennwort]
Ermöglicht nach erfolgreichem Anmelden am fernen System den Zugriff auf Ressourcen durch die Angabe eines zusätzlichen Kennworts.

kennwort nicht angegeben:
Sie werden vom fernen FTP-Dämon aufgefordert, das Kennwort einzugeben.

append_lokale_datei[_ferne_datei]
Fügt eine lokale Datei an eine Datei am fernen Rechner an.
Die aktuellen Werte für Typ, Format, Modus und Struktur bleiben unverändert.

ferne_datei nicht angegeben:
Der Name der lokalen Datei wird übernommen.
Wenn Sie die *ftp*-Funktionen *nmap* und *ntrans* eingeschaltet und zugehörige Argumente angegeben haben, wird der lokale Name diesen Argumenten entsprechend am fernen Rechner bearbeitet.

ascii
Setzt den Dateiübertragungs-Typ auf ASCII.
Da *ascii* der Standardwert ist, benötigen Sie dieses Kommando nur, wenn der Dateiübertragungs-Typ vorher geändert wurde (siehe *tenex*, *type* und *binary*).

bell
Schaltet ein akustisches Kommando-Ende-Zeichen ein oder aus. Standard: off.
Wenn Sie *bell* eingeschaltet haben, ertönt das Zeichen nach der Beendigung folgender *ftp*-Kommandos: *append*, *dir*, *get*, *ls*, *mdelete*, *mdir*, *mget*, *mls*, *mput*, *put*, *quote*, *recv*, *send*.

binary
Setzt den Dateiübertragungs-Typ auf *image* (Binär-Code-Übertragung). Dies ist einerseits bei Übertragung von Programmdateien notwendig, andererseits bei Rechnern, die nicht mit ASCII-Code arbeiten (siehe auch *ascii*, *tenex* und *type*).

bye
Beendet die *ftp*-Sitzung mit dem fernen FTP-Dämon und verläßt *ftp*. Die gleiche Funktion haben *quit* oder END.

case
Schaltet die Umwandlung von Groß- in Kleinschreibung der Dateinamen bei einer Dateiübertragung mit *mget* ein oder aus. Standard: off.
Wenn *case* eingeschaltet ist, werden die Dateinamen des fernen Rechners, die ausschließlich mit Großbuchstaben geschrieben wurden, am lokalen Rechner in Kleinbuchstaben geschrieben.

cd[_fernes_dvz]
Wechselt das Dateiverzeichnis zu *fernes_dvz* am fernen Rechner.

fernes_dvz nicht angegeben:
ftp fordert Sie auf, ein Dateiverzeichnis anzugeben.

cdup
Wechselt das Dateiverzeichnis am fernen Rechner in das dem aktuellen übergeordnete Dateiverzeichnis.

close
Verläßt den FTP-Dämon am fernen Rechner und kehrt zum *ftp*-Kommando-Interpreter am lokalen Rechner zurück. Die während der Sitzung definierten Makros werden gelöscht.
Die gleiche Funktion hat *disconnect*.

cr

Schaltet die Ausgabe des Wagenrücklauf-Zeichens CR (carriage return) ein oder aus. Standard: on.
Die Übertragungseinheiten bei ASCII-Dateien werden mit CR/LF (Zeilenanfang-Zeichen/Neue-Zeile-Zeichen) gesendet. Bei der Ausgabe versteht jeder UNIX-Rechner jedoch LF alleine bereits so, den Text in eine neue Zeile *und* an den Zeilenanfang zu schreiben. Daher werden alle CR intern gestrichen; LF steht dann für LF/CR.
Bei Dateiübertragungen mit Nicht-UNIX-Rechnern kommt es vor, daß Neue-Zeile-Zeichen/Zeilenanfang nur mit LF (Linefeed) ausgedrückt wird. Wenn Sie nun ASCII-Dateien versenden, kann dieses LF von CR/LF nur unterschieden werden, wenn *cr* ausgeschaltet ist.

debug

Schaltet den Testhilfe-Modus ein- oder aus. Standard: off.
Wenn *debug* eingeschaltet ist, gibt *ftp* jedes Kommando am Bildschirm aus, das zum fernen Rechner gesendet wird (siehe auch Option *-d*).

Hinweis
Bitte beachten Sie im Sinne der Systemsicherheit folgenden Hinweis zur Verwendung dieses Kommandos:
Wenn Sie das Kommando *user* benutzen, fragt der Rechner anschließend Ihr Kennwort ab. Ist *debug* eingeschaltet, so wird nach Ihrer Eingabe das Kennwort am Bildschirm angezeigt.

delete[_ferne_datei]

Löscht die Datei *ferne_datei* am fernen Rechner.
Wenn Sie mehrere Dateien löschen wollen, dann benutzen Sie *mdelete*.

ferne_datei nicht angegeben:
ftp fordert Sie auf, einen Dateinamen einzugeben.

dir[_fernes_dvz][_lokale_datei]

Schreibt den Inhalt des fernen Dateiverzeichnisses in die angegebene lokale Datei.

fernes_dvz nicht angegeben:
Für *fernes_dvz* wird das aktuelle Dateiverzeichnis am fernen Rechner gesetzt.

lokale_datei nicht angegeben:
Der Inhalt des Dateiverzeichnisses wird am Bildschirm ausgegeben.

disconnect

Verläßt den FTP-Dämon am fernen Rechner und kehrt zum *ftp*-Kommando-Interpreter am lokalen Rechner zurück (siehe *close*).

form[␣**non-print**]
Setzt das Dateiübertragungsformat auf *non-print*. Dieses Format ist standardmäßig gesetzt. Andere Formate werden zur Zeit nicht unterstützt. Daher erhalten Sie bei der Eingabe des Kommandos immer die Meldung:

```
We only support non-print format, sorry
```

get[␣ferne_datei][␣lokale_datei]
Holt die angegebene Datei vom fernen Rechner und speichert sie auf dem lokalen Rechner unter dem Namen *lokale_datei* ab.
Während der Übertragung gelten die aktuellen Werte für Typ, Format, Modus und Struktur.

ferne_datei nicht angegeben:
ftp fordert Sie auf, den Namen einer fernen Datei anzugeben.

lokale_datei nicht angegeben:
Die übertragene Datei erhält den gleichen Namen wie auf dem fernen Rechner.
Wenn Sie *case*, *nmap* und *ntrans* eingeschaltet und zugehörige Argumente angegeben haben, wird der Name der fernen Datei entsprechend für den lokalen Rechner bearbeitet.

glob
Schaltet die Dateinamen-Expansion für *mget*, *mput* und *mdelete* ein oder aus. Standard: on.
Für *mput* erfolgt die Expansion wie in der Shell (Bearbeitung der Metazeichen *, ? und []).
Für *mget* und *mdelete* wird jeder ferne Dateiname separat am fernen Rechner expandiert.

Ist *glob* ausgeschaltet, so werden alle Dateinamen wie angegeben behandelt.
Die Dateinamen-Expansion kann auch mit der Option *-g* ausgeschaltet werden.

hash
Schaltet die Ausgabe des Nummernzeichens # für jede Datenblock-Übertragung ein oder aus. Standard: off.
Die Größe eines Datenblocks ist 8192 byte.

help[␣ftp_kommando]...
Gibt die Bedeutung des angegebenen *ftp*-Kommandos aus. Sie können mehrere Kommandos angeben (siehe *?*).

lcd[␣dvz]
Wechselt das aktuelle Dateiverzeichnis auf dem lokalen Rechner.

dvz nicht angegeben:
Sie wechseln in Ihr HOME-Dateiverzeichnis auf dem lokalen Rechner.

ls[_fernes_dvz][_lokale_datei]
Schreibt alle im fernen Dateiverzeichnis enthaltenen Dateinamen in die angegebene lokale Datei.

fernes_dvz nicht angegeben:
Für *fernes_dvz* wird das aktuelle Dateiverzeichnis am fernen Rechner gesetzt.

lokale_datei nicht angegeben:
Die Dateinamen werden am Bildschirm ausgegeben.

macdef[_makro-name]
Öffnet ein Makro mit dem Namen *makro-name*. *makro-name* ist eine beliebige Zeichenkette. Nach dem Öffnen definieren Sie das Makro durch Eingabe von *ftp*-Kommandos. Anschließend können Sie das Makro mit *$makro-name* ausführen.

Sie können bis zu 16 Makros pro *ftp*-Sitzung definieren, die zusammen aus höchstens 4096 Zeichen bestehen dürfen. Beim Verlassen eines fernen *ftp*-Dämons mit *close* oder *disconnect* werden alle während der Verbindung definierten Makros gelöscht. Ein Makro, das während der gesamten Sitzung gültig sein soll, muß vor dem Verbindungsaufbau definiert worden sein.

makro-name nicht angegeben:
ftp fordert Sie auf, einen Namen einzugeben.

Ein Makro besteht aus:

- Kommandozeilen
- einer Nullzeile

Kommandozeilen

Kommandozeilen sind durch Neue-Zeile-Zeichen getrennte Zeilen mit jeweils einem auszuführenden *ftp*-Kommando.
Bei der Bearbeitung jeder Kommandozeile sucht der Makroprozessor nach Argumenten, die eines der beiden Sonderzeichen $ und \ enthalten. Diese Argumente vergleicht der Prozessor mit den entsprechenden Argumenten der Makro-Aufrufzeile und ersetzt jene gegebenenfalls wie folgt:

$zahl	wird durch das *zahl*-te Argument der Makro-Aufrufzeile ersetzt.
$i	wird beim *i*-ten Schleifendurchlauf durch das *i*-te Aufruf-Argument ersetzt.
\zeichen	wird durch das Zeichen *zeichen* ersetzt. So kann z.B. das Sonderzeichen $ mit \ entwertet werden.

Nullzeile

Die Nullzeile besteht aus zwei aufeinanderfolgenden Neue-Zeile-Zeichen, z.B. aus ↵ ↵. Sie schließt die Makro-Definition ab.

mdelete[␣ferne_datei]...
Löscht die angegebenen Dateien am fernen Rechner.
Wenn *prompt* eingeschaltet ist, fordert *ftp* Sie bei jeder zu löschenden Datei zur Bestätigung auf. Als Bestätigung können Sie eingeben:

↵	Datei soll gelöscht werden.
j	(ja) Datei soll gelöscht werden.
y	(yes) Datei soll gelöscht werden.
n	(nein) Datei soll nicht gelöscht werden. Nächste Datei anzeigen.

ferne_datei nicht angegeben:
ftp fordert Sie auf, die Namen der zu löschenden Dateien einzugeben.

mdir[␣ferne_datei]...[␣lokale_datei]
Schreibt eine ausführliche Liste der angegebenen fernen Dateien in die lokale Datei *lokale_datei*. Geben Sie für die lokale Datei einen Bindestrich an, dann werden die fernen Dateien auf Ihren Bildschirm gelistet. Geben Sie keines der beiden Argumente an, so fordert *ftp* Sie anschließend dazu auf.

Wenn *glob* eingeschaltet ist, werden die Dateinamen für *ferne_datei* expandiert.

Wenn *prompt* eingeschaltet ist, fragt *ftp* nochmals nach, ob der zuletzt angegebene Dateiname tatsächlich die Zieldatei ist:

```
output to local-file: lokale_datei?
```

Als Bestätigung können Sie eingeben:

↵	Zieldatei soll beschrieben werden.
j	(ja) Zieldatei soll beschrieben werden.
y	(yes) Zieldatei soll beschrieben werden.
n	(nein) Zieldatei soll nicht beschrieben werden. *ftp* tut nichts.

ferne_datei nicht angegeben:
ftp nimmt an, daß *lokale_datei* auf dem fernen Rechner vorhanden ist und gelistet werden soll und fordert Sie anschließend auf, eine Zieldatei am lokalen Rechner anzugeben. Sie erhalten keine Fehlermeldung.

lokale_datei nicht angegeben:
Wenn Sie mehrere Dateinamen für *ferne_datei* angegeben haben, fragt *ftp* nach, ob der zuletzt eingegebene der Name der Zieldatei ist.
Wenn dagegen *glob* eingeschaltet ist und Sie nur einen einzelnen, zu expandierenden Dateinamen für *ferne_datei* eingeben, z.B. *datei.**, dann fordert *ftp* Sie auf, zusätzlich den Namen einer Zieldatei einzugeben.

mget[_ferne_datei]...
Kopiert die angegebenen Dateien aus dem fernen aktuellen Dateiverzeichnis in das lokale aktuelle Dateiverzeichnis. Ist *ferne_datei* der Name eines Dateiverzeichnisses, dann überträgt *ftp* es nicht und gibt auch keine Fehlermeldung aus.

Wenn *glob* eingeschaltet ist, werden die angegebenen Dateinamen am fernen Rechner expandiert.

Wenn Sie *case*, *nmap* und *ntrans* eingeschaltet und zugehörige Argumente angegeben haben, wird jeder Name einer fernen Datei entsprechend für den lokalen Rechner bearbeitet.

Wenn *prompt* eingeschaltet ist, fordert *ftp* Sie bei jeder zu übertragenden Datei zur Bestätigung auf (siehe *mdelete*).

ferne_datei nicht angegeben:
ftp fordert Sie auf, die Namen der zu übertragenden Dateien einzugeben.

mkdir[_dvz]
Erstellt das Dateiverzeichnis *dvz* am fernen Rechner.

dvz nicht angegeben:
ftp fordert Sie auf, den Namen des anzulegenden Dateiverzeichnisses anzugeben.

mls[_ferne_datei]...[_lokale_datei]
Schreibt im Unterschied zu *mdir* nur die Namen der angegebenen fernen Dateien in die lokale Datei *lokale_datei*. Ansonsten funktioniert *mls* wie *mdir*.

mode[_**stream**]
Setzt den Dateiübertragungs-Modus auf *stream*. Dieser Modus ist standardmäßig gesetzt. Andere Modi werden zur Zeit nicht unterstützt. Daher erhalten Sie bei der Eingabe des Kommandos immer die Meldung:

```
We only support stream mode, sorry.
```

mput[_lokale_datei]...
Überträgt die angegebenen lokalen Dateien in das aktuelle Dateiverzeichnis am fernen Rechner. Ist *lokale_datei* ein Dateiverzeichnis, dann fordert *ftp* Sie zwar zur Bestätigung auf, gibt Ihnen jedoch anschließend die Fehlermeldung aus:

```
dvz: not a plain file
```

Dateiverzeichnisse können nicht übertragen werden.
Die Zugriffsrechte der neuen, fernen Dateien werden auf *rw-rw-rw-* gesetzt.

Wenn *glob* eingeschaltet ist, werden die angegebenen Dateinamen noch am lokalen Rechner expandiert.

Wenn Sie *nmap* und *ntrans* eingeschaltet und zugehörige Argumente angegeben haben, wird jeder Name einer lokalen Datei entsprechend für den fernen Rechner bearbeitet.

Wenn *prompt* eingeschaltet ist, fordert *ftp* Sie bei jeder zu übertragenden Datei zur Bestätigung auf (siehe *mget*).

lokale_datei nicht angegeben:
ftp fordert Sie auf, die Namen der zu übertragenden Dateien einzugeben.

nmap[_eingabemuster_ausgabemuster]
Schaltet den Mechanismus zur Dateinamen-Expansion ein oder aus. Standard: on. Dieses Kommando ist notwendig, wenn Sie mit Nicht-UNIX-Systemen verbunden sind, die auf andere Dateinamen-Konventionen zurückgreifen.

Dateinamen am fernen Rechner werden expandiert, wenn Sie

- bei *nmap* beide Argumente angeben
- mehrere Dateien mit *mput* übertragen
- eine Datei mit *put* oder *send* übertragen, jedoch keinen Dateinamen für die ferne Datei angeben

Dateinamen am lokalen Rechner werden expandiert, wenn Sie

- bei *nmap* beide Argumente angeben
- mehrere Dateien mit *mget* übertragen
- eine Datei mit *get* oder *recv* übertragen, jedoch keinen Dateinamen für die lokale Datei angeben

Die Expansion erfolgt nach den Mustern, die Sie durch die Argumente angeben. Die Sonderzeichen der Muster müssen mit Gegenschrägstrich \ enwertet werden, wenn sie wörtlich übertragen werden sollen. Sonderzeichen sind *$1, $2, ..., $9, [,]* und , (Komma).

eingabemuster
Muster für die bereits vorhandenen Dateinamen der zu übertragenden Dateien. *eingabemuster* besteht aus wörtlich zu nehmenden Einzelzeichen dieser Dateinamen und/oder Sequenzen der Sonderzeichen *$1* bis *$9*.

Beispiel 1:

Auf dem fernen Rechner liegen folgende drei Dateien:

- *abt3*
- *abt3.personal*
- *abt3.personal.alt*

Sie wollen sie auf den lokalen Rechner übertragen. Das *eingabemuster* könnte z.B. so aussehen:

$1.$2.$3

Die Sonderzeichen sind dann mit folgenden Werten belegt:

- $1: *abt3*
- $2: *personal*
- $3: *alt*

Die Punkte zwischen den Sonderzeichen sind wörtlich aus den originalen Dateinamen zu übernehmen.

ausgabemuster
Muster für den resultierenden Dateinamen der zu übertragenden Dateien. *ausgabemuster* besteht aus wörtlich zu nehmenden Einzelzeichen dieser Dateinamen, Leerzeichen und/oder Sequenzen der Sonderzeichen *$0* bis *$9* sowie *[*, *]* und , (Komma).

$0 wird ersetzt durch den originalen Dateinamen. *$1* bis *$9* werden ersetzt durch die entsprechenden Werte in *eingabemuster*. Eine Sequenz *[$a,$b]* wird ersetzt durch den in *eingabemuster* festgelegten Wert von *$a*, wenn dieser nicht Null ist, ansonsten durch den dort festgelegten Wert von *$b* (*a* und *b*: Zahl von 1 bis 9).

Beispiel 2:

Die in Beispiel 1 genannten Dateien sollen auf dem lokalen Rechner andere Namen erhalten. Sie geben dafür folgendes *ausgabemuster* ein:

m-$1.[$2,proz].[$3,91]

Die neuen Namen werden dadurch folgendermaßen konstruiert:

m-	allen neuen Namen vorauszustellende Zeichenkette
$1	Wert von *$1* aus *eingabemuster*, d.i. *abt3*
.	folgender Punkt
[$2,proz]	Wert von *$2* aus *eingabemuster*, d.i. *personal*, falls im originalen Dateinamen vorhanden, ansonsten Anfügen der Zeichenkette *proz*
.	folgender Punkt
[$3,91]	Wert von *$3* aus *eingabemuster*, d.i. *alt*, falls im originalen Dateinamen vorhanden, ansonsten Anfügen der Jahreszahl *91*

Wenn Sie die Dateien nun mit *mget* übertragen, erhalten Sie auf dem lokalen Rechner die Dateinamen

- *m-abt3.personal.91*
- *m-abt3.personal.alt*
- *m-abt3.proz.91*

Kein Argument angegeben:
Die Funktion *nmap* wird ausgeschaltet.

ntrans[␣eingabezeichen[␣ausgabezeichen]]
Schaltet den Mechanismus zur Übersetzung der Einzelzeichen von Dateinamen ein oder aus. Standard: on. Dieses Kommando ist notwendig, wenn Sie mit Nicht-UNIX-Systemen verbunden sind, die auf andere Dateinamen-Konventionen zurückgreifen.

Zeichen für Dateinamen am fernen Rechner werden übersetzt, wenn Sie

- bei *ntrans* mindestens *eingabezeichen* angeben
- mehrere Dateien mit *mput* übertragen
- eine Datei mit *put* oder *send* übertragen, jedoch keinen Dateinamen für die ferne Datei angeben

Zeichen für Dateinamen am lokalen Rechner werden übersetzt, wenn Sie

- bei *ntrans* mindestens *eingabezeichen* angeben
- mehrere Dateien mit *mget* übertragen
- eine Datei mit *get* oder *recv* übertragen, jedoch keinen Dateinamen für die lokale Datei angeben

eingabezeichen
Zeichen aus den bereits bestehenden Dateinamen der zu übertragenden Dateien. Dieses Zeichen soll in ein anderes Zeichen übersetzt werden. Sie können mehrere Zeichen angeben, die ohne Leerzeichen aneinandergehängt werden müssen. Jedes *eingabezeichen* wird nur einmal übersetzt.

Beispiel 1:

Sie haben auf einem fernen Rechner folgende drei Dateien, die Sie auf den lokalen Rechner übertragen wollen:

- *Helena_KG*
- *Hello_1*
- *Hello_script*

Die Namenskonvention des lokalen Rechners verlangt jedoch, daß Dateinamen weder Großbuchstaben noch Unterstriche enthalten. Außerdem soll jedes *e* in *a* umgewandelt und *script* durch *scr* abkürzt werden. *eingabezeichen* sieht dann wie folgt aus:

HKG_eipt

ausgabezeichen
Zeichen, in das *eingabezeichen* für die neuen Dateinamen übersetzt werden soll. Sie können mehrere Zeichen angeben, die ohne Leerzeichen aneinandergehängt werden müssen. Die Reihenfolge der neuen Zeichen muß denen in *eingabezeichen* entsprechen. Wenn Sie für das oder die letzten Zeichen in *eingabezeichen* keine Entsprechung angeben, so werden diese Zeichen im neuen Dateinamen gelöscht.

Beispiel 2:

ausgabezeichen muß entsprechend für *eingabezeichen* in Beispiel 1 so aussehen:

hkg.a

Wenn Sie die Dateien nun mit *mget* übertragen, erhalten Sie auf dem lokalen Rechner die Dateinamen:

- *halana.kg*
- *hallo.1*
- *hallo.scr*

ausgabezeichen nicht angegeben:
Alle Zeichen aus *eingabezeichen* werden in den neuen Dateinamen gelöscht.

Kein Argument angegeben:
Die Funktion *ntrans* wird ausgeschaltet.

open␣rechner[␣portnummer]
Verbindet den lokalen Rechner mit dem FTP-Dämon am fernen Rechner *rechner*. Sie können außerdem eine *portnummer* angeben; *ftp* versucht dann, den FTP-Dämon über diesen Port zu erreichen.

prompt
Schaltet die Bestätigungs-Aufforderung bei *mdelete, mget* und *mput* ein oder aus. Standard: on. Ist die Funktion ausgeschaltet, so werden alle angegebenen Dateien ohne Nachfrage übertragen oder gelöscht.

proxy[␣ftp-kommando]
Steuert gleichzeitig eine Verbindung zu zwei fernen Rechnern. So können Sie Dateien zwischen den beiden fernen Rechnern übertragen. *proxy* funktioniert jedoch nur, wenn auf dem zweiten fernen Rechner das Kommando *PASV* unterstützt wird (siehe *remotehelp*).

Bei Ihrer erstmaligen Verwendung von *proxy* sollten Sie folgendermaßen vorgehen:

1. Vom lokalen Rechner eine Verbindung zum ersten fernen Rechner aufbauen.
2. Aus der bestehenden Verbindung die Verbindung zum zweiten fernen Rechner aufbauen.
3. Die am zweiten fernen Rechner verfügbaren *ftp*-Kommandos abfragen.
4. Weitere *ftp*-Kommandos ausführen.
5. Die Verbindung zum zweiten fernen Rechner schließen.

Beispiel (ohne Ausgabe)

```
$ ftp ferner-rechner1
ftp> proxy open ferner-rechner2
ftp> proxy ?
ftp> proxy mget *
ftp> proxy close
```

Folgende *ftp*-Kommandos verhalten sich bei *proxy* anders:

open
Die Datei *.netrc* wird nicht nochmals ausgeführt.

close
Definierte Makros werden nicht gelöscht.

get und *mget*
Dateien werden vom ersten fernen Rechner zum zweiten fernen Rechner übertragen.

put, *mput* und *append*
Dateien werden vom zweiten fernen Rechner zum ersten fernen Rechner übertragen.

put[_lokale_datei][_ferne_datei]
Holt die angegebene Datei vom lokalen Rechner und speichert sie auf dem fernen Rechner unter dem Namen *ferne_datei* ab.
Während der Übertragung gelten die aktuellen Werte für Typ, Format, Modus und Struktur.
Die Zugriffsrechte der neuen, fernen Datei werden auf *rw-rw-rw-* gesetzt.

lokale_datei nicht angegeben:
ftp fordert Sie auf, den Namen einer lokalen Datei anzugeben.

ferne_datei nicht angegeben:
Die übertragene Datei erhält den gleichen Namen wie auf dem lokalen Rechner.

Wenn Sie *case*, *nmap* und *ntrans* eingeschaltet und zugehörige Argumente angegeben haben, wird der Name der lokalen Datei entsprechend für den fernen Rechner bearbeitet.

pwd
Gibt den Namen des aktuellen Dateiverzeichnisses am fernen Rechner aus.

quit
Beendet die *ftp*-Sitzung (siehe *bye*).

quote␣arg1␣arg2␣...
Übergibt die angegebenen Argumente an den fernen FTP-Dämon.
Mit *quote* können Experten, die das FTP-Protokoll kennen, einen FTP-Dämon testen und besondere Funktionen ausführen, die nicht durch *ftp*-Benutzerkommandos implementiert sind.

arg1
arg2
ftp-Kommandos.

recv␣ferne-datei[␣lokale-datei]
Holt die angegebene Datei vom fernen Rechner und speichert sie auf dem lokalen unter dem Namen *lokale-datei* ab (siehe *get*).

remotehelp[␣ftp-kommando]
Gibt die Bedeutung des angegebenen fernen *ftp*-Kommandos aus.

ftp-kommando nicht angegeben:
Sie erhalten eine Liste aller am fernen Rechner bekannten *ftp*-Kommandos. Die Kommandos werden in der Form ausgegeben, in der sie vom *ftp*-Dämon verstanden werden, z.B. *CWD* statt *cd*.

rename␣ferne-datei1␣ferne-datei2
Benennt eine ferne Datei *ferne-datei1* in *ferne-datei2* um.

reset
Löscht die Antwort-Warteschlange.
reset synchronisiert die Sequenz der Kommando-Antwort mit dem FTP-Dämon neu. Dies kann notwendig werden, wenn das FTP-Protokoll vom fernen Dämon verletzt wurde.

rmdir␣dvz
Löscht das Dateiverzeichnis *dvz* am fernen Rechner.

runique
Schaltet das Speichern von Dateien auf dem lokalen Rechner unter einheitlichem Namen ein oder aus. Standard: off. Dies betrifft nur Dateien, die Sie mit *get, mget* oder *recv* übertragen. Existieren im lokalen aktuellen Dateiverzeichnis bereits Dateien, die den gleichen Namen haben wie die zu übertragenden, so werden bei eingeschaltetem *runique* die zu übertragenden Dateien mit einem Suffix von *.1* bis *.98* versehen. Bei *.99* erhalten Sie die Fehlermeldung, daß die letzte Umbenennung nicht stattgefunden hat. *runique* zeigt dann den zuletzt vergebenen Namen an.

send␣lokale-datei[␣ferne-datei]
Holt die angegebene Datei vom lokalen Rechner und speichert sie auf dem fernen Rechner unterd dem Namen *ferne-datei* ab (siehe *put*).

sendport
Schaltet die Verwendung von PORT-Kommandos ein bzw. aus. Standard: on. Es handelt sich hierbei um Kommandos der Dämon-Prozesse (siehe *remotehelp*). Diese Funktion kann notwendig sein, wenn Sie mit Rechnern anderer Hersteller kommunizieren.
Ist die Funktion eingeschaltet, so versucht *ftp*, ein PORT-Kommando zu verwenden, wenn er für jeden Datentransfer eine Verbindung aufbaut. Wenn das Kommando mißlingt, verwendet *ftp* den voreingestellten Datenzugang.
Wenn die Funktion ausgeschaltet ist, werden keine PORT-Kommandos verwendet. Dies ist für die *ftp*-Implementationen sinnvoll, die PORT-Kommandos ignorieren, aber fälschlicherweise anzeigen, daß sie sie angenommen haben.

status
Zeigt an, welche Funktionen aktuell am lokalen Rechner ein- oder ausgeschaltet, welche Werte gesetzt und welche Makros definiert sind.

struct[␣**file]**
Setzt die Dateiübertragungs-Struktur auf *file*. Diese Struktur ist standardmäßig gesetzt. Daher erhalten Sie bei der Eingabe des Kommandos immer die Meldung:

```
We only support file structure, sorry.
```

sunique
Schaltet das Speichern von Dateien auf dem fernen Rechner unter einheitlichem Namen ein oder aus. Standard: off. Dies betrifft nur Dateien, die Sie mit *put, mput* oder *send* übertragen. Existieren im fernen aktuellen Dateiverzeichnis bereits Dateien, die den gleichen Namen haben wie die zu übertragenden, so werden bei eingeschaltetem *sunique* die zu übertragenden Dateien mit einem Suffix von *.1* bis *.98* versehen. Bei *.99* erhalten Sie die Fehlermeldung, daß die letzte Umbenennung nicht stattgefunden hat. *sunique* zeigt dann den zuletzt vergebenen Namen an.

tenex
Verändert den Dateiübertragungs-Typ, um auch mit TENEX-Rechnern kommunizieren zu können. *tenex* verwenden Sie für die Kommunikation mit DEC PDP-10 - und PDP-20 - Rechnern, die mit anderen Bytelängen arbeiten.

trace
Schaltet die Datenpaket-Überwachung ein oder aus (derzeit nicht implementiert).

type[␣typ]
Setzt den Dateiübertragungs-Typ auf *typ*. Standard: *ascii*. *typ* kann *binary* (*image*), *ascii* oder *tenex* sein. *tenex* eignet sich bei lokaler Bytegröße von 8 byte zur Kommunikation mit TENEX-Rechnern.

typ nicht angegeben:
ftp gibt den aktuellen Übertragungs-Typ aus.

user␣benutzername[␣kennwort][␣abrechnungsnr]
Meldet Sie als Benutzer an einem fernen Rechner an. Dies brauchen Sie, wenn Sie zwar mit einem fernen Rechner verbunden sind, aber noch keine Verbindung zu einer Benutzerkennung aufgebaut haben, z.B., wenn Sie *ftp* mit der Option *-n* und der Angabe eines Rechnernamens aufgerufen haben.
Sie werden zur Eingabe von *kennwort* aufgefordert. Eine Abrechnungsnummer ist nur erforderlich, wenn Sie mit Rechnern kommunizieren wollen, die mit Abrechnungsnummern arbeiten.

verbose
Schaltet den Protokoll-Modus ein oder aus. Standard: on. Ist die Funktion eingeschaltet zeigt *verbose* alle Antworten des *ftp*-Dämons an und protokolliert den Verlauf der Dateiübertragung am Bildschirm.

Konventionen für lokale Dateinamen

1. Wenn Sie statt eines anzugebenden lokalen Dateinamens einen Bindestrich angeben, wird die Standard-Eingabe zum Lesen bzw. die Standard-Ausgabe zum Schreiben verwendet.
2. Wenn das erste Zeichen eines Dateinamens das Pipezeichen | ist, wird der Rest des Namens als Shell-Kommando interpretiert. Enthält der Dateiname Leerzeichen, so muß er in Anführungszeichen gesetzt werden, z.B.: *dir . "| pg"* .
3. Ist die Dateinamen-Expansion eingeschaltet, dann werden die lokalen Dateinamen entsprechend den Regeln der Bourne-Shell *sh* expandiert (siehe Option *-g* und *ftp*-Kommando *glob*).
4. Bei *mget* sowie bei *get* ohne Angabe eines lokalen Dateinamens erhalten die übertragenen Dateien die gleichen Namen wie auf dem fernen Rechner, es sei denn, Sie haben vor der Übertragung eine oder mehrere der Funktionen *case, nmap, ntrans* oder *runique* eingeschaltet.
5. Umgekehrt ist der lokale Dateiname gleich dem fernen Dateinamen, wenn Sie bei *put* keinen fernen Dateinamen angeben oder Dateien mit *mput* übertragen. Auch diesen Mechanismus können Sie vor der Übertragung durch eine oder mehrere der Funktionen *case, nmap, ntrans* oder *sunique* ändern.

Abbrechen einer Datenübertragung

Die Datenübertragung unterbrechen Sie mit der Unterbrechungstaste Ihrer Tastatur, z.B. DEL, oder mit dem Shell-Kommando *kill*.

Abbruch mit DEL

Wie der Abbruch mit DEL funktioniert, hängt davon ab,

- ob Sie mit *mget, mput, get* oder *put* übertragen
- ob die Funktion *prompt* ein- oder ausgeschaltet ist
- wie und ob das Kommando *ABOR* vom fernen Dämon unterstützt wird

Die Meldungen hierzu sind selbsterklärend.

Wenn Sie eine Übertragung per *mget* oder *mput* mit DEL unterbrechen, und *prompt* war eingeschaltet, so werden Sie gefragt, ob Sie fortfahren wollen. Antworten Sie mit *n*, so wird die Übertragung abgebrochen. Antworten Sie mit ↵, so wird die Übertragung fortgesetzt. In beiden Fällen wird die Datei, bei der Sie DEL gedrückt haben, nicht übertragen.

Abbruch mit *kill*

Das Abbruchsignal, das mit DEL gesendet wird, wird ignoriert, wenn *ftp* bereits alle Arbeiten am lokalen Rechner beendet hat und nun auf eine Antwort vom Dämon des fernen Rechners wartet.

Eine lange Verzögerung in diesem Modus resultiert entweder aus der Verarbeitung des Kommandos *ABOR* durch den fernen Dämon oder einem unerwarteten Verhalten dieses Dämons.
Ist letzteres der Fall, so muß der lokale *ftp*-Prozeß mit dem Shell-Kommando *kill* abgebrochen werden.

Parameter zur Datenübertragung

FTP legt einige Parameter fest, die eine Datenübertragung beeinflussen können. Die Datenübertragung im Netz ist mit verschiedenen Typen der Zeichendarstellung möglich: ASCII-Code, EBCDI-Code, *image* oder eine lokal definierte Bytegröße von 8 bit (vor allem bei PDP-10 - und PDB-20 - Rechnern). Für die ASCII- und EBCDIC-Typen legen weitere Untertypen fest, ob Zeichen, die das vertikale Dateiformat steuern (z. B. Neue-Zeile-Zeichen), weitergegeben oder in das TELNET-Format bzw. in das ASA-Format umgewandelt werden.

ftp unterstützt den ASCII-Darstellungstyp und die lokal definierten Bytegrößen (8 bit) für die Kommunikation mit TENEX-Rechnern.

Für die Dateistruktur gibt es drei Typen:
- *file* (Datei, keine Sätze)
- *record* (Satz)
- *page* (Seite)

ftp unterstützt nur den Standardwert *file*.

Der Übertragungsmodus kann sein:
- *stream* (Datenstrom)
- *block* (blockweise Übertragung)
- *compressed* (komprimierte Daten)

ftp unterstützt nur den Standardwert *stream*.

Die Datei .netrc

Sobald Sie *ftp* oder das *ftp*-Kommando *open* zusammen mit dem Namen eines fernen Rechners eingeben, wird die Datei *.netrc* gesucht und abgearbeitet. Sie enthält Daten für die Anmeldung am fernen Rechner zur Datenübertragung mit *ftp*. Sie muß im HOME-Dateiverzeichnis auf dem lokalen Rechner eingerichtet werden und sollte die Zugriffsrechte *-rw-------* haben, da sie Kennwörter enthalten kann (siehe unten, *password* und *Beispiel 2*).

Die Anmeldedaten werden durch Schlüsselwörter und ihre Werte spezifiziert. Alle Schlüsselwörter können mehrfach angegeben werden. Schlüsselwort und Wert müssen voneinander und vom nächsten Schlüsselwort durch Leerzeichen, Tabulatoren oder Neue-Zeile-Zeichen getrennt werden. Folgende Schlüsselwörter gibt es:

machine_ferner-rechner
: bezeichnet den Rechner, von oder zu dem Sie übertragen wollen. Wenn *ferner-rechner* zu Ihrer Angabe beim *ftp*-Aufruf oder beim Kommando *open* paßt, werden alle weiteren Schlüsselwörter abgearbeitet.

login_benutzerkennung
: bezeichnet eine Benutzerkennung am fernen Rechner.

password_kennwort
liefert ein Kennwort für *benutzerkennung*.
Wenn dieses Schlüsselwort vorhanden ist, bricht *ftp* den Auto-Login-Prozeß ab, insofern die Datei *.netrc* für andere Benutzer als den Eigentümer lesbar ist.

account_kennwort2
liefert ein zusätzliches Kennwort (siehe auch *ftp*-Kommando *account*).

macdef_makro-name
definiert ein Makro (siehe *ftp*-Kommando *macdef*). Wenn Sie das Makro *init* nennen, wird *macdef* als letztes Schlüsselwort bearbeitet.

DATEI

/usr/adm/wtmp
enthält die Protokolle über alle Anmeldungen am System.

Hinweis zu den Fehlermeldungen

ftp gibt Fehlermeldungen aus, die selbsterklärend sind.

BEISPIELE

1. Anmelden am fernen Rechner über die Datei *.netrc*

   Benutzerin *janne* überträgt häufig Daten an einen der Rechner *bogart* und *sam*. Sie erstellt daher die Datei *.netrc*, durch deren Einträge sie automatisch an dem Rechner angemeldet wird, den sie beim *ftp*-Aufruf angibt. Wenn sie sich an *bogart* anmeldet, werden anschließend ebenso automatisch durch das Makro *init* einige Anfangsarbeiten ausgeführt. Im Makro muß jedes Kommando in einer eigenen Zeile stehen, und das Makro muß mit zwei aufeinanderfolgenden Neue-Zeile-Zeichen abgeschlossen werden.
   Die Datei enthält zwei Kennwörter und erhält daher nur Lese- und Schreibrecht für die Benutzerin (*chmod*).

```
cat > .netrc

machine bogart login janne password wrdl.brm
macdef init
verbose
prompt
runique
sunique ↵
↵
machine sam login janne password s.wetter
END

chmod go-rw .netrc
```

2. Beispielsitzung

   Benutzerin *janne* wird durch die Angabe des Rechnernamens *sam* automatisch an diesem Rechner angemeldet (siehe erstes Beispiel). Sie wechselt dort das aktuelle Dateiverzeichnis (*cd*) und laßt sich den Inhalt listen (*ls*). Die Dateien sollen zunächst zum Rechner *bogart* übertragen werden (*proxy open ...*). Da die Datei *.netrc* bei *proxy* nicht ausgeführt wird, meldet sich *janne* explizit an. Die Dateien sollen an *bogart* andere Namen erhalten (*proxy nmap ...*). Anschließend legt *janne* auf dem lokalen Rechner ein neues Dateiverzeichnis an (*!, mkdir*), wechselt hinein (*lcd*) und überträgt die Dateien ohne Bestätigungsabfrage (*prompt*) von *sam* auf den lokalen Rechner (*mget*).

   In diesem Beispiel wurden alle Protokollmeldungen weggelassen, sonstige Ausgaben reduziert und Leerzeilen nur der Übersichtlichkeit halber eingfügt.

```
ftp sam

ftp> cd MANUAL
ftp> pwd
/home2/janne/MANUAL

ftp> ls -x
BA.grafiken  BA.kap.1  BA.kap.2  BA.kap.3

ftp> proxy open bogart
Name: janne
Password: geheim

ftp> proxy nmap $1 $2.$3 $2.[$3,ba]
ftp> proxy cd BA
ftp> proxy mget BA.*
ftp> proxy ls -x
grafiken.ba  kap.1  kap.2  kap.3

ftp> proxy close
ftp> !

$ mkdir VERSION2
$ END

ftp> lcd VERSION2
ftp> prompt
ftp> mget BA.*
ftp> bye

$
```

SIEHE AUCH

ls, rcp, tar, sh, tftp
ftpd, netrc [7] und [9]
popen() [19]

gcore
Speicherabzug von laufenden Prozessen (get core)

gcore erzeugt einen Speicherabzug für jeden der angegebenen Prozesse. Der Speicherabzug kann für Arbeiten mit Fehlersuchprogrammen (Debugger), wie *sdb*, verwendet werden.

gcore[-o dateiname] prozessnummer ...

-o nicht angegeben
gcore erzeugt je einen Speicherabzug für jeden Prozeß, der mit seiner Prozeßnummer in der Argumentliste aufgeführt ist. Der Speicherabzug wird in der Datei *core.prozessnummer* abgelegt.

-o dateiname
(o - output file) Der Speicherabzug wird, statt in der Datei *core.prozessnummer*, in der Datei *dateiname.prozessnummer* abgelegt.

prozessnummer
Prozeßnummer des Prozesses, für den ein Speicherabzug erzeugt werden soll. Es können mehrere Prozeßnummern angegeben werden.

DATEIEN

./core.prozessnummer
Datei, die den Speicherabzug für den Prozeß mit der Prozeßnummer *prozessnummer* enthält.

BEISPIEL

Geben Sie zunächst mit Hilfe des Kommandos *ps* eine Übersicht über alle laufenden Prozesse aus:

```
$ ps

   PID TTY      TIME COMD
 27623 tty004   0:02 sh
 27691 tty004   0:00 ps
```

In der ersten Spalte stehen die jeweiligen Prozessnummern. Mit *gcore* können Sie jetzt einen Speicherabzug des *sh*-Prozesses in einer Datei ablegen:

```
$ gcore 27623

gcore: core.27623 dumped

$ ls -l

total 64
-rw-r--r--   1 kmdo     kmdoag     30016 Mar 19 13:08 core.27623
-rw-------   1 kmdo     kmdoag       690 Mar 18 14:00 datei
-rw-r--r--   1 kmdo     kmdoag        63 Mar 13 09:29 startup
```

Der Speicherabzug ist in der Datei *core.27623* abgelegt worden und kann jetzt je nach Bedarf weiterverarbeitet werden.

SIEHE AUCH

kill, *ps*
sdb, *ptrace* [19]

gencat
Binär codierten Meldungskatalog erzeugen

Das Kommando *gencat* schreibt den Inhalt einer Meldungstext-Datei in einen binär codierten Meldungskatalog.
Wenn der Meldungskatalog noch nicht existiert, wird er neu erstellt. Wenn der Meldungskatalog bereits existiert, werden die darin enthaltenen Meldungen in den neuen Meldungskatalog mit eingebunden. Wenn die Mengen- und Meldungsnummern von aktuellen und neudefinierten Meldungstexten übereinstimmen, dann ersetzt *gencat* die aktuell im Meldungskatalog stehenden Meldungstexte durch die in der Meldungstext-Datei definierten neuen Meldungstexte.

Mit *gencat* erzeugte Meldungskataloge sind binär codiert, d.h., daß zwischen verschiedenen Maschinentypen keine Kompatibilität garantiert werden kann. Genau wie C-Programme für jeden Maschinentyp neu übersetzt werden müssen, müssen die Meldungskataloge mit *gencat* neu erstellt werden.

Eine Meldungstext-Datei kann einfache Meldungsnummern oder zweistufige Meldungsnummern enthalten. Zweistufige bestehen aus der Meldungsnummer und einer Mengennummer. Bei einfachen Meldungsnummern wird standardmäßig *NL_SETD* für die Mengennummern verwendet.

gencat[_option]_catfile[_msgfile]...

option

-l

Falls *catfile* bereits existiert, werden Informationen über die dort vorhandenen Meldungen auf die Standard-Ausgabe ausgegeben. Die Ausgabe hat folgendes Format:

Mit Option *-X*:
SET mengennr, MESSAGE meldungsnr, TEXT meldungstext
*

Ohne Option *-X*:
SET mengennr, MESSAGE meldungsnr, OFFSET offset, LENGTH meldungslänge
meldungstext
*

offset gibt den Abstand des Meldungstextbeginns vom Anfang an, d.h.
erste Meldung: offset(1) = 0
i-te Meldung: offset(i) = offset(i-1) + meldungslänge(i)

-m

Mit der Option *-m* kann X/Open-konform gearbeitet werden.

Aus Kompatibilitätsgründen zu früheren Versionen von *gencat*, die in einer Reihe von speziellen internationalisierten Produkten veröffentlicht wurden, wird die Option *-m* zur Verfügung gestellt. Sie bewirkt, daß *gencat* eine einzige Datei *catfile* erzeugt, die mit den Formatkatalogen, die von den früheren Versionen erstellt wurden, kompatibel ist. Die Suchroutinen finden heraus, um welchen Katalogtyp es sich handelt, und agieren dementsprechend.

Das Verhalten der Option *-m* ist die Voreinstellung, sie braucht nicht explizit eingeschaltet zu werden.

-X

gencat baut auf dem *mkmsgs*-Dienstprogramm auf. Die *gencat* Datenbank besteht aus zwei Dateien: *catfile.m* ist ein *mkmsgs*-Formatkatalog, *catfile* enthält die Informationen, die zum Übersetzen einer zweistufigen Nummer (set and message number) in eine einfache Meldungsnummer benötigt werden. Einfache Meldungsnummern werden bei *gettxt*-Aufrufen verwendet.

gettxt nimmt an, daß die Kataloge in einem Unterverzeichnis von */usr/lib/locale* abgelegt sind. Diese Einschränkung umgehen Sie, indem Sie einen symbolischen Verweis auf den Katalog */usr/lib/locale/Xopen/LC_MESSAGES* erzeugen, nachdem der Katalog eingerichtet wurde. Über diesen Verweis kann *gettxt* auf den Katalog zugreifen. Der Verweis wird gelöscht, wenn der Katalog geschlossen wird.

Keine Option angegeben
gencat verhält sich, wie bei der Option *-m* beschrieben.

catfile
Name des binär codierten Meldungskatalogs, den *gencat* aus der Meldungstext-Datei *msgfile* erzeugen soll. Die Ausgabe darf nicht auf die Standard-Ausgabe umgelenkt werden.

msgfile
Name der Meldungstext-Datei, die durch *extract* erzeugt wurde. Aus ihr erzeugt *gencat* einen binär codierten Meldungskatalog. Wenn Sie *msgfile* weglassen, liest *gencat* von der Standard-Eingabe. Sie können auch mehrere Meldungsquelldateien angeben.

Format eines Meldungskatalogs

Ein von *gencat* erzeugter Meldungskatalog enthält die folgenden Strukturen in der angegebenen Reihenfolge:

- den Katalog-Kopf *CAT_HDR*, bestehend aus:
  - der Dateiformatkennung (magic number)
  - der Anzahl der Mengen (set) in der Meldungsdatei
  - dem Platz (in byte), den die Datei zum Laden benötigt, die Länge des Dateikopfs nicht mitgerechnet
  - der Position, an der die Meldungsköpfe beginnen, die Länge des Dateikopfs nicht mitgerechnet
  - der Position, an der die Meldungstexte beginnen, die Länge des Dateikopfs nicht mitgerechnet
- einen Mengenkopf *CAT_SET_HDR* für jede vorhandene Menge (set), bestehend aus:
  - der Mengennummer
  - der Anzahl der Meldungen in der Menge
  - dem Start-Offset in der Tabelle
- einen Meldungskopf *CAT_MSG_HDR* für jede vorhandene Meldung, bestehend aus:
  - der Meldungsnummer
  - der Länge der Meldung in byte
  - dem Meldungs-Offset in der Tabelle
- die einzelnen Meldungstexte, die durch \0 voneinander getrennt sind.

BEISPIEL

Erzeugen des binär codierten Meldungskatalogs *catfile* aus der Meldungstext-Datei *msgfile*. Die Meldungstext-Datei wurde vorher mit *extract* erzeugt.

```
$ gencat catfile msgfile
```

SIEHE AUCH

dumpmsg, extract, mkmsgs
catopen(), catgets(), catclose(), gettxt() [19]
nl_types [7] bzw. [19]

getopt
Argumente einer Prozedur nach Optionen durchsuchen (get options)

getopt wird in zukünftigen Versionen nicht mehr unterstützt werden. Es wird durch das eingebaute *sh*-Kommando *getopts* ersetzt. Alte Shell-Prozeduren können Sie mit dem Konvertierungsprogramm */usr/lib/getoptcvt* von *getopt* auf *getopts* umstellen (siehe *getoptcvt*).

getopt liest den Inhalt des Shell-Parameters $* und vergleicht ihn mit dem beim *getopt*-Aufruf angegebenen Argument *optstring*. Aufgrund dieses Vergleichs bringt *getopt* entweder den eingelesenen Inhalt des Shell-Parameters in eine Standard-Form und schreibt ihn so auf die Standard-Ausgabe oder gibt Fehlermeldungen auf die Standard-Fehlerausgabe aus.

getopt können Sie in Shell-Prozeduren zur Analyse der beim Prozedur-Aufruf angegebenen Argumente verwenden. *getopt* unterteilt die in der Kommandozeile angegebenen Argumente in Optionen der Prozedur und ggf. deren Argumente. Die Aufruf-Argumente können so einfacher analysiert und auf ihre Zulässigkeit überprüft werden.

getopt optstring **$***

optstring
: Zeichenkette, die aus Buchstaben und Doppelpunkten : bestehen kann. Die in *optstring* angegebenen Buchstaben werden von *getopt* als zulässige Optionen der Shell-Prozedur betrachtet. Wenn hinter einem Buchstaben ein Doppelpunkt steht, so erwartet *getopt* für diese Option ein Argument.

$*
: Shell-Parameter, der alle Aufrufargumente der zugehörigen Shell-Prozedur enthält.

Analyse der Argumente einer Shell-Prozedur

Beim Aufruf einer Shell-Prozedur weist die Shell dem Stellungsparameter *$0* den Namen der Prozedur und dem Shell-Parameter *$** alle Argumente der Prozedur zu (siehe *sh*). Wenn Sie *getopt* in der Prozedur verwenden, liest *getopt* den Inhalt des Shell-Parameters *$** (d.h. die Argumente der Prozedur) und vergleicht ihn mit *optstring*. Dabei zerlegt *getopt* die angegebene Argumentliste in

- Optionen
- Argumente, die zu einer Option gehören
- sonstige Argumente.

Für die Angabe der Argumente beim Aufruf einer Prozedur schreibt *getopt* die folgenden Regeln vor:

- Die Optionen können in beliebiger Reihenfolge angegeben werden.
- Die Optionen können einzeln, durch Leerzeichen getrennt und jeweils mit einem vorangestelltem Bindestrich - oder zusammengefaßt, ohne trennende Leerzeichen und mit nur einem vorangestellten Bindestrich angegeben werden, z.B.

```
proz -a -b -c
proz -ab -c
proz -abc
proz -b -ac
```

- Bei Optionen, die ein Argument verlangen, kann dieses Argument direkt auf die Option folgen oder von dieser durch Leer- oder Tabulatorzeichen getrennt werden. Von der nachfolgenden Option muß das Argument durch Leer- oder Tabulatorzeichen getrennt werden:

```
proz -a -b arg -c
proz -ab arg -c
proz -abarg -c
proz -acbarg
```

Vorsicht

Eine Gruppe von Argumenten, die von der Option durch ein Leerzeichen getrennt ist und in Anführungszeichen " steht, wird von *getopt* nicht korrekt behandelt:

```
proc -a -b -o "xxx z yy" datei
```

Das Argument *"xxx z yy"* für die Option *-o* wird nicht richtig erkannt. Verwenden Sie in solchen Fällen *getopts* statt *getopt*.

Entsprechen die beim Prozedur-Aufruf angegebenen Optionen der Festlegung *getopt*-Aufruf, gibt *getopt* die Argumente der Prozedur in Standard-Form auf die Standard-Ausgabe aus. Die Optionen werden einzeln, mit führendem Bindestrich und durch Leerzeichen voneinander getrennt ausgegeben. Auf Optionen, die ein Argument haben, folgt dieses Argument durch ein Leerzeichen getrennt. Das Ende der Optionenliste kennzeichnet *getopt* durch zwei Bindestriche --; danach folgen die übrigen Argumente der Prozedur, jeweils durch ein Leerzeichen getrennt.

Beispiel

optstring:	a:b:cdef
Argumente der Prozedur:	-fb arg -caarg -de datei1 datei2
Ausgabe von *getopt*:	-f -b arg -c -a arg -d -e -- datei1 datei2

Siehe auch *Beispiel 1*.

Wenn in der Argumentliste einer Prozedur eine Option nicht zulässig ist oder wenn *getopt* für eine Option, die ein Argument verlangt, kein zugehöriges Argument erkennt, dann gibt *getopt* eine entsprechende Fehlermeldung aus (siehe *FEHLERMELDUNGEN*). Wenn zu einer Option, die kein Argument verlangt, ein Argument angegeben ist, interpretiert *getopt* dieses Argument entweder als Optionenliste oder nimmt an dieser Stelle das Ende der Optionenliste an.

Beispiel

optstring:	abc
Argumente der Prozedur:	-a -barg -c
Ausgabe von *getopt*:	getopt: illegal option -- r getopt: illegal option -- g
optstring:	abc
Argumente der Prozedur:	-a -b arg -c
Ausgabe von *getopt*:	-a -b -- arg -c

Die spezielle Option -- können Sie auch beim Aufruf der Prozedur verwenden, um das Ende der Optionenliste anzuzeigen. Dies ist dann notwendig, wenn ein Argument einer Prozedur, das nicht einer Option zugeordnet ist, mit einem Bindestrich beginnen soll (siehe *Beispiel 1*).

Mit dem eingebauten *sh*-Kommando *set* können Sie die analysierte Argumentliste, die *getopt* auf die Standard-Ausgabe schreibt, weiterverarbeiten: Die Kommandozeile

set -- `getopt ...`

weist das Ergebnis von *getopt* dem Shell-Parameter $* zu. Damit wird der ursprüngliche Inhalt von $* (d.h. die nicht analysierten Aufrufargumente der Prozedur) durch die analysierten Aufrufargumente ersetzt (siehe auch *Beispiel 2*).

FEHLERMELDUNGEN

```
getopt: option requires an argument -- x
```

-x ist im *getopt*-Aufruf als Option mit Argument gekennzeichnet, Sie haben beim Aufruf der Shell-Prozedur, in der dieser *getopt*-Aufruf enthalten ist jedoch kein Argument für *-x* angegebenen.

```
getopt: illegal option --y
```

-y ist nicht in der Liste der zulässigen Optionen (*optstring*) enthalten, Sie haben *-y* jedoch beim Aufruf der Shell-Prozedur angegeben.

BEISPIELE

1. Die Shell-Prozedur *proz* hat folgenden Inhalt:

   ```
   echo $*
   getopt -abo:f $*
   echo $*
   ...
   ```

   Durch den *getopt*-Aufruf wird festgelegt, daß die Prozedur *proz* mit den Optionen *-a*, *-b*, *-o* und *-f* aufgerufen werden kann und daß zu der Option *-o* ein Argument angegeben werden muß.

   Nun rufen Sie die Prozedur auf:

   ```
   $ proz -bf -oarg datei
   -bf -oarg datei
   -b -f -o arg -- datei
   -bf -oarg datei
   ```

   Die *echo*-Ausgaben zeigen, daß die Argumente der Prozedur dem Shell-Parameter $* zugewiesen werden und daß *getopt* an dieser Zuweisung nichts ändert. Da die angegebene Argumentliste korrekt ist, gibt *getopt* keine Fehlermeldung aus, sondern schreibt die Argumentliste in Standard-Form auf die Standard-Ausgabe.

   Rufen Sie nun die Prozedur mit einer unzulässigen Option auf:

   ```
   $ proz -abx datei
   -abx datei
   getopt: illegal option -- x
   -abx datei
   ```

Wenn Sie der Prozedur ein Argument übergeben wollen, das mit einem Bindestrich beginnt, dann müssen Sie beim Prozedur-Aufruf dieses Argument mit -- von der Optionenliste trennen:

```
$ proz -bf -oarg -- -argument
-bf -oarg -- -argument
-b -f -o arg -- -argument
-bf -oarg -- -argument
```

Mit -- kennzeichnen Sie das Ende der Optionenliste; alle auf -- folgenden Argumente interpretiert *getopt* nicht mehr als Optionen bzw. als Argumente zu einer Option.

2. Der folgende Ausschnitt aus einer Shell-Prozedur *proz* zeigt, wie die Argumente einer Prozedur verarbeitet werden können. Für *proz* sind die Optionen *-a*, *-b* und *-o* zulässig, für *-o* ist ein Argument erforderlich. Die Optionen *-a* und *-b* schließen sich gegenseitig aus; werden trotzdem beide angegeben, ist nur die zuletzt angegebene gültig:

```
set -- getopt abo: $*
if [ $? != 0 ]
then
   echo "usage: $0 [-a | -b] [-o <arg>]"
   exit 2
fi
for i in $*
do
   case $i in
   -a | -b)        FLAG=$i; shift;;
   -o)             OARG=$2; shift 2;;
   --)             shift; break;;
   esac
done
```

Durch dieses Programmstück sind z.B. die folgenden Aufrufe von *proz* gleichbedeutend:

```
proz -aoarg datei1 datei2
proz -a- o arg datei1 datei2
proz -oarg -a datei1 datei2
proz -boarg -a datei1 datei2
proz -a -o arg -- datei1 datei2
```

An einem Aufruf der Prozedur *proz* soll das Programmstück erläutert werden. Nehmen wir an, Sie rufen die Prozedur wie folgt auf:

```
$ proz -boarg -a datei1 datei2
```

getopt stellt fest, daß die angegebene Argumentliste korrekt ist, und übergibt diese in folgendem Format an *set*:

```
-b -o arg -a -- datei1 datei2
```

set weist die Elemente dieser Argumentliste der Reihe nach den Stellungsparametern $1, $2, ... und die gesamte Argumentliste dem Shell-Parameter $* zu. Das Ergebnis dieser Zuweisung ist:

```
$1 = -b
$2 = -o
$3 = arg
$4 = -a
$5 = --
$6 = datei1
$7 = datei2
$* = -b -o arg -a -- datei1 datei2
```

Da die beim *proz*-Aufruf angegebene Argumentliste korrekt ist, ist die Bedingung der *if*-Abfrage nicht erfüllt, und die Ausführung der Prozedur wird bei der *for*-Schleife fortgesetzt.

Erster Durchlauf der *for*-Schleife:

$i hat den Wert *-b*. Aufgrund der *case*-Abfrage wird der Wert *-b* der Shell-Variablen FLAG zugewiesen. Nach der *shift*-Anweisung haben die Shell-Parameter $1, $2, ..., $* die folgenden Werte:

```
$1 = -o
$2 = arg
$3 = -a
$4 = --
$5 = datei1
$6 = datei2
$* = -o arg -a -- datei1 datei2
```

Zweiter Durchlauf der *for*-Schleife

$i hat den Wert *-o*, die Prozedur verzweigt in den zweiten Teil der *case*-Abfrage. Aufgrund der *shift*-Anweisung im ersten Durchlauf der *for*-Schleife hat $2 den Wert *arg*; durch die Zuweisung *OARG=$2* wird der Shell-Variablen *OARG* also der Wert des zur Option *-o* gehörenden Arguments zugewiesen. Nach der Anweisung *shift 2* haben die Shell-Parameter die folgenden Werte:

```
$1 = -a
$2 = --
$3 = datei1
$4 = datei2
$* = -a -- datei1 datei2
```

Dritter Durchlauf der *for*-Schleife

$i hat den Wert *arg*. Da es für *arg* keinen *case*-Zweig gibt, geschieht nichts und die Werte der Shell-Parameter $1, $2, ..., $* bleiben unverändert.

Vierter Durchlauf der *for*-Schleife

$i hat den Wert *-a*. Aufgrund der *case*-Abfrage wird der Wert *-a* der Shell-Variablen *FLAG* zugewiesen und damit deren alter Wert *-b* überschrieben. Von den sich ausschließenden Optionen *-a* und *-b* ist also nur die zuletzt angegebene gültig, in diesem Fall *-a*. Die Shell-Parameter haben nach der *shift*-Anweisung die folgenden Werte:

```
$1 = --
$2 = datei1
$3 = datei2
$* = -- datei1 datei2
```

Fünfter Durchlauf der *for*-Schleife

$i hat den Wert --. Aufgrund der *break*-Anweisung im zugehörigen *case*-Zweig wird die *for*-Schleife beendet. Nach der *shift*-Anweisung haben die Shell-Parameter $1, $2, $* die folgenden Werte:

```
$1 = datei1
$2 = datei2
```

Mit einem weiteren Programmstück können nun auch diese Argumente gewonnen und dann zusammen mit den Optionen und deren Argumenten verarbeitet werden.

SIEHE AUCH

getoptcvt, getopts, sh
getopt() [19]

getoptcvt
Konvertierung von getopt-Kommandoaufrufen in getopts-Kommandoaufrufe

getoptcvt liest eine Shell-Prozedur, wandelt die verwendeten Aufrufe von *getopt* in *getopts*-Aufrufe um und schreibt das Ergebnis auf die Standard-Ausgabe.

/usr/lib/getoptcvt[_**-b**]_proz

-b
Die konvertierte Shell-Prozedur *proz* ist auch auf älteren Betriebssystemversionen ablauffähig. */usr/lib/getoptcvt* wandelt die angegebene Shell-Prozedur so um, daß zur Laufzeit festgestellt wird, ob *getopts* oder *getopt* verwendet werden soll.

-b nicht angegeben:
Die konvertierte Shell-Prozedur verwendet immer *getopts*.

proz
Die Shell-Prozedur, die konvertiert werden soll. Die Shell-Prozedur muß korrekte und sinnvolle *getopt*-Aufrufe enthalten wie z.B. *set -- `getopt ... $*`*. Sonst wird ein Fragezeichen '?' ausgegeben, und es erfolgt ein Abbruch mit Fehler.

BEISPIEL

Alle Aufrufe des Kommandos *getopt* innerhalb der Shell-Prozedur *prozalt* sollen in Aufrufe des neueren Kommandos *getopts* umgewandelt werden. Das Ergebnis soll in die Datei *prozneu* geschrieben werden:

```
$ /usr/lib/getoptcvt prozalt >prozneu
```

SIEHE AUCH

getopts, *getopt*

getopts
Argumente einer Prozedur nach Optionen durchsuchen

Das in die Bourne-Shell *sh* eingebaute Kommando *getopts* wird in Shell-Prozeduren verwendet, um die Argumente und Optionen in der Kommandozeile zu analysieren und auf Gültigkeit zu überprüfen. Alle Regeln des Syntax-Standards für Kommandozeilen werden unterstützt.
getopts ersetzt das ältere Kommando *getopt*, das in zukünftigen Betriebssystemversionen nicht mehr unterstützt werden wird. Daher sollten Sie immer *getopts* statt *getopt* verwenden. Vorhandene Shell-Prozeduren können Sie mit *getoptcvt* von *getopt* auf *getopts* umstellen.

getopts können Sie in Shell-Prozeduren zur Analyse der beim Prozedur-Aufruf angegebenen Argumente verwenden. Die einzelnen Optionen und Argumente werden der Reihe nach in Shell-Variablen abgelegt und können so einfach abgefragt bzw. überprüft werden. Wenn in der Argumentliste eine Option nicht zulässig ist oder wenn *getopt* für eine Option, die ein Argument verlangt, kein zugehöriges Argument erkennt, dann gibt *getopt* eine entsprechende Fehlermeldung aus.

Damit alle Prozeduren und Kommandos die Argumentliste einheitlich verarbeiten, sollte die Argumentliste immer mit *getopts* analysiert und auf gültige Optionen untersucht werden.

getopts optstring name[arg]...

optstring
: Zeichenkette, die aus Buchstaben und Doppelpunkten : bestehen kann. Die in *optstring* angegebenen Buchstaben werden von *getopts* als zulässige Optionen der Shell-Prozedur betrachtet. Wenn hinter einem Buchstaben ein Doppelpunkt steht, so erwartet *getopts* für diese Option ein Argument oder eine Gruppe von Argumenten, die durch Leerzeichen oder Tabulatorzeichen von der Option getrennt sein müssen.

name
: Name der Shell-Variablen, die bei jedem Aufruf von *getopts* mit der jeweils nächsten Option belegt wird.

arg ...
: Argumentliste, die von *getopts* analysiert wird.

  arg nicht angegeben:
  getopts analysiert die Argumentliste der Kommandozeile, mit der die Prozedur aufgerufen wurde.

Arbeitsweise

Bei jedem Aufruf von *getopts* wird die Shell-Variable *name* mit dem Wert der jeweils nächsten Option belegt. Die Shell-Variable *OPTIND* wird mit der Nummer des nächsten Arguments belegt, das noch nicht verarbeitet wurde. *OPTIND* wird immer mit dem Wert 1 initialisiert, wenn der Kommandointerpreter *shell* oder eine Shell-Prozedur aufgerufen wird.

Bei Optionen, die ein Argument verlangen, wird die Shell-Variable *OPTARG* mit diesem Argument belegt. Optionen, die ein Argument verlangen, müssen in *optstring* mit einem Doppelpunkt : gekennzeichnet sein.

Wird eine ungültige Option erkannt, so wird die Shell-Variable *name* mit einem Fragezeichen ? belegt. Ungültige Optionen sind solche, die nicht in *optstring* vorkommen.

Wenn das Ende der Optionsliste erreicht wurde, beendet sich *getopts* mit einem Ende-Status ungleich Null. Das Ende der Optionsliste kann auch mit der speziellen Option -- gekennzeichnet werden.

Wird der Wert der Shell-Variablen *OPTIND* verändert oder *getopts* mit unterschiedlichen Argumentlisten aufgerufen, so kann dies zu unerwarteten Ergebnissen führen.

Folgende Abweichungen vom Syntax-Standard für Kommandozeilen werden von *getopts* zwar korrekt behandelt, sollten jedoch nicht verwendet werden, da sie in zukünftigen Betriebssystem-Versionen nicht mehr unterstützt werden:

- Zusammenfassen von Optionen mit Argumenten und Optionen ohne Argumente, z.B.

  ```
  proz -aboxxx datei
  ```

  falls *optstring* den Wert *abo:* hat

- Kein Leerzeichen oder Tabulatorzeichen nach einer Option, die ein Argument verlangt, z.B.

  ```
  proz -ab -oxxx datei
  ```

  falls *optstring* den Wert *abo:* hat

BEISPIEL

Der folgende Ausschnitt aus einer Shell-Prozedur *proz* zeigt, wie die Argumente einer Prozedur verarbeitet werden können. Für *proz* sind die Optionen *-a*, *-b* und *-o* zulässig, für *-o* ist ein Argument erforderlich. Die Optionen *-a* und *-b* schließen sich gegenseitig aus; werden trotzdem beide angegeben, ist nur die zuletzt angegebene gültig:

```
while getopts abo: c
do
      case $c in
      a | b)          FLAG=$c;;
      o)              OARG=$OPTARG;;
      \?)             echo "usage: $0 [-a | -b] [-o <arg>]"
                      exit 2;;
      esac
done
shift `expr $OPTIND - 1`
```

Durch dieses Programmstück sind z.B. die folgenden Aufrufe von *proz* gleichbedeutend:

```
proz -a -b -o "xxx z yy" datei
proz -a -b -o "xxx z yy" -- datei
proz -ab -o xxx,z,yy datei
proz -ab -o "xxx z yy" datei
proz -o xxx,z,yy -b -a datei
```

SIEHE AUCH

getopt, *getoptcvt*, *sh*

gettxt
Zeichenketten in einer Datenbasis für Meldungstexte auffinden (get text)

gettxt sucht in einer Meldungsdatei die einer Meldungsnummer zugeordnete Zeichenkette.

gettxt meldungsdatei:meldungsnummer[,"standard-text"]

meldungsdatei
: Einfacher Name (ohne Pfad-Präfix) der Meldungsdatei, in der *gettxt* nach *meldungsnummer* suchen soll. Der einfache Name kann bis zu 14 Zeichen lang sein und darf keines der folgenden Zeichen enthalten: Null-Byte (\0), Schrägstrich /, Doppelpunkt :

  meldungsdatei muß mit dem Kommando *mkmsgs* angelegt worden sein und muß sich im Dateiverzeichnis */usr/lib/locale/Locale/LC_MESSAGES* befinden. Dabei entspricht der einfache Name des Dateiverzeichnisses */usr/lib/locale/Locale* der Landessprache, in der die Zeichenketten geschrieben sind, die in der darunter liegenden Meldungsdatei enthalten sind. (siehe *setlocale()* [19]).

  In welchem Dateiverzeichnis */usr/lib/locale/Locale* die Meldungsdatei nach der angegebenen *meldungsnummer* durchsucht wird, können Sie mit der Umgebungsvariablen *LC_MESSAGES* (siehe unten) steuern.

meldungsnummer
: Nummer der gesuchten Zeichenkette in *meldungsdatei*. Die in *meldungsdatei* enthaltenen Zeichenketten sind in aufsteigender Reihenfolge von 1 bis n durchnumeriert, wobei *n* die Anzahl der Zeichenketten in *meldungsdatei* ist.

"standard-text"
: Wenn *gettxt* die gesuchte Zeichenkette nicht in der Meldungsdatei finden kann, deren Pfadname durch die Variable *LC_MESSAGES* festgelegt ist, sucht *gettxt* in der Datei */usr/lib/locale/C/LC_MESSAGES/meldungsdatei* nach der *meldungsnummer* zugeordneten Zeichenkette. Wenn *gettxt* auch dort die gesuchte Zeichenkette nicht findet, wird *"Message not found!!"* ausgegeben. Wenn *meldungsdatei* nicht existiert, wird *"standard-text"* ausgegeben.

"standard-text" nicht angegeben oder leer "":
Wenn *gettxt* die gesuchte Zeichenkette nicht in der Meldungsdatei finden kann, deren Pfadname durch die Variable *LC_MESSAGES* festgelegt ist, sucht *gettxt* in der Datei */usr/lib/locale/C/LC_MESSAGES/meldungsdatei* nach der *meldungsnummer* zugeordneten Zeichenkette. Wenn *gettxt* auch dort die gesuchte Zeichenkette nicht findet oder wenn *meldungsdatei* nicht existiert wird *"Message not found!!"* ausgegeben.

DATEIEN

*/usr/lib/locale/Locale/LC_MESSAGES/**
Meldungsdateien für verschiedene Landessprachen, die von *mkmsgs* angelegt wurden.

*/usr/lib/locale/C/LC_MESSAGES/**
Standard-Meldungsdateien, die von *mkmsgs* angelegt wurden.

UMGEBUNGSVARIABLEN

LC_MESSAGES
enthält den Namen des zu durchsuchenden Sprachraums.

LANG
wenn *LC_MESSAGES* nicht gesetzt oder leer ist, wird alternativ der Wert von *LANG* verwendet.

BEISPIEL

Es sind folgende Dateien vorhanden:

Inhalt von *Input_datei1*:

```
Hallo !\n\007
Gute\t... Nacht !\n
Guten Morgen !\n
```

Inhalt von *Input_datei2*:

```
Salut !\n
Bonne nuit !\n
```

Die Meldungsdateien werden folgendermaßen erzeugt und installiert: Erzeugungsverfahren und Installation der Meldungsdateien:

Die Dateiverzeichnisse */usr/lib/locale/german/LC_MESSAGES* und */usr/lib/locale/french/LC_MESSAGES* existieren nicht. Der Systemverwalter legt durch folgenden Aufruf das Dateiverzeichnis */usr/lib/locale/german* an und erzeugt darunter eine Datei namens *hallo*.

```
$ mkmsgs -l german Input_datei1 hallo
```

Der Aufruf

```
$ mkmsgs Input_datei2 hallo
```

erzeugt eine Datei *hallo* im aktuellen Dateiverzeichnis. Um diese danach mit *gettxt* in der Lokale *french* benützen zu können, muß der Systemverwalter folgende Aktionen durchführen:

```
$ mkdir /usr/lib/locale/french
$ mkdir /usr/lib/locale/french/LC_MESSAGES
$ cp hallo /usr/lib/locale/french/LC_MESSAGES/hallo
```

Auf diese Dateien wird folgendermaßen zugegriffen:

Angenommen, *LC_MESSAGES* ist leer und *LANG* hat den Wert *german*:

```
$ gettxt hallo:1 "Hello!\n"
Hallo !
```

Gleichzeitig ertönt das Klingelzeichen.

```
$ gettxt hallo:2 "Good Night!\n"
Gute   ... Nacht !
$ gettxt hallo:3 "Good Morning!\n"
Guten Morgen !
$ LC_MESSAGES=french
$ export LC_MESSAGES
$ gettxt hallo:1 "Hello!\n"
Salut !
$ gettxt hallo:2 "Good Night!\n"
Bonne nuit !\n
$ gettxt hallo:3 "Good Morning!\n"
Message not found!!
```

Der Aufruf *hallo:3* bewirkt die Ausgabe von *Message not found!!*, da es in der Datei */usr/lib/locale/french/LC_MESSAGES/hallo* nur 2 Meldungen gibt.

```
$ LC_MESSAGES=invalid
$ gettxt hallo:1 "Hello!\n"
Hello
$ gettxt hallo:2 "Good Night!\n"
Good Night!
```

Der Aufruf *hallo:1* bzw. der Aufruf *hallo:2* bewirkt die Ausgabe von *"standard-text"*, da es keine Datei */usr/lib/locale/invalid/LC_MESSAGES/hallo* gibt. Ohne die Angabe von *"standard-text"* würde *"Message not found"* ausgegeben werden (*siehe gettxt*).

SIEHE AUCH

exstr, mkmsgs, srchtxt
gettxt(), setlocale() [19]

Kapitel 3, Internationale Umgebung - NLS (Native Language System)

grep
Muster suchen (global regular expression print)

grep liest Zeilen aus einer oder mehreren Dateien oder von der Standard-Eingabe und vergleicht die Zeilen mit einem angegebenen Muster. Ist mittels Optionen nichts anderes angegeben, so schreibt *grep* alle Zeilen, die zu dem Muster passen, auf die Standard-Ausgabe.
Als Muster können Sie einfache reguläre Ausdrücke angeben (siehe *Tabellen und Verzeichnisse, Reguläre Ausdrücke*).
Geben Sie mehrere Eingabedateien an, dann wird jeder Ausgabezeile der Name der betreffenden Datei vorangestellt.

grep[␣option]...␣muster[␣datei]...

Keine Option angegeben
grep gibt alle Zeilen aus, die zu *muster* passen. Geben Sie mehrere Eingabedateien an, dann wird jeder Ausgabezeile der Name der Datei vorangestellt, aus der die Zeile gelesen wurde.

option

-b
(b - block) Jeder Ausgabezeile wird die Nummer des Blockes vorangestellt, in dem sie enthalten ist.
Die Blöcke, aus denen eine Datei besteht, sind je 512 byte groß und werden, mit 0 beginnend, durchnumeriert.
Option *-b* kann hilfreich sein, wenn die Nummern von Blöcken nach dem Kontext ermittelt werden sollen (siehe z.B. das Kommando *od*, Argument *offset*).

-c
(c - count) *grep* gibt nur die Anzahl der gefundenen Zeilen aus (das sind die Zeilen, die *grep* ohne die Option *-c* ausgeben würde, siehe *Beispiel 3*); die Zeilen selbst werden nicht ausgegeben.

-i
(i - ignore) *grep* unterscheidet beim Vergleich nicht zwischen Groß- und Kleinbuchstaben.

-h
(h - hidden) Beim Durchsuchen mehrerer Eingabedateien unterläßt *grep* die Voranstellung der Dateinamen vor jeder Ausgabezeile.

-l

(l - list) *grep* gibt nur die Namen der Dateien aus, die mindestens eine der gefundenen Zeilen enthalten. (das sind die Zeilen, die *grep* ohne die Option *-l* ausgeben würde, siehe *Beispiel 4*). Jeder Dateiname wird nur einmal ausgegeben. Die Zeilen selbst gibt *grep* nicht aus.

-n

(n - number lines)
Jeder Ausgabezeile wird die Zeilennummer aus der betreffenden Eingabedatei vorangestellt, wobei von 1 an numeriert wird. Liest *grep* von der Standard-Eingabe, bezieht sich die Zeilennummer auf die Standard-Eingabe.

-s

(s - silent) Fehlermeldungen, die sich auf nicht vorhandene Dateien oder auf Dateien, für die Sie kein Leserecht haben, beziehen, werden unterdrückt.

-v

(v - vice versa)
grep gibt alle Zeilen aus, die *nicht* zu dem angegebenen Muster passen.

Zusammen mit Option *-c*:
grep gibt nur die Anzahl solcher Zeilen aus.

Zusammen mit Option *-l*:
grep gibt nur die Namen der Dateien aus, die solche Zeilen enthalten.

-y

grep unterscheidet beim Vergleich nicht zwischen Groß- und Kleinbuchstaben.

muster

Einfacher regulärer Ausdruck, mit dem *grep* die Eingabezeilen vergleichen soll (siehe *Tabellen und Verzeichnisse, Reguläre Ausdrücke*).
Enthält *muster* Zeichen, die für die Shell eine Sonderbedeutung haben, dann schließen Sie *muster* in Hochkommas ein: *'muster'*.

datei

Name der Datei, die *grep* durchsuchen soll. Pro Aufruf können Sie mehrere Dateinamen angeben.

datei nicht angegeben:
grep liest die Eingabezeilen von der Standard-Eingabe.

grep, fgrep und egrep

Die Kommandos *grep*, *fgrep* und *egrep* sind von der Oberfläche her weitgehend identisch. Im folgenden sind die wichtigsten Unterschiede zwischen diesen Kommandos aufgeführt.

grep verarbeitet einfache reguläre Ausdrücke. Pro Aufruf können Sie nur einen einzigen regulären Ausdruck angeben.

fgrep verarbeitet nur Zeichenketten. Pro Aufruf können Sie jedoch mehrere Zeichenketten angeben: Die Zeichenketten geben Sie entweder direkt in der Aufrufzeile, getrennt durch Neue-Zeile-Zeichen, an oder Sie übergeben sie in einer Datei.
fgrep kann effizient sehr viele Zeichenketten suchen: *fgrep* sucht jede einzelne Zeile nach allen Zeichenketten ab.

egrep verarbeitet erweiterte reguläre Ausdrücke. Diese umfassen u.a. die einfachen regulären Ausdrücke bis auf eine Ausnahme: Der einfache reguläre Ausdruck \(*regausdruck*\) hat bei erweiterten regulären Ausdrücken keine Sonderbedeutung und wird deshalb auch nicht von *egrep* verarbeitet.
Pro Aufruf können Sie mehrere reguläre Ausdrücke, durch Neue-Zeile-Zeichen getrennt, angeben. *egrep* interpretiert diese Neue-Zeile-Zeichen wie einen senkrechten Strich (Zeichen für die Alternative, siehe *Tabellen und Verzeichnisse, Reguläre Ausdrücke*).
Die regulären Ausdrücke geben Sie entweder direkt in der Aufrufzeile an oder Sie übergeben sie in einer Datei.

ENDE-STATUS

0 Zeilen gefunden
1 keine Zeile gefunden
2 Syntaxfehler oder "Datei kann nicht geöffnet werden". Dieser Ende-Status gilt auch dann, wenn in anderen Eingabedateien Zeilen gefunden wurden.

BEISPIELE

Grundlage für alle Beispiele sind die Dateien *kunden1* und *kunden2*. Sie haben folgenden Inhalt:

kunden1:

```
080685    999.98  20 LE  Art.  038   Fa. Holzinger
120387   1240.25   3 LE  Art.  023   Fa. Wanninger
180588    330.87   1 LE  Art.  332   Fa. Wanninger
```

kunden2:

```
hinterhuber berta, rosenheim, zugspitzstr.1
wanninger herbert, muenchen 5, kirschstr.3
```

1. Zeilen ausgeben, die zu dem angegebenen Muster passen (keine Option und Option *-i*):

   ```
   $ grep Wanninger kunden1 kunden2
   kunden1:120387   1240.25   3 LE  Art.  023   Fa. Wanninger
   kunden1:180588    330.87   1 LE  Art.  332   Fa. Wanninger
   ```

   Wenn Sie auch Zeilen mit kleingeschriebenem *wanninger* ausgeben lassen möchten, geben Sie ein:

   ```
   $ grep -i wanninger kunden1 kunden2
   kunden1:120387   1240.25   3 LE  Art.  023   Fa. Wanninger
   kunden1:180588    330.87   1 LE  Art.  332   Fa. Wanninger
   kunden2:wanninger herbert, muenchen 5, kirschstr.3
   ```

   Kompliziertere Muster stellen Sie mit Hilfe von regulären Ausdrücken dar, z.B.:

   Einträge des Jahres 1987 in der Datei *kunden1* ausgeben - das sind alle Zeilen, die in der 5. und 6. Spalte die Zahl 87 enthalten:

   ```
   $ grep '^....87' kunden1
   120387   1240.25   3 LE  Art.  023   Fa. Wanninger
   ```

2. Zeilen ausgeben, die *nicht* zu dem angegebenen Muster passen (Option-*v*):

   ```
   $ grep -v '^1' kunden1 kunden2
   kunden1:080685    999.98  20 LE  Art.  038   Fa. Holzinger
   kunden2:hinterhuber berta, rosenheim, zugspitzstr.1
   kunden2:wanninger herbert, muenchen 5, kirschstr.3
   ```

3. Anzahl der gefundenen Zeilen ausgeben (Option *-c*):

   Zuerst soll für jede Eingabedatei die Anzahl der Zeilen, die mit einer 1 beginnen, ausgegeben werden.

```
$ grep -c '^1' kunden1 kunden2
kunden1:2
kunden2:0
```

   Nun soll die Anzahl der Zeilen, die *nicht* mit einer 1 beginnen, ausgegeben werden.

```
$ grep -c -v '^1' kunden1 kunden2
kunden1:1
kunden2:2
```

4. Nur Dateinamen ausgeben (Option *-l*):

   Zuerst sollen die Namen der Dateien, die Zeilen mit einer 1 am Anfang enthalten, ausgegeben werden.

```
$ grep -l '^1' kunden1 kunden2
kunden1
```

   Nun sollen die Namen der Dateien, die Zeilen ohne eine 1 am Anfang enthalten, ausgegeben werden.

```
$ grep -l -v '^1' kunden1 kunden2
kunden1
kunden2
```

5. Gefundene Zeilen mit Zeilennummer ausgeben (Option *-n*):

```
$ grep -n -i wanninger kunden1 kunden2
kunden1:2:120387    1240.25    3 LE  Art.  023   Fa. Wanninger
kunden1:3:180588     330.87    1 LE  Art.  332   Fa. Wanninger
kunden2:2:wanninger herbert, muenchen 5, kirschstr.3
```

SIEHE AUCH

ed, *egrep*, *fgrep*, *sed*, *sh*
stdio [19]

groups
Gruppe eines Benutzers ausgeben

groups gibt die Zugehörigkeit eines Benutzers zu einer Benutzergruppe auf die Standard-Ausgabe aus.

groups[_ benutzername]

benutzername
Sie können den Benutzer angeben, dessen Gruppenzugehörigkeit Sie interessiert. Auf der Standard-Ausgabe erscheint der Gruppenname des angegebenen Benutzers.

benutzername nicht angegeben:
groups gibt die Gruppe des Benutzers an, der an der Datensichtstation eingeloggt ist.

DATEIEN

/etc/passwd
Die Datei */etc/passwd* enthält neben den Benutzerkennungen auch die Gruppennummer jedes Benutzers.

/etc/group
Die Datei */etc/group* enthält alle eingerichteten Benutzergruppen. Benutzer können zusätzlich zu der Gruppe, die in der Datei */etc/passwd* für sie festgelegt ist, auch den Gruppen angehören, die in der Datei */etc/group* stehen.

BEISPIELE

1. Der Benutzer Hans möchte seine Gruppenzugehörigkeit erfahren.

```
$ groups ↵
gruppe1
```

2. Die Gruppenzugehörigkeit seiner Kollegin Ilse erfährt Hans, wenn er den Benutzernamen eingibt.

```
$ groups Ilse ↵
gruppe2
```

SIEHE AUCH

group, passwd [7]

hash
Hash-Tabelle der Shell bearbeiten

Das in die Bourne-Shell *sh* eingebaute Kommando *hash* hat zwei Funktionen:

- Es schreibt den Inhalt der Hash-Tabelle auf die Standard-Ausgabe oder trägt das angegebene Kommando in die Hash-Tabelle ein (Format 1).
- Es löscht den Inhalt der Hash-Tabelle (Format 2).

Jede Shell hat ihre eigene Hash-Tabelle, in die sie alle Kommandos einträgt, die mit ihrem einfachen Dateinamen aufgerufen wurden. Wenn Sie ein Kommando mit dem einfachen Dateinamen aufrufen, sucht die Shell dieses Kommando zuerst in der Hash-Tabelle. Dadurch beschleunigt sich der Suchvorgang (siehe *sh*). Steht dieses Kommando noch nicht in der Hash-Tabelle der aktuellen Shell, wird es neu eingetragen.

Wenn Sie eine Subshell starten, ist die Hash-Tabelle der neuen Shell noch leer. Wenn Sie die Subshell beenden, meldet sich die übergeordnete Shell mit ihrer Hash-Tabelle zurück. Die Hash-Tabelle der Subshell ist gelöscht.

Die Beschreibung des Kommandos *hash* in der Korn-Shell finden Sie unter *ksh*.

hash[␣name]...	Format 1
hash ␣**-r**	Format 2

Format 1: Hash-Tabelle ausgeben oder erweitern

hash[␣name]...

name
: einfacher Dateiname eines Kommandos, einer ausführbaren Shell-Prozedur oder eines ausführbaren Programms. Die Shell sucht diese Datei entsprechend dem Inhalt der Variablen *PATH*. Wenn die Shell die Datei gefunden hat, übergibt sie an *hash*:

  - den entsprechenden relativen Pfadnamen der Form *./name*, wenn das aktuelle Dateiverzeichnis der Variablen *PATH* zugewiesen ist und *name* enthält, oder
  - den absoluten Pfadnamen.

  Das Kommando *hash* trägt diesen Pfadnamen in die Hash-Tabelle ein. Ist *name* in keinem Dateiverzeichnis enthalten, das der Variablen *PATH* zugewiesen ist, so erhalten Sie eine Fehlermeldung.

Für *name* können Sie nicht angeben:

- Shell-Prozeduren, für die Sie kein Ausführrecht haben,
- Kommandos, deren *name* einen Schrägstrich / enthält. Sie können also keine absoluten oder relativen Pfadnamen angeben.
- eingebaute *sh*-Kommandos, da sie Unterprogramme von *sh* sind. Ihr Name ist also nicht der Name einer ausführbaren Datei.

Die Shell trägt ebenfalls nur ausführbare Shell-Prozeduren und Kommandos ein, wenn Sie sie mit ihrem einfachen Dateinamen aufrufen.

name nicht angegeben:
Das Kommando *hash* schreibt den Inhalt der Hash-Tabelle auf die Standard-Ausgabe. Die Ausgabe sieht wie folgt aus:

```
hits    cost    command
1       1       /bin/mail
1       2       /usr/bin/tar
1       1       /bin/ls
1       1       /bin/cat
1       1       /bin/chmod
```

Dabei bedeuten die Angaben in den einzelnen Spalten:

hits
: wie oft die Shell dieses Kommando seit dem Eintrag in die Tabelle bereits ausgeführt hat. Die Ziffer 0 kennzeichnet Kommandos, die bisher nur eingetragen, aber noch nicht ausgeführt wurden.
Ein Stern * direkt nach der Zahl kennzeichnet ausführbare Dateien, die nicht in den Standard-Dateiverzeichnissen für *PATH*, also in */bin* bzw. */usr/bin* enthalten sind.

cost
: beim wievielten Pfadnamen in der Variablen *PATH* die Shell das angegebene Kommando gefunden hat.

command
: absoluter Pfadname der ausführbaren Datei bzw. ein Pfadname der Form *./name* für eine Datei im aktuellen Dateiverzeichnis.

Format 2: Hash-Tabelle löschen

hash␣-r

-r
Den Inhalt der Hash-Tabelle löschen.

Wenn Sie den Wert der Variablen *PATH* ändern, wird der Inhalt der Hash-Tabelle automatisch gelöscht.
Wenn Sie die aktuelle Shell beenden, ist auch die Hash-Tabelle als Bestandteil dieser Shell gelöscht.

Sie sollten den Inhalt der Hash-Tabelle in der aktuellen Shell löschen, wenn folgender Fall eingetreten ist:

Sie haben eine ausführbare Datei in einem Dateiverzeichnis erstellt, dessen Pfadname der Variablen *PATH* zugewiesen ist. Ein anderes Dateiverzeichnis, dessen Pfadname ebenfalls der Variablen *PATH* zugewiesen ist, enthält bereits eine Kommando-Datei gleichen Namens, die in der aktuellen Hash-Tabelle eingetragen ist. Dann führt die Shell immer das ältere Kommando aus, auch wenn in der Variablen *PATH* das zugehörige Dateiverzeichnis hinter dem zuerst genannten eingetragen ist.

Wenn Sie die Hash-Tabelle mit *hash -r* löschen, wird beim nächsten Aufruf das Kommando gestartet, das die Shell zuerst findet. In diesem Fall entscheidet also die Reihenfolge der Einträge in der Variablen *PATH*. Der Pfadname des zuerst gefundenen Kommandos wird in die Hash-Tabelle eingetragen; d.h., dieses Kommando wird ab jetzt immer ausgeführt, wenn Sie es mit seinem einfachen Dateinamen aufrufen.

FEHLERMELDUNG

name: not found

Diese Fehlermeldung kann eine der folgenden Ursachen haben:

- *name* ist in keinem der Dateiverzeichnisse enthalten, deren Pfad der Variablen *PATH* zugewiesen ist.
- *name* ist zwar in einem dieser Dateiverzeichnisse enthalten, aber nicht ausführbar.
- *name* ist ein eingebautes *sh*-Kommando oder eine Shell-Funktion.

UMGEBUNGSVARIABLE

PATH
Suchpfad der Shell

BEISPIELE

1. Den Inhalt der Hash-Tabelle ausgeben:

```
$ echo $PATH
/bin:/usr/bin:.
$ hash
hits    cost    command
1       1       /bin/mail
1       2       /usr/bin/tar
2       1       /bin/ls
3       1       /bin/cat
1*      3       ./stich
1       1       /bin/uname
```

2. Ein Kommando in die Hash-Tabelle eintragen:

```
$ hash lpr
$ hash
hits    cost    command
.
.
0       1       /bin/lpr
.
.
```

Das Kommando */bin/lpr* wurde neu in die Hash-Tabelle eingetragen. Es wurde noch nicht ausgeführt, und die Shell hat es in dem Dateiverzeichnis gefunden, dessen Pfadname in der Variablen *PATH* an erster Stelle steht.

3. Der folgende Bildschirm-Dialog soll zeigen, wann die Hash-Tabelle gelöscht wird:

```
$ hash
hits    cost    command
1       1       /bin/mail
1       2       /usr/bin/tar
2       1       /bin/ls
3       1       /bin/cat
0       1       /bin/lpr
1*      3       ./stich
1       1       /bin/uname
$ sh
$ hash
hits    cost    command
$ who
...
$ ls
...
$ hash
hits    cost    command
1       1       /bin/who
1       1       /bin/ls
$ END
$ hash
hits    cost    command
1       1       /bin/mail
1       2       /usr/bin/tar
2       1       /bin/ls
3       1       /bin/cat
0       1       /bin/lpr
1*      3       ./stich
1       1       /bin/uname
```

In der neu gestarteten Subshell ist die Hash-Tabelle leer. Wenn die Subshell beendet wird, gilt wieder die Hash-Tabelle der übergeordneten Shell. Die Hash-Tabelle der Subshell ist gelöscht.

SIEHE AUCH

sh, *ksh*

hashcheck
Überprüfung einer Rechtschreibeliste

hashcheck gehört, zusammen mit den Kommandos *spellin* und *hashmake* zu einer Gruppe ergänzender Kommandos für das Rechtschreibprüfprogramm *spell*. *spell* benutzt Vergleichslisten für die Rechtschreibüberprüfung. Diese Wortlisten können mit den angeführten Kommandos gepflegt und erweitert werden. Auch neue Wortlisten können erstellt werden.

/usr/lib/spell/hashcheck wortliste

hashcheck liest die in *wortliste* angegebene komprimierte Rechtschreibeliste, die mit *spellin* erstellt wurde, und erzeugt daraus wieder den neunstelligen Original-Hash-Code, wie er von *hashmake* erstellt wurde. Dieser wird auf die Standard-Ausgabe geschrieben.

Als Eingabe werden die Systemdateien *hlista* oder *hlistb* verwendet, oder sonstige Dateien, die mit *spellin* erstellt oder modifiziert wurden.

Um zu überprüfen, ob ausgewählte Wörter in einer bestehenden Rechtschreibeliste vorhanden sind, kann man die Ausgabe von *hashcheck*, angewendet auf die bestehende Rechtschreibeliste, und die von *hashmake*, angewendet auf die Liste der ausgewählten Worte, vergleichen.

SIEHE AUCH

hashmake, *spell*, *spellin*

hashmake
Erstellen einer Hash-Liste

hashmake gehört, zusammen mit den Kommandos *spellin* und *hashcheck* zu einer Gruppe ergänzender Kommandos für das Rechtschreibprüfprogramm *spell*. *spell* benutzt Vergleichslisten für die Rechtschreibüberprüfung. Diese Wortlisten können mit den angeführten Kommandos gepflegt und erweitert werden. Auch neue Wortlisten können erstellt werden.

/usr/lib/spell/hashmake

hashmake liest eine Liste von Wörtern von der Standard-Eingabe und schreibt die codierten Wörter als Hash-Liste auf die Standard-Ausgabe. Das Hash-Verfahren produziert einen neunstelligen Code.

Dies ist der erste Schritt, um eine neue Rechtschreibeliste (siehe *spell*) zu erzeugen oder um zusätzliche Wörter an eine bereits bestehende Liste anzuhängen.

hashmake muß immer vor *spellin* benutzt werden.

SIEHE AUCH

hashcheck, *spell*, *spellin*

hd
Dateiinhalt hexadezimal ausgeben (hex dump)

hd gibt den Inhalt von Dateien hexadezimal, oktal, dezimal oder als Zeichenfolge aus. Die Lage von Zeichen innerhalb eines Zeichensatzes ist ebenfalls darstellbar.

```
hd[ -format]  [ -A][ -t][ -s offset[*][w|bk]][ -n zähler[*][w|bk]][ datei]...
```

Weder Format noch Offset noch Zähler angegeben
: *hd* ist identisch mit *hd -abx -A*. D.h., Adressen und Bytes werden hexadezimal ausgegeben. Zusätzlich gibt *hd* alle Bytes, die druckbare Zeichen darstellen, als solche aus und schreibt für die nichtdruckbaren einen Punkt. Dabei stehen die Adressen am Anfang jeder Zeile, die Hexadezimaldarstellung der Bytes folgt in den nächsten Spalten und die Darstellung als Buchstaben bzw. Punkt steht in jeder Zeile ganz rechts.

  Die Adressen werden relativ zum Dateianfang gezählt. Fehlt die Angabe einer Datei, dann wird von der Standard-Eingabe gelesen, sonst wird der Inhalt der angegebenen Dateien aufgelistet.

-format
: Format, das festlegt, wie einzelne Byteblöcke interpretiert und ausgegeben werden sollen.
  Ein Format setzt sich zusammen aus

  - der Byteblockangabe (*a*, *b*, *c*, *l* oder *w*) und
  - der Interpretationsart, gemäß der ein Byteblock auszugeben ist: hexadezimal (*x*), dezimal (*d*) oder oktal (*o*).

  Innerhalb eines Formats werden sämtliche angegebenen Interpretationsarten auf alle angegebenen Byteblöcke angewendet. Formatangaben können zusammengesetzt und wiederholt werden, um Adressen, Zeichen, Wörter etc. verschieden auszugeben. Sie können z.B. *-ax -bx* zusammenfassen zu *-abx*. Oder Sie können mit *-cxdo* alle Zeichen hexadezimal, dezimal und oktal ausgeben lassen.

  Byteblockangabe

  a
  : (a - address) Formatangabe für Adressen. Adressen werden nur auf eine Art interpretiert, hexadezimal, oktal oder dezimal. Die Adresse steht immer am Anfang jeder auszugebenden Zeile bzw. in der ersten Zeile eines Ausgabeblocks, falls die Formate mehrere Zeilen beanspruchten.

  b
  : (b - byte) Formatangabe für Bytes.

c

(c - character) Formatangabe für Zeichen. Alle druckbaren Zeichen werden ausgegeben. C-Escape-Zeichen gibt *hd* so aus, wie sie in der Sprache definiert sind und die restlichen Zeichen oktal, hexadezimal oder dezimal, je nach Interpretationsart.

l

(l - long word) Formatangabe für 4 byte.

w

(w - word) Formatangabe für 2 byte.

Interpretationsart

x

(x - hexadecimal) *hd* interpretiert Adressen oder Byteblöcke als Hexadezimalzahlen.

d

(d - decimal) *hd* interpretiert Adressen oder Byteblöcke als Dezimalzahlen.

o

(o - octal) *hd* interpretiert Adressen oder Byteblöcke als Oktalzahlen.

Keine Interpretationsart, aber Byteblockangabe angegeben:
hd interpretiert gemäß *-xdo*.

Keine Byteblockangabe, außer Adressen, angegeben:
hd verwendet zusätzlich zum angegebenen Adreßformat *-bx*.

Keine Byteblockangabe, aber Interpretationsart angegeben:
hd interpretiert *-acbwl*.

-format nicht angegeben:
hd verhält sich wie *hd -abx -A*.

-A

(A - ASCII) *hd* gibt alle druckbaren Zeichen aus, für nichtdruckbare Zeichen steht ein Punkt. Die Zeichen werden in der Spalte ausgegeben, die rechts auf das erste Ausgabeformat folgt.

-t

(t - text) Ist diese Option gesetzt, dann ignoriert *hd* alle Formatangaben, die keine Adressen betreffen. *hd* druckt jede Textzeile mit der Adreßangabe am Zeilenanfang aus. Zu lange Zeilen werden unterteilt. Kontrollzeichen (Wert 0x00 bis 0x1f) werden entsprechend als Zeichen ^@ bis ^_ ausgegeben. Bytes, bei denen das höchste Bit gesetzt ist, werden, mit einer Tilde ~ davor, ohne das höchste Bit ausgegeben. Den Zeichen Dach ^, Tilde ~ und Gegenschrägstrich \ wird bei der Ausgabe ein Gegenschrägstrich vorangestellt. In Spezialfällen werden Werte numerisch repräsentiert, z.B. DELETE (127) 7-Bit als \177 und DELETE (255) 8-Bit als \377.

-s␣offset[*][**wlbk**]

Relative Adresse, ab der die Ausgabe des Dateiinhalts beginnen soll. Falls Sie keine Datei angeben oder über eine Pipe eingeben, werden entsprechend viele Bytes überlesen. *hd* bricht die Behandlung der aktuellen Datei ab, falls die Adreßangabe fehlerhaft ist.

Die relative Adresse besteht aus einer Zahl, die Sie dezimal, hexadezimal (mit *0x* davor) oder oktal (mit *0* davor) angeben und optional einer Maßeinheit, die Sie direkt an die Zahl anhängen. Mögliche Einheiten sind:

w
2 byte (d.h. ein Wort)

l
4 byte (d.h. ein langes Wort)

b
512 byte (d.h. ein halbes Kbyte)

k
1024 byte (d.h. ein Kbyte)

Um eine Hexadezimalzahl, die ja die Ziffer *b* enthalten darf, von der Einheit *b* zu unterscheiden, schreiben Sie in diesem Fall zwischen die Zahl und die Einheit *b* einen Stern *.

Mögliche Offsetangaben sind z.B.:
-s 111 (111 byte), *-s 124l* (496 byte), *-s 0xa*b* (5120 byte), *-s 011k* (9216 byte).

-s␣offset[*][wlbk] nicht angegeben:
hd gibt ab Dateianfang aus.

-n␣zähler[*][**wlbk**]

Anzahl der Bytes, die *hd* ausgeben soll. Die Byte-Anzahl geben Sie ebenso wie die Relativadresse an: dezimal, hexadezimal oder oktal mit eventuell folgendem *w*, *l*, *b* oder *k* (siehe *-s*).

datei
Name der Datei, die *hd* auflisten soll. Pro Aufruf können Sie mehrere Dateinamen angeben.

datei nicht angegeben:
hd liest die Eingabezeilen von der Standard-Eingabe.

SIEHE AUCH

od

head
Anfang einer Datei ausgeben

head schreibt die ersten Zeilen aus dem Inhalt einer Datei auf die Standard-Ausgabe. Wenn Sie keine Datei angegeben haben, liest *head* von der Standard-Eingabe.

head[_-n][_datei]

-n
Anzahl der auszugebenden Zeilen

-n nicht angegeben:
Die ersten 10 Zeilen werden ausgegeben.

datei
Name der Eingabedatei. Sie können auch mehrere Dateien angeben, die nacheinander bearbeitet werden. Wenn mehrere Dateien angegeben werden, dann beginnt die Ausgabe mit:
==>datei<==

BEISPIEL

Sie wollen sich die ersten 5 Zeilen von drei Dateien ansehen. Dazu geben Sie das Kommando in folgender Form an:

```
$ head -5 datei1 datei2 datei3
```

Der Inhalt der drei Dateien wird folgendermaßen auf die Standard-Ausgabe geschrieben:

```
==>datei1<==
Inhalt von Datei 1

==>datei2<==
Inhalt von Datei 2

==>datei3<==
Inhalt von Datei 3
```

SIEHE AUCH

cat, more, pg, tail

hostname
Rechnernamen ausgeben und festlegen

hostname gibt den Namen des Rechners aus, an dem ein Benutzer arbeitet (Format 1).

Der Systemverwalter kann mit *hostname* den Rechnernamen festlegen bzw. ändern (Format 2).

Der festgelegte Rechnername wird im Begrüßungsbildschirm angezeigt.

hostname	Format 1
/usr/ucb/hostname␣rechnername	Format 2

Format 1: Den Rechnernamen ausgeben

hostname

hostname gibt den Namen des Rechners aus, an dem ein Benutzer aktuell arbeitet.

Format 2: Den Rechnernamen festlegen bzw. ändern

hostname␣rechnername

Format 2 ist nur für den Systemverwalter bestimmt.

rechnername
: Name, den der Rechner erhalten soll. Wird der Rechnername vom Systemverwalter festgelegt oder geändert, so gilt diese Festlegung nur während eines Systemlaufs. Soll der Rechnername auch nach Systemende erhalten bleiben, muß das in der Datei */etc/rc2.d/S11uname* eingetragen werden (siehe *Beispiel 2*).

Vorsicht

Ist der Rechner in einem Netz integriert (z.B. LAN), so darf der Rechnername nicht einfach mit *hostname* geändert werden. Das kann zu Inkonsistenzen im Netz führen, wenn der Rechner im Netz unter einem anderen Namen bekannt ist. Bei einer gewünschten Namensänderung sind die Konventionen der entsprechenden Netzsoftware zu berücksichtigen.

DATEI

/etc/rc2.d/S11uname
: Datei, in der der Rechnername eingetragen ist. Die Datei wird beim Systemstart ausgeführt.

BEISPIELE

1. Ein Benutzer möchte den Namen des Rechners wissen, an dem er gerade arbeitet.

```
$ hostname
portugal
```

   Der Name des Rechners ist *portugal*.

2. Ein Systemverwalter möchte den Rechner *prittric* in *italien* umbenennen. Die Festlegung soll über das Systemende hinaus gültig sein. Dazu trägt er in der Datei */etc/rc2.d/S11uname*, in der *uname -S prittric* steht, folgendes ein:

```
/usr/ucb/hostname italien
```

   Um diesen Eintrag wirksam werden zu lassen, muß das System neu gestartet oder die Datei */etc/rc2.d/s11uname* mit *sh* ausgeführt werden.

SIEHE AUCH

uname

i386
Wahrheitswert über Prozessoridentität zurückgeben

i386 gibt in SINIX V5.41 den Ende-Status 0 ("wahr") zurück.

i386

Die folgenden Kommandos prüfen ebenfalls den Prozessortyp; sie liefern jedoch alle bei einem Intel-386-Prozessor einen Ende-Status ungleich 0 ("falsch") zurück:

i286
iAPX286
mc68k
pdp11
u370
u3b
u3b15
u3b2
u3b5
vax

i386 sowie die anderen hier aufgeführten Kommandos werden häufig in Anweisungsdateien für *make* (makefiles) und in Shell-Prozeduren verwendet, um die Portabilität zu steigern.

ENDE-STATUS

0 bei einem Intel-386-Prozessor
≠0 bei anderen Prozessoren

BEISPIEL

Die Shellprozedur *proztest* ermittelt nach einem Aufruf von i386 aus dem Ende-Status des Kommandos, welchen Prozessor Ihr Rechner besitzt:

```
$ cat proztest
i386
case $? in
     0) echo "Ihr Rechner besitzt einen Intel-386-Prozessor.";;
     *) echo "Unbekannter Prozessor."
esac
$ proztest
Ihr Rechner besitzt einen Intel-386-Prozessor.
```

SIEHE AUCH

sh, *test*, *true*, *uname*
make [19]

ic
internationale Datenbasis übersetzen (internationalisation compiler)

ic erzeugt aus einer Quelldatei für Datenbasen Zwischendateien, die der Systemverwalter zu Datenbasen im 5.4-Format weiterverarbeiten kann.
Ist die Option *-VSINIX* gesetzt, dann verhält sich *ic* wie unter den SINIX-Versionen 5.22 und 5.23 und erzeugt aus der Quelldatei eine binäre internationale NLS-Datenbasis. *ic* liest den Quelltext entweder aus einer Datei oder von der Standard-Eingabe. Die Ausgabe schreibt *ic* in eine oder mehrere Dateien (siehe Option *-o*).

ic benutzt den C-Präprozessor des C-Entwicklungssystems.

ic[␣präprozessor-option]...[␣option]...[␣quelldatei]

präprozessor-option

präprozessor-option kann sein: *-D*, *-U* oder *-I*.
Diese Optionen werden an den C-Präprozessor weitergereicht.

-Dname[=def]
Der symbolischen Konstanten *name* wird der Wert *def* zugewiesen. Diese Option wirkt wie die Präprozessor-Anweisung *#define*.

=*def* nicht angegeben:
name erhält den Wert 1.

-Uname
Definitionen, bei denen der symbolischen Konstanten ein Wert zugewiesen wird, werden entfernt. Diese Option wirkt wie die Präprozessor-Anweisung *#undef*.

-Idir
Include-Dateien werden auch im Dateiverzeichnis *dir* gesucht.

Keine *option* angegeben
ic erzeugt mehrere Zwischendateien und ein Shell-Skript, mit dessen Hilfe der Systemverwalter eine Datenbasis im 5.4-Format erstellen kann.

option

Die Optionen *-V5.4* und *-VSINIX* dürfen nicht zusammen verwendet werden. Die Option *-M* darf nicht zusammen mit *-VSINIX* benutzt werden.

-V5.4
Diese Option ist standardmäßig gesetzt. *ic* erzeugt mehrere Dateien und ein Shellskript, mit dessen Hilfe der Systemverwalter eine Datenbasis im 5.4-Format erstellen kann.

-VSINIX
ic erzeugt aus der Quelldatei eine binäre NLS-Datenbasis.

-M␣parameter
ic verwendet bei der Erstellung der Zwischendateien Informationen, die durch *parameter* festgelegt sind. *parameter* ist der Parameter, den Sie als Komponente *@parameter* bei der Shellvariablen *LANG* gesetzt haben.

-v
Wenn die Quelldatei erfolgreich übersetzt ist, werden auf die Standard-Fehlerausgabe statistische Informationen ausgegeben zu:

- Anzahl der Ein- und Mehrbyte-Zeichen
- Anzahl und Art (z.B. Sortierreihenfolge) der Tabellen in der Quelldatei
- Größe der erzeugten Binärdatei.

-o␣ausgabe
ausgabe muß eine Datei (Option *-VSINIX*) oder ein Dateiverzeichnis (Standard oder Option *-V5.4*) sein. Standardmäßig oder bei Option *-V5.4* schreibt *ic* die erzeugten Dateien und das Shellskript in das Dateiverzeichnis *ausgabe*. Ist die Option *-M* gesetzt, dann wird an den Namen des Dateiverzeichnisses noch *@parameter* angehängt. Sie müssen selbst darauf achten, daß der Name maximal 14 Zeichen lang ist.

Bei Option *-VSINIX* schreibt *ic* die erzeugte binäre Datenbasis in die Datei *ausgabe*.

-o ausgabe nicht angegeben:
Der Name der Ausgabedatei bzw. des Ausgabedateiverzeichnisses ist der Name des CODESET-Abschnitts in der Quelldatei.

quelldatei
Name der Quelldatei, die übersetzt werden soll.

quelldatei nicht angegeben:
ic liest den Quelltext von der Standard-Eingabe.

ENDE-STATUS

0	wenn erfolgreich übersetzt wurde
≠0	wenn Fehler auftraten, die die Erzeugung einer Binärdatei bzw. der Zwischendateien unmöglich machten

FEHLER

Der *ic*-Compiler liefert vier Arten von Fehlermeldungen:

[warning]
Der Compiler hat ein Problem entdeckt, das mit hoher Wahrscheinlichkeit ein Fehler ist, aber auch vom Benutzer so beabsichtigt sein könnte und die Korrektheit der erzeugten Datei(en) nicht beeinträchtigt.

[error nn**]**
Der Compiler hat einen Fehler entdeckt, der so schwerwiegend ist, daß keine korrekte Binärdatei erzeugt werden kann bzw. keine korrekte Zwischendateien erzeugt werden konnten.

[fatal error]
Der Compiler hat einen Fehler entdeckt, der ein Weiterübersetzen unmöglich macht. Diese Fehlermeldung tritt meist während der Übersetzung der Code-Tabelle auf.

[fatal bug]
Eine solche Fehlermeldung wird ausgegeben, wenn ein interner Fehler im Compiler auftritt. Im Dateiverzeichnis */tmp* werden Temporärdateien abgelegt, die bei der Fehleranalyse helfen können.

DATEIEN

/tmp/icXXXXXX
Temporärdateien

LC_TIME
Zwischendatei, die *ic -V5.4* erstellt

colltbl
Zwischendatei, die *ic -V5.4* erstellt

chrtbl
Zwischendatei, die *ic -V5.4* erstellt

montbl
Zwischendatei, die *ic -V5.4* erstellt

Xopen_info
Zwischendatei, die *ic -V5.4* erstellt

install_locale
Shellskript, das *ic -V5.4* erstellt.
Dieses Shellskript darf nur der Systemverwalter aufrufen.

INTERNATIONALE UMGEBUNG

ic ist 8-bit-transparent.

SIEHE AUCH

what [19]

iconv
Code konvertieren (international codeset conversion)

iconv liest Eingabezeichen aus einer Datei oder von der Standard-Eingabe, codiert die Eingabezeichen um und schreibt das Ergebnis auf die Standard-Ausgabe.

Die mit *iconv* möglichen Konvertierungen legen Sie durch Konvertierungstabellen fest, die sich unter */usr/lib/iconv* befinden.

So wandelt *iconv* z.B. die Zeichen aus dem Zeichensatz ISO 8859-1 in landesspezifische Zeichen aus den nationalen Varianten des Zeichensatzes ISO 646 (ASCII-Derivate) um, oder unterstützt die Umwandlung in umgekehrter Richtung (siehe *Beispiele*).

iconv␣-f␣ausgangscode␣-t␣zielcode[␣datei]

-f␣ausgangscode
-t␣zielcode
(f - from, t - to) *iconv* erwartet die Konvertierungstabelle in der Datei */usr/lib/iconv/ausgangscode.zielcode.t*. Zeichen, die im Ziel-Zeichensatz nicht existieren, werden in Unterstrich _ verwandelt.

datei
Name der Datei, deren Inhalt umcodiert werden soll.

datei nicht angegeben:
iconv liest von der Standard-Eingabe.

FEHLERMELDUNG

```
Not supported xx to yy
```

Die gewünschte Umwandlung des Zeichensatzes *xx* in den Ziel-Zeichensatz *yy* wird von *iconv* nicht unterstützt.

DATEIEN

/usr/lib/iconv
In diesem Dateiverzeichnis befinden sich die Standard-Konvertierungstabellen für die Umcodierung

/usr/lib/iconv/iconv_data
Hilfsdatei für *iconv*

/usr/lib/iconv/.t*
Konvertierungstabellen

BEISPIELE

1. Sie wollen alle Konvertierungstabellen auflisten:

```
$ ls /usr/lib/iconv
646da.8859.t  646fr.8859.t  8859.646da.t  8859.646fr.t
646de.8859.t  646it.8859.t  8859.646de.t  8859.646it.t
646en.8859.t  646sv.8859.t  8859.646en.t  8859.646sv.t
646es.8859.t  8859.646.t    8859.646es.t  iconv_data
```

2. Sie wollen den Inhalt der Datei *brief* konvertieren und das Ergebnis in die Datei *brief.conv* schreiben. Der Ausgangs-Zeichensatz ist die deutsche Variante des Zeichensatzes ISO 646, der Ziel-Zeichensatz soll der Zeichensatz ISO 8859-1 sein:

```
$ iconv -f 646de -t 8859 brief > brief.conv
```

SIEHE AUCH

iconv [7]

id
Benutzer- und Gruppennummer und zugehörige Kennungen ausgeben (user and group IDs)

Für den Prozeß, der *id* aufgerufen hat, schreibt *id* folgendes auf die Standard-Ausgabe:

- die Benutzernummer (UID)
- die Benutzerkennung
- die Gruppennummer (GID)
- den Gruppennamen.

Stimmen die effektiven und realen Nummern bzw. Kennungen nicht überein, so gibt *id* beide aus.

id[␣-a]

-a
(a - all) *id* gibt neben der Kennung und dem Namen des Benutzers alle Gruppen an, zu denen der aufrufende Prozeß gehört. Auch alle Gruppen, zu denen der aufrufende Benutzer gehört, werden angegeben.

Wechselwirkung mit su

id gibt Informationen über die *aktuelle* Benutzerkennung aus: Wenn Sie mit dem Kommando *su* die Benutzerkennung wechseln und in dieser neuen Kennung *id* eingeben, dann gibt *id* Informationen über die *neue* Benutzerkennung aus.
Die Kommandos *who* und *logname* verhalten sich anders: Sie liefern Informationen über die Login-Benutzerkennung.

DATEIEN

/etc/passwd
Kennwortdatei. Sie enthält die Benutzerkennungen sowie u.a. die zugehörigen Benutzer- und Gruppennummern.

/etc/group
Gruppendatei. Sie enthält die Gruppennamen sowie die zugehörigen Gruppennummern und Benutzerkennungen.

BEISPIEL

Wenn Sie Ihre aktuelle Benutzernummer, die zugehörige Benutzerkennung sowie Ihre aktuelle Gruppennummer und den Gruppennamen wissen wollen, dann geben Sie ein:

```
$ id
```

id gibt dann z.B. aus:

```
uid=115(doro)    gid=1(other)
```

SIEHE AUCH

logname, newgrp, su, who
getuid() [19]

ipcrm
Einrichtungen zur Interprozeß-Kommunikation entfernen (remove inter-process communication facilities)

ipcrm entfernt ein oder mehrere Semaphore, Nachrichten-Warteschlangen, gemeinsam benutzten Speicher (shared memory) oder Einrichtungen zur Interprozeß-Kommunikation (IPC-Einrichtungen). Diese können Sie entweder durch deren Bezeichner oder durch den Schlüssel, der bei der Erzeugung der jeweiligen IPC-Einrichtung verwandt wurde, angeben.
Bezeichner und Schlüssel von IPC-Einrichtungen erfahren Sie mit Hilfe des Kommandos *ipcs* (Ausgabe-Spalten *ID* bzw. *Key*). Detaillierte Informationen über die Vorgänge bei der Entfernung einer Nachrichten-Warteschlange, eines gemeinsam benutzten Speichersegments oder eines Semaphors erhalten Sie in der Beschreibung von *msgctl*, *shmctl* bzw. *semctl* im *Referenzhandbuch für Programmierer* [19].

ipcrm[␣option]...

option

-q␣msgqid
(q - queue) Die Nachrichten-Warteschlange mit der Kennzahl *msgqid* wird aus dem System entfernt und die damit verbundenen Datenstrukturen werden zerstört.

msgqid
Kennzahl der Nachrichten-Warteschlange, die entfernt werden soll. Diese Kennzahl gibt das Kommando *ipcs* in der Spalte *ID* aus.

-Q␣msgkey
(Q - queue) Die Nachrichten-Warteschlange mit dem Schlüssel *msgkey* wird aus dem System entfernt und die damit verbundenen Datenstrukturen werden zerstört.

msgkey
Schlüssel der Nachrichten-Warteschlange, die entfernt werden soll. Diesen Schlüssel gibt das Kommando *ipcs* in der Spalte *KEY* aus.

-s_semid
(s - semaphor) Das Semaphor mit der Kennzahl *semid* wird aus dem System entfernt, und die damit verbundenen Datenstrukturen werden zerstört.

semid
Kennzahl des Semaphors, das entfernt werden soll. Diese Kennzahl gibt das Kommando *ipcs* in der Spalte *ID* aus.

-S_semkey
(S - semaphor) Das Semaphor mit dem Schlüssel *semkey* wird aus dem System entfernt, und die damit verbundenen Datenstrukturen werden zerstört.

semkey
Schlüssel, mit dem das Semaphor erzeugt wurde, das Sie entfernen möchten. Diesen Schlüssel gibt das Kommando *ipcs* in der Spalte *KEY* aus.

-m_shmid
(m - memory) Der gemeinsam benutzte Speicher (shared memory) mit der Kennzahl *shmid* wird aus dem System entfernt, und die damit verbundenen Datenstrukturen werden zerstört.

shmid
Kennzahl des gemeinsam benutzten Speichers, der entfernt werden soll. Diese Kennzahl gibt das Kommando *ipcs* in der Spalte *ID* aus.

-M_shmkey
(M - memory) Der gemeinsam benutzte Speicher (shared memory) mit dem Schlüssel *shmkey* wird aus dem System entfernt, und die damit verbundenen Datenstrukturen werden zerstört.

shmkey
Schlüssel des gemeinsam benutzten Speichers, der entfernt werden soll. Diesen Schlüssel gibt das Kommando *ipcs* in der Spalte *KEY* aus.

BEISPIEL

Zuerst lassen Sie mit *ipcs* einen Bericht über den Zustand von Einrichtungen zur Interprozeß-Kommunikation ausgeben. Danach entfernen Sie die Nachrichten-Warteschlange mit der Kennzahl 40 aus dem System:

```
$ ipcs
Time is 248bc52b
IPC status from /dev/kmem as of Tue Jun  6 13:35:55 1989
T     ID     KEY        MODE       OWNER    GROUP
Message Queues:
q     40 0x0000004b -Rrw-rw-rw-  michael    qm234
Semaphores:
$ ipcrm -q 40
```

SIEHE AUCH

ipcs

msgctl(), *msgget()*, *msgop()*, *semctl()*, *semget()*, *semop()*, *shmctl()*, *shmget()*, *shmop()* [19]

ipcs
Zustand von Interprozeß-Kommunikationseinrichtungen ausgeben (inter-process communication status)

ipcs gibt Informationen über aktive Einrichtungen zur Interprozeß-Kommunikation (IPC-Einrichtungen) aus. Durch die Angabe von Optionen können Sie steuern,
- über welche Art von IPC-Einrichtungen Informationen ausgegeben werden
- welche Informationen ausgegeben werden

Da sich die Zustände der IPC-Einrichtungen ändern können, während *ipcs* läuft, ist die Information, die *ipcs* liefert, nur zum Zeitpunkt der Abfrage aktuell.

ipcs[_option]...

Keine Option angegeben
ipcs gibt in Kurzform Informationen aus über
- Nachrichten-Warteschlangen (message queues)
- gemeinsam benutzten Speicher (shared memory)
- Semaphore

die derzeit im System aktiv sind. Die Bedeutung der Ausgabe-Spalten ist im Abschnitt *Ausgabe* beschrieben.

option

Es gibt Optionen, um die Art der IPC-Einrichtung festzulegen, und Optionen, um die Art der Informationen festzulegen.

Art der IPC-Einrichtung festlegen

-q
(q - message queues) Informationen über aktive Nachrichten-Warteschlangen ausgeben.

-m
(m - memory) Informationen über aktiven gemeinsam benutzten Speicher (shared memory) ausgeben.

-s
(s - semaphores) Informationen über aktive Semaphore ausgeben.

Ist eine der Optionen *-q*, *-m* oder *-s* gesetzt, so werden nur Informationen über die entsprechenden IPC-Einrichtung ausgegeben. Die Kombination der Optionen ist möglich. Ist keine der Optionen gesetzt, so werden Informationen über alle Arten von IPC-Einrichtungen ausgegeben.

Art der Informationen festlegen

Im folgenden sind die Optionen beschrieben, die festlegen, welche Informationen für die durch die Optionen *-q*, *-m* und *-s* festgelegten IPC-Einrichtungen ausgegeben werden. Die Bedeutung der Ausgabe-Spalten ist im Abschnitt *Ausgabe* beschrieben.

-a

(a - all) Alle Optionen, die die Art der ausgegebenen Informationen bestimmen, sind gesetzt. Dies ist die Kurzform für *-b*, *-c*, *-o*, *-p* und *-t*.

-b

(b - biggest) Es wird die maximal zulässige Größe der jeweiligen IPC-Einrichtung ausgegeben:

Für Nachrichten-Warteschlangen die maximale Anzahl von Bytes in einer Nachricht, die in diese Nachrichten-Warteschlange eingereiht werden soll.

Für gemeinsam benutzten Speicher (shared memory) die Größe der Speichersegmente.

Für Semaphore die Zahl der Semaphore in jeder Semaphor-Gruppe.

-c

(c - creator) Die Benutzerkennung und der Gruppen-Name des Erzeugers der IPC-Einrichtung werden ausgegeben.

-o

(o - outstanding) Informationen über noch zu bearbeitende IPC-Einrichtungen werden ausgegeben:
- Zahl der Nachrichten in Nachrichten-Warteschlangen
- Gesamtzahl der Bytes von Nachrichten in Nachrichten-Warteschlangen
- Anzahl von Prozessen, die einem gemeinsam benutzten Speichersegment zugeordnet sind.

-p

(p - process) Es werden Informationen über Prozeßnummern ausgegeben:
- Prozeßnummer des letzten Prozesses, der eine Nachricht gesandt hat
- Prozeßnummer des letzten Prozesses, der eine Nachricht empfangen hat
- Prozeßnummer des Prozesses, der ein gemeinsam benutztes Speichersegment erzeugt hat
- Prozeßnummer des letzten Prozesses, der ein gemeinsam benutztes Speichersegment benutzt hat (attach, detach).

-t

(t - time) Zeit-Informationen ausgeben. Es wird der Zeitpunkt der letzten Änderung der Zugriffsrechte für alle IPC-Einrichtungen ausgegeben.

Für Nachrichten-Warteschlangen wird der Zeitpunkt ausgegeben, zu dem zuletzt die Systemaufrufe *msgrcv()* (Nachricht aus einer Nachrichten-Warteschlange lesen) oder *msgsnd()* (Nachricht in eine Nachrichten-Warteschlange eintragen) angewandt wurden.

Für gemeinsam benutzten Speicher wird der Zeitpunkt ausgegeben, zu dem zuletzt die Systemaufrufe *shmat()* oder *shmdt()* angewandt wurden.

Für Semaphore wird der Zeitpunkt ausgegeben, zu dem zuletzt der Systemaufruf *semop()* auf ein Semaphor angewandt wurde.

Weitere mögliche Argumente

-C␣coredatei

Die Datei *coredatei* wird anstelle von */dev/kmem* verwendet. Diese Option kann nur von *root* bzw. einem Benutzer mit Zugriffsberechtigung für */dev/kmem* benutzt werden.

-N␣namensliste

Die Datei *namensliste* wird anstelle von */stand/unix* verwendet. Diese Option kann nur von *root* bzw. einem Benutzer mit Zugriffsberechtigung für */dev/kmem* benutzt werden.

-X

(X - XENIX) Zusätzlich zu den Standard-Informationen der Interprozeß-Kommunikationseinrichtungen werden noch Informationen über die XENIX-Interprozeß-Kommunikationseinrichtungen angegeben. Diese Informationen beschreiben eine zweite Menge von Semaphoren und gemeinsam benutztem Speicher.

Die Option *-p* gibt jedoch keine Prozeßinformationen über den gemeinsam benutzten Speicher in XENIX aus. Die *-t* Option gibt keine Zeitinformationen über XENIX-Semaphore und gemeinsam benutzten Speicher aus.

Wenn die Option *-C* oder *-N* gesetzt ist, wird die reale und effektive Benutzer- und Gruppenkennung auf die reale Benutzer- und Gruppenkennung des Benutzers gesetzt, der *ipcs* aufgerufen hat.

AUSGABE

Im folgenden sind die Spalten-Überschriften und die Bedeutung der Spalten in der Ausgabe eines *ipcs*-Aufrufs aufgeführt. Die in runden Klammern (...) eingeschlossenen Buchstaben stehen dabei für die Optionen, die die Ausgabe der entsprechenden Spalten-Überschrift zur Folge haben. *alle* bedeutet, daß die betreffende Spalten-Überschrift in jedem Fall ausgegeben wird.

CBYTES (a,o)
Die Anzahl der Bytes in den Nachrichten, die in der zugehörigen Nachrichten-Warteschlange auf ihre Bearbeitung warten.

CGROUP (a,c)
Der Name der Gruppe zu der der Erzeuger der IPC-Einrichtung gehört.

CREATOR (a,c)
Die Benutzerkennung des Erzeugers der IPC-Einrichtung.

CTIME (a,t)
Der Zeitpunkt, zu dem der Eintrag für die zugehörige IPC-Einrichtung erstellt oder geändert wurde.

GROUP (alle)
Der Name der Gruppe zu der der Eigentümer der IPC-Einrichtung gehört.

ID (alle)
Die Kennzahl der IPC-Einrichtung.

KEY (alle)
Der Schlüssel, der beim Erzeugen der IPC-Einrichtung mit *msgget()* (Nachrichten-Warteschlange einrichten) oder *semget()* (Semaphor-Menge anlegen) als Argument benutzt wurde.

LRPID (a,p)
Die Prozeß-Nummer des Prozesses, der zuletzt eine Nachricht aus der zugehörigen Nachrichten-Warteschlange empfangen hat.

LSPID (a,p)
Die Prozeß-Nummer des Prozesses, der zuletzt eine Nachricht an die zugehörige Nachrichten-Warteschlange geschickt hat.

MODE (alle)
Die Zugriffsmodi und Status-Anzeigen für die IPC-Einrichtung. Der Modus besteht aus 11 Zeichen, die folgende Bedeutung haben:

Das erste Zeichen kann sein:

S wenn ein Prozeß auf ein *msgsnd()* wartet
D wenn das betreffende gemeinsam benutzte Speichersegment freigegeben wurde. Es verschwindet, wenn es vom letzten Prozeß, der dem Segment zugeordnet ist, freigegeben wird.
– wenn kein Prozeß auf ein *msgsnd()* wartet.

Das zweite Zeichen kann sein:

R wenn ein Prozeß auf ein *msgrcv()* wartet
C Das betreffende gemeinsam benutzte Speichersegment wird bereinigt, wenn der erste Zugriff (attach) ausgeführt wird.
– wenn kein Prozeß auf ein *msgrcv()* wartet.

Die folgenden 9 Zeichen werden in drei Gruppen zu je drei Bits unterteilt:

Die erste Gruppe steht für die Zugriffsrechte des Eigentümers der IPC-Einrichtung.

Die zweite steht für die Zugriffsrechte der Benutzer, die zur selben Benutzer-Gruppe wie der Eigentümer der IPC-Einrichtung gehören.

Die dritte Gruppe steht für die Zugriffsrechte aller anderen Benutzer.

In jeder Gruppe steht das erste Zeichen für das Leserecht, das zweite Zeichen für das Schreibrecht bzw. für das Recht, den Eintrag für die IPC-Einrichtung zu ändern. Das dritte Zeichen in jeder Gruppe wird zur Zeit noch nicht benutzt.

Die Zugriffsrechte werden folgendermaßen angegeben:

r Leserecht ist vorhanden
w Schreibrecht ist vorhanden
a Recht, den Eintrag für die IPC-Einrichtung zu ändern, ist vorhanden.
- Das betreffende Recht ist nicht vorhanden.

NSEMS (a,b)
Die Zahl der Semaphore in der zugehörigen Semaphor-Menge.

OTIME (a,t)
Der Zeitpunkt, zu dem die letzte Semaphor-Operation auf die zugehörige Semaphor-Menge angewandt wurde.

OWNER (alle)
Die Benutzerkennung des Eigentümers der IPC-Einrichtung.

QBYTES (a,b)
Die Anzahl der Bytes, die diejenigen Nachrichten maximal enthalten dürfen, die in der Nachrichten-Warteschlange auf ihre Bearbeitung warten.

QNUM (a,o)
Die Anzahl der Nachrichten, die momentan in der zugehörigen Nachrichten-Warteschlange enthalten sind.

RTIME (a,t)
Der Zeitpunkt, zu dem die letzte Nachricht aus der zugehörigen Nachrichten-Warteschlange empfangen wurde.

STIME (a,t)
Der Zeitpunkt, zu dem die letzte Nachricht an die zugehörige Nachrichten-Warteschlange geschickt wurde.

NATTCH (a,o)
Die Anzahl der Prozesse, die dem betreffenden gemeinsam benutzten Speichersegment zugeordnet sind.

SEGSZ (a,b)
Die Größe des betreffenden gemeinsam benutzten Speichersegments.

CPID (a,p)
Die Nummer des Prozesses, der das gemeinsam benutzte Speichersegment angelegt hat.

LPID (a,p)
Die Nummer des letzten Prozesses, der ein gemeinsam benutztes Speichersegment benutzt hat (attach, detach).

ATIME (a,t)
Der Zeitpunkt, zu dem der letzte Zugriff (attach) auf das betreffende gemeinsam benutzte Speichersegment beendet war.

DTIME (a,t)
Der Zeitpunkt, zu dem die letzte Freigabe (detach) eines gemeinsam benutzten Speichersegment beendet war.

T (alle)
Typ der IPC-Einrichtung:

q	steht für Nachrichten-Warteschlange
m	steht für gemeinsam benutzten Speicher (shared memory)
s	steht für Semaphor.

DATEIEN

/stand/unix
System-Namensliste

/dev/kmem
Datei, die ein Abbild des virtuellen Speichers des Betriebssystemkerns Rechners darstellt.

/etc/passwd
Die Datei */etc/passwd* enthält alle eingerichteten Benutzerkennungen.

/etc/group
Die Datei */etc/group* enthält alle eingerichteten Benutzergruppen.

BEISPIEL

Das Programm *server* richtet eine Nachrichten-Warteschlange ein. Sie starten dieses Programm im Hintergrund und fragen dann mit *ipcs* den Zustand von Einrichtungen zur Interprozeß-Kommunikation ab:

```
$ server &
$ ipcs
IPC status from /dev/kmem as of Tue Jun  6 13:35:55 1989
T     ID     KEY        MODE       OWNER    GROUP
Message Queues:
q     40 0x0000004b -Rrw-rw-rw-  michael    qm234
Semaphores:
```

SIEHE AUCH

ipcrm

msgop(), msgctl(), msgget() semctl(), semget(), semop(), shmop(), shmctl() [19]

ismpx
Zustand eines Bildschirms mit Fensterdarstellung abfragen (is multiplexed)

ismpx gibt an, ob seine Standard-Eingabe mit einem *xt*-Kanal verbunden ist, d.h. ob es unter *layers* läuft. Es kann in Shell-Prozeduren verwendet werden, die Programme über eine Leitung an einen Bildschirm mit Fensterdarstellung laden.

ismpx[-s]

Keine Option angegeben
ismpx gibt *yes* aus und liefert den Wert 0 zurück, falls es unter *layers* aufgerufen wurde. Anderenfalls wird *no* ausgegeben und der Wert 1 zurückgeliefert.

-s
Es wird nichts ausgegeben, sondern nur der jeweilige Ende-Status zurückgegeben.

BEISPIEL

Es soll getestet werden, ob der Aufruf unter *layers* erfolgte und gegebenenfalls die Fenstergröße ausgegeben werden:

```
if ismpx -s
then
    jwin
fi
```

SIEHE AUCH

layers, *jterm*, *jwin*
xt [7], [18]

join
Zwei Dateien nach Vergleichsfeldern verbinden

join vergleicht zwei Dateien nach Vergleichsfeldern und verbindet alle Zeilenpaare, deren Vergleichsfeld identisch ist. Das Ergebnis gibt *join* auf die Standard-Ausgabe aus.

Beim *join*-Aufruf legen Sie für jede der beiden Dateien fest, welches Feld das Vergleichsfeld sein soll. Ein Feld wird von zwei Feldtrennzeichen begrenzt. *join* vergleicht jede Zeile der ersten Datei mit den Zeilen der zweiten Datei. Für jedes Zeilenpaar mit identischem Vergleichsfeld gibt *join* auf die Standard-Ausgabe eine Ausgabezeile aus, die sich aus bestimmten Feldern beider Zeilen zusammensetzt.

Vor dem Aufruf beachten

Jede Eingabedatei muß in ihrem Vergleichsfeld gemäß der ASCII-Reihenfolge sortiert sein (siehe *sort*). Bei Standard-Feldtrennung (*join* ohne Option *-t*) dürfen beim Sortieren führende Feldtrennzeichen nicht berücksichtigt werden (siehe *sort*, Option *-b*). Geben Sie dagegen *join* mit Option *-t* an, müssen Sie führende Trennzeichen beim Sortieren berücksichtigen (siehe *sort* ohne Option *-b*).

join[_option]..._datei1_datei2

Keine Option angegeben
: Vergleichsfeld für beide Dateien ist das erste Feld. Feldtrennzeichen für die Eingabezeilen sind Leerzeichen, Tabulatorzeichen und Neue-Zeile-Zeichen. Aufeinanderfolgende Feldtrennzeichen werden als ein einziges Feldtrennzeichen interpretiert; führende Feldtrennzeichen werden ignoriert.
Für jedes Zeilenpaar mit identischem Vergleichsfeld gibt *join* auf die Standard-Ausgabe eine Ausgabezeile aus. Diese Ausgabezeile enthält in dieser Reihenfolge:

- das gemeinsame Vergleichsfeld
- den Rest der Eingabezeile aus der ersten Datei
- den Rest der Eingabezeile aus der zweiten Datei.

Die Felder der Ausgabezeilen sind durch ein Leerzeichen voneinander getrennt.

option

-an
: (a - additional output) *join* gibt zusätzlich zur normalen Ausgabe alle Zeilen der *n*-ten Eingabedatei aus, deren Vergleichsfeld mit keinem Vergleichsfeld der anderen Datei übereinstimmt.
Für *n* können Sie 1 oder 2 angeben. Soll die Ausgabe für beide Vergleichsdateien erfolgen, dann geben Sie *-a1 -a2* an.

-e␣zeichenkette
(e - empty output fields) *join* ersetzt leere Ausgabefelder durch die angegebene Zeichenkette.

-j[n]␣m
Als Vergleichsfeld für die *n*-te Datei wird das *m*-te Feld festgelegt. Für *n* können Sie 1 oder 2 angeben, für *m* eine ganze Zahl größer gleich 1.
Wenn Sie für die andere Datei keine Option *-j* angeben, dann ist das Vergleichsfeld für diese andere Datei das 1. Feld.

n nicht angegeben:
Vergleichsfeld für beide Dateien ist das *m*-te Feld.

-j nicht angegeben:
Vergleichsfeld für beide Dateien ist das 1. Feld.

-o␣liste
(o - output format) *join* ändert das Format der Ausgabezeilen: Die Ausgabezeilen enthalten dann der Reihe nach die in *liste* angegebenen Felder. Das gemeinsame Vergleichsfeld wird nur dann ausgegeben, wenn Sie es ausdrücklich in *liste* angegeben haben.

Für *liste* geben Sie eine Liste an, die aus Elementen der Form *n.m* besteht, wobei *n* gleich 1 oder 2 und *m* größer gleich 1 ist. Ein Element *n.m* steht für das *m*-te Feld der *n*-ten Datei. Die Elemente trennen Sie durch Leer- oder Tabulatorzeichen.

-tc
Das Zeichen *c* wird als Feldtrennzeichen sowohl für die Eingabe- als auch für die Ausgabezeilen definiert. Jedes Vorkommen von *c* wird als Feldtrennzeichen interpretiert, d.h.

- zwei aufeinanderfolgende *c* kennzeichnen ein leeres Feld und
- ein führendes *c* kennzeichnet ein leeres erstes Feld.

Zusätzlich ist das Neue-Zeile-Zeichen Feldtrennzeichen für die Eingabezeilen.
Die Standard-Feldtrennzeichen Leer- und Tabulatorzeichen werden nur dann als Feldtrennzeichen interpretiert, wenn Sie diese Zeichen für *c* angeben.

datei1 datei2
Namen der beiden Dateien, die *join* nach Vergleichsfeldern verbinden soll.
Wenn Sie für *datei1* einen Bindestrich - angeben, liest *join* von der Standard-Eingabe.

Vorsicht

Wenn die Dateien nicht nach ihrem Vergleichsfeld sortiert sind, dann bearbeitet *join* nicht alle Zeilen!

Wenn Sie für *datei1* einen numerischen Dateinamen (z.B. 1.2) angeben und diesem Dateinamen die Option *-o* unmittelbar voranstellen, dann kann das zu Problemen führen. Wenn Sie also einen numerischen Dateinamen angeben wollen, dann geben Sie ihn mit Schrägstrich (z.B. ./1.2) an.

BEISPIELE

1. In der Datei *ort* ist einem Namen ein Ort zugeordnet, in der Datei *betrag* sind denselben Namen je ein Betrag und ein Datum zugeordnet. Beide Dateien sind nach den Namen sortiert. *join* soll beide Dateien nach den Namen verbinden.

   Inhalt der Datei *ort*:

```
Albert München
Hugo Stuttgart
Ilse Hamburg
```

   Inhalt der Datei *betrag*:

```
Albert   287.56  20.03.88
Hugo      23.15  25.06.87
Hugo     167.87  16.12.87
Ilse    1212.12  12.12.88
Ilse       1.98  01.01.88
```

   Verbinden der beiden Dateien nach dem ersten Vergleichsfeld:

```
$ join ort betrag
Albert Muenchen 287.56 20.03.88
Hugo Stuttgart 23.15 25.06.87
Hugo Stuttgart 167.87 16.12.87
Ilse Hamburg 1212.12 12.12.88
Ilse Hamburg 1.98 01.01.88
```

   Verbinden der beiden Dateien und spaltenweise formatieren mit *awk*:

```
$ join ort betrag | awk '{printf("%-10s %-15s %-10s %-10s\n",$1,$2,$3,$4)}'
Albert     Muenchen        287.56     20.03.88
Hugo       Stuttgart        23.15     25.06.87
Hugo       Stuttgart       167.87     16.12.87
Ilse       Hamburg        1212.12     12.12.88
Ilse       Hamburg           1.98     01.01.88
```

2. In der Datei *stadt* ist einer Stadt ein Name zugeordnet, in der Datei *betrag* (siehe *Beispiel 1*) ist einem Namen ein Betrag und ein Datum zugeordnet. Die Datei *stadt* ist nach den Städten, die Datei *betrag* nach den Namen sortiert. *join* soll beide Dateien nach den Namen verbinden.

   Inhalt der Datei *stadt*:

```
Augsburg      Egon
Hamburg       Ilse
Muenchen      Albert
Muenchen      Franz
Stuttgart     Hugo
```

   Vergleichsfeld für die Datei *stadt* ist das 2. Feld, für die Datei *betrag* das 1. Feld. Bevor Sie die Dateien verbinden, müssen Sie die Datei *stadt* nach dem 2. Feld sortieren. Formatieren Sie anschließend wieder spaltenweise mit *awk*:

```
$ sort -b +1 stadt | join -j1 2 - betrag | \
> awk '{printf("%-10s %-15s %-10s %-10s\n",$1,$2,$3,$4)}'
Albert     Muenchen        287.56     20.03.88
Hugo       Stuttgart       23.15      25.06.87
Hugo       Stuttgart       167.87     16.12.87
Ilse       Hamburg         1212.12    12.12.88
Ilse       Hamburg         1.98       01.01.88
```

SIEHE AUCH

awk, *comm*, *sort*, *uniq*

jsh
Bourne-Shell mit Auftragssteuerung (job control)

jsh ist eine Bourne-Shell (*sh*) mit Auftragssteuerung (job control). Mit der Auftragssteuerung können Sie zusätzlich zur normalen Funktionalität der Bourne-Shell:

- Aufträge stoppen,
- Aufträge in den Hintergrund schicken,
- Aufträge aus dem Hintergrund in den Vordergrund holen und
- über Auftragsnummern einfacher auf Aufträge zugreifen.

Die Auftragssteuerung werden Sie im Regelfall nur bei einer Shell ausnutzen, mit der Sie interaktiv arbeiten, also einer Login-Shell oder einer Subshell. Eine Shell, die Shell-Prozeduren ausführt, kann diese Erweiterung nicht ausnutzen.

Die Auftragsteuerung ist realisiert durch

- die Einführung von Aufträgen (jobs) mit verschiedenen Zuständen,
- neue eingebaute Shell-Kommandos
- die Erweiterung bestehender eingebauter Shell-Kommandos.

jsh [_option...][_argument]...

Die Beschreibung von *option* und *argument* finden Sie bei der Beschreibung der Bourne-Shell *sh*: In diesem Bereich gibt es keine Unterschiede zwischen den beiden Shells.

ARBEITSWEISE

Für *jsh* ist jedes Kommando oder jede Pipeline, die von Ihnen an der Datensichtstation eingegeben wurde, ein Auftrag. Jeder exitierende Auftrag nimmt einen der drei möglichen Zustände ein:

Vordergrund
: Dieser Auftrag hat Lese- und Schreibzugriff auf das kontrollierende Terminal.

Hintergrund
: Diesem Auftrag wurde der Lesezugriff auf das kontrollierende Terminal entzogen, er besitzt aber möglicherweise Schreibzugriff (siehe *stty*).

gestoppt
: Dieser Auftrag wurde angehalten - normalerweise durch das Signal *SIGTSTP* (siehe *signal()* [19]).

Jedem gestarteten Auftrag weist *jsh* eine positive ganze Zahl zu, die sogenannte Auftragsnummer. Diese Nummer wird von *jsh* als Bezeichner zur Identifizierung eines Auftrags weiter verfolgt. Zusätzlich kennt die Shell noch den *aktuellen* und den *vorherigen* Auftrag. Der aktuelle Auftrag ist der zuletzt aufgerufene oder wieder aufgerufene Auftrag. Der vorherige Auftrag ist der erste nicht-aktuelle Auftrag.

Die Syntax eines Auftragsbezeichners sieht wie folgt aus:

%auftragsbezeichner

auftragsbezeichner kann sein:

% oder **+**
der aktuelle Auftrag

-
der vorherige Auftrag

?zeichenkette
der Auftrag, der in der Kommandozeile eindeutig *zeichenkette* enthält

nummer
der Auftrag mit der Auftragsnummer *nummer*

präfix
der Auftrag, dessen Kommando mit der Zeichenkette *präfix* beginnt. *präfix* darf entwertete Leerzeichen enthalten.
Beispiel: Das Hintergrund-Kommando *ls -l ..* können Sie mit *%ls* ansprechen.

Einen Auftrag können Sie durch Drücken der Tastenkombination CTRL Z anhalten. Diese sorgt dafür, daß ein *STOP*-Signal an den Auftrag geschickt wird. *jsh* zeigt dann an, daß der Auftrag gestoppt wurde und gibt ein Bereitzeichen aus. Die Tastenkombination CTRL Z bewirkt eine sofortige Reaktion (vergleichbar der Unterbrechungstaste DEL).

Vorsicht

Verwenden Sie *jsh* zusammen mit *shl*, dann müssen Sie das *switch*-Zeichen (siehe *stty*) neu definieren, damit das *suspend*-Zeichen nicht deaktiviert wird.

KOMMANDOS

Die folgenden eingebauten Shell-Kommandos können Sie nach dem Bereitzeichen PS1 (Standard: $_) eingeben:

bg[_%auftragsbezeichner]...
Die Ausführung eines gestoppten Auftrags wird im Hintergrund fortgesetzt. Der oder die gestoppten Aufträge werden durch *auftragsbezeichner* angegeben. Bei fehlendem *auftragsbezeichner* wird der aktuelle Auftrag fortgesetzt.

fg[␣%auftragsbezeichner]...
Es wird entweder die Ausführung eines gestoppten Auftrags im Vordergrund fortgesetzt oder ein Auftrag, der im Hintergrund ausgeführt wird, in den Vordergrund geholt. Der oder die Aufträge werden durch *auftragsbezeichner* angegeben. Ist *auftragsbezeichner* nicht angegeben, so wird der aktuelle Auftrag verwendet.

jobs[-l|-p][␣%auftragsbezeichner]... Format 1
jobs␣**-x**␣kommando[␣argument]... Format 2

Format 1: Information über Aufträge ausgeben

Information über Aufträge, die im Hintergrund ausgeführt werden oder gestoppt sind, wird auf die Standard-Ausgabe geschrieben. Fehlt *auftragsbezeichner*, dann wird die Information über alle Aufträge auf die Standard-Ausgabe geschrieben.

Die folgenden Optionen verändern die Ausgabe von *jobs*:

-l
Die Prozeßgruppennummer und das aktuelle Verzeichnis der Aufträge werden auf die Standard-Ausgabe geschrieben.

-p
Nur die Prozeßgruppennummer der Aufträge wird auf die Standard-Ausgabe geschrieben.

Format 2: Kommando auf Aufträge ausführen

Dieses Kommando ersetzt jeden Auftragsbezeichner, der in *kommando* und *argument* gefunden wird, durch die entsprechende Prozeßgruppennummer und führt dann das so erhaltene Kommando mit den Argumenten aus.

kill[␣signal]␣%auftragsbezeichner
Dies ist die in die Shell *sh* eingebaute Version des *kill*-Kommandos. Sie bietet, im einzigen Unterschied zum Kommando *kill*, die Möglichkeit der Angabe der Prozeßnummer durch einen *auftragsbezeichner*.

stop␣%auftragsbezeichner...
Die Ausführung des Hintergrundauftrags *auftragsbezeichner* wird gestoppt.

suspend
Die Ausführung der aktuellen *jsh* wird gestoppt. Dies trifft jedoch nicht zu, wenn sie eine Login-Shell ist.

wait[␣%auftragsbezeichner]...
Diese Version des eingebauten Shell-Kommandos *wait* versteht *auftragsbezeichner* als Argument. Bei fehlendem *auftragsbezeichner* verhält sich *wait* wie die in *sh* eingebaute Version, sonst wird auf die Beendigung des oder der Hintergrundaufträge gewartet.

BEISPIEL

Im folgenden finden Sie ein kurzes Beispiel für den Umgang mit Aufträgen in der Shell *jsh*.

Sie rufen die Shell auf mit der Eingabe von:

```
$ jsh
```

Ändern Sie zur besseren Orientierung den Prompt:

```
$ PS1="jsh> "
```

Sie Suchen nach einer Datei namens *test*. Starten Sie dazu das Kommando *find* im Hintergrund:

```
jsh> find / -name test -print &
```

Die Shell gibt als Ergebnis die Jobnummer 1 und die Prozessnummer 28071 aus:

```
[1] 28071
```

Sie lassen sich eine Liste mit allen aktuellen Aufträgen ausgeben:

```
jsh> jobs
[1] + Running                    find / -name test -print
```

Sie beenden Auftrag Nummer 1 (find-Kommando):

```
jsh> stop %1
```

Sie lassen sich eine Liste mit allen aktuellen Aufträgen ausgeben:

```
jsh> jobs
[1] + Stopped (signal)           find / -name test -print
```

Sie lassen den aktuellen Auftrag (find-Kommando) im Hintergrund weiterlaufen:

```
jsh> bg %%
```

Die Shell gibt das reaktivierte Kommando aus:

```
[1] find / -name test -print &
```

Sie brechen den Auftrag *find* ab:

```
jsh> kill %find
```

Sie lassen sich eine Liste aller aktuellen Aufträge ausgeben.

```
jsh> jobs
[1] + Terminated                 find / -name test -print
```

Sie verlassen die jsh-Shell wieder mit

```
jsh> exit
```

und erhalten darauf das Bereitzeichen der Bourne-Shell:

```
$
```

SIEHE AUCH

sh

jterm
Shell-Fenster auf einem Bildschirm mit Fensterdarstellung zurücksetzen

Mit *jterm* wird ein Shell-Fenster auf einem Bildschirm mit Fensterdarstellung zurückgesetzt. Dies ist nötig, wenn ein Bildschirmprogramm geladen wurde, das die Bildschirmattribute für dieses Shell-Fenster verändert hat. *jterm* arbeitet nur unter *layers*. Normalerweise wird *jterm* verwendet, um das Standard-Bildschirm-Emulationsprogramm neu zu starten, nachdem mit einem Anwendungsprogramm ein anderes benutzt wurde.

jterm

SIEHE AUCH

layers, *ismpx*, *jwin*

jwin
Größe eines Shell-Fensters abfragen

jwin arbeitet nur unter *layers* und gibt die Größe des Shell-Fensters aus, das mit der Standard-Eingabe verbunden ist. Breite und Höhe des Fensters werden in byte ausgegeben (Anzahl der Zeichen bzw. Anzahl der Zeilen). Für pixelorientierte Geräte werden zusätzlich Breite und Höhe des Fensters in Bildpunkten ausgegeben.

jwin

FEHLERMELDUNGEN

```
jwin: runs only under layers
```

layers ist nicht aktiv.

```
jwin: not mpx
```

layers ist nicht aktiv, die Leitung wird nicht gemultiplext.

BEISPIEL

```
$ jwin
bytes:  86 25
bits:   780 406
```

SIEHE AUCH

layers, *ismpx*

keyload
Tastaturtabellen laden

Mit dem Kommando *keyload* können Sie Ihre Tastatur auf Landessprachen abgestimmt belegen, indem Sie die entsprechenden Tastaturtabellen laden.
Rufen Sie *keyload* an dem Bildschirm auf, dessen Tastaturtabelle geladen werden soll, dann stellt *keyload* die Leitungsparameter selbst entsprechend ein.
Da *keyload* in der Regel von *getty* aus gestartet wird, wird im Programm selbst die Schnittstelle der Datensichtstation nicht immer passend eingestellt. Bei direkter Verwendung des Kommandos *keyload* muß der Anwender für die richtige Einstellung sorgen (siehe *Beispiel*).
Es ist zu empfehlen, vor der Benutzung von *keyload* die entsprechende Tastatur über die Tastatur-Umschalt-Strings einzustellen. Dabei geben Sie ein:

ESC [7u	für deutsche Tastatur
ESC [6u	für andere Tastaturen

keyload lädt die deadkey- und composekey-Tabellen. Diese müssen sich unter dem gleichen Namen wie die Tastaturtabellen in folgenden Dateiverzeichnissen befinden: */etc/ckeytables* (composekeys), */etc/dkeytables* (deadkeys).
keyload interpretiert genau jene Bildschirme als 8-Bit-Bildschirme, die in */etc/inittab* mit *N* konfiguriert sind. Alle anderen Terminals werden als 7-Bit-Terminals interpretiert. Beachten Sie, daß *keyload* für Pseudo-Bildschirme nur dann verwendet werden sollte, wenn die Verbindung 8-bit-fähig ist.

Vorsicht

keyload sollte nur von erfahrenen Anwendern benutzt werden. Die Funktionalität von *keyload* ist auch im Bediensystem enthalten und kann dort komfortabler genutzt werden.

keyload[_-t_typ][_tty_nr]_tabelle

-t_typ

Mit dieser Option geben Sie den Typ des zur Tastatur gehörenden Bildschirms an. *typ* kann sein:

new	alle Vorgänger des Bildschirms 97801-480 im 7-bit-Modus
7new	alle zum Bildschirm 97801-480 kompatiblen Bildschirme im 7-bit-Modus
8bin	alle zum Bildschirm 97801-480 kompatiblen Bildschirme im 8-bit-Modus
bin	obsolet; wird nur aus Kompatibilitätsgründen weitergeführt

tty_nr
Für *tty_nr* geben Sie die Gerätedatei des zur Tastatur gehörenden Bildschirms an. Es genügt die Angabe des relativen Pfadnamens im Verzeichnis */dev*.

tty_nr nicht angegeben:
Es wird die Datensichtstation verwendet, die mit der Standard-Ausgabe verbunden ist.

tabelle
Für *tabelle* geben Sie den Namen der Tastaturtabelle an, die geladen werden soll, z.B. *franz* für Französisch oder *ital* für Italienisch. Die möglichen Namen erfahren Sie anhand der Dateinamen im Dateiverzeichnis */etc/keytables*.

Die folgenden Tastaturbelegungstabellen stehen Ihnen für die entsprechenden Sprachen zu Verfügung:
englisch, dänisch, deutsch, französisch, italienisch, norwegisch, schwedisch, deutschsprachige Schweiz, französischsprachige Schweiz, spanisch.

DATEIEN

/etc/ckeytables
Dateiverzeichnis, das die composekey-Tabellen in den verschiedenen Landessprachen enthält.

/etc/dkeytables
Dateiverzeichnis, das die deadkey-Tabellen in den verschiedenen Landessprachen enthält.

/etc/keytables
Dateiverzeichnis, das die Tastaturtabellen in den verschiedenen Landessprachen enthält.

BEISPIEL

Sie wollen für den Bildschirm *tty003* die französischen Tastaturtabellen laden. Dabei wird angenommen, daß Sie an einem anderen Bildschirm arbeiten. Hierzu geben Sie folgendes ein:

```
$ cat > /dev/tty003
  ESC [6u
  DEL

$ STTY='stty -g < /dev/tty003'

$ stty -icanon -echo -opost ixon min 12 time 0 < /dev/tty003

$ keyload -t 8bin tty003 franz

$ stty $STTY < /dev/tty003
```

Zunächst wird die internationale Tastatur eingestellt (durch *cat* wird die Eingabe auch an den eigenen Bildschirm geschickt, wodurch auch der Bildschirm an dem Sie arbeiten umgestellt wird. Dies müssen Sie gegebenenfalls korrigieren). Danach wird die momentane Einstellung der Datensichtstation *tty003* in der Variablen *STTY* gespeichert und diese Datensichtstation mit *stty* für das Kommando *keyload* vorbereitet. Nach der Ausführung von *keyload* wird die gespeicherte Einstellung wieder hergestellt.

kill
Signale an Prozesse senden

Das Kommando *kill* sendet ein Signal an eine durch die Prozeßnummer (PID) bestimmte Menge von Prozessen. Um die Prozeßnummer des Prozesses zu erfahren, an den Sie ein Signal senden wollen, verwenden Sie das Kommando *ps*.

kill[␣-signal]␣prozessnummer␣... Format 1
kill␣-signal␣-gruppennummer␣... Format 2
kill␣-l Format 3

Format 1: Signale an Prozesse senden

kill[␣-signal]␣prozessnummer␣...

-signal
Signal, das an die Prozesse gesendet werden soll. Sie können dieses Signal in Form einer Zahl oder als symbolischen Namen angeben. Ein symbolischer Name ist die Bezeichnung eines Signals entsprechend der Definition in der Include-Datei *<sys/signal.h>*; das Präfix SIG wird allerdings nicht angegeben. Die Liste dieser Namen können Sie sich mit der Option *-l* ausgeben lassen.

Alle in der Include-Datei *<sys/signal.h>* definierten Signale können angegeben werden. Folgende Signale sind auf Kommando-Ebene von Bedeutung:

Signal. Nr.	Symb. Name	Bedeutung
1	SIGHUP	Verbindung zu Datensichtstation unterbrechen (hangup)
2	SIGINT	unterbrechen durch DEL (interrupt)
3	SIGQUIT	abbrechen (quit)
9	SIGKILL	unbedingter Prozeßabbruch (kill)
15	SIGTERM	Programmbeendigung (software termination)

signal nicht angegeben:
kill sendet das Signal SIGTERM (15) an die angegebenen Prozesse. Dadurch werden Prozesse, die dieses Signal nicht abfangen oder ignorieren, in der Regel abgebrochen.

prozessnummer
Nummer des Prozesses, an den Sie ein Signal senden wollen.

Benutzer ohne Systemverwalterrechte können nur an eigene Prozesse Signale senden.
Der Systemverwalter kann Signale an alle Prozesse senden.

Die aktuellen Prozeßnummern gibt das Kommando *ps* aus.

0 als Prozeßnummer bedeutet: Das angegebene Signal wird an alle Prozesse aus Ihrer Prozeßgruppe gesendet.

Format 2: Signale an Prozeßgruppen senden

kill -signal -gruppennummer ...

-signal
Signal, das an die Prozesse einer Prozeßgruppe gesendet werden soll. Sie können dieses Signal in Form einer Zahl oder als symbolischen Namen angeben. Ein symbolischer Name ist die Bezeichnung eines Signals entsprechend der Definition in der Include-Datei *<sys/signal.h>*; das Präfix SIG wird allerdings nicht angegeben. Die Liste dieser Namen können Sie sich mit der Option *-l* ausgeben lassen.

-gruppennummer
kill sendet das Signal an alle Prozesse der Prozeßgruppe *gruppennummer*.

Benutzer ohne Systemverwalterrechte können nur an eigene Prozeßgruppen Signale senden.
Der Systemverwalter kann Signale an alle Prozeßgruppen senden.

Wird *kill* zusammen mit dem Signal SIGKILL aufgerufen, so wird neben den Prozessen der angegebenen Prozeßgruppe auch der Prozeßgruppenführer abgebrochen.

Wenn Sie für *gruppennummer* die Zahl *1* angeben, so wird *signal* an alle Prozesse gesendet, deren realer User gleich dem effektiven User des kill-Kommandos ist.

Format 3: Symbolische Signalnamen auflisten

kill -l

-l
kill listet die symbolischen Signalnamen auf.

FEHLERMELDUNG

`no such process`

Sie haben für *prozessnummer* einen ungültigen Wert angegeben.

`no such process group`

Sie haben für *gruppennummer* einen ungültigen Wert angegeben.

DATEIEN

<sys/signal.h>
Include-Datei, in der die symbolischen Namen der Signale definiert sind.

BEISPIELE

1. Der Prozeß mit der Nummer 312 wird durch das Signal SIGKILL (9) beendet:

```
$ kill -9  312
```

2. Die im Hintergrund laufenden *find*-Kommandos werden durch das an die Nummer ihrer Prozeßgruppe gesendete Signal SIGTERM (15) beendet.

```
$ ps
   PID  PGID   SID TTY      TIME COMD
 13320 13278 13278 tty004   0:08 find
 13324 13278 13278 tty004   0:00 ps
 13278 13278 13278 tty004   0:01 sh
 13322 13278 13278 tty004   0:04 find

$ kill -TERM -13278

$ ps
   PID  PGID   SID TTY      TIME COMD
 13336 13278 13278 tty004   0:00 ps
 13278 13278 13278 tty004   0:01 sh
13320 Terminated
13322 Terminated
```

SIEHE AUCH

ps, sh, trap
kill(), signal() [19]

ksh
Kommandointerpreter und Programmiersprache Korn-Shell

Die Kommandobeschreibung ist in folgende Abschnitte unterteilt:

- Wesentliche Erweiterungen der Korn-Shell gegenüber der Bourne-Shell
- Bereitzeichenausgabe
- Definitionen
- Kommandos
- Zusammengesetzte Anweisungen
- Kommentare
- Alias-Variablen
- Kommandoersetzung
- Korn-Shell-Variablen und Parameterersetzung
- Blank-Ersetzung
- Dateinamen-Erzeugung
- Entwertung von Metazeichen (quoting)
- Arithmetische Berechnungen
- Bedingte Ausdrücke
- Eigenschaften von Dateien
- Eigenschaften und Vergleiche von Zeichenketten
- Algebraischer Vergleich ganzer Zahlen
- Bedingungen verknüpfen oder negieren
- Ein- und Ausgabe eines Kommandos umlenken
- Umgebung
- Funktionen
- Aufträge
- Signale
- Ausführung
- Kommando-Wiederaufruf
- Option des Zeileneditors
- vi-Editiermodus
- Eingabeanweisungen
- Positionieranweisungen
- Suchanweisungen
- Anweisungen zur Textmodifikation
- Sonstige Editieranweisungen
- Eingebaute Kommandos

Die Korn-Shell ist ein programmierbarer Kommandointerpreter, der seine Kommandos von einer Datensichtstation oder aus einer Datei liest, interpretiert und für deren Ausführung sorgt.
Ähnlich wie die Standard-Shell (die Bourne-Shell *sh*) oder die C-Shell (*csh*) ist die Korn-Shell eine Benutzerschnittstelle zum Betriebsystem-Kern. Die Korn-Shell ist von Syntax und Semantik weitgehend kompatibel zur Bourne-Shell und stellt eine Erweiterung dieser dar. Bourne-Shell-Prozeduren sind auch unter der Korn-Shell ablauffähig. Die C-Shell hat einen ähnlich mächtigen Funktionsumfang wie die Korn-Shell, besitzt aber eine an die Programmiersprache C angepaßte Syntax.
Für den geübten Bourne-Shell-Anwender sind unten die wesentlichen Erweiterungen von Bourne- zu Korn-Shell dargestellt. Der ungeübte Benutzer findet in den Manualseiten der Bourne-Shell (siehe *sh*) und den Seiten für die eingebauten Bourne-Shell-Kommandos ausführlichere Erklärungen.

Wesentliche Erweiterungen der Korn-Shell gegenüber der Bourne-Shell

- Bidirektionale Pipelines (|&) zwischen Kommandos
- Die *select*-Anweisung ermöglicht die interaktive Auswahl aus einer Wortliste.
- *time* als Anweisung
- Aliasing für Kommandonamen
- Tilde-Ersetzung für Dateinamen
- Geänderte Syntax der Kommandoersetzung ergänzt durch arithmetische Ausdrücke.
- Parameter können über das eingebaute *ksh*-Kommando *typeset* Attribute wie Format oder Datentyp erhalten.
- Eindimensionale Felder werden unterstützt.
- Die automatisch gesetzten und die von der Korn-Shell verwendeten Variablen wurden ergänzt.
- Die Ersetzung von Dateinamen ist um Musterlisten erweitert. Bei der Umlenkung von Ein- und Ausgabe können Muster zur Erzeugung eines einzelnen Dateinamens eingesetzt werden.
- Das *let*-Kommando wurde für Integer-Arithmetik eingebaut.
- Bedingte Ausdrücke mit einem ähnlichen Umfang wie das *test*-Kommando wurden als Anweisung eingebaut.
- Funktionen können nun lokale Variable haben, sind "exportierbar" und über einen Suchpfad auffindbar.

- Auftragssteuerung (job control), ist über die *monitor*-Option des eingebauten Kommandos *set* einsetzbar, mit einfachem Zugriff auf Prozesse - ähnlich dem der Shell *jsh* (Bourne-Shell mit Auftragssteuerung).
- Bei der Bearbeitung der Kommandozeile wird zuerst nach eingebauten Kommandos und dann nach Funktionen gesucht.
- Stellungsparameter werden vor Funktionsaufrufen gesichert und danach wiederhergestellt.
- *hash* entfällt und wird durch Kommando-Pfad ersetzt (siehe *alias*)
- Kommandowiederaufruf: Die abgesetzten Kommandos einer interaktiven Korn-Shell werden in einer *History*-Datei gespeichert. Auf diese kann dann über das eingebaute Kommando *fc* zugegriffen werden. Dabei ist einfacher Textersatz vor der Ausführung möglich. Durch eine Option ist das Editieren (*ed*- oder *vi*-Modus) einzelner Kommandozeilen aus der *History*-Datei vor deren Ausführung möglich.
- Beim Aufruf durch *login* kann nach der Datei *$HOME/.profile* noch die Datei, deren Name durch den Wert der Shell-Variable *ENV*, geben ist, aufgerufen werden.
- Die eingebauten Kommandos wurden zum Teil in ihrer Funktion erweitert, durch neue ergänzt, oder entfernt:

Erweiterte Kommandos

Kommando	Erweiterung
.	Argumente
:	Argumente
cd	Ersetzung
export	Wertzuweisung
getopts	Shell-Variable
read	Bidirektionale Pipeline
readonly	Wertzuweisung
set	u.a. Feld-Wertzuweisung
shift	Arithmetischer Ausdruck
trap	Spezielle Signale
umask	Symbolische Angaben wie *chmod*
unset	Keine Readonly-Variablen
wait	Auftragssteuerung

Neue Kommandos

Kommando	Funktion
alias	Kommandonamen
bg	Auftragssteuerung
fc	Zugriff auf die History-Datei
fg	Auftragssteuerung
jobs	Auftragssteuerung
kill	Auftragssteuerung
let	Integer-Aritmetik
print	Ausgabemechanismus ähnlich *echo*
typeset	Attribute für Shell-Variable
unalias	Alias-Variable löschen
whence	Kommando-Lokalisation

Entfallene Kommandos

Kommando	ersetzt durch
test	bedingte Ausdrücke als Anweisung
hash	*alias -t*

ksh[_option..][_datei][_argument]...

Kommandos werden nacheinander aus der Datei */etc/profile* und dann, falls eine der Dateien existiert, entweder aus *.profile* des aktuellen Verzeichnisses oder aus *$HOME/.profile* gelesen. Danach werden noch die Kommandos aus der Datei gelesen, deren Name durch den Wert der Shell-Variable *ENV* nach Parametersubstitution gegeben wird.

option

Die folgenden Optionen werden von *ksh* beim Aufruf interpretiert:

-c_kommando_zeichenkette
Die Kommandos werden aus *kommando_zeichenkette* gelesen.

-s
Ist Option *-s* angegeben oder fehlen *datei* und *argument*, dann liest die Korn-Shell die Kommandos von der Standard-Eingabe und schreibt die Ausgaben auf die Standard-Fehlerausgabe, während die eingebauten *ksh*-Kommandos auf die Standard-Ausgabe schreiben.

-i

Ist Option *-i* angegeben oder sind Standard-Eingabe und -Ausgabe mit einer Datensichtstation verbunden, dann wird eine interaktive Korn-Shell aufgerufen. In diesem Fall wird zum einen TERM ignoriert, so daß *kill 0* keine interaktive Shell beendet, und zum anderen INTR abgefangen und ignoriert, damit *wait* unterbrechbar ist. Auf jeden Fall wird QUIT von der Korn-Shell ignoriert.

-r

Mit Option *-r* wird eine eingeschränkte Korn-Shell gestartet.

Die restlichen Optionen sind beim eingebauten Kommando *set* der Korn-Shell beschrieben.

datei

Wurde die Option *-s* nicht angegeben, dafür aber *datei*, dann wird entsprechend dem Suchpfad nach der Shell-Prozedur *datei* gesucht. Für die Prozedur muß das Leserecht gesetzt sein. Falls das s-Bit für Eigentümer oder Gruppe gesetzt ist, so wird dieses ignoriert. Kommandos werden wie im Folgenden beschrieben gelesen.

argument

Als Argumente können Sie Kommandos angeben. Der Kommandoname wird als nulltes Argument übergeben. Die Kommandos sind unten beschrieben.
Wird die Korn-Shell vom Systemaufruf *exec* aufgerufen und ist das erste Zeichen des nullten Arguments ein Bindestrich -, dann wird die Shell als *login*-Shell behandelt.

Bereitzeichenausgabe

Wenn Sie die Korn-Shell interaktiv benutzen, dann gibt die Shell den Wert der Umgebungsvariablen *PS1* als Bereitzeichen aus, bevor sie ein Kommando einliest.

Geben Sie zu irgendeiner Zeit ein Neue-Zeile-Zeichen ein und benötigt die Shell dann noch weitere Eingabe zur Vervollständigung des Kommandos, dann zeigt sie dies durch das zweite Bereitzeichen an. Das zweite Bereitzeichen besteht im allgemeinen aus dem Wert von *PS2*.

Definitionen

Ein Metazeichen (metacharacter) ist eines der folgenden Zeichen:

; & ( ) | < > Leerzeichen Tabulatorzeichen Neue-Zeile-Zeichen

Ein *Blank* ist ein Tabulator oder ein Leerzeichen. Ein Bezeichner besteht aus einer Folge von Buchstaben, Ziffern und dem Unterstrich _, die mit einem Buchstaben oder dem Unterstrich beginnt. Bezeichner werden als Namen für Funktionen und Variable benutzt. Ein Wort ist eine Folge von Zeichen, die durch ein oder mehrere nicht entwertete Metazeichen getrennt wird.

Kommandos

Ein Kommando ist eine Folge von Zeichen in der Syntax der Korn-Shell Programmiersprache. Die Shell liest jedes Kommando und führt die notwendige Aktion entweder direkt oder durch den Aufruf eines eigenen Dienstprogramms aus.

Ein eingebautes Kommando wird von der Korn-Shell direkt, ohne Erzeugung eines weiteren Prozesses, ausgeführt. Die meisten eingebauten Kommandos können Sie als separate Dienstprogramme implementieren. Einige Ausnahmen, die zu unerwünschten Nebeneffekten führen können, sind weiter unten dokumentiert.

Ein einfaches Kommando besteht aus einer Folge von Wörtern, die durch Blanks (Leerzeichen) getrennt sind. Davor kann noch eine Liste von Variablenwertzuweisungen stehen (siehe *Umgebung*). Das erste Wort steht für den Namen des auszuführenden Kommandos. Die restlichen Wörter werden als Argumente an das aufgerufene Kommando übergeben, bis auf später beschriebene Ausnahmen.

Der Kommandoname wird als nulltes Argument übergeben. Der Wert eines einfachen Kommandos ist sein Ende-Status, wenn es normal endet oder (Oktal) *200 + status*, wenn es abnormal endet (signal enthält eine Liste der *status*-Werte).

Eine Pipeline besteht aus mehreren Kommandos, die durch den senkrechten Strich | getrennt sind. Die Standard-Ausgabe jedes Kommandos (außer des letzten) ist durch einen *pipe*-Systemaufruf mit der Standard-Eingabe des folgenden Kommandos verbunden. Jedes Kommando läuft als ein eigener Prozeß, die Korn-Shell wartet, bis der letzte endet. Der Wert einer Pipeline ist der Wert des letzten Kommandos.

Eine Kommandofolge ist die Aneinanderreihung einer oder mehrerer Pipelines, die durch eines der Symbole

; & && ||

getrennt und optional durch

; & |&

abgeschlossen werden. Die Symbole ; & und |& haben gleichen Vorrang, der niedriger ist als der von && und ||, die beide ebenfalls gleichen Vorrang haben. Der Strichpunkt ; nach einer Pipeline steht für sequentielle Ausführung der Pipeline, während das kommerzielle Und & asynchrone Ausführung bedeutet, d.h. die Shell wartet nicht auf die Beendigung der Pipeline. Das Symbol |& nach einer Kommandofolge veranlaßt die asynchrone Ausführung der Pipeline oder des Kommandos mit einer bidirektionalen Pipeline zur Vater-Shell.

Auf die Standard-Eingabe des erzeugten Kommandos kann die Vater-Shell mit Hilfe der Option *-p* des eingebauten Kommandos *print* schreiben. Von der Standard-Ausgabe kann die Vater-Shell mit *read -p* lesen. Beide eingebauten Kommandos werden später beschrieben. Das Symbol && (||) veranlaßt die Ausführung der auf das Symbol folgenden Kommandofolge nur dann, wenn die vorausgehende Pipeline den Wert Null (einen Wert ungleich Null) liefert. In einer Kommandofolge könne eine beliebige Menge von Neue-Zeile-Zeichen als Trenner anstelle eines Strichpunkts vorkommen.

Ein Kommando ist entweder ein einfaches Kommando oder eine der folgenden zusammengesetzten Anweisungen. Wenn nicht anders angegeben, dann ist der Wert eines Kommandos immer der des letzten ausgeführten einfachen Kommandos.

Zusammengesetzte Anweisungen

```
for␣bezeichner␣[␣in␣wort...];do␣kommandofolge;done
```

oder

```
for␣bezeichner␣[␣in␣wortliste]
do␣kommandofolge
done
```

Mit der *for*-Anweisung können Sie *kommandofolge* mehrmals wiederholen. Dabei wird *bezeichner* auf jedes Wort der *wortliste* gesetzt und die Schleife einmal durchlaufen. Ist *in wortliste* nicht angegeben, dann wird *kommandofolge* für jeden gesetzten Stellungsparameter ("*$@*") einmal durchlaufen. Die Ausführung endet, wenn *wortliste* abgearbeitet ist.

```
select␣bezeichner␣[␣in␣wortliste]
do␣kommandofolge
done
```

Mit der *select*-Anweisung können Sie *kommandofolge* mehrmals durch Eingabe gesteuert wiederholen. *select* schreibt *wortliste* auf die Standard-Ausgabe. Dabei wird jedem Wort eine Nummer vorangestellt. Danach wird das Bereitzeichen *PS3* ausgegeben und eine Zeile von der Standard-Eingabe eingelesen. Der Inhalt der gelesenen Zeile wird der Variable *REPLY* als Wert zugewiesen. Enthält diese Zeile die Nummer eines der angezeigten Wörter, dann wird an *bezeichner* das entsprechende Wort als Wert zugewiesen. Bei leerer Zeile wird die Ausgabe von *wortliste* wiederholt und *bezeichner* der Name der Shell-Prozedur zugewiesen. Wurde *in* wortliste weggelassen, dann werden dafür, wie bei der *for*-Schleife, die Stellungsparameter eingesetzt. Die *select*-Schleife wird solange durchlaufen, bis sie durch das eingebaute Kommando *break* oder durch Eingabe von Dateiende abgebrochen wird.

Beispiel

Mit dieser Korn-Shell-Prozedur können Sie Informationen über jede Datei des aktuellen Verzeichnisses einzeln abfragen:

```
select datei in `ls`
do
  if [ -z "$datei" ]
  then
     echo "Zahl auswaehlen"
  else
     file $datei
     ls -lsid $datei
  fi
done
```

case *wort* **in**
[(]*muster*[|*muster*]...)*kommandofolge*;;]...
esac

Die *case*-Anweisung gestattet Ihnen, mustergesteuert auf eine Kommadofolge zu verzweigen. Die erste *kommandoliste*, deren *muster* auf *wort* paßt, wird ausgeführt. Die Muster werden in derselben Art angegeben, wie sie für die Erzeugung von Dateinamen verwendet werden (siehe *Dateinamen-Erzeugung*).

Vorsicht

Die optionale öffnende Klammer vor *muster* müssen Sie angegeben, wenn Sie *case*-Anweisungen innerhalb der Kommandosubstitution mit *$(...)* verwenden. Damit erreichen Sie paarweise Klammerung innerhalb von *$(...)*.

if *kommandofolge1*
then *Kommandofolge2*
[**elif** *kommandofolge3*
then *kommandofolge4*]...
[**else** *kommandofolge5*]
fi

Mit der *if*-Anweisung können Sie entsprechend einer Bedingung verschiedene Kommandofolgen ausführen lassen. *kommandofolge1* folgend auf *if* wird als Bedingung ausgeführt. Ist deren Ende-Status 0 gleich wahr, dann wird *kommandofolge2* (*then* folgend) ausgeführt. Bei falsch wird die Bedingung *kommandofolge3* nach *elif* ausgeführt. Ist der Ende-Status dieser Bedingung 0, wird *kommandofolge4* hinter dem nächsten *then* ausgeführt. Im falsch-Fall wird *kommandofolge5* nach *else* ausgeführt. Wurde keine der Kommandofolgen nach *then* oder *else* ausgeführt, dann bekommt die *if*-Anweisung den Ende-Status 0 zugewiesen.

```
while␣kommandofolge1
do␣kommandofolge2
done

until␣kommandofolge1
do␣kommandofolge2
done
```

Durch die *while-* und *until*-Anweisungen erhalten Sie Schleifen mit Abbruchbedingung. Die *while*-Anweisung führt die Bedingung *kommandofolge1* aus. Im wahr-Fall (Ende-Status 0 des letzten ausgeführten einfachen Kommandos) wird der Schleifenkörper *kommandofolge2* ausgeführt und die Bedingung erneut geprüft, während im falsch-Fall die Schleife beendet wird. Wurde kein Kommando aus *kommandofolge2* (nach *do*) ausgeführt, dann erhält die *while*-Anweisung den Ende-Status 0 zugewiesen.

Die *until*-Anweisung können Sie anstelle der *while*-Anweisung bei negierter Bedingung verwenden. *until* prüft nach, ob die Bedingung falsch ist und bricht, sobald das Ergebnis wahr wird, die Schleife ab.

```
(kommandofolge)
```

kommandofolge wird in einer eigenen Umgebung ausgeführt.

Vorsicht

Werden zur Schachtelung zwei aufeinanderfolgende öffnende runde Klammern (benötigt, dann müssen Sie beide durch ein Leerzeichen trennen. Es könnte sonst zu einer Verwechslung mit einem arithmetischen Ausdruck kommen (siehe *Kommandoersetzung*).

```
{␣kommandofolge;}
```

kommandofolge wird in der aktuellen Korn-Shell einfach ausgeführt.

Vorsicht

Im Unterschied zu den runden Klammern (und) sind die geschweiften Klammern { und } reservierte Wörter. Damit sie als solche erkannt werden, müssen { und } am Zeilenanfang oder nach einem Strichpunkt ; stehen.

```
[[␣bedingung␣]]
```

Der bedingte Ausdruck **bedingung** wird bewertet, und der Ende-Status bei wahr auf 0 und bei falsch auf 1 gesetzt.

Siehe *Bedingter Ausdruck* für eine Beschreibung von *bedingung*.

function bezeichner { kommandofolge;}
bezeichner () { kommandofolge;}

Definition der Funktion mit dem Namen *bezeichner*. Dieser Name wird wie bei einem Kommando zum Aufruf verwendet. Der Funktionskörper besteht aus der *kommandofolge*, die in geschweifte Klammern eingeschlossen ist.

Vorsicht

Auf die öffnende Klammer { muß ein Leerzeichen folgen!

time pipeline

Die Kommandos von *pipeline* werden ausgeführt und als Statistik die verstrichene Zeit, die *user*- und die *system*-Zeit auf der Standard-Fehlerausgabe berichtet. Die folgenden reservierten Wörter werden nur als erstes Wort eines Kommandos erkannt und wenn sie nicht entwertet sind.

```
if      then    elif    else    fi      case    esac    for
select  while   until   do      done    {       }       function
time    [[      ]]
```

Kommentare

Beginnt ein Wort mit dem Nummernzeichen #, dann werden die restlichen Zeichen der Zeile einschließlich # ignoriert.

Alias-Variablen

Wenn Sie für das erste Wort eines Kommandos eine Alias-Variable definiert haben, dann wird dieses durch den Text der Alias-Variablen ersetzt. Der Name einer Alias-Variablen kann aus einer beliebigen Anzahl von Zeichen bestehen, zu denen nicht die Metazeichen, die Entwertungszeichen, die Zeichen für die Dateinamen-, die Parameter- und Kommandoersetzung und das Gleichheitszeichen = gehören. Die Ersetzungs-Zeichenkette kann jede korrekte Korn-Shell-Prozedur, inklusive der vorhergehenden Metazeichen-Aufzählung, enthalten. Das erste Wort eines jeden Kommandos im ersetzten Text wird wiederum auf weitere Alias-Variablen getestet. Wenn das letzte Zeichen des Alias-Werts ein Leerzeichen ist, dann wird das auf die Alias-Ersetzung folgende Wort ebenfalls auf Alias-Ersetzung untersucht.

Mit Alias-Variablen können Sie eingebaute Shell-Kommandos neu definieren, Sie können sie aber nicht zur Neudefinition der reservierten Wörter (siehe oben) verwenden. Mit Hilfe des *alias*-Kommandos können Alias-Variablen definiert, exportiert und zum Auflisten auf die Standard-Ausgabe geschrieben werden, während sie mit dem *unalias*-Kommando wieder gelöscht werden können. Exportierte Alias-Variablen bleiben wirksam für Prozeduren, die per Name aufgerufen werden, müssen aber neu für explizite weitere Aufrufe der Korn-Shell initialisiert werden.

Aliasing wird beim Lesen von Prozeduren durchgeführt, und nicht wenn diese ausgeführt werden. Dementsprechend muß eine Alias-Definition vor deren erster Verwendung beim Lesen eines Kommandos mit der Alias-Variablen erfolgen.

Alias-Variablen werden häufig als Abkürzung für volle Pfadnamen verwendet. Eine Option des eingebauten *alias*-Kommandos sorgt dafür, daß der Wert der Alias-Variablen automatisch auf den vollen Pfadnamen des zugehörenden Kommandos gesetzt wird. Diese Alias-Variablen nennt man "mit Pfad versehene Alias-Variablen" (tracked alias). Der Wert einer solchen Variablen wird beim ersten Aufsuchen des zugehörenden Kommandos definiert und jedesmal zurückgesetzt, wenn die *PATH*-Variable neu gesetzt wird. Diese Alias-Variablen behalten den Zustand "mit Pfad versehen", so daß der nächste folgende Zugriff den Wert neu definieren wird. Einige mit Pfad versehene Alias-Variablen sind mit in die Korn-Shell übersetzt und compiliert. Die Option *-h* des eingebauten Kommandos *set* macht jeden aufgerufenen Kommandonamen zu einer mit Pfad versehenen Alias-Variablen.

Die folgenden exportierten Alias-Variablen sind in die Korn-Shell compiliert und können neu definiert oder mit *unalias* gelöscht werden:

```
autoload='typeset -fu'
false='let 0'
functions='typeset -f'
hash='alias -t'
history='fc -l'
integer='typeset -i'
nohup='nohup '
r='fc -e -'
true=':'
type='whence -v'
```

Beispiele

Durch die folgenden Alias-Variablen können Sie *l*, *ll* und *lf* nachbilden:

```
alias l='/bin/ls -m'
alias ll='/bin/ls -l'
alias lf='/bin/ls -CF'
```

Tilde-Ersetzung

Nach der Alias-Ersetzung wird jedes Wort getestet, ob es mit einer nicht entwerteten Tilde ~ beginnt. Wenn dies der Fall ist, dann wird das Wort bis zum nächsten Schrägstrich / auf die Übereinstimmung mit einem Login-Namen aus der Datei */etc/passwd* untersucht. Ist dies gegeben, dann wird die Tilde und der Login-Namen durch das Login-Verzeichnis des gefundenen Benutzers ersetzt. Dies wird Tilde-Ersetzung genannt. Wird keine Übereinstimmung gefunden, dann bleibt der Originaltext unverändert. Eine alleinstehende Tilde oder eine Tilde vor einem Schrägstrich wird durch *$HOME* ersetzt. Eine Tilde gefolgt von einem Plus- oder Minuszeichen wird durch *$PWD* (+) oder *$OLDPWD* (-) ersetzt.

Tilde-Ersetzung wird auch versucht, wenn der Wert einer Variablenwertzuweisung mit einer Tilde beginnt.

Beispiel

Mit dem Kommando

```
vi ~/.profile
```

können Sie von beliebiger Stelle die Datei *.profile* in Ihrem Login-Verzeichnis editieren.

Kommandoersetzung

Durch die Standard-Ausgabe eines Kommandos können Sie auf zwei Arten einen Teil eines Wortes oder ein ganzes Wort ersetzen. Bei der ersten (neuen) Art wird das Kommandos in Dollar runde Klammer auf, runde Klammer zu *$(...)* eingeschlossen. Bei der zweiten (archaischen) Art wird das Kommando in Gegenhochkommata `...` eingeschlossen. Hier wird die Zeichenkette zwischen den Hochkommata auf Entwertungen oder Anführungszeichen untersucht, bevor das Kommando ausgeführt wird (siehe *Entwertung*). Bei beiden Arten werden Neue-Zeile-Zeichen der Kommandoausgabe gelöscht.

Die Ersetzung *$(cat datei)* kann durch die äquivalente, aber schnellere Version *$(< datei)* durchgeführt werden. Die Ersetzung wird von eingebauten Shell-Kommandos, die keine Umlenkung der Ein- oder Ausgabe durchführen, ohne die Erzeugung eines neuen Prozesses durchgeführt.

Beispiel

Wenn Sie in einem Verzeichnis alle Dateien mit der Endung .c und der Zeichenkette *include* editieren wollen, dann hilft Ihnen folgendes:

```
vi $( grep -l include *.c )
```

Ein arithmetischer Ausdruck, in Dollar doppelte runde Klammer auf und doppelte runde Klammer zu *$((...))* eingeschlossen, wird durch den Wert des arithmetischen Ausdrucks ersetzt.

Beispiel

Sie wollen den vorletzten Parameter einer Shell-Prozedur ausgeben:

```
eval print \$$(( $#-1 ))
```

Korn-Shell-Variablen und Parameterersetzung

Ein Parameter wird durch einen Bezeichner, eine oder mehrere Ziffern oder eines der folgenden Zeichen dargestellt:

* @ # ? - $!

Eine Variable (ein Parameter gekennzeichnet durch einen Bezeichner) hat einen Wert und keine oder mehrere Attribute. Variablen können Sie Werte und Attribute durch das eingebaute Shell-Kommando *typeset* zuweisen. Die verwendbaren Attribute werden bei *typeset* beschrieben. Exportierte Parameter geben ihren Wert und die Attribute an die Umgebung weiter.

Die Korn-Shell stellt eindimensionale Felder zur Verfügung. Ein Element des Feldes wird über einen Index angesprochen. Ein Index wird durch eckige Klammer auf [, gefolgt von einem arithmetischen Ausdruck und eckige Klammer zu], beschrieben (zu den arithmetischen Ausdrücken siehe *Arithmetische Berechnungen*).

Einem Feld können Sie mit dem eingebauten Shell-Kommando *set* Werte zuweisen:
set -A bezeichner werte...
Die Werte aller Indizes müssen im zulässigen Wertebereich liegen. Felder müssen Sie nicht deklarieren, jeder Zugriff auf ein Feldelement mit zulässigem Index ist erlaubt. Wenn erforderlich, wird das Feld angelegt. Greifen Sie auf das Feld ohne Index zu, dann greifen Sie auf das nullte Element zu.

bezeichner=wert[␣bezeichner=wert]...
: Einer Variablen können Sie in dieser Form einen Wert zuweisen. Ist für eine Variable das Integer-Attribut gesetzt, dann kann bei arithmetischen Berechnungen ihr Wert eingesetzt werden. An Stellungsparameter (gekennzeichnet durch eine Zahl) können Sie Werte mit Hilfe des eingebaute Shell-Kommandos *set* zuweisen. Der Parameter *$0* wird beim Aufruf der Shell auf das nullte Argument gesetzt.

${parameter**}**
: Das Dollar-Zeichen dient zur Einführung von ersetzbaren Parametern. Die Korn-Shell liest alle Zeichen von *${* bis *}* als Teil desselben Wortes, auch wenn darin Klammern oder Metazeichen enthalten sind. Falls existent, wird der Wert des Parameters eingesetzt. Die geschweiften Klammern benötigen Sie auch, wenn hinter *parameter* Buchstaben, Ziffern oder Unterstrich folgen, die nicht Teil des Namens des Parameters sind, oder wenn eine Variable indiziert wird. Besteht *parameter* aus einer oder mehreren Ziffern, dann ist es ein Stellungsparameter. Einen Stellungsparameter mit mehreren Ziffern müssen Sie in geschweifte Klammern einschließen.

  Wird *parameter* durch die Zeichen Stern * oder Klammeraffe @ beschrieben, dann werden alle Stellungsparameter ab *$1* eingesetzt. Als Trenner wird dabei das erste Zeichen der Variablen *IFS* verwendet (siehe unten). Ein Feldbezeichner mit Index Stern * oder Klammeraffe @ wird durch die Werte aller Elemente ersetzt. Auch hier wird der gleiche Trenner verwendet.

${#parameter}
Wird *parameter* durch die Zeichen Stern * oder Klammeraffe @ gegeben, dann wird die Anzahl der Stellungsparameter eingesetzt. Sonst wird die Länge des Wertes von *parameter* eingesetzt.

${#bezeichner[*]}
Die Anzahl der Elemente des Feldes *bezeichner* wird eingesetzt.

${#parameter:-wort}
Wenn *parameter* gesetzt und nicht die leere Zeichenkette ist, dann wird der Wert, sonst *wort* eingesetzt.

${#parameter:=wort}
Ist *parameter* nicht gesetzt oder der Wert gleich der leeren Zeichenkette, dann wird *parameter* auf *wort* gesetzt. Anschließend wird der Wert eingesetzt. Stellungsparametern können Sie mit dieser Methode keine Werte zuweisen.

${#parameter:?wort}
Ist *parameter* gesetzt und der Wert nicht gleich der leeren Zeichenkette, dann wird der Wert eingesetzt; sonst wird *wort* ausgegeben und die Shell beendet. Fehlt *wort*, dann wird eine Standard-Fehlermeldung ausgegeben.

${#parameter:+wort}
Wenn *parameter* gesetzt und nicht die leere Zeichenkette ist, dann wird *wort* eingesetzt, sonst die leere Zeichenkette.

${#parameter#muster}
${#parameter##muster}
Sollte das Korn-Shell-Muster *muster* auf den Anfang des Wertes von *parameter* passen, dann besteht der Ersetzungstext aus dem Wert von *parameter*, aus dem der, auf *muster* passende, Teil gelöscht wurde. Sonst wird der Wert von *parameter* eingesetzt. Bei der ersten Form der Angabe wird das kürzeste passende Muster, in der zweiten Form das längste Auftreten des Musters gelöscht.

${#parameter%muster}
${#parameter%%muster}
Paßt das Korn-Shell-Muster *muster* auf das Ende des Wertes von *parameter*, dann wird der Ersetzungstext aus dem Wert von *parameter* ohne den gelöschten auf *muster* passenden Teil gebildet. Bei der ersten Form der Angabe wird das kürzeste, bei der zweiten Form das längste passende Muster gelöscht.

Bei den letzten 8 Ersetzungen wird *wort* erst dann bewertet, wenn es als Ersetzungs-Zeichenkette verwendet wird.

Beispiel

```
echo ${dvz:-$(pwd)}
```

pwd wird erst dann ausgeführt, wenn *dvz* nicht gesetzt oder gleich der leeren Zeichenkette ist.

Wird der Doppelpunkt : bei den obigen Ausdrücken weggelassen, dann kontrolliert die Korn-Shell nur, ob *parameter* gesetzt ist oder nicht. Die Überprüfung auf die leere Zeichenkette entfällt.

Die folgenden Parameter werden automatisch von der Korn-Shell gesetzt:

#
Die Anzahl (dezimal) der positionellen Parameter.

-
Alle Optionen, die beim Aufruf der Korn-Shell oder durch das eingebaute Kommando *set* gesetzt wurden.

?
Der Ende-Status des zuletzt ausgeführten Kommandos.

$
Die Prozeß-Nummer der aktuellen Korn-Shell.

_
Am Anfang wird der Wert von Unterstrich _ auf den absoluten Pfadnamen der Korn-Shell oder der ausgeführten Korn-Shell-Prozedur, wie er an die Umgebung übergeben wird, gesetzt. Später wird _ immer das letzte Argument des vorhergehenden Kommandos als Wert zugewiesen. Dieser Parameter wird nicht für asynchrone Hintergrund-Kommandos gesetzt. Bei der Suche nach Post wird der Parameter _ für die Speicherung des Namens der passenden *MAIL*-Datei verwendet.

Beispiel

```
cat /usr/tmp/mydir/xyz* > alles
tail $_
```

Das Kommando *tail* greift über *$_* auf die letzte Datei des vorhergehenden *cat*-Kommandos zu.

!
Die Prozeß-Nummer des zuletzt als Hintergrundprozeß gestarteten Kommandos.

ERRNO
Der Wert dieses Parameters wird vom letzten fehlerhaften Systemaufruf gesetzt. Der Wert ist systemabhängig und dient der Fehlersuche.

LINENO
Die Zeilenummer der aktuellen Zeile in der Prozedur oder in der Funktion, die gerade ausgeführt wird.

OLDPWD
Das letzte aktuelle Dateiverzeichnis, gesetzt durch das *cd*-Kommando.

OPTARG
Der Wert des letzten Options-Arguments, das von dem eingebauten Kommando *getopt* bearbeitet wurde.

OPTIND
Der Index des letzten Options-Arguments, das von dem eingebauten Kommando *getopt* bearbeitet wurde.

PPID
Die Prozeß-Nummer des Vater-Prozesses der aktuellen Korn-Shell.

PWD
Das aktuelle Dateiverzeichnis, gesetzt durch das *cd*-Kommando.

RANDOM
Jedesmal, wenn auf diese Variable zugegriffen wird, wird eine Zufallszahl aus dem Wertebereich von 0 bis 32767 berechnet. Die Folge der Zufallszahlen kann durch eine Wertzuweisung an die Variable *RANDOM* initialisiert werden.

REPLY
Diese Variable wird bei fehlenden Argumenten von der *select*-Anweisung oder vom eingebauten Kommando *read* gesetzt.

SECONDS
Beim Zugriff auf diese Variable enthält ihr Wert die Zeit in Sekunden, die seit dem Aufruf der Korn-Shell vergangen ist. Wird der Variablen ein Wert zugewiesen, dann wird ihr Wert beim Zugriff auf den Zuweisungswert plus die vergangene Zeit seit der Zuweisung gesetzt.

Die folgenden Variablen werden von der Korn-Shell benutzt:

CDPATH
Der Suchpfad für das *cd*-Kommando.

COLUMNS
Wenn diese Variable gesetzt ist, dann wird ihr Wert für die Definition der Breite des Editierfensters beim Editiermodus der Korn-Shell und für die Ausgabe des *select*-Liste verwendet.

EDITOR
Endet der Wert dieser Variable mit *vi* und ist die Variable *VISUAL* nicht gesetzt, dann wird die entsprechende Option (siehe *Eingebaute Kommandos, set*) gesetzt.

ENV
Wenn diese Variable gesetzt ist, dann enthält ihr Wert den Pfadnamen der Prozedur, die bei Aufruf der Korn-Shell ausgeführt wird (siehe *Aufruf*). Diese Prozedur wird meist für Funktions- und Alias-Definitionen benutzt. Auf den Wert der Variablen wird Parameterersetzung zur Dateinamen-Erzeugung durchgeführt.

FCEDIT
Der Name des Standard-Editors für das eingebaute Kommando *fc*.

FPATH
Der Suchpfad für Funktionsdefinitionen. Dieser Pfad wird verwendet, wenn auf eine Funktion mit Attribut *-u* zugegriffen wird und wenn kein Kommando gefunden wurde. Wird eine ausführbare Datei gefunden, dann wird sie gelesen und in der momentane Umgebung ausgeführt.

IFS
(Internal field separators) Interner Feldtrenner der Korn-Shell, der zur Trennung von Wörtern dient, die aus Kommando- oder Parameterersatz entstehen. Der Feldtrenner wird auch durch das eingebaute Kommando *read* verwendet. Normalerweise ist der Wert auf Leer-, Tabulator- und Neue-Zeile-Zeichen gesetzt. Das erste Zeichen der *IFS*-Variable wird zur Trennung der Argumente bei der Ersetzung von *"$*"* verwendet (siehe *Entwertung*).

HISTFILE
Ist diese Variable bei Aufruf der Korn-Shell gesetzt, dann wird der Wert als Pfadname für die Datei zur Speicherung der Kommando-history verwendet (siehe *Kommando-Wiederaufruf*).

HISTSIZE
Wenn diese Variable bei Aufruf der Korn-Shell gesetzt ist, dann behält die Shell den Text eingegebener Kommandos in Erinnerung. Sie können mindestens auf die als Wert angegebene Anzahl von früher eingegebenen, und der Korn-Shell zugänglichen, Kommandos zurückgreifen. Der Standard-Wert ist 128.

HOME
Das Standard-Argument (Login-Dateiverzeichnis) für das Kommando *cd*.

LINES
Wenn diese Variable gesetzt ist, dann wird ihr Wert für die Berechnung der Spaltenzahl für die Ausgabe von *select*-Listen verwendet. *select*-Listen werden vertikal ausgegeben, bis ungefähr zwei drittel der Zeilenzahl *LINES* gefüllt sind.

MAIL
Enthält der Wert dieser Variable den Namen einer *mail*-Datei und ist die Variable *MAILPATH* nicht gesetzt, dann informiert Sie die Korn-Shell über das Eintreffen von Post in dieser Datei.

MAILCHECK
Der Wert dieser Variable gibt an, nach welchem Zeitintervall in Sekunden die Korn-Shell jeweils nach Änderungen der Modifikationszeit der, durch die Variablen *MAIL* oder *MAILPATH*, ausgewiesenen Dateien sehen soll. Der Standard-Wert für *MAILCHECK* ist 600. Wenn die angegebene Zeit verstrichen ist, dann prüft die Korn-Shell vor der Ausgabe des nächsten Bereitzeichens nach.

MAILPATH
Eine durch Doppelpunkt *:* getrennte Liste von Dateinamen. Bei gesetzter Variable informiert Sie die Korn-Shell über jede Änderung an den Dateien der Liste, die innerhalb der letzten *MAILCHECK* Sekunden erfolgt sind. Jeder Dateiname kann in der Liste von einem Fragezeichen *?* und einem Mitteilungstext, der ausgegeben werden soll, gefolgt werden. Diese Mitteilung wird der Parameter-Ersetzung mit der Variable *$_* unterzogen. *$_* enthält zu diesem Zeitpunkt den Namen der Datei, die sich geändert hat. Die Standard-Mitteilung ist: "*you have mail in $_*".

PATH
Der Suchpfad für Kommandos (siehe *Ausführung*). Sie können den Wert dieser Variable nicht verändern, wenn sie eine eingeschränkte Korn-Shell benutzen.

PS1
Der Wert dieser Variablen wird für den Parameterersatz expandiert, um das Bereitzeichen der Korn-Shell zu definieren. Der Standardwert ist "*$_*". Das Ausrufezeichen *!* im Bereitzeichen wird durch die Kommando-Nummer (siehe *Wiederaufruf*) ersetzt.

PS2
Das zweite Bereitzeichen, das die Korn-Shell ausgibt, wenn sie noch weitere Eingabe nach einem Neue-Zeile-Zeichen erwartet. Der Standardwert ist ">␣".

PS3
Das Bereitzeichen für die *select*-Anweisung, das zur Abfrage der Nummer innerhalb der *select*-Schleife verwendet wird. Der Standardwert ist "*#?*␣".

PS4

Der Wert dieser Variablen wird vor jeder Zeile einer Ausführungs-Verfolgung (execution trace) ausgegeben. Der Wert dieser Variablen wird für den Parameterersatz expandiert. Der Standardwert ist "+␣".

SHELL
Der Pfadnamen der Korn-Shell wird in der Umgebung gehalten. Beim Aufruf wird die Korn-Shell zu einer eingeschränkten Shell, wenn auf den Dateinamesteil des Pfadnamens (siehe *basename*) das Muster **r*sh** paßt.

TMOUT
Ist der Wert dieser Variablen positiv, dann beendet die Korn-Shell selbständig, wenn nach Ausgabe des Bereitzeichens (*PS1*) nicht innerhalb der angegebenen Zeitspanne (in Sekunden) ein Kommando eingegeben wird.
(Vorsicht: Die Korn-Shell kann mit einem Maximalwert für *TMOUT* compiliert worden sein, der nicht überschritten werden kann.)

VISUAL
Endet der Wert dieser Variablen mit *vi*, dann wird die entsprechende Option (siehe *Eingebaute Kommandos*) gesetzt.

Die Korn-Shell weist den folgenden Variablen Standardwerte zu:

PATH, PS1, PS2, MAILCHECK, TMOUT und *IFS*.

Die Variablen *HOME, MAIL* und *SHELL* werden vom Kommando *login* gesetzt.

Blank-Ersetzung

Nach Parameter- und Kommandoersetzung wird das Ergebnis nach Feldtrennzeichen durchsucht und an den Fundstellen in selbständige Argumente unterteilt. Die Feldtrenner werden durch den Wert der Variable *IFS* definiert. Explizit vorhandene leere Zeichenketten (z.B. "" oder '') bleiben dabei erhalten. Implizit vorhandene leere Zeichenketten, wie sie z.B. von Parametern ohne Wert herrühren, werden entfernt.

Dateinamen-Erzeugung

Nach den verschiedenen Ersetzungen wird jedes Wort auf das Auftreten von Stern *, Fragezeichen ? oder öffnende eckige Klammer [hin untersucht. Dies geschieht aber nur dann, wenn die Option *-f* (siehe *set*) nicht gesetzt wurde. Wird eines dieser Zeichen in einem Wort gefunden, dann wird dieses Wort als Muster betrachtet. Das Wort wird dann durch lexikographisch sortierte Dateinamen ersetzt, die auf das Muster passen. Wurde für das Muster kein Dateiname gefunden, dann wird das Wort unverändert gelassen. Wenn Sie Muster für die Erzeugung von Dateinamen verwenden, müssen Sie den Zeichen Punkt . und Schrägstrich / besondere Aufmerksamkeit widmen - bei anderen Ersetzungen gilt diese Sonderbehandlung der beiden Zeichen nicht. Punkt am Anfang eines Dateinamens oder unmittelbar nach /, sowie / selber müssen explizit passen.

*
wird durch jede Zeichenkette, auch die leere, ersetzt.

?
wird durch ein beliebiges Zeichen ersetzt. (Zur Behandlung von / und . siehe oben.)

[...]
wird durch genau ein Zeichen ersetzt, das in der Zeichenmenge innerhalb der eckigen Klammern enthalten ist.
Ein Zeichenpaar, getrennt durch den Bindestrich -, steht für alle Zeichen, die lexikographisch zwischen diesem Paar (inklusiv) liegen. Der Bindestrich kann als erstes oder letztes Zeichen in die Menge aufgenommen werden.
Ein Ausrufezeichen ! nach der öffnenden eckigen Klammer [negiert die Zeichenmenge, d.h es werden alle nicht enthaltenen Zeichen angesprochen.

Eine *musterliste* ist eine Liste aus einem oder mehreren Mustern, die voneinander durch den senkrechten Strich | getrennt werden. Zusammengesetzte Muster können aus einem oder mehreren der folgenden Konstrukte geformt werden:

?(musterliste**)**
steht für kein- oder einmaliges Auftreten eines der angegebenen Muster.

***(**musterliste**)**
steht für kein- oder mehrmaliges Auftreten eines der angegebenen Muster.

+(musterliste**)**
steht für mindestens einmaliges Auftreten eines der angegebenen Muster.

@(musterliste**)**
Es muß genau eines der angegebenen Muster passen.

!(musterliste**)**
steht für alles, nur nicht für die angegebenen Muster.

Entwertung von Metazeichen (quoting)

Jedes der oben definierten Metazeichen (siehe *Definitionen*) hat eine spezielle Bedeutung für die Korn-Shell und dient auch als Trenner von Wörtern, falls es nicht entwertet wurde. Ein Zeichen kann durch Voranstellen des Gegenschrägstriches \ entwertet werden, damit es allein für sich steht. Das Paar *Neue-Zeile-Zeichen* wird von der Korn-Shell ignoriert oder gelöscht. Alle Zeichen, die in Hochkommata '...' eingeschlossen sind, sind entwertet. Ein einzelnes Hochkomma kann jedoch nicht darin vorkommen. Zeichenketten in Anführungszeichen "..." eingeschlossen unterliegen der Parameter- und Kommandoersetzung. Durch Gegenschrägstrich können Sie hier Gegenschrägstrich \, Gegenhochkomma `, Anführungszeichen " und Dollarzeichen $ entwerten. Die Bedeutung der Angaben $* und $@ sind gleich, wenn sie nicht in Anführungszeichen eingeschlossen sind oder als Dateiname oder Wert für die Variablenwertzuweisung verwendet werden. Ihre Bedeutung ist unterschiedlich, wenn beide für sich in Anführungszeichen eingeschlossen als Kommandoargument verwendet werden.

"$"* entspricht dann *"$1␣$2␣..."*, wenn ␣ das erste Zeichen im Wert der Variable *IFS* ist, und *"$@"* steht dann für *"$1"␣"$2"␣...*, d.h. die einzelnen Aufrufargumente bleiben erhalten.

Bei in Gegenhochkommata `...` eingeschlossenen Zeichenketten können Sie mit dem Gegenschrägstrich \ den Gegenschrägstrich \, das Gegenhochkomma ` und das Dollarzeichen *$* entwerten. Sollte das Ganze noch in Anführungszeichen "...`...`..." eingeschlossen sein, dann können Sie mit den Gegenschrägstrich \ auch noch das Anführungszeichen " entwerten.

Die spezielle Bedeutung der reservierten Wörter oder Alias-Variablen kann durch Entwertung jedes einzelnen Zeichen anulliert werden. Das Erkennen der Namen von Funktionen und von eingebauten Kommandos kann nicht auf diese Weise unterdrückt werden.

Arithmetische Berechnungen

Durch das eingebaute Kommando *let* steht Ganzzahl-Arithmetik zur Verfügung. Die Berechnungen werden auf der Basis von *Long*-Arithmetik durchgeführt. Konstanten werden in der Form [*basis#*]*n* dargestellt. Dabei ist *basis* eine ganze Zahl zwischen 2 und 36 und gibt die Basis an, und *n* ist eine Zahl zu dieser Basis. Fehlt *basis#*, dann wird im Zehnersystem gerechnet.

Der aritmetischer Ausdruck ist stark an die Programmiersprache C angelehnt. Er benutzt dieselbe Syntax, gleiche Vorrangregeln und Assoziativität. Alle unerläßlichen Operatoren außer + +, --, *?:* und Komma , sind vorhanden. Auf den Wert von Variablen kann über deren Namen zugegriffen werden, Sie müssen kein Dollarzeichen verwenden. Wenn der Wert einer Variablen eingesetzt wird, dann wird ihr Wert als arithmetischer Ausdruck berechnet.

Durch die Option *-i* des eingebauten Kommandos *typeset* kann als Attribut für die interne Darstellung des Wertes einer Variablen die Ganzzahldarstellung gewählt werden. Bei jeder Wertzuweisung auf eine Variable mit dem *-i*-Attribut wird eine aritmetische Berechnung durchgeführt. Wird keine Basis für die Berechnungen angegeben, dann wird die Basis der ersten Wertzuweisung an die Variable verwendet. Diese Basis wird auch bei der Durchführung von Parameterersetzung verwendet.

Da einige der aritmetischen Operatoren für die Korn-Shell entwertet werden müssen, wurde eine alternative Form zum eingebauten Kommando *let* eingeführt. Bei jedem Kommando, das mit doppelter runder Klammer auf *((* beginnt, werden alle Zeichen bis zum schließenden runden Klammerpaar *))* als entwertet genommen. *((a=a+b))* entspricht *let "a=a+b"*.

Bedingte Ausdrücke

Einen bedingten Ausdruck verwenden Sie zum Testen der Eigenschaften von Dateien, für algebraische Vergleiche und zum Vergleich von Zeichenketten. In der Korn-Shell werden die bedingten Ausdrücke innerhalb der Anweisung *[[...]]* angegeben. Die Ersetzung von Blanks und die Erzeugung von Dateinamen werden nicht auf die Wörter des bedingten Ausdrucks angewandt. Jeder bedingter Ausdruck kann aus einem oder mehreren der folgenden unitären oder binären Ausdrücken gebildet werden:

Eigenschaften von Dateien

Ist in den folgenden Ausdrücken *datei* von der Form /dev/fd/*n* (fd - file descriptor, *n* ist eine ganze Zahl), dann wird der Test auf die geöffnete Datei mit Dateikennzahl *n* durchgeführt.

-a datei
(a - access) wahr, wenn *datei* existiert.

-b datei
(b - block device) wahr, wenn *datei* existiert und ein blockorientiertes Gerät ist.

-c datei
(c - character device) wahr, wenn *datei* existiert und ein zeichenorientiertes Gerät ist.

-d datei
(d - directory) wahr, wenn *datei* existiert und ein Dateiverzeichnis ist.

-f datei
(f - file) wahr, wenn *datei* existiert und eine einfache Datei ist.

-g datei
(g - group ID) wahr, wenn *datei* existiert und das set-user-ID-Bit für die Gruppe gesetzt ist.

-k datei
(k - sticky) wahr, wenn *datei* existiert und das sticky- oder t-Bit gesetzt ist.

-o option
(o - option) wahr, wenn die angegebene Option *option* aktiv ist, wobei *option* mit dem vollen Optionsnamen angegeben sein muß, z.B. *errexit*. (*option* kann mit *set* gesetzt werden).

-p datei
(p - pipe) wahr, wenn *datei* existiert und eine benannte Pipe (FIFO) oder eine Pipe ist.

-r datei
(r - read) wahr, wenn *datei* existiert und der aktuelle Prozeß das Leserecht hat.

-s datei
(s - size) wahr, wenn *datei* existiert und nicht leer ist.

-t dateikennzahl
(t - terminal) wahr, wenn *dateikennzahl* geöffnet und einer Datensichtstation zugeordnet ist.

-u datei
(u - user ID) wahr, wenn *datei* existiert und das set-user-ID-Bit für den Eigentümer gesetzt ist.

-w datei
(w - write) wahr, wenn *datei* existiert und der aktuelle Prozeß das Schreibrecht hat.

-x datei
(x - execute) wahr, wenn *datei* existiert und der aktuelle Prozeß das Ausführrecht hat. Existiert *datei* und ist sie ein Dateiverzeichnis, dann muß für wahr der aktuelle Prozeß das Recht zum Durchlaufen haben.

-G datei
(G - group) wahr, wenn *datei* existiert und die Gruppe der Datei der effektiven Gruppennummer des aktuellen Prozesses entspricht.

-L datei
(L - symbolic link) wahr, wenn *datei* existiert und ein symbolischer Link ist.

-O datei
(O - owner) wahr, wenn *datei* existiert und der Eigentümer der Datei der effektiven Benutzernummer des aktuellen Prozesses entspricht.

-S datei
(S - socket) wahr, wenn *datei* existiert und ein Socket ist.

datei1 **-nt** datei2
(nt - newer than) wahr, wenn *datei1* existiert und neuer als *datei2* ist.

datei1 **-ot** datei2
(ot - older than) wahr, wenn *datei1* existiert und älter als *datei2* ist.

datei1 **-ef** datei2
(ef - equal file) wahr, wenn *datei1* und *datei2* existieren und beide ein Verweis auf dieselbe Datei sind.

Eigenschaften und Vergleiche von Zeichenketten

-n zeichenkette
(n - non zero) wahr, wenn die *zeichenkette* existiert und nicht die leere Zeichenkette ist, also eine Länge größer 0 hat.

-z zeichenkette
(z -zero) wahr, wenn die angegebene *zeichenkette* die leere Zeichenkette ist, also die Länge 0 hat.

zeichenkette = muster
wahr, wenn *zeichenkette1* auf *muster* paßt.

zeichenkette != muster
wahr, wenn *zeichenkette1* auf *muster* paßt.

zeichenkette1 < zeichenkette2
wahr, wenn *zeichenkette1* alphabetisch (ASCII-Ordnung) vor *zeichenkette2* liegt.

zeichenkette1 > zeichenkette2
wahr, wenn *zeichenkette1* alphabetisch (ASCII-Ordnung) nach *zeichenkette2* liegt.

Algebraischer Vergleich ganzer Zahlen

zahl1 **-eq** zahl2
(eq - equal) wahr, wenn *zahl1* gleich *zahl2* ist.

zahl1 **-ne** zahl2
(ne - not equal) wahr, wenn *zahl1* ungleich *zahl2* ist.

zahl1 **-lt** zahl2
(lt - less than) wahr, wenn *zahl1* kleiner *zahl2* ist.

zahl1 **-gt** zahl2
(gt - greater than) wahr, wenn *zahl1* größer *zahl2* ist.

zahl1 **-le** zahl2
(le - less than or equal) wahr, wenn *zahl1* kleiner oder gleich *zahl2* ist.

zahl1 **-ge** zahl2
(ge - greater than or equal) wahr, wenn *zahl1* größer oder gleich *zahl2* ist.

Bedingungen verknüpfen oder negieren

Mehrere Bedingungen können Sie miteinander zu einem Ausdruck verknüpfen. Die folgenden Konstrukte sind nach Priorität geordnet, Klammerung hat die höchste, das logische ODER die niedrigste Priorität:

(␣bedingung␣)
bedingung steht hier für (eine oder) mehrere Bedingungen, die beliebig miteinander verknüpft sind. Der Ausdruck ist wahr, wenn *bedingung* wahr ist.

!␣bedingung
Negation: wahr, wenn *bedingung* falsch ist.

bedingung1␣**&&**␣bedingung2
Logisches UND: wahr, wenn *bedingung1* und *bedingung2* wahr sind.

bedingung1␣||␣bedingung2
Logisches ODER: wahr, wenn entweder *bedingung1* oder *bedingung2* wahr ist.

Ein- und Ausgabe eines Kommandos umlenken

Vor der Ausführung eines Kommandos können Sie dessen Ein- und Ausgabe umlenken. Dazu benutzen Sie eine spezielle Notation, die von der Korn-Shell interpretiert wird. Die folgenden Angaben können bei einem einfachen Kommando an beliebiger Stelle oder vor oder nach einem Kommando stehen. Diese Angaben werden nicht an das aufzurufende Kommando übergeben, sondern von der Korn-Shell interpretiert. Parameter- und Kommandoersetzung werden durchgeführt, bevor *datei* oder *dateikennzahl* eingesetzt werden. Die Dateinamen-Erzeugung wird nur dann durchgeführt, wenn das Muster zu genau einem Dateinamen führt. Blank-Ersetzung wird nicht durchgeführt.

<datei
lenkt die Standard-Eingabe (Dateikennzahl 0) des Kommandos auf *datei* um, das Kommando liest seine Eingabe aus *datei*.

>datei
lenkt die Standard-Ausgabe (Dateikennzahl 1) des Kommandos auf *datei* um, das Kommando schreibt seine Ausgabe in *datei*. Existiert die Datei noch nicht, wird sie neu angelegt. Existiert *datei* als einfache Datei bereits und ist die Option *noclobber* (siehe *Eingebaute Kommandos, set -o*) gesetzt, dann gibt dies einen Fehler. Ist *noclobber* nicht gesetzt, dann wird der bisherige Inhalt der Datei gelöscht.

>|datei
Entspricht >datei, mit dem Unterschied, daß die Einstellung der Option *noclobber* ignoriert wird.

>>datei
lenkt die Standard-Ausgabe (Dateikennzahl 1) des Kommandos auf *datei* um, das Kommando schreibt seine Ausgabe in *datei*. Wenn *datei* bereits existiert, wird die Ausgabe an den bisherigen Inhalt angehängt. Sonst wird die Datei neu angelegt.

<>datei
datei wird zum Lesen und Schreiben als Standard-Eingabe geöffnet.

<<[-]zeichenkette
leitet ein Here-Dokument ein. An *zeichenkette* wird weder Parameter- und Kommandoersetzung noch Dateinamen-Erzeugung durchgeführt. Die Eingabe für die Korn-Shell wird bis ausschließlich zu einer Zeile, die nur *zeichenkette* enthält, oder bis zum Dateiende gelesen. Das so gelesene Here-Dokument findet als Standard-Eingabe für das Kommando Verwendung.

Ist eines der Zeichen von *zeichenkette* entwertet, dann sind für die Korn-Shell alle Zeichen des Here-Dokuments entwertet.

Vorsicht

zeichenkette in der schließenden Zeile darf nicht entwertet sein.

Ist *zeichenkette* nicht entwertet, dann

- wird Parameter- und Kommandoersetzung durchgeführt
- wird die Zeichenfolge *Neue-Zeile-Zeichen* ignoriert
- müssen durch Gegenschrägstrich \ der Gegenschrägstrich \, das Dollarzeichen *$*, das Gegenhochkomma ` und der erste Buchstabe von *zeichenkette* entwertet werden, wenn Bedarf dafür im Text besteht.

Durch die Angabe von <<- werden alle führenden Tabulatorzeichen von den Zeilen des Here-Dokuments und vor *zeichenkette* gelöscht.

<&dateikennzahl
>&dateikennzahl
Bei der ersten Form wird die Standard-Eingabe durch Duplizieren von *dateikennzahl* umgelenkt. Die Standard-Eingabe liest aus der Datei, die an die Dateikennzahl angeschlossen ist. Die zweite Form gilt analog für die Standard-Ausgabe.

<&-
>&-
Durch die erste Form wird für das Kommando die Standard-Eingabe geschlossen, das Kommando erhält nur Datei-Ende als Eingabe. Die zweite Form gilt für die Standard-Ausgabe, das Kommando gibt nichts aus.

<&p
>&p
Die Eingabe vom Koprozeß bei einer bidirektionalen Pipeline wird auf die Standard-Eingabe umgelenkt. Analog wird die Ausgabe auf die Standard-Ausgabe umgelenkt.

Wird einer der obigen Anweisungen eine Nummer vorangestellt, dann wird die Dateikennzahl mit dieser Nummer (anstelle der 0 bzw. *stdin* und 1 bzw. *stdout*) angesprochen.

Beispiel

Das folgende Beispiel öffnet Dateikennzahl 2 (*stderr*) zum Schreiben als ein Duplikat von Dateikennzahl 1 (*stdout*):

```
... 2>&1
```

Die Reihenfolge der Angabe der Umlenkungen ist signifikant. Die Korn-Shell bewertet jede Umlenkung bezogen auf die Verbindung (Dateikennzahl, Datei) zum Zeitpunkt der Bewertung. D.h.

```
... 1>datei 2>&1
```

verbindet zuerst die Standard-Ausgabe (Dateikennzahl 1) mit *datei* und verbindet dann die Standard-Fehlerausgabe (Dateikennzahl 2) mit der Datei, die mit Dateikennzahl 1 (also mit *datei*) verbunden ist. *datei* enthält nach Ausführung die Standard-Ausgabe und die Fehlermeldungen des Kommandos.
Wäre die Reihenfolge umgekehrt, dann wäre Dateikennzahl 2 mit der Datensichtstation (falls Dateikennzahl 1 damit verbunden war) und Dateikennzahl 1 mit *datei* verbunden. Das heißt also: *datei* enthält dann nur noch die Standard-Ausgabe, aber nicht die Fehlermeldungen.

Wird ein Kommando mit *&* im Hintergrund gestartet, ohne aktive Auftragssteuerung, dann wird standardmäßig die Standard-Eingabe mit der leeren Datei */dev/null* verbunden. Bei aktiver Auftragssteuerung enthält die Umgebung für die Ausführung des Kommandos die Dateikennzahlen der ausführenden Korn-Shell, wie sie von den Anweisungen für Ein- und Ausgabe geändert wurden.

Umgebung

Die Umgebung eines Prozesses enthält eine Liste aus Paaren *name=wert*, die an ein ausgeführtes Programm in gleicher Weise wie eine normale Argumentliste übergeben wird. Die Namen *name* müssen Bezeichner im Sinne der Korn-Shell sein, die Werte *wert* Zeichenketten (auch die leere). Die Korn-Shell und die Umgebung beeinflußen sich gegenseitig. Beim Aufruf durchsucht die Korn-Shell die Umgebung und erzeugt eine Variable für jeden gefundenen Namen, weist ihr den entsprechenden Wert zu und markiert sie als exportiert. Ausgeführte Kommandos erben diese Umgebung. Modifiziert der Benutzer die Werte dieser Variablen oder fügt neue Variablen mit Hilfe der eingebauten Kommandos *export* oder *typeset -x* dazu, dann werden diese Teil der Umgebung. Die Umgebung, die von jedem ausgeführten Kommando gesehen wird, besteht aus den Paaren *name=wert*, die ursprünglich von der Korn-Shell geerbt wurden, deren Werte von der aktuellen Shell modifiziert worden sein können, plus den Erweiterungen die mit *export* oder *typeset -x* markiert wurden.

Die Umgebung eines einfachen Kommandos oder einer Funktion kann durch Voranstellen von Variablenwertzuweisungen erweitert werden. Eine Variablenwertzuweisung wird als Wort behandelt und hat die Form *bezeichner=wert*.

Die folgenden Zeilen sind für *kommando* äquivalent:

```
TERM=450 kommando argumente
( export TERM ; TERM=450 ; kommando argumente )
```

(Bei manchen eingebauten Kommandos gibt es Abweichungen zu dieser Regel. Im Abschnitt *Eingebaute Kommandos* sind diese mit + bzw. + + markiert.)

Wurde die Option *-k* beim Aufruf der Korn-Shell oder durch das eingebaute Kommando *set* gesetzt, dann werden alle Variablenwertzuweisungen in die Umgebung exportiert, auch wenn sie nach dem Kommandonamen kommen.

Beispiel

Das folgende Beispiel gibt zuerst *a=b c* und dann *c* aus:

```
echo a=b c
set -k
echo a=b c
```

Diese Eigenschaft sollten Sie nur in Verbindung mit Prozeduren, die für ältere Versionen der Bourne-Shell geschrieben wurden, verwenden. Greifen Sie bei neuen Prozeduren nicht darauf zurück, denn die Option *-k* wird in zukünftigen Programmversionen nicht mehr implementiert sein.

Funktionen

Das reservierte Wort *function* (siehe *Zusammengesetzte Kommandos*) wird zur Definition von Korn-Shell-Funktionen benutzt. Funktionen werden von der Shell eingelesen und intern gespeichert. Alias-Namen werden aufgelöst, wenn die Funktion gelesen wird. Funktionen werden wie Kommandos ausgeführt, Argumente werden als Stellungsparameter übergeben (siehe *Ausführung*).

Funktionen werden im aktuellen Prozeß ausgeführt und haben Zugriff auf alle geöffneten Dateien und das aktuelle Verzeichnis.

Signale (traps), die vom Aufrufer abgefangen werden, werden innerhalb der Funktion auf ihre Standard-Aktion zurückgesetzt. Ein Signal, das von der Funktion weder abgefangen noch ignoriert wird, führt zum Abbruch der Funktion. Dieses Signal wird an den Aufrufer der Funktion weitergegeben. Eine *EXIT-trap*-Behandlung, die innerhalb einer Funktion gesetzt wurde, wird nach Beendigung der Funktion in der Umgebung des Aufrufers ausgeführt. Normalerweise werden die Variablen von Aufrufer und Funktion gemeinsam benutzt.

Das eingebaute Kommando *typeset* kann jedoch dazu benutzt werden, innerhalb einer Funktion lokale Variablen zu definieren, deren Gültigkeitsbereich dann die Funktion und alle aufgerufenen Funktionen umfaßt.

Das eingebaute Kommando *return* wird zur Rückkehr von Funktionen verwendet. Fehler, die innerhalb einer Funktion auftreten, geben die Kontrolle an den Aufrufer zurück.

Die Bezeichner oder Namen von Funktionen können Sie mit Hilfe des eingebauten Kommandos *typeset* und einer der Optionen *-f* oder *+f* auflisten, den Text der Funktionen mit der Option *-f* auslisten lassen. Eine Funktion können Sie mit dem eingebaute Kommando *unset* und der Option *-f* löschen.

Normalerweise kann man auf die Funktionen nicht zugreifen, während die Korn-Shell eine Prozedur ausführt.

Durch die Option *-xf* des eingebauten Kommandos *typeset* können Funktionen als exportierbar markiert werden. Diese Funktionen können Sie dann in Prozeduren, die ohne einen weiteren Aufruf der Korn-Shell ausgeführt werden, verwenden. Funktionen, die über mehrere Aufrufe der Korn-Shell bekannt sein sollen, müssen Sie in der *ENV*-Datei mit *typeset -xf* definieren.

Beispiel

Die folgende Funktion *lh* gibt Ihnen die obersten zwei Schichten der Dateiverzeichnis-Hierarchie aus, an der Sie stehen oder die Sie als Argument angegeben haben. Einfache Dateien als Argument werden ignoriert.

```
function lh
{
  for i in ${*:-.}
  do
     if [[ -d $i ]]
     then
        print $i:
        cd $i
        ls -CF $( ls )
        cd - >/dev/null
     fi
  done
}
```

Die *for*-Schleife arbeitet die Argumente einzeln ab. Haben Sie kein Argument angegeben, wird das aktuelle Verzeichnis (.) für *$** gesetzt.
Liegt ein Verzeichnis vor, dann wird dessen Name und Inhalt angezeigt: *cd $i* wechselt zu dem anzuzeigenden Verzeichnis, *$(ls)* wird durch den Inhalt des Verzeichnisses *$i* ersetzt und *ls -CF verzeichnis_inhalt* gibt Ihnen die Dateien, gefolgt von den Inhalten der Verzeichnisse, mit Markierungen für Ausführbarkeit und Verzeichnisse wieder. Das folgende *cd*-Kommandos geht "schweigsam" zum Ausgangsverzeichnis (vor dem ersten *cd $i*) zurück.

Aufträge

Ist die *monitor*-Option des eingebauten Kommandos *set* eingeschaltet, dann verknüpft eine interaktive Korn-Shell mit jeder Pipeline einen Auftrag. Die Shell hält eine Tabelle der aktuellen Aufträge, die Sie mit dem eingebauten Kommando *jobs* auf die Standard-Ausgabe schreiben können. Jedem Auftrag wird ein kleine ganze Zahl zugewiesen.
Wird ein Auftrag im Hintergrund (*&*) gestartet, dann gibt die Shell eine Zeile der folgenden Form aus:

```
[1] 1234
```

Diese Zeile besagt, daß der Auftrag im Hintergrund mit der Auftragsnummer 1 gestartet wurde und einen (Top-Level) Prozeß mit der Prozeßnummer 1234 hat.

Möchten Sie andere Prozesse starten, während ein Prozeß ausgeführt wird, dann brauchen Sie nur die Tastenkombination CTRL Z zu drücken. Diese bewirkt, daß ein STOP-Signal an den laufenden Prozeß gesendet wird. Die Korn-Shell wird dann anzeigen, daß der Auftrag gestoppt (*stopped*) wurde, und ein Bereitzeichen ausgeben. Sie können dann den Zustand dieses Auftrages manipulieren: Den Auftrag mit Hilfe des eingebauten Kommandos *bg* (background) im Hintergrund weiterlaufen lassen, ihn im gestoppten Zustand lassen und andere Kommandos ausführen oder wieder in den Vordergrund durch das eingebaute Kommando *fg* (foreground) holen.

CTRL Z wird sofort behandelt, wenn es gedrückt wird und ist in seiner Auswirkung mit der Unterbrechung CTRL D vergleichbar. Ungelesene Eingaben und noch nicht angezeigte Ausgaben werden weggeworfen.

Ein Hintergrundauftrag wird gestoppt, wenn er versucht, von einer Datensichtstation zu lesen. Hintergrundaufträge dürfen normalerweise Ausgaben produzieren. Sie können dies durch das Kommando *stty tostop* verbieten. Wenn Sie diese *stty*-Option setzen, dann wird der Hintergrundauftrag gestoppt, wenn er versucht, auf die Datensichtstation auszugeben.

Sie haben mehrere Möglichkeiten, auf die Aufträge zuzugreifen. Sie können über die Prozeßnummer eines Prozesses des Auftrages oder mit einem der folgenden Ausdrücke zugreifen:

%nummer
der Auftrag mit der gegebenen Auftrags-*nummer*.

%zeichenkette
jeder Auftrag, dessen Kommando-Zeile mit *zeichenkette* beginnt.

%?zeichenkette
jeder Auftrag, dessen Kommando-Zeile *zeichenkette* enthält.

%%
der aktuelle Auftrag.

%+
synonym für *%%*

%-
der letzte Auftrag.

Die Korn-Shell registriert jeden Zustandswechsel eines Auftrags. Normalerweise informiert sie Sie sofort, wenn ein Auftrag gestoppt wurde und nicht weiter ausgeführt werden kann. Damit Ihre Arbeit nicht gestört wird, wird diese Information vor der Ausgabe eines Bereitzeichens ausgegeben.

Wurde die *monitor*-Option eingeschaltet, dann löst jeder beendete Hintergrundauftrag jeden für *CHLD* gesetzten *trap* aus.

Wenn Sie versuchen, die Korn-Shell zu verlassen, während Sie noch gestoppte Aufträge im Hintergrund haben, dann werden Sie gewarnt:

```
You have stopped (running) jobs
```

Sie können dann das Kommando *jobs* benutzen, um sich eine Übersicht der augenblicklichen Situation zu verschaffen. Wenn Sie dies getan haben, oder unmittelbar nach dem ersten Versuch ein zweites Mal die Korn-Shell verlassen möchten, werden Sie kein zweites Mal gewarnt und die gestoppten Aufträge werden beendet.

Signale

Die Signale INT und QUIT werden für ein aufgerufenes Hintergrund-Kommando (&) ignoriert, wenn die Option *monitor* der Auftragssteuerung nicht aktiv ist. Sonst haben die Signale die Werte, die von der Korn-Shell von ihrem Vaterprozeß geerbt wurden. (Lesen Sie dazu auch beim eingebauten Kommando *trap* nach.)

Ausführung

Jedesmal, wenn ein Kommando ausgeführt wird, werden die bereits beschriebenen Ersetzungen in dieser Reihenfolge durchgeführt:

- Entwertung
- Parameter-Ersetzung
- Tilde-Ersetzung
- Alias-Ersetzung
- Dateinamen-Erzeugung
- Ein- und Ausgabeumlenkung
- Kommandoersetzung

Entspricht der Kommandoname dem Namen eines eingebauten Kommandos, dann wird dieses in der aktuellen Korn-Shell ausgeführt. Als nächstes wird geprüft, ob der Kommandoname dem Namen einer benutzerdefinierten Funktion entspricht. Ist dies der Fall, dann werden die Stellungsparameter gesichert und auf die Werte der Argumente des Funktionsaufrufs gesetzt. Wenn die Funktion beendet ist oder das eingebaute Kommando *return* ausgeführt wurde, werden die Stellungsparameter wiederhergestellt und jeder in der Funktion für *EXIT* gesetzte *trap* ausgeführt. Der Exit-Wert ist der Wert des letzten Kommandos in der Funktion. Eine Funktion wird auch in der aktuellen Korn-Shell ausgeführt. Ist ein Kommandoname nicht unter den eingebauten Kommandos oder den benutzerdefinierten Funktionen zu finden, dann wird ein Prozeß erzeugt und versucht, das Kommando durch den Systemaufruf *exec* ausführen zu lassen.

Die Variable *PATH* definiert den Suchpfad für die Dateiverzeichnisse mit den Kommandos. Die einzelnen Verzeichnisse werden durch Doppelpunkt *:* getrennt. Der Standard-Suchpfad ist */usr/bin:*. Damit sind, in dieser Reihenfolge, das Verzeichnis */usr/bin* und das aktuelle Verzeichnis gemeint. Das aktuelle Verzeichnis kann durch einen Doppelpunkt am Anfang oder Ende oder durch zwei (oder mehr) aufeinanderfolgende Doppelpunkte angegeben werden. Der Suchpfad wird nicht benutzt, wenn der Kommandoname einen Schrägstrich / enthält. Ohne / im Kommandonamen wird jedes Verzeichnis im Suchpfad nach einer ausführbaren Datei durchsucht. Existiert für die Datei das Ausführrecht und ist sie eine einfache Datei ohne das *a.out*-Format, dann wird angenommen, daß sie eine Shell-Prozedur enthält. In diesem Fall wird eine Subshell zum Lesen der Datei aufgerufen. Alle nicht exportierten Alias-Variablen, Funktionen und Umgebungsvariablen werden nicht in die Subshell kopiert. Ein geklammertes Kommando wird in einer Subshell ohne Löschung der nicht exportierten Teile ausgeführt.

Kommando-Wiederaufruf

Der Text der letzten *HISTSIZE* Kommandos, die an einer Datensichtstation eingegeben wurden, wird in einer *History*-Datei gespeichert. Der Standardwert für *HISTSIZE* ist 128. Wenn die Variable *HISTFILE* nicht gesetzt oder die durch den Wert angegebene Datei nicht schreibbar ist, wird die Datei *$HOME/.sh_history* zur Speicherung benutzt. Eine Korn-Shell kann auf alle Kommandos von interaktiven Korn-Shells zugreifen, welche die gleiche *History*-Datei benutzen. Das eingebaute Kommando *fc* können Sie zum Auslisten oder Editieren eines Bereiches dieser Datei benutzt. Der Dateibereich, der angesprochen wird, kann durch eine Zahl, den ersten Buchstaben des Kommandos oder eine Zeichenkette aus dem Kommando angesprochen werden. Der Bereich kann aus einem oder mehreren Kommandos bestehen. Wenn Sie keinen Editor durch ein Argument beim Aufruf von *fc* definieren, dann wird der Wert der Variablen *FCEDIT* verwendet. Ist *FCEDIT* nicht definiert, dann wird */usr/bin/ed* benutzt. Die editierten Kommandos werden ausgegeben und nach Verlassen des Editors wieder ausgeführt.

Der Editorname Bindestrich - (wenn FCEDIT=-) wird zum Überspringen der Editierphase und zur direkten Ausführung benutzt. In diesem Fall kann eine Variable der Form *alt=neu* zur Änderung des Kommandos vor der Ausführung benutzt werden. Ist beispielsweise *r* die Aliasvariable für *fc -e -*, und Sie geben *r unsinn=sinn c* ein, dann wird das letzte Kommando, das mit dem Zeichen *c* begann, ausgeführt. Zuvor wird aber das erste Auftreten von *unsinn* im Kommando durch *sinn* ersetzt.

Option des Zeileneditors

Normalerweise tippen Sie jede Kommandozeile an der Datensichtstation ein und schließen sie durch ein Neue-Zeile-Zeichen ab. Ist die Option *vi* aktiv (siehe *set*), dann können Sie die Kommandozeile editieren.
Diese Editieroption wird automatisch gesetzt, wenn eine der Variablen *EDITOR* oder *VISUAL* ein Wert zugewiesen wird, der auf *vi* endet.

Die Editiermöglichkeit fordert von der Datensichtstation:

- Der Wagenrücklauf (return) kann ohne Zeilenvorschub (line feed) verwendet werden.
- Das Leerzeichen (␣) überschreibt das vorhandene Zeichen am Bildschirm.

Benutzer mit ADM-Terminals sollten den "space-advance"-Schalter auf "space" schalten. Die Schalterstellung eines Hewlett-Packard-Terminals der Serie 2621 sollte "bcGHxZ␣etX" sein.

Der Editiermodus ist so implementiert, daß Sie wie durch ein Fenster auf die aktuelle Zeile schauen. Die Breite des Fensters wird, falls definiert, durch den Wert der Variablen *COLUMNS* bestimmt, sonst ist das Fenster 80 Zeichen breit. Ist die Zeile breiter als die Fensterbreite minus zwei, dann werden Sie durch eine Markierung am Fensterrand darauf aufmerksam gemacht. Bewegen Sie die Schreibmarke an den Rand, wird das Fenster um diese Position zentriert dargestellt.

Als Markierung werden verwendet:

> wenn die Zeile rechts zu lang ist
< wenn die Zeile links zu lang ist
* wenn die Zeile links und rechts zu lang ist

Das Such-Kommando hat in jedem Editiermodus Zugriff auf die *History*-Datei. Zum Suchen werden nur Zeichenketten, keine Muster, verwendet. Die einzige Ausnahme: Der Zirkumflex ^ als erstes Zeichen in der Zeichenkette bindet an das erste Zeichen in der Zeile.

vi-Editiermodus

Es gibt zwei Modi bei der Eingabe:

- Wenn Sie ein Kommando eingeben, befinden Sie sich im Eingabemodus.
- Durch Drücken der Taste ESC wechseln Sie in den Kommandomodus.

Im Kommandomodus können Sie dann editieren, also die Schreibmarke bewegen oder Zeichen löschen und einfügen. Die meisten Anweisungen können, wie beim *vi*, mit Wiederholungsfaktoren versehen werden.

Wenn Sie sich im *vi*-Modus befinden, ist bei den meisten Systemen der Eingabe-Modus die Standard-Einstellung. ESC schaltet den Eingabe-Modus ab, so daß Sie die Kommandozeile ändern können. Dieses Schema verbindet die Vorteile des Eingabe-Modus als Standard-Einstellung mit dem Echo des Raw-Modus beim Voraustippen.

Eingabeanweisungen

Standardmäßig befindet sich der Editor im Eingabemodus.

⌫
(⌫ steht für das mit dem Kommando *stty* definierte *erase*-Zeichen.) löscht das Zeichen vor der Schreibmarke.

CTRL W
löscht das vorherige, durch ein Leerzeichen getrennte Wort.

CTRL D
beendet die Korn-Shell, falls die Option *ignoreeof* (siehe *set*) nicht eingeschaltet ist.

CTRL V
entwertet das nächste Zeichen. Sie können Editier-Zeichen und *erase*- und *kill*-Zeichen in eine Kommandozeile oder in eine Such-Zeichenkette einfügen, wenn Sie davor CTRL V tippen. CTRL V entwertet für den Editor die Bedeutung des nächsten Zeichens.

\
entwertet das nächste *erase*- oder *kill*-Zeichen.

Positionieranweisungen

Diese Anweisungen bewegen die Schreibmarke.

[zahl]**l**
geht ein (*zahl*) Zeichen vorwärts (nach rechts).

[zahl]**w**
geht ein (*zahl*) Wort (alphanumerische Zeichen) vorwärts (nach rechts).

[zahl]**W**
geht an den Anfang des nächsten (*zahl*-ten) Wortes, das nach Blank kommt.

[zahl]**e**
geht an das Ende des (*zahl*-ten) Wortes.

[zahl]**E**
geht an das Ende des durch ein Leerzeichen getrennten (*zahl*-ten) Wortes.

[zahl]**h**
geht ein (*zahl*) Zeichen zurück (nach links).

[zahl]**b**
geht ein (*zahl*) Wort zurück (nach links).

[zahl]**B**
geht an den Anfang des nächsten (*zahl*-ten) Wortes, das nach einem Leerzeichen kommt.

[zahl]**|**
geht auf die Spalte *zahl*.

[zahl]**f**zeichen
Sucht das nächste (*zahl*-te) *zeichen* in der aktuellen Zeile.

[zahl]**F**zeichen
sucht rückwärts das nächste (*zahl*-te) *zeichen* in der aktuellen Zeile.

[zahl]**t**zeichen
entspricht *f* gefolgt von *h*.

[zahl]**T**zeichen
entspricht *F* gefolgt von *l*.

[zahl]**;**
wiederholt (*zahl*-mal) die letzte Einzelzeichen-Suchanweisung mit *f*, *F*, *t* oder *T*.

[zahl]**,**
wiederholt in entgegengesetzter Richtung (*zahl*-mal) die letzte Einzelzeichen-Suchanweisung.

0
geht an den Zeilenanfang

^
geht auf das erste sichtbare (von Leerzeichen verschiedene) Zeichen der Zeile.

$
geht an das Zeilenende

Suchanweisungen

Diese Anweisungen greifen auf Ihre *History*-Datei zu.

[zahl]**k**
holt das letzte Kommando. Durch jedes eingegebene *k* wird ein älteres Kommando geholt.

[zahl]**-**
Entspricht *k*.

[zahl]**j**
holt das nächste Kommando. Durch jedes eingegebene *j* wird ein neueres Kommando geholt.

[zahl]**+**
entspricht *j*.

[zahl]**G**
auf das Kommando mit der Nummer *zahl* wird zugegriffen. Standardmäßig wird auf das zuletzt eingegebene Kommando zugegriffen.

/zeichenkette
sucht rückwärts durch die *History*-Datei nach einem älteren Kommando, das *zeichenkette* enthält. *zeichenkette* wird durch das Neue-Zeile-Zeichen abgeschlossen. Ist Zirkumflex ^ dem ersten Zeichen von *zeichenkette* vorangestellt, dann muß am Anfang der Zeile *zeichenkette* stehen. Ist *zeichenkette* leer, wird die letzte *zeichenkette* verwendet.

?zeichenkette
entspricht / mit Vorwärtssuche.

n
sucht die nächste passende Zeile für die letzte Such-Zeichenkette.

N
sucht in umgekehrter Richtung die nächste passende Zeile für die letzte Such-Zeichenkette. Durchsucht die *History*-Datei nach *zeichenkette* der letzten /-Anweisung.

Anweisungen zur Textmodifikation

Diese Anweisungen ändern die Kommandozeile.

a
geht in den Eingabemodus und fügt den Text nach dem aktuellen Zeichen an.

A
entspricht *$a*, der Text wird ans Zeilenende angefügt.

[zahl]**c**positionierung
c[zahl]positionierung
Ab dem aktuellen Zeichen werden alle Zeichen bis zu dem Zeichen, das durch *positionierung* erreicht werden kann, gelöscht und dann in den Eingabemodus umgeschaltet. Ist *positionierung* *c*, dann wird die ganze Zeile gelöscht und in den Eingabemodus umgeschaltet.

C
Ab dem aktuellen Zeichen wird bis zum Zeilenende gelöscht und in den Eingabemodus gewechselt.

S
entspricht *cc*.

D
Ab dem aktuellen Zeichen wird bis zum Zeilenende gelöscht, entspricht *d$*.

[zahl]**d**positionierung
d[zahl]positionierung
Ab dem aktuellen Zeichen werden alle Zeichen bis zu dem Zeichen, das durch *positionierung* erreicht werden kann, gelöscht. Ist *positionierung* *c*, dann wird die ganze Zeile gelöscht.

i
geht in den Eingabemodus und fügt den Text vor dem aktuellen Zeichen ein.

I
entspricht *0i*, der Text wird am Zeilenanfang eingefügt.

[zahl]**P**
wiederholt die letzte Textänderung (*zahl*-mal) vor dem aktuellen Zeichen.

[zahl]**p**
wiederholt die letzte Textänderung (*zahl*-mal) nach dem aktuellen Zeichen.

R
geht in den Eingabemodus und überschreibt den alten Text ab dem aktuellen Zeichen.

[zahl]**r**zeichen
ersetzt das (die nächsten *zahl*) Zeichen ab dem aktuellen Zeichen durch *zeichen* und bewegt die Schreibmarke auf das letzte geänderte Zeichen.

[zahl]**x**
löscht das aktuelle Zeichen (und *zahl* minus 1 Zeichen danach).

[zahl]**X**
löscht das (*zahl*) Zeichen vor dem aktuellen Zeichen.

[zahl]**.**
wiederholt die letzte Textänderungs-Anweisung (*zahl*-mal).

[zahl]**~**
invertiert die Groß-/Kleinschreibung von *zahl* Zeichen ab dem aktuellen Zeichen und bewegt die Schreibmarke entsprechend *zahl* mit.

[zahl]**_**
fügt das letzte bzw. *zahl*-te Wort der vorhergehenden Anweisung an und geht dann in den Eingabemodus.

*****
Mit dem aktuellen Wort wird durch Anfügen eines Sterns versucht, eine Dateinamen-Erzeugung durchzuführen. Wird keine Datei gefunden, dann ertönt ein akustisches Signal. Ansonsten wird Wort durch den gefundenen Dateinamen ersetzt und in den Eingabemodus gewechselt.

\
Ergänzung von Dateinamen:
ersetzt das aktuelle Wort durch den längsten gemeinsamen Präfix von allen Dateinamen, die durch die Dateinamen-Erzeugung mit dem aktuellen Wort und angefügten Stern hervorgehen. Ist die Ergänzung eindeutig, wird bei einem Verzeichnis ein Schrägstrich /, sonst ein Leerzeichen angefügt.

Sonstige Editieranweisungen

[zahl]**y**positionierung
y[zahl]positionierung
Ab dem aktuellen Zeichen wird der aktuelle Text bis zu dem Zeichen, das durch *positionierung* erreicht werden kann, im Löschpuffer gespeichert. Der Text und die Schreibmarke bleiben unverändert.

Y
Ab dem aktuellen Zeichen wird der aktuelle Text bis zum Zeilenende im Löschpuffer gespeichert. Entspricht *y$*.

u
Die letzte Änderungsanweisung wird rückgängig gemacht.

U
Alle Änderungen an der Kommandozeile werden rückgängig gemacht.

[zahl]**v**
liest aus der *History*-Datei *zahl* Zeilen in den Eingabepuffer mit Hilfe des Kommandos
fc -e ${VISUAL:-${EDITOR:-vi}} zahl
Fehlt *zahl*, dann wird die aktuelle Zeile genommen.

CTRL L
Zeilenvorschub ausgeben und aktuelle Zeile drucken. Wirkt nur im Kommandomodus.

CTRL J
(Neue-Zeile-Zeichen)
Aktuelle Zeile unabhängig vom momentanen Modus ausführen.

CTRL M
(Wagenrücklauf)
Aktuelle Zeile unabhängig vom momentanen Modus ausführen.

#
schließt die Zeile ab, nachdem ihr ein Nummernzeichen # vorangestellt wurde. Damit können Sie eine Zeile in die *History*-Datei einfügen, die nicht ausgeführt wird.

=
zeigt alle Dateinamen, auf die das aktuelle Wort mit angefügtem Stern als Muster paßt.

@buchstabe
durchsucht Ihre Alias-Tabelle nach einer Variablen mit dem Namen *_buchstabe* und nimmt, bei Erfolg, den Wert als Eingabe.

Eingebaute Kommandos

Die folgenden einfachen Kommandos werden in der Korn-Shell selbst ausgeführt. Ein- und Ausgabeumlenkung ist dabei erlaubt. Falls nicht anders angegeben wird die Ausgabe auf die Standard-Ausgabe geschrieben; der Ende-Status ist 0, wenn kein Syntax-Fehler erkannt wurde.

Kommandos, denen in der Beschreibung + oder + + vorangestellt sind, werden, wie nachfolgend beschrieben, anders behandelt:

- Dem Komandonamen vorangestellte Variablenwertzuweisungen bleiben aktiv, wenn das Kommando beendet ist.
- Ein- und Ausgabeumlenkungen werden nach den Variablenwertzuweisungen bearbeitet.
- Fehler in einer Prozedur brechen die Ausführung der Prozedur ab.
- Nur + +: Hat ein Wort das Format der Variablenwertzuweisung *name=wert*, dann wird *wert* wie bei normalen Wertzuweisungen ersetzt.
  D.h. Tilde-Ersetzung wird nach dem Gleichheitszeichen = durchgeführt, Dateinamen-Erzeugung und Worttrennung (durch Blank-Ersetzung) jedoch nicht.

Die Kommandos mit + sind:

Doppelpunkt *:*, Punkt *.*, *break*, *continue*, *eval*, *exec*, *exit*, *newgrp*, *return*, *shift*, *times*, *trap*, *wait*.

Die Kommandos mit + + sind:

alias, *export*, *readonly*, *typeset*.

+:[␣argument]...
Dieses Kommando führt nur Parameterersatz auf *argument* durch.

+.␣datei[␣argument]...
liest die komplette *datei* und führt dann die Kommandos in der Umgebung der aktuellen Korn-Shell aus. Der Suchpfad *PATH* wird zum Suchen des Dateiverzeichnisses mit *datei* benutzt.
Falls Sie *argumente* angegeben, dann werden diese zu den Stellungsparametern, sonst bleiben die alten erhalten. Der Endstatus ist der Ende-Status des letzten ausgeführten Kommandos.

+ +**alias**[␣-t][␣name]
alias ohne Argumente schreibt die Alias-Tabelle in der Form *name*=*wert* auf die Standard-Ausgabe.

-t
wird zum Setzen und Auflisten von mit Pfad versehenen Alias-Variablen (tracked alias) verwendet. Der *wert* einer solchen Variablen besteht aus dem vollen Pfadnamen, der zu *name* führt. Durch Ändern des Wertes von *PATH* wird *wert* undefiniert, *name* bleibt aber mit Pfad. Bei fehlender Option *-t* wird für jeden Namen ohne Wert in der Argumentliste das Paar *name*=*wert* ausgegeben.

+ +**alias**[␣-x][␣name=[wert]]...
Ein Leerzeichen am Ende von *wert* bedeutet, daß bei der Kommandoabarbeitung auch das nächste Wort in der Kommandozeile auf Alias-Ersetzung überprüft wird.

-x
wird zum Setzen und Ausgeben von exportierten Alias-Variablen verwendet. Eine exportierte Alias-Variable ist für Prozeduren definiert, die mit ihrem Namen aufgerufen werden.

Der Ende-Status ist ungleich 0, wenn für *name* kein *wert* definiert ist.

bg[␣job]...
Jeder der angegebenen Aufträge wird in den Hintergrund geschickt. Der aktuelle Auftrag wird in den Hintergrund geschickt, wenn *job* nicht angegeben ist. Der Abschnitt *Aufträge* enthält eine Beschreibung des Formats von *job*.

+**break**[␣zahl]
bricht die umschließende *for-*, *while-*, *until-* oder *select*-Schleife ab. Ist *zahl* angegeben, wird in der *zahl*ten umschließenden Schleife mit der Ausführung fortgefahren. Ist *zahl* gleich 0, wird die äußerste Schleife verlassen. *break 1* entspricht *break*.

+**continue**[␣zahl]
fährt mit der nächsten Iteration der umschließenden *for-*, *while-*, *until-* oder *select*-Schleife fort. Ist *zahl* angegeben, wird in der *zahl*ten umschließenden Schleife mit der nächsten Iteration fortgefahren. Ist *zahl* gleich 0, wird hinter der äußersten Schleife fortgefahren. *continue 1* entspricht *continue*.

cd[␣dvz] Format 1
cd␣alt␣neu Format 2

Format 1: Dateiverzeichnis mit CDPATH wechseln

cd wechselt das aktuelle Verzeichnis nach *dvz*. Haben Sie den Bindestrich - für *dvz* angegeben, wechselt *cd* zum vorhergehenden Verzeichnis zurück. Die Variable *PWD* erhält als Wert das aktuelle Verzeichnis, *OLDPWD* den alten Wert von *PWD* zugewiesen. Die Variable *CDPATH* definiert den Suchpfad für Dateiverzeichnisse, also Namen von Verzeichnissen, von denen eines *dvz* enthält.
Die Namen der einzelnen Verzeichnisse in *CDPATH* werden, wie bei *PATH*, durch Doppelpunkt *:* getrennt. Der Standard-Suchpfad ist leer, dies entspricht dem aktuellen Verzeichnis. Beachten Sie, daß das aktuelle Verzeichnis durch die leere Zeichenkette dargestellt wird und an beliebiger Stelle von *CDPATH*, also auch nach dem Gleichheitszeichen, stehen kann. Beginnt *dvz* mit einem Schrägstrich /, dann wird der Suchpfad nicht benutzt, sonst wird jedes Verzeichnis des Suchpfads in der Reihenfolge der Angabe nach *dvz* durchsucht.

Format 2: Dateiverzeichniswechsel mit Textersetzung

cd ersetzt die Zeichenkette *alt* durch *neu* im Namen des aktuellen Verzeichnisses (*PWD*) und versucht, in dieses neue Verzeichnis zu wechseln.

echo[␣argument]...
Lesen Sie die Beschreibung des Kommandos *echo*.

+eval[␣argument]...
Die Argumente werden als Eingabe der Korn-Shell gelesen und das oder die daraus entstehenden Kommandos ausgeführt. Auf diese Weise werden die Argumente zweimal von der Korn-Shell bewertet.

+exec[␣argument]... Format 1
exec[␣umlenkung]... Format 2

Format 1: Korn-Shell überlagern

Rufen Sie *exec* mit *argument* auf, dann wird das durch die Argumente beschriebene Kommando anstelle der aktuellen Korn-Shell ausgeführt, ohne einen neuen Prozeß zu erzeugen. Ein- und Ausgabenumlenkung können Sie dabei angeben, sie werden von der aktuellen Korn-Shell vorgenommen und gelten dann für das Kommando.

Format 2: Ein- bzw. Ausgabe umlenken

Haben Sie *exec* mit *umlenkung* aufgerufen, dann werden Ein- bzw. Ausgabe umgelenkt, wie es im Abschnitt *Ein- und Ausgabe für ein Kommando umlenken* beschrieben ist. In diesem Fall wird jede durch diesen Aufruf geöffnete Dateikennzahl, die größer als 2 ist, beim Aufruf eines anderen Programms geschlossen.

+**exit**[␣zahl]
Dieses Kommando beendet die Korn-Shell mit dem Ende-Status *zahl*. Haben sie *zahl* nicht angegeben, wird der Ende-Status des letzten ausgeführten Kommandos verwendet. Dateiende ist für die Korn-Shell gleichbedeutend mit *exit*, falls nicht die Option *ignoreeof* (siehe *set*) gesetzt ist.

+ +**export**[␣name=[wert]]...
Die angegebenen Namen werden markiert, so daß sie in eine neue Umgebung exportiert werden. Wenn Sie *export* ohne Argumente aufrufen, werden alle zu exportierenden Variablen auf die Standard-Ausgabe geschrieben.

fc[␣**-e**␣editor][␣**-lnr**][␣erstes][␣letztes] Format 1
fc[␣**-e**␣-][␣alt=neu][␣kommando] Format 2

Format 1: Kommandos aus der History-Datei editieren

Aus den letzten *HISTSIZE* Kommandos, die an der Datensichtstation eingegeben wurden, wird der Bereich von *erstes* bis *letztes* Kommando zum Editieren und anschließenden Ausführen ausgewählt. Die Argumente *erstes* und *letztes* können Sie als Zahl oder als Zeichenkette angeben.
Eine positive Zahl steht für die Nummer des Kommandos. Eine negative Zahl wird als Differenz zur Nummer des aktuellen Kommandos interpretiert.
Eine Zeichenkette wird dazu benutzt, das zuletzt ausgeführte Kommando, das mit dieser Zeichenkette beginnt, auszuwählen. Ist *letztes* nicht angegeben, wird es auf *erstes* gesetzt. Ist *erstes* auch nicht gegeben, wird standardmäßig das letzte Kommando zum Editieren und *-16* zum Auflisten genommen.

-e␣editor
Durch *editor* wird der zu benutzende Editor bezeichnet. Fehlt diese Angabe, wird der Wert der Variablen *FCEDIT* als Editor verwendet.
Ist *FCEDIT* nicht gesetzt, dann wird standardmäßig */usr/bin/ed* eingesetzt.

-l
Die Kommandos werden nur auf die Standard-Ausgabe geschrieben und können nicht editiert und ausgeführt werden.

-n
Die Nummern der Kommandos werden beim Schreiben durch die Option *-l* unterdrückt.

-r
Die Kommandos werden in umgekehrter Reihenfolge zur Verfügung gestellt.

Format 2: Kommando wieder ausführen

Nach der Ersetzung der Zeichenkette *alt* durch *neu* wird das Kommando erneut ausgeführt.

fg[␣job]...
Jeder der angegebenen Aufträge wird in den Vordergrund gebracht. Der aktuelle Auftrag wird in den Vordergrund gebracht, wenn *job* nicht angegeben ist. Der Abschnitt *Aufträge* enthält eine Beschreibung des Formats von *job*.

getopts␣optionszeichenkette␣name[␣argument]...
getopts überprüft *argument* auf legale Optionen. Fehlt *argument*, werden die Stellungsparameter verwendet. Eine Option beginnt grundsätzlich mit einem Plus- oder Minuszeichen. Eine Option, die nicht mit + oder - beginnt, oder das Argument -- schließen die Optionen ab. In *optionszeichenkette* sind die Buchstaben enthalten, die *getopts* als Optionen erkennen soll. Folgt auf einen Buchstaben ein Doppelpunkt *:*, dann wird angenommen, daß die Option ein Argument hat. Die Optionen können von den Argumenten durch Blanks getrennt sein.

Jedesmal, wenn *getopts* aufgerufen wird, weist es den erkannten Options-Buchstaben der Variablen *name* als Wert zu. Dem zugewiesenen Buchstaben wird ein Pluszeichen vorangestellt, wenn die Option mit + begonnen hat. Der Index des nächsten Arguments wird *OPTIND* zugewiesen. Falls ein Options-Argument existiert, wird es *OPTARG* zugewiesen.

Ist das erste Zeichen von *optionszeichenkette* einen Doppelpunkt *:*, dann reagiert *getopts* wie folgt auf Fehler, während es sonst eine Fehlermeldung ausgibt. Der Buchstabe einer falschen Option wird *OPTARG* zugewiesen und dem Wert von *name* ein Fragezeichen *?*, wenn die Option unbekannt ist oder ein Doppelpunkt *:*, wenn eine erforderliche Option fehlt. Der Ende-Status ist ungleich 0, wenn keine weiteren Optionen mehr existieren.

jobs[␣**-lnp**][␣job]...
Dieses Kommando gibt es nur auf Systemen mit Auftragssteuerung. *jobs* schreibt auf die Standard-Ausgabe Informationen zu den angegebenen Aufträgen oder, falls *job* fehlt, zu allen aktiven Aufträgen.

-l
Diese Option gibt zusätzlich noch die Prozeßnummern aus.

-n
Nur die gestoppten und die seit dem letzten Aufruf von *jobs* beendeten Aufträge werden ausgegeben.

-p
Nur die Prozeßgruppe wird ausgegeben.

Der Abschnitt *Aufträge* enthält eine Beschreibung des Formats von *job*.

kill[␣-signal]␣job␣... Format 1
kill␣**-l** Format 2

Format 1: Signale an Prozesse senden

Schickt das Signal *TERM* oder das angegebene Signal *signal* an den durch *job* bezeichneten Prozeß oder Auftrag. *signal* können Sie durch die Signalnummer oder den Signalnamen beschreiben. Eine Liste der Signale finden Sie in der Datei */usr/include/sys/signal.h* - die Namen werden ohne die Vorsilbe *SIG* benutzt. Das Signal *CONT* wird vorausgeschickt, wenn an einen gestoppten Auftrag oder Prozeß die Sinale *TERM* oder *HUP* geschickt werden. Das Argument *job* kann die Prozeßnummer eines Prozesses sein, der nicht Element eines aktiven Auftrags ist.

Format 2: Signalnummern und -namen ausgeben

kill -l schreibt zeilenweise alle Signalnummern gefolgt von deren Namen auf die Standard-Ausgabe.

Der Abschnitt *Aufträge* enthält eine Beschreibung des Formats von *job*.

let␣ausdruck␣...
Jedes Argument ist ein arithmetischer Ausdruck. Die berechneten Ergebnisse werden ausgegeben. Der Ende-Status ist 0, wenn der Wert des letzten Ausdrucks ungleich 0 war, und 1 sonst.

+newgrp[␣argument]...
Entspricht *exec␣/usr/bin/newgrp␣argument␣...*

print[␣**-Rnprsu**[dateikennzahl]][␣argument]...
print ist der Ausgabemechanismus der Korn-Shell. Ohne Option oder mit einer der Optionen - oder -- schreibt *print* auf die Standard-Ausgabe, wie es beim Kommando *echo* beschrieben ist.

-R oder **-r**
Im Raw-Modus werden alle Steuerzeichen von *echo* ignoriert. Die Option *-R* schreibt alle folgenden Optionen und Argumente mit Ausnahme der Option *-n*.

-n
An die Ausgabe wird kein Neue-Zeile-Zeichen angefügt.

-p
Statt auf die Standard-Ausgabe werden die Argumente auf die Pipeline zu dem durch |& erzeugten Prozeß geschrieben.

-s
Die Argumente werden in die *History*-Datei und nicht auf die Standard-Ausgabe geschrieben.

-udateikennzahl
Die einstellige *dateikennzahl* wird anstelle der Standard-Ausgabe zum Schreiben der Argumente verwendet.

pwd
Entspricht **print␣-r␣-␣$PWD**

read[␣**-prsu**[dateikennzahl]][␣name**?**abfrage][␣name]...
read ist der Eingabemechanismus der Korn-Shell. *read* liest eine Zeile von der Standard-Eingabe, zerlegt sie in Felder und weist das erste Feld dem ersten *name*n, das zweite Feld dem zweiten *name*n, usw. und den Rest dem letzten *name*n zu. Fehlt *name*, dann wird *REPLY* standardmäßig als *name* verwendet.
Enthält das erste Argument ein Fragezeichen *?*, wird bei interaktiver Korn-Shell der Rest des Wortes als ein Bereitzeichen auf die Standard-Fehlerausgabe geschrieben.
Der Ende-Status ist 0, wenn nicht auf Dateiende gestoßen wurde.

-r
Im Raw-Modus hat der Gegenschrägstrich \ am Zeilenende keine Sonderfunktion, d.h. die Zeile wird nicht fortgesetzt.

-p
Statt von der Standard-Eingabe wird von der Pipeline zu dem, durch |*&* erzeugten Prozeß gelesen. Bei Dateiende von der Pipeline wird so bereinigt, daß durch |*&* ein neuer Prozeß erzeugt werden kann.

-s
Die Eingabe wird als Kommando in die *History*-Datei geschrieben.

-u
Die einstellige *dateikennzahl* wird statt der Standard-Eingabe zum Lesen verwendet. Die Dateikennzahl kann mit dem eingebauten Kommando *exec* geöffnet werden. Der Standardwert für *dateikennzahl* ist 0.

+ +readonly[␣name=wert][␣name]...
Die angegebenen Variablen *name* werden als nur lesbar markiert. Die Werte dieser Variablen können durch weitere Wertzuweisungen nicht mehr verändert werden.

+return[␣zahl]
In einer Funktion aufgerufen, kehrt dieses Kommando zur aufrufenden Prozedur zurück. Der *return*-Status wird durch *zahl* oder das letzte ausgeführte Kommando bestimmt. Wird *return* außerhalb von Funktionen oder in einer mit Punkt ausgeführten Prozedur aufgerufen, dann entspricht dies dem Aufruf von *exit*.

set[␣option][␣argument]...
Mit *set* können Sie drei Dinge tun:

- Stellungsparameter setzen,
- Variablen auf die Standard-Ausgabe schreiben oder
- Korn-Shell Optionen setzen.

Den Stellungsparametern werden durch die Argumente *argument* neue Werte zugewiesen, falls nicht die Option *-A* benutzt wird. Das erste Argument geht an *$1*, das zweite an *$2*, usw.

Sind weder *option* noch *argument* angegeben, dann schreibt die Korn-Shell die Namen und Werte aller Variablen auf die Standard-Ausgabe.

Die Optionen der Korn-Shell können Sie während der Sitzung ändern. Durch ein Minuszeichen - vor dem Buchstaben werden die Optionen eingeschaltet, durch ein Pluszeichen + ausgeschaltet. Die aktuell gesetzten Optionen finden Sie in der Variable *$-*. Alle Optionen können Sie auch beim Aufruf der Korn-Shell verwenden. Es stehen folgende Optionen zur Verfügung:

-A␣name
Wertzuweisung für Felder:
Löscht den Wert der Variablen *name* und weist ihr dann sequentiell aus der *argument*-Liste Werte zu. Verwenden Sie das Pluszeichen *+A*, dann werden die Werte vor der Zuweisung nicht gelöscht.

-a
Ab sofort werden alle neu definierten Variablen automatisch exportiert.

-e
Wenn ein Kommando einen Ende-Status ungleich 0 hat, wird die Signalbehandlung für *ERR* ausgeführt (falls sie definiert ist) und die Korn-Shell beendet.

-f
Die Dateinamen-Erzeugung wird unwirksam gemacht.

-h
Zu jedem Kommando, das ausgeführt wird, wird in der Alias-Tabelle eine mit Pfad versehene Alias-Variable (tracked alias) angelegt.

-k
Alle Argumente mit Variablenwertzuweisungen werden in die Umgebung eines Kommandos aufgenommen, nicht nur die vor dem Kommandonamen angegebenen.

-m
Hintergrundkommandos werden in einer eigenen Prozeßgruppe ausgeführt. Die Beendigung wird in einer Zeile berichtet. Der Ende-Status von Hintergrundaufträgen wird durch eine Beendigungsmeldung quittiert. Bei einer interaktiven Korn-Shell wird diese Option automatisch eingeschaltet.

-n
Liest die Kommandos und überpruft sie auf Syntax-Fehler, führt sie jedoch nicht aus. Von einer interaktiven Korn-Shell wird diese Option ignoriert.

-o␣argument
argument kann einer der folgenden Optionsnamen sein:

allexport
entspricht *-a*

errexit
entspricht *-e*

bgnice
Alle Hintergrundaufträge werden mit niedrigerer Priorität ausgeführt. Dies ist die standardmäßige Voreinstellung.

ignoreeof
Die Korn-Shell wird nicht durch Dateiende beendet. Benutzen Sie dazu das eingebaute Kommando.

keywords
entspricht *-k*

markdirs
Alle Dateiverzeichnisse, die bei der Dateinamen-Erzeugung entstehen, erhalten eine Schrägstrich / am Ende angefügt.

monitor
entspricht *-m*

noclobber
Bei der Umlenkung der Ausgabe mit > werden keine existierenden Dateien überschrieben.

noexec
entspricht *-n*

noglob
entspricht *-f*

nolog
Funktionsdefinitionen werden nicht in der *History*-Datei gespeichert.

nounset
entspricht *-u*

privileged
entspricht *-p*

verbose
entspricht *-v*

trackall
entspricht *-h*

vi
Versetzt Sie in den Eingabemodus eines *vi*-ähnlichen Zeilen-Editors bis Sie ESC drücken. Sie befinden sich dann im Kommandomodus. Ein Neue-Zeile-Zeichen sendet die Zeile.

viraw
Mit eingeschaltetem *vi*-Modus wird jedes Zeichen bearbeitet, sobald es getippt wird.

xtrace
entspricht *-x*

-p
Wenn die effektive Benutzernummer (Gruppennummer) ungleich der realen Benutzernummer (Gruppennummer) ist, wird die Bearbeitung der Datei *$HOME/.profile* weggelassen und anstelle der *ENV*-Datei die Datei */etc/suid_profile* verwendet.
Schalten Sie diese Option durch das Pluszeichen (*+p*) aus, werden die effektive Benutzer- und Gruppennummer auf die reale Benutzer- und Gruppennummer gesetzt.

-s
Sortiert die Stellungsparameter lexikographisch. Vorsicht: Diese Option entspricht nicht der Option *-s*, die beim Aufruf der Korn-Shell verwendet werden kann.

-t
Beendet die Korn-Shell nach dem Lesen und Ausführen eines Kommandos.

-u
Behandelt nicht gesetzte Parameter als Fehler bei der Ersetzung.

-v
Gibt die Eingabezeile so aus, wie sie die Korn-Shell liest.

-x
Gibt die Kommandos und ihre Argumente so aus, wie sie ausgeführt werden.

-

Schaltet die Optionen *-v* und *-x* aus und nimmt den Rest der Kommandozeile als *argument...* - es wird nicht nach weiteren Optionen gesucht.

--

Keine der Optionen wird geändert. Benutzen Sie diese Option, wenn Sie an *$1* einen Wert, der mit Bindestrich beginnt, zuweisen wollen. Folgt auf diese Option kein Argument, werden die Werte der Stellungsparameter gelöscht.

+shift[␣zahl]

Die Stellungsparameter *$zahl+1*, ... werden in *$1*, ... unbenannt, d.h. die Werte der Stellungsparameter werden nach links verschoben. Bei fehlender *zahl* wird standardmäßig 1 eingesetzt. *$0* bleibt bei dieser Aktion erhalten. Das Argument *zahl* kann ein arithmetischer Ausdruck sein, der als Ergebnis eine positive ganze Zahl kleiner gleich *$#* ergibt.

+times

Druckt die aufaddierten Benutzer- und System-Zeiten von der aktuellen Korn-Shell und den daraus gestarteten Prozessen aus.

+trap[␣kommando][␣signal]...

kommando kann eine Liste von Kommandos enthalten, die von der Korn-Shell gelesen werden und beim Empfang eines der Signale *signal* ausgeführt werden sollen. (Vorsicht: *kommando* wird zweimal interpretiert - das erste Mal, wenn die Korn-Shell *trap* ausführt und das zweite Mal, wenn *kommando* für ein Signal ausgeführt wird.) *signal* können Sie durch die Signalnummer oder den Signalnamen beschreiben (siehe *kill*). *trap*-Kommandos werden in der Reihenfolge der Signalnummern ausgeführt.

trap-Kommandos werden abhängig von den Signalnummern ausgeführt. Jeder Versuch, eine Signalbehandlung für ein Signal zu setzen, das beim Aufruf der aktuellen Korn-Shell auf ignorieren gesetzt war, bleibt ohne Wirkung. Die Signalbehandlung(en) für *signal* werden auf ihre Standardwerte gesetzt, wenn Sie *kommando* weglassen, oder den Bindestrich - dafür angeben. Wenn Sie die leere Zeichenkette ("") als *kommando* für *signal* angeben, wird das Signal von der Korn-Shell und von allen aus ihr aufgerufenen Prozessen ignoriert. Steht *ERR* für *signal*, wird *kommando* immer dann ausgeführt, wenn ein Kommando einen Ende-Status ungleich 0 hat. Geben Sie *DEBUG* für *signal* an, wird *kommando* nach jedem Kommando ausgeführt. Ist *signal 0* oder *EXIT* und wurde das *trap*-Kommando innerhalb einer Funktion aufgerufen, wird *kommando* nach Beenden der Funktion ausgeführt. Wurde analog *trap* außerhalb einer Funktion aufgerufen, wird *kommando* beim Beenden der Korn-Shell ausgeführt.

Ein *trap*-Kommando, ohne *kommando* und *signal* aufgerufen, schreibt die Liste der Signalnummern mit den dazuzugehörigen Kommandos auf die Standard-Ausgabe.

+ +typeset[␣option][␣name[=wert]]...
Mit *typeset* können Sie zwei Dinge tun:

- Variablen mit Zusatzinformation auf die Standard-Ausgabe schreiben
- Attribute und Werte von Variablen setzen.

Sind weder *option* noch *name*[=*wert*] angegeben, dann schreibt die Korn-Shell die Namen und Attribute aller Variablen auf die Standard-Ausgabe.
Mit Option, aber ohne *name* werden alle Variablen (und deren Attribute) aufgelistet, auf welche die Option zutrifft.
Benutzen Sie das Pluszeichen + anstelle des Bindestrichs -, werden die Werte der Variablen nicht gedruckt.

Durch *option* können Sie der Variablen *name*[=*wert*] ein oder mehrere Attribute geben. Rufen Sie *typeset* innerhalb einer Funktion auf, wird eine neue Instanz der Variablen *name* angelegt. Attribut und Wert werden nach Verlassen der Funktion wieder hergestellt.
Durch ein Minuszeichen - vor dem Buchstaben werden die Attribute eingeschaltet, durch ein Pluszeichen + ausgeschaltet.

Die folgende Liste von Attributen können Sie als Optionen angeben:

-H
Diese Option bietet auf Nicht-UNIX-Rechnern die Möglichkeit, Dateinamen so umzuwandeln, daß sie entsprechend der UNIX-Konventionen den Rechnernamen enthalten.

-L[␣zahl]
Der Wert wird linksbündig abgelegt und führende Blanks werden gelöscht.
zahl ungleich 0 definiert die Länge des Feldes, sonst wird die Länge des Feldes durch die Länge der ersten Wertzuweisung bestimmt.
Diese Länge ist im folgenden bestimmend. Wird der Variablen ein neuer Wert zugewiesen, dann wird entweder bei zu langer Zuweisung rechts abgeschnitten oder mit Leerzeichen aufgefüllt.
Führende Nullen werden ebenfalls gelöscht, wenn die Option *-Z* gesetzt ist. Die Option *-R* wird von *-L* ausgeschaltet.

-R[␣zahl]
Rechtsbündige Ablage des Werts, führende Blanks werden am Anfang von *wert* eingefügt.
zahl ungleich 0 definiert die Länge des Feldes, sonst wird die Länge des Feldes durch die Länge der ersten Wertzuweisung bestimmt. Diese Länge ist im folgenden bestimmend.
Wird der Variable ein neuer Wert zugewiesen, dann wird entweder bei zu langer Zuweisung rechts abgeschnitten oder links mit Leerzeichen aufgefüllt.
Die Option *-L* wird von *-R* ausgeschaltet.

-Z[␣zahl]
Rechtsbündige Ablage des Werts, führende Nullen werden am Anfang von *wert* eingefügt, wenn das erste Zeichen ungleich Blank eine Ziffer ist und die Option *-L* nicht eingeschaltet ist. *zahl* ungleich 0 definiert die Länge des Feldes, sonst wird die Länge des Feldes durch die Länge der ersten Wertzuweisung bestimmt.

-f
Die Namen gehören zu Funktionen und nicht zu Variablen. Die Korn-Shell legt die Funktionen in der Datei *.sh_history* ab. Sie können deshalb die Definition einer Funktion am Bildschirm nicht ausgeben lassen, wenn die Datei *.sh_history* nicht vorhanden ist, oder wenn die Option *nolog* beim Lesen der Funktion gesetzt war. Sie können keine Wertzuweisungen vornehmen und nur die folgenden Optionen sind in Verbindung mit *-f* zulässig:

-t
Die Ausführungsüberwachung wird für die Funktion eingeschaltet.

-u
Die Funktion wird als nichtdefiniert markiert. Die Variable *FPATH* wird zum Suchen der Funktion verwendet, wenn die Funktion das nächste Mal aufgerufen wird.

-x
Die Funktionsdefinition bleibt über namentliche Aufrufe von Korn-Shell-Prozeduren erhalten, d.h. die Funktion wird exportiert.

-i[␣zahl]
Die Variable ist eine ganze Zahl (integer). Diese Option beschleunigt die Berechnung. *zahl* ungleich 0 definiert die Basis der Ausgabe, sonst bestimmt die erste Wertzuweisung die Basis.

-l
Alle Großbuchstaben werden in Kleinbuchstaben umgewandelt. Die Option *-u* wird ausgeschaltet.

-r
Die angegebenen Variablen werden als nur lesbar (readonly) markiert. Die Werte dieser Variablen können nicht mehr durch Zuweisungen verändert werden.

-t
Markiert die Variable: Marken sind benutzerspezifisch und haben für die Korn-Shell keine spezielle Bedeutung.

-u
Alle Kleinbuchstaben werden in Großbuchstaben umgewandelt. Die Option *-l* wird ausgeschaltet.

-x
Die angegebenen Namen werden so markiert, daß sie automatisch in neue Umgebungen exportiert werden.

ulimit[␣option] Format 1
ulimit[␣option]␣grenzwert Format 2

Mit *ulimit* können Sie die Grenzwerte Ihrer Prozesse abfragen und setzen. Die Grenzwerte sind in *getrlimit* beschrieben.

Format 1: Grenzwerte abfragen

ulimit schreibt die durch *option* (siehe unten) abgefragten Grenzwerte auf die Standard-Ausgabe. Sie können die Optionen beliebig kombinieren. Wollen Sie alle Grenzwerte gleichzeitig sehen, verwenden Sie die Option *-a*.

Format 2: Grenzwerte setzen

ulimit setzt den durch Option bezeichnten Grenzwert auf *grenzwert*. Die Zeichenkette *unlimited* fordert den maximalen Grenzwert. Sie können mit jedem Aufruf immer nur einen Grenzwert neu setzen. Jeder Benutzer kann einen "weichen" Grenzwert (soft limit) auf einen Wert kleiner dem "harten" Grenzwert (hard limit) setzen. Als Benutzer ohne Systemverwalter-Privilegien können Sie jeden harten Grenzwert zwar erniedrigen, aber nur der Systemverwalter kann einen harten Grenzwert auch erhöhen (siehe *su*).

Durch die Option *-H* sprechen Sie einen harten Grenzwert, durch die Option *-S* einen weichen Grenzwert an. Ist keine dieser beiden Optionen angegeben, dann setzt *ulimit* beide Grenzwerte und zeigt den weichen Wert an.

Durch folgende Optionen können Sie die abzufragenden oder zu setzenden Grenzwerte angeben. Ist keine Option angegeben, wird *-f* verwendet.

-c
maximale Größe einer *core*-Datei (in 512 Byte-Blöcken)

-d
maximale Größe eines Datensegments oder *Heap* (in Kbytes)

-f
maximale Dateigröße (in 512 Byte-Blöcken)

-n
maximale Anzahl von Dateikennzahlen plus 1

-s
maximale Größe des Stacksegments (in Kbytes)

-t
maximal verbrauchbare CPU-Zeit (in Sekunden)

-v
maximale Größe des virtuellen Speichers (in Kbytes)

umask[␣maske]
Mit *umask* können Sie die aktuelle Schutzbit-Maske abfragen oder neu setzen. Fehlt *maske*, wird die aktuelle Schutzbit-Maske auf die Standard-Ausgabe geschrieben. Ist *maske* angegeben, wird die Schutzbit-Maske gesetzt (siehe *umask()* [14]). *maske* kann entweder als Oktalwert angegeben oder symbolisch umschrieben werden (siehe *chmod*). Bei einer symbolischen Angabe berechnet sich die neue Schutzbit-Maske aus dem Komplement der Anwendung von *maske* auf das Komplement der aktuellen Schutzbit-Maske.

unalias␣name␣...
Die durch *name*␣... gegebenen Variablen werden aus der Alias-Tabelle gelöscht.

unset[␣**f**]␣name␣...
Die Werte und Attribute der durch *name*␣... gegebenen Variablen werden gelöscht. Die Werte und Attribute von als nur lesbar markierten Variablen können nicht gelöscht werden. *unset* der Variablen *ERRNO*, *LINENO*, *MAILCHECK*, *OPTARG*, *OPTIND*, *RANDOM*, *SECONDS*, *TMOUT* und _ entfernt die Spezialfunktion dieser Variablen, auch wenn ihnen später wieder Werte zugewiesen werden. Durch die Option *-f* werden die Namen von Funktionen angesprochen.

+wait[␣job]
Wartet auf den Auftrag und berichtet seinen Ende-Status. Ist *job* nicht angegeben, wird auf alle momentan aktiven Sohnprozesse gewartet. Der Ende-Status von *wait* ist der des Prozesses, auf den gewartet wurde. Der Abschnitt *Aufträge* enthält eine Beschreibung des Formats von *job*.

whence[␣**-pv**]␣name␣...
Für jeden Namen wird angezeigt, wie er als Kommandoname interpretiert werden würde.

-v
gibt eine ausführlichere Liste aus.

-p
durchsucht den Suchpfad auch dann, wenn *name* eine Alias-Variable, eine Funktion oder ein reserviertes Wort ist.

ENDE-STATUS

Normalerweise gibt die Korn-Shell den Ende-Status des letzten ausfgeführten Kommandos zurück (siehe *exit*).
Von der Korn-Shell erkannte Fehler, wie z.B. Syntaxfehler, führen zu einem Ende-Status ungleich 0. Wird die Korn-Shell nicht-interaktiv benutzt, dann wird die Bearbeitung der Prozedur-Datei abgebrochen. Von der Korn-Shell erkannte Laufzeit-Fehler werden durch Ausgabe des Kommandonamens und der Fehlerbedingung berichtet. Ist die Zeilennummer der Zeile mit dem fehlerhaften Kommando größer eins, dann wird nach dem Kommando- oder Funktionsnamen noch die Zeilennummer in eckigen Klammern *[...]* angegeben.

DATEIEN

/etc/passwd
/etc/profile
/etc/suid_profile
$HOME/.profile
*/tmp/sh**
/dev/null

SIEHE AUCH

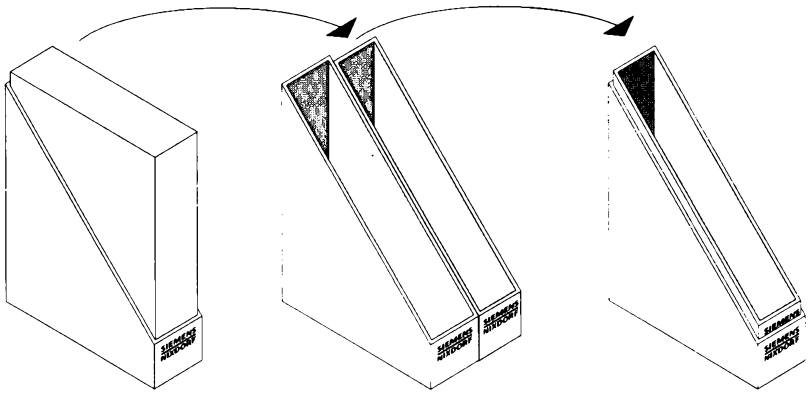
cat, cd, chmod, cut, echo, env, paste, stty, test, umask, vi

dup(), exec(), fork(), getrlimit(), ioctl(), lseek(), pipe(), signal(), umask(), ulimit(), wait(), rand(), a.out [19]
newgrp, profile, environ [7]

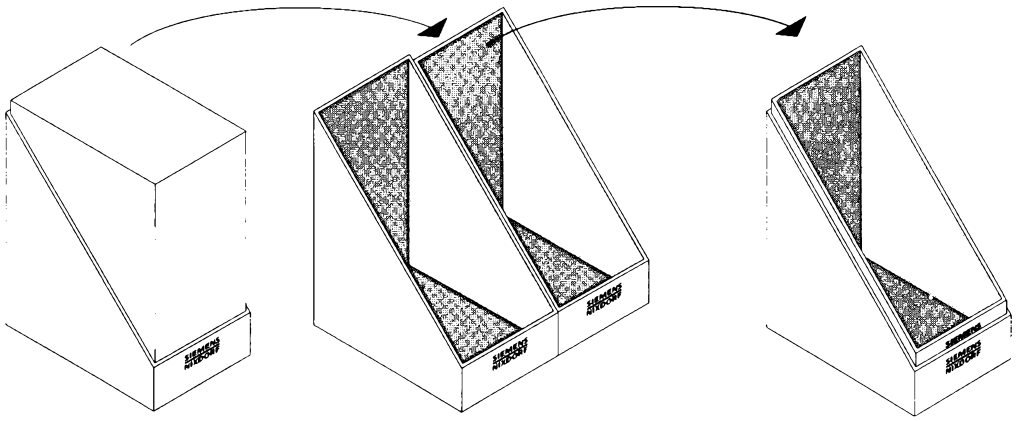
M.I.Bolski und D.G. Korn, *The Korn-Shell and Programming Language* [33]

Sammelboxen

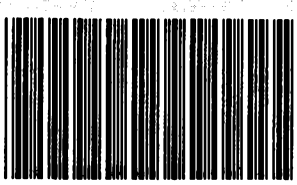
Für Handbücher des vorliegenden Formates bieten wir zweiteilige Sammelboxen in zweierlei Größen an. Der Bestellvorgang entspricht dem für Handbücher.



Breite: ca. 5 cm
Bestellnummer: U3775-J-Z18-1



Breite: ca. 10 cm
Bestellnummer: U3776-J-Z18-1



9Y501094

A 033/92

Herausgegeben von/Published by
Siemens Nixdorf Informationssysteme AG
Postfach 2160, W-4790 Paderborn
Postfach 83 09 51, W-8000 München 83

Bestell-Nr./Order No. ***U8480-J-Z145-1***
Printed in the Federal Republic of Germany
11840 AG 11915. (14800)