# Betriebssystem SINIX
# Kommandos

# Beschreibung
# Teil 2, M–Z

Ausgabe Mai 1988 (SINIX V5.2)

Bestell-Nr. U3903-J-Z95-1
Printed in the Federal Republic of Germany
6150 AG 5881. (7690)



02151

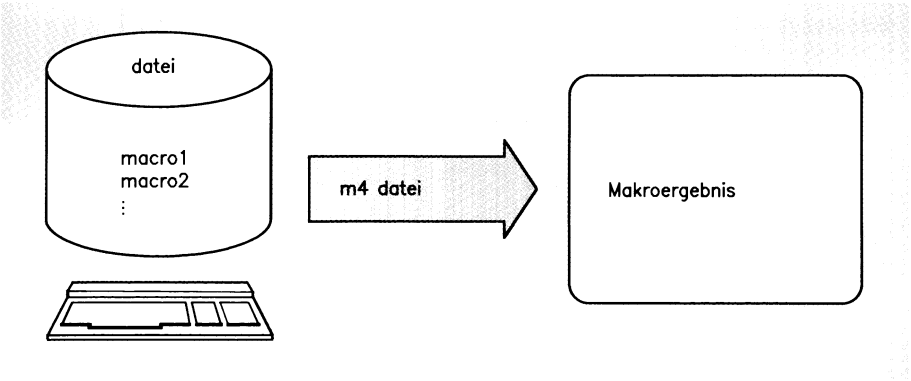
# Inhalt

Seite

**Teil 2**

## NAME

**m4** - Makroprozessor




## DEFINITION

**m4**[␣schalter][␣datei...]

## BESCHREIBUNG

*m4* ist ein Makroprozessor, der als "Front end" für C und andere Sprachen gedacht ist. *m4* schreibt sein Ergebnis auf Standard-Ausgabe.

## SCHALTER

**-e**

Interaktiver Modus. Unterbrechungssignale werden ignoriert, und die Ausgabe wird nicht zwischengespeichert.

**-s**

Zeilensynchrone Ausgabe für den C-Präprozessor aktivieren (z.B. #line-Anweisungen).

**-B**int

Die Größe des Puffers, in den gelöschte Daten und eingegebene Argumente geschrieben werden, soll vom Standard abweichen.

**-H**int

Größe der Hash-Tabellen ändern. Die Größe sollte in Form einer Primzahl angegeben werden.

**-S**int
Die Größe des Call-Stacks (Aufruf-Kellers) ändern. Makros benötigen drei Positionen, andere Argumente eine Position.

**-T**int
Die Größe des Token-Puffers ändern.

Die bisher genannten Schalter müssen vor den Dateinamen und vor den folgenden Schaltern stehen.

**-D**name[=wert]
*name* wird *wert* zugewiesen; fehlt *wert*, so wird ihm eine leere Zeichenkette zugewiesen.

**-U**name
Die Wertzuweisung an *name* wird aufgehoben.

## MAKROS

Makroaufrufe haben folgendes Format:

```
name(arg1,␣arg2,␣...,␣argn)
```

Die öffnende runde Klammer ( muß unmittelbar auf den Namen des Makros folgen.
Folgt dem Namen eines definierten Makros keine (, dann wird er als Aufruf dieses Makros ohne Argumente interpretiert.
Ein Makroname kann aus Buchstaben, Ziffern und Unterstreichungszeichen bestehen; das erste Zeichen darf jedoch keine Zahl sein.

Führende, nicht apostophierte Leer-, Tabulator- und Neue-Zeile-Zeichen werden bei der Eingabe von Argumenten ignoriert. Mit öffnenden und schließenden Hochkommas können Zeichenketten apostrophiert werden. Die apostrophierte Zeichenkette hat den Wert der Zeichenfolge ohne die Apostrophe.

Die Argumente für einen Makro müssen durch eine schließende Klammer abgeschlossen werden. Werden weniger Argumente angegeben, als in der Makrodefinition enthalten sind, so werden die letzten Argumente als leere Zeichenfolgen interpretiert. Die Auswertung des Makros wird während der Sammlung der Argumente wie normal fortgesetzt; öffnende und schließende Klammern im Wert eines geschachtelten Makro-Aufrufs haben dieselbe Wirkung wie Klammern im ursprünglichen Eingabetext. Nach der Sammlung der Argumente wird der Makro wieder in den Eingabestrom zurückgesetzt und nochmals ausgewertet.

Bei *m4* stehen die folgenden eingebauten Makros zur Verfügung. Sie können umdefiniert werden; ihre ursprüngliche Bedeutung geht dabei jedoch verloren. Sie haben, sofern nichts anderes angegeben wurde, den Wert Null.

**define**
Das zweite Argument wird als Wert des Makros genommen; das erste Argument ist der Name des Makros. Jedes *$n* in der Ersetzungszeichenfolge wird durch das *n*te Argument ersetzt; *n* ist dabei eine Zahl. Argument 0 ist der Name des Makros; fehlende Argumente werden durch leere Zeichenketten ersetzt. $# wird durch die Anzahl der Argumente ersetzt; $* wird durch die Liste aller durch Kommas voneinander getrennten Argumente ersetzt; $@ entspricht $*, nur wird jedes Argument apostrophiert.

**undefine**
Die Definition des Makros, die in seinem Argument angegeben wird, wird aufgehoben.

**defn**
Die apostrophierte Definition der Argumente wird ausgegeben. Dieser Makro ist zur Umbenennung von Makros hilfreich, besonders wenn es sich dabei um eingebaute Makros handelt.

**pushdef**
Entspricht *define*, nur werden die ursprünglichen Definitionen gesichert.

**popdef**
Die aktuelle Definition der Argument(e) des Makros wird gelöscht; die alte Definition (falls vorhanden) wird wieder gültig.

**ifdef**
Ist das erste Argument definiert, so erhält der Makro den Wert des zweiten Argumentes, andernfalls den Wert des dritten Argumentes. Gibt es kein drittes Argument, so hat er den Wert Null.

**shift**
Alle Argumente außer dem ersten werden ausgegeben. Die übrigen Argumente werden apostrophiert und, durch Kommas voneinander getrennt, in den Eingabestrom zurückgegeben. Durch die Apostrophierung wird die Wirkung der zusätzlichen Analyse, die nachfolgend durchgeführt wird, aufgehoben.

**changequote**
Die Apostrophzeichen für das erste und zweite Argument werden geändert. Die Apostrophzeichen dürfen aus bis zu fünf Zeichen bestehen. Ohne Argumente stellt *changequote* die ursprünglichen Werte wieder her (d.h. ` ').

**changecom**
Die Kommentarzeichen (Standard: # links für Kommentaranfang und Neue-Zeile-Zeichen für Kommentarende) werden geändert. Werden keine Argumente angegeben, so können nachfolgend keine Kommentare angegeben werden. Wird ein Argument angegeben, so wird dieses Argument als einleitendes Kommentarzeichen, das Neue Zeile-Zeichen als abschließendes Kommentarzeichen verwendet. Werden zwei Argumente angegeben, so werden beide Kommentarzeichen entsprechend geändert. Kommentarzeichen dürfen aus bis zu fünf Zeichen bestehen.

**divert**
*m4* verwaltet 10 Ausgabeströme, denen die Nummern 0 bis 9 zugewiesen sind. Die letztendliche Ausgabe ist die Verknüpfung der Ströme in numerischer Reihenfolge. Zu Beginn ist der Strom 0 der aktuelle Strom. Der *divert*-Makro ändert den aktuellen Ausgabestrom in den durch sein Argument (eine Zahl) angegebenen Ausgabestrom. Wenn eine Zahl angegeben wird, die nicht im Bereich 0 bis 9 liegt, geht die Ausgabe verloren.

**undivert**
*undivert* führt zu sofortiger Ausgabe des Textes in den durch Argumente (Zahl von 0 bis 9) angegebenen Ausgabeströmen oder, wenn keine Argumente angegeben wurden, in allen Ausgabeströmen. Der Text kann in einen anderen Ausgabestrom umgeleitet werden. Neues Umleiten hebt die alte Umleitung auf.

**divnum**
Der Wert des aktuellen Ausgabestroms wird ausgegeben.

**dnl**
Die Zeichen bis zum nächsten Neue Zeile-Zeichen (einschließlich) werden eingelesen und gelöscht.

**ifelse**
Dieser Makro hat drei oder mehr Argumente. Ist das erste Argument mit dem zweiten identisch, so hat der Makro den Wert des dritten Arguments. Wenn dies nicht der Fall ist und mehr als vier Argu-

mente angegeben werden, so wird dies mit den Argumenten 4, 5, 6 und 7 wiederholt. Andernfalls hat der Makro den Wert der vierten Zeichenkette bzw., wenn es keine vierte Zeichenkette gibt, den Wert der leeren Zeichenfolge.

**incr**
Dieser Makro liefert den um 1 erhöhten Wert seines Arguments. Der Wert des Arguments wird berechnet, indem eine ursprüngliche Zeichenkette aus Ziffern als Dezimalzahl interpretiert wird.

**decr**
Dieser Makro liefert den um 1 erniedrigten Wert seines Arguments.

**eval**
Dieser Makro interpretiert sein Argument als arithmetischen Ausdruck, wobei eine 32-Bit-Arithmetik verwendet wird. Als Operatoren sind +, -, *, /, %, ^ (Exponentialrechnung), bitweises &, |, ^, und ~, Vergleichsoperatoren sowie Klammern zulässig. Oktale und hexadezimale Zahlen können wie in C angegeben werden. Das zweite Argument gibt die Basis für das Ergebnis an (Standard: 10). Mit dem dritten Argument kann angegeben werden, aus wievielen Ziffern das Ergebnis mindestens bestehen soll.

**len**
Die Zahl der Zeichen im Argument.

**index**
Dieser Makro gibt in seinem ersten Argument an, an welcher Stelle das zweites Argument beginnt (die äußerst linke Position hat die Nummer 0); gibt es kein zweites Argument, so liefert der Makro den Wert -1.

**substr**
Dieser Makro liefert eine Teilzeichenkette seines ersten Argumentes. Das zweite Argument ist eine Zahl größer Null, mit der das erste Zeichen ausgewählt wird; das dritte steht für die Länge der Teilzeichenfolge. Das dritte Argument muß so gewählt werden, daß es bis zum Ende der ersten Zeichenkette reicht.

**translit**
Dieser Makro wandelt die Zeichen in seinem ersten Argument, die im durch das zweite Argument angegebenen Zeichensatz enthalten sind, in Zeichen aus dem Zeichensatz um, der durch das dritte Argument angegeben wird. Es sind keine Abkürzungen zulässig.

**include**
Dieser Makro liefert den Inhalt der Datei, deren Name als sein Argument angegeben ist.

**sinclude**
Dieser Makro ist identisch mit *include*, nur werden keine Meldungen ausgegeben, wenn auf die Datei nicht zugegriffen werden kann.

**syscmd**
Das im ersten Argument angegebene Systemkommando wird ausgeführt. Es wird kein Wert zurückgegeben.

**sysval**
Der Ende-Status des letzten Aufrufs von *syscmd*.

**maketemp**
Die Zeichenkette *XXXXX* im Argument dieses Makros wird durch die aktuelle Prozeßnummer ersetzt.

**m4exit**
*m4* wird sofort verlassen. Als Ende-Status wird das erste Argument ausgegeben; fehlt dieses Argument, so wird der Wert 0 (Standard) ausgegeben.

**m4wrap**
Das Argument 1 wird bei einem EOF auf einen Keller zurückgeschrieben.

*Beispiel*

```
m4wrap(`cleanup()')
```

**errprint**
Das Argument dieses Makros wird in die Fehlerausgabe-Datei geschrieben.

**dumpdef**
Die aktuellen Namen und Spezifikationen für die angegebenen Argumente werden ausgegeben; werden keine Argumente angegeben, so werden alle Namen und Spezifikationen ausgegeben.

**traceon**
Werden keine Argumente angegeben, so wird für die Durchführung aller Makros (auch für die eingebauten Makros) ein Protokoll ausgegeben. Andernfalls wird nur für die angegebenen Makros ein Protokoll ausgegeben.

**traceoff**
Die Protokollierung wird allgemein und für alle angegebenen Makros unterdrückt. Das Protokoll von Makros, deren Protokollierung explizit mit *traceon* eingeschaltet worden ist, kann nur durch explizites Aufrufen von *traceoff* unterdrückt werden.

## OPERANDEN

datei...
Name der Datei(en), die abgearbeitet werden.
*Standard (keine Angabe): m4* liest von Standard-Eingabe.

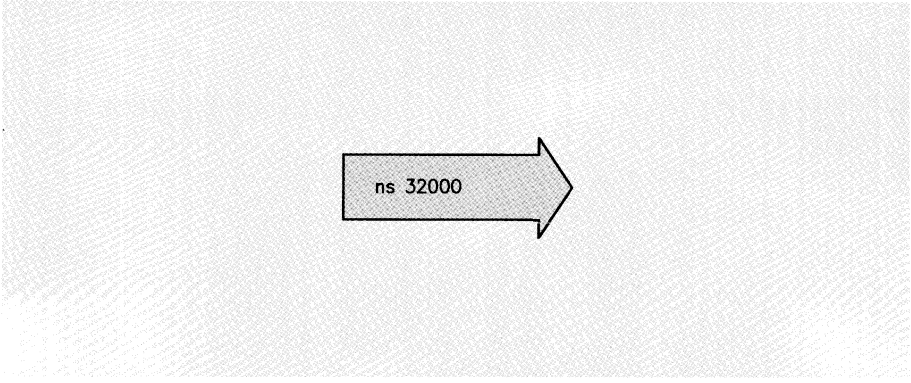
-
*m4* liest von Standard-Eingabe.

## SIEHE AUCH

*cc*(1D), *cpp*(1D)

## NAME

**ns32000, pdp11, u370, u3b, u3b10, u3b2, u3b5, vax**
- Wahrheitswerte über Art des Prozessors ausgeben




## DEFINITION

```
ns32000
pdp11
u370
u3b
u3b10
u3b2
u3b5
vax
```

## BESCHREIBUNG

Das Kommando *ns32000* liefert den Wert wahr (Ende-Status 0). Die übrigen Kommandos liefern den Wert falsch (ungleich Null). Diese Kommandos werden häufig in *make*(1D) Beschreibungsdateien und Shell-Prozeduren verwendet, um die Portabilität zu erhöhen.

## BEISPIEL

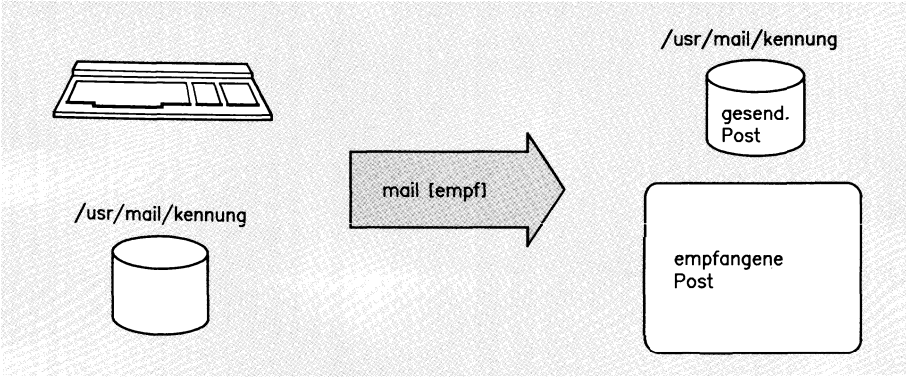
```
$ pdp11
$ echo $?
255

$ ns32000
$ echo $?
0
```

## SIEHE AUCH

*make*(1D), *sh*(1), *test*(1), *true*(1)

## NAME

**mail** - Post senden und empfangen (send or receive mail)




## DEFINITION

**mail**[␣**-t**]␣name...
**mail**[␣schalter][␣**-f**␣datei]

## BESCHREIBUNG

Mit dem Kommando *mail* können Sie:

- Nachrichten an andere Benutzer senden, die am gleichen oder an einem anderen mit Ihrem Rechner vernetzten Rechner arbeiten.

  Hierfür verwenden Sie das 1.Format: **mail**[␣**-t**]␣name...

- Nachrichten, die Sie erhalten haben, aus Ihrem Briefkasten holen, lesen und weiterleiten. Ihr Briefkasten ist die Datei */usr/mail/login-name*.

  Hierfür verwenden Sie das 2. Format: **mail**[␣schalter][␣**-f**␣datei]

Benutzer, die Post erhalten haben, werden nach dem Login mit der Meldung *you have mail* (Sie haben Post) davon informiert.

## SCHALTER

**1. Format: mail**[␣**-t**]␣name...

kein Schalter
*mail* liest von Standard-Eingabe die Nachricht, die Sie versenden möchten und setzt sie in den Briefkasten des Benutzers *name*.
Die Eingabe der Nachricht beenden Sie entweder mit einem . (Punkt) in der ersten Spalte oder mit der Taste END.
Automatisch wird jeder Nachricht der Name des Absenders, Datum und Uhrzeit der Absendung und der Name des Adressaten vorangestellt. Nachrichtenzeilen, die mit "From" beginnen, wird das Zeichen > vorangestellt.

**-t**
Die Namen aller Adressaten werden der Nachricht vorangestellt.

**2. Format: mail**[␣schalter][␣**-f**␣datei]

kein Schalter
*mail* gibt Ihre Post aus. Nach dem Prinzip *last in first out* wird die neueste Nachricht zuerst ausgegeben. *mail* holt die Nachrichten aus Ihrem Briefkasten */usr/mail/login-name* und gibt sie nacheinander auf Standard-Ausgabe aus.
Im Anschluß an jede ausgegebene Nachricht gibt *mail* als Eingabeaufforderung ein ? (Fragezeichen) aus und wartet auf Ihre Antwort, um herauszufinden, was mit der soeben ausgegebenen Nachricht geschehen soll.
Die einzelnen Anworten schließen Sie mit der Taste ↵ ab. Folgende Antworten können Sie geben:

| Antwort | Bedeutung |
|---|---|
| ↵ | Nächste Nachricht ausgeben. |
| + | Nächste Nachricht ausgeben. |
| **d** | Nachricht löschen und nächste Nachricht ausgeben. |
| **p** | Die zuletzt ausgegebene Nachricht wiederholen. |
| - | Die vorhergehende Nachricht wiederholen. |

**s**[datei] Nachricht mit Kopfzeile in *datei* schreiben. Standard für *datei*: mbox. Im Briefkasten wird die Nachricht gelöscht.

**w**[datei] Nachricht ohne Kopfzeile in *datei* schreiben. Standard für *datei*: mbox. Im Briefkasten wird die Nachricht gelöscht.

**m**[name] Nachricht an den Benutzer *name* weitersenden. *name* ist ein oder mehrere Login-Namen. Standard für *name*: eigener Login-Name.

Wenn Sie Nachrichten, die an Sie gerichtet sind, automatisch auch an bestimmte andere Benutzer weiterleiten möchten, müssen Sie in Ihrem HOME-Dateiverzeichnis eine Datei mit dem Namen *.forward* eröffnen. In diese Datei schreiben Sie jeweils in eine Zeile einen Login-Namen eines Benutzers, an den die Nachrichten weitergeleitet werden sollen. Falls die Datei */$HOME/.forward* existiert, muß darin Ihr eigener Login-Name aufgeführt sein, wenn Sie selbst Ihre eigene Post empfangen wollen. Die LoginNamen von Benutzern, die über LAN (Local Area Network) angeschlossen sind, müssen Sie in folgender Form angeben:

`login-name[@hostname]`

*hostname* ist der Rechnername, den der jeweilige Benutzer mit */etc/sysname* abfragen kann.

Wenn Sie für *@hostname* nichts angeben, wird Ihr eigener Rechnername eingesetzt.

[!] Die Möglichkeit, Nachrichten weiterzuleiten, ist besonders hilfreich, wenn alle Nachrichten eines Benutzers in einem Rechnerverbund an einen bestimmten Rechner geschickt werden sollen.

**q** Unbearbeitete Nachrichten im Briefkasten aufbewahren und *mail* beenden.

| | |
|---|---|
| **x** | Alle Nachrichten im Briefkasten aufbewahren und *mail* beenden. Auch die mit *d* gelöschten Nachrichten bleiben erhalten. |
| !kommando | Die Shell vorübergehend aufrufen und *kommando* ausführen. |
| * | Liste der *mail*-Kommandos ausgeben. |

**-e**
Nachrichten werden nicht ausgegeben. Es wird lediglich ein Endestatus-Wert zurückgeliefert.

**-p**
Alle Nachrichten werden ohne ? ausgegeben.

**-q**
*mail* soll nach einem Unterbrechungssignal beendet werden. Standardmäßig wird beim Empfang eines Unterbrechungssignals nur die Ausgabe der aktuellen Nachricht abgebrochen.

**-r**
Die ältesten Nachrichten werden zuerst ausgegeben.

**-f**␣datei
*mail* sichert die Nachrichten in *datei*, nicht im Briefkasten. Standard für *datei*: mbox

## OPERANDEN

**1. Format: mail**[␣**-t**]␣name...

name
*name* ist der Login-Name eines Benutzers, an den die Nachricht geschickt werden soll. Wenn Sie einen unzulässigen Namen eingeben oder *mail* während der Eingabe ein Unterbrechungssignal empfängt, wird die Nachricht in der Datei *dead.letter* abgesichert, um eine Korrektur und ein nochmaliges Aussenden der Nachricht zu ermöglichen. Die Datei *dead.letter* wird im aktuellen Dateiverzeichnis angelegt und nur bei Bedarf eröffnet. Der alte Inhalt dieser Datei wird, falls vorhanden, dabei zerstört.

Mit *mail* können Sie auch Benutzer an einem fernen Rechner erreichen. Dem Login-Namen des Benutzers muß dann der Systemname und ein Ausrufungszeichen vorangestellt werden. Alle Zeichen nach

dem ersten Ausrufungszeichen werden durch das entfernt stehende System interpretiert. Wenn die Nachricht nicht direkt von einem Rechner zum anderen gesendet werden kann, sondern über mehrere Rechner laufen muß, so muß in *name* eine entsprechende Anzahl von Ausrufungszeichen enthalten sein. Geben Sie für *name* z.B. *a!b!cde* an, so wird die Nachricht an den Benutzer *b!cde* in System *a* geschickt. System *a* interpretiert diese Angabe als Aufforderung, die Nachricht an den Benutzer *cde* im System *b* zu schicken. Dies kann hilfreich sein, wenn das Ausgangssystem zwar zu System *a*, nicht aber zu System *b* Zugang hat, und System *a* Zugang zu System *b* hat. Zulässig ist auch die Angabe: benutzer@hostname

! An ferne Rechner können selbstverständlich nur dann Nachrichten übermittelt werden, wenn die geeigneten Übertragungswege vorhanden sind.

**2. Format: mail**[␣schalter][␣**-f**␣datei]

datei
Name der Datei, in der *mail -f* die Nachrichten sichert.
Standard: mbox

**ENDESTATUS**

Ist der Schalter *-e* gesetzt, so liefert *mail* folgende Ende-Status-Werte:

0 Benutzer hat Post
1 Benutzer hat keine Post

**DATEIEN**

*/etc/passwd*
Kennwort-Datei
*$HOME/mbox*
Datei mit gespeicherten Nachrichten
*/usr/mail/login-name*
Briefkasten
*dead.letter*
Datei im aktuellen Dateiverzeichnis, mit geretteten Nachrichten
*$HOME/.forward*
Datei mit den Login-Namen der Benutzer, an die empfangene Nachrichten weitergeleitet werden sollen.

**BEISPIEL**

1. Herr Kurz (login-name: *kurz*) schickt an Herrn Lang (login-name: *lang*), der am gleichen Rechner arbeitet, eine Nachricht:

```
$ mail lang
Ihr Bericht über die Tagung ist zu ausführlich. Bitte
überarbeiten und gekürzte Neufassung an mich !
.
$
```

2. Herr Lang, der nach dem Login am Bildschirm die Meldung *you have mail* erhält, liest seine Post

```
$ mail
From kurz Wed Dec 30 10:55:16 1987
Date: Wed, 30 Dec 87 10:55:13  +0100
From: <kurz>
Apparently-To: lang

Ihr Bericht über die Tagung ist zu ausführlich. Bitte
überarbeiten und gekürzte Neufassung an mich !

?
```

   und löscht sie dann:

```
?
d ↵
$
```

3. Die Herren Wichtig und Tüchtig (Login-Namen: *wichtig, tüchtig*) bitten Herrn Kurz, daß er sie in Zukunft auch von den Nachrichten informiert, die er erhält. Sie arbeiten am Rechner namens *Brummbox*. Herr Kurz schreibt hierfür in die Datei */usr/kurz/.forward*:

```
kurz
wichtig@Brummbox
tüchtig@Brummbox
```

4. Herr Lang hat die 2. Fassung des Berichts fertiggestellt. Sie steht in der Datei *ber.2*, die er an Herrn Kurz schickt:

```
$ mail kurz < ber.2
$
```

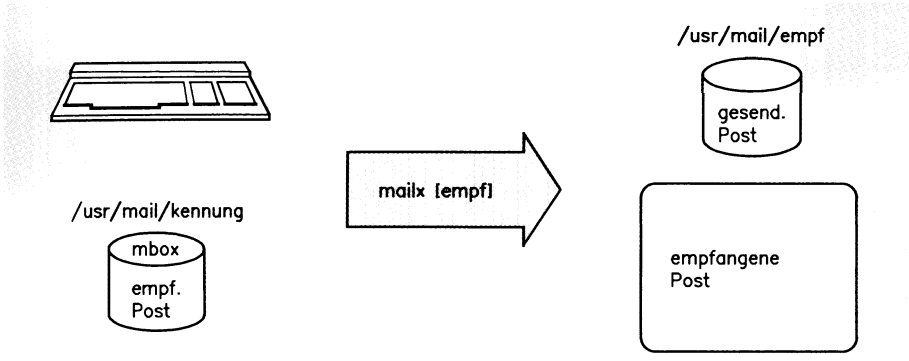
Den Bericht erhalten die Herren Kurz, Wichtig und Tüchtig. Er wird jeweils im Briefkasten des Empfängers */usr/mail/login-name* abgelegt und kann mit *mail* gelesen werden.

## SIEHE AUCH

*mailx*(1), *rmail*(1)

## NAME

**mailx** - Nachrichten interaktiv verarbeiten




## DEFINITION

**mailx**[␣schalter][␣name...]

## BESCHREIBUNG

*mailx* bietet eine komfortable, flexible Möglichkeit zum elektronischen Senden und Empfangen von Nachrichten. Für den Empfang von Nachrichten bietet *mailx* Erleichterungen für das Sichern bzw. Löschen und Beantworten von Nachrichten. Beim Senden von Nachrichten bietet *mailx* die Möglichkeit, die eingegebenen Nachrichten nochmals zu überprüfen und sie gegebenenfalls zu korrigieren.

Die ankommenden Nachrichten werden für jeden Benutzer in einer StandardDatei gespeichert, die als der *briefkasten* dieses Benutzers bezeichnet wird. Sollen mit *mailx* Nachrichten gelesen werden, so wird standardmäßig dieser Briefkasten durchsucht. Die eingelesenen Nachrichten werden in einer zweiten Datei aufbewahrt, sofern nichts anderes angegeben wird. Diese zweite Datei hat den Namen *mbox* und befindet sich normalerweise im *HOME*-Dateiverzeichnis des Benutzers (nähere Informationen zu dieser Datei sind unter *MBOX* im Abschnitt *Umgebungsvariablen* enthalten). Die Nachrichten bleiben in dieser Datei, bis sie explizit aus ihr entfernt werden.

Beim Lesen von Nachrichten befindet sich *mailx* im Kommandomodus. Die Kopfzeilen der ersten Nachrichten werden ausgegeben, gefolgt von einer Eingabeaufforderung; *mailx* kann dann normale

Kommandos empfangen (siehe *Kommandos*, unten). Beim Senden von Nachrichten befindet *mailx* sich im Eingabemodus. Wird in der Kommandozeile kein *subject* angegeben, so wird die Eingabeaufforderung *Subject:* angezeigt. Die eingegebene Nachricht wird von *mailx* eingelesen und in einer Zwischendatei gespeichert. Im Eingabemodus werden Kommandos eingegeben, indem an den Anfang der Zeile das Escape-Zeichen Tilde (~) eingegeben wird, gefolgt vom Anfangsbuchstaben eines Kommandos und von (optionalen) Argumenten. Eine Auflistung dieser Kommandos im Eingabemodus ist im Abschnitt *Tilde-Kommandos* enthalten.

Das Verhalten von *mailx* hängt von einer Reihe von *Umgebungsvariablen* ab.
Dabei handelt es sich um Optionen und Parameter, denen Werte zugewiesen sind; die Umgebungsvariablen werden über die Kommandos *set* und *unset* gesetzt und aufgehoben. Eine Auflistung dieser Parameter ist im Abschnitt *Umgebungsvariablen* enthalten.

## SCHALTER

**-e**

Es wird nachgesehen, ob Post da ist. *mailx* liefert den Ende-Status zurück (0, wenn Post da ist) und wird dann beendet, ohne daß gegebenenfalls Nachrichten ausgegeben worden sind.

**-f**[␣dateiname]

Die Nachrichten sollen aus *dateiname*, nicht aus dem *briefkasten* gelesen werden. Wird kein Dateiname angegeben, so wird der *briefkasten* verwendet.

**-F**

Die verschickte Nachricht wird in einer Datei mitprotokolliert, die nach dem ersten Empfänger benannt ist. Die Variable *record* wird, falls sie gesetzt ist, übergangen (siehe *Umgebungsvariablen*).

**-h**n

Die Zahl der bisher durchgeführten "Sprünge" von einer Station zur anderen innerhalb des Netzwerks. Hiermit soll vermieden werden, daß das Netzwerk-Verteilungsprogramm in einer unendlichen Schleife steckenbleibt.

**-H**
Es werden nur die Kopfzeilen ausgegeben.

**-i**
Der Empfang des Signals SIGINT wird ignoriert; siehe auch *ignore* (*Umgebungsvariablen*).

**-n**
Zur Initialisierung der Parameter sollen nicht die Kommandos in der Datei *mailx.rc* verwendet werden.

**-N**
Ausgabe der Kopfzeilen direkt nach Kommandoaufruf unterdrükken.

**-r**␣adresse
*adresse* soll an das Netzwerk-Verteilungsprogramm übergeben werden. Alle Tilde-Kommandos werden aufgehoben.

**-s**␣titel
Als *subject* soll in das Kopffeld *titel* eingesetzt werden.

**-u**␣benutzer
Der *briefkasten* des *benutzer*s wird gelesen (der *briefkasten* darf bei diesem Schalter nicht lesegeschützt sein).

## OPERANDEN

name...
Namen der Nachrichtenempfänger

keine Angabe
*mailx* versucht, Nachrichten aus dem *briefkasten* zu lesen.

## KOMMANDOS

Reguläre Kommandos haben folgendes Format:

[ `kommando` ] [ `nachrichtenliste` ] [ `argumente` ]

Wird im Kommandomodus kein Kommando eingegeben, so wird automatisch das Kommando *print* ausgeführt. Im Eingabemodus sind Kommandos durch das Escape-Zeichen Tilde ( ~ ) gekennzeichnet; die Zeilen, die nicht als Kommandos interpretiert werden, werden als Teil der Nachricht interpretiert.

Jeder Nachricht wird eine Nummer zugeordnet; es gibt zu jeder Zeit eine "aktuelle Nachricht", die in den angezeigten Kopfzeilen durch ein > gekennzeichnet.
Bei vielen Kommandos kann eine Nachrichten-Liste (*nachrichten*) angegeben werden, die abgearbeitet werden soll (Standard: die aktuelle Nachricht). *nachrichten* enthält durch Leerzeichen voneinander getrennte Spezifikationen für Nachrichten. *nachrichtenliste* ist eine Liste von durch Leerzeichen getrennten Nachrichtennummern, die folgendermaßen angegeben werden können:

*n*
Nummer der Nachricht *n*.

.
Die aktuelle Nachricht.

^
Die erste nicht gelöschte Nachrichten.

$
Die letzte Nachricht.

*
Alle Nachrichten.

*n-m*
Alle Nachrichten mit den Nummern *n* bis *m* (einschließlich).

*benutzer*
Alle Nachrichten von *benutzer*

*/zeichenkette*
Alle Nachrichten mit *zeichenkette* in *subject* (es wird nicht zwischen Groß- und Kleinbuchstaben unterschieden).

:c
Alle Nachrichten vom Typ *c*; *c* kann dabei sein:

| | |
|---|---|
| **d** | gelöschte Nachrichten |
| **n** | neue Nachrichten |
| **o** | alte Nachrichten |
| **r** | bereits gelesene Nachrichten |
| **u** | noch nicht gelesene Nachrichten |

Ob eine Spezifikation sinnvoll ist, hängt vom Kontext ab, in dem das Kommando aufgerufen wird.

Bei den übrigen Argumenten handelt es sich in der Regel um beliebige Zeichenketten, deren Zweck vom jeweiligen Kommando abhängt. Sind Dateinamen zulässig, so können diese Metazeichen enthalten, die für den Kommandointerpreter eine spezielle Bedeutung haben. Bei einigen Kommandos sind Sonderzeichen zulässig; sie werden nachfolgend bei den jeweiligen Kommandos angegeben.

Wird *mailx* aufgerufen, so durchsucht es eine systemweite Start-Datei nach Kommandos, mit denen bestimmte Parameter initialisiert werden; dann durchsucht es eine "private" Datei (*$HOME/.mailrc*), um benutzerspezifische Variablen zu initialisieren. In diesen Dateien sind die meisten regulären Kommandos zulässig; mit den meisten dieser Kommandos werden Ausgabeattribute gesetzt und Alias-Listen erstellt. Folgende Kommandos dürfen in den Dateien nicht enthalten sein: *!*, *Copy*, *edit*, *followup*, *Followup*, *hold*, *mail*, *preserve*, *reply*, *Reply*, *shell* und *visual*. Fehler in der Startdatei bewirken, daß die übrigen Zeilen der Datei ignoriert werden.

### mailx-Kommandos in alphabetischer Reihenfolge

Im folgenden werden die *mailx* Kommandos vollständig aufgeführt. Die meisten der Kommandonamen können abgekürzt werden; die Abkürzung steht unter der vollständigen Form des jeweiligen Kommandos.

**!** Shell-Kommando
Aufruf der Shell (siehe *Umgebungsvariablen* der Shell)

**#** Kommentar
Kein Kommando, sondern Kommentar. Ist für *.mailrc-Dateien* nützlich.

**=**
Die Nummer der aktuellen Nachricht wird ausgegeben.

**?**
Ein Überblick über alle *mailx*-Kommandos wird ausgegeben.

**alias**␣alias␣name...
**a**␣alias␣name ...

**group**␣alias␣name ...
**g**␣alias␣name ...
Für die angegebenen *namen* wird ein *alias* definiert. Wird ein *alias* als Empfänger angegeben, so werden dafür die Namen eingesetzt. Dieses Kommando wird oft in *.mailrc* Dateien verwendet.

**alternates**[␣name ...]
**alt**[␣name ...]
Für den Login-Namen des Benutzers werden eine Reihe von Alternativen angegeben. Antwortet ein Benutzer auf eine Nachricht, so werden diese Namen aus der Liste der Empfänger der Antwort entfernt. Werden keine Argumente angegeben, so gibt *alt*ernates die aktuell gültigen Alternativen aus. Siehe auch *allnet* (*Umgebungsvariablen*).

**cd**[␣dateiverzeichnis]
**chdir**[␣dateiverzeichnis]
Dateiverzeichnis wechseln. Wird kein *dateiverzeichnis* angegeben, so wird standardmäßig das *$HOME*-Verzeichnis verwendet.

**copy**[␣dateiname]
**c**[␣dateiname]

**copy**[␣liste]␣name
**c**[␣liste]␣name
Nachrichten werden in die Datei *name* geschrieben, ohne daß sie als gesichert gekennzeichnet werden. Ansonsten hat es dieselbe Wirkung wie das *s*ave-Kommando.

**Copy**[␣nachrichten]
**C**[␣nachrichten]
Die angegebenen Nachrichten werden in einer Datei gespeichert; deren Name vom Namen des Autors der jeweiligen Nachricht abgeleitet ist. Die Nachrichten werden dabei nicht als gesichert gekennzeichnet. Ansonsten hat es dieselbe Wirkung wie *S*ave.

**delete**[␣nachrichten]
**d**[␣nachrichten]
Aus dem *briefkasten* werden Nachrichten entfernt. Ist *autoprint* aktiv, so wird die Nachricht, die auf die zuletzt gelöschte folgt, ausgegeben (siehe *Umgebungsvariablen*.)

**discard**[␣Kopfzeilen-Feld ...]
**di**[␣Kopfzeilen-Feld ...]

**ignore**[␣Kopfzeilen-Feld ...]
**ig**[␣Kopfzeilen-Feld ...]
Bei der Ausgabe von Nachrichten werden die angegebenen Kopfzeilen-Felder weggelassen. In einem solchen Kopffeld können z.B. *status* und *cc* enthalten sein. Mit der Abspeicherung der Nachricht werden diese Felder jedoch ebenfalls abgespeichert. Dieses Kommando wird durch *P*rint und *T*ype aufgehoben.

**dp**[␣liste]
**dt**[␣liste]
Die angegebenen Nachrichten werden aus dem *briefkasten* entfernt; die unmittelbar auf die zuletzt gelöschte Nachricht folgende Nachricht wird ausgegeben. Dieses Kommando hat also dieselbe Wirkung wie ein *d*elete Kommando in Verbindung mit einem *p*rint Kommando.

**echo**␣zeichenkette ...
**ec**␣zeichenkette ...
Die angegebenen Zeichenketten werden ausgegeben (wie bei echo(1)).

**edit**[␣nachrichten]
**e**[␣nachrichten]
Die angegebenen Nachrichten werden editiert. Die Nachrichten werden in eine Zwischendatei geschrieben; über die Variable *EDITOR* wird der Name des Editors bestimmt. Standardmäßig wird der Editor *ed* verwendet.

**exit**
**ex**
**xit**
**x**
*mailx* wird beendet, ohne daß sich am *briefkasten* etwas ändert; in *mbox* (siehe auch *q*uit) werden keine Nachrichten abgespeichert.

**file**[␣dateiname]
**fi**[␣dateiname]
**folder**[␣dateiname]
**fold**[␣dateiname]
Die aktuelle Nachrichten-Datei wird verlassen; dann wird die Datei *dateiname* eingelesen. Im folgenden sind die Sonderzeichen aufge-

führt, die anstelle eines *dateinamen*s verwendet werden dürfen; sie stehen für folgende Dateien:

| | |
|---|---|
| % | der aktuelle *briefkasten*. |
| %benutzer | der *briefkasten* für *benutzer*. |
| # | die letzte Datei. |
| & | die aktuelle *mbox*-Datei. |

Standard: der aktuelle *briefkasten*.

**folders**
Die Namen der Dateien im Dateiverzeichnis werden ausgegeben, die über die Umgebungsvariable *folder* angegeben wurden (siehe *Umgebungsvariablen*).

**followup**[␣message]
**fo**[␣message]
Eine Nachricht wird beantwortet und die Antwort in eine Datei geschrieben, deren Name vom Namen des Autors der Nachricht abgeleitet ist. Die Wirkung der Variablen *record* wird dadurch gegebenenfalls aufgehoben. Siehe auch Kommandos *F*ollowup, *S*ave und *C*opy sowie *outfolder* (*Umgebungsvariablen*).

**Followup**[␣nachrichten]
**F**[␣nachrichten]
Die erste in *nachrichten* enthaltene Nachricht wird beantwortet, wobei die Nachricht an die Autoren aller in *nachrichten* aufgeführten Nachrichten gesandt wird. Die *subject*-Zeile wird der ersten Nachricht entnommen, und die Antwort wird in eine Datei geschrieben, deren Name vom Namen des Autors der ersten Nachricht abgeleitet wurde. Siehe auch Kommandos *f*ollowup, *S*ave, und *C*opy commands sowie *outfolder* (*Umgebungsvariablen*).

**from**[␣nachrichten]
**f**[␣nachrichten]
Die Kopfzeilen der angegebenen Nachrichten werden ausgegeben.

**group**␣alias␣name ...
**g**␣alias␣name ...

**alias**␣alias␣name ...
**a**␣alias␣name ...
Für die angegebenen *name*n kann ein *alias* angegeben werden. Wird *alias* als Empfänger angegeben, so werden dafür die *name*n eingesetzt. Dies ist nützlich in der Datei *.mailrc*.

**headers**[␣nachricht]
**h**[␣nachricht]
Die Kopfzeilen-Seite wird ausgegeben, auf der die Kopfzeile der angegebenen Nachricht enthalten ist. Die Variable *screen* bestimmt über die Zahl der Kopfzeilen pro Seite (siehe *Umgebungsvariablen*). Siehe auch *z*-Kommando.

**help**
Eine Übersicht der verfügbaren Kommandos wird ausgegeben.

**hold**[␣nachrichten]
**ho**[␣nachrichten]
**preserve**[␣nachrichten]
**pre**[␣nachrichten]
Die angegebenen Nachrichten werden im *briefkasten* aufgewahrt.

**if**␣s|r
*mail*-Kommando ...
**else**␣*mail*-Kommando . . .
**endif**

**i**␣s|r
*mail*-Kommando. . .
**el**␣*mail*-Kommando . . .
**en**
Ist das Programm im *Sende*-Modus, so führt es, wenn *s* angegeben ist, die nachfolgenden *mail*-Kommandos aus, bis es auf eine *el*se oder *en*dif Anweisung stößt; ist das Argument *r* angegeben, so werden die *mail*-Kommandos nur im *Empfangs*-Modus abgearbeitet. Dieses Kommando ist in der Datei *.mailrc* hilfreich.

**ignore**[␣Kopfzeilen-Feld ...]
**ig**[␣Kopfzeilen-Feld ...]

**discard**[␣Kopfzeilen-Feld ...]
**di**[␣Kopfzeilen-Feld ...]
Bei der Ausgabe von Nachrichten auf dem Bildschirm wird die Anzeige der angegebenen Kopffelder (z.B. *status* und *cc*) unterdrückt. Die Felder werden beim Sichern der Nachrichten nicht entfernt. Mit den Kommandos *P*rint und *T*ype wird dieses Kommando aufgehoben.

**list**
**l**
Alle verfügbaren Kommandos werden ohne zusätzliche Erläuterungen ausgegeben.

**mail**␣name ...
**m**␣name ...
An die angegebenen Benutzer wird eine Nachricht übermittelt.

**mbox**[␣nachrichten]
**mb**[␣nachrichten]
Die angegebenen Meldungen werden bei einer normalen Beendigung von *mailx* in die Datei *mbox* geschrieben. Nähere Informationen über diese Datei sind unter *MBOX* (*Umgebungsvariablen*) nachzulesen. Siehe auch Kommandos *ex*it und *q*uit.

**next**[␣nachricht]
**n**[␣nachricht]
Sprung zur nächsten Meldung, die zu *nachricht* paßt. Es kann eine *nachrichten*-Liste angegeben werden; jedoch wird dann nur die erste gültige Meldung aus dieser Liste verwendet. Dies ist hilfreich, wenn zur nächsten Meldung eines bestimmten Benutzers gesprungen werden muß, da, falls kein Kommando angegeben wäre, der Name als Kommando interpretiert werden würde.
Für Informationen über die Spezifikation von Nachrichten siehe oben *nachrichtenliste*.

**pipe**[␣nachrichten][␣Shell-Kommando]
**pi**[␣nachrichten][␣Shell-Kommando]
|[nachrichten][␣Shell-Kommando]
Die Nachricht wird mit einer Pipe an das *Shell-Kommando* weitergegeben. Die Nachricht wird so behandelt, als würde sie eingelesen. Werden keine Argumente angegeben, so wird die aktuelle Nachricht an das Kommando übergeben, das über die Variable *cmd* angegeben wird. Ist die Variable *page* gesetzt, so wird jeder Nachricht ein Formularvorschub-Zeichen angehängt (siehe *Umgebungsvariablen*).

**preserve**[␣nachrichten]
**pre**[␣nachrichten]
**hold**[␣nachrichten]
**ho**[␣nachrichten]
Die angegebenen Nachrichten werden im *briefkasten* aufbewahrt.

**Print**[␣nachrichten]
**P**[␣nachrichten]
**Type**[␣nachrichten]
**T**[␣nachrichten]
Die angegebenen Nachrichten werden einschließlich aller Kopfzeilen-Felder auf dem Bildschirm ausgegeben. Dieses Kommando hebt das *ig*nore Kommando auf, mit dem die Anzeige der Kopfzeilen-Felder unterdrückt wird.

**print**[␣nachrichten]
**p**[␣nachrichten]
**type**[␣nachrichten]
**t**[␣nachrichten]
Die angegebenen Nachrichten werden ausgegeben. Ist *crt* gesetzt, so werden alle Nachrichten, die aus mehr Zeilen bestehen, als durch *crt* festgelegt ist, durch das Kommando paginiert, das über die Umgebungsvariable *PAGER* angegeben wurde (Standard: *pg*). (Siehe *Umgebungsvariablen*.)

**quit**
**q**
*mailx* wird beendet; gelesene Nachrichten werden in die Datei *mbox*, ungelesene Nachrichten in den *briefkasten* geschrieben. Nachrichten, die explizit in einer Datei gesichert worden sind, werden gelöscht.

**Reply**[␣nachrichten]
**Respond**[␣nachrichten]
**R**[␣nachrichten]
Den Autoren der in *nachrichten* enthaltenen Nachrichten wird eine Antwort übermittelt. Die *subject*-Zeile wird der ersten Nachricht entnommen. Ist *record* ein Dateiname zugeordnet, so wird die Antwort am Ende dieser Datei abgespeichert (siehe *Umgebungsvariablen*).

**reply**[␣nachricht]
**r**[␣nachricht]
Die angegebene *nachricht* wird beantwortet; die Antwort wird an alle Adressaten der Nachricht gesandt. Ist *record* ein Dateiname zugeordnet, so wird die Antwort am Ende dieser Datei eingefügt (siehe *Umgebungsvariablen*).

**Save**[␣nachrichten]
**S**[␣nachrichten]
Die angegebenen Nachrichten werden in eine Datei geschrieben, deren Name vom Autor der ersten Nachricht abgeleitet ist (die NetzwerkAdressen wurden entfernt). Siehe auch die Kommandos *C*opy, *f*ollowup und *F*ollowup sowie *outfolder* (*Umgebungsvariablen*).

**save**[␣dateiname]
**s**[␣dateiname]

**save**[␣nachrichten]␣dateiname
**s**[␣nachrichten]␣dateiname
Die angegebenen Nachrichten werden in die Datei *dateiname* geschrieben. Existiert die angegebene Datei nicht, so wird sie angelegt. Die Nachricht wird bei der Beendigung von *mailx* aus dem *briefkasten* entfernt, es sei denn, die Variable *keepsave* ist gesetzt (siehe auch *Umgebungsvariablen* sowie die Kommandos *ex*it und *q*uit).

**set**[␣name]
**se**[␣name]
**set**[␣name = zeichenkette]
**se**[␣name = zeichenkette]
**set**[␣name = zahl]
**se**[␣name = zahl]
Eine Variable namens *name* wird definiert; *name* kann eine leere Zeichenkette, eine Zeichenkette oder eine Zahl sein. Wird nur *se*t angegeben, so werden alle definierten Variablen und ihre Werte ausgegeben. Ausführliche Informationen über die *mailx*-Variablen sind im Abschnitt *Umgebungsvariablen* enthalten.

**Shell**
**sh**
Ein interaktiver Kommandointerpreter wird aufgerufen, das ist i.a. die shell selbst (siehe *Umgebungsvariablen*).

**size**[␣nachrichten]
**si**[␣nachrichten]
Die Länge der angegebenen Nachrichten wird ausgegeben (Zahl der Zeichen).

**source**␣dateiname
**so**␣dateiname
Kommandos werden aus der angegebenen Datei gelesen; dann wird wieder der Kommando-Modus eingeschaltet.

**top**[␣nachrichten]
**to**[␣nachrichten]
Die ersten Zeilen der angegebenen Nachrichten werden ausgegeben. Ist die Variable *toplines* gesetzt, so wird sie als Anzahl der auszugebenden Zeilen interpretiert (siehe *Umgebungsvariablen*). Standardmäßig besteht die Ausgabe aus 5 Zeilen.

**touch**[␣nachrichten]
**tou**[␣nachrichten]
Die angegebenen Nachrichten werden markiert, damit sie so behandelt werden wie beschrieben. Ist eine der Nachrichten in *nachrichten* nicht über das *s*ave-Kommando in einer Datei gesichert worden, so wird sie bei normaler Beendigung von *mailx* in die Datei *mbox* geschrieben. (Siehe *ex*it und *q*uit.)

**Type**[␣nachrichten]
**T**[␣nachrichten]

**Print**[␣nachrichten]
**P**[␣nachrichten]
Die angegebenen Nachrichten werden einschließlich aller Kopffelder auf dem Bildschirm ausgegeben. Das *ig*nore-Kommando, mit dem die Anzeige der Kopffelder unterdrückt wird, wird dadurch aufgehoben.

**type**[␣nachrichten]
**t**[␣nachrichten]

**print**[␣nachrichten]
**p**[␣nachrichten]
Die angegebenen Nachrichten werden ausgegeben. Ist die Variable *crt* gesetzt, so werden alle Nachrichten, die aus mehr Zeilen bestehen, als über die Variable *crt* angegeben ist, durch das Kommando paginiert, das über die Variable *PAGER* angegeben wurde (Standard: *pg*). Siehe (*Umgebungsvariablen*.)

**undelete**[␣nachrichten]
**u**[␣nachrichten]
Die angegebenen, gelöschten Nachrichten werden wiederhergestellt. Es werden nur die Nachrichten wiederhergestellt, die im aktuellen Lauf von *mailx* gelöscht worden sind. Ist *autoprint* aktiv, so wird die letzte der wiederhergestellten Nachrichten ausgegeben (siehe *Umgebungsvariablen*).

**unset**␣name ...
**uns**␣name ...
Die angegebenen Variablen werden gelöscht. Umgebungsvariablen können nicht gelöscht werden.

**version**
**ve**
Die aktuelle Versions- und Freigabenummer werden ausgegeben.

**visual**[␣nachrichten]
**v**[␣nachrichten]
Die angegebenen Nachrichten werden mit einem bildschirmorientierten Editor korrigiert. Die Nachrichten werden in eine Zwischendatei geschrieben; über die Variable *VISUAL* kann der Name des gewünschten Editors angegeben werden (siehe *Umgebungsvariablen*).

**write**[␣nachrichten] dateiname
**w**[␣nachrichten] dateiname
Die angegebenen Nachrichten werden in die angegebene Datei, jedoch ohne Kopfzeile und abschließende Leerzeile, geschrieben. Ansonsten wirkt dieses Kommando wie das *s*ave-Kommando.

**xit**
**exit**
**x**
*mailx* wird beendet, ohne daß der *briefkasten* verändert worden ist. In der Datei *mbox* werden keine Nachrichten gesichert (siehe auch *q*uit).

**z**[+-]
Die Anzeige der Kopfzeilen wird um eine Bildschirmseite nach oben oder unten gerollt. Die Zahl der dargestellten Kopfzeilen kann über die Variable *screen* angegeben werden (siehe *Umgebungsvariablen*).

### Tilde-Kommandos für den Eingabemodus

Die folgenden Kommandos können nur im *Eingabe*-Modus eingegeben werden. Die Kommandozeilen müssen mit dem Escape-Zeichen Tilde (~) beginnen. Informationen darüber, wie dieses Sonderzeichen geändert werden kann, sind unter *escape* (*Umgebungsvariablen*) enthalten.

~ !Shell-Kommando
Der Kommandointerpreter wird aufgerufen.

~.
Das Dateiende wird simuliert. (Die Eingabe der Nachricht(en) wird beendet.)

~ :*mail*-Kommando
~ ␣*mail*-Kommando
Das *mail*-Kommando wird ausgeführt. Nur zulässig, wenn während des Einlesens von Nachrichten welche gesendet werden.

~ **?**
Eine Auflistung der Tilde-Kommandos wird ausgegeben.

~ **A**
Die bei *Sign* angegebene Zeichenkette wird in die Nachricht eingefügt (siehe *Umgebungsvariablen*).

~ **a**
Die bei *sign* angegebene Zeichenkette wird in die Nachricht eingefügt (siehe *Umgebungsvariablen*).

~ **b**name ...
*name* wird in die Bcc- (blind carbon copy)-Liste eingefügt.

~ **c**name ...
*name* wird in die Cc- (carbon copy)-Liste eingefügt.

~ **d**
Die Datei *dead.letter* wird eingelesen. Nähere Informationen über diese Datei sind unter *DEAD* (*Umgebungsvariablen*) enthalten.

~ **e**
Der Nachrichtenpuffer wird editiert. Siehe auch *EDITOR* (*Umgebungsvariablen*).

~ **f**[nachrichten]
Die angegebenen *nachrichten* werden weitergeleitet. Die Nachrichten werden unverändert in die Nachricht eingefügt.

~ **h**
Eine Aufforderung zur Eingabe der *subject*, To- Cc- und Bcc-Listen wird ausgegeben.
Wird im Feld ein Anfangswert ausgegeben, so kann er editiert werden, als ob er gerade eingegeben worden wäre.

~ **i**␣zeichenkette
Der Wert der angegebenen Variablen wird in den Text der Nachricht eingefügt. So sind z.B. ~*A* und ~*i Sign* gleichbedeutend.

~ **m**[nachrichten]
Die angegebenen Nachrichten werden in die Post eingefügt, wobei der neue Text zum nächsten Tabulatorstop verschoben wird. Dieses Kommando ist nur gültig, wenn gleichzeitig eine Nachricht geschickt und eine Nachricht empfangen wird.

~ **p**
Die eingegebenen Nachrichten werden ausgegeben.

~ **q**
Der Empfang eines Unterbrechungssignals wird simuliert, um den Eingabemodus zu verlassen. Wurde eine Nachricht eingegeben, so wird sie in die Datei *dead.letter*. geschrieben. Nähere Informationen über diese Datei sind unter *DEAD* (*Umgebungsvariablen*) enthalten.

~ **r**dateiname
~ **<** dateiname
~ **<** **!**Shell-Kommando
Die angegebene Datei wird eingelesen. Beginnt das Argument mit einem *!*, so werden die darauffolgenden Zeichenketten als ein Systemkommando interpretiert und ausgeführt; die Standard-Ausgabe davon wird in die Nachricht eingefügt.

~ **s**zeichenkette ...
In der *subject*-Zeile soll *zeichenkette* stehen.

~ **t**name ...
Der angegebene *name* wird in die To-Liste eingefügt.

~ **v**
Zur Korrektur der Teilnachricht wird der gewünschte bildschirmorientierte Editor aufgerufen. Siehe auch *VISUAL* (*Umgebungsvariablen*).

~ **w**␣dateiname
Die angegebene Nachricht wird ohne Kopfzeile in die angegebene Datei geschrieben.

~ **x**
*mailx* wird wie bei ~*q* beendet, nur wird die Nachricht nicht in die Datei *dead.letter* geschrieben.

~ Shell-Kommando
Der Hauptteil der Nachricht wird an das angegebene *Shell-Kommando* übergeben. Geht aus dem Ende-Status des *Shell-Kommandos* hervor, daß das Kommando erfolgreich ausgeführt wurde, so wird die Nachricht durch die Ausgabe des *Shell-Kommandos* ersetzt.

## UMGEBUNGSVARIABLEN

Im folgenden sind die Umgebungsvariablen aufgeführt; sie können nicht durch *mailx* geändert werden.

**HOME** = directory
Das Haupt-Dateiverzeichnis des Benutzers

**MAILRC** = dateiname
Der Name der Start-Datei. Standard: *$HOME/.mailrc*.

Bei den folgenden Variablen handelt es sich um interne *mailx*-Variablen. Sie können aus der Ausführungs-Umgebung importiert oder über das *set*-Kommando gesetzt worden sein. Mit dem *unset*-Kommando können Variablen gelöscht werden.

**allnet**
Alle Netzwerk-Namen, deren letzer Teil (Login-Name) übereinstimmt, werden als identisch angesehen. Dies bewirkt, daß die für *nachrichten* angegebenen Spezifikationen eine ähnliche Wirkung haben. Standard: *noallnet*. Siehe auch Kommando *alternates* und Variable *metoo*.

**append**
Bei Beendigung von *mailx* werden die Nachrichten nicht an den Anfang, sondern an das Ende der Datei *mailx* gesetzt. Standard: *noappend*.

**askcc**
Nach der Eingabe der Nachricht wird der Benutzer zur Eingabe der Cc-Liste aufgefordert. Standard: *noaskcc*.

**asksub**
Die Aufforderung zur Eingabe des *subject* wird angezeigt, wenn es nicht auf der Kommandozeile über den Schalter *-s* angegeben wurde. Standardmäßig wird diese Aufforderung angezeigt.

**autoprint**
Nach der Durchführung der Kommandos *delete* und *undelete* werden Nachrichten automatisch ausgegeben. Standard: *noautoprint*.

**bang**
Das ! hat eine spezielle Bedeutung, wie z.B. in *vi*(1). Standard: *nobang*.

**cmd** = Shell-Kommando
Für das *pipe* Kommando wird das Standard-Kommando angegeben. Es gibt keinen Standard-Wert.

**conv** = conversion
*uucp*-Adressen werden im angegebenen Format ausgegeben. Standardmäßig findet keine Umwandlung statt. Siehe auch *sendmail* und Schalter *-U*.

**crt** = n
Nachrichten mit mehr als *n* Zeilen werden mit einer Pipe an das Kommando übergeben, das über die Variable *PAGER* (Standard: *pg*(1)) angegeben wurde. Diese Funktion ist standardmäßig nicht aktiv.

**DEAD** = dateiname
Der Name der Datei, in der im Falle eines Unterbrechungssignals oder eines Übermittlungsfehlers der bereits eingegebene Teil der Nachricht gespeichert werden soll. Standard: *$HOME/dead.letter*.

**debug**
Im Falle eines Fehlers werden ausführliche Fehlermeldungen ausgegeben. Es werden keine Nachrichten übermittelt. Standard: *nodebug*.

**dot**
Ist in einer Zeile nur ein einzelner Punkt enthalten, so wird er als Dateiende-Zeichen interpretiert. Standard: *nodot*.

**EDITOR** = Shell-Kommando
Das Kommando, das beim *e*dit-oder ~*e*-Kommando aufgerufen werden soll. Standard: *ed*(1).

**escape** = c
Statt ~ wird *c* als Escape-Zeichen interpretiert.

**folder** = dateiverzeichnis
Das Dateiverzeichnis, in dem Dateien mit Nachrichten standardmäßig angelegt werden sollen. Benutzerdefinierten Dateinamen, die mit einem Plus-Zeichen (+) beginnen, wird der Name des betreffenden Dateiverzeichnisses vorangestellt; damit erhält man den realen Dateinamen. Eine derartige Namensbildung ist jedoch nur möglich, wenn *folder* eine exportierte Umgebungsvariable ist. Beginnt der Name des *dateiverzeichnisses* nicht mit einem Schrägstrich (/), so wird dem Dateinamen *$HOME* vorangestellt. Für die Variable *folder* gibt es keine Standardeinstellung. Siehe auch *outfolder*.

**header**
Beim Aufrufen von *mailx* werden die Kopfzeilen ausgegeben (Standard).

**hold**
Die im Briefkasten enthaltenen Nachrichten werden nicht in die Standard-Datei *mbox* geschrieben, sondern aufbewahrt. Standard: *nohold*.

**ignore**
Bei der Eingabe von Nachrichten wird der Empfang des **SIGINT**-Signals ignoriert. Besonders praktisch bei Wahlleitungen, bei denen häufig Störungen auftreten. Standard: *noignore*.

**ignoreeof**
Während der Eingabe einer Nachricht wird das Dateiende-Zeichen ignoriert. Die Eingabe muß entweder mit dem ~-Kommando oder einem Punkt abgeschlossen werden (außer dem Punkt darf dann in der Zeile nichts mehr stehen). Standard: *noignoreeof*. Siehe auch *dot*.

**keep**
Ist der *briefkasten* leer, so wird er nicht gelöscht, sondern auf die Länge null gebracht. Standard: der Briefkasten wird gelöscht.

**keepsave**
Nachrichten, die gesichert worden sind, werden nicht gelöscht, sondern im *briefkasten* aufbewahrt. Standard: *nokeepsave*.

**MBOX** = dateiname
Der Name der Datei, in der eingelesene Nachrichten gesichert werden sollen. Diese Funktion wird durch das Kommando *x*it sowie das Kommando *s*ave aufgehoben. Standard: *$HOME/mbox*.

**metoo**
Erscheint der Login-Name des Benutzers als Adressat, so wird er nicht aus der Liste gestrichen. Standard: *nometoo*.

**LISTER** = Shell-Kommando
Die Kommandos (und deren Schalter), die bei der Ausgabe des Inhaltsverzeichnisses des *folder*-Dateiverzeichnisses zur Anwendung kommen sollen. Standard: *ls*.

**onehop**
Wenn eine Nachricht beantwortet wird, die ursprünglich an mehrere Empfänger geschickt wurde, müssen für die Antwort die Adressen der Empfänger normalerweise relativ zum Rechner des ursprünglichen Absenders angegeben werden. Ist jedoch diese Variable gesetzt, können die Empfängeradressen direkt angegeben werden, vorausgesetzt, es wird mit einem Rechnernetz gearbeitet, in dem Nachrichten direkt von einem Rechner zu einem anderen gesendet werden können.

**outfolder**
Die Dateien, in die abgeschickte Nachrichten geschrieben werden, werden in dem Dateiverzeichnis angelegt, das über die Variable *folder* angegeben wurde, es sei denn, es handelt sich um einen absoluten Pfadnamen. Standard: *nooutfolder*. Siehe *folder* sowie die Kommandos *S*ave, *C*opy, *fo*llowup und *F*ollowup.

**page**
Beim *pi*pe-Kommando wird jeder Nachricht, die durch eine Pipe geschickt wird, ein Seitenvorschub-Zeichen angefügt. Standard: *nopage*.

**PAGER** = Shell-Kommando
Das Kommando, das als Filter zur Paginierung der Ausgabe benutzt wird (Standard: *pg*). Damit können auch die zu verwendenden Schalter angegeben werden.

**prompt** = zeichenkette
Im *Kommando*-Modus wird als Eingabeaufforderung *zeichenkette* ausgegeben. Standard: *?*.

**quiet**
Die Anzeige der einleitenden Nachricht und der Version beim Aufruf von *mailx* wird unterdrückt (Standard: *noquiet*).

**record** = dateiname
Alle abgeschickten Nachrichten werden in der Datei *dateiname* gespeichert. Diese Funktion ist standardmäßig nicht aktiv. Siehe auch *outfolder*.

**save**
Beim Empfang eines Unterbrechungssignals oder einem Übermittlungsfehler werden die Nachrichten in *dead.letter* gespeichert. Nähere Informationen zu dieser Datei sind unter *DEAD* enthalten. Standardmäßig an.

**screen** = *number*
Gibt die Zahl der Kopfzeilem pro Bildschirmseite für das Kommando *headers* an.

**sendmail** = Shell-Kommando
Anderes Kommando zum Übermitteln von Nachrichten. Standard: *mail*.

**sendwait**
Vor der Rückkehr zum *mail*-Kommando soll auf die Beendigung des *mail*-Dämon-Prozesses gewartet werden. Standard: *nosendwait*.

**SHELL** = Shell-Kommando
Der Name des bevorzugten Kommandointerpreters. Standard: *sh*.

**showto**
Bei der Ausgabe der Kopfzeilen wird, wenn die Nachricht vom Benutzer gesendet wird, nicht der Name des Autors, sondern derjenige des Empfängers ausgegeben.

**sign** = zeichenkette
Die Variable, die beim Kommando *~a* (autograph) in den Nachrichtentext eingefügt wird (siehe auch *~i* (*Tilde-Kommandos*)).

**Sign** = zeichenkette
Die Variable, die beim Kommando *~A* in die Nachricht eingefügt werden soll. Hierfür gibt es keine Standard-Einstellung (siehe auch *~i* (*Tilde-Kommandos*)).

**toplines** = nummer
Die Anzahl der Kopfzeilen, die beim *to*-Kommando ausgegeben werden soll. Standard: 5 Zeilen.

**VISUAL** = Shell-Kommando
Der Name des bevorzugten bildschirmorientierten Editors. Die Standardeinstellung ist *vi*(1).

## DATEIEN

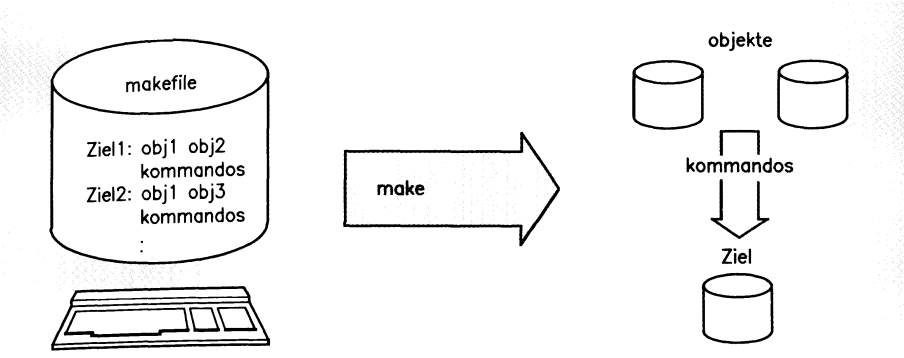
*$HOME/.mailrc*
Die Datei, die nach dem Starten von *mailx* abgearbeitet wird.
*mailx.rc*
Systemweite Start-Datei.
*$HOME/mbox*
Zweite Datei, in die Nachrichten geschrieben werden.
*dead.letter*
Nachrichten, die nicht übermittelt werden können.

## SIEHE AUCH

*mail*(1), *pg*(1), *ls*(1), *vi*(1)

## NAME

**make** - Gruppen von Dateien verwalten




## DEFINITION

**make**[␣schalter...][␣makrodef...][␣ziel...]

## BESCHREIBUNG

*make* dient zur Verwaltung von Dateien, die voneinander abhängen. Die Dateien enthalten üblicherweise Programme eines größeren Programmsystems, können aber auch beliebige Dateien sein. In einer Beschreibungsdatei, dem makefile, legen Sie fest, welche Kommandos jeweils ablaufen sollen, wenn sich bestimmte Dateien geändert haben.

### Wie erstellen Sie ein makefile?

Die Hauptbestandteile eines makefile sind:

- Ziele: Das sind Dateien, die Sie aktualisieren wollen, wenn sich eine der Dateien, von denen das Ziel abhängt, geändert hat.
- Objekte: Das sind die Dateien, von denen das Ziel abhängt.
- Kommandos: Diese legen fest, wie aus den Objekten das Ziel zu erzeugen ist.

*Syntax*

Ein Eintrag im makefile hat das folgende Format:

```
ziel1 [ziel2...]: [objekt...][;kommando] [#...]
[⇥ kommando] [#...]
```

Zeichenreihen, die nach einem Strichpunkt stehen, und alle folgenden Zeilen, die mit einem Tabulatorzeichen ⇥ beginnen, interpretiert *make* als Kommandos. Die erste Zeile, die nicht mit einem Tabulatorzeichen oder einem Kommentarzeichen # anfängt, beginnt einen weiteren Eintrag bzw. eine Makro-Definition (siehe unten). Text zwischen # und einem Neue-Zeile-Zeichen betrachtet *make* als Kommentar.

! Der Editor *ced*(1) akzeptiert die Tabulatortaste ⇥ nicht; wenn Sie jedoch mindestens 8 Leerzeichen am Zeilenanfang stehen haben, macht *ced* automatisch immer ein Tabulatorzeichen daraus.

*Wie arbeitet make?*

Geben Sie beim *make*-Aufruf ein Ziel an, dann bearbeitet *make* dieses Ziel und alle Ziele, von denen das angegebene Ziel abhängt.
Geben Sie kein Ziel an, dann bearbeitet *make* das erste Ziel im makefile und alle Ziele, von denen das erste Ziel abhängt.

*make* prüft für die betreffenden Ziele und Objekte den Zeitpunkt der letzten Änderung. Ist eines der Objekte neueren Datums als das Ziel, führt *make* die darunter stehenden Kommandos aus. Ist ein Ziel oder Objekt nicht vorhanden, dann wird es von *make*, falls möglich, mit dem betreffenden makefile-Kommando bzw. mit Hilfe einer Regel erzeugt.

*make* gibt auf der Standard-Ausgabe jedes makefile-Kommando aus, das es ausführen läßt. Wenn das angegebene Ziel *ziel* bereits auf dem neuesten Stand ist, gibt *make* aus:
`'ziel' is up to date.`
(Ausnahme: siehe Schalter *-s* und das spezielle Ziel *.SILENT*)

*Beispiel*

Ein C-Programm *prog* wird aus drei Modulen *teil1.o*, *teil2.o* und *teil3.o* gebunden. Zu diesen Modulen gehören die C-Quellprogramme *teil1.c*, *teil2.c* und *teil3.c*.
*teil1.c* und *teil2.c* verwenden beide die Include-Datei *def.h*, d.h. sie enthalten die Anweisung:

```
#include "def.h"
```

Ändern Sie eines der drei Quellprogramme, dann muß dieses Quellprogramm neu übersetzt und anschließend *prog* neu gebunden werden.
Ändern Sie die Include-Datei *def.h*, dann müssen die Quellprogramme *teil1.c* und *teil2.c* neu übersetzt und anschließend *prog* neu gebunden werden.
Diese Aufgabe kann *make* für Sie erledigen. Dazu legen Sie eine Datei *makefile* oder *Makefile* an. Sie können die Datei auch anders nennen, müssen dann aber den Dateinamen beim *make*-Aufruf mit Schalter *-f* angeben.
In die Datei tragen Sie folgenden Text ein (beachten Sie die acht Leerzeichen vor den Kommandozeilen!):

```
prog: teil1.o teil2.o teil3.o
        cc teil1.o teil2.o teil3.o -lm -o prog
teil1.o: teil1.c def.h
        cc -c teil1.c
teil2.o: teil2.c def.h
        cc -c teil2.c
teil3.o: teil3.c
        cc -c teil3.c
```

Vier Ziele sind definiert: *prog*, *teil1.o*, *teil2.o* und *teil3.o*. Das Ziel *prog* hängt ab von *teil1.o*, *teil2.o* und *teil3.o*; diese sind wiederum Ziele: z.B. hängt *teil1.o* ab von *teil1.c* und *def.h* usw.
Beachten Sie: *teil1.c* hängt nicht von *def.h* ab! (Denn wenn Sie *def.h* ändern, dann muß *teil1.c* zwar neu übersetzt, aber selbst nicht verändert werden.)

Angenommen, Sie ändern nun *teil2.c*. Anschließend rufen Sie *make* auf:

`make prog` bzw. `make -f datei prog`

oder (was in diesem Fall dasselbe bewirkt)

`make` bzw. `make -f datei`

*make* beginnt beim Ziel *prog*. Die zugehörigen Objekte sind wieder Ziele; folglich prüft *make* diese Ziele, nämlich *teil1.o*, *teil2.o* und *teil3.o*.
*make* erkennt, daß das Objekt *teil2.c* neueren Datums ist als das zugehörige Ziel *teil2.o*. Folglich wird *teil2.c* übersetzt und daraus *teil2.o* erzeugt. *teil2.o* ist dann neueren Datums als *prog*; folglich wird *prog* aus *teil1.o*, *teil2.o* und *teil3.o* neu gebunden.

*Reihenfolge der Ziele*

Sie können die Ziele im makefile in beliebiger Reihenfolge definieren. *make* arbeitet jedoch schneller, wenn Sie die folgende Reihenfolge einhalten:

Wenn aus dem Objekt *objekt* das Ziel *ziel1* erzeugt werden soll und aus *ziel1* das Ziel *ziel2*, dann schreiben Sie *ziel2* vor *ziel1* ins makefile.

Im obigen Beispiel ist diese Reihenfolge eingehalten.

*make* bearbeitet die Einträge im makefile rekursiv:
*make* beginnt bei dem im Aufruf angegebenen Ziel bzw. beim ersten Ziel, wenn nichts angegeben ist. Sind unter den Objekten dieses Zieles weitere Ziele, sucht *make* diese auf usw. Hat *make* alle Abhängigkeiten durchlaufen, führt es schrittweise vom letzten zum ersten Ziel alle Kommandos aus, die warten mußten.

Die Reihenfolge, in der *make* die Einträge im makefile bearbeitet, ist somit durch die Abhängigkeiten von Zielen und Objekten bestimmt, gleichgültig, in welcher Reihenfolge Sie die Ziele definiert haben. Da *make* die Kommandos, die es ausführen läßt, auf der Standard-Ausgabe protokolliert, können Sie mitverfolgen, wie *make* arbeitet.

### Kommentare

Zeichenfolgen, die mit dem Zeichen # beginnen, interpretiert *make* als Kommentar. Ein Kommentar endet am Ende derselben Zeile.

*Beispiel*

```
prog: teil1.o teil2.o teil3.o
        cc teil1.o teil2.o teil3.o -lm -o prog     # prog binden
  .
  .
```

### Abhängigkeiten und Ableitungsregeln

Bei einigen Zielen sind die zugehörigen Objekte aus dem Namen der Ziele ableitbar.
Beispiel: Endet der Name eines Zieles mit *.o*, dann kommen für die zugehörigen Objekte entsprechende Namen in Frage, die mit *.c* oder *.s* usw. enden.

Für solche Fälle hat *make* intern eine Liste von vordefinierten Abhängigkeiten zwischen Dateien und entsprechende Regeln zur Generierung der Zieldateien gespeichert. Deshalb müssen Sie manche Abhängigkeiten und Kommandos im makefile nicht explizit angeben: *make* erzeugt die benötigten Dateien automatisch aus den passenden Objekten.

Wenn Sie das Kommando

```
make -fp - 2>/dev/null </dev/null
```

eingeben, dann schreibt *make* die vordefinierten Regeln, die Liste der Abhängigkeiten, die Makros, die in den vordefinierten Regeln vorkommen, und die Umgebungsvariablen auf die Standard-Ausgabe (siehe Abschnitt "Makros").

Die Liste der Abhängigkeiten steht hinter dem speziellen Ziel *.SUFFIXES*:

```
.SUFFIXES: .o .c .c~ .y .y~ .l .l~ .s .s~ .sh .sh~ .h .h~
```

Diese Liste gibt die möglichen Endungen (Suffixe) an, die Dateinamen haben können. Die Tilde ~ bezieht sich auf eine SCCS-Datei. Die Zeichenreihe *.c~* zum Beispiel bezeichnet eine SCCS-C-Quelldatei, genauer: eine SCCS-Datei *s.name.c*, die verschiedene Versionen von C-Quellcode enthält.

Die Reihenfolge der Suffixe in der Liste ist wichtig: Wenn *make* eine Datei erzeugen soll, für die Generierung aber kein Eintrag im makefile steht, dann durchsucht *make* die Suffixliste von links nach rechts. *make* verwendet das erste Suffix, für das eine Datei und eine passende Regel vorhanden ist.

Die Namen von Regeln setzen sich normalerweise aus zwei Suffixen zusammen: dem Suffix für die Datei, aus der die Zieldatei erzeugt werden soll, und dem Suffix der Zieldatei.
Regeln, die nur ein Suffix aufweisen, z.B. *.c*, geben an, wie *datei* aus *datei.c* zu erzeugen ist. Eine solche Regel können Sie verwenden, wenn Ziele aus nur einer Quelldatei (z.B. Shell-Prozeduren, einfache C-Programme) erzeugt werden sollen.

*Beispiel*

```
.c.o:
        cc -c $<
```

Diese Regel gibt an, wie aus einer *.c*-Datei die zugehörige *.o*-Datei zu erzeugen ist; deshalb heißt die Regel *.c.o*. Das Makro $< bezeichnet hier die *.c*-Datei (siehe Abschnitt "Makros").
Aufgrund dieser Regel können Sie das Beispiel im Abschnitt "Wie erstellen Sie ein makefile?" abgekürzt so schreiben:

```
prog: teil1.o teil2.o teil3.o
        cc teil1.o teil2.o teil3.o -lm -o prog
teil1.o teil2.o: def.h
```

*make* "weiß" automatisch, wie die *.o*-Dateien zu erzeugen sind!

Das folgende Beispiel betrifft das SCCS.
Die Regel *.c~.o* gibt an, wie *make* aus einer SCCS-C-Quelldatei *s.name.c* die zugehörige *.o*-Datei *name.o* erzeugen soll. Vereinfacht lautet diese Regel folgendermaßen:

```
.c~.o
        get -p $< > $*.c
        cc -c $*.c
        rm -f $*.c
```

*get*(1D) holt die jüngste Version aus der SCCS-Datei *s.name.c* und schreibt sie in die Datei *name.c*. Anschließend wird das C-Programm in *name.c* übersetzt; das übersetzte Programm wird in der Datei *name.o* abgelegt. Zum Schluß wird die Datei *name.c* gelöscht.

Sie können auch selbst Regeln erstellen und ins makefile schreiben. Regeln im makefile haben höhere Priorität als die vordefinierten Regeln.
Die vordefinierte Suffixliste von *make* können Sie durch weitere Suffixe ergänzen, indem Sie ins makefile das Ziel *.SUFFIXES* schreiben, dahinter einen Doppelpunkt und dann die zusätzlichen Suffixe. Geben Sie hinter *.SUFFIXES:* nichts an, dann betrachtet *make* die aktuelle Suffixliste als gelöscht.

**Makros**

*make* ermöglicht die Verwendung von Makros. Es gibt vier Arten:

1. Makros, die Sie beim *make*-Aufruf definieren
2. Makros, die Sie im makefile definieren
3. Umgebungsvariablen
4. Vordefinierte Makros

*Makro-Definition*

Beim *make*-Aufruf geben Sie Makro-Definitionen so an, wie im Abschnitt "OPERANDEN" beschrieben.
Ins makefile schreiben Sie Makro-Definitionen in folgendem Format:

```
makro␣=␣zeichenfolge
```

Die Leerzeichen können Sie auch weglassen. Die Zeichenfolge *zeichenfolge* kann leer sein. Mit einem Neue-Zeile-Zeichen oder einem Kommentarzeichen # schließen Sie die Definition ab.

*Makro-Aufruf*

Beim Aufruf eines Makros wird der Makroname durch die Zeichenfolge ersetzt, für die er definiert wurde. Ein Aufruf hat die Form:

```
$(makro)
```

Ist *makro* nur ein Buchstabe, können Sie die Klammern weglassen. Wollen Sie das Dollarzeichen *$* erhalten, dann schreiben Sie ein weiteres Dollarzeichen davor: *$$* wertet *make* aus zu *$*.

*Beispiel*

```
OBJ = teil1.o teil2.o teil3.o
prog: $(OBJ)
        cc $(OBJ) -lm -o prog
  .
  .
```

Für *$(OBJ)* setzt *make* jedesmal die Zeichenfolge *teil1.o teil2.o teil3.o* ein. Damit sparen Sie Schreibarbeit.

Den Makro-Aufruf *$(makro)* können Sie mit einer Ersetzungsanweisung kombinieren:

```
$(makro:teilzf=ersatz)
```

*teilzf* ist eine Teilzeichenfolge von *zeichenfolge*.
Diese Teilzeichenfolge wird überall in *zeichenfolge* ersetzt durch die Zeichenfolge *ersatz*. *ersatz* kann auch leer sein.
Die beteiligten Zeichenfolgen werden durch Leer-, Tabulator-, Neue Zeile-Zeichen und den Zeilenanfang begrenzt.

*Beispiel*

Wenn Sie das Makro *OBJ* wie oben definieren, dann bezeichnet

```
$(OBJ:.o=.c)
```

die Zeichenfolge *teil1.c teil2.c teil3.c*.

*Umgebungsvariablen*

Umgebungsvariablen interpretiert *make* als Makrodefinitionen und verarbeitet sie entsprechend.
Die Umgebungsvariable *MAKEFLAGS* wird von *make* so behandelt, als enthielte sie jeden zulässigen Schalter (außer *-f*), der für den *make*-Aufruf definiert ist. Wenn die Variable *MAKEFLAGS* nicht in der Umgebung vorhanden ist, dann "erfindet" *make* die Variable, weist ihr die Schalter zu, die Sie beim *make*-Aufruf angegeben haben, und gibt die Variable an Kommandos weiter, die im makefile aufgerufen werden.

*Beispiel*

Einem Programmpaket seien die makefiles *mf1, mf2, mf3* usw. zugeordnet. Das makefile *mf1* enthalte das Kommando

```
$(MAKE) -f mf2,
```

das makefile *mf2* enthalte das Kommando

```
$(MAKE) -f mf3
```

usw. Rufen Sie nun *make* mit Schalter *-n* auf:

```
make -fn mf1
```

Das Kommando $(MAKE) wird dann trotz Schalter *-n* ausgeführt; an den entsprechenden *make*-Aufruf wird automatisch der Schalter *-n* übergeben. Somit können Sie das Kommando *make -n* rekursiv auf das ganze Programmsystem anwenden und beobachten, welche Kommandos in den makefiles normalerweise ausgeführt worden wären. Dies ist eine Möglichkeit, eine Fehlersuche in allen makefiles des Programmpakets durchzuführen, ohne tatsächlich etwas zu verändern.

Ähnlich wie in diesem Beispiel gibt *make* auch alle anderen Variablen bzw. Makro-Definitionen an *make*-Aufrufe weiter, die im makefile stehen.

**Vordefinierte Makros**

Die vordefinierten Regeln benutzen

- Makros, die Kommandos bezeichnen, z.B. *CC, LD, LEX, YACC, GET*
- Makros, die Schalter für die betreffenden Kommandos bezeichnen, z.B. *CFLAGS, LDFLAGS, LFLAGS, YFLAGS, GFLAGS*

Diese Makros sind vordefiniert. Wenn Sie das Kommando

```
make -fp - 2>/dev/null </dev/null
```

eingeben, dann schreibt *make* die Definitionen auf die Standard-Ausgabe. Sie können diese Makros aber auch neu definieren. Auf diese Weise können Sie die Dateien im makefile z.B. mit verschiedenen Compilern übersetzen lassen oder den makefile-Kommandos verschiedene Schalter mitgeben.

*Beispiel*

Die vordefinierte *.c.o*-Regel (siehe oben) lautet mit den vordefinierten Makros folgendermaßen:

```
.c.o
        $(CC) $(CFLAGS) -c $<
```

Wollen Sie Ihre C-Quellprogramme mit einem anderen C-Compiler *newcc* übersetzen lassen, dann rufen Sie *make* so auf:

```
make CC=newcc
```

Wollen Sie Ihre C-Quellprogramme mit *cc* und Schalter *-p* übersetzen lassen, dann rufen Sie *make* so auf:

```
make "CFLAGS = -p"
```

Darüber hinaus kennt *make* fünf vordefinierte Makros, die zur Erstellung von Ableitungsregeln verwendet werden können. Diese Makros sind fest vorgegeben; Sie können sie nicht neu definieren. Ihre Bedeutung ist:

$* Der Name des aktuellen Zieles ohne Suffix.
Dieses Makro wird nur für Ableitungsregeln ausgewertet.

$@ Der Name des aktuellen Zieles einschließlich Suffix.
Dieses Makro wird nur für explizit aufgeführte Abhängigkeiten ausgewertet.

$< Dieses Makro wird nur für Ableitungsregeln und das spezielle Ziel *.DEFAULT* ausgewertet und bezeichnet die "berechnete" Abhängigkeit. Das ist das Objekt, das zur Erzeugung des Ziels in den entsprechenden Kommandos eingesetzt werden muß.

$? Alle Objekte, von denen das Ziel abhängt und die neuer sind als das Ziel.

$% *make* wertet dieses Makro nur dann aus, wenn das Ziel ein Modul einer Bibliothek der Form *lib.a(datei.o)* ist. In diesem Fall wertet *make $@* als die Bibliothek *lib.a* und *$%* als Bibliotheksmodul *datei.o* aus.

*Beispiel*

Die Regel *.c.o* kann sowohl mit dem Makro *$** als auch mit *$<* formuliert werden:

```
.c.o:
        cc -c $*.c
```

oder

```
.c.o:
        cc -c $<
```

Ein weiteres Beispiel finden Sie im Abschnitt "Bibliotheken".

An *$*$*, *$@$*, *$<$* und *$\%$* können Sie die Buchstaben *D* ("directory") oder *F* ("file") anhängen. *D* bezeichnet den Dateiverzeichnisteil des entsprechenden Dateinamens, *F* bezeichnet den Dateiteil. Zum Beispiel steht *$(@D)* für den Dateiverzeichnisteil der Zeichenfolge *$@*. Ist kein Dateiverzeichnisteil vorhanden, dann setzt *make* stattdessen *./* ein.

*Prioritäten*

Für die verschiedenen Makro-Arten sind folgende Prioritäten festgelegt:

Schalter *-e* nicht eingeschaltet:

1. Makro-Definitionen, die Sie beim *make*-Aufruf angeben
2. Makro-Definitionen, die Sie ins makefile schreiben
3. Umgebungsvariablen
4. Makros, die bei *make* vordefiniert sind (z.B. *CC, CFLAGS*)

Mit Schalter *-e*:

1. Makro-Definitionen, die Sie beim *make*-Aufruf angeben
3. Umgebungsvariablen
2. Makro-Definitionen, die Sie ins makefile schreiben
4. Makros, die bei *make* vordefiniert sind (z.B. *CC, CFLAGS*)

**Spezielle Ziele**

Im makefile können Sie die folgenden speziellen Ziele definieren; nach dem Zielnamen muß ein Doppelpunkt stehen:

*.DEFAULT*
: Wenn *make* eine Datei erstellen soll, für deren Erstellung weder entsprechende Kommandos im makefile stehen noch vordefinierte oder im makefile explizit angegebene Abhängigkeiten und Regeln vorhanden sind, dann führt *make* die unter *.DEFAULT* stehenden Kommandos aus.

*.PRECIOUS*
Wenn *make* durch das Signal *SIGINT* oder *SIGQUIT* abgebrochen wird, dann wird normalerweise das Ziel gelöscht. Ist jedoch im makefile definiert, daß das Ziel von dem speziellen Ziel *.PRECIOUS* abhängt, dann wird es nicht gelöscht.

*.SILENT*
Normalerweise meldet *make* jedes Kommando, das ausgeführt wird, auf der Standard-Ausgabe.
Wenn Sie das Ziel *.SILENT* irgendwo ins makefile schreiben, dann wird die Protokollierung auf der Standard-Ausgabe unterdrückt.

*.IGNORE*
Wenn Sie das Ziel *.IGNORE* ins makefile schreiben, dann ignoriert *make* Fehlermeldungen, die von Kommandos im makefile ausgegeben werden.

*.SUFFIXES*
Hinter diesem speziellen Ziel können Sie eine Liste von Suffixen angeben, um allgemeine Abhängigkeiten zwischen Dateien zu definieren. Mehrfach vorhandene Suffixlisten addieren sich; geben Sie hinter *.SUFFIXES:* keine Suffixe an, dann betrachtet *make* die Suffixliste als gelöscht. Siehe Abschnitt "Abhängigkeiten und Ableitungsregeln".

**Die Steuerzeichen @ und -**

Vor jedes Kommando im makefile können Sie das Zeichen @ oder oder beide schreiben, z.B. *@-kommando*. Die Zeichen haben folgende Wirkung:

@ *make* protokolliert das betreffende Kommando nicht auf der Standard-Ausgabe, wenn es ausgeführt wird.

– Liefert das betreffende Kommando einen Ende-Status ungleich 0, dann bricht *make* die Bearbeitung des makefile nicht ab, sondern ignoriert den Fehler.

**Bibliotheken**

Namen von Zielen oder Objekten, die Klammern enthalten, interpretiert *make* als Bibliotheken; die Zeichenfolge in Klammern steht für ein Modul der Bibliothek. Zum Beispiel bezeichnet *lib.a(datei.o)* eine Bibliothek *lib.a*, die das Modul *datei.o* enthält. Der Ausdruck *lib.a(datei1.o datei2.o)* ist nicht zulässig.
Regeln für die Erstellung bzw. Aktualisierung von Bibliotheken haben Namen der Form *.suffix.a*; *suffix* ist der Suffix der Dateien, die in die Bibliothek eingegliedert werden sollen.

*Beispiel*

Angenommen, die Dateien *datei1.c, datei2.c* und *datei3.c* enthalten C-Quellcode. Die Bibliothek *lib.a* enthalte die zugehörigen Objektmodule *datei1.o, datei2.o* und *datei3.o*. Das folgende makefile dient zur Verwaltung der Bibliothek *lib.a*:

```
lib.a: lib.a(datei1.o) lib.a(datei2.o) lib.a(datei3.o)
        @echo lib.a ist aktualisiert.
.c.a:
        $(CC) -c $(CFLAGS) $<
        ar rv $@ $*.o
        rm -f $*.o
```

Die Regel *.c.a* müssen Sie nicht ins makefile schreiben, da diese Regel bereits vordefiniert ist.
Den Beschreibungstext im makefile können Sie aber auch anders formulieren:

```
lib.a: lib.a(datei1.o) lib.a(datei2.o) lib.a(datei3.o)
        $(CC) -c $(CFLAGS) $(?:.o=.c)
        ar rv lib.a $?
        rm $?
        @echo lib.a ist aktualisiert.
.c.a:;
```

Mit dieser makefile-Version kostet die Pflege der Bibliothek wesentlich weniger Zeit als mit der obigen Version.
Beachten Sie: Die *.c.a*-Regel kann hier nicht verwendet werden, da damit jeweils nur ein Bibliotheksmodul erstellt und eingegliedert werden kann.

## SCHALTER

Die Reihenfolge der Schalter ist beliebig.
Sie können mehrere Schalter auch kombinieren, z.B. *make -fn mkf*.

**-f␣name**

Für *name* können Sie entweder einen Dateinamen oder das Zeichen Zeichen - angeben.
Wenn Sie einen Dateinamen angeben, dann benutzt *make* den Inhalt dieser Datei als Eingabe.
Wenn Sie das Zeichen - angeben, dann liest *make* von der Standard-Eingabe. Wie üblich schicken Sie eine Zeile mit ↵ ab. Wenn Sie Ihre Eingabe an *make* beenden wollen, dann drücken Sie die Taste END.

Sie können den Schalter *-f* bei einem *make*-Aufruf mehrmals angeben.

*Standard (kein Schalter -f):*
*make* erwartet die Eingabe aus einer Datei namens *makefile*. Wenn eine solche Datei nicht existiert, dann sucht *make* nach einer Datei namens *Makefile*, *s.makefile* oder *s.Makefile*, und zwar in der angegebenen Reihenfolge.

**-i**

*make* ignoriert Fehlermeldungen, die von Kommandos im makefile ausgegeben werden.
Genauso wirkt der Eintrag *.IGNORE:* im makefile.

**-k**

Endet ein Kommando im makefile mit Ende-Status ungleich 0, dann bricht *make* nur die Bearbeitung des davon abhängigen Ziels ab. Weitere, von diesem Kommando nicht abhängige Ziele bearbeitet *make* noch.

**-n**

*make* listet alle Kommandos im makefile, die ausgeführt würden, auf der Standard-Ausgabe auf, führt die Kommandos aber nicht aus.

*Ausnahme*
Wenn im makefile das Kommando *make* als Makro *$(MAKE)* steht, dann wird dieses *make*-Kommando ausgeführt.

*make* gibt auch Kommandozeilen aus, die mit dem Zeichen *@* beginnen.

**-t**

*make* ruft für jedes bearbeitete Ziel das Kommando *touch*(1) auf; dadurch wird der Zeitpunkt der letzten Änderung der Zieldatei auf das aktuelle Datum und die aktuelle Zeit gesetzt. Die Kommandos im makefile, die normalerweise ausgeführt würden, listet *make* nur auf.

**-r**

*make* benutzt nicht die vordefinierten Regeln.
Dieser Schalter wirkt genauso, wie wenn Sie am Anfang des makefile das spezielle Ziel *.SUFFIXES:* mit einer leeren Suffix-Liste angeben.

**-s**

Die Protokollierung auf der Standard-Ausgabe wird unterdrückt ("silently"). Normalerweise meldet *make* jedes Kommando, das es gerade aufruft.
Genauso wirkt der Eintrag *.SILENT:* im makefile.

**-q**

*make* überprüft nur, ob die Zieldatei auf dem neuesten Stand ist ("question"). Wenn ja, liefert *make* den Ende-Status 0, sonst ungleich 0.

**-d**

Dieser Schalter dient der Fehlersuche ("debug mode"). *make* gibt ausführliche Informationen über die betroffenen Dateien aus.

**-e**

Umgebungsvariablen überschreiben Makro-Definitionen im makefile, d.h. haben höhere Priorität.

## OPERANDEN

makrodef

Für *makrodef* können Sie eine Makro-Definition angeben. Die Makro-Definition geben Sie folgendermaßen an:

```
makro=zeichenfolge  oder  "makro␣=␣zeichenfolge"
```

*zeichenfolge* kann auch leer sein.
Enthält die Makro-Definition Leerzeichen, dann müssen Sie sie in Anführungszeichen einschließen.
Sie können hintereinander mehrere Makro-Definitionen angeben.

*Beispiel*

```
make "LINT = lint -p" "CFLAGS=" pgm
```

ziel

Für *ziel* können Sie den Namen eines Ziels angeben, das im makefile steht. *make* bearbeitet dann dieses Ziel und alle Ziele, von denen das angegebene Ziel abhängt.
Sie können auch mehrere Ziele angeben.

*Standard (keine Angabe):*
*make* bearbeitet das erste Ziel und alle Ziele, von denen das erste Ziel abhängt.

**DATEIEN**

*makefile* oder *Makefile*
Beschreibungsdatei
*s.makefile* oder *s.Makefile*
Beschreibungsdatei für SCCS-Dateien

**HINWEIS**

- Pro Kommandozeile wird ein eigener Prozeß erzeugt.
  Eingebaute Kommandos, z.B. *cd*(1), wirken daher nur für die Zeile, in der sie stehen. Mehrere Kommandos in einer Zeile können Sie mit einem Semikolon ; trennen.
- Eine Fortsetzungszeile erwartet *make*, wenn eine Zeile im makefile mit einem Gegenschrägstrich \ endet.
- Die drei Zeichen @, = und : in Dateinamen können zu Problemen führen.

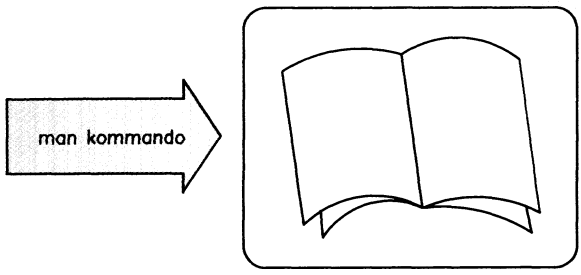
**SIEHE AUCH**

*admin*(1D), *ar*(1), *cc*(1D), *get*(1D), *lex*(1D), *ld*(1D), *yacc*(1D).

Eine Einführung ins SCCS ("Source Code Control System") finden Sie im Buch SINIX Einführung [1].

## NAME

**man** - Einträge des SINIX-Handbuches ausgeben




## DEFINITION

**man**[␣schalter][␣kapitelnummer]␣titel

## BESCHREIBUNG

*man* schreibt zu den Einträgen in diesem Handbuch und im Buch SINIX CES V5.2 die englische On-Line-Dokumentation auf Standard-Ausgabe. *titel* ist der Name des gewünschten Eintrags, *kapitel* die im Suffix des Titels enthaltene Kapitelnummer. *titel* geben Sie in Kleinbuchstaben ein. Die Nummer des *kapitel*s darf nicht mit einem Buchstaben versehen sein. Fehlt *kapitel*, so wird die gesamte On-Line-Dokumentation nach *titel* durchsucht und jedes Vorkommen von *titel* ausgegeben.

## SCHALTER

**-T**term

Der Eintrag wird entsprechend dem Typ der Datensichtstation *term* ausgegeben. Einen Überblick über die zulässigen Werte von *term* erhält man durch Eingabe von *help term2*. Sollen die Informationen auf einem Zeilendrucker ausgegeben werden, so sollte *-lp* verwendet werden.

**-w**

Auf der Standard-Ausgabe werden nur die zu */usr/catman* bzw. (wenn der Schalter *-d* gesetzt ist) die zum aktuellen Dateiverzeichnis relativen Pfadnamen der gewünschten Einträge ausgegeben.

**-d**
Nicht */usr/catman*, sondern das aktuelle Dateiverzeichnis wird durchsucht; hierzu muß der Dateiname vollständig angegeben werden (also *cu.1c*, nicht nur *cu*).

**-c**
*man* ruft *col*(1) auf.

*man* überprüft die Umgebungsvariable $TERM (siehe *environ*(5)) und versucht, die Schalter auszuwählen, mit denen die Ausgabe entsprechend der verwendeten Dateinsichtstation formatiert werden kann. Die Variable $TERM können Sie mit dem Schalter -T*term* aufheben.

## OPERANDEN

kapitel
Nummer des Kapitels, aus dem Sie den Eintrag *titel* lesen möchten. Einen Eintrag mit dem Titel *getopt* z.B. gibt es in Kapitel1 und Kapitel3. Die Kapitelnummer stimmt mit der Nummer des Suffixes überein. Alle Einträge im vorliegenden Manual gehören zum Kapitel 1.

Mit dem Kommandoaufruf

```
$ man man
```

würde auf der Datensichtstation der aktuelle Eintrag sowie alle Einträge namens *man* ausgegeben, die in anderen Kapiteln des Handbuches enthalten sein könnten.

titel
Name des Eintrags, den Sie am Bildschirm lesen möchten.

## DATEIEN

*/usr/catman/?_man/man[1-8]/**
Vorformatierte Handbuch-Einträge

## HINWEIS

Eine seitenweise Ausgabe an Ihrer Datensichtstation erreichen Sie mit folgender Eingabe: man kommando | pg ↵. Weiterblättern können Sie mit ↵.

## BEISPIEL

Beschreibung des Kommandos *mkdir*(1):

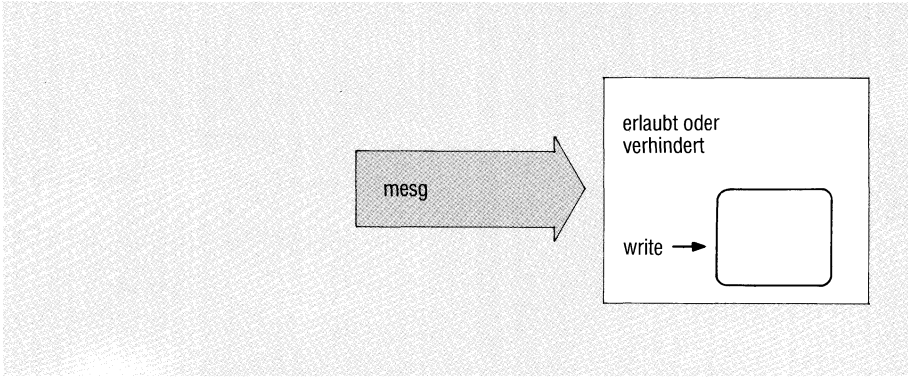
```
$ man 1 mkdir
      MKDIR(1)                    UNIX System V                MKDIR(1)



      NAME
            mkdir - make a directory

      SYNOPSIS
            mkdir dirname ...

      DESCRIPTION
            Mkdir creates specified directories in mode 777 (possibly
            altered by umask(1)). Standard entries, ., for the
            directory itself, and .., for its parent, are made
            automatically.

            Mkdir requires write permission in the parent
            directory.

      SEE ALSO
            sh(1), rm(1), umask(1).

      DIAGNOSTICS
            Mkdir returns exit code 0 if all directories were
            successfully made; otherwise, it prints a
            diagnostic and returns non-zero.
```

## SIEHE AUCH

*term*(5)

## NAME

**mesg** - Senden von Nachrichten verbieten oder erlauben (permit or deny messages)




## DEFINITION

**mesg[␣n][␣y]**

## BESCHREIBUNG

Mit dem Kommando *mesg* können Sie bestimmen, ob andere Benutzer Ihnen mit *write*(1) Nachrichten auf den Bildschirm schreiben können oder nicht.

## OPERANDEN

keine Angabe

*mesg* gibt den aktuellen Stand aus.

**n**

Andere Benutzer dürfen keine Nachrichten an die Datensichtstation des aufrufenden Benutzers (z.B. mit *write*(1)) senden.

**y**

Die Schreiberlaubnis wird wieder erteilt.

**ENDESTATUS**

0 Senden von Nachrichten erlaubt
1 Senden von Nachrichten nicht erlaubt
2 Fehler

**BEISPIEL**

1. Hubert fragt den *mesg*-Status ab:

```
$ mesg
is n
```

2. Susanne möchte Hubert, der am Bildschirm 23 arbeitet, eine Nachricht schicken. Sie bekommt jedoch die Meldung *permission denied*, noch ehe sie die Nachricht schreibt.

```
$ write hubert tty23
permission denied !
$
```

3. Hubert ändert den *mesg*-Status, weil er wieder Nachrichten bekommen möchte und fragt ihn anschließend ab:

```
$ mesg y
$ mesg
is y
$
```

4. Nun hat Susanne, die am Bildschirm 06 arbeitet, zumindest bei der Nachrichtenübermittlung, mehr Erfolg:

```
$ write hubert tty23
Kommst Du heute abend mit ins Kino?
Susanne
END
$
```

Am Bildschirm von Hubert erscheint:

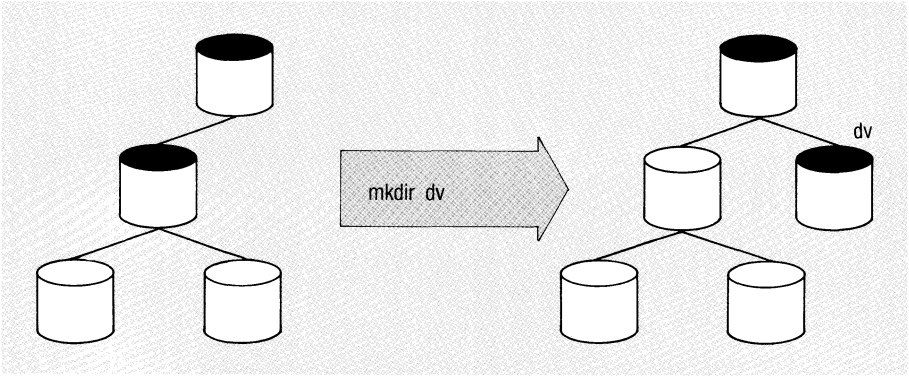
```
message from maxim!susanne on tty06 at 15:31...
Kommst Du heute abend mit ins Kino?
Susanne
EOF
```

**SIEHE AUCH**

*wall*(1), *write*(1)

## NAME

**mkdir** - Dateiverzeichnis erstellen (make a directory)




## DEFINITION

**mkdir** dateiverzeichnis...

## BESCHREIBUNG

Mit dem Kommando *mkdir* können Sie ein neues Dateiverzeichnis einrichten.

*mkdir* trägt im neuen Dateiverzeichnis folgende Verweise ein:

| | | |
|---|---|---|
| . | (Punkt) | für das Dateiverzeichnis selbst |
| .. | (Punkt Punkt) | für das übergeordnete Dateiverzeichnis |

! *mkdir* kann nur ausgeführt werden, wenn Sie im übergeordneten Dateiverzeichnis die Schreibberechtigung besitzen.

## OPERANDEN

dateiverzeichnis

Name des Dateiverzeichnisses, das Sie einrichten möchten.

Wenn Sie *dateiverzeichnis* ohne Pfadnamen angeben, trägt *mkdir* das neue Dateiverzeichnis im aktuellen Dateiverzeichnis ein.
Wenn Sie *dateiverzeichnis* mit Pfadnamen angeben, trägt *mkdir* das neue Dateiverzeichnis in dem Dateiverzeichnis ein, das über den Zugriffspfad angegeben ist.

**ENDESTATUS**

| | |
|---|---|
| 0 | bei Erfolg: Dateiverzeichnisse wurden angelegt |
| ungleich 0 | bei Fehler |

**BEISPIEL**

Anlegen des Dateiverzeichnisses *briefe* im Dateiverzeichnis */usr/helga/sonstiges*:

```
$ pwd
/usr/helga
$ mkdir sonstiges
$ ls -l
total 145
drwx--x--x  2 helga  gruppe1   520  Jan 4  16:21  sonstiges
    .
    .
    .
$
```

**SIEHE AUCH**

*rm*(1), *rmdir*(1)

## NAME

**mknod** - FIFO-Datei erstellen

## DEFINITION

```
mknod␣name␣p
```

## BESCHREIBUNG

*mknod* erstellt eine FIFO-Datei (eine Pipe, die über einen Namen angesprochen werden kann).

## OPERANDEN

name
steht für den Namen der FIFO-Datei.

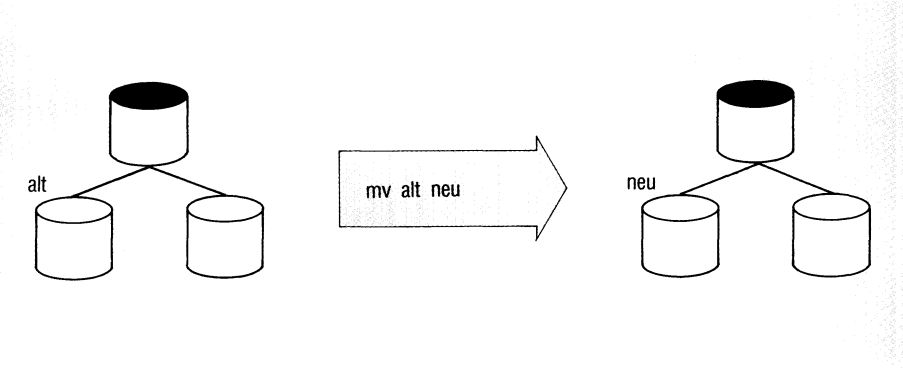
**p**
Das zweite Argument muß der Buchstabe *p* sein; damit wird angegeben, daß eine FIFO generiert werden soll.

## SIEHE AUCH

*mknod*(2)

## NAME

**mv** - Dateien umbenennen oder übertragen (move or rename files)




## DEFINITION

**mv[␣-f]␣datei1[␣datei2...]␣dateineu**

## BESCHREIBUNG

Mit *mv* können Sie eine Datei umbenennen oder im Dateibaum an einen anderen Ort versetzen. Im Unterschied zum Kopieren mit *cp*(1) erzeugt *mv* innerhalb eines Dateisystems keine Kopie der versetzten oder umbenannten Datei. Wird eine Datei über die Grenzen eines Dateisystems versetzt, kann es vorkommen, daß sie kopiert und anschließend gelöscht wird. In diesem Fall gehen alle link-Verbindungen mit anderen Dateien verloren.

## SCHALTER

**-f**

Wenn Sie für *dateineu* eine Datei mit eingeschränkter Schreibberechtigung angeben, werden Sie normalerweise gefragt, ob *mv* wirklich ausgeführt werden soll.
Ist Schalter *-f* gesetzt, unterbleibt diese Frage.

## OPERANDEN

datei1[␣datei2...]

Name der Datei(en), die umbenannt oder versetzt werden sollen.

Wenn *dateineu* noch nicht existiert und dasselbe übergeordnete Dateiverzeichnis wie *datei1* hat, darf *datei1* ein Dateiverzeichnis sein.
Wenn *datei1* eine einfache Datei und *dateineu* ein Verweis auf eine andere Datei mit Verweisen ist, so bleiben die anderen Verweise erhalten, und eine neue Datei mit dem Namen *dateineu* wird angelegt.

dateineu

Wenn *dateineu* eine einfache Datei ist, wird *datei1* entsprechend umbenannt.
Wenn *dateineu* ein Dateiverzeichnis ist, werden die angegebenen Dateien in dieses Dateiverzeichnis versetzt.
Wenn *dateineu* eine Datei ist, für die nur eine eingeschränkte Schreibberechtigung besteht, so wird der Modus ausgegeben und Sie werden zur Beantwortung einer Frage aufgefordert. Wenn Ihre Antwort mit *y* beginnt und Sie die Schreibberechtigung besitzen, so wird *mv* ausgeführt. Ist Schalter *-f* gesetzt oder ist die Standard-Eingabe keine Datensichtstation, unterbleibt diese Frage.

## BEISPIEL

1. Die Datei *lieder* soll umbenannt werden in *popsongs* und versetzt werden in das Dateiverzeichnis */usr/petra/kunst/musik*:

```
mv lieder /usr/petra/kunst/musik/popsongs
```

2. Die Dateien *efeu*, *papyrus* und *flieder* sollen ihren Namen behalten und ins Dateiverzeichnis */usr/petra/pflanzen* eingetragen werden:

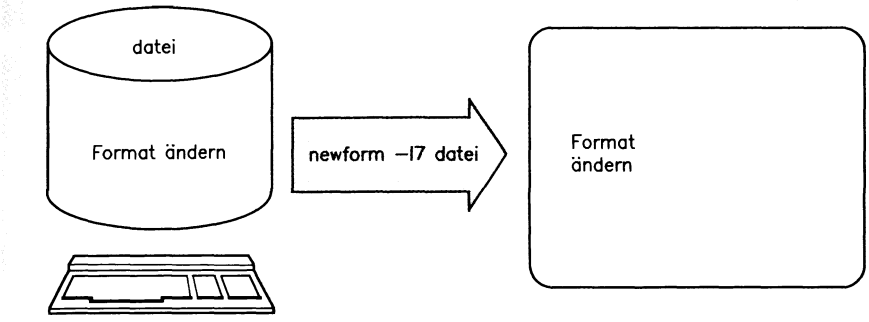
```
mv efeu papyrus flieder /usr/petra/pflanzen
```

## SIEHE AUCH

*chmod*(1), *cp*(1), *cpio*(1), *find*(1), *ln*(1), *rm*(1)

## NAME

**newform** - Format einer Textdatei ändern (new format of a text file)




## DEFINITION

**newform**[␣schalter][␣datei...]

## BESCHREIBUNG

*newform* liest die in den angegebenen *dateien* enthaltenen Zeilen ein und kopiert sie auf die Standard-Ausgabe; die Ausgabe wird entsprechend den angegebenen Schaltern formatiert.

## SCHALTER

Mit Ausnahme von *-s* können Sie die Schalter in beliebiger Reihenfolge und beliebig oft angeben. Außerdem können sie zwischen den Namen der *dateien* stehen. Die Schalter werden nacheinander abgearbeitet. Die Angabe *-e15 -l60* führt also zu einem anderen Ergebnis als *-l60 -e15*. Die Schalter sind für alle *dateien* gültig.

**-i**tabspez

Die Tabulatorzeichen werden entsprechend *tabspez* zu Leerzeichen expandiert. Außerdem kann *tabspez* auch in Form von –– angegeben werden; *newform* sucht die Tabulator-Spezifikationen dann in der ersten Zeile der Standard-Eingabe. *tabspez* hat standardmäßig den Wert *-8*. Wird für *tabspez* der Wert *-0* angegeben, so erwartet *newform* keine Tabulatorzeichen; werden dennoch Tabulatorzeichen eingegeben, so wird der Wert *-1* verwendet.

**-o**tabspez
Bei der Ausgabe werden Leerzeichen entsprechend *tabspez* durch Tabulatorzeichen ersetzt. *tabspez* wird im selben Format wie bei *-itabspez* angegeben. Fehlt *tabspez*, so hat *tabspez* standardmäßig den Wert *-8*. Der Wert *-0* bedeutet, daß bei der Ausgabe keine Leerzeichen in Tabulatorzeichen umgewandelt werden sollen.

**-l**n
Die effektive Zeilenlänge wird auf *n* Zeichen festgelegt (Standard: 72 Zeichen). Fehlt *-l*, so beträgt die Standard-Zeilenlänge 80 Zeichen. Zu beachten ist, daß Tabulator- und Rücksetzungszeichen (backspaces) als jeweils ein Zeichen interpretiert werden (mit dem Schalter *-i* können Tabulatorzeichen auf eine bestimmte Zahl von Leerzeichen erweitert werden).

**-b**n
Die Zeile wird an ihrem Anfang um *n* Zeichen gekürzt, wenn ihre Länge die effektive Zeilenlänge (siehe *-ln*) übersteigt. Standardmäßig wird die Zeile gerade so weit gekürzt, daß sie die effektive Zeilenlänge erhält; hierzu muß der Schalter *-b* ohne die Angabe *n* gesetzt sein. Dieser Schalter kann verwendet werden, um die sequence numbers aus einem COBOL-Programm wie folgt zu löschen:

```
newform -l1 -b7 file-name
```

*l1* muß angegeben werden, um die effektive Zeilenlänge kürzer zu setzen als jede in der Datei existierende Zeile, so daß der Schalter *-b* aktiv wird.

**-e**n
Wirkt wie *-bn*, nur wird die Zeile an ihrem Ende gekürzt.

**-c**k
Das Präfix-/Suffix-Zeichen wird zu *k* geändert Standard: Leerzeichen.

**-p**n
Einer Zeile werden *n* Zeichen (siehe *-ck*) vorangestellt, wenn ihre Länge unter der effektiven Zeilenlänge liegt. Standardmäßig werden einer Zeile gerade so viele Zeichen vorangestellt, daß ihre Länge der effektiven Zeilenlänge entspricht.

**-a**n
Wirkt wie *-pn*, nur werden die Zeichen am Ende der Zeile eingefügt.

**-f**

Die Zeile, die *tabspez* enthält, wird vor allen Ausgabezeilen ausgegeben. *tabspez* entspricht dabei dem beim letzten *-o*-Schalter angegebenen Format. Ist *-o* nicht gesetzt, so hat *tabspez* den Standard-Wert *-8*.

**-s**

Die vorangestellten Zeichen in jeder Zeile werden bis zum ersten Tabulatorzeichen abgeschnitten; bis zu 8 der gekürzten Zeichen werden ans Zeilenende gesetzt. Werden mehr als 8 Zeichen abgeschnitten (das erste Tabulatorzeichen nicht mitgezählt), so wird das achte Zeichen durch ein * ersetzt; die darauffolgenden Zeichen werden gelöscht. Das erste Tabulatorzeichen wird in jedem Fall gelöscht.

Wird dieser Schalter bei einer Datei verwendet, in der nicht alle Zeilen ein Tabulatorzeichen enthalten, so wird eine Fehlermeldung abgegeben und das Kommando beendet. Die gekürzten Zeichen werden intern gespeichert, bis sämtliche der übrigen Schalter auf diese Zeile angewandt worden sind. Daraufhin werden die Zeichen ans Ende der bearbeiteten Zeile gesetzt.

Im folgenden Beispiel soll eine Datei, deren Textzeilen Ziffern und ein oder mehrere Tabulatorzeichen vorangestellt sind, konvertiert werden; am Anfang der resultierenden Datei soll der Text stehen, alle Tabulatorzeichen außer dem ersten sollen zu Leerzeichen expandiert werden, und die Zeilen sollen gegebenenfalls bis zu Spalte 72 mit Leerzeichen aufgefüllt (bzw. bei Spalte 72 abgeschnitten) werden; die zuvor vorangestellten Ziffern sollen ab Zeile 73 stehen. Hierzu ist folgendes Kommando notwendig:

```
newform -s -i -l -a -e dateiname
```

## OPERANDEN

datei...

Name der zu bearbeitenden Datei(en).

keine Angabe

*newform* liest von Standard-Eingabe.

## FEHLER

*usage: ...*
*newform* wurde mit einem ungültigen Schalter aufgerufen.

*not -s format*
Eine der Zeilen enthält kein Tabulatorzeichen.

*can't open file*
Datei kann nicht geöffnet werden.

*internal line too long*
Eine Zeile enthält nach ihrer Erweiterung mehr als 512 Zeichen im internen Arbeitspuffer.

*tabspec in error*
Eine Tabulatordefinition hat ein unkorrektes Format, oder die Tabulatorstops werden nicht in aufsteigender Folge angegeben.

*tabspec indirection illegal*
Ein *tabspez*, das aus einer Datei (oder der Standard-Eingabe) eingelesen wurde, darf kein *tabspez* enthalten, das auf eine andere Datei (bzw. Standard-Eingabe) Bezug nimmt.

## ENDESTATUS

0 bei Erfolg
1 Fehler

## HINWEIS

- Sind die Schalter *-i* und *-o* gesetzt, so berücksichtigt *newform* auch Rücksetzungszeichen (backspaces), um die Tabulatorstops in die korrekten logischen Spalten zu setzen.
- Soll ein *tabspez* von der Standard-Eingabe gelesen werden (Schalter *-i--* oder *-o--*), so gibt *newform* keine Eingabeaufforderung aus.

– Wenn der Schalter *-f* gesetzt ist und der zuletzt verwendete *-o*-Schalter *-o--* war und diesem entweder ein *-o--* oder ein *-i--* voranging, so ist die Formatzeile mit der Tabulatordefinition unkorrekt.

## BEISPIEL

Alle Zeilen aus *datei*, die länger als 6 Zeichen sind, werden rechts um drei Zeichen gekürzt.

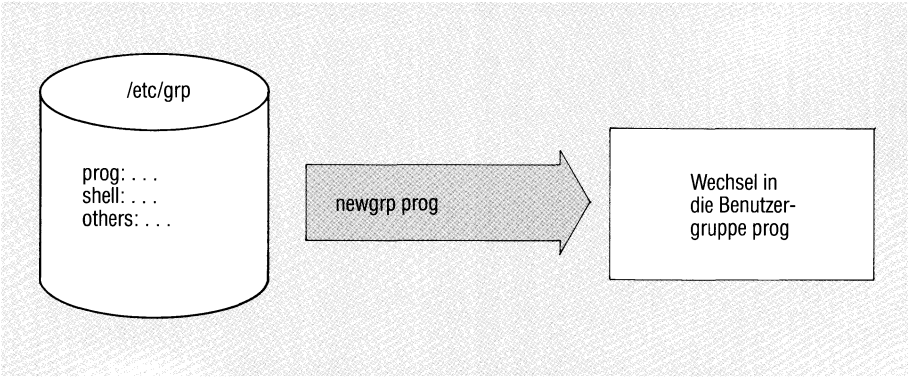
```
$ cat datei
montag
dienstag
mittwoch
donnerstag
freitag
samstag
sonntag
$ newform -l6 -e3 datei
montag
diens
mittw
donners
frei
sams
sonn
```

## SIEHE AUCH

*csplit*(1), *tabs*(1)

## NAME

**newgrp** - Benutzergruppe wechseln (change to a new group)




## DEFINITION

**newgrp**␣gruppe

## BESCHREIBUNG

Mit *newgrp* können Sie in eine andere Benutzergruppe wechseln. Das wirkt sich aus auf:

- Ihre Zugriffsrechte für bestehende Dateien und
- die Gruppenidentifikation für Dateien, die Sie neu anlegen.

In eine andere Gruppe wechseln können Sie nur, wenn Ihre Kennung in der Datei */etc/group* als berechtigt eingetragen ist.

! Die Shell führt das Kommando *newgrp* direkt aus, überschreibt aber den alten Prozeß. Das heißt, *newgrp* wirkt bis zum Ende der Sitzung oder bis erneut das Kommando *newgrp* eingegeben wird.

## OPERANDEN

gruppe
: Gruppenname, der in der Datei */etc/group* festgelegt ist. Nicht die Gruppennummer.

## DATEIEN

*/etc/group*
: Datei mit Gruppennamen

*/etc/passwd*
: Datei mit Kennwörtern

## BEISPIEL

Wechseln in die Gruppe mit dem Gruppennamen *consul*:

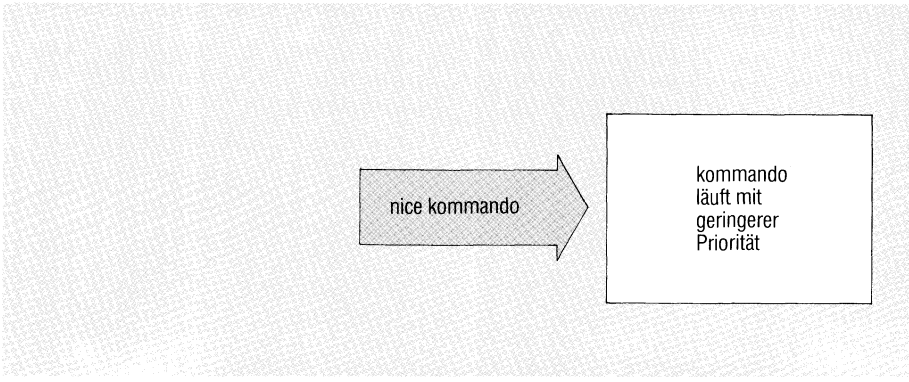
```
newgrp consul
```

## SIEHE AUCH

*sh*(1)

## NAME

**nice** - Priorität von Kommandos ändern




## DEFINITION

**nice**[␣-zahl]␣kommando[␣arg ...]

## BESCHREIBUNG

Das Kommando *nice* bewirkt, daß *kommando* mit einer niedrigeren Priorität ausgeführt wird. Mit *-zahl*, die im Bereich 1 bis 19 liegen kann, geben Sie den Betrag an, um den die Prioritätsangabe erhöht werden soll. *kommando* läuft dann mit niedrigerer Priorität ab.

Hoher Zahlenwert = niedrige Priorität
Niedriger Zahlenwert = hohe Priorität

Der Systemverwalter kann eine negative Zahl angeben (z.B. --10) und damit die Priorität eines Kommandos erhöhen.

Geben Sie für *zahl* einen Wert größer 19 an, so wird die Prioritätsangabe um den Betrag 19 erhöht bzw. erniedrigt.

## ENDESTATUS

*nice* liefert den Ende-Status des ausgeführten *kommando*s zurück.

## BEISPIEL

Die Shell-Prozedur *auftrag* soll im Hintergrund und mit verminderter Priorität ablaufen:

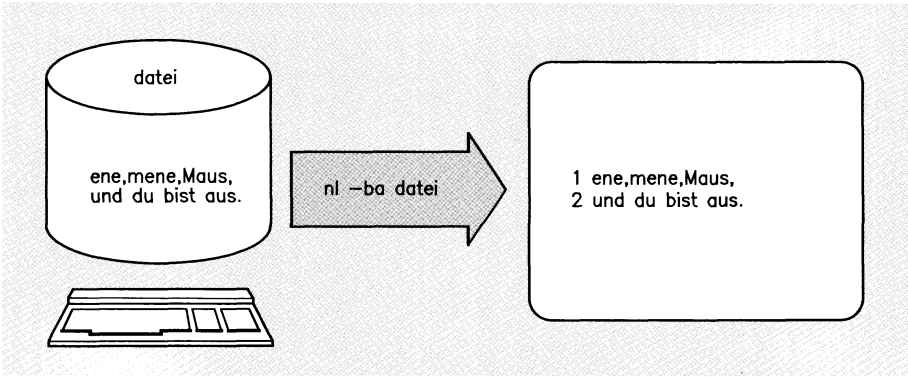
```
$ nice -15 auftrag&
345
$
```

## SIEHE AUCH

*nohup*(1), *nice*(2)

## NAME

**nl** - Textzeilen numerieren (line numbering filter)




## DEFINITION

**nl**[␣schalter][␣datei]

## BESCHREIBUNG

Mit dem Kommando *nl* können Sie Zeilen in Dateien oder Zeilen, die Sie von Standard-Eingabe eingeben, numerieren lassen.

*nl* unterteilt den eingelesenen Text in logische Seiten. Zu Beginn jeder logischen Seite wird der Zeilenzähler zurückgesetzt. Eine logische Seite besteht aus einem Kopfteil, einem Hauptteil sowie einem Fußteil. Jeder dieser Teile kann auch leer sein. Für die Zeilennumerierung der drei Teile einer logischen Seite können Sie verschiedene Schalter setzen; so können Sie z.B. festlegen, daß im Kopf- und Fußteil alle Zeilen, im Hauptteil dagegen nur die leeren Zeilen numeriert werden sollen.

Der Beginn einer logischen Seite wird angezeigt durch eine Eingabezeile, die ausschließlich das (die) folgende(n) Begrenzungszeichen enthält:

| Zeile | Beginn von |
|---|---|
| \:\:\: | Kopfteil |
| \:\: | Hauptteil |
| \: | Fußteil |

Sofern Sie nichts anderes angegeben haben, interpretiert *nl* den eingelesenen Text vollständig als Hauptteil einer logischen Seite.

## SCHALTER

Die Reihenfolge der Schalter ist beliebig; zwischen den Schaltern kann auch ein optionaler Dateiname stehen.

**-b**typ
Mit diesem Schalter geben Sie an, welche Zeilen im Hauptteil einer logischen Seite numeriert werden sollen. Für *typ* können Sie folgendes einsetzen:

**a**
Numerierung aller Zeilen;

**t**
Nur die Zeilen sollen numeriert werden, die druckbare Zeichen enthalten;

**n**
Keine Numerierung;

**p**regulärer ausdruck
Nur die Zeilen sollen numeriert werden, die eine zu *regulärer ausdruck* passende Zeichenkette enthalten. Siehe Tabelle *Reguläre Ausdrücke* im Anhang.

Standard: **t**

**-h**typ
Wie *-btyp*, nur für den Kopfteil. Standardmäßig werden die Zeilen nicht durchnumeriert (*typ* gleich *n*).

**-f**typ
Wie *-btyp*, nur für den Fußteil. Standardmäßig werden die Zeilen nicht numeriert (*typ* gleich *n*).

**-p**
Der Zeilenzähler wird am Anfang einer logischen Seite nicht zurückgesetzt.

**-v**startnum
Der Wert, auf den der Zeilenzähler zu Beginn einer logischen Seite zurückgesetzt werden soll (Standard: 1).

**-i**diff
Die Differenz zwischen zwei Zeilennummern (Standard: 1).

**-s**trenn
Das bzw. die Zeichen, mit dem bzw. mit denen die Zeilennummern und die zugehörigen Textzeilen voneinander getrennt werden sollen (Standard: Tabulatorzeichen).

**-w**breite
Die Zahl der Zeichen in der Zeilennummer (Standard: 6).

**-n**format
Format der Zeilennummern. Für *format* können Sie angeben:

**ln**
linksbündig, führende Nullen werden unterdrückt.
**rn**
rechtsbündig, führende Nullen werden unterdrückt.
**rz**
rechtsbündig, führende Nullen werden nicht unterdrückt.

Standard: **rn**

**-l**n
Die Anzahl von aufeinanderfolgenden Leerzeilen, die als eine Zeile interpretiert werden sollen. So bewirkt z.B. die Angabe *-l2*, daß bei mehreren aufeinanderfolgenden Leerzeilen nur jede zweite Leerzeile numeriert wird (hierzu müssen die Schalter *-ha*, *-ba* und/oder *-fa* geeignet gesetzt sein.) Standard: Jede Leerzeile wird numeriert (*n* gleich 1).

**-d**xx
Anstelle der Standardzeichen \: zur Kennzeichnung des Beginns eines logischen Seitenteils können Sie zwei andere Zeichen definieren. Geben Sie nur ein Zeichen ein, so wird für die zweite Zeichenposition das Standard-Zeichen : verwendet. *-d* und die Begrenzungszeichen dürfen nicht durch ein Leerzeichen voneinander getrennt werden. Soll zu Beginn einer logischen Seite ein Gegenschrägstrich stehen, so müssen Sie zwei Gegenschrägstriche eingeben.

## OPERANDEN

datei
Name der Datei, deren Zeilen numeriert werden sollen.

keine Angabe
*nl* liest von Standard-Eingabe.

## BEISPIEL

1. Die Zeilen in der Datei *limmerick* sollen numeriert werden:

```
$ cat limmerick
Eine Frau aus Clausthal-Zellerfeld,
die täglich in den Keller fällt,
hält den Rekord
in diesem Sport,
weil sie von Tag zu Tag schneller fällt.
$ nl limmerick > limmerick1
$ cat limmerick1
1  Eine Frau aus Clausthal-Zellerfeld,
2  die täglich in den Keller fällt,
3  hält den Rekord
4  in diesem Sport,
5  weil sie von Tag zu Tag schneller fällt.
```

2. In *datei1* sollen alle Zeilen ab Zeile 10 durchnumeriert werden. Die Differenz zwischen aufeinanderfolgenden Zeilen beträgt 10.

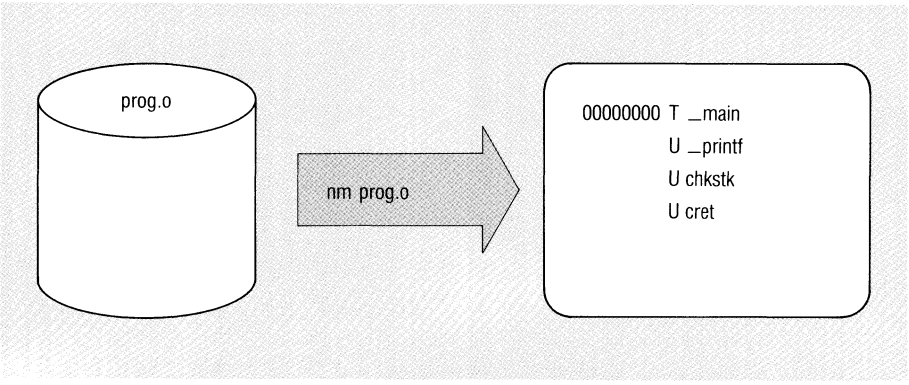
```
$ cat datei1
alfred
susi
hans
peter
helga
janine
petra
fritz
$ nl -v10 -i10 datei1 >datei2
10  alfred
20  susi
30  hans
40  peter
50  helga
60  janine
70  petra
80  fritz
$
```

## SIEHE AUCH

*pr*(1)

## NAME

**nm** - Symboltabelle einer Objektdatei ausgeben (print name list of an object file)




## DEFINITION

**nm**[␣schalter]␣datei...

## BESCHREIBUNG

*nm* gibt die Symboltabelle jeder angegebenen Objektdatei *datei* aus. Für *datei* können Sie eine relokierbare oder eine absolute Objektdatei bzw. eine Bibliothek mit relokierbaren oder absoluten Objektmodulen angeben. Für jedes Symbol gibt *nm* die folgenden Informationen aus:

- den Wert des Symbols, der, abhängig von der Speicherklasse, entweder eine relative oder eine absolute Adresse sein kann. Er kann aber auch leer sein (bei U).

- die Art des Symbols, die durch einen der folgenden Buchstaben ausgedrückt wird:

  U undefiniertes Symbol
  A absolutes Symbol
  T Textsegmentsymbol
  D Datensegmentsymbol
  B bss-Segmentsymbol
  F Dateinamensymbol

R Registername
C common-Symbol

Bei lokalen Symbolen wird der entsprechende Buchstabe klein, bei globalen groß geschrieben.

- den Symbolnamen.

## SCHALTER

**-o**
Der Wert der Symbole wird nicht dezimal, sondern oktal ausgegeben.

**-x**
Der Wert der Symbole wird nicht dezimal, sondern hexadezimal ausgegeben.

**-e**
Nur externe und statische Symbole werden ausgegeben.

**-f**
Vollständige Ausgabe ("full output"). Auch normalerweise unterdrückte Symbole (z.B. lokale Labels) werden ausgegeben.

**-u**
Nur undefinierte Symbole werden ausgegeben.

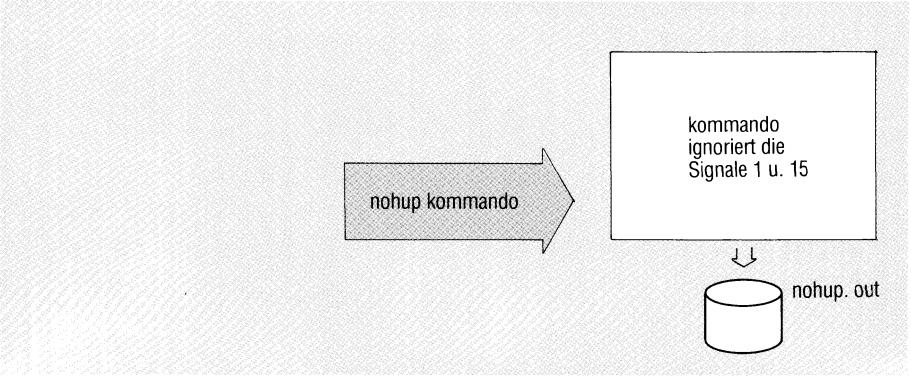
**-O**
*nm* gibt vor jedem Symbol den zugehörigen Modulnamen aus. Dieser Schalter ist vor allem nützlich, wenn Sie beim *nm*-Aufruf eine Bibliothek angeben.

## SIEHE AUCH

*cc*(1D), *ld*(1D).

## NAME

**nohup** - Kommando ausführen und dabei Signale ignorieren




## DEFINITION

**nohup**␣kommando[␣arg...]

## BESCHREIBUNG

*nohup* führt *kommando* so aus, daß es auf die Signale **SIGHUP** und **SIGQUIT** nicht reagiert. Wenn Sie die Ausgabe nicht umleiten, so wird die Standard-Ausgabe ebenso wie die Standard-Fehlerausgabe in *nohup.out* gespeichert. Ist *nohup.out* im aktuellen Dateiverzeichnis schreibgeschützt, so wird die Ausgabe nach *$HOME/nohup.out* umgeleitet.

Sie können dem Kommando Argumente mitgeben, indem Sie sie wie gewöhnlich hinter den Kommandonamen schreiben.

## OPERANDEN

kommando
ein beliebiges Kommando oder eine Shell-Prozedur

arg...
Argumente von *kommando*

## DATEIEN

*nohup.out*
Ausgabedatei, sofern Sie die Ausgabe nicht umgeleitet haben.

## HINWEIS

Es ist häufig wünschenswert, *nohup* auf Pipelines oder Kommandolisten anzuwenden. Hierzu müssen die betreffenden Pipelines oder Kommandolisten in einer einzelnen Datei gespeichert werden. Diese Prozedur kann als *Kommando* ausgeführt werden; *nohup* wird dann auf jedes Element der Datei angewandt.

*nohup* wird gewöhnlich angewandt, um zu verhindern, daß ein Programm beendet wird, wenn der Benutzer sich vom System abmeldet.

## BEISPIEL

Sie starten folgenden Auftrag:

```
nohup programm
```

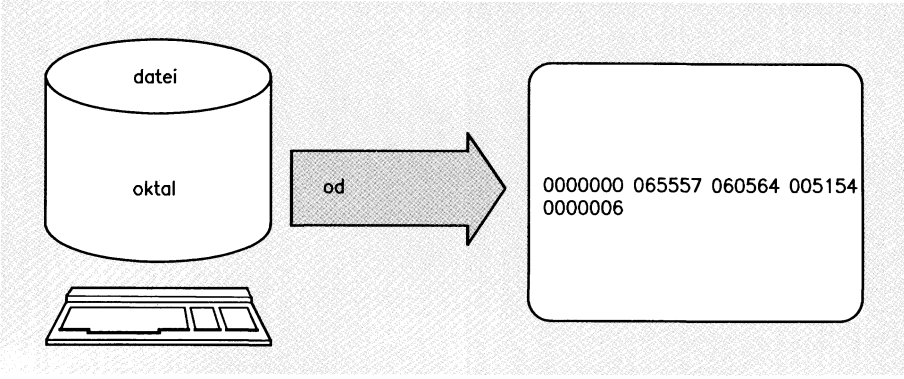
Dann melden Sie sich vom System ab. *programm* wird trotzdem nicht abgebrochen. Die Ausgabe wird in die Datei *nohup.out* geschrieben.

## SIEHE AUCH

*kill*(1), *sh*(1), *signal*(2)

## NAME

**od** - Inhalt einer Datei oktal ausgeben (octal dump)




## DEFINITION

**od**[␣schalter][␣datei][␣[+]offset[.][**b**]]

## BESCHREIBUNG

*od* schreibt den Inhalt von *datei* auf die Standard-Ausgabe; für die Ausgabe können Sie über Schalter ein oder mehrere Formate angeben. Die Ausgabe wird mit einem Dateiendezeichen beendet.

Im folgenden bezieht sich *wort* auf eine 2-Byte-Einheit.

## SCHALTER

**-b**

oktale Bytes.

**-c**

ASCII-Zeichen. Nicht-druckbare Zeichen werden gemäß den in der C-Sprache gültigen Konventionen folgendermaßen ausgegeben:

```
NUL  = \0,
BS   = \b,
FF   = \f,
NL   = \n,
```

```
CR    = \r,
HT    = \t;
```

die übrigen nicht druckbaren Zeichen werden in Form von 3 Oktalziffern ausgegeben.

**-d**
*worte* dezimal ohne Vorzeichen.

**-o**
*worte* oktal ohne Vorzeichen.

**-s**
*worte* dezimal mit Vorzeichen.

**-x**
*worte* hexadezimal ohne Vorzeichen.

Standard (keine Angabe): oktal ohne Vorzeichen.

## OPERANDEN

offset
Mit dem Argument *offset* wird angegeben, ab welcher Stelle in der Datei mit der Ausgabe begonnen werden soll. Dieses Argument wird normalerweise als oktale Zahl interpretiert. Folgt diesem Argument ein . (Punkt), so wird es als Dezimalzahl interpretiert.

**[+]b**
*b* bewirkt, daß die Distanz in Blöcken zu je 512 Bytes gezählt wird. Fehlt das Argument *datei*, so muß *offset* ein + vorangestellt werden.

datei
Name der Eingabedatei.

keine Angabe für datei
*od* liest von Standard-Eingabe.

## BEISPIEL

Oktale Ausgabe des Inhalts der Datei *text*:

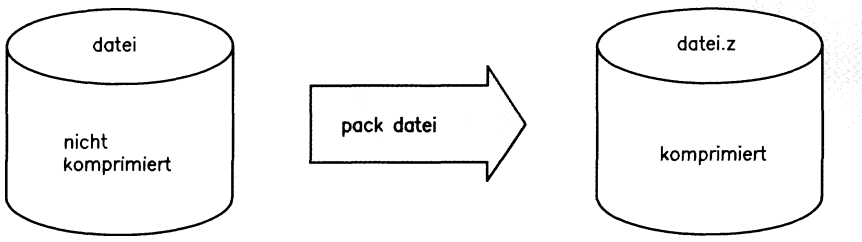
```
$ cat text
Noch kann man alles verstehen.
$ od text
00000000 067516 064143 065440 067141 020156 060555 020156 066141
00000020 062554 020163 062566 071562 062564 062550 027156 000012
0000037
$
```

## SIEHE AUCH

ASCII-Tabelle im Anhang.

## NAME

**pack** - Dateien komprimieren




## DEFINITION

**pack**[␣-][␣-f]␣datei...

## BESCHREIBUNG

*pack* versucht, die angegebenen Dateien in einer komprimierten Form abzuspeichern. Soweit möglich (und sinnvoll), wird jede Eingabedatei *datei* durch eine komprimierte Datei *datei.z* ersetzt; die komprimierte Datei hat denselben Zugriffsmodus, dasselbe Änderungsdatum und denselben Eigentümer wie *datei*. Nach erfolgreicher Durchführung von *pack* wird *datei* gelöscht. Mit *unpack* oder *pcat* können Sie *datei* wiederherstellen.

Der Umfang der Komprimierung hängt von der Größe der Eingabedatei und der Verteilung der Häufigkeit des Auftretens einzelner Zeichen ab. Da der erste Teil einer *.z*-Datei aus Codierungsregeln bestehen kann, lohnt es sich gewöhnlich nicht, kleine Dateien zu komprimieren, es sei denn, die Datei hat eine sehr ungleichmäßige Zeichendichte (z.B. wenn sie Graphiken oder Abbildungen enthält).

Die Dateien werden in der Regel um 35-40% komprimiert. Die komprimierten Versionen von Objektdateien, bei denen ein umfangreicherer Zeichensatz zur Anwendung kommt und die eine gleichmäßigere Zeichendichte aufweisen, sind lediglich um 10% kleiner als die Originaldateien.

[!] Komprimierte Dateien sind nicht in jedem Fall auf andere Systeme portierbar.

## SCHALTER

**-**

Der Schalter - bewirkt, daß auf Standard-Ausgabe statistische Informationen über die erfolgte Komprimierung ausgegeben werden.

**-f**

Die Komprimierung von *datei* wird erzwungen; auf diese Weise kann *pack* auf ein ganzes Dateiverzeichnis angewandt werden, selbst wenn bei einigen der darin enthaltenen Dateien kein Speicherplatz gespart wird.

## OPERANDEN

datei

Name der Datei, die komprimiert werden soll. Der Name darf höchstens 12 Zeichen lang sein, damit die Namenserweiterung auf *datei.z* noch möglich ist.

Nicht möglich ist eine Komprimierung, wenn

- *datei* offensichtlich bereits komprimiert ist;
- der Name von *datei* aus mehr als 12 Zeichen besteht;
- auf die *datei* Verweise vorhanden sind;
- *name* ein Dateiverzeichnis ist;
- *datei* nicht geöffnet werden kann;
- *datei* leer ist;
- die Komprimierung keine Platzeinsparung mit sich bringt;
- eine Datei mit dem Namen *datei.z* bereits existiert;
- die *.z*-Datei nicht erstellt werden kann oder
- während der Durchführung des Kommandos ein E/A-Fehler auftrat.

## ENDESTATUS

Der Endestatus von *pack* gibt an, wieviele Dateien nicht komprimiert werden konnten.

## BEISPIEL

Die Datei *dat*, die in unkomprimiertem Zustand 5226 Byte belegt, wird komprimiert.

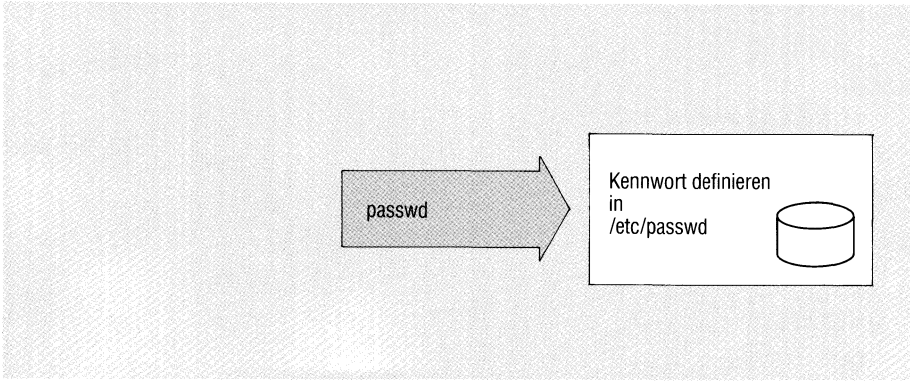
```
$ ls -l
total 8
-rw-------  1  fritz    gruppe1    5226 Aug 19 09:27 dat

$ pack dat

$ ls -l
total 8
-rw-------  1  fritz    gruppe1    2473 Aug 19 09:27 dat
```

## SIEHE AUCH

*pcat*(1), *unpack*(1)

## NAME

**passwd** - Kennwort für Benutzerkennung eintragen oder ändern (change login password)




## DEFINITION

**passwd**[␣name]

## BESCHREIBUNG

Mit *passwd* können Sie Ihr loginKennwort ändern. Der Systemverwalter kann auch die Kennwörter von anderen Benutzern ändern.

*passwd* fragt Sie zunächst nach Ihrem alten Kennwort (old password:), falls sie eines hatten, dann zweimal nach dem neuen Kennwort (new password:). Ihre Eingabe wird weder beim neuen, noch beim alten Kennwort auf dem Bildschirm ausgegeben.

*passwd* überprüft, ob das neue Kennwort das korrekte Format hat. Die zweite Eingabe des neuen Kennwortes wird mit der ersten verglichen. Werden Unterschiede festgestellt, so muß die Kennwort-Eingabe nochmals von vorne begonnen werden (maximal weitere zwei Durchgänge).

Der Systemverwalter kann ein beliebiges Kennwort ändern. Er braucht kein altes Kennwort eingeben, das neue Kennwort wird nicht auf das korrekte Format überprüft. Ein leeres Kennwort kann der Systemverwalter erzeugen, indem er statt eines Kennworts nur ↵ eingibt.

## OPERANDEN

name

Login-Name des Benutzers, dessen Kennwort geändert werden soll. Wenn Sie nicht der Systemverwalter sind, können Sie nur Ihren eigenen Login-Namen angeben. Standard (keine Angabe): Login-Name des Aufrufers von *passwd*.

## DATEIEN

*/etc/passwd*

Datei, in der die Kennwörter enthalten sind.

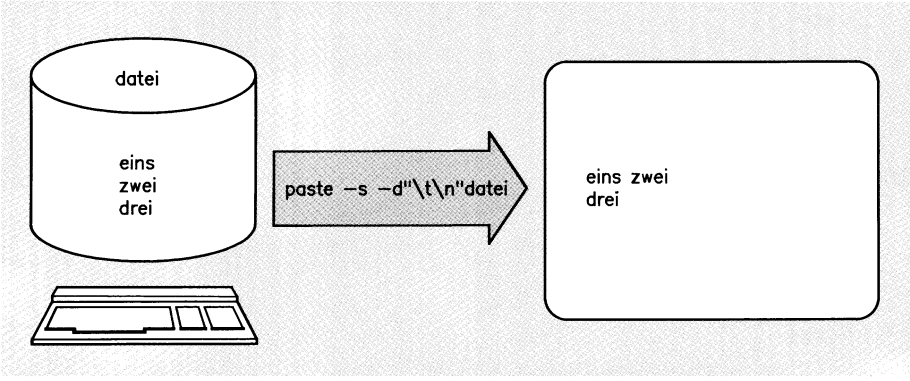
## BEISPIEL

Benutzer *aladin*, der am Rechner *alibaba* arbeitet, möchte sein Kennwort *sesam* ändern. Das neue Kennwort soll *mases* heißen:

```
$ passwd aladin
Changing local password for aladin on alibaba
Old password:           Blinde Eingabe von sesam
New password:           Blinde Eingabe von mases
Retype new password:    Nochmal blinde Eingabe von mases
$
```

## NAME

**paste** - Zeilen zusammenfügen




## DEFINITION

**paste**[␣**-d**list]␣datei...
**paste**␣**-s**[␣**-d**list]␣datei...

## BESCHREIBUNG

**1. Format: paste**[␣**-d**list]␣datei...

*paste* fügt Zeile *n* aus *datei1* und Zeile *n* aus *datei2* usw. zusammen. Jede Datei wird dann als eine oder mehrere Spalten einer Tabelle interpretiert; die Spalten werden nebeneinandergesetzt und auf Standard-Ausgabe geschrieben.

**2. Format: paste**␣**-s**[␣**-d**list]␣datei...

*paste* fügt aufeinanderfolgende Zeilen der Eingabedatei *datei* zusammen.

Bei beiden Formaten wird an die Nahtstellen zwischen den einzelnen Teil-Zeilen ein Tabulatorzeichen gesetzt, es sei denn, Schalter *-d* ist gesetzt (siehe unten).

Die resultierenden Zeilen werden auf die Standard-Ausgabe geschrieben.

## SCHALTER

**-d**list

An den Nahtstellen zwischen den einzelnen Teil-Zeilen steht nicht das Tabulatorzeichen, sondern ein Zeichen aus *list*. Die Zeichen in *list* werden nacheinander verwendet; ist *paste* beim letzten Zeichen angelangt, so geht es wieder an den Anfang der Liste. Ist der Schalter *-s* nicht gesetzt, werden die Zeilen aus der letzten Datei nicht mit einem Zeichen aus *list*, sondern mit einem Neue-Zeile-Zeichen abgeschlossen. In *list* können folgende Escape-Sequenzen enthalten sein: \n (Neue Zeile), \t (Tabulatorzeichen), \\ Gegenschrägstrich und \0 (Leere Zeichenfolge, nicht das Zeichen 0).
Diese Escape-Sequenzen müssen Sie in Anführungsstriche "..." einschließen.

**-s**

Die in den *dateien* aufeinanderfolgenden Zeilen werden miteinander verknüpft. An die Nahtstellen zwischen den einzelnen Zeilen werden Tabulatorzeichen gesetzt, es sei denn, mit dem Schalter *-d* wurde *list* mit anderen Zeichen angegeben. Das letzte Zeichen in der Datei ist jedoch in jedem Fall ein Neue-Zeile-Zeichen.

## OPERANDEN

datei...

Namen der Dateien, deren entsprechende Zeilen zusammengefügt werden sollen.

keine Angabe

*paste* liest von Standard-Eingabe

## BEISPIEL

1. Die entsprechenden Zeilen aus den Dateien *zahlen* und *buchstaben* werden gegenübergestellt.

```
$ cat zahlen          $ cat buchstaben
1                     a
2                     b
3                     c
4                     d
5                     e
6                     f
7                     g
8                     h
9                     i
```

```
10                      j
11                      k
12                      l
13                      m
14                      n
15                      o
16                      p
17                      q
18                      r
19                      s
20                      t
21                      u
22                      v
23                      w
24                      x
25                      y
26                      z

$ paste zahlen buchstaben
1       a
2       b
3       c
4       d
5       e
6       f
7       g
8       h
9       i
10      j
11      k
12      l
13      m
14      n
15      o
16      p
17      q
18      r
19      s
20      t
21      u
22      v
23      w
24      x
25      y
26      z
$
```

2. Mit dem folgenden Kommando wird der Inhalt des aktuellen Dateiverzeichnisses vierspaltig ausgegeben. Bündige Spalten erhalten Sie allerdings nur, wenn die Dateinamen nicht über den nächsten Tabulatorstop hinausreichen.

   ```
   $ ls | paste - - - -
   ```

   Die Ausgabe von *ls* wird zur Eingabe von *paste*, das diese in vier Spalten nebeneinanderschreibt.

3. Im folgenden Beispiel werden jeweils zwei Zeilen aus *datei* verknüpft, da das Verknüpfungszeichen nach der zweiten Zeile ein Neue-Zeile-Zeichen ist.

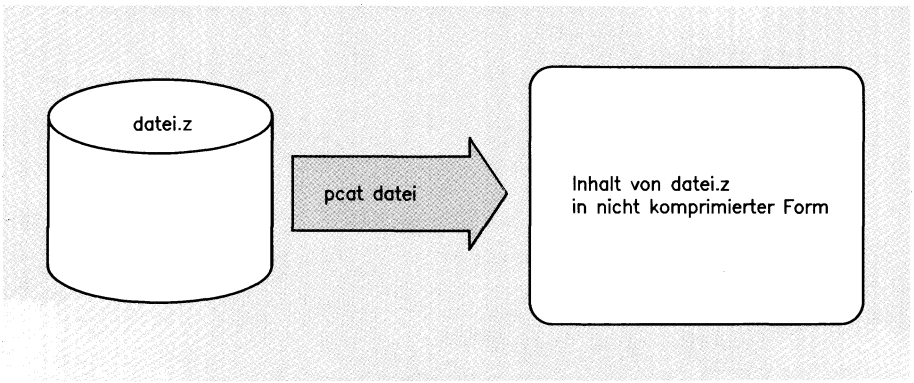
   ```
   $ paste -s -d"dt\n" datei
   ```

**SIEHE AUCH**

*cut*(1), *grep*(1), *pr*(1)

## NAME

**pcat** - komprimierte Dateien ausgeben




## DEFINITION

**pcat** datei...

## BESCHREIBUNG

*pcat* hat für komprimierte Dateien dieselbe Wirkung wie *cat*(1) für normale Dateien, nur kann *pcat* nicht als Filter verwendet werden. Die angegebenen Dateien werden wieder in den Originalzustand gebracht und auf Standard-Ausgabe geschrieben.

## OPERANDEN

datei

Name der Datei, deren Inhalt ausgegeben werden soll. Sie können entweder den Namen den komprimierten Datei oder den Namen der Originaldatei angeben:

```
pcat datei.z
   oder
pcat datei
```

Soll von einer komprimierten Datei namens *datei.z* eine Kopie in Originalgröße namens *abc* gemacht werden, ohne daß *datei.z* zerstört wird, so ist dies so möglich:

```
pcat datei >abc
```

## ENDESTATUS

*pcat* liefert die Zahl der Dateien, deren Originalzustand nicht wiederhergestellt werden konnte, zurück. Dies kann folgende Gründe haben:

- der Dateiname ohne *.z* besteht aus mehr als 12 Zeichen;
- die Datei kann nicht geöffnet werden oder
- die Datei ist offensichtlich nicht das Ergebnis von *pack*.

## BEISPIEL

Die Datei *liesmich* wird komprimiert zu Datei *liesmich.z* und mit *cat* und *pcat* wird versucht, sie zu lesen.

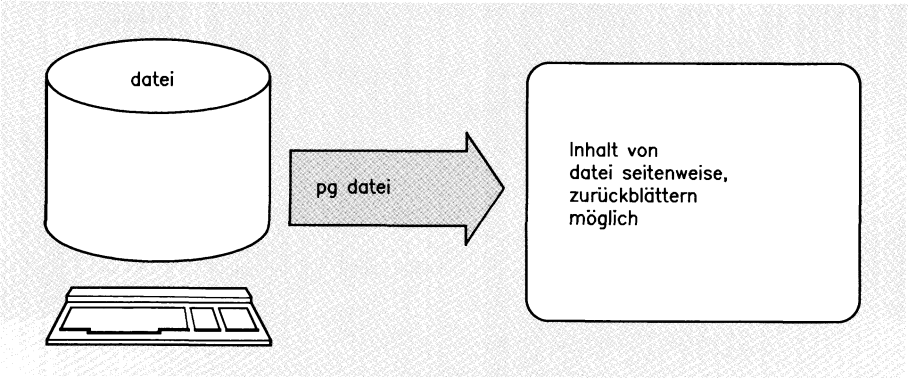
```
$ cat liesmich Noch bin ich nicht komprimiert!

$ pack -f liesmich pack: liesmich: -36.4% Compression

$ pcat liesmich.z Noch bin ich nicht komprimiert!
```

## SIEHE AUCH

*cat*(1), *pack*(1), *unpack*(1)

## NAME

**pg** - Dateien seitenweise ausgeben




## DEFINITION

**pg**[␣schalter][␣datei...]

## BESCHREIBUNG

Mit *pg* können Sie Dateien seitenweise auf einer Datensichtstation ausgeben lassen. Jeweils nach einer Seite gibt *pg* eine Eingabeaufforderung aus. Dies beantworten Sie mit einem Kommando (siehe unten *Kommandos*). Wenn Sie als Kommando die Taste ↵ drücken, wird am Bildschirm die nächste Seite ausgegeben. Seiten, die Sie bereits durchgeblättert haben, können Sie wieder zurückholen und überprüfen.

## SCHALTER

-zahl

Eine ganze Zahl, mit der Sie die gewünschte Größe (in Zeilen) einer Bildschirmseite angeben. Bei einer Datensichtstation mit maximal 24 Zeilen besteht eine Bildschirmseite standardmäßig aus 23 Zeilen.

**-p**zeichenkette

*pg* soll *zeichenkette* als Eingabeaufforderung verwenden. Ist in *zeichenkette* das Zeichen %d enthalten, so wird bei der Anzeige der Eingabeaufforderung das erste Auftreten von %d in *zeichenkette* durch die aktuelle Seitennummer ersetzt. Standardmäßig wird als Eingabeaufforderung ein Doppelpunkt : verwendet.

**-c**
Die Schreibmarke wird an den Anfang gesetzt, und der Bildschirm wird vor der Ausgabe jeder Seite gelöscht.

**-e**
*pg* macht am Ende einer Datei keine Pause.

**-f**
*pg* unterteilt normalerweise Zeilen, die über den rechten Bildschirmrand hinausgehen; sind im Text jedoch bestimmte Zeichenketten (z.B. die Escape-Sequenzen, mit denen Text unterstrichen werden kann), so kann dies zu unerwünschten Ergebnissen führen. Ist der Schalter *-f* gesetzt, so werden überlange Zeilen von *pg* nicht unterteilt.

**-n**
Normalerweise müssen Kommandos mit einem Neue-Zeile-Zeichen abgeschlossen werden. Ist der Schalter *-n* gesetzt, so gilt ein Kommando bereits nach dem ersten Buchstaben als vollständig eingegeben.

**-s**
Alle Nachrichten und Eingabeaufforderungen werden bei der Ausgabe mit inverser Schrift dargestellt.

+n
Die Ausgabe soll erst bei Zeile *n* beginnen.

+/rA/
Die Ausgabe soll bei der Zeile beginnen, die die erste zu dem regulären Ausdruck *rA* passende Zeichenkette enthält. Siehe Tabelle *Reguläre Ausdrücke* im Anhang.

## OPERANDEN

datei...
Name der Datei, die seitenweise auf Standard-Ausgabe ausgegeben werden soll.

**-**
*pg* liest von Standard-Eingabe.

keine Angabe
*pg* liest von Standard-Eingabe.

## KOMMANDOS

Die Eingabeaufforderung (Standard: Doppelpunkt) nach der Ausgabe einer Bildschirmseite beantworten Sie mit Kommandos, die im folgenden erläutert werden. Es gibt drei Möglichkeiten. Sie können:

- die Ausgabe fortsetzen
- nach einem bestimmten Ausdruck suchen
- die Umgebung ändern

Soll die Ausgabe fortgesetzt werden, so stellen Sie dem betreffenden Kommando eine *Adresse* (eine Zahl, auch mit Vorzeichen) voran. Mit dieser Adresse geben Sie die Stelle an, an der in der Datei die Ausgabe fortgesetzt werden soll. Je nach Kommando wird diese Adresse entweder als Seiten- oder als Zeilennummer interpretiert. Eine Zahl mit Vorzeichen gibt eine Position relativ zur aktuellen Seite bzw. Zeile, eine Zahl ohne Vorzeichen eine Position relativ zum Dateianfang an. Wenn Sie keine Adresse angeben, wird eine Standard-Adresse eingesetzt.

### Kommandos zur Fortsetzung der Ausgabe

[adresse]Neue-Zeile-Zeichen
Ausgabe der nächsten Bildschirmseite.
Standard für *adresse*: +1

[adresse]**l**
Bei Angabe einer relativen Adresse simuliert *pg* das Auf- oder Abwärtsrollen der Bildschirmseite um die Zahl der mit *l* angegebenen Zeilen. Bei Angabe einer absoluten Zeile gibt dieses Kommando die Bildschirmseite aus, die bei der angegebenen Zeile beginnt.
Standard für *adresse*: +1

[adresse]**d**
[adresse]**^D**
*pg* simuliert das Auf- oder Abwärtsrollen der Bildschirmanzeige um eine halbe Seite.
Standard für *adresse*: +1

.
**^L**
Die aktuelle Bildschirmseite wird nochmals ausgegeben.

**$**
Die letzte Seite der Datei wird ausgegeben. Vorsicht ist geboten, wenn die Eingabe eine Pipe ist.

### Kommandos zum Suchen nach Mustern im Text

Die Muster haben die Form von *einfachen regulären Ausdrücken*. (Siehe Tabelle *Reguläre Ausdrücke* im Anhang.) Ein *muster* muß durch ein Neue-Zeile-Zeichen abgeschlossen werden, unabhängig davon, ob der Schalter *-n* gesetzt ist.

i/rA/
: Das *i*-te (Standard: $i=1$) Auftreten einer zu dem regulären Ausdruck *rA* passenden Zeichenkette ab der nächsten Bildschirmseite wird gesucht. Beim Ende der aktuellen Datei wird die Suche beendet.

**i^**rA**^**
i**?**rA**?**
: Nach dem *i*-ten (Standard: $i=1$) Auftreten einer zu dem regulären Ausdruck *rA* passenden Zeichenkette wird ab der vorhergehenden Seite in Rückwärtsrichtung gesucht. Der Suchvorgang wird am Anfang der Datei beendet.

Nach Abschluß des Suchvorganges stellt *pg* normalerweise die gefundene Zeile oben auf dem Bildschirm dar. Ist dies nicht erwünscht, so können Sie zusätzlich *m* oder *b* an das Suchkommando anfügen; die Zeile wird dann in der Mitte bzw. unten am Bildschirm ausgegeben.

### Kommandos zum Verändern der Umgebung

i**n**
: Die *i*-te im *pg*-Kommando spezifizierte Datei wird durchgeblättert. *i* ist eine Zahl ohne Vorzeichen; Standard: 1.

i**p**
: Die *i*-te vorhergehende Datei wird angezeigt; *i* ist eine Zahl ohne Vorzeichen; Standard: 1.

i**w**
: Ein anderes Textfenster soll angezeigt werden. Mit *i* können Sie die Größe der Bildschirmseite bestimmen.

**s**datei
Die Eingabe soll in der Datei namens *datei* gespeichert werden. Nur die aktuell ausgegebene Datei wird gespeichert. Zwischen *s* und *datei* kann ein Leerzeichen stehen. Dieses Kommando muß in jedem Fall mit einem Neue-Zeile-Zeichen abgeschlossen werden, unabhängig davon, ob der Schalter *-n* gesetzt ist, oder nicht.

**h**
Eine Kurzerläuterung zu den verfügbaren Schaltern wird ausgegeben.

**q**
**Q**
*pg* wird beendet.

!kommando
*kommando* wird an den Kommandointerpreter übergeben, dessen Name über die Umgebungsvariable SHELL ermittelt wird. Ist diese Variable nicht vorhanden, so wird der Standard-Kommandointerpreter verwendet. Dieses Kommando muß in jedem Fall durch ein Neue-Zeile-Zeichen abgeschlossen werden, unabhängig davon, ob der Schalter *-n* gesetzt ist, oder nicht.

## HINWEIS

- Beim Empfang eines Abbruch- oder Unterbrechungssignals unterbricht *pg* die Anzeige der Bildschirmseiten und gibt die Eingabeaufforderung aus. Sie können dann ganz normal eines der obengenannten Kommandos eingeben. Leider gehen dabei Teile der Ausgabe verloren, da beim Empfang eines der genannten Signale alle Zeichen aus der Warteschlange der Datensichtstation entfernt werden.

- Ist die Standard-Ausgabe keine Datensichtstation, so verhält *pg* sich wie das Kommando *cat*; nur wird, falls mehrere Dateien angegeben werden, jeder Datei ein Kopfteil vorangestellt.

- Während es auf die Eingabe des Benutzers wartet, reagiert *pg* auf die Signale **SIGINT** und **SIGQUIT**, indem es sich beendet. Werden diese Signale jedoch zwischen zwei Eingabeaufforderungen empfangen, so bricht *pg* die Ausführung des aktuellen Kommandos ab und gibt eine Eingabeaufforderung aus. Diese Kommandos sollten nur mit Vorsicht verwendet werden, wenn *pg* die Eingabe aus einer

Pipe liest, da ein Unterbrechungssignal in den meisten Fällen auch meisten Fällen auch die anderen Kommandos in der Pipeline beendet.

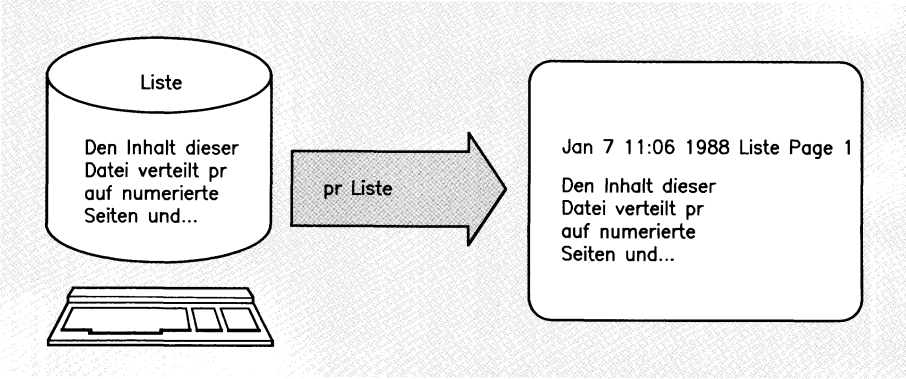
- Sind die Tabulatorstops auf der aktuellen Datensichtstation nicht bei jeder achten Zeichenposition gesetzt, so kann es zu einer fehlerhaften Ausgabe kommen.
- Wird *pg* in Kombination mit einem Kommando benutzt, durch das die Ein-/Ausgabe-Eigenschaften der Datensichtstation verändert werden, so werden die Einstellungen möglicherweise nicht mehr vollständig wiederhergestellt.

**SIEHE AUCH**

*grep*(1)

## NAME

**pr** - Dateien zum Ausdrucken aufbereiten (print files)




## DEFINITION

**pr**[␣-schalter][␣datei...]

## BESCHREIBUNG

Mit *pr* können Sie Dateien zum Ausdrucken aufbereiten und auf Standard-Ausgabe schreiben. *pr* teilt die Dateien standardmäßig in Seiten auf. Jede Seite hat eine Kopfzeile, die aus der Seitennummer, Datum und Uhrzeit sowie dem Namen der Datei besteht.

Standardmäßig haben alle ausgegebenen Spalten dieselbe Breite; die einzelnen Spalten sind zumindest durch ein Leerzeichen voneinander getrennt. Überlange Zeilen werden abgeschnitten.

Wenn *pr* auf eine Datensichtstation ausgibt, werden keine Fehlermeldungen ausgegeben, bis *pr* mit der Ausgabe fertig ist.

## SCHALTER

Die *schalter* können Sie einzeln angeben oder in beliebiger Reihenfolge kombinieren.

+k
Mit der Ausgabe wird bei Seite *k* begonnen (Standard: Seite 1).

-k
Nicht mit Schalter *-m* zu verwenden! Die Ausgabe soll aus *k* Spalten (Standard: 1 Spalte) bestehen. Bei einer mehrspaltigen Ausgabe werden die Schalter *-e* und *-i* vorausgesetzt.

**-a**
Nur zusammen mit Schalter *-k*! Mehrspaltiger Ausdruck.

**-m**
Schalter *-k* wird aufgehoben! Alle angegebenen *dateien* werden nebeneinander ausgegeben; in jeder Spalte ist eine Datei enthalten.

**-d**
Ausgabe mit doppeltem Zeilenabstand.

-eck
Die *eingegebenen* Tabulatorzeichen werden auf die Zeichenpositionen $k+1, 2*k+1, 3*k+1$ usw. erweitert. Wenn Sie für *k* den Wert 0 oder keinen Wert angeben, werden die Tabulatorstops an jeder achten Zeichenposition gesetzt. Wenn Sie für *c* ein beliebiges nichtnumerisches Zeichen angeben, wird es als Tabulatorzeichen interpretiert (Standardmäßig ist *c* das Tabulatorzeichen).

-ick
Bei der *Ausgabe* sollen aufeinanderfolgende Leerzeichen durch Tabulatorzeichen an den Stellen $k+1$, $2*k+1$, $3*k+1$ usw. ersetzt werden. Wenn Sie für *k* keinen Wert oder den Wert 0 angeben, wird von den Standard-Tabulatorstops (jede achte Zeichenposition) ausgegangen. Wenn Sie für *c* ein beliebiges nicht-numerisches Zeichen eingegeben, wird es als Ausgabe-Tabulatorzeichen interpretiert (Standard).

-nck
Die ausgegebenen Zeilen sollen durchnumeriert werden; die Zeilennummern sollen aus *k* (Standard: 5) Ziffern bestehen. Die Zeilennummern belegen die ersten $k+1$ Zeichenpositionen jeder Spalte auf einer normalen Ausgabe; ist der Schalter *-m* gesetzt, so belegen sie die ersten $k+1$ Zeichenpositionen jeder Zeile. Das gegebenenfalls für *c* angegebene Zeichen (ein beliebiges nicht-numerisches Zeichen) wird an die Zeilennummer angehängt, um sie von dem darauffolgenden Zeileninhalt zu trennen (standardmäßig wird hierzu ein Tabulatorzeichen verwendet).

**-wk**
Die Zeilen einer mehrspaltigen Ausgabe sollen aus *k* Zeichen bestehen. Bei gleicher Breite der Spalten bestehen die Zeilen standardmäßig aus 72 Zeichen; ansonsten gibt es keine Begrenzung.

**-ok**
Die Ausgabe soll *k* (Standard: 0) Zeichen vom linken Rand entfernt beginnen. Die Zahl der Zeichen pro Ausgabezeile ergibt sich durch Addition dieser und der mit Parameter *w* angegebenen Zahl.

**-lk**
Die ausgegebenen Seiten sollen eine Länge von *k* (Standard: 56) Zeilen haben. Würde *k* nicht einmal für den Kopf- und Fußteil jeder Seite ausreichen, so wird automatisch der Schalter *-t* gesetzt; das heißt, Kopf- und Fußteil werden nicht ausgegeben, um die Ausgabe des Hauptteils zu ermöglichen.

**-h**kopfzeile
*kopfzeile* wird anstelle des Dateinamens als Seitenüberschrift verwendet.

**-p**
Wird die Ausgabe auf eine Datensichtstation geschrieben, so macht *pr* zwischen den einzelnen Seiten eine Pause (beim Ertönen eines akustischen Signals muß die Wagenrücklauf-Taste betätigt werden).

**-f**
Am Ende einer Seite steht ein Seitenvorschub-Zeichen (Standard: Mehrere Zeilenvorschub-Zeichen). Ist die Standard-Ausgabe eine Datensichtstation, so wird vor dem Anfang der ersten Seite eine Pause gemacht.

**-r**
Es werden keine Fehlermeldungen abgegeben, wenn Dateien nicht geöffnet werden können.

**-t**
Weder der fünfzeilige Seitenkopf noch der fünfzeilige Nachspann wird ausgegeben. Nach der Ausgabe der letzten Zeile einer Datei wird *pr* sofort beendet; es erfolgt also kein Zeilenvorschub zum Ende der Seite.

**sc**
Überlange Zeilen werden nicht gekürzt und die ausgegebenen Spalten werden durch das für *c* angegebene Zeichen (Standard: Tabulatorzeichen) voneinander getrennt.

**OPERANDEN**

datei...
Name der Datei(en), die Sie zum Druck aufbereiten möchten.

-
*pr* liest von Standard-Eingabe.

keine Angabe
*pr* liest von Standard-Eingabe.

**BEISPIEL**

1. *datei1* und *datei2* sollen dreispaltig und mit doppeltem Zeilenabstand ausgegeben werden. Die Überschrift soll *Dateiliste* sein:

```
pr -3dh "Dateiliste" datei1 datei2
```

2. *datei1* wird in *datei2* geschrieben, wobei in den Spalten 10, 19, 28, ... Tabulatorstops gesetzt werden:

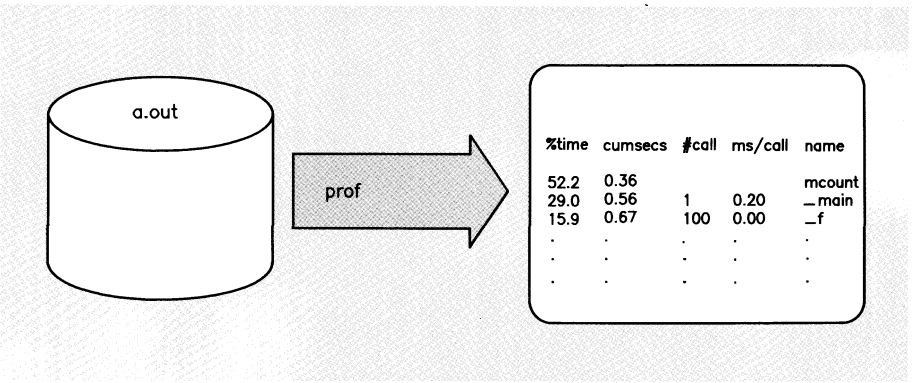
```
pr -e9 -t <datei1 >datei2
```

3. Die Datei *monate* soll in drei Spalten mit zweistelliger Numerierung ausgegeben werden.

```
$ cat monate
januar
februar
maerz
april
mai
juni
juli
august
september
oktober
november
dezember

$ pr -3 -n.2 monate

Aug 25 15:21 1987  monate Page 1

  1.januar          5.mai         9.september
  2.februar         6.juni        10.oktober
  3.maerz           7.juli        11.november
  4.april           8.august      12.dezember
```

## NAME

**prof** - Zeittabelle für ein Programm aufstellen (display profile data)




## DEFINITION

**prof**[␣schalter...][␣datei]

## BESCHREIBUNG

*prof* stellt eine Zeittabelle für ein bereits abgelaufenes C-Programm auf. Die Tabelle zeigt auf, wie das Programm seine CPU-Zeit verbraucht hat.

Um die Tabelle zu erstellen, wertet *prof* eine zum Programm gehörige Profildatei (i.a. *mon.out*) aus. Eine solche Profildatei wurde erstellt, wenn Sie das Programm mit *cc*(1D), Schalter *-p*, übersetzt haben.
Wenn Sie ein Programm mit *cc -p* übersetzen, dann wird beim Starten des Programms automatisch die C-Funktion *monitor*(3C) aufgerufen. Wenn das Programm normal beendet wird, erstellt die Funktion *monitor* eine Profildatei.

Der Name der erzeugten Profildatei ist vom Wert der Umgebungsvariablen *PROFDIR* abhängig. Ist diese Variable nicht gesetzt, dann wird im aktuellen Dateiverzeichnis die Profildatei *mon.out* erstellt. Ist *PROFDIR=zeichenfolge*, dann wird die Profildatei *zeichenfolge/pid.progname* erstellt; *progname* ist *argv[0]* ohne Zugriffspfad, und *pid* ist die Prozeßnummer des Programms. Ist der Variablen *PROFDIR* eine leere Zeichenfolge zugeordnet, dann wird keine Profildatei erstellt.

*prof* liest die Symboltabelle in der Objektdatei *a.out* bzw. *prog* und wertet damit die Profildatei aus. Auf der Standard-Ausgabe gibt *prof* für jedes externe Symbol (z.B. eine Funktion) eine Zeile mit folgenden Informationen aus:

- Zeit in Prozent, die das Symbol bei der Programmausführung verbraucht hat (Spalte *%time*),
- Zeit in Sekunden, die das Symbol und alle Symbole, die vorher aufgelistet wurden, verbraucht haben (Spalte *cumsecs*).
  Wieviel Zeit (in Sekunden) ein Symbol verbraucht hat, erfahren Sie, wenn Sie die Differenz bilden zwischen dieser Angabe und der Angabe aus der Zeile davor.
- Anzahl der Aufrufe des Symbols (Spalte *#call*),
- Zeit in Millisekunden, die pro Aufruf durchschnittlich verbraucht worden ist (Spalte *ms/call*),
- Symbolname (Spalte *name*).

Alle Zeiten sind CPU-Zeiten.

## SCHALTER

Mit den Schaltern *-t*, *-c*, *-a* und *-n* legen Sie fest, in welcher Reihenfolge *prof* die Zeilen mit den CPU-Informationen ausgibt. Diese Schalter dürfen Sie nicht miteinander kombinieren.

**-t**
*prof* sortiert die Ausgabezeilen nach der Ausführungszeit in Prozent in absteigender Folge (Standard).

**-c**
*prof* sortiert die Ausgabezeilen nach der Anzahl der Aufrufe in absteigender Folge.

**-a**
*prof* sortiert die Ausgabezeilen nach Symboladressen in aufsteigender Folge.

**-n**
*prof* sortiert die Ausgabezeilen alphabetisch nach dem Symbolnamen.

Geben Sie Schalter *-o* oder *-x* an, dann gibt *prof* die Adresse jedes behandelten Symbols aus. Diese beiden Schalter dürfen Sie nicht zusammen angeben.

**-o**
*prof* gibt jede Symboladresse oktal zusammen mit dem Symbolnamen aus.

**-x**
*prof* gibt jede Symboladresse hexadezimal zusammen mit dem Symbolnamen aus.

Die folgenden Schalter können Sie beliebig kombinieren.

**-g**
*prof* behandelt auch nicht-globale Symbole (statische Funktionen).

**-z**
*prof* behandelt alle Symbole, selbst wenn sie nicht aufgerufen wurden und daher keine Ausführungszeit beanspruchen.

**-m**␣mdata
*prof* wertet nicht *mon.out* aus, sondern die Profildatei *mdata*.

## OPERANDEN

datei
*datei* ist der Name eines Objektmoduls oder eines ausführbaren Programms. *prof* soll für dieses (bereits abgelaufene) Programm die Zeittabelle erstellen. *prof* nimmt die Symboltabelle von *datei*, um die Profildatei auszuwerten. Die Symboltabelle darf deshalb nicht fehlen.

*datei* muß von *cc*(1D) mit dem Schalter *-p* erstellt worden sein. Die Profildatei wurde dann automatisch nach der Programmausführung erstellt.
*Standard (keine Angabe):*a.out

## DATEIEN

*mon.out*
Profildatei.
*a.out*
enthält die Symboltabelle.

## HINWEIS

Wenn Sie *prof* mehrmals nacheinander für identische Programmabläufe aufrufen, dann kann die ausgegebene CPU-Zeit variieren. Das liegt daran, daß der Prozeß den Cache-Speicher mit anderen Prozessen teilen muß und bei den verschiedenen Abläufen nicht immer denselben Speicherplatz zugeteilt bekommt. Auch wenn es scheint, daß das Programm die Maschine allein benutzt, können versteckte Hintergrundprozesse oder asynchrone Prozesse die Daten beeinflussen.

In seltenen Fällen können die Meßintervalle genauso lang wie ein Schleifendurchgang sein. Die Meßergebnisse werden dadurch stark verfälscht. Die Anzahl der Aufrufe wird jedoch in jedem Fall korrekt registriert.

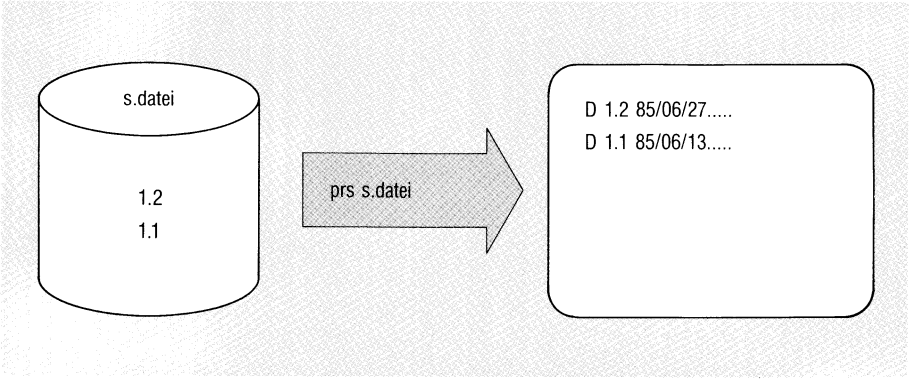
Nur Programme, die *exit*(2) aufrufen oder von *main* aus auf Systemebene zurückkehren, erzeugen mit Sicherheit eine Profildatei, es sei denn, der *monitor*-Aufruf am Schluß der Programmausführung ist explizit im Programm codiert.

## SIEHE AUCH

*cc*(1D), *exit*(2), *profil*(2), *monitor*(3C).

## NAME

**prs** - Informationen über eine SCCS-Datei ausgeben (print an SCCS file)




## DEFINITION

**prs**[␣schalter...]␣datei...

## BESCHREIBUNG

*prs* gibt auf der Standard-Ausgabe Teile einer SCCS-Datei oder die gesamte SCCS-Datei aus. Sie können das Format festlegen, in dem die Ausgabe erfolgen soll.

### SCCS-Daten-Schlüsselwörter

SCCS-Daten-Schlüsselwörter geben an, welche Teile einer SCCS-Datei *prs* ausgeben soll. Sie können jeden Teil einer SCCS-Datei mit einem passenden Daten-Schlüsselwort ansprechen.

*prs* ersetzt ein Daten-Schlüsselwort, das in *datspez* (siehe Schalter *-d*) vorkommt, durch seinen Wert. Das Format eines Wertes kann sein:

S *prs* gibt den Wert des Daten-Schlüsselworts aus ("simple"),

M *prs* gibt den Wert des Daten-Schlüsselworts und das Zeichen "Neue Zeile" aus ("multi-line").

## Tabelle der SCCS-Daten-Schlüsselwörter

| Daten-Schlüssel-wort | Bedeutung des Daten-Schlüsselworts | Teil der s-Datei, in dem der Wert steht | Wert | Format |
|---|---|---|---|---|
| :Dt: | Deltainformation (für ein bestimmtes Delta) | Delta-Tabelle | *) | S |
| :DL: | Deltazeilenstatistik | Delta-Tabelle | :Li:/:Ld:/:Lu: | S |
| :Li: | Zeilen, die das Delta ein gefügt hat | Delta-Tabelle | nnnnn | S |
| :Ld: | Zeilen, die das Delta gelöscht hat | Delta-Tabelle | nnnnn | S |
| :Lu: | Zeilen, die das Delta unverändert ließ | Delta-Tabelle | nnnnn | S |
| :DT: | Deltatyp | Delta-Tabelle | D oder R | S |
| :I: | SID-Nummer | Delta-Tabelle | :R:.:L:.:B:.:S: | S |
| :R: | Releasenummer | Delta-Tabelle | nnnn | S |
| :L: | Levelnummer | Delta-Tabelle | nnnn | S |
| :B: | Zweignummer (Branch) | Delta-Tabelle | nnnn | S |
| :S: | Folgenummer (Sequence) | Delta-Tabelle | nnnn | S |
| :D: | Datum, an dem das Delta erstellt wurde | Delta-Tabelle | :Dy:/:Dm:/:Dd: | S |
| :Dy: | Jahr, in dem das Delta erstellt wurde | Delta-Tabelle | nn | S |
| :Dm: | Monat, in dem das Delta erstellt wurde | Delta-Tabelle | nn | S |
| :Dd: | Tag, an dem das Delta erstellt wurde | Delta-Tabelle | nn | S |
| :T: | Uhrzeit, zu der das Delta erstellt wurde | Delta-Tabelle | :Th:::Tm:::Ts: | S |
| :Th: | Stunde, in der das Delta erstellt wurde | Delta-Tabelle | nn | S |
| :Tm: | Minute, in der das Delta erstellt wurde | Delta-Tabelle | nn | S |
| :Ts: | Sekunde, in der das Delta erstellt wurde | Delta-Tabelle | nn | S |

| Daten-Schlüs-sel-wort | Bedeutung des Daten-Schlüsselworts | Teil der s-Datei, in dem der Wert steht | Wert | For-mat |
|---|---|---|---|---|
| :P: | Benutzer, der das Delta erstellt hat | Delta-Tabelle | login-Name | S |
| :DS: | laufende Nummer des Deltas | Delta-Tabelle | nnnn | S |
| :DP: | laufende Nummer des Vorgängerdeltas | Delta-Tabelle | nnnn | S |
| :DI: | laufende Nummer der Deltas, die für das Delta nicht berücksich-tigt, ignoriert wurden | Delta-Tabelle | :Dn: / :Dx: / :Dg: | S |
| :Dn: : | laufende Nummer der Deltas, die berück-sichtigt wurden | Delta-Tabelle | :DS:␣:DS:␣... | S |
| :Dx: | laufende Nummer der Deltas, die nicht be-rücksichtigt wurden | Delta-Tabelle | :DS:␣:DS:␣... | S |
| :Dg: | laufende Nummer der Deltas, die ignoriert wurden | Delta-Tabelle | :DS:␣:DS:␣... | S |
| :MR: | MR-Nummern des Deltas | Delta-Tabelle | Text | M |
| :C: | Kommentar | Delta-Tabelle | Text | M |
| :UN: | Benutzerliste | Benutzerliste | Text | M |
| :FL: | Parameterliste | Parameter | Text | M |
| :Y: | Modultyp (*admin -ft*) | Parameter | Text | S |
| :MF: | *admin* wurde mit Schal-ter *-fv* aufgerufen? | Parameter | yes oder no | S |
| :MP: | Name des Programms, das die MR-Nummern prüft (*admin -fv*) | Parameter | Text | S |
| :KF: | *admin* wurde mit Schal-ter *-fi* aufgerufen? | Parameter | yes oder no | S |
| :BF: | *admin* wurde mit Schal-ter *-fb* aufgerufen? | Parameter | yes oder no | S |
| :J: | *admin* wurde mit Schal-ter *-fj* aufgerufen? | Parameter | yes oder no | S |
| :LK: | Gesperrte Releasenum-mern (*admin -fl*) | Parameter | :R:␣... **) | S |

| Daten-Schlüs-sel-wort | Bedeutung des Daten-Schlüsselworts | Teil der s-Datei, in dem der Wert steht | Wert | For-mat |
|---|---|---|---|---|
| :Q: | Text bei *admin -fq* | Parameter | Text | S |
| :M: | Modulname (*admin -fm*) | Parameter | Text | S |
| :FB: | minimale Releasenummer (*admin -ff*) | Parameter | :R: **) | S |
| :CB: | maximale Releasenummer (*admin -fc*) | Parameter | :R: **) | S |
| :Ds: | Voreinstellung der Deltanummer für *get* (*admin -fd*) | Parameter | :I: **) | S |
| :ND: | *admin* wurde mit Schalter *-fn* aufgerufen? | Parameter | yes oder no | S |
| :FD: | beschreibender Text | beschr. Text | Text | M |
| :BD: | verwalteter Text | verwalt. Text | Text | M |
| :GB: | verwalteter Versions text, mit *get* geholt | verwalt. Text | Text | M |
| :W: | Format einer *what*-Zeichenreihe | Parameter/ Delta-Tabelle | :Z::M:\t:I: | S |
| :A: | Format einer *what*-Zeichenreihe | Parameter/ Delta-Tabelle | ***) | S |
| :Z: | Zeichenreihe, die das Kommando *what* erkennt | kein Teil der s-Datei | @(#) | S |
| :F: | Name der SCCS-Datei | kein Teil der s-Datei | Text | S |
| :PN: | vollständiger Pfadname der SCCS-Datei | kein Teil der s-Datei | Text | S |

*) :DT:␣:I:␣:D:␣:T:␣:P:␣:DS:␣:DP

**) Diese Darstellung beschreibt nur das Format der Ausgabe. Sie können z.B. statt :FB: nicht :R: verwenden.

***) :Z::Y:␣:M:␣:I::Z:

**SCHALTER**

Die Reihenfolge der Schalter ist beliebig.

Kein Schalter:

*prs* gibt für alle Versionen aus:

- die Deltainformation :Dt: (siehe Tabelle der SCCS-Daten-Schlüsselwörter),
- die Anzahl der Zeilen, die eingefügt, gelöscht und nicht geändert wurden,
- die MR-Liste,
- den Kommentar.

**-d**[datspez]

Mit *datspez* bestimmen Sie das Ausgabeformat und spezifizieren,

- welchen Text und welche Daten *prs* in
- welcher Form ausgeben soll.

*prs* gibt aus:

- den Text, den Sie in *datspez* angeben,
- die Werte der Daten-Schlüsselwörter (siehe oben), die in *datspez* vorkommen. *prs* holt den jeweiligen Wert aus der SCCS-Datei und gibt ihn statt des Schlüsselworts aus.
- wenn *datspez* fehlt:
  - die Deltainformation :Dt:,
  - die Anzahl der Zeilen, die eingefügt, gelöscht und nicht geändert wurden,
  - die MR-Liste,
  - den Kommentar.

*datspez* ist eine Zeichenfolge, die bestehen kann aus:

- SCCS-Daten-Schlüsselwörtern und/oder
- beliebigem sonstigen ASCII-Text.

Daten-Schlüsselwörter können beliebig oft und an beliebiger Stelle in *datspez* vorkommen.

Im sonstigen Text bedeutet t Tabulatorzeichen und n "Neue Zeile".

Wenn *datspez* Leer- oder Tabulatorzeichen enthält, dann müssen Sie *datspez* in Anführungsstriche setzen: "*datspez*".

**-r**[SID]
*SID* ist die SID-Nummer des Deltas, über das Sie Information einholen wollen.

*Standard (keine Angabe):*
Die SID-Nummer der Version, die Sie als letztes erstellt haben.

**-e**
Zu diesem Schalter müssen Sie zusätzlich den Schalter *-r* oder *-c* angeben.

*Mit Schalter -r:*
*prs* gibt Informationen aus über

- die Version mit Nummer *SID* (siehe Schalter *-r*),
- alle Versionen, die Sie vor der Version mit der Nummer *SID* erstellt haben.

*Mit Schalter -c:*
*prs* gibt Informationen aus über

- alle Versionen, die Sie vor dem bei Schalter *-c* angegebenen Zeitpunkt erstellt haben.

**-l**
Zu diesem Schalter müssen Sie zusätzlich den Schalter *-r* oder *-c* angeben.

*Mit Schalter -r:*
*prs* gibt Informationen aus über

- die Version mit Nummer *SID* (siehe Schalter *-r*),
- alle Versionen, die Sie nach der Version mit der Nummer *SID* erstellt haben.

*Mit Schalter -c:*
*prs* gibt Informationen aus über

- alle Versionen, die Sie nach dem bei Schalter *-c* angegebenen Zeitpunkt erstellt haben.

-c[datum-zeit]
Zu diesem Schalter müssen Sie zusätzlich den Schalter *-e* oder *-l* angeben.
*prs* gibt Informationen aus über alle Versionen, die Sie

- vor (bei Schalter *-e*) bzw.
- nach (bei Schalter *-l*)

dem Zeitpunkt *datum-zeit* erstellt haben.
*datum-zeit* hat das folgende Format:

`JJ[MM[TT[hh[mm[ss]]]]]`

*JJ, MM, TT, hh, mm* und *ss* sind zweistellige Zahlen, die das Jahr, den Monat, den Tag, die Stunde, die Minute bzw. die Sekunde bezeichnen. Angaben, die fehlen, bekommen den maximal möglichen Wert. Zwischen die zweistelligen Zahlen können Sie beliebig viele nicht-numerische Zeichen (außer Leer- und Tabulatorzeichen) einfügen.

*Beispiel*

```
-c8702 ist gleichbedeutend mit -c870228235959,
-c87/9/3,09:22:25 ist gleichbedeutend mit -c870903092225.
```

**-a**
Normalerweise gibt *prs* nur Information aus über Deltas, die existieren (Deltatyp D) und nicht über Deltas, die *rmdel* gelöscht hat (Deltatyp R). Mit diesem Schalter bekommen Sie auch Informationen über Deltas vom Typ R.

## OPERANDEN

datei
Für *datei* können Sie den Namen einer s-Datei, eines Dateiverzeichnisses oder das Zeichen - angeben. *prs* behandelt den Namen eines Dateiverzeichnisses, wie wenn Sie die Namen aller Dateien des Verzeichnisses einzeln aufführen würden. *prs* ignoriert die Namen von Dateien, die keine s-Dateien sind oder für die Sie keine Leseberechtigung haben.

Wenn Sie statt *datei* das Zeichen - angeben, dann liest *prs* von der Standard-Eingabe. *prs* behandelt jede Eingabezeile als Name einer Datei oder eines Dateiverzeichnisses.

Sie können beliebig viele Namen angeben. Die gesetzten Schalter gelten dann für alle entsprechenden Dateien.

**DATEIEN**

/tmp/pr?????
Zwischendateien

**BEISPIEL**

1. *prs* mit Formatangabe

   Nach dem folgenden Kommando

```
$ prs -d"Die Datei :F: ist eine SCCS-Datei.\n:P:
  hat die neueste Version am :Dd:.:Dm:.19:Dy:
  erstellt.\nDie Deltainformation lautet:\n:Dt:"
  s.ente
```

   gibt *prs* aus:

```
Die Datei s.ente ist eine SCCS-Datei.
gudrun hat die neueste Version am 27.06.1985 erstellt.
Die Deltainformation lautet:
D 1.4 85/06/27 15:10:32 gudrun 7 6
```

2. *prs* ohne Formatangabe

   Um die Deltainformation aller Versionen zu bekommen, muß der Aufruf lauten:

```
$ prs -a s.ente
```

   *prs* gibt dann aus:

```
s.ente:

D 1.4 85/06/27 15:10:32 gudrun 7 6  00000/00004/00008
MRs:
COMMENTS:
nein

D 1.3 85/06/27 10:55:38 gudrun 6 5  00001/00000/00011
MRs:
COMMENTS:
keine

D 1.2 85/06/27 10:49:45 gudrun 5 1  00000/00000/00011
MRs:
COMMENTS:
peter piper picked a peck of pickled peppers
```

```
R 1.3 85/06/14 15:22:32 gudrun 4 3  00000/00000/00011
MRs:
COMMENTS:
"%Z%%M%%I%"

R 1.2 85/06/14 15:04:20 gudrun 3 1  00000/00000/00011
MRs:
COMMENTS:
so ein unsinn
*** CHANGED *** 85/06/14 15:09:33 gudrun
kein unsinn

R 1.2 85/06/13 17:18:14 gudrun 2 1  00000/00000/00011
MRs:
COMMENTS:
nix

D 1.1 85/06/13 16:42:20 gudrun 1 0  00011/00000/00000
MRs:
COMMENTS:
date and time created 85/06/13 16:42:20 by gudrun

$
```

3. Benutzerliste ausdrucken

   Um die Benutzerliste auszudrucken, können Sie folgendes Kommando eingeben:

```
$ prs -d"Benutzerliste der SCCS-Datei :F:\n:UN:" s.abc
Benutzerliste der SCCS-Datei s.abc:
gudrun
nepomuk
stanislaus
godwina
albertine
$
```

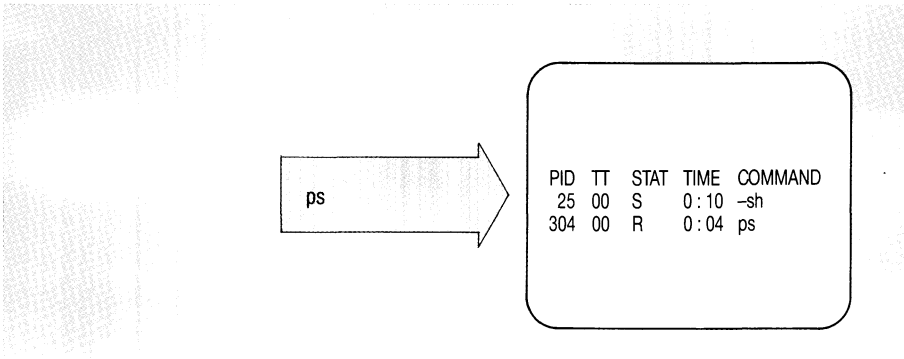
## SIEHE AUCH

*admin*(1D), *delta*(1D), *get*(1D), *what*(1D).

Eine Einführung ins SCCS ("Source Code Control System") finden Sie im Buch SINIX Einführung [1]. Dort sind auch alle Begriffe erklärt, die das SCCS betreffen.

## NAME

**ps** - Prozeßdaten abfragen (process status)




## DEFINITION

**ps**[␣schalter]

## BESCHREIBUNG

*ps* informiert Sie über Prozesse.

## SCHALTER

kein Schalter

*ps* gibt Informationen über Prozesse aus, die Ihre Benutzernummer haben und mit einer Datensichtstation verbunden sind. Die Ausgabe hat folgende Spalten: (Bedeutung siehe unten):

| PID | TT | STAT | TIME | COMMAND |
|---|---|---|---|---|
| Prozeß-nummer | Datensicht-station | Zustand | verbrauchte Zeit | Kommando |

**a**

listet alle Prozesse auf (auch die anderer Benutzer), die mit einer Datensichtstation verbunden sind.

**c**

gibt den Kommandonamen aus, so wie er systemintern für Abrechnungszwecke gespeichert ist, statt der Kommandoargumente, die im Adreßraum des Prozesses abgelegt sind. Dies ist zwar weniger informativ, dafür aber zuverlässiger, da ein Prozeß die Daten in seinem Adreßraum zerstören kann.

**e**

gibt zusätzlich zu den Kommandoargumenten die Prozeßumgebung aus. Bei Angabe zusammen mit *c* oder *v* wird *e* ignoriert.

**g**

listet alle Prozesse auf, auch Prozeßgruppenführer. D.h. es werden auch Kommandointerpreter und Prozesse ausgegeben, die bei freien Datensichtstationen auf ein Login warten. Solche Prozesse werden ohne diesen Schalter nicht aufgeführt.

**l**

gibt lange Liste aus, die zusätzlich folgende Spalten enthält:

```
F UID PPID CP PRI NI ADDR SIZE RSS WCHAN
```

(Bedeutung siehe unten)

**s**

gibt zusätzlich für jeden Prozeß die Spalte SSIZ aus, die die Größe des Systemkern-Kellers (Kernel-Stack) ausgibt. Nur für den Systemverwalter interessant!

**tm**

Immer als letzten Schalter angeben! listet nur Prozesse auf, deren kontrollierende Datensichtstation *m* ist.
*m* kann sein:

| | | |
|---|---|---|
| keine Angabe | : | aktuelle Datensichtstation |
| n | : | Nummer, z.B. 03 für tty03 |
| 00 | : | Konsole |
| d0 | : | ttyd0 |
| ? | : | Prozesse ohne Datensichtstation |

**u**

Benutzerorientierte Ausgabe. Es werden zusätzlich folgende Spalten ausgegeben:

`USER %CPU %MEM SIZE RSS`
(Bedeutung siehe unten)

**v**

Statistik über den virtuellen Speicher. Es werden zusätzlich folgende Spalten ausgegeben:

`SL RE PAGEIN SIZE RSS LIM TSIZ %CPU %MEM`
(Bedeutung siehe nächste Seiten)

**w**

Breites Ausgabe-Format (132 Zeichen statt 80).
Durch Wiederholung, z.B. ww, beliebige Breite.

**x**

Gibt auch Prozesse ohne Datensichtstation aus.

**#**

Immer als letzten Schalter angeben!
Für # müssen Sie eine Prozeßnummer angeben. Es wird dann nur Information für den Prozeß mit dieser Nummer ausgegeben.

## BEDEUTUNG DER SPALTEN

PID

Prozeßnummer. Unter dieser Nummer kann z.B. der Prozeß mit *kill* beendet werden.

TT

Datensichtstation, die den Prozeß steuert. Bei Prozeß ohne Datensichtstation, Ausgabe:?

STAT

Prozeßstatus: bis zu 4 Zeichen

1. Zeichen: Ablaufzustand

| | |
|---|---|
| R | Prozeß läuft |
| T | Prozeß ist angehalten |

D Prozeß wartet während Platten-/Band- o.ä. Ein-/Ausgaben
S Prozeß schläft < 20 Sek.
I Prozeß schläft > 20 Sek.

2. Zeichen: Ort des Prozesses

W Prozeß ausgelagert
␣ Prozeß im Speicher
> Speicherlimit überschritten (nicht ausgelagert)

3. Zeichen: Prozeß-Priorität

N Prozeß mit verminderter Priorität
␣ ohne Prioritäts-Änderung
< Prozeß mit erhöhter Priorität

TIME
Gesamte Ausführungszeit (Benutzer- und Systemzeit) des Prozesses in Minuten und Sekunden

COMMAND
Der Name des laufenden Kommandos

USER
Benutzername des Prozeßeigentümers

%CPU
CPU-Verbrauch des Prozesses

NICE oder NI
Wert, mit dem die Prozeßpriorität errechnet wird;

SIZE
Virtuelle Prozeß-Größe aufgerundet in KBytes;

RSS
Momentaner Prozeß-Hauptspeicherbedarf aufgerundet in KBytes;

LIM
Selbstgesetzte Speichergrenze; sonst xx;

TSIZ
Größe des Textsegments;

%MEM
Durch den Prozeß belegter Hauptspeicherplatz, in %;

RE
Residente Zeit des Prozesses in Sek;

SL
Zeit in Sek., die der Prozeß schon schläft;

PAGEIN
Plattenzugriffe des Prozesses für ausgelagerte Seiten;

UID
Benutzernummer des Eigentümers des Prozesses;

PPID
Prozeßnummer des Vaterprozesses;

CP
Prozessorzeit für das Scheduling;

PRI
Prozeßpriorität. Hohe Werte entsprechen niedriger Ablaufpriorität.

ADDR
Auslagerungsadresse des Prozesses;

WCHAN
Nummer des Ereignisses, auf das der Prozeß wartet; Ist die Spalte leer, läuft der Prozeß. Angabe ist für Normalbenutzer belanglos.

Weitere mögliche Spalten:

F
Flags des Prozesses (hexadezimal):

| | | |
|---|---|---|
| | 000000 | Prozeß wird erzeugt |
| SLOAD | 000001 | Prozeß im Speicher |
| SSYS | 000002 | Systemprozeß für Auslagerung (swapper) oder Paging (pager) |
| STRC | 000010 | Prozeß wird überwacht (traced) |
| SWTED | 000020 | weiteres Flag für Ablaufverfolgung (trace) |
| SWTOK | 000040 | während Ablaufverfolgung (trace) darf Textsegment beschrieben werden. |
| SOMASK | 000080 | nach Empfang eines Signals alte Maske wiedereinstellen |
| SVFORK | 000100 | Prozeß wurde durch *vfork* erzeugt |
| SNOVM | 000200 | kein virtueller Speicher; Vaterprozeß nach einem *vfork* |

| | | |
|---|---|---|
| STIMO | 000400 | Die Ruhezeit (sleep) ist abgelaufen. |
| SOUSIG | 000800 | Prozeß benutzt alten Signalmechanismus |
| SSEL | 001000 | wählt aus |
| SIGWOKE | 002000 | Signal weckte Prozeß auf |
| SWPSYNC | 004000 | synchron mit Swapp-Prozeß während Auslagerung |
| SFSWAP | 008000 | erzwingt Auslagerung |
| SSHMEM | 010000 | Dem Prozeß sind ein oder mehrere gemeinsam benutzte Speichersegmente zugeordnet. |

(Kombination von mehreren Angaben wird durch Addition der entsprechenden Flags angezeigt.)

## DATEIEN

*/dev/kmem*
Systemkernspeicher (Kernel-Speicher)

*/dev/drum*
swap-Datei

*/dev*
wird nach swap-Datei und Namen der Datensichtstationen durchsucht

## FEHLER

Wenn der Rechner stark belastet ist, kann es vorkommen, daß *ps* die Daten eines Prozesses gerade nicht zur Verfügung hat, da diese zwischen Swapbereich und Hauptspeicher transferiert werden. Es kommt dann zu folgenden Meldungen:

```
"Error locating name for PID ... from /dev/mem"
"Error locating name for PID ... from /dev/drum"
```

Am besten geben Sie in diesem Fall das Kommando *ps* erneut ein.

## HINWEIS

Es können weitere Argumente angegeben werden:

- Datei, die die Namensliste des Systems enthält.
- Name einer *swap*-Datei, wenn nicht */dev/drum*.

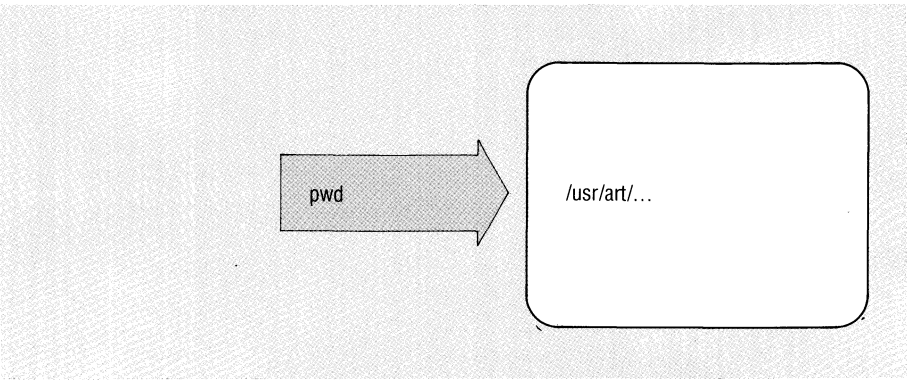
## BEISPIEL

Welche Prozesse laufen zur Zeit?

```
$ ps -a
PID TT STAT  TIME COMMAND
 13 co S     0:06 update
 15 co S     0:06 cron
 17 co S     1:48 daemon
 21 co S     1:41 lp9001
 25 co SW    0:00 remind
235 co S     0:04 sh
271 co R     0:00 ps
 28 03 SW    0:04 sh
216 03 SW    0:08 msh
231 03 SW    0:00 sh
```

## NAME

**pwd** - Pfadnamen des aktuellen Dateiverzeichnisses ausgeben (path of working directory)




## DEFINITION

**pwd**

## BESCHREIBUNG

*pwd* schreibt den Pfadnamen des aktuellen Dateiverzeichnisses auf Standard-Ausgabe. Dieses Kommando wird direkt von der Shell ausgeführt.

## BEISPIEL

Sie wollen Ihr aktuelles Dateiverzeichnis als HOME-Dateiverzeichnis definieren:

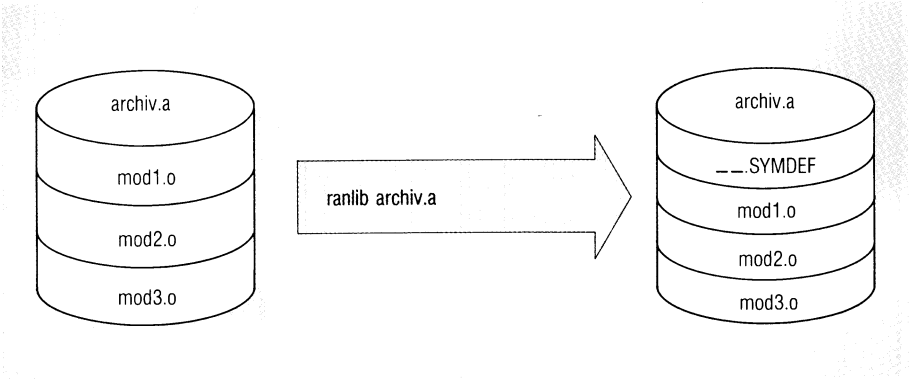
```
$ pwd
/usr/art/cobol/prg
$ HOME=`pwd`
$ echo $HOME
/usr/art/cobol/prg
$
```

## SIEHE AUCH

*cd*(1), *getcwd*(3C)

## NAME

**ranlib** - Bibliothek mit einer Symboltabelle versehen (convert archives to random libraries)




## DEFINITION

**ranlib** archiv...

## BESCHREIBUNG

*ranlib* legt eine Symboltabelle für eine Bibliothek an. Die Symboltabelle wird an den Anfang der Bibliothek gestellt. Sie heißt __.SYMDEF. In der Symboltabelle stehen die Namen der globalen Symbole, die in den Objektmodulen der Bibliothek enthalten sind.
Danach ruft *ranlib* das Kommando *ar*(1) auf, um die Bibliothek neu zu erstellen. Im Dateisystem des aktuellen Dateiverzeichnisses muß deswegen genügend freier Platz für Zwischendateien vorhanden sein.

Hat *ranlib* eine Symboltabelle __.SYMDEF für die Bibliothek erstellt, dann kann der Binder *ld*(1D) wahlfrei auf die Bibliothek zugreifen und deshalb die benötigten Objektmodule schneller finden und binden ("random access"). Siehe die Beschreibung von *ld*.

Jedesmal wenn Sie die Bibliothek bearbeiten, also z.B. ein Modul in die Bibliothek aufnehmen oder aus der Bibliothek löschen, dann müssen Sie *ranlib* erneut aufrufen, um die Symboltabelle __.SYMDEF zu aktualisieren.

## OPERANDEN

archiv

Die Datei *archiv* sollte eine Bibliothek sein, die Sie mit *ar*(1) angelegt haben.

## HINWEIS

*ld*(1D) gibt eine Warnung aus, wenn nach dem Anlegen der Symboltabelle durch *ranlib* an der Bibliothek etwas verändert wurde. Eine Warnung wird auch dann ausgegeben, wenn die Bibliothek nur kopiert wurde. Nach der Warnung durchsucht *ld* die Bibliothek dann nur sequentiell einmal.

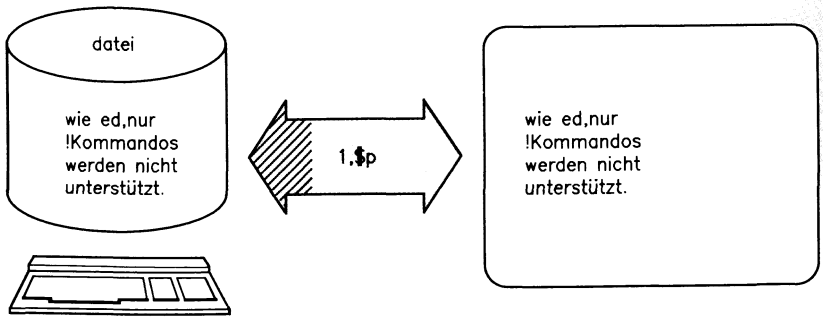
Sie behandeln am besten alle Bibliotheken, die Sie mit *ar*(1) bearbeiten, sofort danach mit *ranlib*. Ein "makefile" (siehe Kommando *make*(1D)) kann automatisch den *ranlib*-Aufruf übernehmen.

## SIEHE AUCH

*ar*(1), *cc*(1D), *ld*(1D), *lorder*(1D), *tsort*(1D).

## NAME

**red** - Eingeschränkter zeilenorientierter Editor im Dialogbetrieb




## DEFINITION

**red**[␣-][␣-**p**␣zeichenkette][␣datei]

## BESCHREIBUNG

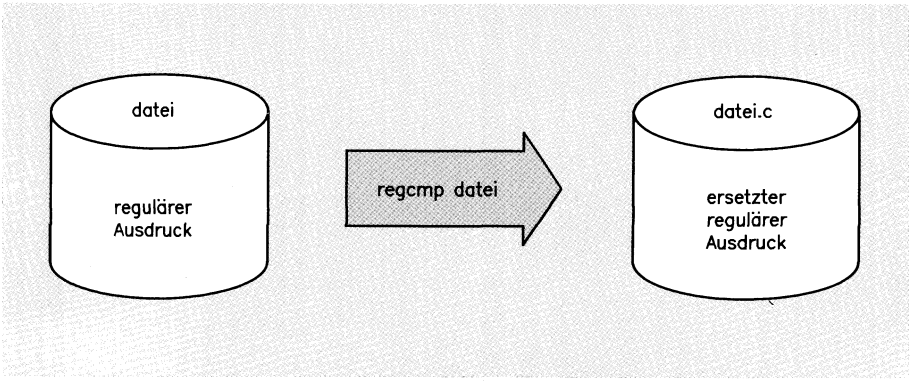
*red* ist die eingeschränkte Variante von *ed*(1). Mit diesem Editor können Sie nur Dateien bearbeiten, die im aktuellen Dateiverzeichnis enthalten sind. !*Kommandos* werden nicht unterstützt. Versuche, diese Einschränkungen zu umgehen, führen zu einer Fehlermeldung.

## SIEHE AUCH

*ed*(1)

## NAME

**regcmp** - regulären Ausdruck übersetzen (regular expression compile)




## DEFINITION

**regcmp**[␣-]␣datei...

## BESCHREIBUNG

*regcmp* übersetzt den in *datei* enthaltenen regulären Ausdruck und setzt das Ergebnis in *datei*.**i**. Ist Schalter - gesetzt, wird das Ergebnis in *datei*.c gesetzt. Bei den Einträgen in *datei* handelt es sich um einen Namen (eine C-Variable), gefolgt von einem oder mehreren Leerzeichen und einem in Anführungszeichen eingeschlossenen regulären Ausdruck. Die Ausgabe von *regcmp* erfolgt in C-Quellcode. Die übersetzten regulären Ausdrücke werden in Form von *extern char*-Vektoren ausgegeben. *datei*.i kann so in C-Programme eingebracht (included) werden.

## HINWEIS

Wenn Sie *regcmp* verwenden, müssen Sie die regulären Ausdrücke nicht mehr von Ihrem C-Programm aus übersetzen lassen (siehe *regexp*(3X)). Dies verringert sowohl die Ausführungszeit als auch die Größe des Programms.

## BEISPIEL

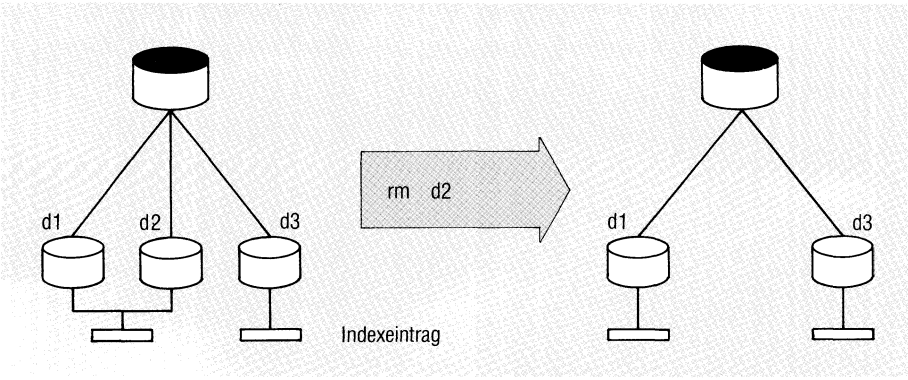
```
name    "([A-Za-z][A-Za-z0-9 ]*)$0"
telno   "\({0,1}([2-9][01][1-9])$0\){0,1}*"
        "([2-9][0-9]{2})$1[-]{0,1}"
        "([0-9]{4})$2"
```

## SIEHE AUCH

*regexp*(3X)

## NAME

**rm** - Dateien löschen (remove files)




## DEFINITION

**rm**[␣schalter]␣datei...

## BESCHREIBUNG

*rm* löscht für eine oder mehrere Dateien den Eintrag im Dateiverzeichnis. Wenn Sie eine Datei angeben, auf die es mehrere Verweise gibt, wird lediglich ein Verweis gelöscht, die Datei selbst bleibt vorhanden. Nur wenn ein Eintrag der letzte Verweis auf eine Datei war, wird die Datei gelöscht. Sie können Einträge nur löschen, wenn Sie für das Dateiverzeichnis, in dem die Datei steht, die Schreiberlaubnis haben. Für die Datei selbst brauchen Sie jedoch weder die Lese- noch die Schreiberlaubnis.

## SCHALTER

kein schalter

Wenn Sie für *datei* die Schreiberlaubnis haben, löscht *rm* den Eintrag ohne Warnung!

Wenn Sie für *datei* keine Schreiberlaubnis haben und die Standard-Eingabe eine Datensichtstation ist, werden die Zugriffsrechte und ein ? ausgegeben. Geben Sie darauf eine Zeile ein, die mit *y* (yes) beginnt, wird der Eintrag gelöscht; andernfalls bleibt er erhalten. Wenn die Standard-Eingabe keine Datensichtstation ist, wird der Eintrag ohne Rückfrage gelöscht.

**-f**

Die Einträge werden ohne Rückfrage gelöscht.

**-r**

Sie können für *datei* ein Dateiverzeichnis angeben. *rm* gibt dann nicht, wie normalerweise, eine Fehlermeldung aus. *rm* löscht rekursiv den gesamten Inhalt des angegebenen Dateiverzeichnisses ebenso wie das Dateiverzeichnis selbst.

**-i**

*rm* fragt für jede Datei und, wenn der Schalter *-r* gesetzt ist, für jedes Dateiverzeichnis, ob sie bzw. es gelöscht werden soll.

## OPERANDEN

datei

Name der Datei bzw. des Dateiverzeichnisses, die/das gelöscht werden soll.

[!] Das übergeordnete Dateiverzeichnis (..) kann nicht gelöscht werden; dadurch sollen die fatalen Folgen aus einem Kommando wie dem folgenden verhindert werden:

```
rm -r .*
```

## BEISPIEL

1. Löschen aller Dateien, die auf *.prog* enden, mit Abfrage:

```
$ rm -i *.prog
ablauf.prog:? (y/n) y
code.prog:? (y/n) y
eingabe.prog:? (y/n) n
zufall.prog:? (y/n) n
$
```

2. Löschen des Dateiverzeichnisses *norm* mit allen Dateien und Unterverzeichnissen:

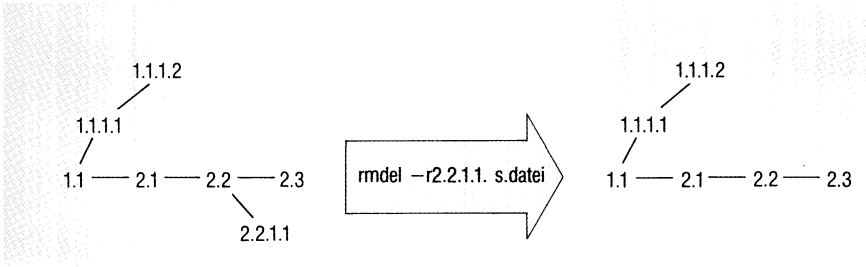
```
$ rm -r norm
$
```

**SIEHE AUCH**

*rmdir*(1), *unlink*(2)

## NAME

**rmdel** - Delta einer SCCS-Datei löschen (remove a delta from an SCCS file)




## DEFINITION

**rmdel␣-rSID␣datei...**

## BESCHREIBUNG

*rmdel* löscht eine Version aus einer SCCS-Datei. Die Version muß die jüngste Version sein, die innerhalb eines Zweiges (Branch) oder im Stamm des SID-Baumes erstellt wurde, d.h. sie muß ein Blatt des SID-Baumes sein. *rmdel* löscht keine Version, aus der gerade ein neues Delta erarbeitet wird und für die ein Eintrag in der p-Datei existiert.

Der Typ des Deltas ändert sich von D (für "Delta") zu R (für "removed"). Der Typ eines Deltas steht in der Delta-Tabelle der SCCS-Datei.

## SCHALTER

**-r**SID

*rmdel* löscht die Version mit der SID-Nummer *SID*.

## OPERANDEN

datei

Als *datei* können Sie den Namen einer s-Datei, eines Dateiverzeichnisses oder das Zeichen - angeben. *rmdel* behandelt den Namen eines Dateiverzeichnisses, wie wenn Sie die Namen aller Dateien des Verzeichnisses einzeln aufführen würden. *rmdel* ignoriert Namen von Dateien, die keine s-Dateien sind oder für die Sie keine Leseberechtigung haben.

Wenn Sie statt *datei* das Zeichen - angeben, dann liest *rmdel* von der Standard-Eingabe. *rmdel* behandelt jede Eingabezeile als Name einer Datei oder eines Dateiverzeichnisses.

Sie können beliebig viele Namen angeben. *rmdel* löscht dann aus allen entsprechenden s-Dateien die Version mit der Nummer *SID*.

**DATEIEN**

x-Datei
Siehe Kapitel "SCCS" im Buch SINIX Einführung [1].

z-Datei
Siehe Kapitel "SCCS" im Buch SINIX Einführung [1].

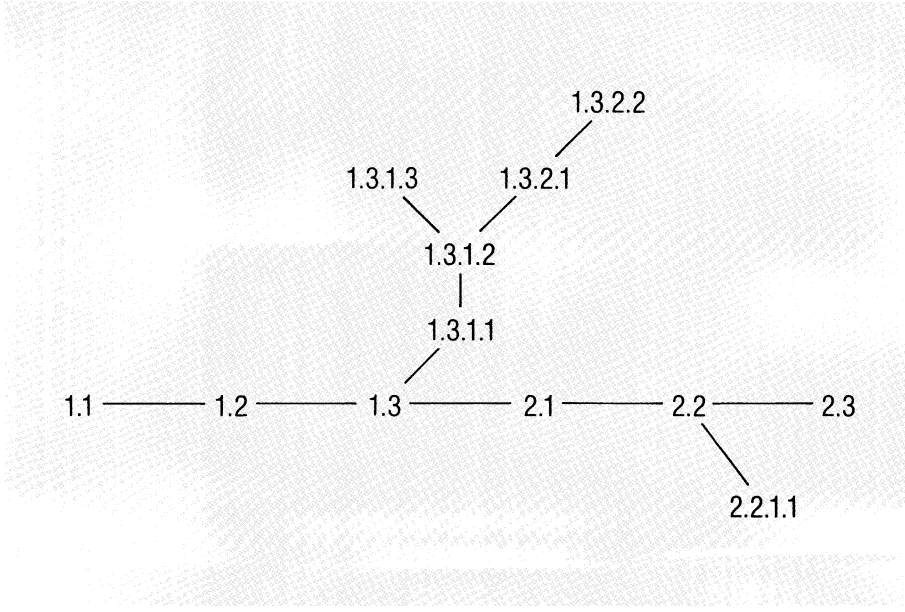
**HINWEIS**

Eine Version darf nur von folgenden Benutzern gelöscht werden:

- vom Benutzer, der das Delta erstellt hat und
- vom Eigentümer der SCCS-Datei und des Dateiverzeichnisses.

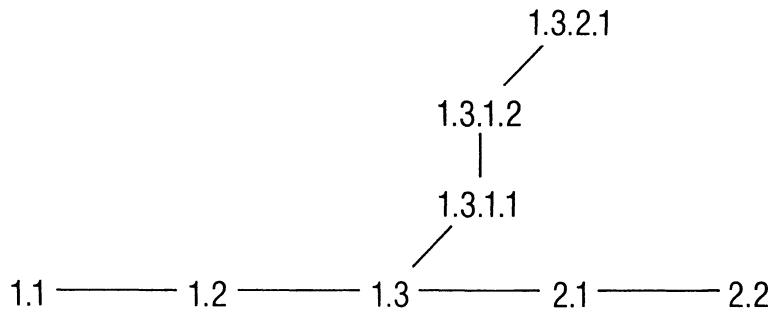
**BEISPIEL**

Sie haben eine SCCS-Datei mit folgendem SID-Baum erstellt:



Mit *rmdel* können Sie nur die Blätter des Baumes entfernen:

```
$ rmdel -r1.3.1.3 s.datei
$ rmdel -r1.3.2.2 s.datei
$ rmdel -r2.3 s.datei
$ rmdel -r2.2.1.1 s.datei
```




Als nächstes können Sie weitere Versionen entfernen:

```
$ rmdel -r1.3.2.1 s.datei
$ rmdel -r2.2 s.datei
$ rmdel -r2.1 s.datei
.
.
.
```

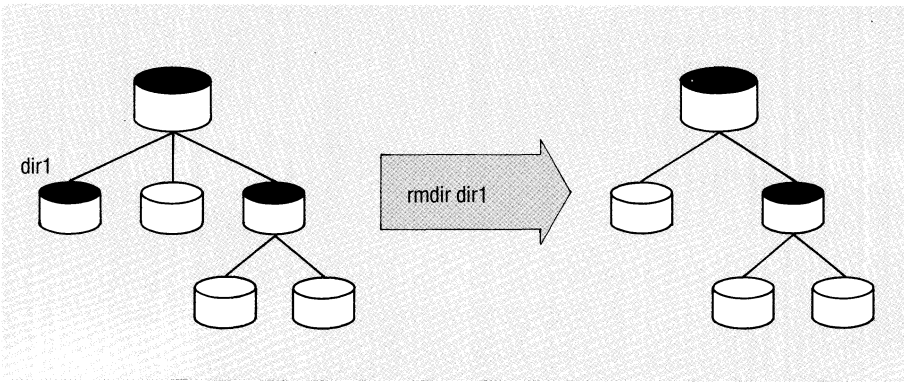
**SIEHE AUCH**

*admin*(1D), *delta*(1D), *get*(1D), *prs*(1D), *unget*(1D)

Eine Einführung ins SCCS ("Source Code Control System") finden Sie im Buch SINIX Einführung [1]. Dort sind auch alle Begriffe erklärt, die das SCCS betreffen.

## NAME

**rmdir** - Dateiverzeichnisse löschen (remove directories)




## DEFINITION

**rmdir** dateiverzeichnis...

## BESCHREIBUNG

*rmdir* löscht ein oder mehrere leere Dateiverzeichnisse. Dateiverzeichnisse mit Inhalt löschen Sie mit *rm*(1).

## BEISPIEL

Löschen der Dateiverzeichnisse *dir1* und *dir2*:

```
$ rmdir dir1 dir2
rmdir: dir1 not empty
$
```

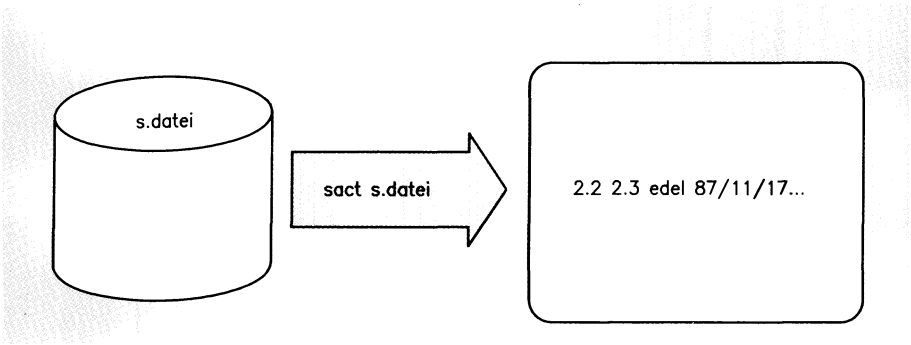
Dateiverzeichnis *dir1* wurde nicht gelöscht, da es noch Dateien enthält. Sie können es einschließlich aller darin enthaltenen Dateien löschen mit *rm -r dir1*.

## SIEHE AUCH

*rm*(1), *unlink*(2)

## NAME

**sact** - Editieraktivitäten einer SCCS-Datei ausgeben




## DEFINITION

**sact** datei...

## BESCHREIBUNG

*sact* informiert Sie über die anstehenden Deltas der angegebenen SCCS-Dateien. Ein Delta steht an, wenn eine Version mit *get*(1D), Schalter *-e*, zum Editieren aus dem SCCS geholt, das neue Delta aber noch nicht mit *delta*(1D) ins SCCS eingegliedert worden ist.

Wenn für die SCCS-Datei *datei* kein Delta ansteht, dann gibt *sact* aus:

```
No outstanding deltas for: datei
```

Sonst gibt *sact* für jedes anstehende Delta eine Zeile mit fünf Feldern aus, die durch je ein Leerzeichen voneinander getrennt sind. Die Felder enthalten folgende Information:

Feld 1 enthält die SID-Nummer des Deltas, das mit *get -e* zum Editieren geholt worden ist.

Feld 2 enthält die SID-Nummer des Deltas, das neu erstellt werden soll.

Feld 3 enthält den Login-Namen des Benutzers, der *get -e* aufgerufen hat und das neue Delta erstellen will.

Feld 4 enthält das Datum (Jahr/Monat/Tag), an dem *get -e* aufgerufen worden ist.

Feld 5 enthält den Zeitpunkt (Stunde:Minute:Sekunde), zu dem *get -e* aufgerufen worden ist.

## OPERANDEN

datei

Für *datei* können Sie den Namen einer s-Datei, eines Dateiverzeichnisses oder das Zeichen - angeben. *sact* behandelt den Namen eines Dateiverzeichnisses, wie wenn Sie die Namen aller Dateien des Verzeichnisses einzeln aufführen würden. *sact* ignoriert die Namen von Dateien, die keine s-Dateien sind oder für die Sie keine Leseberechtigung haben.

Wenn Sie für *datei* das Zeichen - angeben, dann liest *sact* von der Standard-Eingabe. *sact* behandelt jede Eingabezeile als Name einer Datei oder eines Dateiverzeichnisses.

Sie können beliebig viele Namen angeben.

## BEISPIEL

Im aktuellen Dateiverzeichnis stehen die SCCS-Dateien *s.hansi* und *s.hugo*. Sie wollen aus beiden SCCS-Dateien die jüngste Version, nämlich 2.2, holen und daraus ein neues Delta erstellen.

```
$ get -e s.hansi s.hugo

s.hansi:
2.2
new delta 2.3
5 lines

s.hugo:
2.2
new delta 2.3
7 lines
$ sact s.hansi s.hugo

s.hansi:
2.2 2.3 edel 87/11/17 12:55:31

s.hugo:
2.2 2.3 edel 87/11/17 12:55:31
$
```

Mit einem Editor verändern Sie die beiden Versionen, die *get*(1D) geholt hat. Anschließend gliedern Sie sie mit dem Kommando

```
$ delta s.hansi s.hugo
```

ins SCCS ein. Mit *sact* können Sie sich nun vergewissern, daß keine Deltas mehr anstehen:

```
$ sact s.hansi s.hugo

s.hansi:
No outstanding deltas for: s.hansi

s.hugo:
No outstanding deltas for: s.hugo
$
```

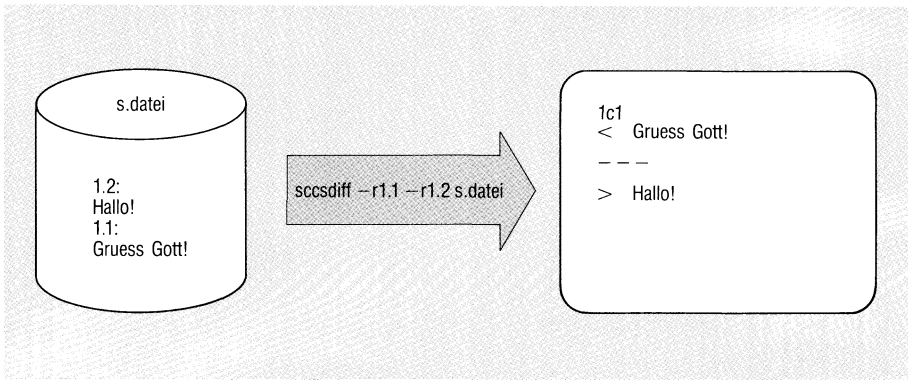
## SIEHE AUCH

*admin*(1D), *delta*(1D), *get*(1D), *unget*(1D).

Eine Einführung ins SCCS ("Source Code Control System") finden Sie im Buch SINIX Einführung [1]. Dort sind auch alle Begriffe erklärt, die das SCCS betreffen.

## NAME

**sccsdiff** - Zwei Versionen einer SCCS-Datei vergleichen




## DEFINITION

**sccsdiff␣-rSID1␣-rSID2[␣-p][␣-sn]␣datei...**

## BESCHREIBUNG

*sccsdiff* vergleicht zwei Versionen einer SCCS-Datei. Die Versionen werden an *bdiff*(1D) in der angegebenen Reihenfolge übergeben.

*sccsdiff* gibt (genau wie *bdiff*(1D) und *diff*(1)) aus:

- die Zeilen, in denen sich die Versionen unterscheiden,
- *ed*-Kommandos, mit denen man aus Version *SID1* Version *SID2* erzeugen kann (siehe *ed*(1)).

Wenn sich die beiden Versionen nicht unterscheiden, gibt *sccsdiff* aus:

```
datei: No differences
```

## SCHALTER

**-r**SID1 **-r**SID2

*SID1* und *SID2* sind die SID-Nummern der Versionen, die *sccsdiff* vergleichen soll. *sccsdiff* übergibt die Versionen in der angegebenen Reihenfolge an *bdiff*(1D).

**-p**

Die Ausgabe von *sccsdiff* wird mit einer Pipe an *pr*(1) weitergeleitet.

-sn

Dieser Schalter ist immer dann sinnvoll, wenn *diff*(1) wegen hoher Systemauslastung nicht ausgeführt werden kann.
*n* ist die Segmentgröße, die *sccsdiff* an *bdiff*(1D) und *bdiff* an *diff* übergibt.
*n* muß eine ganze Zahl größer als 0 sein. *bdiff* ignoriert am Anfang der Dateien die Zeilen, die sich nicht unterscheiden. Den Rest der Dateien teilt *bdiff* in Segmente von *n* Zeilen. *bdiff* ruft *diff* auf. *diff* vergleicht die entsprechenden Segmente der beiden Dateien. Der Wert für *n* muß so gewählt sein, daß *diff* keine Schwierigkeiten hat, Segmente mit *n* Zeilen zu vergleichen.

*Standard (keine Angabe):*
n = 3500

## OPERANDEN

datei

*datei* ist der Name der s-Datei, deren Versionen *sccsdiff* vergleichen soll.

Sie können beliebig viele SCCS-Dateien angeben. Die gesetzten Schalter gelten dann für alle angegebenen Dateien.

## DATEIEN

/tmp/get?????
Zwischendateien

## HINWEIS

Da *bdiff*(1D) die Versionen in Segmente aufteilt, können nur die Unterschiede zwischen sich entsprechenden Segmenten festgestellt werden. *bdiff* vergleicht also z.B. nicht das 4. Segment in Version 1 mit dem 5. Segment in Version 2. Deshalb gibt *bdiff* die Unterschiede oft umständlicher aus, als es notwendig wäre.

**BEISPIEL**

In der Datei *lebenslust* steht folgender Text:

```
Null Bock auf nix.
Null Bock auf gar nix.
Null Bock auf überhaupt nix.
```

```
$ admin -ilebenslust s.lebenslust
```

richtet die s-Datei *s.lebenslust* ein. Mit

```
$ get -e s.lebenslust
```

holen Sie die s-Datei. Den Text in der editierten Datei *lebenslust* ändern Sie zu:

```
Nix.
Kein Bock auf nix.
Null Bock auf gar nix.
Null Bock auf überhaupt nix.
```

und übergeben ihn mit

```
$ delta s.lebenslust
```

als Version 1.2 dem SCCS. *sccsdiff* stellt die Unterschiede zwischen den beiden Versionen fest:

```
$ sccsdiff -r1.1 -r1.2 s.lebenslust
1c1,2
< Null Bock auf nix.
---
> Nix.
> Kein Bock auf nix.
$
```

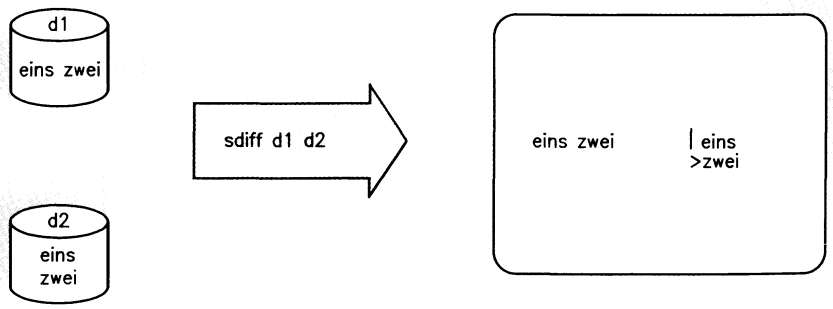
**SIEHE AUCH**

*admin*(1D), *bdiff*(1D), *delta*(1D), *diff*(1), *get*(1D), *prs*(1D).

Eine Einführung ins SCCS ("Source Code Control System") finden Sie im Buch SINIX Einführung [1]. Dort sind auch alle Begriffe erklärt, die das SCCS betreffen.

## NAME

**sdiff** - Dateien vergleichen und nebeneinander ausgeben




## DEFINITION

**sdiff**[␣schalter ...]␣datei1␣datei2

## BESCHREIBUNG

*sdiff* benutzt die Ausgabe von *diff*(1), um den Inhalt von zwei Dateien und ihre Unterschiede anzuzeigen; die Zeilen, zwischen denen Unterschiede bestehen, werden dabei gekennzeichnet. In der Ausgabe stehen links Zeilen aus *datei1*, rechts Zeilen aus *datei2*. Sind die entsprechenden Zeilen identisch, so werden sie durch Leerzeichen voneinander getrennt. Ist eine Zeile nur in *datei1* vorhanden, so wird sie durch ein < gekennzeichnet; ist eine Zeile nur in *datei2* vorhanden, so wird sie durch ein > gekennzeichnet; unterschiedliche Zeilen werden durch ein | gekennzeichnet.

*Beispiel*

```
Wenn in datei1 die Zeilen:   x
                             a
                             b
                             c
                             d

und in datei2 die Zeilen:    y
                             a
                             d
                             c
```

stehen, erhalten Sie mit dem Aufruf

```
$ sdiff datei1 datei2
```

folgende Ausgabe:

```
x       |       y
a               a
b       <
c       <
d               d
        >       c
```

## SCHALTER

**-w␣n**
Die Ausgabezeile soll *n* Zeichen breit sein (Standard: 130 Zeichen).

**-l**
Bei identischen Zeilen soll nur *datei1* angezeigt werden.

**-s**
Identische Zeilen werden unterdrückt.

**-o␣**ausgabedatei
*ausgabedatei* ist der Name einer Datei, in der *datei1* und *datei2* gemischt werden können. Identische Zeilen werden in *ausgabedatei* gespeichert. Unterschiede werden wie bei *diff*(1) in Sets, z.B. alle aufeinanderfolgenden < ausgegeben.

Nach der Ausgabe eines Sets von unterschiedlichen Zeilen gibt *sdiff* die Eingabeaufforderung % aus; diese müssen Sie durch eine der folgenden Eingaben beantworten:

| | |
|---|---|
| **l** | Einfügen der linken Spalte in *ausgabedatei* |
| **r** | Einfügen der rechten Spalte in *ausgabedatei* |
| **s** | (*silent*) Identische Zeilen werden unterdrückt |
| **v** | Hebt Schalter *s* auf |
| **e l** | Aufruf des Editors mit der linken Spalte |
| **e r** | Aufruf des Editors mit der rechten Spalte |
| **e b** | Aufruf des Editors mit den verknüpften linken und rechten Spalte |

**e** Aufruf des Editors mit einer leeren Datei
**q** Verlassen des Programms

Beim Verlassen des Editors wird die editierte Datei am Ende von *ausgabedatei* eingefügt.

## BEISPIEL

Vergleich und Ausgabe der Dateien *dat1* und *dat2*:

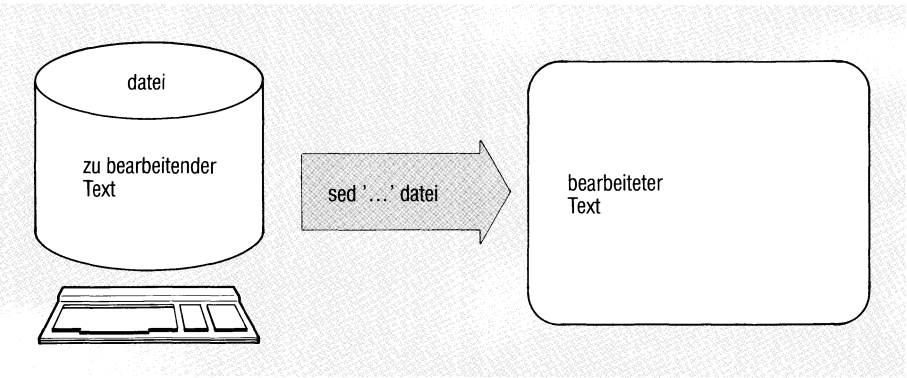
```
$ cat dat1
Löwe 23.07. - 23.08.
Es lohnt sich, an eine
radikale Umstellung zu
denken. Was Sie in der
Liebe wünschen, wird
erfüllt.

Glauben Sie das etwa?


$ cat dat2
Skorpion 24.10. - 22.11.
Nicht den Mut verlieren,
falls etwas schief geht.
Die Liebe bringt Ihnen
Ausgleich für des Tages
Last und Müh.

Glauben Sie das etwa?
München 18.1.88


$ sdiff -w 60 dat1 dat2
Löwe 23.07.- 23.08.             |   Skorpion 24.10. - 22.11.
Es lohnt sich, an eine          |   Nicht den Mut verlieren,
radikale Umstellung zu          |   falls etwas schief geht.
denken. Was Sie in der          |   Die Liebe bringt Ihnen
Liebe wünschen, wird            |   Ausgleich für des Tages
erfüllt.                        |   Last und Müh.

Glauben Sie das etwa?               Glauben Sie das etwa?
                                >   München 18.1.88
$
```

## SIEHE AUCH

*diff*(1), *ed*(1)

## NAME

**sed** - Editor im Prozedurbetrieb




## DEFINITION

**sed**[␣**-n**][**-e**␣skript][␣**-f**␣skriptdatei][␣datei...]

## BESCHREIBUNG

*sed* liest eine Datei ein, editiert sie entsprechend den Anweisungen im angegebenen Skript und gibt die bearbeitete Datei auf Standard-Ausgabe aus. Wenn Sie die Änderungen sichern möchten, müssen Sie die Standard-Ausgabe umleiten.

## SCHALTER

**-f**␣skriptdatei

*sed* liest das Skript aus *skriptdatei*.

**-e**␣skript

*sed* liest das Skript von der Kommandoaufrufzeile. Wenn *skript* mehr als ein Wort oder Sonderzeichen enthält, müssen Sie es in Hochkommas einschließen: 'Skript'

Die Schalter *-e* und *-f* können Sie mehrmals angeben; *sed* liest dann die Kommandos aus allen angegebenen Skripts.

**-n**

Die Ausgabe der eingelesenen Zeilen wird unterdrückt.

## OPERANDEN

datei
Name der Datei, die *sed* bearbeiten soll.

keine Angabe
*sed* liest von Standard-Eingabe.

## ARBEITSWEISE

*sed* kopiert zeilenweise den Inhalt der Eingabedatei in einen *Musterspeicher*, führt hintereinander alle Skript-Kommandos aus, die diesen Musterspeicher adressieren, kopiert den bearbeiteten Inhalt des Musterspeichers auf die Standard-Ausgabe (nicht bei Schalter *-n*) und löscht den Musterspeicher.

Ein Skript besteht aus *sed*-Kommandos der Form:

`[adresse[,adresse]]sed-Kommando[argument...]`

In einer Zeile steht jeweils ein *sed*-Kommando.

Einige der *sed*-Kommandos benutzen einen *Haltespeicher*, in dem der *Musterspeicher* vollständig oder teilweise zur späteren Wiederverwendung gespeichert wird.

## ADRESSEN

| Eine **adresse** kann sein: | Bedeutung: |
|---|---|
| **$** | letzte Zeile der Datei |
| ganze Zahl *n* | bezeichnet die Zeile mit der Nummer *n*. Die Nummern ergeben sich, indem die Zeilen aller Eingabedateien fortlaufend durchnumeriert werden. |
| /muster/ | bezeichnet eine Zeile, die eine zu *muster* passende Zeichenfolge enthält. Diese Art von Adressen heißt auch Kontextadresse, da eine Zeile durch Textausschnitte (Kontext) adressiert wird. *muster* ist ein einfacher regulärer Ausdruck wie bei *ed* (siehe auch Tabelle *Reguläre Ausdrücke* im Anhang) mit folgenden Änderungen: |

- In einer Kontextadresse ist die Angabe \?*regulärer ausdruck*? (? ist ein beliebiges Zeichen) gleichbedeutend mit */regulärer Ausdruck/*. Zu beachten ist, daß das zweite *x* in der *adresse* *xabc**xdefx* für sich selbst steht, so daß der reguläre Ausdruck folgendermaßen lautet: *abcxdef*.
- Die Escape-Sequenz *n* paßt zum Neue-Zeile-Zeichen im Musterspeicher.
- Ein Punkt . paßt zu jedem Zeichen außer dem Neue-Zeile-Zeichen im Musterspeicher.
- Eine Kommandozeile, die *keine Adresse* enthält, gilt für jede Zeile im Musterspeicher.
- Eine Kommandozeile mit einer Adresse gilt für jede Zeile im Musterspeicher, die zu der Adresse paßt.
- Zwei durch ein Komma voneinander getrennte Adressen beziehen sich auf den Bereich zwischen dem ersten Musterspeicher, der zur ersten Adresse paßt und dem nächsten Musterspeicher, der zur zweiten Adresse paßt (ist die zweite Adresse eine Zahl, die kleiner gleich der ersten ist, so wird nur die erste Zeile ausgewählt). Ist der ganze Bereich abgearbeitet, so wird der Prozeß wiederholt, d.h. *sed* sucht wieder nach einem Musterspeicher, der zur ersten Adresse paßt.

Sollen auf einen ausgewählten Musterspeicher mehrere *sed*-Kommandos angewendet werden, muß diese Kommandoliste in geschweifte Klammern {*sed*-Kommando...} gesetzt werden. Die schließende } muß an einem Zeilenanfang stehen.

Im folgenden sind die verschiedenen *sed*-Kommandos aufgeführt; die maximal zulässige Anzahl von Adressen steht in Klammern vor dem Kommandonamen.

Das Argument *text* besteht aus einer oder mehreren Zeilen, von denen alle bis auf die letzte mit einem \ enden müssen, um das anschließende Neue-Zeile-Zeichen zu entwerten. Sind im Text Gegenschrägstriche enthalten, so werden sie wie Gegenschrägstriche in der Ersetzungszeichenfolge eines *s*-Kommandos behandelt; auf diese Weise kann verhindert werden, daß führende Leer- und Tabulatorzeichen in *text* von *sed* entfernt werden, wie dies normalerweise mit jeder Skript-Zeile geschieht. Die Argumente *rdatei* und *wdatei* müssen, falls vorhanden, mit genau einem Leerzeichen davor am Ende der Kommandozeile stehen. Eine *wdatei* wird vor dem Beginn der Ausführung angelegt. Das Argument *wdatei* darf höchstens 10 mal vorkommen.

(1)**a**\
text
Vor dem Lesen der nächsten Zeile wird *text* auf die Standard-Ausgabe geschrieben.

(2)**b**marke
Verzweigung zum *:*-Kommando, das mit der Marke *marke* markiert ist. Fehlt *marke*, so verzweigt *sed* zum Ende des Skripts.

(2)**c**\
text
Verändern (change). Der Musterspeicher wird gelöscht. Wenn nur eine oder keine Adresse angegeben ist oder *sed* sich am Ende eines über 2 Adressen angegebenen Bereichs befindet, so schreibt *sed text* auf die Standard-Ausgabe. Dann wird der nächste Zyklus gestartet.

(2)**d**
Der Musterspeicher wird gelöscht; anschließend wird der nächste Zyklus gestartet.

(2)**D**
Der Anfang des Musterspeichers wird bis zum ersten Neue-Zeile-Zeichen gelöscht; anschließend wird der nächste Zyklus gestartet.

(2)**g**
Der Inhalt des Musterspeichers wird durch den Inhalt des Haltespeichers ersetzt.

(2)**G**
Der Inhalt des Haltespeichers wird an den Musterspeicher angefügt.

(2)**h**
Der Inhalt des Haltespeichers wird durch den Inhalt des Musterspeichers ersetzt.

(2)**H**
Der Inhalt des Musterspeichers wird an den Haltespeicher angefügt.

(1)**i**\
text
Einfügen (insert). *text* wird auf die Standard-Ausgabe geschrieben.

(2)**l**
Der Inhalt des Musterspeichers wird in einer eindeutigen Form auf die Standard-Ausgabe geschrieben. Nicht-druckbare Zeichen wer-

den durch ihren ASCII-Code dargestellt. Überlange Zeilen werden unterteilt.

(2)**n**
Der Musterspeicher wird auf die Standard-Ausgabe geschrieben. Der Inhalt des Musterspeichers wird durch die nächste Eingabezeile ersetzt.

(2)**N**
Die nächste Eingabezeile wird einschließlich des Neue-Zeile-Zeichens an den Musterspeicher angefügt (die Nummer der aktuellen Zeile ändert sich).

(2)**p**
Ausgeben (print). Der Inhalt des Musterspeichers wird auf die Standard-Ausgabe geschrieben.

(2)**P**
Der Anfang des Musterspeichers wird bis zum ersten Neue-Zeile-Zeichen (einschließlich) auf die Standard-Ausgabe geschrieben.

(1)**q**
Beenden (quit). *sed* wird beendet. Es wird kein neuer Zyklus gestartet.

(2)**r**rdatei
Der Inhalt von *rdatei* wird gelesen und vor dem Einlesen der nächsten Eingabezeile auf die Standard-Ausgabe geschrieben.

(2)**s**/rA/ersetzungszeichenfolge/schalter
Alle Zeichenfolgen im Musterspeicher, zu denen der *reguläre Ausdruck rA* paßt, werden durch *ersetzungszeichenfolge* ersetzt. Als Trennzeichen kann außer dem / jedes beliebige Zeichen verwendet werden. Nähere Informationen hierzu sind in der Beschreibung *ed*(1), Kommando *s*, enthalten. Schalter:

**n** $n = 1\text{-}512$
Nur jede *n*-te Zeichenfolge, zu der der *reguläre Ausdruck* paßt, soll ersetzt werden.

**g** Global
Alle Zeichenfolgen im Musterspeicher, zu denen der *reguläre Ausdruck* paßt, werden ersetzt, nicht nur die erste jeder Zeile.

**p**
Der Musterspeicher wird ausgegeben, falls eine Ersetzung durchgeführt wurde.

**w** wdatei
Schreiben (write). Der Musterspeicher wird an *wdatei* angefügt, falls eine Ersetzung durchgeführt wurde.

(2)**t**marke
Test. Verzweigung zu dem Kommando *:*, das mit der Marke *marke* markiert ist, falls seit dem letzten Lesen einer Ausgabezeile oder dem letzten Aufrufen eines *t*-Kommandos eine Ersetzung durchgeführt wurde. Fehlt *marke*, so verzweigt *sed* zum Ende des Skripts.

(2)**w** wdatei
Schreiben (write). Der Musterspeicher wird an *wdatei* angefügt.

(2)**x**
Austausch der Inhalte des Muster- und Haltespeichers

(2)**y**/folge1/folge2/
Zeichen ersetzen (transform). Jedes Vorkommen eines Zeichens aus *folge1* wird durch das entsprechende Zeichen aus *folge2* ersetzt. *folge1* und *folge2* müssen gleich lang sein.

(2)**!**kommando
Negation (don't). Das *kommando* (bzw. die Kommandoliste, falls *kommando* in geschweifte Klammern eingeschlossen ist) wird nur auf Zeilen angewandt, die nicht durch die Adresse(n) ausgewählt sind.

(0)**:**marke
Dieses Kommando setzt nur die *marke*, die von den Kommandos *b* und *t* angesprungen werden kann.

(1)**=**
Die aktuelle Zeilennummer wird auf der Standard-Ausgabe in einer separaten Zeile ausgegeben.

(2){
Die bis zu einem } folgenden Kommandos werden nur ausgeführt, wenn durch die Adressen Bezug auf den Musterspeicher genommen wird. Die schließende } muß am Zeilenanfang stehen.

(0)
Ein leeres Kommando wird ignoriert.

(0) #
Ist das erste Zeichen der ersten Zeile einer Skriptdatei ein #, so wird der darauffolgende Zeileninhalt als Kommentar interpretiert, es sei denn, das erste Zeichen nach dem # ist ein *n* (die Standard-Ausgabe wird dann unterdrückt). Auf *#n* folgende Zeichen werden ebenfalls ignoriert. Die Skriptdatei muß zumindest eine Kommandozeile enthalten.

## BEISPIEL

In alle Leerzeilen einer Datei soll XXXXX geschrieben werden:

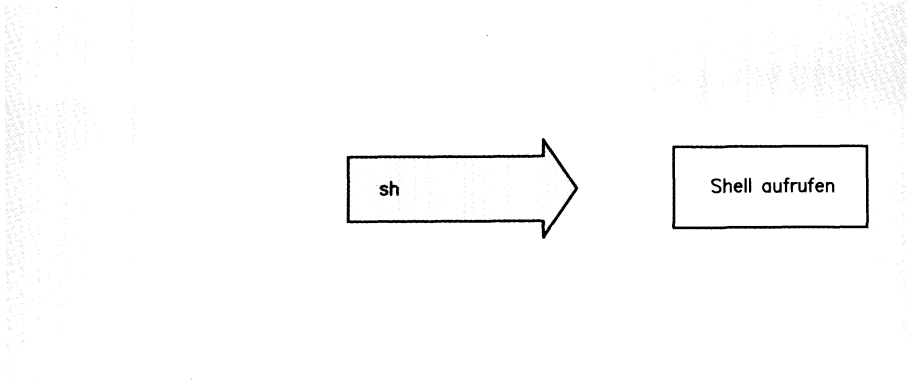
```
$ sed -e '/^$/s/^/XXXXX/' datei
```

## SIEHE AUCH

*awk*(1), *ed*(1), *grep*(1)

## NAME

**sh** - Kommandointerpreter Shell




## DEFINITION

**sh**[␣schalter][␣argument...]

## BESCHREIBUNG

Die Shell *sh* ist ein Interpreter, der Kommandos ausführt. Kommandos können in einer Datei stehen oder Sie können sie von der Datensichtstation aus eingeben. Eine Datei, in der Kommandos stehen, ist eine Shell-Prozedur. Näheres zur Arbeitsweise der Shell und zum Erstellen von Shell-Prozeduren finden Sie im Manual SINIX Einführung [1].

In der folgenden Beschreibung gelten folgende Definitionen:

| | |
|---|---|
| *Blank*: | Tabulator- oder Leerzeichen. |
| *name*: | Folge von Buchstaben, Ziffern oder Unterstreichungszeichen. Zu Beginn eines Namens steht ein Buchstabe oder ein Unterstreichungszeichen. |
| *parameter*: | Ein Name, eine Ziffer oder eines der Zeichen: *, @, #, ?, -, $ und ! |

## AUFRUF DER SHELL

Wenn die Shell über eine *exec*-Funktion (siehe *exec*(2)) aufgerufen wird und das erste Zeichen des Arguments 0 ein - ist, so werden die Kommandos zunächst aus den Dateien */etc/profile* und *$HOME/.profile* gelesen (falls vorhanden). Danach werden die Kommandos wie im folgenden beschrieben eingelesen. Die unten aufge-

führten Schalter werden von der Shell nur beim Aufruf ausgewertet; wenn nicht *-c* oder *-s* gesetzt ist, wird das erste Argument als Name einer Datei interpretiert, die Kommandos enthält, und die übrigen Argumente werden als Stellungsparameter interpretiert und dieses Kommando dabei übergeben.

**-c***zeichenfolge*
Ist dieser Schalter gesetzt, so werden die Kommandos aus *zeichenfolge* gelesen.

**-s**
Wenn der Schalter *-s* gesetzt ist oder keine Argumente mehr übrig bleiben, so werden die Kommandos von der Standard-Eingabe gelesen. Gegebenenfalls noch vorhandene Argumente geben die Stellungsparameter an. Die Shell-Ausgabe (außer *Eingebaute Shell-Kommandos*) wird in die Datei mit der Dateikennzahl 2 geschrieben.

**-i**
Wenn der Schalter *-i* gesetzt ist oder die Shell-Ein/Ausgabe einer Datensichtstation zugeordnet ist, so ist die betreffende Shell als *interaktiv* definiert. Dies hat zur Folge, daß das Signal **SIGTERM** ignoriert wird (d.h. mit *kill 0* wird eine interaktive Shell nicht beendet) und **SIGINT** abgefangen und ignoriert wird (d.h. *wait* ist unterbrechbar). **SIGQUIT** wird von der Shell ignoriert.

**-r**
Ist der Schalter *-r* gesetzt, so wird die eingeschränkte Version der Shell (siehe unten *Eingeschränkte Shell*) verwendet.

Die übrigen Schalter und Argumente werden im Abschnitt *set* (*Eingebaute Shell-Kommandos*) besprochen.

## OPERANDEN

arg...
Als Argumente können Sie der Shell Kommandos und Shell-Prozeduren übergeben. Welche Kommandos, Sonderzeichen, Variablen usw. es gibt, können Sie der folgenden Beschreibung entnehmen.

## SONDERZEICHEN DER SHELL

Die folgenden Zeichen sind Sonderzeichen der Shell. Sie haben für die Shell eine besondere Bedeutung, die in den folgenden Abschnitten beschrieben wird. Entwerten können Sie ein einzelnes Sonderzeichen, indem Sie ein \ (Gegenschrägstrich) davorschreiben, beliebig viele Sonderzeichen, indem Sie sie zwischen '...' schreiben.

### Steuerzeichen

**;**
**&**
**|**
**&&**
**||**
**(...)**
**{␣...;}**
name**()**␣**{**␣...**;}**
**#** kommentar
Neue-Zeile-Zeichen
Leerzeichen
Tabulatorzeichen

### Zeichen zur Generierung von Dateinamen

*****
**?**
**[...]**
**[!...]**

### Entwertungszeichen für Sonderzeichen und Apostrophierung

\
**'** . . . **'**
**"** . . . **"**
**`** . . . **`**
. . .

### Zeichen zur Umlenkung von Standard-Ein-/Ausgabe

>
>>
<
<<

<<-
2>
>&n
>&-
<&n
<&-

### Shell-Variablen

${name
${name:-wort}
${name:=wort}
${name:?wort}
${name:+wort}
$n
$*
$@
$#
$-
$$
$?
$$
$!

## KOMMANDOS

Kommandos können Sie interaktiv eingeben oder in eine Datei schreiben. Abgeschlossen wird die Eingabe eines Kommandos mit dem Neue-Zeile-Zeichen ↵ oder mit einem ; (Strichpunkt).
Mit dem Zeichen # können Sie Kommentare schreiben. Die Shell ignoriert alle Zeichen, die zwischen dem Zeichen # und dem nächsten Neue-Zeile-Zeichen stehen. Für die Shell gibt es drei Arten von Kommandos: *Einfache Kommandos, Kommandos zur Ablaufsteuerung, Eingebaute Shell-Kommandos.*

### Einfache Kommandos

Ein *einfaches Kommando* ist eine Folge von nicht-leeren *worten*, die durch Leerzeichen oder Tabulatorzeichen voneinander getrennt sind. Das erste *wort* ist der Name des auszuführenden Kommandos. Die übrigen *worte* werden außer im unten beschriebenen Fall an das aufgerufene Kommando als Argumente übergeben. Der Name des Kom-

mandos wird als Argument 0 (siehe *exec*(2)) übergeben. Der *wert* eines *einfachen Kommando*s ist sein Ende-Status, wenn es normal beendet wurde, andernfalls (oktal) 0200+status. Eine Auflistung der Statuswerte finden Sie im Abschnitt **HINWEIS**.

Eine *Pipeline* ist eine Folge von einem oder mehreren, durch ein Pipezeichen | (senkrechter Strich) voneinander getrennten Kommandos. Die Standard-Ausgabe jedes Kommandos mit Ausnahme des letzten Kommandos ist über eine Pipe mit dem nächsten Kommando verbunden (siehe *pipe*(2)), so daß die Ausgabe des ersten Kommandos als Eingabe des nächsten verwendet wird usw. Jedes Kommando wird in einem eigenständigen Prozeß ausgeführt; die Shell wartet auf die Beendigung des letzten Prozesses, bevor sie die nächste Eingabe bearbeitet. Der *Endestatus* einer *Pipeline* entspricht dem Ende-Status des letzten Kommandos.

Eine *liste* ist eine Folge von einer oder mehreren Pipelines, die durch eines der folgenden Zeichen voneinander getrennt sind: ;, &, &&, ||
Sie kann durch ; oder ein & abgeschlossen werden.
Die Zeichen ; und & haben dieselbe Priorität, jedoch eine niedrigere Priorität als && und ||. Die Zeichen && und || haben ebenfalls dieselbe Priorität.
Das Zeichen ; bewirkt, daß die vorhergehende Pipeline sequentiell ausgeführt wird.
Das Zeichen & bewirkt, daß die unmittelbar davor stehende Pipeline asynchron ausgeführt (=ein Hintergrundprozeß gestartet) wird, d.h. die Shell wartet nicht auf die Beendigung dieser Pipeline. Die Signale SIGINT und SIGQUIT werden ignoriert; andernfalls haben die Signale die Werte, die die Shell von ihrem Vaterprozeß erbt (siehe unten, eingebautes Shell-Kommando *trap*).
Das Zeichen && bewirkt, daß die nachfolgende *liste* nur dann abgearbeitet wird, wenn die vorhergehende Pipeline einen Ende-Status gleich Null liefert. Das Zeichen || bewirkt, daß die nachfolgende *liste* nur dann abgearbeitet wird, wenn die vorhergehende Pipe einen Endestatus ungleich Null liefert.
In *liste* dürfen beliebig viele Neue-Zeile-Zeichen enthalten sein; sie können dann anstelle von Strichpunkten zur Abgrenzung der Kommandos verwendet werden.

**Kommandos zur Ablaufsteuerung**

Mit den folgenden Kommandos zur Ablaufsteuerung können Sie die Shell wie eine Programmiersprache verwenden. Soweit nichts anderes angegeben, liefert ein Kommando zur Ablaufsteuerung den Wert, den das letzte *einfache Kommando* zurückgeliefert hat.

**for** name [**in** wort ...]
**do** liste
**done**

Bei jeder Ausführung eines *for*-Kommandos nimmt die Variable *name* den Wert des nächsten *wort*es an, das in *wort* ... enthalten ist. Wenn *in wort...* fehlt, führt das *for*-Kommando die *do-liste* einmal für jeden gesetzten Stellungsparameter aus (siehe Abschnitt *Parametersubstitution*). Die Kommandoliste wird so oft durchlaufen, wie *wort...* Wörter enthält.

**case** wort **in**
[muster[|muster]...)liste;;...]
**esac**

*case* vergleicht *wort* nacheinander mit den angegebenen *muster*n; paßt eines der Muster, so wird die zugehörige Kommando-*liste* ausgeführt. Die Muster dürfen dieselben Metazeichen enthalten, wie sie bei der Generierung von Dateinamen (siehe unten) verwendet werden.

**if** liste
**then** liste
[**elif** liste
**then** liste...]
[**else** liste]
**fi**

Wenn die auf *if* folgende Kommando*liste* den Endestatus 0 liefert, so wird die Kommando-*liste* nach dem ersten *then* ausgeführt. Andernfalls wird die auf *elif* folgende Kommando*liste* abgearbeitet. Liefert diese Liste den Wert 0, so wird die Kommando-*liste* nach dem nächsten *then* ausgeführt. Andernfalls wird die auf *else* folgende Kommando*liste* ausgeführt. Wird keine der auf *else* oder *then* folgenden Kommando*listen* ausgeführt, so liefert das *if*-Kommando den Ende-Status 0.

**while** liste
**do** liste
**done**

*while* führt die auf *while* folgende Kommando*liste* wiederholt aus. Hat das letzte Kommando in der *liste* den Endestatus 0, so wird die auf *do* folgende *liste* ausgeführt; andernfalls wird die Schleife beendet. Werden keine der in der *do-liste* enthaltenen Kommandos ausgeführt, so liefert *while* den Ende-Status 0. Die Umkehrung der *while*-Schleife ist die *until*-Schleife; die Schleife wird dann bei einem Ende-Status ungleich 0 verlassen.

**until** liste
**do** liste
**done**

Die Kommandos in der nach *until* stehenden Kommando*liste* werden ausgeführt. Solange diese Kommando*liste* einen Endestatus ungleich 0 meldet, wird die hinter *do* stehende Kommando*liste* ausgeführt.

**(**liste**)**

*liste* wird in einer Subshell abgearbeitet.

{liste;}

Die *liste* wird abgearbeitet; anstelle des Semikolons kann auch ein Neue-Zeile-Zeichen stehen.

**name(){**liste;**}**

Der Funktion *name* wird die in { und } eingeschlossene Kommando*liste* zugeordnet (anstelle des Semikolons kann auch ein Neue-Zeile-Zeichen stehen). Durch den Aufruf von *name* kann nun die Ausführung von *liste* erreicht werden (siehe Abschnitt *Durchführung von Kommandos*).

Die folgenden *worte* werden als Kommando zur Ablaufsteuerung nur erkannt, wenn sie am Zeilenanfang stehen und nicht apostrophiert sind:

```
if      for
then    while
else    until
elif    do
fi      done
case    {
esac    }
```

### Eingebaute Shell-Kommandos

Die eingebauten Shell-Kommandos unterscheiden sich von den übrigen in diesem Manual beschriebenen Kommandos dadurch, daß die Shell sie selbst interpretiert und verarbeitet. Die anderen Kommandos führt die Shell ungeprüft aus. Für die eingebauten Shell-Kommandos erzeugt die Shell keinen eigenen Prozeß, weshalb sie wesentlich schneller sind.

**:**
Leeres Kommando (tut nichts); Die Argumente werden wie gewöhnlich analysiert. Dieses Kommando liefert den Ende-Status 0.

**.datei**
Die Shell führt die in *datei* enthaltenen Kommandos aus. Über die Variable PATH wird der Suchpfad für das Dateiverzeichnis angegeben, in dem *datei* enthalten ist.

**break** [n]
Die umgebende *for*- bzw. *while*-Schleife wird verlassen. Mit *n* (optional) wird angegeben, wieviele Schachtelungen verlassen werden sollen.

**continue** [n]
Die Shell springt zum nächsten Iterationspunkt der umgebenden *for*- bzw. *while*-Schleife (d.h. der Rest bis zum Ende der Schleife wird übersprungen). Mit *n* (optional) kann dabei die Schachtelungstiefe angegeben werden.

**cd** [arg]
*arg* wird zum neuen aktuellen Dateiverzeichnis. Fehlt *arg*, so wird standardmäßig der Wert von $HOME eingesetzt. Die Variable CDPATH definiert den Suchpfad für das Verzeichnis, das *arg* enthält. Es können mehrere, durch Doppelpunkt voneinander getrennte Dateiverzeichnisse angegeben werden. Standardmäßig ist der Suchpfad leer. Mit einem leeren Pfadnamen wird in CDPATH der Name des aktuellen Dateiverzeichnisses angegeben, der sofort nach dem Gleichheitszeichen oder zwischen Doppelpunkten an einer beliebigen anderen Stelle der Pfadnamenliste stehen kann. Beginnt *arg* mit einem /, so wird der Suchpfad nicht benutzt; andernfalls wird in jedem Dateiverzeichnis im Suchpfad nach *arg* gesucht, und der Name des gewählten Dateiverzeichnisses wird ausgegeben. Das *cd*-Kommando kann in der eingeschränkten Version der Shell nicht ausgeführt werden.

**echo** [arg...]
Die Argumente werden ausgegeben. Nähere Informationen sind im Eintrag *echo*(1) enthalten.

**eval** [arg...]
Die Argumente werden als Kommando betrachtet und als solches ausgeführt.

**exec** [arg...]
Das über die Argumente angegebene Kommando wird anstelle der Shell ausgeführt, ohne daß hierzu ein neuer Prozeß erzeugt wird. Es können Ein/Ausgabe-Argumente angegeben werden; sind dies die einzigen Argumente, so werden die Shell-Ein/Ausgabedateien geändert.

**exit** [n]
Die Shell wird mit dem über *n* angegebenen Ende-Status beendet. Fehlt *n*, so wird der Ende-Status des zuletzt ausgeführten Kommandos ausgegeben (die Beendigung der Shell kann auch durch ein DateiendeZeichen verursacht werden).

**export** [name...]
Die angegebenen *name*n werden markiert und automatisch in die *Umgebung* der nachfolgend ausgeführten Kommandos exportiert. Sind keine Namen angegeben, so werden alle exportierten Namen ausgegeben. Die Namen von Funktionen können nicht exportiert werden.

**hash** [-rname...]
Die Shell ermittelt und speichert die Position aller über *name* angegeben Kommandos (d.h. ihren Suchpfad). Ist der Schalter *-r* gesetzt, so "vergißt" die Shell alle gespeicherten Positionen. Wird dieses Kommando ohne Argument aufgerufen, so werden die Kommandos ausgegeben, deren Suchpfad gespeichert ist.

**pwd**
Der Name des aktuellen Dateiverzeichnisses wird ausgegeben. Nähere Informationen siehe *pwd*(1).

**read** [name...]
Es wird eine Zeile von der Standard-Eingabe gelesen und die darin angegebenen Wörter nacheinander den angegebenen Variablen *name* zugewiesen; überzählige Wörter werden der letzten Variablen zugewiesen. Als Trennzeichen sind nur die in der Variablen IFS enthaltenen Zeichen zulässig. Dieses Kommando liefert den Ende-Status 0, es sei denn, es wird durch ein Dateiende-Zeichen beendet.

**readonly** [name...]
Die angegebenen *name*n werden als *schreibgeschützt* markiert; die Werte der so markierten *name*n werden als Konstanten betrachtet, deren Werte durch nachfolgende Wertzuweisungen nicht geändert werden darf. Fehlt das Argument *name*, so werden lediglich die *namen* aller schreibgeschützten Variablen ausgegeben.

**return** [n]
Eine Funktion wird beendet, wobei *n* als Ende-Status zurückgeliefert wird. Fehlt *n*, so wird der Ende-Status des zuletzt ausgegebenen Kommandos ausgegeben.

**set** [-schalter [arg...]]

**-a**
Die Variablen, die modifiziert oder exportiert wurden, sollen markiert werden.

**-e**
Die Shell wird sofort beendet, sobald ein Kommando einen Ende-Status ungleich null ausgibt.

**-f**
Die Generierung von Dateinamen durch Metazeichen wird unterdrückt.

**-h**
Die Namen und Positionen von Funktionen sollen bei ihrer Definition und nicht erst bei ihrer Ausführung gespeichert werden.

**-k**
Alle Kennwortparameter werden für ein Kommando in die Umgebung eingebracht (Standard: nur diejenigen Parameter, die vor dem Kommandonamen stehen).

**-n**
Die Kommandos werden nur gelesen, jedoch nicht ausgeführt.

**-t**
Die Shell soll nach dem Lesen und Ausführen eines Kommandos beendet werden.

**-u**
Bei der Substitution soll die Verwendung von nicht initialisierten Variablen als Fehler betrachtet werden.

**-v**
Die Shell-Eingabe wird so, wie sie gelesen wird, ausgegeben.

**-x**
Die Kommandos mit ihren eingesetzten Parametern (Argumenten) werden bei der Kommandoausführung ausgegeben.

**--**
Keiner der Schalter soll geändert werden; Dies ist hilfreich, wenn $1 auf - gesetzt werden soll.

Wird anstelle eines - das Zeichen + verwendet, so werden diese Schalter ausgeschaltet. Diese Schalter können auch beim Aufruf der Shell verwendet werden. Die aktuell gesetzten Shell-Schalter sind in $- zu finden. Bei den restlichen Argumenten handelt es sich um Stellungsparameter, die nacheinander $1, $2, .... zugewiesen werden. Ohne Angabe von Argumenten werden die Werte aller Parameter ausgegeben.

**shift** [n]
Die Stellungsparameter werden um *n* Stellen nach links verschoben, d.h. *$n+1* wird umbenannt in *$1*, *$n+2* in *$2* usw. Standard für *n* ist 1.

**test**
Bedingungen werden ausgewertet. Nähere Informationen siehe *test*(1).

**times**
Die bisher verbrauchte Benutzer- und Systemzeit für Prozesse, die von der Shell gestartet wurden, wird ausgegeben.

**trap** [arg] [n]...
Wird das Signal *n* an die Shell gesandt, so soll das Kommando *arg* gelesen und ausgeführt werden (*arg* wird einmal beim Aufruf von *trap*, einmal bei Beendigung von *trap* gelesen). Sind mehrere *trap*-Kommandos angegeben, so werden sie in der Reihenfolge der aufsteigenden Signalnummern ausgeführt. *trap* wird nicht ausgeführt, wenn das betreffende Signal an die Shell gesandt, jedoch ignoriert wurde. Fehlt *arg*, so werden alle Signale *n* auf ihre ursprünglichen Werte zurückgesetzt (d.h. *trap* wird beendet). Ist *arg* eine leere Zeichenfolge, so wird das betreffende Signal von der Shell und den von ihr aufgerufenen Kommandos ignoriert. Bei *n* gleich 0 wird das Kommando *arg* bei Verlassen der Shell ausgeführt. Wird das Kommando *trap* ohne Argumente aufgerufen, so werden die Kommandos aufgelistet, die zu den jeweiligen Signalnummern gehören.

**type** [name...]
Die Shell gibt für jeden angegebenen *namen* aus, welches Programm ausgeführt würde, falls es als Kommandoname verwendet würde.

**ulimit** [-fn]
Ist der Schalter *-f n* gesetzt, so können Dateien, die von der Shell und ihren Sohnprozessen geschrieben werden, maximal *n* * 512 Byte groß werden. (Dateien, die nur gelesen werden sollen, dürfen dagegen beliebig groß sein). Nur vom Systemverwalter kann diese Grenze hinaufgesetzt werden. Fehlt *n*, so wird der aktuelle Grenzwert ausgegeben. Ist kein Schalter gesetzt, so wird das Kommando so ausgeführt, als wäre Schalter *-f* gesetzt.

**umask** [ooo]
Die Maske, über die bei der Erstellung von Dateien die Einstellung der Zugriffsrechte geregelt wird (siehe *umask*(1)), wird auf die Oktalzahl *ooo* gesetzt. Fehlt *ooo*, so wird die aktuelle Schutzbiteinstellung ausgegeben.

**unset** [name...]
Die Definition der in *name* angegebenen Shellvariablen wird aufgehoben. Bei den Variablen PATH, PS1, PS2, MAILCHECK und IFS ist dies nicht möglich.

**wait** [n]
Die Shell wartet, bis der angegebene Prozeß beendet ist, und gibt seinen Ende-Status aus. Fehlt die Angabe von *n*, so wird die Beendigung aller momentan aktiven Sohnprozesse der betreffenden Shell abgewartet; das Kommando gibt dann den Ende-Status 0 aus.

## GENERIERUNG VON DATEINAMEN

Mit den folgenden Zeichen können Sie Muster für Dateinamen angeben. Ein Muster beschreibt eine Menge von Zeichenfolgen in Dateinamen. Jede dieser Zeichenfolgen paßt zu dem Muster. Die Shell erzeugt aus Dateinamen, in denen ein solches Muster vorkommt, alle Dateinamen, die zu dem Muster passen.

*****
steht für null, ein oder mehrere beliebige Zeichen in einem *wort*. Die Shell interpretiert dann *wort* als Muster und ersetzt es durch alle alphabetisch sortierten Dateinamen, die zu *wort* passen.

**?**
steht für ein beliebiges Zeichen in einem *wort*. Die Shell interpretiert *wort* als Muster und ersetzt es durch alle alphabetisch sortierten Dateinamen, die zu *wort* passen.

**[...]**
steht für ein beliebiges der in den Klammern eingeschlossenen Zeichen. Zwei Zeichen, die durch ein - voneinander getrennt sind, stehen für ein Zeichen, das in der ASCII-Tabelle (siehe Anhang) zwischen diesen beiden Zeichen steht.

**[!...]**
steht für ein beliebiges Zeichen, das nicht in den Klammern steht. Auch hier können Sie einen Bereich angeben.

! Dateinamen, die mit einem . (Punkt) beginnen, werden nicht expandiert. Soll nach ihnen gesucht werden, muß der Punkt explizit angegeben werden.

## APOSTROPHIER-MECHANISMUS

Mit dem Apostrophier-Mechanismus kann man der Shell mitteilen, wie sie ein Zeichen bzw. eine Zeichenkette interpretieren soll. Es gibt folgende Möglichkeiten, in einem *wort* Zeichen zu apostrophieren:

"zeichenkette"

- Die Shell liest *zeichenkette*
- Die Shell ersetzt vorhandene Stellungs- und Kennwortparameter
- Die Shell führt *zeichenkette* im Sinn des davorstehenden Kommandos aus
- Die Bedeutung folgender Sonderzeichen bleibt erhalten: $ ` \

'zeichenkette'

- Die Shell liest *zeichenkette*, ignoriert jedoch ihre Bedeutung, d.h. Kommandonamen und Sonderzeichen gelten als einfache Zeichen. Wenn Sie eine Zeichenkette in '...' einschließen, ist dies gleichbedeutend damit, daß Sie vor jedes einzelne Zeichen das Fluchtsymbol \schreiben.

\zeichen

- Mit dem Fluchtsymbol \ werden Sonderzeichen für die Shell entwertet. Wenn Sie mehrere Sonderzeichen entwerten möchten, können Sie das auch tun, indem Sie sie in Hochkommas '...' einschließen.

`zeichenkette`

- Die Shell interpretiert *zeichenkette* als Kommando, führt das Kommando aus und ersetzt *zeichenkette* durch seine Ausgabe. Nachfolgende Neue-Zeile-Zeichen werden gelöscht.

## EIN/AUSGABE-UMLEITUNG

Bevor ein Kommando ausgeführt wird, kann seine Ein- und Ausgabe durch Angabe von speziellen Zeichen umgeleitet werden. Diese Zeichen sind im folgenden aufgeführt; sie können an einer beliebigen Stelle innerhalb eines *einfachen Kommandos* vorkommen, einem Kommando vorangehen oder ihm folgen; sie werden nicht an das aufgerufene Kommando übergeben. Vor der Verwendung von *wort* oder *zahl* findet eine Substituierung statt:

< wort
Die Datei namens *wort* soll die Standard-Eingabe sein (Dateikennzahl 0).

> wort
Die Datei namens *wort* soll die Standard-Ausgabe sein (Dateikennzahl 1).
Existiert die angegebene Datei bereits, wird sie überschrieben! Existiert die angegebene Datei nicht, so wird sie erstellt.

>>wort
Die Datei *wort* soll die Standard-Ausgabe sein. Existiert die Datei, so wird die Ausgabe an diese Datei angehängt; andernfalls wird die Datei erstellt.

<<[-]wort
Die Shell-Eingabe wird bis zu einer Zeile, die nur aus *wort* besteht, oder bis zum Dateiende-Zeichen gelesen. Das so entstandene Dokument wird zur Standard-Eingabe. Ist ein Zeichen in *wort* apostrophiert, so werden die im Dokument enthaltenen Zeichen nicht interpretiert; andernfalls wird eine Parameter- und Kommandosubstitution vorgenommen, das (nicht entwertete) Neue-Zeile-Zeichen wird ignoriert, und die Zeichen \, $, ` sowie das erste Zeichen von *wort* müssen durch ein \ entwertet werden. Folgt auf << ein -, so werden in *wort* und allen nachfolgenden Zeilen bis zum Ende des Dokumentes alle führenden Tabulatorzeichen entfernt.

<&zahl
Die Datei mit Dateikennzahl *zahl* soll die Standard-Eingabedatei sein. Analog wird mit >&*zahl* die Datei angegeben, die als Standard-Ausgabe verwendet werden soll.

<&-
Die als Standard-Eingabe angegebene Datei wird geschlossen. Analog wird mit >&– die als Standard-Ausgabe verwendete Datei geschlossen.

Steht vor einem der obengenannten Kommandos eine Zahl, so wird die Dateikennzahl der Datei zugeordnet, die mit Zahl angegeben wurde (Standard: 0 oder 1).

So bewirkt z.B. das Kommando

```
... 2>&1
```

daß die Dateikennzahl 2 mit der Datei verknüpft wird, der aktuell die Dateikennzahl 1 zugeordnet ist.

Umleitungen müssen in der Reihenfolge angegeben werden, in der sie ausgeführt werden sollen. Bei der Abarbeitung geht die Shell von links nach rechts vor. So wird z.B. bei

```
... 1>xxx 2>&1
```

zuerst die Dateikennzahl 1 der Datei *xxx* zugeordnet. Dann wird die Dateikennzahl 2 der Datei mit der Dateikennzahl 1 zugeordnet (d.h. *xxx*). Wären die Umlenkungen in der umgekehrten Reihenfolge angegeben, so wäre die Dateikennzahl 2 der Datensichtstation zugeordnet, falls dieser vorher die Dateikennzahl 1 noch zugeordnet war, und die Dateikennzahl 1 wäre der Datei *xxx* zugeordnet.

Folgt auf ein Kommando ein *&*, so ist die Standard-Eingabe für dieses Kommando standardmäßig die leere Datei */dev/null*. Andernfalls enthält die Umgebung für das Kommando die Dateikennzahlen der aufrufenden Shell, wie sie durch Ein/Ausgabedefinitionen geändert wurden.

## VARIBLEN FÜR DIE SHELL

Die Shell erlaubt die Verwendung von Variablen. Eine Variable besteht aus einem Namen und einem Wert. Eine Variable definiert man so:

```
name=wert
```

Ansprechen kann man dann den Wert der Variablen mit:

$name (siehe unten, *Parametersubstitution*)

Im folgenden sind die Variablen der Shell beschrieben, die auch als Umgebungsvariablen bezeichnet werden.

HOME
: Das Standard-Dateiverzeichnis (HOME-Dateiverzeichnis) für das *cd*-Kommando.

PATH
: Der Suchpfad für Kommandos (siehe **Ausführung**, unten). Standard: /bin:/usr/bin

CDPATH
Der Suchpfad für das *cd*-Kommando. Die Verwendung und die Syntax entsprechen weitgehend derjenigen von PATH.

MAIL
Ist diesem Parameter der Name einer Briefkasten-Datei zugeordnet, so informiert die Shell den Benutzer, wenn in der angegebenen Datei Nachrichten enthalten sind. MAILPATH darf dabei nicht gesetzt sein. Standard: /usr/spool/mail/login-Kennung

MAILCHECK
Mit diesem Parameter wird angegeben, in welchen Zeitabständen (in Sekunden) die über die Parameter MAILPATH und MAIL angegebene Datei auf neu angekommene Nachrichten durchsucht werden soll. Standardmäßig erfolgt dies alle 600 Sekunden (10 Minuten). Ist MAILCHECK auf 0 gesetzt, so wird die Prüfung vor jeder Eingabeaufforderung vorgenommen. Standard: 600

MAILPATH
Über diesen Parameter können, durch einen Doppelpunkt voneinander getrennt, die Namen von Dateien angegeben werden; ist dieser Parameter gesetzt, so informiert die Shell den Benutzer darüber, ob in einer dieser Dateien für ihn Nachrichten angekommen sind. Folgt einem Dateinamen ein % und eine Nachricht, so wird diese Nachricht immer dann ausgegeben, wenn in einer der angegebenen Dateien eine Nachricht eingetroffen ist.

PS1
Die erste Eingabeaufforderung, mit der die Shell zeigt, daß sie auf eine Eingabe wartet.
Standard: $

PS2
Die zweite Eingabeaufforderung; dieses Zeichen wird nur ausgegeben, wenn nach dem Drücken der Taste ↵ für ein Kommando weitere Eingaben benötigt werden.
Standard: >

IFS
Die Zeichen, die von der Shell als Trennzeichen interpretiert werden.
Standard: Leerzeichen, Tabulatorzeichen, Neue Zeile-Zeichen.

SHACCT
Ist diesem Parameter der Name einer Datei zugewiesen, für die der Benutzer die Schreiberlaubnis besitzt, so schreibt die Shell nach der Ausführung jeder Shell-Prozedur Abrechnungsinformationen in die angegebene Datei.

SHELL
Wird die Shell aufgerufen, so untersucht sie die Umgebung (siehe unten, *Umgebung*) nach der Umgebungsvariablen SHELL.

Die Shell versteht auch folgende vordefinierte Zeichen, die sie durch ihren Wert ersetzt, wenn $ vorangestellt wird.

| | |
|---|---|
| # | Anzahl der Stellungsparameter (Dezimalzahl) |
| – | Schalter, mit denen die Shell aufgerufen wurde oder die durch das *set*-Kommando gesetzt wurden. |
| ? | Der Endestatus des letzten synchron ausgeführten Kommandos |
| $ | Prozeßnummer der betreffenden Shell |
| ! | Die Prozeßnummer des zuletzt aufgerufenen Hintergrundprozesses |
| * | Alle vorhandenen Stellungsparameter in der Form "$1$2$3..." |
| @ | Alle vorhandenen Stellungsparameter in der Form "$1", "$2", "$3", ... |

## UMGEBUNG

Unter *Umgebung* versteht man eine Reihe von Name-Wert-Paaren, die an das auszuführende Programm wie eine normale Argumentenliste übergeben werden. Die Shell benutzt die Umgebung auf unterschiedliche Weise. Wird die Shell aufgerufen, so überprüft sie die Umgebung und erstellt für jeden gefunden Namen eine Variable; jeder Variablen wird ein Wert zugeordnet. Wenn der Benutzer den Wert einiger dieser Variablen ändert oder neue Variablen erstellt, so wirkt sich dies nur dann auf die Umgebung aus, wenn die Shell-Variablen mit Hilfe des *export*-Kommandos in die Umgebung eingebracht werden (siehe auch *set -a*). Eine Variable kann mit dem *unset*-Kommando aus der Umgebung entfernt werden. Die Umgebung eines Kommandos besteht daher aus:

- unveränderten, von der Shell geerbten, Name-Wert-Paaren ohne die durch *unset* aufgehobenen Variablen
- plus den über *export* exportierten geänderten und neu hinzugefügten Name-Wert-Paaren.

Die Umgebung für ein *einfaches Kommando* kann erweitert werden, indem ihm eine oder mehrere Wertzuweisungen an Variable vorangestellt werden. So sind die beiden Kommandozeilen

TERM = 123 kommando
(export TERM; TERM = 123; kommando)

(*kommando* benutzt den Wert der Umgebungsvariablen TERM) gleichbedeutend, was die Ausführung von *kommando* angeht.

Ist der Schalter *-k* gesetzt, so werden alle Kennwortparameter in die Umgebung eingebracht, auch wenn sie auf den Kommandonamen folgen. Mit den folgenden Kommandos wird die Ausgabe *a=b c* und *c* erzeugt:

```
echo a=b c
set -k
echo a=b c
```

## PARAMETERSUBSTITUTION

In einer Shell-Prozedur ersetzt die Shell Parameter durch ihren Wert. Parameter, die ersetzt werden sollen, erkennt die Shell an dem Zeichen $. Es gibt zwei Arten von Parametern, die man mit $*parameter* ansprechen kann: *Stellungsparameter* und *Kennwortparameter*.

Ein *Stellungsparameter* ist eine der Ziffern 1-9, mit denen die Argumente durchgezählt werden, die beim Prozeduraufruf übergeben werden. Einem Stellungsparameter kann über das eingebaute Shell-Kommando *set* ein Wert zugewiesen werden.

An *Kennwortparameter* (auch *Variablen* genannt)
können folgendermaßen Werte zugewiesen werden:

```
name=wert [ name=wert] . . .
```

*wert* wird nicht mit einem Muster verglichen; eine Funktion und eine Variable darf nicht den gleichen Namen haben.

Im folgenden finden Sie die verschiedenen Arten, wie *parameter* in einer Shell-Prozedur angesprochen werden können. Anstelle von *parameter* kann jeweils entweder eine der Ziffern 1-9 (Stellungsparameter) oder *name* (Kennwortparameter) stehen:

${parameter}
Der Wert des Parameters wird (falls vorhanden) eingesetzt. Die geschweiften Klammern sind nur dann notwendig, wenn auf *parameter* ein Buchstabe, eine Ziffer oder ein Unterstreichungszeichen folgt, das nicht als Teil seines Namens interpretiert werden soll. Ist *parameter* ein * oder ein @, so werden alle Stellungsparameter ab $1 ersetzt. Der Parameter $0 wird beim Aufruf der Shell durch das Argument 0 gesetzt.

${parameter:-wort}
Ist *parameter* gesetzt und nicht leer, so wird sein Wert eingesetzt; andernfalls wird *wort* als Wert eingesetzt.

${parameter:=wort}
Wenn *parameter* leer oder nicht gesetzt ist, so wird ihm der Wert *wort* zugewiesen. Danach wird der Ausdruck durch den Wert von *parameter* ersetzt. Stellungsparametern kann auf diese Weise kein Wert zugewiesen werden.

${parameter:?wort}
Ist *parameter* gesetzt und nicht leer, so wird sein Wert eingesetzt; andernfalls wird *wort* ausgegeben und die Shell beendet. Fehlt *wort*, so wird die Meldung "parameter null or not set" ausgegeben.

${parameter:+wort}
Ist *parameter* gesetzt und nicht leer, so wird sein Wert eingesetzt; andernfalls wird eine leere Zeichenfolge eingesetzt.

Bei den obengenannten Substituierungen wird *wort* nur dann ausgewertet, wenn sein Wert eingesetzt werden soll; so wird beispielsweise *pwd* im folgenden Beispiel nur ausgewertet, wenn *d* leer oder nicht gesetzt ist.

```
echo ${d:-`pwd`}
```

Wenn Sie in den obengenannten Ausdrücken den Doppelpunkt : weglassen, so überprüft die Shell nur, ob *parameter* gesetzt ist, oder nicht.

## ARBEITSWEISE

Die Shell bearbeitet die ihr übergebenen Argumente in folgender Reihenfolge:

- Kommandosubstitutionen und Parametersubstitutionen werden durchgeführt
- Die Ergebnisse werden nach internen Feldtrennzeichen (siehe Variable IFS) durchsucht und in verschiedene Argumente aufgeteilt. Explizit leere Argumente "" oder '' bleiben erhalten, implizit leere Argumente (das Ergebnis von nicht definierten Parametern) werden gelöscht.
- Dateinamen werden über Metazeichen generiert.
- Ein/Ausgabeumleitungen werden ausgeführt.
- Die angegebenen Kommandos werden ausgeführt und Argumente übergeben.

## AUSFÜHRUNG VON KOMMANDOS

Bei jeder Durchführung eines Kommandos werden die obengenannten Substituierungen durchgeführt. Stimmt der Kommandoname mit einem der *Eingebauten Shell-Kommandos* überein, so wird es innerhalb des Shell-Prozesses ausgeführt. Paßt der Kommandoname nicht zu einem *Eingebauten Shell-Kommando*, so wird ein neuer Prozeß erzeugt und getestet, ob das Kommando über eine *exec*-Funktion (siehe *exec*(2)) ausgeführt werden kann.

Die Variable PATH definiert den Suchpfad für das Dateiverzeichnis, in dem das Kommando enthalten ist. Es können mehrere durch Doppelpunkt voneinander getrennte Dateiverzeichnisse angegeben werden. Das aktuelle Dateiverzeichnis wird durch einen leeren Pfad angegeben, der dann unmittelbar auf das Gleichheitszeichen folgt oder zwischen zwei Doppelpunkten steht. Enthält ein Kommando das Zeichen /, so wird der Suchpfad nicht benutzt; solche Kommandos können von der eingeschränkten Version der Shell nicht ausgeführt werden. Werden mehrere Dateiverzeichnisse angegeben, so werden sie entsprechend ihrer Anordnung von links nach rechts nach einer ausführbaren Datei durchsucht, die den Namen des Kommandos trägt. Wenn für die Datei Ausführungsrecht besteht, sie aber kein Programm ist, so wird davon ausgegangen, daß diese Datei Shell-Kommandos enthält; in diesem Fall wird eine Sub-Shell erzeugt, die diese Datei lesen soll. Ein in

Klammern eingeschlossenes Kommando wird ebenfalls in einer Sub-Shell ausgeführt.

Die Stelle im Suchpfad, an der ein Kommando gefunden wurde, wird von der Shell gespeichert (auf diese Weise lassen sich später unnötige *exec*-Aufrufe vermeiden). Wurde das Kommando in einem relativen Dateiverzeichnis gefunden, so muß seine Position bei jedem Wechsel des aktuellen Dateiverzeichnisses neu bestimmt werden. Die Shell "vergißt" alle gespeicherten Stellen, wenn die PATH-Variable geändert oder das Kommando *hash -r* (siehe unten) ausgeführt wird.

## EINGESCHRÄNKTE SHELL

Die eingeschränkte Shell ist eine Version der Shell, in der dem Benutzer nur eine eingeschränkte Umgebung zur Verfügung gestellt wird. Folgende Möglichkeiten von *sh* entfallen bei der eingeschränkten Shell:

- Aktuelles Dateiverzeichnis wechseln, siehe *cd*(1);
- Wert von PATH ändern;
- Kommandonamen mit einem / angeben und
- Ausgabe umleiten mit > und >>

Nach dem Aufruf der Shell werden zunächst die in der Datei *.profile* enthaltenen Kommandos ausgeführt. Erst dann sind die Einschränkungen wirksam.

Stellt sich heraus, daß ein auszuführendes Kommando eine Shell-Prozedur ist, so ruft die eingeschränkte Shell die Shell *sh* auf, die das Kommando dann ausführt. Auf diese Weise können dem Benutzer Shell-Prozeduren zur Verfügung gestellt werden, welche die Möglichkeiten der Standard-Shell voll nutzen können, während dem Benutzer nur eine eingeschränkte Auswahl an Kommandos zur Verfügung steht. Voraussetzung hierfür ist jedoch, daß der Benutzer keine Schreib- und Ausführerlaubnis in demselben Dateiverzeichnis hat.

Der Benutzer, der *.profile* erstellt hat, kann also den Handlungsspielraum eines anderen Benutzers festlegen, indem er dafür sorgt, daß nach dem Starten bestimmte Aktionen ausgeführt werden und der andere Benutzer in einem bestimmten Dateiverzeichnis (nicht unbedingt das Login-Dateiverzeichnis) bleibt.

**ENDESTATUS**

Stellt die Shell einen Fehler fest (z.B. in der Syntax), so liefert sie einen Ende-Status ungleich 0. Ist die Shell nicht interaktiv, so wird die Durchführung der Shell-Datei beendet. Andernfalls liefert die Shell den Ende-Status des zuletzt ausgeführten Kommandos (siehe auch Kommando *exit*).

**DATEIEN**

*/etc/profile*
Systemweite Start-Datei

*$HOME/.profile*
Individuelle Startdatei

*/dev/null*
Das *null*-Gerät

**HINWEIS**

- Wenn ein Kommando ausgeführt wird und ein Kommando mit demselben Namen in einem Dateiverzeichnis installiert wird, das im Suchpfad vor dem Dateiverzeichnis mit dem ursprünglichen Kommando steht, so führt die Shell weiterhin das ursprüngliche Kommando aus. In einem solchen Fall sollte mit einem *hash*-Kommando Abhilfe geschaffen werden.

- Wird das aktuelle Dateiverzeichnis bzw. das übergeordnete Dateiverzeichnis umbenannt (*mv*(1)), so hat *pwd* möglicherweise nicht das korrekte Ergebnis. In einem solchen Fall sollte mit einem *cd*-Kommando, bei dem der vollständige Pfadname angegeben wird, Abhilfe geschaffen werden.

- Die Ausgabe von eingebauten Kommandos sollte nicht für andere Programme verwendet werden, da es für ihr Format keine strengen Regeln gibt.

– Die folgenden Zahlen können beim Kommando *trap* für die einzelnen Signale verwendet werden:

```
1   SIGHUP
2   SIGINT
3   SIGQUIT
9   SIGKILL
15  SIGTERM
```

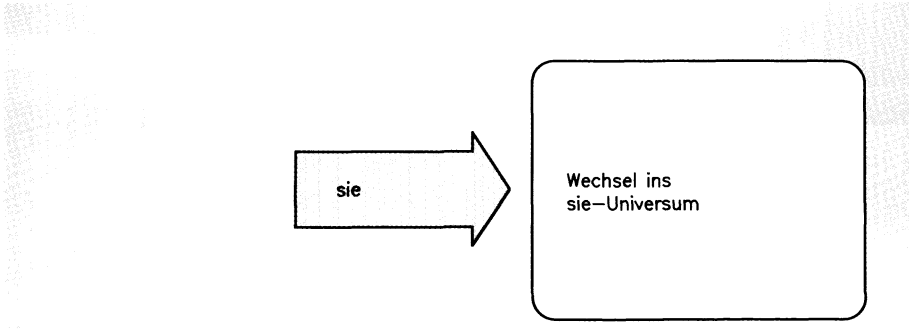
Eine Beschreibung dieser Signale ist im Eintrag *signal*(2) zu finden.

## SIEHE AUCH

*cd*(1), *echo*(1), *pwd*(1), *test*(1), *umask*(1), *dup*(2), *exec*(2), *exit*(2), *fork*(2), *pipe*(2), *signal*(2), *system*(2), *ulimit*(2), *umask*(2), *wait*(2).

## NAME

**sie** - in das sie-Universum wechseln




## DEFINITION

**sie**[␣kommando]

## BESCHREIBUNG

*sie* führt das angegebene Kommando *kommando* im *sie*-Universum aus. Bevor *sie kommando* startet, passiert folgendes:

- Das *sie*-Universum wird zu Ihrer aktuellen Arbeitsumgebung.
- Ihre Shell-Variable PATH erhält den Wert der Variablen SIEPATH.
- Ihre Shell-Variable SHELL erhält den Wert der Variablen SIESHELL.

kommando

Name des Kommandos, das im *sie*-Universum ausgeführt werden soll. Wenn das Kommando ausgeführt worden ist, arbeiten Sie wieder im vorhergehenden Universum und mit den vorher gültigen Shell-Variablen PATH und SHELL.

Standard (keine Angabe): *sie* ruft im *sie*-Universum eine neue Shell auf und führt sie als Login-Shell aus. Es passiert folgendes:

- Ihr Dateiverzeichnis $HOME wird zu Ihrem aktuellen Dateiverzeichnis.
- Die Datei /etc/profile wird ausgeführt.
- Die Datei $HOME/.profile wird ausgeführt.

Wenn Sie diese Shell mit [END] beenden, kehren Sie in Ihre vorherige Shell-Umgebung zurück: Sie arbeiten wieder in dem Universum und mit den Shell-Variablen PATH und SHELL, die vor dem Aufruf des Kommandos *sie* gültig waren.

[!] Wenn Sie beim Wechsel des Universums Ihr aktuelles Dateiverzeichnis beibehalten wollen, dann geben Sie ein:

```
sie sh ↵
```

In diesem Fall rufen Sie eine neue Shell auf, aber keine Login-Shell. Solange Sie in dieser Shell arbeiten, befinden Sie sich im *sie*-Universum. Wenn Sie diese Shell beenden, kehren Sie zum vorher gültigen Universum zurück.

Wenn Sie nicht mehr in das vorher gültige Universum zurückkehren wollen, dann geben Sie ein:

```
exec sie ↵
```

In diesem Fall ersetzt die neue Login-Shell die vorher gültige.

**BEISPIEL**

Für die folgenden Beispiele wird vorausgesetzt, daß Sie im *att*-Universum arbeiten und daß Ihre Datei $HOME/.profile die folgenden Zeilen enthält:

```
PS1="<`universe`>$ "
export PS1
```

Beim Aufruf der Login-Shell wird die Datei $HOME/.profile ausgeführt. Mit der Variablen PS1 definieren Sie die Eingabeaufforderung der Shell (Standard $) so um, daß nach jedem Aufruf der Login-Shell immer das gerade aktuelle Universum (Kommando *universe*) ausgegeben wird.

1. Sie wollen das Universum endgültig wechseln, also nicht mehr in das jetzt gültige *att*-Universum zurückkehren:

```
<att>$ universe ↵
att
<att>$ pwd ↵
/usr1/harald/texte
<att>$ exec sie ↵
<sie>$ pwd ↵
/usr1/harald
```

2. Sie wollen beim Wechsel des Universums keine Login-Shell, sondern nur eine einfache Subshell erzeugen:

```
<att>$ universe ↵
att
<att>$ pwd ↵
/usr1/harald/texte
<att>$ sie sh
<att>$ pwd ↵
/usr1/harald/texte
<att>$ universe ↵
sie
```

3. Sie befinden sich im *sie*-Universum und wollen nur ein Kommando im *att*-Universum ausführen:

```
<sie>$ att ls -l /bin
total 3306
-rwx--x--x 1 bin   bin       57344 Apr   2  13:58 adb
-rwx--x--x 1 bin   bin       28672 Apr   2  13:57 ar
-rwx--x--x 1 bin   bin       57344 Apr   2  13:57 as
-rwxr-xr-x 4 root  daemon    14336 Apr  21  19:47 att
-rwx--x--x 1 root  daemon      145 Apr  21  21:40 basename
.
.
.
```

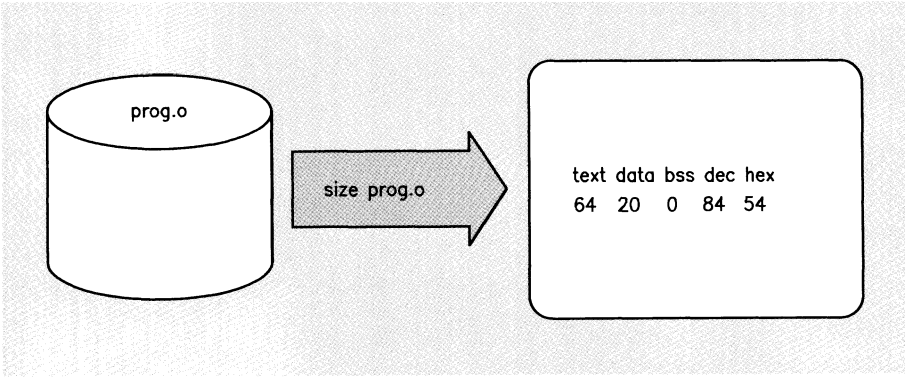
Beachten Sie dabei, daß der bedingte symbolische Verweis /bin das Dateiverzeichnis /usr/att/bin anspricht, denn das Kommando *ls -l* wird im *att*-Universum ausgeführt.

**SIEHE AUCH**

*att*(1), *env*(1), *login*(1), *universe*(1)

## NAME

**size** - Größe einer Objektdatei ausgeben (size of an object file)




## DEFINITION

**size**[␣datei...]

## BESCHREIBUNG

*size* gibt die Größe einer Objektdatei, d.h. eines Objektmoduls oder eines ablauffähigen Programms, aus.

*size* gibt auf der Standard-Ausgabe (dezimal) die Anzahl der Bytes an, die belegt werden

- vom Textsegment,
- vom Datensegment und
- vom bss-Segment.

Das Datensegment ist der initialisierte Datenbereich, das bss-Segment der nicht-initialisierte Datenbereich.

Außerdem gibt *size* die Summe dieser drei Zahlen aus.

**OPERANDEN**

datei

Für *datei* geben Sie den Namen einer Objektdatei oder einer Bibliothek an. Wenn Sie eine Bibliothek angeben, dann bearbeitet *size* alle Module der Bibliothek.
Sie können beliebig viele Namen angeben.

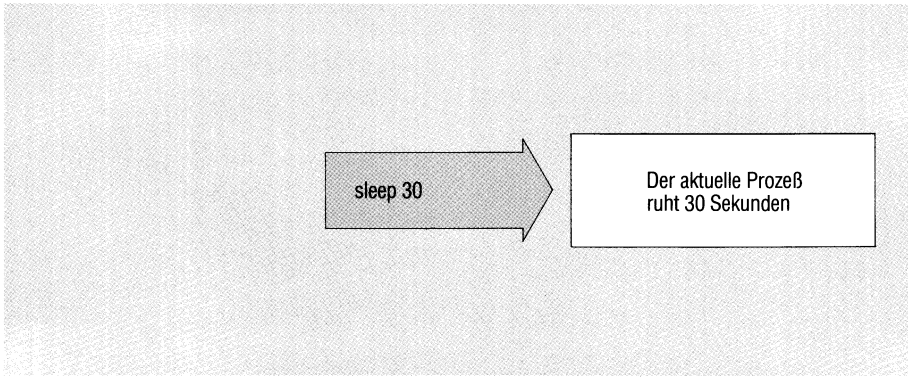
*Standard (keine Angabe):* a.out

**SIEHE AUCH**

*cc*(1D), *ld*(1D), *strip*(1D).

## NAME

**sleep** - Prozesse zeitweise stillegen




## DEFINITION

**sleep** *zeit*

## BESCHREIBUNG

Mit *sleep* können Sie die Ausführung des nächsten Kommandos um eine frei wählbare Zeitspanne hinauszögern. *sleep* benutzt man vor allem in Shell-Prozeduren. Mit *zeit* geben Sie an, für wieviele Sekunden der Prozeß ruhen soll. Die Ausführung der nachfolgenden Kommandos wird dann um die angegebene Zeit verschoben.

## BEISPIEL

1. *kommando* soll erst nach 105 Sekunden ausgeführt werden:

```
(sleep 105; kommando)&
```

2. Mit dem folgenden Beispiel wird angegeben, in welchen Zeitabständen ein *kommando* ausgeführt werden soll:

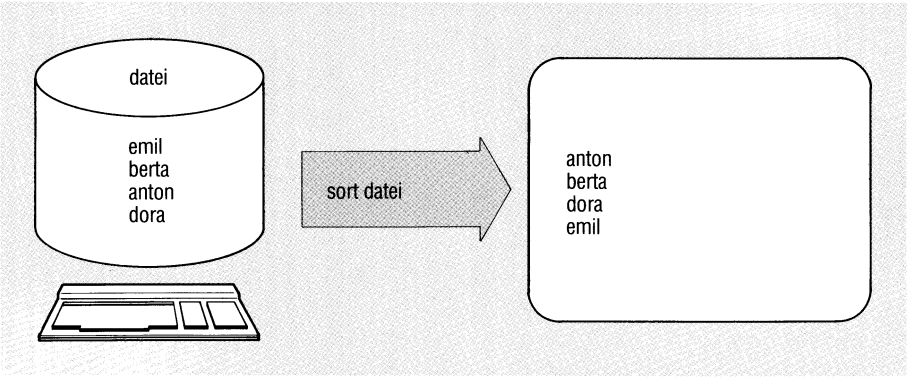
```
while true
do
     kommando
     sleep 37
done
```

**SIEHE AUCH**

*alarm*(2), *sleep*(2)

## NAME

**sort** - Dateien sortieren und/oder mischen (sort and/or merge files)




## DEFINITION

**sort**[␣schalter][␣datei...]

## BESCHREIBUNG

Mit *sort* können Sie die Zeilen einer Datei nach bestimmten Kriterien sortieren oder die Zeilen mehrerer Dateien zusammensortieren und mischen. *sort* schreibt das Ergebnis auf Standard-Ausgabe.

*sort* teilt die Zeilen einer Datei in Felder ein. Ein Feld ist eine Folge von Zeichen, die durch ein Feld-Trennzeichen oder Neue-Zeile-Zeichen abgegrenzt wird. Standardmäßig sind Leerzeichen und Tabulatorzeichen Feldtrenner. Bei einer Folge von Trennzeichen zählt nur das erste als Trenner, die restlichen werden dem folgenden Feld zugeordnet.

Beim Sortieren arbeitet *sort* mit einem Sortierschlüssel, der aus einem oder mehreren Sortierfeldern besteht. Wird nichts angegeben, gilt standardmäßig die ganze Zeile als Sortierschlüssel. Wenn Sie nur Ausschnitte der Zeilen betrachten wollen, können Sie Sortierfelder angeben. Ein Sortierfeld wird anhand der Felder einer Zeile angegeben durch Positionsangaben in der Form:

+pos -pos (siehe unten)

## SCHALTER

**Schalter, die das Verhalten von *sort* ändern**

kein Schalter
*sort* sortiert die den Eingabezeilen entnommenen Sortierfelder lexikographisch gemäß dem ASCII-Zeichensatz.

**-c**
*sort* prüft lediglich, ob die Eingabedatei bereits entsprechend den Sortierkriterien sortiert ist. Es erfolgt nur Ausgabe, wenn die Datei nicht sortiert ist.

**-m**
Die Eingabedateien sind bereits sortiert und sollen lediglich gemischt werden.

**-u**
(unique). Mehrfache Zeilen mit identischen Sortierfeldern sollen nur einmal ausgegeben werden.

**-o**[␣...]ausgabe
Mit *ausgabe* geben Sie den Namen einer Ausgabedatei an, die anstelle der Standard-Ausgabe verwendet werden soll. Hierbei darf es sich auch um eine der Eingabedateien handeln. Zwischen *-o* und *ausgabe* können Leerzeichen stehen.

**-y**[kmem]
Die Sortiergeschwindigkeit hängt in hohem Maße von dem zur Verfügung stehenden Speicherplatz ab. Der Speicherplatz sollte entsprechend der Größe der zu sortierenden Datei gewählt werden (eine kleine Datei in einem großen Speicher zu sortieren, ist Verschwendung). Ist dieser Schalter nicht gesetzt, so fängt *sort* mit einer Standard-Speichergröße an und benützt mehr Speicher, falls mehr benötigt wird. Wird für diesen Schalter ein Wert für *speicher* angegeben, so beginnt *sort* mit der angegebenen Speichergröße, es sei denn, die zulässige Höchst- oder Untergrenze wird über- bzw. unterschritten (in diesem Fall wird die vordefinierte Höchst- bzw. Untergrenze verwendet). So ist z.B. bei der Angabe *-y0* gewährleistet, daß mit dem minimalen Speicherplatz gestartet wird. Wird *-y* ohne Argument angegeben, fängt *sort* mit dem größtmöglichen Speicherplatz an.

-zrecsz
Die Länge der längsten Zeile, die während der Sortierphase gelesen wurde, wird gespeichert; auf diese Weise ist gewährleistet, daß für die Misch-Phase Puffer mit ausreichender Größe bereitgestellt werden. Sind die Schalter *-c* oder *-m* gesetzt (d.h. es wird nicht sortiert), so wird eine Standard-Größe verwendet. Zeilen, die für die reservierte Puffergröße zu lang sind, führen zur abnormalen Beendigung von *sort*. Dies kann verhindert werden, indem die Länge der längsten zu mischenden Zeile bzw. ein noch größerer Wert (in Byte) für *recsz* angegeben wird.

**Schalter zum Ändern der Sortierkriterien**

**-d**
Nur im US-ASCII-Zeichensatz enthaltene Buchstaben, Zahlen, Leer- und Tabulatorzeichen sollen beim Vergleich berücksichtigt werden.

**-f**
Groß- und Kleinbuchstaben sollen gleich behandelt werden.

**-i**
Zeichen, die nicht im Bereich 040-0176 (oktal) enthalten sind, sollen beim Sortieren in nicht-numerischer Reihenfolge ignoriert werden.

**-M**
Vergleichen nach Monatsnamen. Die ersten drei Nicht-Leerzeichen des Feldes werden als Großbuchstaben betrachtet und entsprechend verglichen und sortiert. Dabei gilt: *"JAN" < "FEB" < ... < "DEC"*. Ungültige Felder werden als < *"JAN"* einsortiert. Mit dem Schalter *-M* wird auch automatisch der Schalter *-b* gesetzt.

**-n**
Zahlenwerte am Anfang des Sortierfeldes, die aus Leerzeichen, Tabulatorzeichen, einem Minus-Zeichen sowie aus den Ziffern 0-9 und optional einem Dezimalpunkt bestehen können, werden nach dem Zahlenwert sortiert. Mit dem Schalter *-n* wird automatisch der Schalter *-b* gesetzt. Der Schalter *-b* hat jedoch nur eine Wirkung, wenn nach Sortierfeldern sortiert wird (d.h. nicht nach der ganzen Zeile).

**-b**
Führende Trennzeichen werden bei der Ermittlung von Anfang und Ende eines Sortierfeldes ignoriert.

**-r**
Sortieren in umgekehrter Reihenfolge.

Werden diese Schalter vor der ersten Positionsangabe für Sortierfelder angegeben, gelten sie global für alle Sortierfelder. Wird ein Schalter hinter einer Positionsangabe angegeben, hebt er globale Einstellungen auf. D.h. für dieses Sortierfeld gilt die Änderung des Sortierkriteriums gemäß diesem Schalter.

**Schalter zum Sortieren nach bestimmten Feldern**

+pos1
-pos2
Mit *+pos1* und *-pos2* kann angegeben werden, daß ein Sortierfeld bei *pos1* beginnen und bei *pos2* enden soll.Die Zeichen an *pos1* und *pos2* gehören noch zum Sortierfeld (vorausgesetzt, *pos1* steht vor *pos2*). Fehlt *-pos2*, so geht das Feld bis Zeilenende.

Die Argumente *pos1* und *pos2* haben das Format:

m.n

*+m.n*
spricht das n+1te Zeichen im m+1ten Feld an. Fehlt *.n*, so ist dies mit der Angabe .0 gleichbedeutend; hiermit wird das erste Zeichen des m+1ten Feldes angesprochen. Bei *m.nb* wird ab dem ersten Nicht-Leerzeichen im m+1-ten Feld gezählt, z.B. wird durch *+m.0b* das erste Nicht-Leerzeichen im m+1-ten Feld angesprochen.

*-m.n*
steht für das nte Zeichen (einschließlich Trennzeichen) nach dem letzten Zeichen im mten Feld. Fehlt *.n*, so wird .0 eingesetzt, was für das letzte Zeichen des mten Feldes steht. Mit Schalter *b* wird *n* ab dem letzten der vorangestellten Leerzeichen im m+1ten Feld gezählt; *-m.1b* bezieht sich auf das erste Zeichen im m+1ten Feld, das kein Leer- oder Tabulatorzeichen ist.

Gibt es mehrere Sortierfelder, so sortiert *sort* zunächst nach dem ersten, bei Gleichheit nach dem nächsten usw.

**-tx**
*x* soll das Feld-Trennzeichen sein; *x* gehört nicht zu einem Feld (obwohl es Teil eines Sortierfeldes sein kann). Jedes Trennzeichen ist signifikant, d.h. *xx* begrenzt ein leeres Feld.

**OPERANDEN**

datei...

Name der Datei(en), die Sie sortieren oder mischen möchten. Standard (keine Angabe): *sort* liest von Standard-Eingabe.

-

*sort* liest von Standard-Eingabe.

**ENDESTATUS**

| | |
|---|---|
| 0 | bei Erfolg |
| ungleich 0 | wenn *sort* einen Fehler (z.B. zu lange Eingabezeilen) feststellt oder beim Schalter *-c* eine noch nicht sortierte Datei vorliegt; es werden entsprechende Fehlermeldungen ausgegeben. |
| | Fehlt in der letzten Zeile einer Eingabedatei ein Neue-Zeile-Zeichen, so fügt *sort* ein solches Zeichen ein, gibt eine Warnung aus und setzt die Ausführung fort. |

**BEISPIEL**

1. Inhalt von *indatei* soll sortiert werden; das zweite Feld ist der Sortierschlüssel.

```
sort +1 -2 indatei
```

2. Der Inhalt von *indatei1* und *indatei2* soll in umgekehrter Reihenfolge sortiert werden; die Ausgabe soll in *outdatei* geschrieben werden, und das erste Zeichen im zweiten Feld soll als Sortierschlüssel dienen:

```
sort -r -o outdatei +1.0 -1.2 indatei1 indatei2
```

3. Der Inhalt von *indatei1* und *indatei2* soll in umgekehrter Reihenfolge sortiert werden, wobei das erste Nicht-Leerzeichen im zweiten Feld als Sortierschlüssel dient.

```
sort -r -o outdatei +1.0b -1.1b indatei1 indatei2
```

4. Die Datei */etc/passwd* soll ausgegeben werden; als Sortierschlüssel sollen die Benutzernummern (das dritte Feld) verwendet werden:

```
sort -t: +2n -3 /etc/passwd
```

5. Die bereits sortierte Datei *indatei* soll ausgegeben werden. Dabei soll von mehreren Zeilen, bei denen das dritte Feld übereinstimmt, jeweils nur die erste Zeile ausgegeben werden.

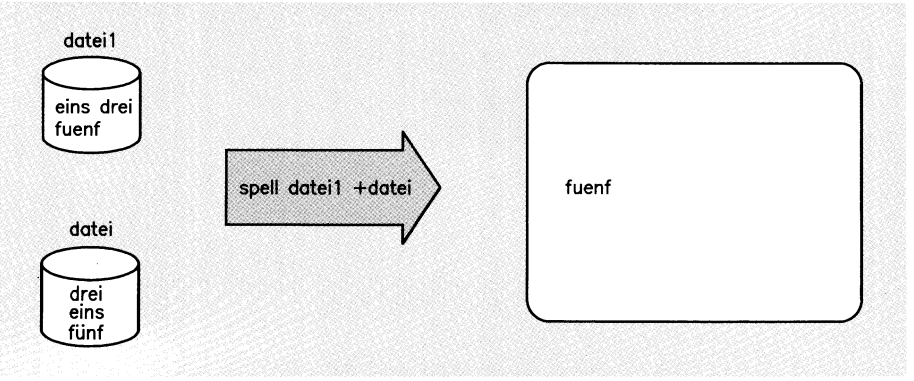
```
sort -um +2 -3 indatei
```

**SIEHE AUCH**

*comm*(1), *join*(1), *uniq*(1)

## NAME

**spell** - Rechtschreibfehler suchen (find spelling errors)




## DEFINITION

**spell**[␣schalter][␣+lokaldatei][␣datei...]

## BESCHREIBUNG

*spell* können Sie zur Überprüfung eines Textes nach Rechtschreibfehlern verwenden. *spell* liest Wörter aus der angegebenen *datei* und vergleicht sie mit einer Wortliste. Wörter, die weder in dieser Liste enthalten noch ableitbar sind (z.B. durch Präfixe, Suffixe oder Deklination), werden auf die Standard-Ausgabe geschrieben.

## SCHALTER

**-v**

Alle Wörter werden buchstabengetreu ausgegeben, die nicht in der Wortliste enthalten sind; außerdem werden mögliche Ableitungen aus Wörtern, die in der Wortliste enthalten sind, angezeigt.

**-b**

Es wird überprüft, ob die Wörter in der korrekten englischen Schreibweise (Standard: amerikanische Schreibweise) eingegeben wurden. Als korrekt angesehen werden z.B. Schreibweisen wie *centre*, *colour*, *programme*, *speciality*, *travelled* usw. sowie die Schreibung *-ise* in Wörtern wie *standardise*.

**-x**
Für jedes Wort wird mit = der denkbare Stamm ausgegeben.

**-l**
*spell* bearbeitet alle Include-Dateien.

**-i**
*spell* ignoriert alle Include-Dateien.

**+**lokaldatei
Alle Wörter, die in *lokaldatei* stehen, werden nicht ausgegeben. *lokaldatei* ist eine von Ihnen erstellte Datei, in der eine sortierte Wortliste enthalten ist (ein Wort pro Zeile). Die in dieser Datei enthaltenen Wörter sieht *spell* als korrekt geschrieben an.

## BEISPIEL

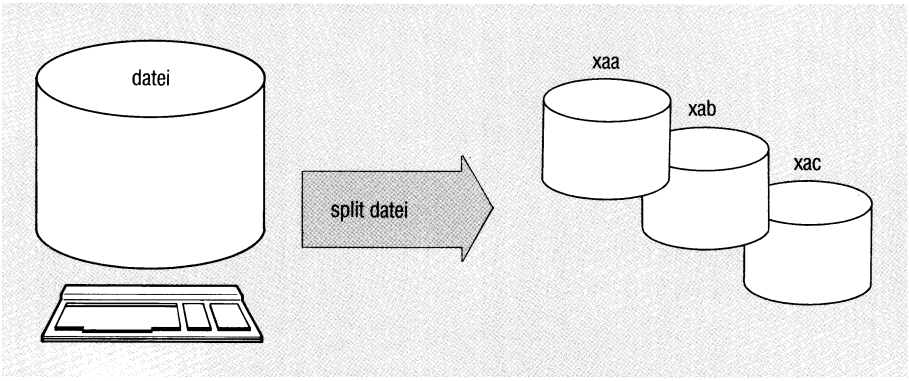
Die Datei *farben* soll anhand der in *liste* enthaltenen Wörter nach Rechtschreibfehlern überprüft werden:

```
$ cat farben
bleu gelb roth grun lila schwarz weis orrange

$ cat liste
blau
gelb
grün
lila
orange
rot
schwarz
weiß

$ spell -v +liste farben
bleu
grun
orrange
roth
weis
```

Die Ausgabe zeigt alle Wörter aus *farben*, die nicht in *liste* enthalten sind.

## NAME

**split** - Datei auf mehrere Dateien aufteilen (split a file into pieces)




## DEFINITION

**split[␣-n][␣datei[␣name]]**

## BESCHREIBUNG

Mit dem Kommando *split* können Sie Dateien in kleinere Abschnitte aufteilen. Die Abschnitte werden in einzelne Ausgabedateien geschrieben.

## SCHALTER

-n

Anzahl der Zeilen, die die einzelnen Abschnitte der aufgeteilten Datei haben sollen.
Standard (keine Angabe): 1000

## OPERANDEN

datei

Name der Datei, die in Abschnitte aufgeteilt werden soll. Geben Sie für *datei* das Zeichen - an, liest *split* von Standard-Eingabe. Standard (keine Angabe): *split* liest von Standard-Eingabe.

name
Name der Ausgabedateien, wobei die erste Ausgabedatei den Namen *name***aa**, die zweite den Namen *name***ab** usw. erhält.

*name* darf maximal 12 Zeichen lang sein. Fehlt *name*, wird der Name *x* eingesetzt.

## BEISPIEL

1. Der Inhalt der Datei *beispiel* ist zu je 20 Zeilen auf verschiedene Dateien zu verteilen:

```
$ split -20 beispiel
$ ls
beispiel
xaa
xab
xac
xad
```

2. Je zwei Zeilen der Standard-Eingabe sind in Dateien namens *aus.*. zu schreiben:

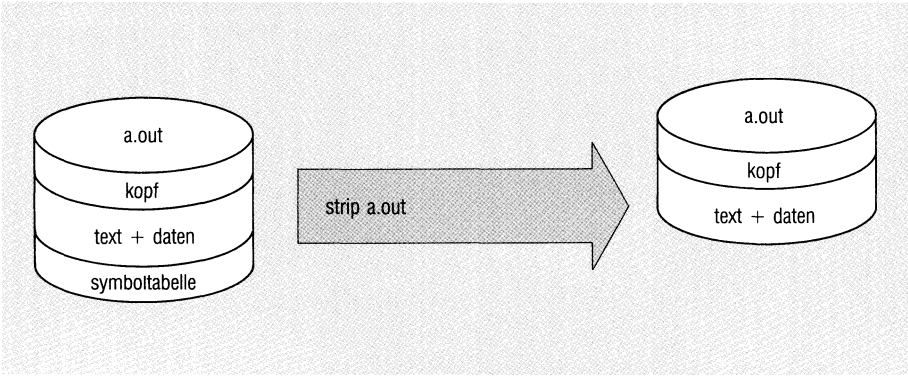
```
$ split -2 - aus
Was wahr ist, war immer wahr
und wird immer wahr bleiben.
Was aber nicht wahr ist, war nie wirklich
und wird nie wirklich werden.
END

$ ls
ausaa
ausab

$
```

## SIEHE AUCH

*csplit*(1)

## NAME

**strip** - Symboltabelle entfernen




## DEFINITION

**strip** datei...

## BESCHREIBUNG

*strip* entfernt die Symboltabelle, die normalerweise in Objektmodulen und ausführbaren Programmen steht.

*strip* hilft Speicherplatz sparen, da sich dadurch die Dateigröße verringert. Sie sollten *strip* aber nur auf ein fehlerfreies Programm anwenden, da eine Fehlersuche ohne Symboltabelle schwieriger ist.

Die Wirkung von *strip* entspricht der des Schalters *-s* bei *ld*(1D).

## OPERANDEN

datei

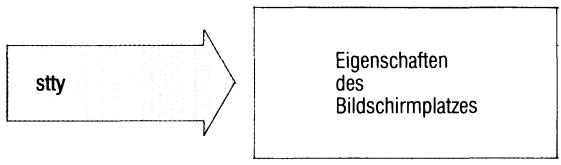
*datei* ist der Name eines Objektmoduls oder eines ablauffähigen Programms. Sie können auch mehrere Namen angeben.

## SIEHE AUCH

*ar*(1), *cc*(1D), *ld*(1D).

## NAME

**stty** - Eigenschaften einer Datensichtstation ändern




## DEFINITION

**stty**[␣schalter]

## BESCHREIBUNG

Mit *stty* kann man für das Gerät, das die Standard-Eingabe für *stty* ist, Ein-/Ausgabe-Eigenschaften einstellen oder die aktuell gesetzten Eigenschaften abfragen.

## SCHALTER

**Schalter, die die Ausgabe von *stty* betreffen**

kein Schalter
gibt einige eingestellte Eigenschaften aus;

**-a**
gibt alle Einstellungen aus;

**-g**
gibt die aktuellen Einstellungen in einem Format aus, das als Eingabe für ein weiteres *stty*-Kommando verwendet werden kann.

**Schalter zum Einstellen der Ein-/Ausgabe-Eigenschaften**

Ausführliche Information über die Eigenschaften der ersten fünf Gruppen finden Sie bei *ioctl*(2). Die Eigenschaften der letzten Gruppe werden eingestellt, indem Schalter aus den ersten fünf Gruppen kombiniert werden. Viele Kombinationen sind nicht sinnvoll, allerdings wird das von *stty* nicht überprüft.

Es gibt folgende Schalter:

**Kontrollmerkmale**

**parenb (-parenb)**
Einschalten (ausschalten) der Erzeugung und Erkennung des Paritätsbits.

**parodd (-parodd)**
Ungerade (gerade) Parität einstellen.

**cs5 cs6 cs7 cs8**
Zeichengröße wählen.

**0**
"Telefon"-Leitungen sofort abbauen. Das gilt für alle Datensichtstationsleitungen - nicht nur für Modem-Leitungen. Das Signal SIGHUP wird an alle Prozesse gesendet, die mit diesen Datensichtstationen verbunden sind.

n
Datenübertragungsrate, falls möglich, auf *n* Baud setzen. (Nicht jede Hardware-Schnittstelle unterstützt jede Geschwindigkeit!)

**hupcl (-hupcl)**
Bei dem letzten *close* soll die Leitung abgebaut (nicht abgebaut) werden.

**hup (-hup)**
Wie *hupcl* (*-hupcl*)

**cstopb (-cstopb)**
zwei (ein) Stop-Bit pro Zeichen

**cread (-cread)**
Empfänger einschalten (ausschalten)

**clocal (-clocal)**
Die Leitung soll ohne (mit) Modemsteuerung betrieben werden.

**loblk (-loblk)**
Ausgabe eines nicht aktuellen Layer (nicht) blockieren.

**Eingabe-Merkmale**

**ignbrk (-ignbrk)**
Bei Eingabe Break (Unterbrechungstaste)(nicht) ignorieren.

**brkint (-brkint)**
Bei Eingabe von Break SIGINT senden (nicht senden).

**ignpar (-ignpar)**
Paritätsfehler ignorieren (nicht ignorieren).

**parmrk (-parmrk)**
Paritätsfehler markieren (nicht markieren).

**inpck (-inpck)**
Bei Eingabe Paritätsprüfung einschalten (ausschalten).

**istrip (-istrip)**
Eingabe-Zeichen auf 7 Bit maskieren (nicht maskieren).

**inlcr (-inlcr)**
Bei Eingabe Neue-Zeile-Zeichen (nicht) auf Wagenrücklauf abbilden.

**igncr (-igncr)**
Bei Eingabe Wagenrücklauf ignorieren (nicht ignorieren).

**icrnl (-icrnl)**
Bei Eingabe Wagenrücklauf (nicht) auf Neue-Zeile-Zeichen abbilden.

**iuclc (-iuclc)**
Bei Eingabe Großbuchstaben (nicht) in Kleinbuchstaben umwandeln.

**ixon (-ixon)**
Die Ausgabe soll (nicht) mit einer START/STOP Kontrolle arbeiten. (ASCII DC3 = STOP, ASCII DC1 = START).

**ixany (-ixany)**
Die Ausgabe soll durch ein beliebiges Zeichen (nur durch DC1) fortgesetzt werden.

**ixoff (-ixoff)**
Das System soll (nicht) START/STOP Zeichen senden, wenn die Eingabe-Warteschlange fast leer/voll ist.

**Ausgabe-Merkmale**

**opost (-opost)**
Ausgabe (nicht) nachbearbeiten; alle anderen Ausgabe-Merkmale ignorieren.

**olcuc (-olcuc)**
Bei Ausgabe Kleinbuchstaben in Großbuchstaben (nicht) umwandeln.

**onlcr (-onlcr)**
Bei Ausgabe Neue-Zeile-Zeichen (nicht) auf Wagenrücklauf abbilden.

**ocrnl (-ocrnl)**
Bei Ausgabe Wagenrücklauf (nicht) auf Neue-Zeile-Zeichen abbilden.

**onocr (-onocr)**
Wagenrücklauf nicht (doch) auf Spalte 0 ausgeben.

**onlret (-onlret)**
Bei Ausgabe Wagenrücklauffunktion durch Neue-Zeile-Zeichen (nicht) ausführen.

**ofill (-ofill)**
Füllzeichen (Zeitabstimmung) für Verzögerungen verwenden.

**ofdel (-ofdel)**
Abbruchzeichen (Null-Zeichen) als Füllzeichen verwenden.

**cr0 cr1 cr2 cr3**
Verzögerungsart für Wagenrücklauf wählen.

**nl0 nl1**
Verzögerungsart für Neue-Zeile-Zeichen wählen.

**tab0 tab1 tab2 tab3**
Verzögerungsart für Tabulatorzeichen wählen.

**bs0 bs1**
Verzögerungsart für Rücksetztaste (backspace) wählen.

**ff0 ff1**
Verzögerungsart für Seitenvorschub wählen.

**vt0 vt1**
Verzögerungsart für Vertikaltabulator wählen.

**Lokale Merkmale**

**isig (-isig)**
Die Zeichen sollen (nicht) auf INTR und QUIT und SWTCH abgeprüft werden.

**icanon (-icanon)**
Kanonische Eingabebehandlung; Die Sonderfunktionen der Zeichen ERASE und KILL sollen bei Eingabe dieser Zeichen (nicht) ausgeführt werden.

**xcase (-xcase)**
Kleinbuchstaben sollen (nicht) in Großbuchstaben umgewandelt werden, und umgekehrt.

**echo (-echo)**
Jedes eingegebene Zeichen soll (nicht) angezeigt werden.

**echoe (-echoe)**
ERASE-Zeichen sollen als Zeichenfolge Rücksetzungszeichen-Leerzeichen-Rücksetzungszeichen angezeigt werden (Rücksetzungszeichen = backspace).

*Hinweis*
In diesem Modus wird das mit ERASE überschriebene Zeichen auf vielen Datensichtstationen gelöscht; da jedoch die jeweils aktuelle Spalte nicht gemerkt wird, kann es bei entwerteten Zeichen, Tabulatorzeichen und Rücksetzungszeichen Problemen kommen.

**echok (-echok)**
nach dem KILL-Zeichen Neue-Zeile-Zeichen (nicht) ausgeben.

**echonl (-echonl)**
Neue-Zeile-Zeichen (nicht) ausgeben.

**noflsh (-noflsh)**
Der Puffer soll nach INTR, QUIT und SWTCH (nicht) gelöscht werden.

### Zuweisungen an Steuerzeichen

Steuerzeichen c
Dem *Steuerzeichen* wird *c* zugeordnet; *Steuerzeichen* kann dabei sein: ERASE, KILL, INTR, QUIT, SWTCH, EOF, EOL, MIN, oder TIME (MIN und TIME werden bei *-icanon* benutzt). Ist *c* das Zeichen ^ vorangestellt, so wird es als CTRL-Zeichen interpretiert (z.B. entspricht ^D einem [CTRL] D); ^? wird als [DEL] interpretiert, ^- als nicht definiertes Zeichen.

**line** i
Das Leitungsprotokoll *i* ($0 < i < 127$) soll verwendet werden.

### Kombinierte Merkmale

**evenp** oder **parity**
*parenb* und *cs7* einschalten.

**oddp**
*parenb*, *cs7* und *parodd* einschalten.

**-parity**, **-evenp** oder **-oddp**
*parenb* ausschalten, *cs8* einschalten.

**raw (-raw** oder **cooked)**
Raw-Ein- und Ausgabemodus einschalten (ausschalten); der Raw-Ausgabemodus entspricht *cs*8, ohne ERASE, KILL, INTR, QUIT, SWTCH, EOT, ohne Nachverarbeitung der Ausgabe und ohne Parität.

**nl (-nl)**
*icrnl* und *onlcr* ausschalten (einschalten). Bei *-nl* wird zusätzlich *inlcr*, *igncr*, *ocrnl*, und *onlret* ausgeschaltet.

**lcase (-lcase)**
*xcase*, *iuclc* und *olcuc* einschalten (ausschalten).

**LCASE (-LCASE)**
Wie *lcase* (*-lcase*).

**tabs (-tabs** oder **tab8)**
Tabulatorzeichen bei der Ausgabe unverändert ausgeben (in Leerzeichen expandieren).

**ek**
Die Zeichen ERASE und KILL werden auf die Standardwerte zurückgesetzt.

**sane**
Alle Merkmale werden auf sinnvolle Werte zurückgesetzt.

## BEISPIEL

1. Abfragen der aktuellen Einstellung:

```
$ stty -a
speed 38400 baud; line = 2; intr = DEL; quit = ^|; erase = ^h;
kill = ^x; eof = ^d; eol <undef>; swtch <undef>
parenb parodd cs7 -cstopb hupcl cread clocal -loblk
ixon -ixany -ixoff
isig icanon -xcase echo echoe echok -echonl -noflsh
opost -olcuc onlcr -ocrnl -onocr -onlret -ofill -ofdel
```

2. Ändern der Einstellung, so daß weitere Eingaben an der Tastatur nicht am Bildschirm ausgegeben werden. Die Einstellung kann man dann nur blind abfragen:

```
$ stty -echo
$ stty -a
speed 38400 baud; line = 2; intr = DEL; quit = ^|; erase = ^h;
kill = ^x; eof = ^d; eol <undef>; swtch <undef>
parenb parodd cs7 -cstopb hupcl cread clocal -loblk
ixon -ixany -ixoff
isig icanon -xcase -echo echoe echok -echonl -noflsh
opost -olcuc onlcr -ocrnl -onocr -onlret -ofill -ofdel
```

3. Wiederherstellen der Darstellung der Eingaben am Bildschirm:

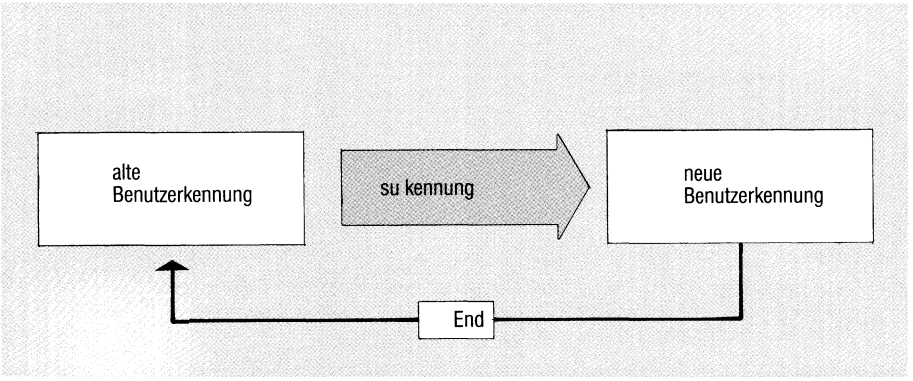
```
$ stty echo
$
```

## SIEHE AUCH

*ioctl*(2), *termio*(7)

## NAME

**su** - Benutzerkennung vorübergehend wechseln (become super-user or another user)




## DEFINITION

**su**[␣-][␣name[␣arg...]]

## BESCHREIBUNG

Mit *su* können Sie vorübergehend unter einer anderen Benutzerkennung arbeiten, ohne daß Sie sich vorher vom System abmelden müssen.

## ARBEITSWEISE

*su* kann nur aufgerufen werden, wenn das geeignete Kennwort eingegeben wird, es sei denn, der Systemverwalter ruft das Kommando auf. Ist das Kennwort korrekt, so ruft *su* eine neue Shell auf, wobei die reale und effektive Benutzerkennung auf diejenige des angegebenen Benutzers gesetzt wird.

Neuer Kommandointerpreter wird das in */etc/passwd* für den angegebenen Benutzer definierte Programm. Wenn keines definiert ist, wird der Standard-Kommandointerpreter *sh*(1) verwendet. Durch die Eingabe von END können die normalen Benutzer-Privilegien wiederhergestellt werden.

## OPERANDEN

-

Wenn Sie den Operanden - angeben und in der Kennwortdatei für den Benutzer *name* *sh*(1) als Kommandointerpreter eingetragen ist, dann wird die Umgebung so abgeändert, als würden Sie sich als der angegebene Benutzer *name* anmelden.

Wenn - fehlt und in der Kennwortdatei für den Benutzer *name* *sh*(1) als Kommandointerpreter eingetragen ist, dann wird Ihre Umgebung unverändert weitergegeben; eine Ausnahme bildet hier die Umgebungsvariable PATH: Wenn Sie als neuer Benutzer der Systemverwalter sind, wird PATH auf einen "sicheren" Wert gesetzt.

name

Benutzerkennung, unter der Sie vorübergehend arbeiten möchten. Sie benötigen hierfür das Kennwort von *name*, nachdem Sie nach der Eingabe von *su name* gefragt werden.

Nur der Systemverwalter kann ohne Kennwort unter fremden Kennungen arbeiten.

Zur neuen Umgebung siehe Beschreibung des Operanden!

Standard (keine Angabe): Sie arbeiten als Systemverwalter weiter, vorausgesetzt, Sie wissen das Kennwort.

arg...

Stehen auf der Kommandozeile weitere Argumente, so werden sie an den Kommandointerpreter weitergegeben.

## DATEIEN

*/etc/passwd*
Systemweite Kennwort-Datei

*/etc/profile*
Systemweite Profildatei

*$HOME/.profile*
Individuelle Profildatei

**HINWEIS**

Am Benutzernamen ändert sich bei diesem Kommando nichts; (siehe *who*(1)).

**BEISPIEL**

Sie möchten Systemverwalterfunktionen ausführen:

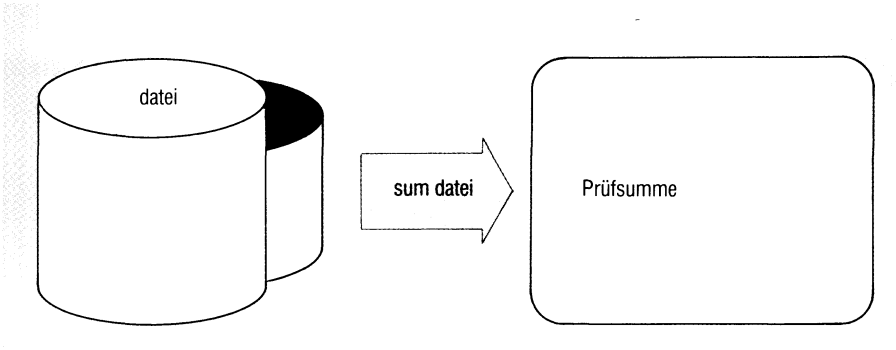
```
$ su
password: kennwort eingeben
# chown ...
# END
$
```

**SIEHE AUCH**

*sh*(1), *login*(1), *newgrp*(1)

## NAME

**sum** - Prüfsumme einer Datei berechnen (print checksum and block count of a file)




## DEFINITION

**sum**[␣**-r**][␣datei...]

## BESCHREIBUNG

*sum* berechnet für die angegebene Datei eine Prüfsumme und gibt sie aus; außerdem gibt es an, wieviel Speicherplatz (Einheit: 512 Byte) durch die Datei beansprucht wird.

## SCHALTER

**-r**

Zur Berechnung der Prüfsumme wird ein anderer Algorithmus verwendet.

## OPERANDEN

datei

Name der Datei, deren Prüfsumme berechnet werden soll.

keine Angabe

*sum* liest von Standard-Eingabe.

**BEISPIEL**

1. Prüfsumme der Datei *monate* nach dem ersten Algorithmus

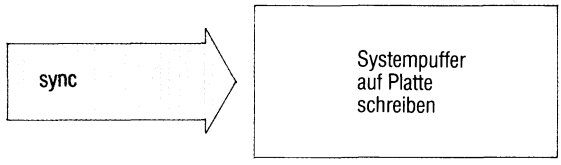
```
$ sum monate
6658 1 monate
```

2. Prüfsumme der Datei *monate* nach dem zweiten Algorithmus

```
$ sum -r monate
62412     1 monate
$
```

## NAME

**sync** - Systempuffer zurückschreiben




## DEFINITION

**sync**

## BESCHREIBUNG

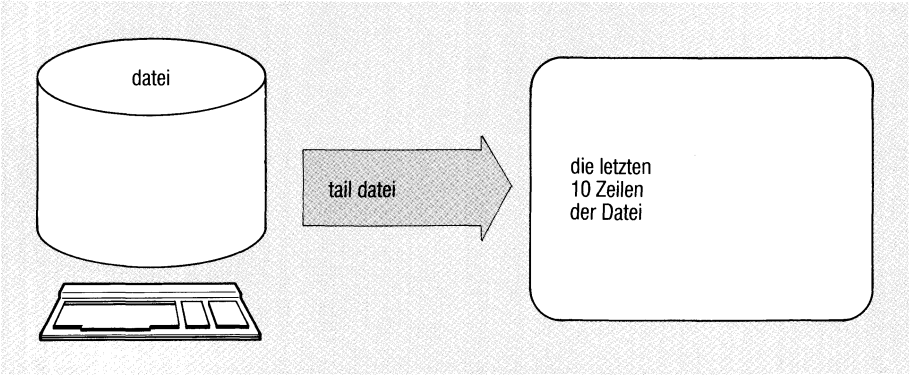
*sync* bewirkt, daß alle bisher noch nicht auf Festplatte oder Diskette geschriebenen Systempuffer herausgeschrieben werden. Wenn das System heruntergefahren werden soll, muß vorher *sync* aufgerufen werden, um sicherzustellen, daß alle noch nicht gespeicherten Änderungen abgesichert werden. Einzelheiten siehe *sync*(2).

## SIEHE AUCH

*sync*(2).

## NAME

**tail** - Den letzten Teil einer Datei ausgeben




## DEFINITION

**tail[␣+/-[anzahl]][schalter][␣datei]**

## BESCHREIBUNG

*tail* schreibt den Inhalt der angegebenen Datei ab einer festgelegten Stelle auf die Standard-Ausgabe.

## SCHALTER

**l**

**b**

**c**

Nur zusammen mit *+anzahl* oder *-anzahl*,
Beschreibung siehe *OPERANDEN*, *anzahl*
Standard: Schalter *l*

**-f**

Wenn die Eingabedatei keine Pipe ist, so wird das Programm nach dem Kopieren der Eingabezeilen nicht beendet, sondern tritt in eine Endlosschleife ein. Dabei wartet es mindestens eine Sekunde lang und versucht dann, weitere Zeilen der Eingabedatei zu lesen und zu kopieren. Somit kann mit *tail* das Wachstum einer Datei überwacht werden, in die durch einen anderen Prozeß geschrieben wird.

So werden z.B. mit dem Kommando

```
tail -f fred
```

die letzten 10 Zeilen der Datei *fred* gelesen, dann alle Zeilen, die während der Durchführung von *tail* an *fred* angehängt werden. Mit dem Kommando

```
tail -15cf fred
```

werden die letzten 15 Zeichen in der Datei *fred* ausgegeben, dann alle Zeilen, die an *fred* zwischen dem Aufrufen und Beenden von *tail* angehängt werden.

**OPERANDEN**

+anzahl
−anzahl

Zusammen mit Schalter *l*
Mit *anzahl* geben Sie an, in der wievielten Zeile vom Dateianfang an (+ *anzahl*) bzw. vom Dateiende an (-*anzahl*) *tail* mit der Ausgabe beginnen soll.

Zusammen mit Schalter *b*
Mit *anzahl* geben Sie an, im wievielten Block zu 512 Byte vom Dateianfang an (+ *anzahl*) bzw. vom Dateiende an (-*anzahl*) *tail* mit der Ausgabe beginnen soll.

Zusammen mit Schalter *c*
Mit *anzahl* geben Sie an, beim wievielten Zeichen vom Dateianfang an (+ *anzahl*) bzw. vom Dateiende an (-*anzahl*) *tail* mit der Ausgabe beginnen soll.

Standard (keine Angabe für *anzahl*): -10

datei

Name der Datei, die *tail* ausgeben soll. Standard (keine Angabe): *tail* liest von Standard-Eingabe

**HINWEIS**

- Abschnitte, die vom Dateiende aus gezählt werden, werden von *tail* zwischengespeichert; es werden daher maximal 4K ausgegeben.
- Wird *tail* auf zeichenorientierten Gerätedateien verwendet, so kann es zu Problemen kommen.

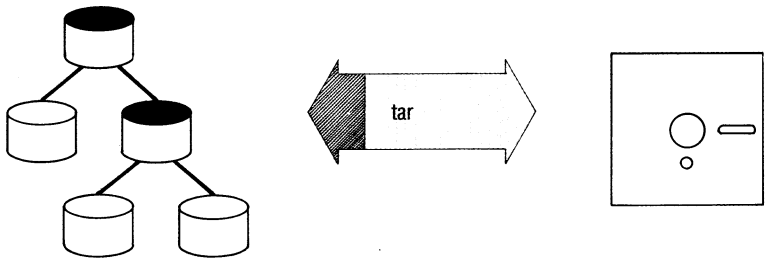
**BEISPIEL**

Ausgeben der letzten 5 Zeilen der Datei *monate*, in der jeweils in einer Zeile ein Monatsname steht:

```
$ tail -5 monate
August
September
Oktober
November
Dezember
$
```

## NAME

**tar** - Dateien archivieren (file archiver)




## DEFINITION

**tar␣schalter[␣datei...]**

## BESCHREIBUNG

*tar* erstellt Archive von Dateien oder liest sie ein.

## SCHALTER

*schalter* bestehen hier aus einer Funktion (ein Buchstabe ohne - Gedankenstrich davor) und aus einem oder mehreren optionalen Attributen, die Sie unmittelbar an die Funktion anhängen. Es muß mindestens eine Funktion angegeben werden.

### Funktionen

**r**

Die angegebene(n) *datei*(en) wird bzw. werden ans Ende eines bestehenden Archivs angehängt.

**x**

Die angegebene(n) *datei*(en) wird bzw. werden aus dem Archiv eingelesen. Ist *datei* der Name eines Dateiverzeichnisses, dessen Inhalt in das Archiv geschrieben worden ist, so wird der Inhalt dieses Dateiverzeichnisses rekursiv kopiert. Ist eine archivierte *datei* nicht im System vorhanden, so wird die Datei erstellt; sie erhält dieselben

Modi wie die archivierte Datei, mit Ausnahme des s-Bits für Eigentümer und Gruppe, die nicht übernommen werden. (Letzteres gilt wiederum nur, wenn der Benutzer nicht der Systemverwalter ist). Ist die Datei vorhanden, so werden ihre Zugriffsrechte nicht geändert, außer wie oben beschrieben. Eigentümer, Gruppe und der Zeitpunkt der letzten Änderung werden, falls möglich, von der archivierten Datei übernommen. Fehlt *datei*, so wird der gesamte Inhalt des Archivs eingelesen. Haben mehrere Dateien im Archiv denselben Namen, so wird deren Inhalt durch den Inhalt der zuletzt kopierten Datei überschrieben.

**t**

Ein Inhaltsverzeichnis des Archivs wird ausgegeben.

**u**

Die angegebene(n) *datei*(en) werden an das Ende des bestehenden Archivs angehängt, wenn sie noch nicht im Archiv sind oder verändert worden sind seit dem Zeitpunkt, als sie das letzte Mal ins Archiv geschrieben wurden.

**c**

Ein neues Archiv wird erstellt; das Sichern beginnt am Anfang des Archivs, nicht wie sonst hinter der letzten Datei des Archivs.

### Attribute

#s

Mit # wird die Laufwerksnummer eines Magnetbandgerätes (0, 1, ... ,7) angegeben; mit *s* wird die Schreibdichte (*l* - 800 bpi, *m* - 1600 bpi oder *h* - 6250 bpi) angegeben.

**v**

Normalerweise gibt *tar* kein Protokoll aus; wird jedoch die Zusatzfunktion *v* (verbose) verwendet, so wird für jede bearbeitete Datei der Buchstabe der Funktion, gefolgt vom Namen der Datei, ausgegeben. Mit der Funktion *t* werden zusätzlich zu den Dateinamen weitere Informationen ausgegeben.

**w**

*tar* gibt die auszuführende Aktion aus, gefolgt vom Namen der Datei; dann wartet es auf die Bestätigung durch den Benutzer. Wird eine Eingabe angegeben, die mit *y* anfängt, wird die Aktion durchgeführt. Jede andere Eingabe bedeutet "nein". In Verbindung mit der Funktion *t* wird dieses Attribut ignoriert.

**f**

*tar* interpretiert das nächste Argument als Name des Archivs, das anstelle des systemabhängigen Standard-Archivs benutzt werden soll. Wird anstelle eines Dateinamens das Zeichen - angegeben, so schreibt *tar* auf die Standard-Ausgabe bzw. liest von der Standard-Eingabe. Somit kann *tar* als Anfang oder Ende einer Pipeline verwendet werden. Im folgenden Beispiel wird *tar* benutzt, um Hierarchien innerhalb des Dateibaums zu verschieben (der Unterbaum des Dateisystems, der am Dateiverzeichnis dv1 hängt, wird an das Dateiverzeichnis dv2 angehängt):

```
(cd dv1; tar cf - .) | (cd dv2; tar xf -)
```

**b**

*tar* interpretiert das nächste Argument als Blockungsfaktor. Standardmäßig wird der Blockungsfaktor 1 verwendet; er kann maximal den Wert 20 annehmen. Dieses Attribut sollte nur bei Magnetbändern verwendet werden (siehe Attribut *f*, oben). Die Blockgröße wird beim Lesen von Magnetbändern (Funktionen *x* and *t*) automatisch festgestellt.

**l**

*tar* soll mitteilen, wenn es nicht alle Verweise auf die Dateien, die in das Archiv geschrieben werden, auflösen kann. Fehlt *l*, so werden keine Meldungen ausgegeben. Dieses Attribut ist nur in Kombination mit den Funktionen *c*, *r* und *u* sinnvoll.

**m**

*tar* soll das Änderungsdatum nicht von der gesicherten Datei übernehmen, sondern auf das aktuelle Datum setzen. Dieses Attribut darf bei der Funktion *t* nicht verwendet werden.

o

Die kopierten Dateien sollen die Benutzer- und Gruppenkennung des Benutzers erhalten, der das Programm aufgerufen hat (andernfalls erhalten sie diejenigen aus dem Archiv). Dieses Attribut ist nur mit der Funktion *x* sinnvoll.

## OPERANDEN

datei

Namen der Datei(en) (bzw. Dateiverzeichnisse), die archiviert oder aus dem Archiv gelesen werden sollen. Der Name eines Dateiverzeichnisses bezieht sich in jedem Fall auf alle Dateien und Unterverzeichnisse, die in diesem Dateiverzeichnis enthalten sind.

## FEHLER

*tar* gibt Meldungen aus, wenn falsche Schalter angegeben sind oder beim Lesen bzw. Schreiben ein Fehler auftritt. Auch wenn für die Verweistabellen nicht genügend Speicherkapazität zur Verfügung steht, wird eine Fehlermeldung ausgegeben.

## BEISPIEL

1. Im folgenden Beispiel werden alle im aktuellen Dateiverzeichnis und seinen Unterverzeichnissen enthaltenen Dateien auf eine Diskette geschrieben; gleichzeitig werden die Namen der Dateien ausgegeben:

```
tar cvf /dev/fl2
```

2. Die Datei *dokumente* wird auf Magnetbandkassette archiviert. Ein neues Archiv wird erstellt, Blockungsfaktor ist 20, die Geräteadresse lautet */dev/rts0* :

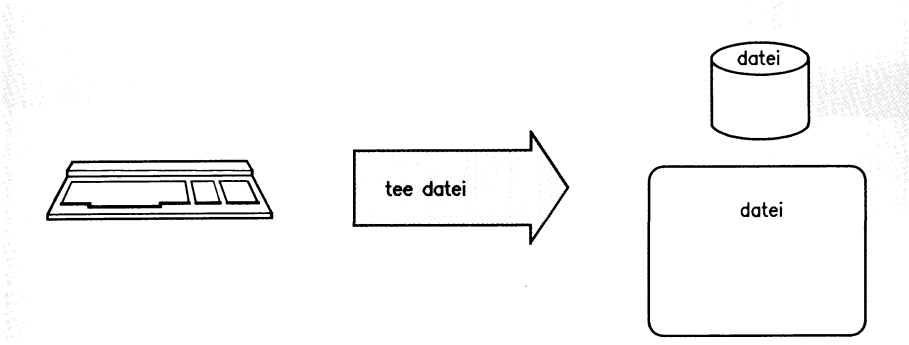
```
$tar cvbf 20 /dev/rts0 dokumente
```

## SIEHE AUCH

*cpio*(1)

## NAME

**tee** - Pipes zusammenfügen und Eingabe kopieren




## DEFINITION

**tee[␣-i][␣-a][␣datei...]**

## BESCHREIBUNG

*tee* überträgt Daten von der Standard-Eingabe auf die Standard-Ausgabe und setzt gleichzeitig eine Kopie in die angegebene(n) *datei*(en).

## SCHALTER

**-i**

Unterbrechungssignale werden ignoriert.

**-a**

Die Ausgabe von *tee* überschreibt *datei* nicht, sondern wird am Ende von *datei* angefügt.

## OPERANDEN

datei

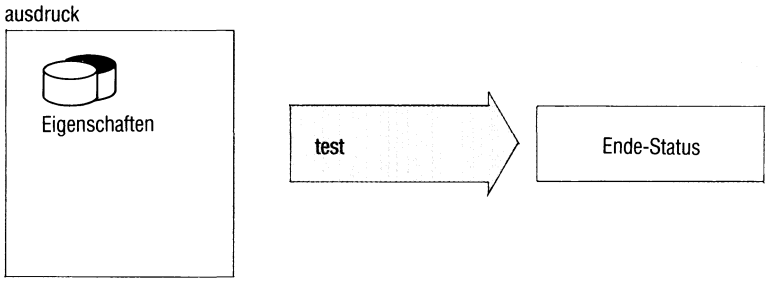
Name der Datei, in die die Ausgabe geschrieben werden soll.

## BEISPIEL

```
$ tee ausgabe
Dieser Satz kommt in die Datei "ausgabe".
Dieser Satz kommt in die Datei "ausgabe".
END

$ tee -a ausgabe
Durch den Schalter -a wird die Datei "ausgabe" nicht überschrieben.
Durch den Schalter -a wird die Datei "ausgabe" nicht überschrieben.
END

$ cat ausgabe
Dieser Satz kommt in die Datei "ausgabe".
Durch den Schalter -a wird die Datei "ausgabe" nicht überschrieben.
$
```

## NAME

**test** - Bedingungen prüfen




## DEFINITION

**test**␣*ausdruck*
**[**␣*ausdruck*␣**]**

## BESCHREIBUNG

*test* wertet die Bedingung in *ausdruck* aus; ist die Bedingung erfüllt, so liefert es den Wert 0 (wahr), andernfalls einen Wert ungleich 0 (falsch) zurück. Auch wenn keine Argumente angegeben sind, liefert *test* einen Wert ungleich 0 zurück.

! Wenn Sie die zweite Form des Kommandos verwenden, d.h. die eckigen Klammern, dann müssen die Klammern von *ausdruck* durch Leerzeichen getrennt sein.

*ausdruck* kann folgende Bedingungen enthalten:

**-r** datei
wahr, wenn *datei* vorhanden ist und der Benutzer die Leseerlaubnis hat.

**-w** datei
wahr, wenn *datei* vorhanden ist und der Benutzer die Schreiberlaubnis hat.

**-x** datei
wahr, wenn *datei* vorhanden ist und der Benutzer die Ausführerlaubnis hat.

**-f** datei
wahr, wenn *datei* vorhanden und eine normale Datei ist.

**-d** datei
wahr, wenn *datei* vorhanden und ein Dateiverzeichnis ist.

**-c** datei
wahr, wenn *datei* vorhanden und eine zeichenorientierte Gerätedatei ist.

**-b** datei
wahr, wenn *datei* vorhanden und eine blockorientierte Gerätedatei ist.

**-p** datei
wahr, wenn *datei* vorhanden und eine named pipe (FIFO) ist.

**-u** datei
wahr, wenn *datei* vorhanden ist und für den Eigentümer das s-Bit gesetzt ist.

**-g** datei
wahr, wenn *datei* vorhanden ist und für die Gruppe das s-Bit gesetzt ist.

**-s** datei
wahr, wenn *datei* vorhanden und nicht leer ist.

**-t** [dateikennzahl]
wahr, wenn die geöffnete Datei mit der angegebenen *dateikennzahl* (Standard: 1) mit einer Datensichtstation verbunden ist.

**-z**s1
wahr, wenn die Länge der Zeichenfolge *s1* 0 ist.

**-n**s1
wahr, wenn die Länge der Zeichenfolge *s1* ungleich 0 ist.

s1 = s2
wahr, wenn die Zeichenfolgen *s1* und *s2* identisch sind.

s1 != s2
wahr, wenn die Zeichenfolgen *s1* und *s2* unterschiedlich sind.

s1
wahr, wenn die Zeichenfolge *s1* nicht leer ist.

! Zeichenfolgen werden nur bis zum ersten Leerzeichen verglichen, auch wenn sie in Anführungszeichen stehen.

n1 **-eq** n2
wahr, wenn die ganzen Zahlen *n1* und *n2* algebraisch gleich sind. Anstelle von *eq* kann auch einer der Vergleichsoperatoren *-ne* (ungleich), *-gt* (größer), *-ge* (größer gleich), *-lt* (kleiner) und *-le* (kleiner gleich) verwendet werden.

Zur Verknüpfung können die folgenden Operatoren verwendet werden:

**!**
Verneinung

**-a**
Logisches UND

**-o**
Logisches ODER (*-a* hat eine höhere Priorität als *-o*).

**(**ausdruck**)**
Klammerung zur Zusammenfassung von Bedingungen

Alle Operatoren und Parameter werden von *test* als eigenständige Argumente interpretiert. Klammern können für die Shell bedeutungstragende Zeichen sein; daher müssen sie eventuell entwertet werden.

**HINWEIS**
Dieses Kommando wird in der Regel direkt von der Shell ausgeführt.

## BEISPIEL

1. Die folgende Shell-Prozedur prüft, ob der angegebene Parameter der Name der Datei oder eines Dateiverzeichnisses ist.

```
if test -f "$1"
then
echo $1 ist eine Datei
elif test -d "$1"
then
echo $1 ist ein Dateiverzeichnis
fi
```

2. Zweite Schreibweise:

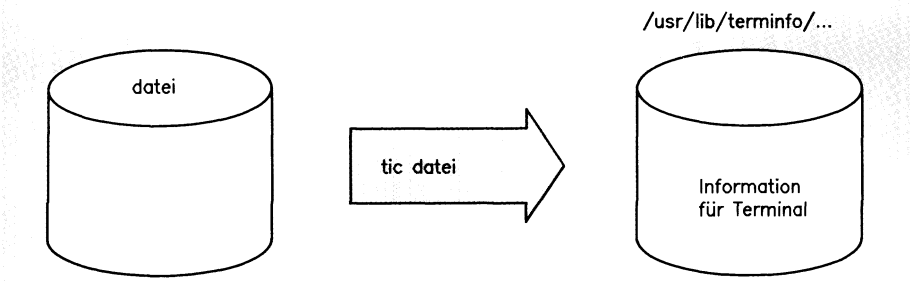
```
Anstelle von            if test -f "$1"
können Sie schreiben:   if [ -f "$1" ]
```

## SIEHE AUCH

*find*(1), *sh*(1)

## NAME

**tic** - Informationen über Datensichtstation übersetzen (terminfo compiler)




## DEFINITION

**tic[␣-v[n]]␣datei...**

## BESCHREIBUNG

*tic* übersetzt Informationen über die benutzte Datensichtstation ("terminal information"). Die Ergebnisse werden in das Dateiverzeichnis */usr/lib/terminfo* geschrieben.

*tic* übersetzt alle *terminfo*-Beschreibungen, die in den angegebenen Dateien enthalten sind. Wird das Feld *use=* gefunden, so durchsucht *tic* zunächst die aktuelle Datei und dann die Datei *./terminfo.src*.

Ist die Umgebungsvariable *TERMINFO* gesetzt, so werden die Ergebnisse nicht in das Dateiverzeichnis */usr/lib/terminfo* geschrieben, sondern dieser Umgebungsvariablen zugewiesen.

## SCHALTER

**-v**[n]

Ist der Schalter *-v* (verbose) eingeschaltet, so gibt *tic* ein Protokoll über die Übersetzung der Informationen aus.
Geben Sie zusätzlich eine ganze Zahl *n* an, dann gibt *tic* entsprechend mehr Protokoll-Informationen aus.

**DATEIEN**

/usr/lib/terminfo/*/*
Beschreibung der Datensichtstation

**HINWEIS**

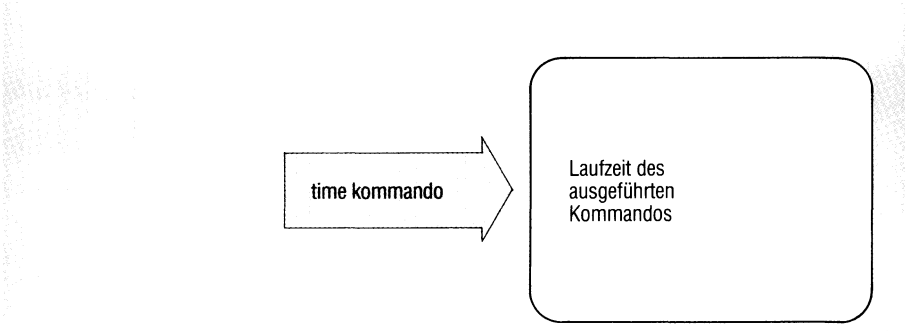
- *tic* sollte nicht die Datei *./terminfo.src* durchsuchen, sondern nach einem vorhandenen, bereits übersetzten Eintrag suchen.
- Einschränkungen:
  - Die übersetzten Einträge dürfen insgesamt eine Länge von höchstens 4096 Byte haben.
  - Das Namensfeld darf eine Länge von maximal 128 Byte haben.

**SIEHE AUCH**

*curses*(3X), *terminfo*(4).

## NAME

**time** - Laufzeit eines Kommandos messen (time a command)




## DEFINITION

**time** kommando[ arg...]

## BESCHREIBUNG

Mit *time* können Sie die Laufzeit eines Kommandos messen. *kommando* wird ausgeführt; *time* gibt dann in Sekunden folgende Zeiten aus: *real, user, sys.*

- *real* ist die Laufzeit, d.h. die Zeit zwischen Programmaufruf und -abschluß.
- *user* ist die Zeit, in der sich das Programm im Benutzermodus befunden hat.
- *sys* ist die Zeit, in der sich das Programm im Systemmodus befunden hat.

Die Informationen werden auf die Standard-Fehlerausgabe geschrieben.

An *Kommando* können Sie beim Aufruf auch Argumente *arg...* übergeben.

**HINWEIS**

Rufen Sie *time* auf einem Multiprozessor-System auf, so ist die Summe aus der Zeit im Benutzermodus und der Zeit im Systemmodus möglicherweise größer als die reale Laufzeit.

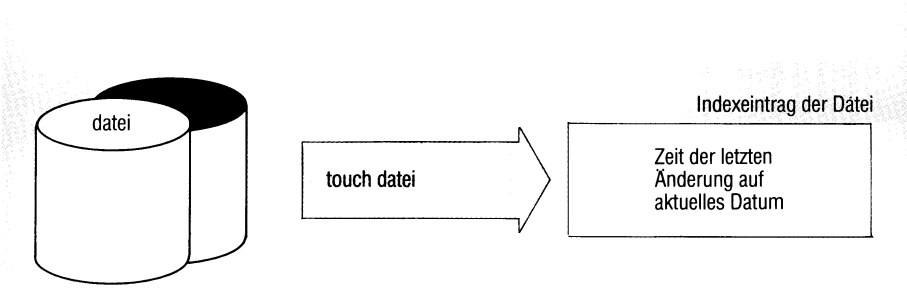
**BEISPIEL**

Die Ausführungszeit des *ls*-Kommandos soll gemessen werden. Die Ausgabe von *ls* wird umgeleitet.

```
$ time ls -l >liste
real        2.0
user        0.9
sys         0.8
```

## NAME

**touch** - Zeitpunkt der letzten Änderung auf aktuelles Datum setzen




## DEFINITION

**touch**[␣**-amc**][␣mmddhhmm[yy]]␣datei...

## BESCHREIBUNG

Mit *touch* können Sie die Zeit der letzten Änderung und die Zeit des letzten Zugriffs einer Datei auf ein gewünschtes Datum setzten. Ist die *datei* nicht vorhanden, so wird sie angelegt. Wird kein Datum angegeben, so wird das aktuelle Datum eingesetzt. Der Schalter *-a* bewirkt, daß das Datum des letzten Zugriffs, Schalter *-m*, daß das Datum der letzten Änderung aktualisiert wird (Standard: *-am*). Ist der Schalter *-c* gesetzt, so erstellt *touch* keine Datei, wenn *datei* nicht bereits vorhanden ist (es wird keine Meldung ausgegeben).

## ENDESTATUS

*touch* gibt die Zahl der Dateien aus, deren Zeiten nicht geändert werden konnte (auch die Dateien, die nicht vorhanden waren und nicht angelegt wurden, werden gezählt).

**HINWEIS**

Nur aus Zahlen bestehende Dateinamen können zu Problemen führen, da *touch* diese möglicherweise als Datumsangabe interpretiert.

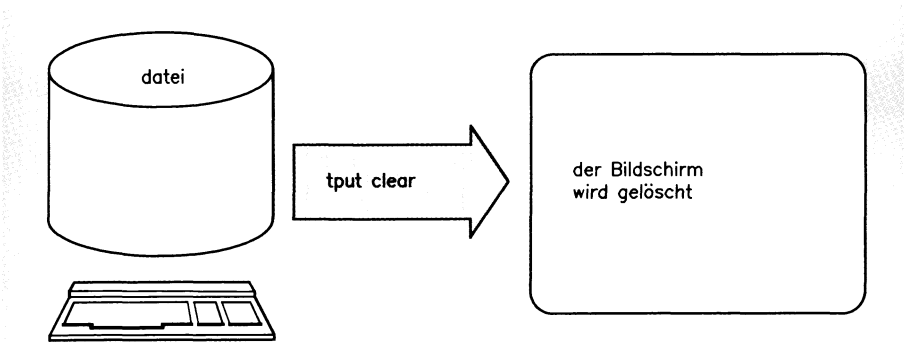
**BEISPIEL**

Das Datum des letzten Zugriffs von *datei* soll auf den 26.8. 9 Uhr gesetzt werden:

```
$ touch -a 08260900 datei
$ ls -l datei
-rwxrwxrwx 1 doro            736 Aug 26 09:00 datei
```

## NAME

**tput** - Terminfo-Datenbank abfragen




## DEFINITION

**tput**[␣**-T**typ]␣capname

## BESCHREIBUNG

*tput* benutzt die *terminfo*(4)-Datenbank, um die Eigenschaften der betreffenden Datensichtstation der Shell zur Verfügung zu stellen. *tput* liefert eine Zeichenfolge, wenn das Attribut (*cap*ability *name*) vom Typ Zeichenfolge ist, eine ganze Zahl, wenn das Attribut vom Typ ganze Zahl ist. Ist das Attribut ein Boolscher Wert, so liefert *tput* einfach einen Ende-Status (0 für WAHR, 1 für FALSCH) und erzeugt keine Ausgabe.

**-T**typ
: Der Typ der Datensichtstation. Normalerweise ist dieser Parameter nicht notwendig, da diese Angabe normalerweise der Umgebungsvariablen $TERM entnommen wird.

capname
: Das Attribut der *terminfo*-Datenbank wird ausgegeben. Siehe *terminfo*(4).

## FEHLER

Bei einem Fehler gibt *tput* Fehlermeldungen und die folgenden Fehlercodes aus:

-1 Eingabefehler
-2 Angegebener Datensichtstationtyp ungültig
-3 Angegebenes Attribut ungültig

Auch wenn für eine Datensichtstation ein *attribut* angefordert wird, bei der diesem *attribut* kein Wert zugeordnet ist, wird der Code *-1* ausgegeben.

## DATEIEN

*/etc/term/?/**
Dateien, in denen die Datensichtstation beschrieben wird.
*/usr/include/term.h*
Definitionsdatei
*/usr/include/curses.h*
Definitionsdatei

## BEISPIEL

tput clear
Die Kommandofolge, mit der die Anzeige auf der aktuellen Datensichtstation gelöscht wird, wird auf den Bildschirm geechot. Der Bildschirmschirm wird tatsächlich gelöscht.

tput cols
Die Anzahl der Spalten auf der aktuellen Datensichtstation wird ausgegeben.

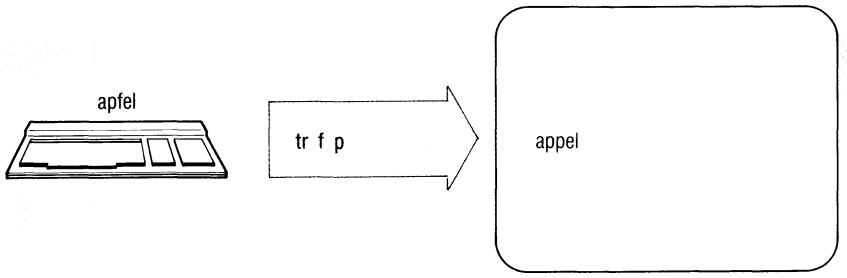
tput hc
Der Endestatus wird gesetzt, um anzuzeigen, ob die aktuelle Datensichtstation ein Hardcopy-Terminal ist.

## SIEHE AUCH

*stty*(1),
*terminfo*(4)

## NAME

**tr** - Zeichen durch andere ersetzen (translate characters)




## DEFINITION

**tr[␣schalter][␣zeichenfolge1[␣zeichenfolge2]]**

## BESCHREIBUNG

*tr* kopiert die Standard-Eingabe in die Standard-Ausgabe, wobei einzelne Zeichen ersetzt oder gelöscht werden. Eingabezeichen, die in *Zeichenfolge1* stehen, werden durch die korrespondierenden Zeichen in *Zeichenfolge2* ersetzt.

## SCHALTER

**-c**

Die Zeichen in *zeichenfolge1* werden bezüglich des erweiterten ASCII-Zeichensatzes (ASCII 001 bis 377 oktal) komplementiert; d.h. alle Zeichen aus diesem Zeichensatz, die nicht in *Zeichenfolge1* stehen, werden behandelt, als stünden sie in *Zeichenfolge1*.

**-d**

Alle Zeichen in *zeichenfolge1* werden gelöscht. *zeichenfolge2* wird ignoriert.

**-s**

Alle Folgen von gleichen Zeichen in *zeichenfolge2* werden bei der Ausgabe zu einem Zeichen.

## OPERANDEN

zeichenfolge1

Zeichen, die durch Zeichen aus *zeichenfolge2* ausgetauscht bzw. gelöscht werden sollen. Bereiche oder Folgen von gleichen Zeichen können Sie in Zeichenfolgen folgendermaßen verkürzt angeben:

[a-z]

Steht für die Zeichenfolge der ASCII-Zeichen von *a* bis *z* einschließlich.

[a*n]

Steht für *n* Wiederholungen von *a*. Ist die erste Ziffer von *n* 0, so wird *n* als Oktalzahl interpretiert, andernfalls als Dezimalzahl. Wenn *n* fehlt oder 0 ist, so wird von einer sehr großen Zahl ausgegangen; dies ist nützlich um *zeichenfolge2* aufzufüllen.

Das Entwertungszeichen \ kann verwendet werden, um die Sonderbedeutung des nachfolgenden Zeichens in einer Zeichenfolge aufzuheben. Folgen auf das Zeichen \ eine, zwei oder drei Oktalzahlen, so stehen sie für den ASCII-Code, der diesen Ziffern zugeordnet ist.

## BEISPIEL

1. Im folgenden Beispiel werden alle Worte in *datei1* nach *datei2* geschrieben, wobei jedes Wort in *datei2* in eine Zeile gesetzt wird; jedes Wort sei dabei eine alphabetische Zeichenfolge Die Zeichenfolgen werden apostrophiert, um zu verhindern, daß Sonderzeichen von der Shell interpretiert werden; 012 ist der ASCII-Code für das Neue-Zeile-Zeichen.

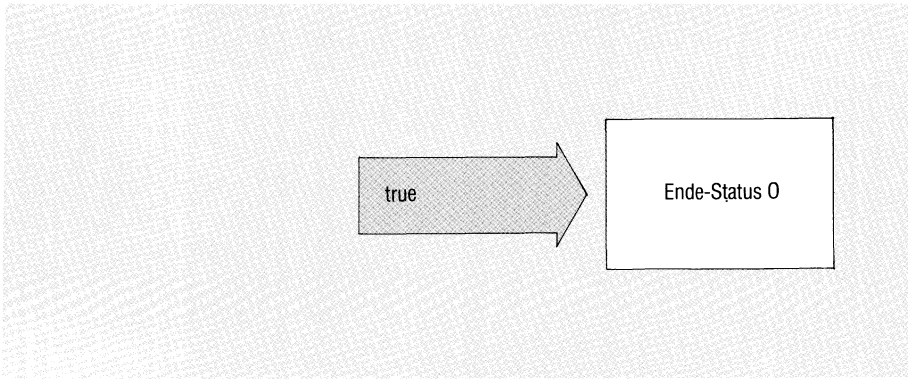
```
tr -cs "[A-Z][a-z]" "[\012*]" <datei1 >datei2
```

2. Alle Kleinbuchstaben aus *dat1* sollen als Großbuchstaben erscheinen:

```
$ cat dat1
Montag Dienstag Mittwoch Donnerstag Freitag Samstag Sonntag

$ tr "[a-z]" "[A-Z]" < dat1
MONTAG DIENSTAG MITTWOCH DONNERSTAG FREITAG SAMSTAG SONNTAG
```

## NAME

**true** - Leeres Kommando mit Endestatus 0




## DEFINITION

```
true
```

## BESCHREIBUNG

*true* hat nur die Funktion, den Endestatus 0 zu liefern. *true* verwendet man in Shell-Prozeduren, um die Bedingung *wahr* zu erzeugen. Die Bedingung *falsch* (Endestatus ungleich 0) können Sie mit dem Kommando *false*(1) erzeugen.

## ENDESTATUS

immer 0

## BEISPIEL

Mit dieser Prozedur wird eine Endlosschleife gestartet:

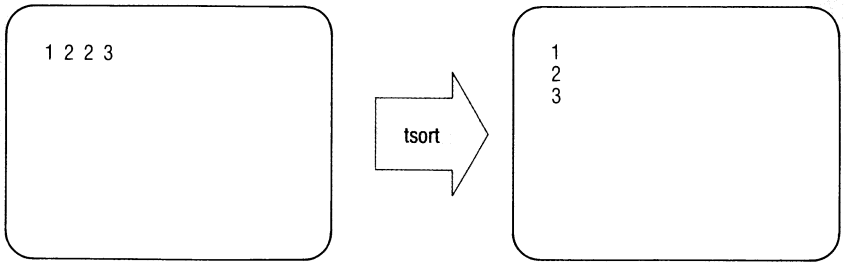
```
while true
do
        kommando
done
```

## SIEHE AUCH

*sh*(1)

## NAME

**tsort** - paarweise geordnete Objektmodule vollständig ordnen (topological sort)




## DEFINITION

**tsort**[␣datei]

## BESCHREIBUNG

*tsort* versucht, aus partiell geordneten Wortpaaren eine vollständig geordnete Liste zu erstellen. Die Wortpaare stehen in der Datei *datei*. Geben Sie keine Datei an, dann liest *tsort* von der Standard-Eingabe.

Die Wörter können beliebige nichtleere, druckbare ASCII-Zeichenreihen ohne Leer- oder Tabulatorzeichen sein. Sie sind durch ein oder mehrere Leerzeichen voneinander getrennt. Je zwei Wörter bilden ein Paar.

Sind die beiden Wörter eines Paares verschieden, so bedeutet das, daß *tsort* das erste Wort auch in der sortierten Liste vor dem zweiten Wort einordnen soll.

Sind in einem Wortpaar beide Wörter gleich, so fügt *tsort* das Wort lediglich in die Liste ein; es wird nicht festgelegt, an welcher Stelle es stehen soll.

*tsort* schreibt die sortierte Liste auf die Standard-Ausgabe.

Mit *lorder*(1D) und *tsort* können Sie Objektmodule innerhalb einer Bibliothek so ordnen, daß ein Objektmodul, das Funktionsaufrufe und Zuweisungen an externe Variablen enthält, vor den Objektmodulen steht, die die entsprechenden Funktions- und Variablendefinitionen enthalten. Der Binder *ld*(1D) kann eine so geordnete Bibliothek schneller durchsuchen. Siehe auch *ranlib*(1D)!

**HINWEIS**

Entdeckt *tsort* einen Zyklus (z.B. a b b a), dann gibt *tsort* eine Fehlermeldung aus.

**BEISPIEL**

In der Datei *datei* steht:

```
a b b c b d f b
```

*tsort* erstellt daraus die folgende geordnete Liste:

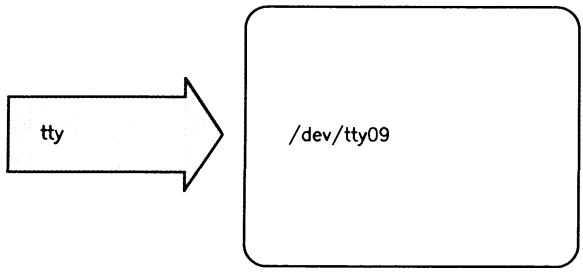
```
$ tsort datei
a
f
b
c
d
```

**SIEHE AUCH**

*ar*(1), *lorder*(1D), *ranlib*(1D).

## NAME

**tty** - Namen der Datensichtstation ausgeben




## DEFINITION

**tty[␣-l][␣-s]**

## BESCHREIBUNG

*tty* gibt den Pfadnamen Ihrer Datensichtstation aus.

## SCHALTER

**-s**

*tty* gibt den Namen Ihrer Datensichtstation nicht aus, sondern liefert nur den Endestatus zurück.

**-l**

*tty* gibt die Nummer der synchronen Datenleitung aus, an die die Datensichtstation angeschlossen ist.

## ENDESTATUS

0 Die Standard-Eingabe ist eine Datensichtstation
1 Die Standard-Eingabe ist keine Datensichtstation
2 Angegebene Schalter ungültig

**FEHLER**

Wenn die Standard-Eingabe keine Datensichtstation ist und der Schalter *-s* nicht gesetzt ist, meldet *tty* einen Fehler.

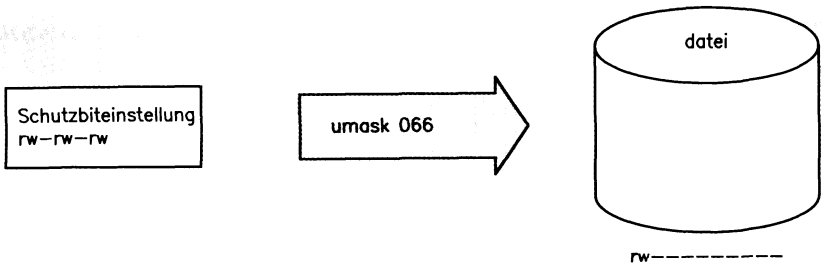
**BEISPIEL**

Ausgeben des Namens der Datensichtstation:

```
$ tty
/dev/tty10
$
```

## NAME

**umask** - Standardeinstellung der Schutzbits ändern




## DEFINITION

**umask**[␣nnn]

## BESCHREIBUNG

Mit *umask* können Sie die Standard-Schutzbiteinstellung für neu erstellte Dateien beeinflussen. *nnn* sind drei oktale Ziffern, von denen sich die erste auf die Zugriffsrechte des Eigentümers einer Datei, die zweite auf die Zugriffsrechte der Gruppe und die dritte auf die Zugriffsrechte der anderen bezieht (siehe *chmod*(1)).

Die Bits, die in der binären Schreibweise der von Ihnen angegebenen oktalen Ziffern *nnn* auf 1 gesetzt sind, werden für alle künftig neu erstellten Dateien auf 0 gesetzt.

Fehlt *nnn*, wird der derzeitige Wert von *umask* ausgegeben.

## HINWEIS

Dieses Kommando wird von der Shell direkt ausgeführt.

## BEISPIEL

Löschen der Schreibberechtigung für die Gruppe und für die Anderen:

```
$ umask 022
```

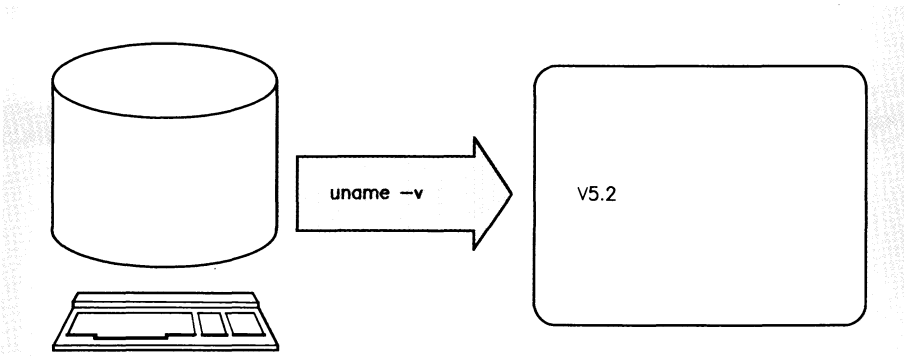
War die Schutzbiteinstellung vorher 777, ist sie jetzt 755, war sie vorher 666, ist sie jetzt 644.

## SIEHE AUCH

*chmod*(1), *umask*(2)

## NAME

**uname** - Namen des aktuellen Systems ausgeben




## DEFINITION

**uname**[␣schalter]

## BESCHREIBUNG

*uname* schreibt den Namen des aktuellen Systems auf die Standard-Ausgabe. Die Schalter können Sie setzen, wenn Sie nur bestimmte Informationen ausgegeben haben möchten:

## SCHALTER

**-s**

Der Name des Betriebssystems wird ausgegeben (Standard). Unter diesem Namen ist das Betriebssystem auf der lokalen Installation bekannt.

**-n**

Der Knotenname wird ausgegeben. Der Knotenname kann ein Name sein, unter dem das System innerhalb eines Netzes angesprochen werden kann.

**-r**

Der Stand des Betriebssystems wird ausgegeben.

**-v**

Die Version des Betriebssystems wird ausgegeben.

**-m**
Der Name des Maschinentyps wird ausgegeben, z.B. MX2.

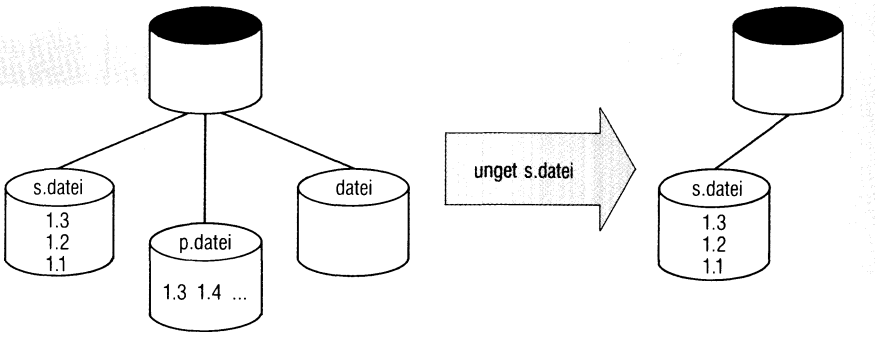
**-a**
Alle obengenannten Informationen werden ausgegeben.

## SIEHE AUCH

*uname*(2)

## NAME

**unget** - get-Kommando rückgängig machen (undo a previous get of an SCCS file)




## DEFINITION

**unget[␣-rSID][␣-s][␣-n]␣datei...**

## BESCHREIBUNG

*unget* macht Ihr letztes Kommando

```
$ get -e ... datei
```

ungeschehen. Sie wollen nicht, wie vorher beabsichtigt, ein neues Delta für die SCCS-Datei *datei* erstellen.

*unget* gibt auf der Standard-Ausgabe die SID-Nummer des Deltas aus, das nicht mehr erstellt werden soll.

## SCHALTER

Die Reihenfolge der Schalter ist beliebig.

**-r**SID

Mit *SID* legen Sie eindeutig fest, welches Delta nicht mehr erstellt werden soll. Nach

```
$ get -e ...
```

erschien diese SID-Nummer nach "new delta" auf der Standard-Ausgabe. Der Schalter ist nur nötig, wenn unter demselben login-Namen mehrere Versionen derselben SCCS-Datei zum Editieren geholt sind und entsprechende Einträge in der p-Datei stehen.

*unget* meldet einen Fehler, wenn

- der Schalter *-r* unbedingt nötig ist und fehlt,
- die angegebene Versionsnummer *SID* nicht eindeutig oder unkorrekt ist.

*Standard (keine Angabe):*
SID-Nummer des Deltas, das unter Ihrem login-Namen als letztes erstellt werden sollte.

**-s**

*unget* gibt auf der Standard-Ausgabe nicht die SID-Nummer des Deltas aus, das Sie nicht mehr erstellen wollen.

**-n**

*unget* löscht nicht die g-Datei, die *get*(1D) erstellt hat. Normalerweise löscht *unget* die Dateien im aktuellen Dateiverzeichnis, die *get* angelegt hat.

## OPERANDEN

datei

Als *datei* können Sie den Namen einer s-Datei, eines Dateiverzeichnisses oder das Zeichen - angeben. *unget* behandelt den Namen eines Dateiverzeichnisses, wie wenn Sie die Namen aller Dateien des Verzeichnisses einzeln aufführen würden. *unget* ignoriert Namen von Dateien, die keine s-Dateien sind oder für die Sie keine Leseberechtigung haben.

Wenn Sie statt *datei* das Zeichen - angeben, dann liest *unget* von der Standard-Eingabe. *unget* behandelt jede Eingabezeile als Name einer Datei oder eines Dateiverzeichnisses.

Sie können beliebig viele Namen angeben. Die gesetzten Schalter gelten dann für alle entsprechenden Dateien.

## DATEIEN

g-Datei

Siehe Schalter *-n*.

p-Datei

*unget* löscht den Eintrag in der p-Datei, der nicht mehr gelten soll. Wenn die p-Datei danach leer ist, wird sie gelöscht.

## HINWEIS

*unget* erwartet, daß im aktuellen Dateiverzeichnis die g-Datei zu der s-Datei steht, für die keine neue Version erstellt werden soll. Sie müssen daher *unget* aus demselben Dateiverzeichnis aufrufen, aus dem der entsprechende *get*-Aufruf erfolgte.

## BEISPIEL

Sie haben die Version 1.3 der Datei *s.ente* aus dem SCCS geholt und wollen die Version 2.1 einführen. Aus einem anderen Dateiverzeichnis holen Sie die Version 1.3, um die Version 1.4 zu errichten.
Wenn Sie jetzt eine der beiden neuen Versionen nicht erstellen wollen, müssen Sie die entsprechende Versionsnummer angeben.

```
$ admin -fj s.ente
$ get -e -r2 s.ente
1.3
new delta 2.1
12 lines
$ get -e s.ente
ERROR [s.ente]: writable 'ente' exists (ge4)
$ mkdir XXX
$ cd XXX
$ get -e ../s.ente
1.3
WARNING: being edited: '1.3 2.1 gudrun 85/06/27 14:08:26' (ge18)
new delta 1.4
12 lines
$ unget ../s.ente
ERROR [../s.ente]: SID must be specified (un1)
$ unget -r1.4 ../s.ente
1.4
$
```

Um auch noch den *get*-Aufruf im ersten Dateiverzeichnis rückgängig zu machen, brauchen Sie die SID-Nummer nicht anzugeben. *unget* soll die g-Datei nicht löschen.

```
$ cd ..
$ ls
XXX
ente
p.ente
s.ente
$ unget -n s.ente
2.1
$ ls
XXX
ente
s.ente
$
```

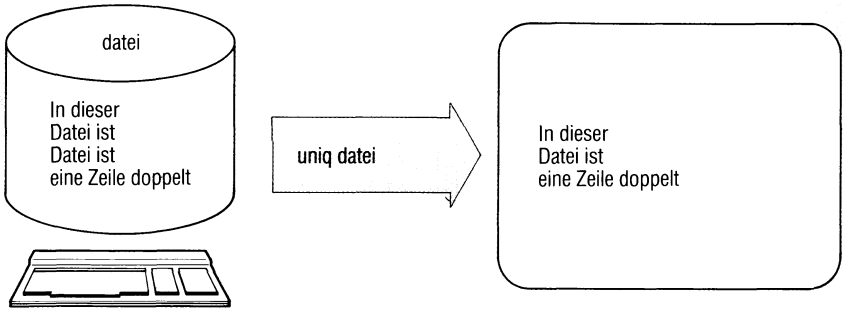
## SIEHE AUCH

*admin*(1D), *delta*(1D), *get*(1D), *rmdel*(1D), *sact*(1D).

Eine Einführung ins SCCS ("Source Code Control System") finden Sie im Buch SINIX Einführung [1]. Dort sind auch alle Begriffe erklärt, die das SCCS betreffen.

## NAME

**uniq** - Mehrfache Zeilen suchen (unique lines)




## DEFINITION

**uniq**[␣-schalter[␣+n][␣-n]][␣eingabe[␣ausgabe]]

## BESCHREIBUNG

*uniq* sucht in der Datei *eingabe* nach aufeinanderfolgenden gleichen Zeilen, schreibt die Datei in die Datei *ausgabe* und läßt dabei die Wiederholungen weg.

! Nur bei aufeinanderfolgenden Zeilen können Übereinstimmungen festgestellt werden.

## SCHALTER

**-u**

Nur die Zeilen werden ausgegeben, die in *eingabe* nicht wiederholt vorkommen.

**-d**

Nur eine Kopie der Zeilen wird ausgegeben, die wiederholt vorkommen.

**-c**

Eine Statistik über die Häufigkeit der mehrfach vorkommenden Zeilen wird ausgegeben.

**OPERANDEN**

eingabe
Name der Datei, die untersucht werden soll. Standard (keine Angabe): *uniq* liest von Standard-Eingabe.

ausgabe
Name der Datei, in die die Ausgabe geschrieben werden soll. Standard (keine Angabe): *uniq* schreibt auf Standard-Ausgabe.

-n
Die ersten *n* Felder ab Zeilenanfang bzw. ab Feld n+1 werden ignoriert. Ein Feld ist als eine Folge von Zeichen definiert, die durch Tabulator- und Leerzeichen voneinander getrennt sind.

+n
Die ersten *n* Zeichen werden ignoriert.

**BEISPIEL**

Durchsuchen einer Datei nach gleichen Zeilen unabhängig davon, wo sie in der Datei stehen. Für jede dieser Zeilen ist auszugeben, wie oft sie vorkommt.

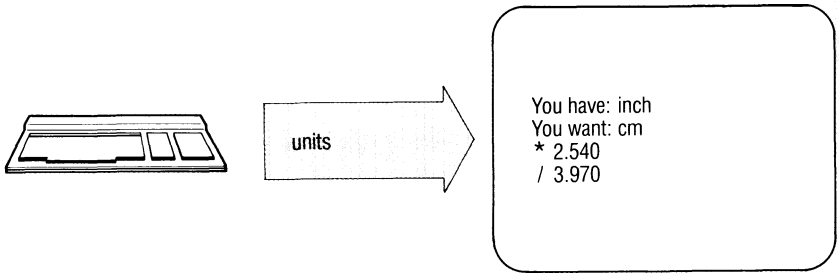
```
$ sort datei | uniq -c
```

**SIEHE AUCH**

*comm*(1), *sort*(1)

## NAME

**units** - Einheiten umrechnen




## DEFINITION

**units**

## BESCHREIBUNG

*units* wandelt in einer Einheit angegebene Mengen in andere Einheiten um. Bei der Umwandlung werden folgende Fragen ausgegeben:

```
You have: inch
You want: cm
        * 2.540000e+00
        / 3.937008e-01
```

Eine Menge ist das Produkt von zwei oder mehreren Einheiten, denen ein numerischer Multiplikator vorangestellt sein kann. Exponenten werden durch nachgestellte, positive ganze Zahlen gekennzeichet; eine Division kann durch einen Schrägstrich angegeben werden:

```
You have: 15 lbs force/in2
You want: atm
        * 1.020689e+00
        / 9.797299e-01
```

*units* führt nur multiplikative Umrechnungen durch; es kann also Kelvin nach Rankine, nicht aber Celsius nach Fahrenheit umrechnen. Zulässig sind die meisten der gebräuchlichen Einheiten, Abkürzungen und metrischen Präfixe; aber auch die folgenden weniger bekannten Einheiten und Naturkonstanten dürfen verwendet werden:

| | |
|---|---|
| **pi** | Verhältnis von Umfang zu Durchmesser |
| **c** | Lichtgeschwindigkeit |
| **e** | Ladung eines Elektrons |
| **g** | Erdbeschleunigung |
| **force** | wie *g* |
| **mole** | Mol |
| **water** | Druck pro Wassersäulen-Einheit |
| **au** | (astronomical unit) |

*pound* ist im Gegensatz zu *lb* nicht zulässig; zusammengesetzte Namen werden zusammen bearbeitet (z.B. *lightyear*). Britische Einheiten, die sich von ihrem amerikanischen Äquivalent unterscheiden, werden folgendermaßen mit einem Präfix versehen: *brgallon*. Eine vollständige Liste der zulässigen Einheiten erhält man folgendermaßen:

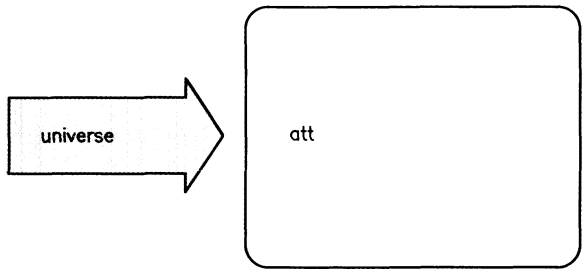
```
cat /usr/lib/unittab
```

## DATEIEN

*/usr/lib/unittab*

## NAME

**universe** - das aktuelle Universum ausgeben




## DEFINITION

**universe**

## BESCHREIBUNG

*universe* gibt das aktuelle Universum aus.

## BEISPIEL

Die Eingabeaufforderung der Shell (Standard $) soll so umdefiniert werden, daß sie das jeweils aktuelle Universum anzeigt. Definieren Sie hierfür in der Datei $HOME/.profile, die bei jedem Aufruf der Login-Shell ausgeführt wird, die Variable PS1 wie folgt:

```
PS1="<`universe`>$ "
export PS1
```

Falls das *att*-Universum Ihr aktuelles Universum ist, gibt die Shell jetzt als Eingabeaufforderung aus:

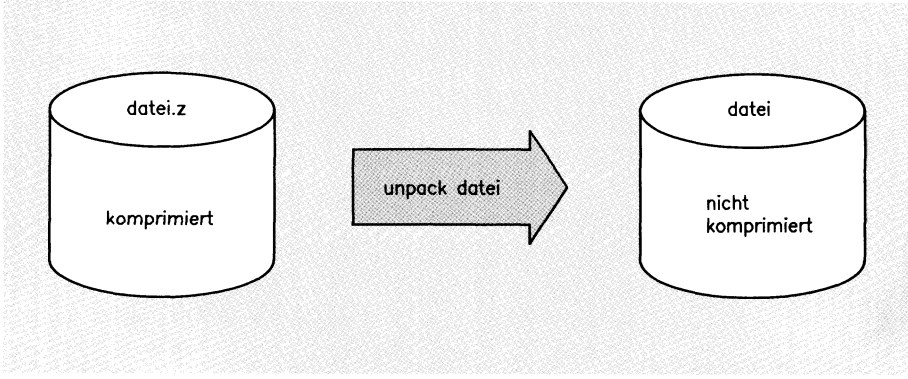
```
<att>$
```

## SIEHE AUCH

*att*(1), *environ*(5), *login*(1), *sie*(1)

## NAME

**unpack** - Komprimierte Dateien expandieren




## DEFINITION

**unpack** name...

## BESCHREIBUNG

*unpack* stellt von Dateien, die mit *pack* komprimiert worden sind, den Originalzustand her. Für jede datei *name* sucht *unpack* nach einer Datei namens *name.z*. Ist diese Datei offensichtlich eine komprimierte Datei, so wird sie durch die entkomprimierte Version ersetzt. Der Name der neuen Datei hat nicht mehr die Endung *.z*; außerdem hat sie dieselben Zugriffsrechte, dasselbe Zugriffs- und Änderungsdatum und denselben Eigentümer wie die komprimierte Datei.

Der Ende-Status von *unpack* gibt an, von wievielen Dateien der Originalzustand nicht wiederhergestellt werden konnte. Außer den unter *pcat* genannten Gründen kann dies folgende Ursachen haben:

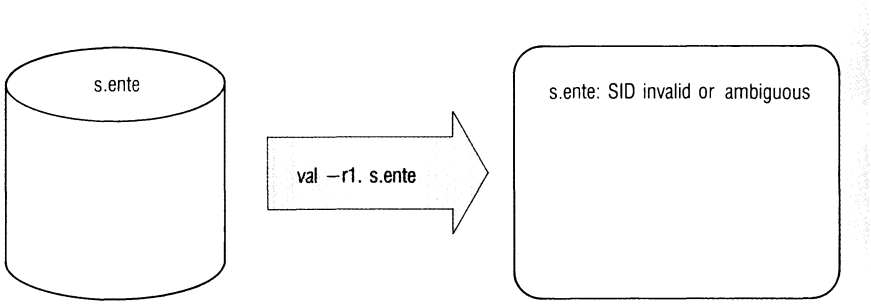
- es existiert bereits eine Datei namens *name* oder
- die betreffende Datei kann nicht erstellt werden.

## SIEHE AUCH

*pack*(1), *cat*(1)

## NAME

**val** - SCCS-Dateien auf Konsistenz prüfen (validate SCCS file)




## DEFINITION

```
val[␣schalter...]␣datei...      oder
val␣-
```

## BESCHREIBUNG

*val* überprüft zunächst, ob *datei* eine SCCS-Datei ist; je nachdem, welche Schalter eingeschaltet sind, überprüft *val* zusätzlich bestimmte Eigenschaften.
*val* betrachtet alle Eigenschaften, die nicht zutreffen, als Fehler. Die Fehler, die bei den angegebenen Dateien auftreten, schreibt *val* auf die Standard-Ausgabe.
Zusätzlich liefert *val* einen 8 Bit langen Ende-Status-Wert, den Sie mit dem Kommando *echo $?* abfragen können.

Wenn Sie *val* mit dem Zeichen - aufrufen, dann erwartet *val* die Angabe der Schalter und Dateinamen auf der Standard-Eingabe. *val* betrachtet jede Zeile als Argumentenliste. Jede Zeile muß folgendes Format haben:

`[schalter...]␣datei...`

Wie üblich schicken Sie eine Zeile mit [↵] ab. Wenn Sie die Eingabe an *val* beenden wollen, drücken Sie [END].

## SCHALTER

Die Reihenfolge der Schalter ist beliebig.

**-r**SID

*SID* sollte eine Deltanummer sein. *val* prüft, ob *SID*

- mehrdeutig oder
- ungültig ist.

Wenn beides nicht zutrifft, prüft *val*, ob *SID*

- existiert.

*Beispiel*

| | |
|---|---|
| -r1 | gibt nicht klar an, welche Version (z.B. 1.1, 1.2 oder 1.3) gemeint ist. |
| -r1.1.0.0 | ist eine ungültige Deltanummer. |

**-s**

*val* gibt auf der Standard-Ausgabe keine Fehlermeldungen aus.

**-m**mod

*mod* ist eine ASCII-Zeichenreihe. *val* vergleicht *mod* mit dem Modulnamen (Identifikations-Schlüsselwort %*M*%) in *datei* (siehe *admin*(1D), Schalter *-fm*).

**-y**typ

*typ* ist eine ASCII-Zeichenreihe. *val* vergleicht *typ* mit dem Modultyp (Identifikations-Schlüsselwort %*Y*%) in *datei* (siehe *admin*(1D), Schalter *-ft*).

## OPERANDEN

datei

*datei* ist der Name einer s-Datei.
Sie können auch mehrere s-Dateien angeben. Die gesetzten Schalter gelten dann für alle angegebenen Dateien.

## ENDESTATUS

*val* verwaltet ein 8-Bit-Wort, in dem alle Bits mit 0 vorbesetzt sind. Jedes Bit repräsentiert einen möglichen Fehler. *val* setzt ein bestimmtes Bit auf 1, wenn der entsprechende Fehler auftritt. Tritt der gleiche Fehler nochmals auf (bei Angabe mehrerer Dateien bzw. bei *val* -), dann bleibt das Bit auf 1 gesetzt. Jedem Bit, das auf 1 gesetzt ist,

entspricht ein Zahlenwert. Der Ende-Status ist die Summe der Zahlenwerte aller Bits, die auf 1 gesetzt sind. Der Ende-Status ist 0, wenn *val* bei keiner Datei, deren Namen Sie angeben, und in keiner Eingabezeile (bei *val* -) einen Fehler gefunden hat.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Zahlenwert | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

| Bit | Fehler, bei dem das Bit auf 1 gesetzt wird | Fehlerausgabe von val |
|---|---|---|
| 7 | *datei* fehlt | : missing file argument |
| 6 | Schalterangabe unkorrekt | : unknown or duplicate keyletter argument |
| 5 | SCCS-Datei nicht SCCS-gemäß verändert | datei: corrupted SCCS file |
| 4 | *datei* kann nicht geöffnet werden oder ist keine SCCS-Datei | datei: can't open file or file not SCCS |
| 3 | SID ungültig oder mehrdeutig | datei: SID invalid or ambiguous |
| 2 | SID existiert nicht | datei: SID nonexistent |
| 1 | %Y% und *typ* verschieden | datei: %Y%, -y mismatch |
| 0 | %M% und *mod* verschieden | datei: %M%, -m mismatch |

**BEISPIEL**

Die Datei *s.ente* soll den Modulnamen *entlein* und den Modultyp *text* bekommen.

```
$ admin -fmentlein -fttext s.ente
$ val -ytext -mentlein -r1.2 s.ente
```

*val* gibt keine Fehlermeldung aus. *s.ente* hat den Modulnamen *entlein*, ist vom Typ *text*,und außerdem existiert die Version 1.2 von *s.ente*.

```
$ val -r1. -ytext s.ente
    s.ente: SID invalid or ambiguous
$ echo $?
8
$
```

Versionsnummer 1. ist nicht korrekt. Der Ende-Status ist 8.

Sie können die obigen Kommandos auch so an *val* übergeben:

```
$ val -
-ytext -mentlein s.ente
-r1. -ytext s.ente
    s.ente: SID invalid or ambiguous
END
$ echo $?
8
$
```

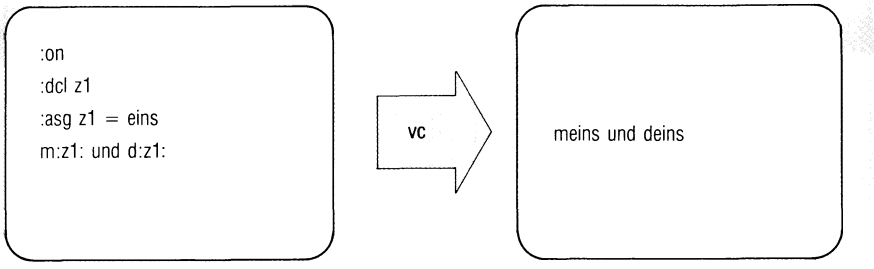
## SIEHE AUCH

*admin*(1D), *delta*(1D), *get*(1D), *prs*(1D).

Eine Einführung ins SCCS ("Source Code Control System") finden Sie im Buch SINIX Einführung [1]. Dort sind auch alle Begriffe erklärt, die das SCCS betreffen.

## NAME

**vc** - Textdarstellung kontrollieren (version control)




## DEFINITION

**vc[␣schalter...][␣schlüsselwort = wert...]**

## BESCHREIBUNG

*vc* kopiert ASCII-Text zeilenweise von der Standard-Eingabe auf die Standard-Ausgabe. Ob und wie kopiert wird, können Sie festlegen durch:

- Schalter
- Kontrollanweisungen, die Sie im Eingabetext einfügen und die zwischen Textzeilen stehen, die *vc* kopieren soll,
- Wertzuweisungen an Schlüsselwörter.

Jede Zeile der Eingabe enthält entweder einen Text, der kopiert werden soll, oder eine Kontrollanweisung.

Schlüsselwörter, die im Text und/oder den Kontrollanweisungen vorkommen, können durch die entsprechenden Werte ersetzt werden.

*vc* ersetzt die Schlüsselwörter immer durch ihren Wert, wenn

- sie in einer Kontrollanweisung stehen und(!)
- ein Kontrollzeichen davor und danach geschrieben ist.

In Textzeilen ersetzt *vc* die Schlüsselwörter nur dann durch ihren Wert, wenn

- die Textzeile irgendwo nach der Kontrollanweisung *:on* steht und ein Kontrollzeichen vor und nach dem Schlüsselwort steht oder
- der Schalter *-a* eingeschaltet ist und ein Kontrollzeichen vor und nach dem Schlüsselwort steht.

Kontrollzeichen ist standardmäßig : (Doppelpunkt). Mit dem Schalter *-c* können Sie ein anderes Zeichen als Kontrollzeichen wählen.

*vc* betrachtet eine Zeile (vorausgesetzt, : ist Kontrollzeichen):

- als Kontrollanweisung, wenn die Zeile mit dem Zeichen : beginnt (siehe auch Schalter *-t*). *vc* kopiert die Zeile nicht.
  (*Ausnahme*: Kontrollanweisung *::text*)
- als Textzeile, wenn sie mit einem beliebigen ASCII-Zeichen außer : beginnt.

Wenn *vc* ein Kontrollzeichen nicht als solches interpretieren soll, dann müssen Sie ihm den Gegenschrägstrich \ voranstellen.
Beginnt eine Zeile mit \: (und ist : Kontrollzeichen), dann interpretiert *vc* die Zeile als Textzeile. *vc* kopiert die Zeile ohne den ersten Gegenschrägstrich \.
Beginnt eine Zeile mit \ und ist das zweite Zeichen kein Kontrollzeichen, dann kopiert *vc* die Zeile vollständig.

**Kontrollanweisungen**

Kontrollanweisungen sind Eingabezeilen, die mit dem Kontrollzeichen : beginnen (siehe auch Schalter *-t* und *-c*). Es gibt folgende Kontrollanweisungen:

**:dcl**␣schlüsselwort[,schlüsselwort...]
Deklaration der Schlüsselwörter.
Alle Schlüsselwörter, die in der Eingabe vorkommen, müssen deklariert werden. Schlüsselwörter, denen Sie keinen Wert zuweisen, haben als Wert die leere Zeichenreihe.

**:asg**␣schlüsselwort = wert
*vc* weist dem Schlüsselwort (ähnlich wie in der Kommandozeile) einen Wert zu. Diese Kontrollanweisung überschreibt alle Wertzuweisungen an ein Schlüsselwort, die in der Kommandozeile oder in früheren Kontrollanweisungen vorkommen. Schlüsselwörter, denen Sie keinen Wert zuweisen, haben als Wert die leere Zeichenreihe.

**:if**␣bedingung
.
.
**:end**
Wenn die Bedingung *bedingung* wahr ist, kopiert *vc* alle Textzeilen, die zwischen der *if*-Anweisung und der *end*-Anweisung stehen. Kontrollanweisungen, die in den Zwischenzeilen vorkommen, führt *vc* aus.

Wenn *bedingung* falsch ist, kopiert *vc* keine Zeile zwischen den beiden Anweisungen. Kontrollanweisungen, die in den Zwischenzeilen vorkommen, werden ignoriert und nicht ausgeführt.

*bedingung* hat die folgende Syntax:

```
bedingung  ist    [not] A

A          ist    B
           oder   B | A          /* log. ODER */

B          ist    ausdruck
           oder   ausdruck & B   /* log. UND */

ausdruck   ist    ( A )
           oder   wert op wert

op         ist    =              /* gleich */
           oder   !=             /* ungleich */
           oder   <              /* kleiner als */
           oder   >              /* größer als */

wert       ist    ASCII-Zeichenreihe
           oder   numerische Zeichenreihe
```

Links stehen die Namen der Elemente, aus denen sich *bedingung* zusammensetzen kann. Rechts steht, welche Struktur die Elemente haben können.
*wert* kann z.B. entweder eine ASCII-Zeichenreihe oder eine numerische Zeichenreihe sein.
*bedingung* kann entweder *not A* oder *A* sein.

*Bedeutung der Operatoren:*

Die Klammern ( und ) dienen dazu, um den Vorrang von Operatoren zu ändern oder logische Ausdrücke zusammenzufassen.

*not* darf nur direkt nach *:if* stehen (mit Leerzeichen dazwischen). *not* kehrt den Wahrheits-Wert der gesamten restlichen Bedingung um.

Die Vergleichsoperatoren < und > vergleichen nur ganze Zahlen ohne Vorzeichen. Alle anderen Operatoren behandeln die Werte als Zeichenreihen.

Die Operatoren =, !=, > und < haben gleichen Vorrang.
& hat niedrigeren Vorrang, und der Operator | hat den niedrigsten Vorrang.

Mindestens ein Leer- oder Tabulatorzeichen muß die Werte von den Operatoren oder Klammern trennen.

*Beispiel*

| *bedingung* | Wert von *bedingung* |
|---|---|
| not 777 < 766 | wahr |
| not ( 777 > 766 ) | falsch |
| 012 > 12 | falsch |
| 012 != 12 | wahr |
| ( hans = uschi ) \| ( a = a ) | wahr |
| hans = uschi & a = a | falsch |

**::text**
*text* ist ein beliebiger ASCII-Text, den *vc* auf die Standard-Ausgabe kopiert. Die beiden Doppelpunkte am Anfang der Zeile werden nicht kopiert. Schlüsselwörter, die im Text vorkommen und zwischen Doppelpunkten stehen, ersetzt *vc* durch ihre Werte. Dabei ist es egal, ob der Schalter *-a* eingeschaltet ist oder nicht.

**:on**
In den folgenden Textzeilen ersetzt *vc* die Schlüsselwörter, die zwischen Kontrollzeichen stehen, durch ihre Werte.

**:off**
In den folgenden Textzeilen ersetzt *vc* die Schlüsselwörter nicht durch ihre Werte.

**:ctl␣**char
*char* ist bis auf Widerruf das Kontrollzeichen.

**:msg␣**meldung
*vc* schreibt *meldung*, einen ASCII-Text, auf die Standard-Fehlerausgabe.

**:err␣**meldung
*vc* schreibt *meldung*, einen ASCII-Text, auf die Standard-Fehlerausgabe. Danach folgt eine Fehlermeldung von *vc* und die Angabe der Zeile, in der der Fehler aufgetreten ist. *vc* beendet seine Ausführung und hat den Ende-Status 1.

## SCHALTER

Die Reihenfolge der Schalter ist beliebig.

**-a**
*vc* ersetzt auch in Textzeilen alle Schlüsselwörter, die zwischen Kontrollzeichen stehen. Der kopierte Text enthält dann statt der Schlüsselwörter die entsprechenden Werte.

**-t**
*vc* ignoriert alle Zeichen einer Zeile vom Zeilenanfang an bis einschließlich dem ersten Tabulatorzeichen der Zeile, wenn der Rest der Zeile eine Kontrollanweisung darstellt, und führt die Kontrollanweisung aus. *vc* kopiert die gesamte Zeile, wenn der Rest der Zeile keine Kontrollanweisung ist.

**-c**char
Sie wollen *char* statt : als Kontrollzeichen verwenden.

**-s**
*vc* unterdrückt Warnungen, die normalerweise auf der Standard-Fehlerausgabe ausgegeben werden. Fehlermeldungen gibt *vc* weiterhin aus.

## OPERANDEN

schlüsselwort

*schlüsselwort* ist eine alphanumerische ASCII-Zeichenreihe, die aus maximal 9 Zeichen bestehen darf. Das erste Zeichen muß ein Buchstabe sein.

wert

*wert* ist eine beliebige Zeichenreihe, die keine Leer- und Tabulatorzeichen enthalten darf und die *ed* erstellen kann. Ein numerischer Wert ist eine Folge von Ziffern ohne Vorzeichen. Soll *wert* das Kontrollzeichen enthalten, so müssen Sie dem Kontrollzeichen das Fluchtsymbol \ voranstellen.

## ENDESTATUS

0 wenn *vc* normal beendet wird,
1 wenn während der Durchführung von *vc* ein Fehler auftritt.

## BEISPIEL

In der Datei *allerlei* steht folgender Eingabetext mit Kontrollanweisungen:

```
:on

:dcl z1,z2,z3
:asg z1=1
:asg z2=2
:asg z3=3

:if ( :z1: > :z2: )
Regenwetter ist herrlich!
:end

\:z1: + :z1: =??
:::z1:, :z2:, :z3:, fertig ist die Zauberei!
:ctl %
%asg z1=eins
%asg z2=ein
m:z1:, d:z1: und gar k:z1:.
m%z1%, d%z1% und gar k%z1%.

%if ( 5 != 4 )
Du hast k%z2%e Chance.
Nutze sie!
%end
```

```
%if  hans = uschi  &  a = a
Am Anfang war das Wort.
%end

%if ( hans = uschi ) | ( a = a )
Aller Anfang ist schwer!
%end

%off

%err Schon wieder ein Fehler!
```

Nach dem Aufruf

```
$ vc < allerlei
```

folgt die Ausgabe von *vc*:

```
1 + 1 =??
1, 2, 3, fertig ist die Zauberei!
m:z1:, d:z1: und gar k:z1:.
meins, deins und gar keins.

Du hast keine Chance.
Nutze sie!



Aller Anfang ist schwer!



ERROR: Schon wieder ein Fehler!
ERROR: err statement on line 35 (vc15)
$
```

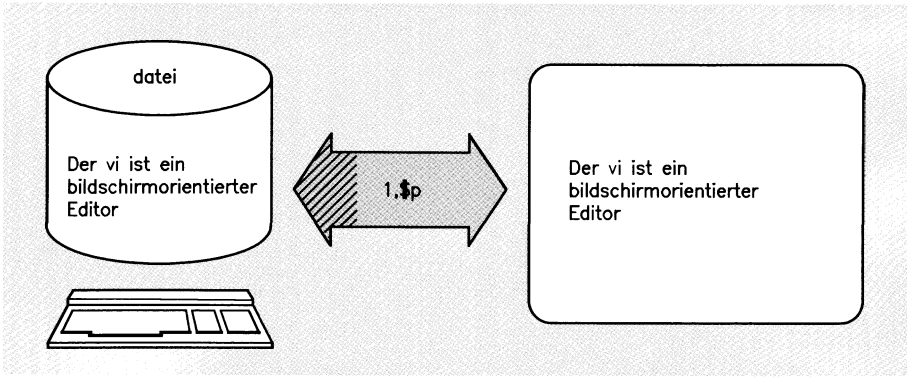
*vc* bricht mit Ende-Status 1 ab.

## SIEHE AUCH

*ed*(1).

## NAME

**vi** - Bildschirmorientierter Editor




## DEFINITION

**vi[␣-t␣tag][␣-r][␣-wn][␣-R][␣+kommando][␣datei...]**

## BESCHREIBUNG

*vi* (visual) ist ein bildschirmorientierter Texteditor. Er stellt die erweiterte Form des Zeileneditors *ex*(1) dar; es ist möglich, zwischen diesen beiden Editoren hin- und herzuschalten und von *vi* aus *ex* Kommandos auszuführen. Eine ausführliche Einführung in den *vi* mit Beispielsitzung finden Sie im Manual SINIX Einführung [1].

Beim Arbeiten mit *vi* ist der Bildschirm ein Fenster in die editierte Datei. Werden an einer Datei Änderungen vorgenommen, so wird die Bildschirmanzeige entsprechend verändert. Wird die Schreibmarke auf dem Bildschirm an eine bestimmte Stelle bewegt, so können an der entsprechenden Stelle in der Datei Änderungen vorgenommen werden.

In der Umgebungsvariablen TERM muß der Typ der benutzten Datensichtstation festgelegt sein; die Art des Terminals muß in einer geeigneten Datenbank beschrieben sein. Wie es auch bei *ex* der Fall ist, können Skripts zur Initialisierung des Editors in der Umgebungsvariablen EXINIT oder der Datei *.exrc* im aktuellen bzw. HOME-Dateiverzeichnis enthalten sein.

Abbruch mit ESC:Q



## SCHALTER

**-t** tag
Die Datei, die das *tag* enthält, wird editiert; Die Schreibmarke wird auf die Definition des *tag* positioniert.

**-r**
*datei* wird nach einem Absturz des Editors oder des Systems wiederhergestellt; fehlt *datei*, so wird eine Liste aller geretteten Dateien ausgegeben.

**-w**n
Die Bildschirmanzeige soll standardmäßig aus *n* Zeilen bestehen.

**-R**
*Readonly*-Modus; hiermit wird ein versehentliches Überschreiben der Datei verhindert.

**+**kommando
Das angegebene *ex-Kommando* wird vor dem Beginn des Editierens ausgeführt.

**-l**
lisp Modus wird gesetzt (siehe *ex*(1))

## KOMMANDOS

### Allgemeines

Eine vollständige Beschreibung des *ex*-Editors ist im Eintrag *ex*(1) enthalten. Im folgenden wird ausschließlich der *Bildschirm*-Modus des Editors beschrieben.

Nach dem Aufrufen befindet sich *vi* im *Kommando*-Modus; Wird ein Kommando zum Einfügen oder Ändern von Text aufgerufen, so wird in den *Eingabe*- Modus umgeschaltet.
Durch Eingabe des ESC-Zeichens wird der Eingabe-Modus verlassen. Im Kommando-Modus dagegen wird mit diesem Zeichen ein teilweise eingegebenes Kommando gelöscht. Wenn weder der Editor sich im Eingabe-Modus befindet, noch ein Kommando teilweise eingegeben wurde, so wird mit ESC ein akustisches Signal abgegeben.

In der letzten (untersten) Bildschirmzeile werden Suchkommandos (/ und ?), *ex*-Kommandos (:) sowie Systemkommandos (!) angezeigt. Außerdem werden in dieser Zeile Fehler- und andere Meldungen ausgegeben.

Wurde während der Texteingabe oder während der Eingabe eines Kommandos in der untersten Bildschirmzeile das Signal **SIGINT** ([DEL]) empfangen, so wird die Eingabe beendet (bzw. das Kommando gelöscht) und der Editor wieder in den Kommando-Modus geschaltet. Der Empfang eines **SIGINT** im Kommando-Modus bewirkt die Abgabe eines akustischen Signales, mit dem im allgemeinen ein Fehler gemeldet wird (z.B. die Betätigung einer unzulässigen Taste).

Alle Zeilen einer Datei, in denen nur das Zeichen ~ steht, existieren nicht und können also nicht angesprochen werden.
Auf einer Datensichtstation mit begrenzter lokaler Intelligenz können auf dem Bildschirm Leerzeilen vorkommen, die mit einem @ markiert sind; für diese Leerzeilen gibt es in der Datei keine Entsprechungen (sie können durch die Eingabe eines [CTRL] R gelöscht werden; der Editor baut den Bildschirm dann ohne diese Lücken neu auf).

**Kommandos**

Den meisten Kommandos kann als Argument eine Zahl vorangestellt werden; die Zahl gibt dann entweder eine Größe oder (bei Anzeige- oder Positionierungs-Kommandos) eine Position an, oder sie kann (bei Kommandos zur Änderung von Text) als Wiederholungsfaktor verwendet werden. Der Einfachheit halber wird dieses optionale Argument bei allen Kommandos als *zahl* bezeichnet.

Auf die folgenden Operatoren kann ein Positionierungskommando folgen, um den zu bearbeitenden Bereich anzugeben: *c*, *d*, *y*, <, >, ! und =. Im angegebenen Bereich sind, beginnend bei der aktuellen Position, alle Zeilen bis zur (jedoch nicht durch einschließlich der) durch das Kommando angegebenen Position enthalten. Werden durch ein Kommando nur Zeilen bearbeitet, so sind alle Zeilen betroffen, die vollständig oder teilweise in diesem Bereich enthalten sind. Andernfalls wird genau der angegebene Bereich behandelt.

Im folgenden werden Steuerzeichen im Format ^X angegeben; diese Angabe ist gleichbedeutend mit [CTRL] [X].

Sofern nicht anders angegeben, werden die Kommandos im Kommando-Modus interpretiert und haben im Eingabe-Modus keine spezielle Bedeutung.

**^B**

Die Bildschirmanzeige wird in Rückwärtsrichtung gerollt; es wird also die vorhergehende Bildschirmseite angezeigt. Mit einer Zahl kann angegeben werden, um wieviele Bildschirmseiten zurückgeblättert werden soll. Falls möglich, werden auf die neue Bildschirmseite zwei Zeilen der vorhergehenden Bildschirmseite übernommen.

**^D**

Die Bildschirmanzeige wird um ein halbes Fenster in Vorwärtsrichtung gerollt. Mit einer Zahl kann angegeben werden, um wieviele (logische) Zeilen die Bildschirmanzeige gerollt werden soll; sie wird für spätere Aufrufe der Kommandos *^D* und *^U* gespeicheret.

Im Eingabemodus wird die Schreibmarke entsprechend *autoindent* oder *^T* um *shiftwidth* Positionen nach rechts bewegt.

**^E**

Die Bildschirmanzeige wird um eine Zeile vorwärts gerollt; falls möglich, bleibt die Position des Cursors dabei unverändert.

**^F**

Die Bildschirmanzeige wird um eine Seite in Vorwärtsrichtung gerollt; mit einer Zahl kann angegeben werden, um wieviele Seiten die Bildschirmanzeige gerollt werden soll. Falls möglich, werden auf die neue Bildschirmseite zwei Zeilen der jeweils vorhergehenden übernommen.

**^G**
**^g**

Der Name der aktuellen Datei und andere Informationen wie die Zahl der Zeilen und die aktuellen Position werden ausgegeben (entspricht dem *ex* Kommando *f*.)

**^H**

Die Schreibmarke wird um eine Position nach links bewegt (höchstens bis zum linken Rand); über eine Zahl kann angegeben werden, um wieviele Positionen die Schreibmarke nach links bewegt werden soll (wie bei *h*.)

Im Eingabemodus wird die Schreibmarke um eine Position nach links bewegt; das Zeichen an dieser Position wird nicht gelöscht.

**^J**

Die Schreibmarke wird innerhalb der aktuellen Spalte in die nächste Zeile bewegt; soll die Schreibmarke eine oder mehrere Zeilen überspringen, so kann ein Wiederholungsfaktor angegeben werden (wie *^N* und *j*.)

**^L**
**^l**

Die Bildschirmanzeige wird gelöscht und dann wieder aufgebaut. Dieses Kommando wird verwendet, wenn die Bildschirmanzeige aus irgendeinem Grund fehlerhaft ist.

**^M**

Die Schreibmarke wird zum zum ersten Zeichen in der nächsten Zeile bewegt, bei dem es sich nicht um ein Leer- oder Tabulatorzeichen handelt. Mit einer Zahl kann ein Wiederholungsfaktor angegeben werden.

**^N**

Entspricht *^J* und *j*.

**^P**

Die Schreibmarke wird innerhalb der aktuellen Spalte in die darüberliegende Zeile bewegt. Über eine Zahl kann ein Wiederholungsfaktor angegeben werden (wie *k*).

**^R**

Die aktuelle Bildschirmanzeige wird neu aufgebaut; die mit @ gekennzeichneten, "falschen" Zeilen (d.h. Zeilen, die in der in der Datei keine Entsprechung haben), werden gelöscht.

**^T**

Wenn die Schreibmarke am Anfang der Zeile steht oder ihm nur Leeroder Tabulatorzeichen vorangestellt sind, werden *shiftwidth* Leer- oder Tabulatorzeichen eingefügt. Soll die Schreibmarke diese Leer- oder Tabulatorzeichen nach vorne überspringen, so muß das Kommando *KD* verwendet werden.

**^U**

Die Bildschirmanzeige wird um eine halbe Seite nach oben, d.h. Richtung Dateianfang gerollt. Mit einer Zahl kann angegeben werden, um wieviele (logische) Zeilen die Anzeige gerollt werden kann; diese Zahl wird für spätere Aufrufe der Kommandos *^D* und *^U* gespeichert.

**^V**

Im Eingabemodus wird das nächste Zeichen entwertet, damit in die Datei Sonderzeichen (auch ESC-Zeichen) eingefügt werden können.

**^W**

Im Eingabemodus wird die Schreibmarke zum vorherigen Wort bewegt; es werden keine Zeichen gelöscht.

**^Y**

Die Bildschirmanzeige wird um eine Zeile nach oben (Richtung Dateianfang) gerollt; falls möglich, bleibt die Position des Cursors unverändert.

**^[**

Ein teilweise eingegebenes Kommando wird gelöscht; wurde kein Kommando teilweise eingegeben, so wird ein akustisches Signal ausgegeben.

Befindet sich *vi* im Eingabemodus, so wird in den Kommandomodus umgeschaltet.

Wird in der untersten Bildschirmzeile ein Kommando eingegeben (*ex*-Kommando oder Suchmuster mit oder *?*), so wird die Eingabe abgeschlossen und das Kommando ausgeführt.

**Leerzeichen**

Die Schreibmarke wird um eine Position nach rechts bewegt (höchstens bis zum Zeilenende). Über eine Zahl kann ein Wiederholungsfaktor angegeben werden (wie *l*.)

**!**

Ein Operator, der die angegebenen Zeilen aus dem Puffer als Standard-Eingabe an das angegebene Systemkommando übergibt; diese Zeilen werden dann durch die Standard-Ausgabe des Kommandos ersetzt. Auf ! folgt ein Positionierungs-Kommando sowie das Systemkommando (die Eingabe wird wie üblich durch Betätigung der Wagenrücklauf-Taste abgeschlossen); über das Positionierungskommando werden die Zeilen angegeben, die übergeben werden sollen (alle Zeilen von der aktuellen Zeile bis zur angegebenen Zeile). Dem *!* kann ein Wiederhlolungsfaktor vorangestellt werden.

Mit zwei ! (Ausrufezeichen), denen ein Wiederholungsfaktor vorangestellt ist, kann angegeben werden, wieviele Zeilen ab der aktuellen Zeile übergeben werden sollen.

"

Wird dem Namen eines Puffers vorangestellt. Es gibt Puffer mit den Namen 1 bis 9, in die *vi* gelöschten Text speichert. In den Puffern *a-z* kann der Benutzer gelöschten oder mit *yank* gesichterten Text speichern (siehe auch *y*, unten).

$

Die Schreibmarke wird zum Ende der aktuellen Zeile bewegt; über eine Zahl kann ein Wiederholungsfaktor angegeben werden (mit 2$ z.B. wird die Schreibmarke zum Ende der nächsten Zeile bewegt).

%

Die Schreibmarke wird zu der (geschweiften) Klammer bewegt, die zu der (geschweiften) Klammer an der aktuellen Schreibmarkenposition paßt.

&

Wirkt wie das *ex* Kommando *&* (das letzte Substituierungs-Kommando wird wiederholt).

'

Folgt darauf ein ', so wird die Schreibmarke zum Beginn der vorhergehenden aktuellen Zeile bewegt. Folgt auf dieses Zeichen einer der Buchstaben *a-z*, so wird die Schreibmarke zu der Zeile bewegt, die mit diesem Buchstaben markiert ist (siehe *m* Kommando), und zwar zum ersten Zeichen in dieser Zeile, bei dem es sich nicht um ein Leer- oder Tabulatorzeichen handelt.

Wird dieses Kommando mit einem Operator kombiniert, mit dem der Umfang des zu bearbeitenden Textes angegeben wird (z.B. *d*), so werden vollständige Zeilen behandelt (siehe auch `).

Folgt darauf ein `, so wird die Schreibmarke exakt an die vorhergehende aktuelle Position gesetzt (die aktuelle Position wird vor der Durchführung jedes nicht-relativen Positionierungskommandos markiert). Folgt darauf einer der Buchstaben *a-z*, so wird die Schreibmarke zu der Zeile bewegt, die mit diesem Buchstaben markiert wurde (siehe *m* Kommando), und zwar zur markierten Zeichenposition.

In Verbindung mit einem Operator wie *d*, mit dem der zu bearbeitende Bereich angegeben wird, wird das betreffende Kommando auf den Bereich angewandt, der sich zwischen der markierten Position und der aktuellen Position in der Zeile befindet (siehe auch '.)

[[

Zurück auf vorangehende Abschnittsgrenze positionieren. Ein Abschnitt ist festgelegt durch den Wert der *ex*-Variablen *sections*. Zeilen, die mit Seitenvorschub (^L) oder { beginnen, zählen auch als Abschnittsgrenze.

Ist die *ex*-Variable *lisp* gesetzt, zählt jede ( am Anfang einer Zeile als Abschnittsgrenze.

]]

Vorwärts auf nächste Abschnittsgrenze positionieren. (siehe [[).

^

Die Schreibmarke wird zum ersten Zeichen in der aktuellen Zeile bewegt, bei dem es sich nicht um ein Leer- oder Tabulatorzeichen handelt.

(

Die Schreibmarke wird zurück zum Anfang eines Satzes bewegt. Ein Satz wird durch das Zeichen . (Punkt), ein ! oder ein ? abgeschlossen, auf das entweder das Zeilenende oder zwei Leerzeichen folgen. Zwischen ., ! oder ? und dem Zeilenende bzw. den Leerzeichen kann eine beliebige Anzahl der der Zeichen ), ] " und ` stehen. Mit einer Zahl kann ein Wiederholungsfaktor angegeben werden.

Ist die Variable *lisp* gesetzt, so wird die Schreibmarke an den Anfang eines *lisp s*-Ausdruckes bewegt. Auch der Beginn eines Abschnittes oder Absatzes wird als Beginn eines Satzes interpretiert (siehe *{* und *[[*).

)

Die Schreibmarke wird zum Anfang des nächsten Satzes bewegt. Über eine Zahl kann ein Wiederholungsfaktor angegeben werden (siehe *(*.)).

{

Die Schreibmarke wird zum Anfang des aktuellen Absatzes bewegt. Durch welchen Makro ein Absatz eingeleitet wird, ist in der Variablen *paragraphs* vorgegeben. Auch eine vollständig leere Zeile und der Beginn eines Abschnittes (siehe *[[*, oben) wird als Beginn eines Absatzes interpretiert. Über eine Zahl kann ein Wiederholungsfaktor angegeben werden.

}

Die Schreibmarke wird zum Anfang des nächsten Absatzes bewegt. Über eine Zahl kann ein Wiederholungsfaktor angegeben werden (siehe *{*).

|

Eine Zahl muß angegeben werden; die Schreibmarke wird dann (falls möglich) zu der Spalte bewegt, die über diese Zahl angegeben wurde).

+

Die Schreibmarke wird zum ersten Zeichen in der nächsten Zeile bewegt, bei dem es sich nicht um ein Leer- oder Tabulatorzeichen handelt; eine Zahl kann angegeben werden, um wieviele Zeilen die Schreibmarke in im Vorwärtsrichtung bewegt werden soll (wie bei *^M*).

,

Die Umkehrung des letzten *f*, *F*, *t* oder *T* Kommandos; die Zeile wird also in der entgegengesetzten Richtung durchsucht. Über eine Zahl kann ein Wiederholungsfaktor angegeben werden.

-

Die Schreibmarke wird zum ersten Zeichen in der vorhergehenden Zeile bewegt, bei dem es sich nicht um ein Leer- oder Tabulatorzeichen handelt. Über eine Zahl kann ein Wiederholungsfaktor angegeben werden.

.

Das letzte Kommando, mit dem der Puffer geändert wurde, wird wiederholt. Über eine Zahl kann ein Wiederholungsfaktor angegeben werden.

/

Eine Zeichenfolge in der letzten Bildschirmzeile wird als regulärer Ausdruck interpretiert, und das nächste Vorkommen einer passenden Zeichenfolge wird gesucht. Die Suche beginnt, wenn die Eingabe des Musters durch die Betätigung der WagenrücklaufTaste abgeschlossen wird; die Suche kann durch das Signal **SIGINT** beendet werden.

Wird dieses Kommando in Kombination mit einem Operator verwendet, um einen zu bearbeitenden Textbereich zu definieren, so wird der Bereich bearbeitet, der zwischen der aktuellen Schreibmarken-Position und dem Beginn der passenden Zeichenfolge steht.

**0**

Die Schreibmarke wird zum ersten Zeichen in der aktuellen Zeile bewegt (mit einer vorangestellten Ziffer ungleich 0 wird es nicht als Kommando interpretiert).

:

Leitet ein *ex* Kommando ein. Der Doppelpunkt (:) wird ebenso wie das eingegebene Kommando in der untersten Bildschirmzeile ausgegeben; ausgeführt wird es, nachdem die Eingabe durch Betätigung der Wagenrücklauf-Taste abgeschlossen wurde.

;

Die letzte Suche nach einem einzelnen Zeichen mit *f F t* oder *T* wird wiederholt. Über eine Zahl kann ein Wiederholungsfaktor angegeben werden.

<

Die Zeilen werden um *shiftwidth* Positionen nach links verschoben. Hierauf kann ein Kommando angegeben werden, mit dem die Schreibmarke zu den angegebenen Zeilen bewegt werden kann. Wird eine Zahl angegeben, so wird sie zu das Positionierkommando übergeben.

<< bewirkt, daß die aktuelle Zeile verschoben wird (oder *zahl* Zeilen, ab der aktuellen Zeile verschoben werden).

>

Ein Operator, mit dem Zeilen um *shiftwidth* Positionen nach rechts verschoben werden (siehe <).

**=**

Ist die Variable *lisp* gesetzt, so werden die angegebenen Zeilen so eingerückt, als wären bei ihrer Eingabe *lisp* und *autoindent* gesetzt. Über eine Zahl kann angegeben werden, wieviele Zeilen bearbeitet werden sollen; dieselbe Wirkung hat ein Positionierbefehl.

**?**

Suche in Rückwärtsrichtung; also das Gegenstück zu / (siehe /).

**A**

Am Ende der Zeile wird Text eingefügt (wie *$a*).

**B**

Die Schreibmarke wird zum Anfang des vorhergehenden Wortes bewegt (ein Wort darf keine Leerzeichen enthalten). Über eine Zahl kann ein Wiederholungsfaktor angegeben werden.

**C**

Der Rest der aktuellen Zeile wird durch Text ersetzt (wie *c$*).

**D**

Der Rest der aktuellen Zeile wird gelöscht (wie *d$*).

**E**

Die Schreibmarke wird ans Ende eines Wortes bewegt (ein Wort darf keine Leerzeichen enthalten). Über eine Zahl kann angegeben werden, um wieviele Wörter die Schreibmarke in Vorwärtsrichtung bewegt werden soll.

**F**

Auf dieses Kommando muß ein einzelnes Zeichen folgen; nach diesem Zeichen sucht *vi* in der aktuellen Zeile in Rückwärtsrichtung und bewegt die Schreibmarke gegebenenfalls an diese Stelle. Über eine Zahl kann ein Wiederholungsfaktor angegeben werden.

**G**

Ist dem Kommando eine Zeilennummer vorangestellt, so wird die Schreibmarke zu dieser Zeile bewegt; andernfalls wird er zum Ende der Datei bewegt.

**H**

Die Schreibmarke wird zur ersten Bildschirmzeile bewegt. Wird eine Zahl angegeben, so wird die Schreibmarke zu der Zeile bewegt, die um *zahl* Zeilen vom oberen Rand des Bildschirms entfernt ist. Die Schreibmarke wird in beiden Fällen zum ersten Zeichen in der Zeile bewegt, bei dem es sich nicht um ein Leer- oder Tabulatorzeichen handelt.

**I**

Am Anfang einer Zeile wird Text eingefügt.

**J**

Die nachfolgende Zeile wird an die aktuelle Zeile angehängt, wobei an der Nahtstelle die geeignete Anzahl von Leer- oder Tabulatorzeichen eingefügt wird: ein Leerzeichen zwischen Wörtern, zwei Leerzeichen nach einem Punkt, kein Leerzeichen, wenn das erste Zeichen in der nächsten Zeile eine schließende Klammer *)* ist. Über eine Zahl kann gegebenenfalls die Anzahl der zusammenzufügenden Zeilen angegeben werden.

**L**

Die Schreibmarke wird zum ersten Zeichen in der letzten Bildschirmzeile bewegt, bei der es sich nicht um ein Leer- oder Tabulatorzeichen handelt. Wird eine Zahl angegeben, so wird die Schreibmarke zu der Zeile bewegt, die sich *zahl* Zeilen oberhalb der untersten Bildschirmzeile befindet. Wird ein Operator verwendet, so sind ganze Zeilen betroffen.

**M**

Die Schreibmarke wird zum ersten Zeichen in der mittleren Bildschirmzeile bewegt, bei dem es sich nicht um ein Leer- oder Tabulatorzeichen handelt.

**N**

Das Gegenstück zu *n*; *vi* sucht nach der Zeichenfolge, die zum nächsten durch */* oder *?* eingeschlossenen Muster paßt, jedoch in der umgekehrten Richtung.

**O**

Oberhalb der aktuellen Zeile wird eine neue Zeile eingefügt, und der Eingabe-Modus wird eingeschaltet.

**P**

Der zuletzt gelöschte Text wird vor/über der Schreibmarke eingefügt. Handelte es sich bei dem gelöschten Text um ganze Zeilen, so werden oberhalb der Schreibmarke auch ganze Zeilen eingefügt; andernfalls wird der Text unmittelbar vor der Schreibmarke eingefügt.

Diesem Kommando kann der Name eines Puffers (" *x*) vorangestellt sein, dessen Inhalt eingefügt werden soll.

**Q**

*vi* wird beendet, und der *ex*-Kommandomodus wird eingeschaltet.

**R**

Die Zeichen auf dem Bildschirm werden durch die eingegebenen Zeichen ersetzt, bis die Eingabe durch ein ESC abgeschlossen wird.

**S**

Ganze Zeilen werden ersetzt (entspricht *cc*). Mit einer Zahl kann ein Wiederholungsfaktor angegeben werden.

**T**

Auf dieses Zeichen muß ein einzelnes Zeichen folgen; *vi* durchsucht die aktuelle Zeile in Rückwärtsrichtung nach diesem Zeichen und bewegt die Schreibmarke gegebenenfalls hinter dieses Zeichen. Über eine Zahl kann ein Wiederholungsfaktor angegeben werden.

**U**

Die letzten Änderungen an der aktuellen Zeile werden rückgängig gemacht.

**W**

Die Schreibmarke wird zum Beginn des nächsten Wortes in der aktuellen Zeile bewegt; in einem Wort dürfen keine Leer- oder Tabulatorzeichen enthalten sein. Über eine Zahl kann ein Wiederholungsfaktor angegeben werden.

**X**

Das Zeichen vor der Schreibmarke wird gelöscht. Über eine Zahl kann ein Wiederholungsfaktor angegeben werden; jedoch sind nur die Zeichen in der aktuellen Zeile betroffen.

**Y**

Eine Kopie der aktuellen Zeile wird in den temporären Puffer geschrieben (entspricht *yy*). Über eine Zahl kann angegeben werden, wieviele Zeilen kopiert werden sollen. Sollen die Zeilen in einen bestimmten Puffer geschrieben werden, so kann ein Puffername angegeben werden.

**ZZ**

Der Editor wird verlassen; wurden am Puffer seit dem letzten *write* Kommando Änderungen vorgehommen, so wird der Inhalt des Puffers in die Datei mit dem aktuellen Dateinamen geschrieben (wie beim *ex*-Kommando *x*).

**a**

Der Eingabemodus wird eingeschaltet; der eingegebene Text wird hinter der aktuellen Position des Cursors eingefügt. Über eine Zahl kann angegeben werden, wieviele Kopien des Textes eingefügt werden sollen; der einzufügende Text muß dann jedoch in einer Zeile enthalten sein.

**b**

Die Schreibmarke wird zum Beginn des vorhergehenden Wortes in der aktuellen Zeile bewegt. Ein Wort ist eine Folge von alphanumerischen Zeichen oder eine Folge von Sonderzeichen. Über eine Zahl kann ein Wiederholungsfaktor angegeben werden.

**c**

Auf dieses Kommando muß ein Positionierkommando folgen. Der Text im angegebenen Bereich wird gelöscht und der Eingabemodus wird eingeschaltet. Der Text im angegebenen Bereich wird durch den Text ersetzt, der dann eingegeben wird. Wenn mehr als ein Teil einer einzigen Zeile ersetzt wird, wird der gelöschte Text in den Puffern 1-9 gespeichert, d.h. er kann mit *p* wiederhergestellt werden. Ist die aktuelle Zeile nur teilweise betroffen, so wird das letzte zu löschende Zeichen mit einem *$* gekennzeichent. Wird eine Zahl angegeben, so wird sie dem Positionierkommando übergeben. Das Kommando *cc* bewirkt, daß die aktuelle Zeile vollständig geändert wird.

**d**

Auf dieses Kommando muß ein Positionierungskommando folgen. Der angegebene Bereich wird gelöscht. Wenn mehr als ein Teil einer einzigen Zeile gelöscht wird, wird der gelöschte Text in den Puffern 1 - 9 gespeichert, d.h. er kann mit *p* wiederhergestellt werden. Wird eine Zahl angegeben, so wird sie an das Positionierungskommando übergeben. Das Kommando *dd* bewirkt, daß die aktuelle Zeile vollständig gelöscht wird.

**e**

Die Schreibmarke wird an das nächste Wortende bewegt. Über eine Zahl kann ein Wiederholungsfaktor angegeben werden. Ein Wort ist wie bei *b* definiert.

**f**

Auf dieses Kommando muß ein einzelnes Zeichen folgen; *vi* durchsucht die aktuelle Zeile in Vorwärtsrichtung nach diesem Zeichen und bewegt die Schreibmarke gegebenenfalls an diese Stelle. Über eine Zahl kann ein Wiederholungsfaktor angegeben werden.

**h**

Die Schreibmarke wird um ein Zeichen nach links bewegt (entspricht ^*H* und [←]). Über eine Zahl kann ein Wiederholungsfaktor angegeben werden.

**i**

Der Eingabemodus wird eingeschaltet, und der eingegebene Text wird vor der aktuellen Position des Cursors eingefügt (siehe *a*).

**j**

Die Schreibmarke wird innerhalb der aktuellen Spalte in die nächste Zeile bewegt (entspricht ^*J* und ^*N*)

**k**

Die Schreibmarke wird in die darüberliegende Zeile in die aktuelle Spalte bewegt (entspricht ^*P* und [↑].)

**l**

Die Schreibmarke wird um ein Zeichen nach rechts bewegt (entspricht <Leerzeichen> plus [→].)

**m**

Auf dieses Kommando muß ein Kleinbuchstabe *x* folgen; mit diesem Buchstaben wird die aktuellen Position des Cursors markiert. Soll die Schreibmarke später genau an diese Position bewegt werden, so lautet die Adresse *`x*; die Zeile wird mit *'x* angesprochen.

**n**

Der letzte / bzw. *?* Suchbefehl wird wiederholt.

**o**

Unterhalb der aktuellen Zeile wird eine Zeile eingefügt, und *vi* wird in den Eingabemodus umgeschaltet; ansonsten wie *O*.

**p**

Der Text wird hinter die Schreibmarke bzw. unterhalb der Schreibmarke positioniert; ansonsten wie *P*.

**r**

Auf dieses Kommando muß ein einzelnes Zeichen folgen; durch dieses Zeichen wird das Zeichen unterhalb des Cursors ersetzt (bei dem neuen Zeichen kann es sich auch um ein Neue Zeile-Zeichen handeln). Wird eine Zahl *n* angegeben, so werden alle folgenden *n* Zeichen durch das angegebene Zeichen ersetzt.

**s**

Das Zeichen unterhalb des Cursors wird gelöscht, und *vi* wird in den Eingabemodus umgeschaltet; das gelöschte Zeichen wird dann durch den eingegebenen Text ersetzt. Über eine Zahl kann angegeben werden, wieviele Zeichen in der aktuellen Zeile ersetzt werden sollen. Das letzte zu löschende Zeichen wird wie bei *c* durch ein *$* markiert.

**t**

Auf dieses Kommando muß ein einzelnes Zeichen folgen; *vi* durchsucht die aktuelle Zeile in Vorwärtsrichtung nach diesem Zeichen und bewegt die Schreibmarke gegebenenfalls an die Position unmittelbar vor diesem Zeichen. Über eine Zahl kann ein Wiederholungsfaktor angegeben werden.

**u**

Das letzte Kommando, durch das der aktuelle Puffer geändert wurde, wird rückgängig gemacht. Durch ein erneutes *u* Kommando kann auch das letzte *u* Kommando rückgängig gemacht werden, usw. Folgt dieses Kommando auf ein *insert*-Kommando, mit dem zwei oder mehrere Zeilen eingefügt wurden, so werden diese Zeilen in den Puffern 1-9 gespeichert.

**w**

Die Schreibmarke wird zum Anfang des nächsten Wortes bewegt; ein Wort ist wie bei *b* definiert. Über eine Zahl kann angegeben werden, zum wievielten Wort die Schreibmarke bewegt werden soll.

**x**

Das Zeichen unterhalb des Cursors wird gelöscht. Wird eine Zahl *n* angegeben, so werden, beginnend bei der aktuellen Position des Cursors, *n* Zeichen in der aktuellen Zeile gelöscht.

**y**

Auf dieses Kommando muß ein Positionierungs-Kommando folgen. Der angegebene Text wird in einen Zwischenspeicher geschrieben. Ist dem Kommando der Name eines Puffers (*"x*) vorangestellt, so wird der Text zusätzlich in diesen Puffer geschrieben. Das Kommando *yy* bewirkt, daß die aktuelle Zeile vollständig in den Puffer geladen wird.

**z**

Der Bildschirm wird neu aufgebaut; die Position der aktuellen Zeile wird durch die folgenden Zeichen angegeben: Beim Zeichen <Wagenrücklauf> wird sie in die erste Bildschirmzeile gesetzt, beim Zeichen . in die Mitte des Bildschirms, und beim Zeichen - in die unterste Bildschirmzeile. Zwischen *z* und einem der obengenannten Zeichen kann eine Zahl stehen; über diese Zahl wird die Größe der neu aufgebauten Bildschirmanzeige angegeben. Ist *z* eine Zahl vorangestellt, so bezieht diese sich auf die Nummer der Zeile, die anstelle der aktuellen Zeile in die Mitte des Bildschirms gesetzt werden soll.

## SIEHE AUCH

*ex*(1)

Manual SINIX Einführung [1].

## NAME

**wait** - Auf Prozeß-Ende warten

## DEFINITION

**wait[␣PID]**

## BESCHREIBUNG

Wird kein Argument angegeben, so wartet *wait*, bis alle durch die aktuelle Shell mit & gestarteten Hintergrundprozesse beendet sind; wird ein Prozeß wegen eines Fehlers beendet, so wird eine entsprechende Fehlermeldung abgegeben. Wenn die Zahl PID (process identity) angegeben wird und es sich dabei um die Prozeßnummer eines Hintergrundprozesses handelt, so wartet *wait* auf den Abschluß dieses Prozesses. Ist PID dagegen kein Hintergrundprozeß, so wartet *wait* den Abschluß aller von der aktuellen Shell gestarteten Hintergrundprozesse ab.

## HINWEIS

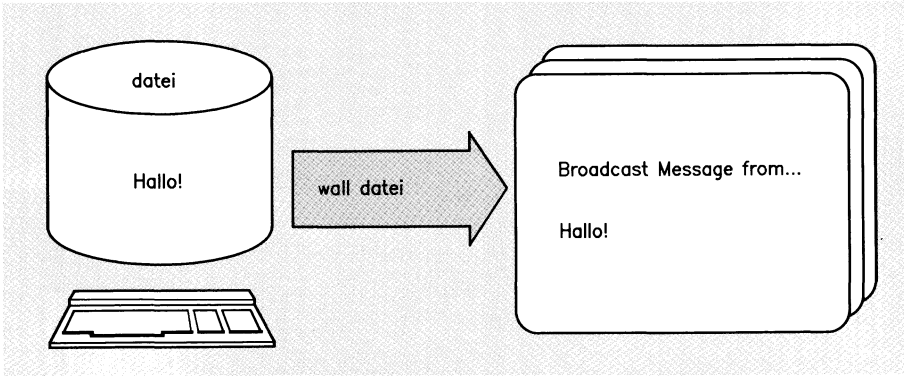
Dieses Kommando erzeugt in keinem Fall einen eigenen Prozeß.

## SIEHE AUCH

*sh*(1), *wait*(2)

## NAME

**wall** - An alle Benutzer schreiben (write to all users)




## DEFINITION

```
wall
```

## BESCHREIBUNG

*wall* liest von der Standard-Eingabe eine Nachricht bis zur Erkennung eines Dateiende-Zeichens. Dann gibt es diese Nachricht auf den Datensichtstationen aller Benutzer aus, die momentan beim System angemeldet sind; der Meldung wird ein Code vorangestellt, aus dem der Sender der Nachricht hervorgeht.

[!] – Nur der Systemverwalter kann trotz der Schutzmechanismen gegen unerwünschte Nachrichten, die sich die Benutzer errichten können, (siehe *mesg*(1)) Nachrichten an alle senden.

– Standardmäßig kann nur der Systemverwalter *wall* aufrufen!

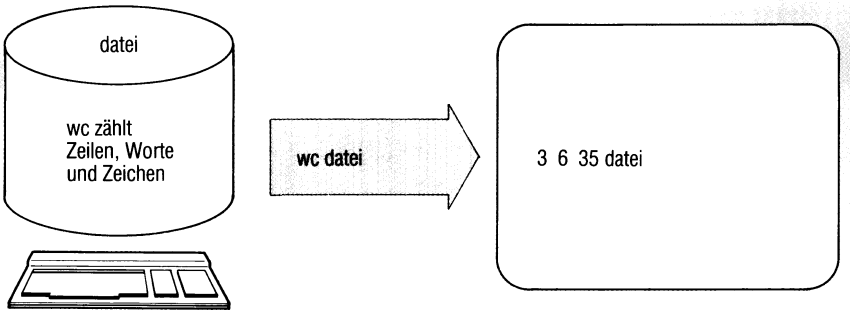
– *wall* kann man z.B. benutzen, um alle Benutzer vor einem Herunterfahren des Systems zu warnen.

## SIEHE AUCH

*mesg*(1), *write*(1)

## NAME

**wc** - Wörter zählen (word count)




## DEFINITION

**wc[␣schalter][␣datei...]**

## BESCHREIBUNG

*wc* zählt die in *datei* enthaltenen Zeilen, Wörter und Zeichen.

## SCHALTER

**-l**

*wc* gibt die Anzahl der Zeilen aus.

**-w**

*wc* gibt die Anzahl der Wörter aus. Ein Wort ist als Folge von Zeichen definiert, das durch Leer-, Tabulator- oder ein Neue-Zeile-Zeichen abgeschlossen ist.

**-c**

*wc* gibt die Anzahl der Zeichen aus.

Standard (keine Angabe): *-lwc*

**OPERANDEN**

datei

Name der Datei, deren Zeilen, Wörter und Zeichen gezählt werden sollen.
Der Name wird zusammen mit den ermittelten Werten ausgegeben.
Standard (keine Angabe): *wc* liest von Standard-Eingabe

datei...

Bei mehreren Dateien gibt *wc* zusätzlich für alle angegebenen Dateien eine Gesamtsumme aus.

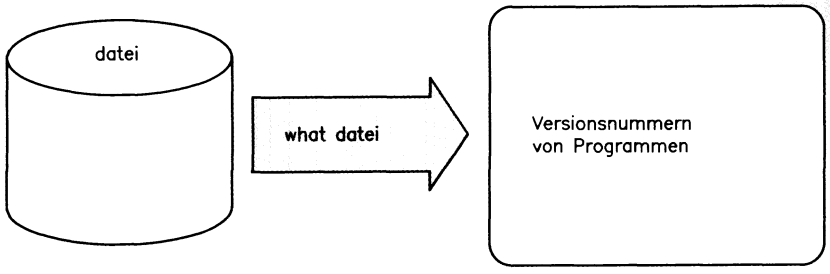
**BEISPIEL**

Für die Dateien *logik, plan* und *rest* soll die Anzahl der Zeilen, Wörter und Zeichen ausgegeben werden.

```
$ wc logik plan rest
     27     139    1077 logik
      5      15     140 plan
      3       6      51 rest
     35     160    1268 total
```

## NAME

**what** - Dateien identifizieren




## DEFINITION

**what[␣-s]␣**datei...

## BESCHREIBUNG

*what* durchsucht die angegebenen Dateien nach jedem Vorkommen des Musters @(#) und gibt die darauffolgenden Zeichen bis zum ersten ", >, \, Neue-Zeile-Zeichen oder dem ASCII-Zeichen NUL aus.

Der Text, den *what* ausgibt, dient dazu, den Inhalt der Datei zu identifizieren.

## SCHALTER

**-s**

Die Ausführung von *what* wird beendet, sobald *what* das erste Muster @(#) gefunden und die entsprechenden Zeichen ausgegeben hat.

## OPERANDEN

datei

Für *datei* können Sie den Namen einer beliebigen Datei angeben. Sie können beliebig viele Dateinamen angeben.

**ENDESTATUS**

0 wenn *what* das Muster gefunden hat
1 in allen anderen Fällen.

**HINWEIS**

Sie können die *what*-Zeichenreihe `@(#)` und die darauffolgenden Informationen per Hand in die Dateien einfügen. Üblicherweise wird *what* jedoch in Verbindung mit dem SCCS verwendet (siehe *get*(1D)). Eine Einführung ins SCCS finden Sie im Buch SINIX Einführung [1].

**BEISPIEL**

Das C-Programm *prog.c* enthalte die folgende Anweisungszeile:

```
char ident[] = "@(#)Version 2.3, erstellt am 10/11/87";
```

Sie übersetzen das C-Programm; das übersetzte bzw. ausführbare Programm stehe in der Datei *prog.o* bzw. *a.out*. Mit dem Kommando

```
$ what prog.c prog.o a.out
```

erhalten Sie die folgende Information:

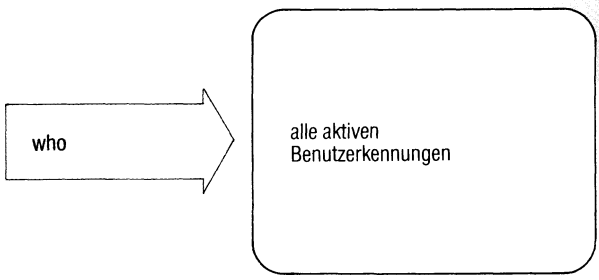
```
prog.c:
        Version 2.3, erstellt am 10/11/87
prog.o:
        Version 2.3, erstellt am 10/11/87
a.out:
        Version 2.3, erstellt am 10/11/87
```

**SIEHE AUCH**

*get*(1D).

## NAME

**who** - Aktive Benutzerkennungen anzeigen (who is on the system)




## DEFINITION

**who[␣dateiname][␣am␣i]**

## BESCHREIBUNG

*who* gibt für jeden momentan am System angemeldeten Benutzer folgendes aus:

- Benutzername
- Name der Datensichtstation
- Anmeldezeit

## OPERANDEN

dateiname

*who* bezieht seine Informationen aus der Datei *dateiname*. Üblicherweise wird hier die Datei */usr/adm/wtmp* angegeben. Sie enthält einen Bericht über alle Logins, seit diese Datei erstellt wurde.

Für Anmeldungen, Abmeldungen und Systemabstürze seit Erstellung der Datei */usr/adm/wtmp* erstellt *who* eine Ausgabe wie folgt:

Für Anmelden (login):

- Benutzername
- Name der Datensichtstation (ohne */dev*)
- Anmeldezeit

Für Abmelden (logout):

- wie bei *login*, aber ohne Benutzername

Für Neuladen (reboot):

- *x* statt dem Gerätenamen *n*
- Zeitpunkt, der den Systemabsturz angibt.

Standard (keine Angabe): *who* bezieht seine Informationen aus der Datei */etc/utmp*.

**am i**
**are you**
*who* erstellt die Ausgabe für Ihren Benutzernamen.

## DATEIEN

*/etc/utmp*
Datei, aus der *who* seine Informationen holt.

## BEISPIEL

```
$ who am i
Bolte!moritz   11y10   Jan  19 13:00
```

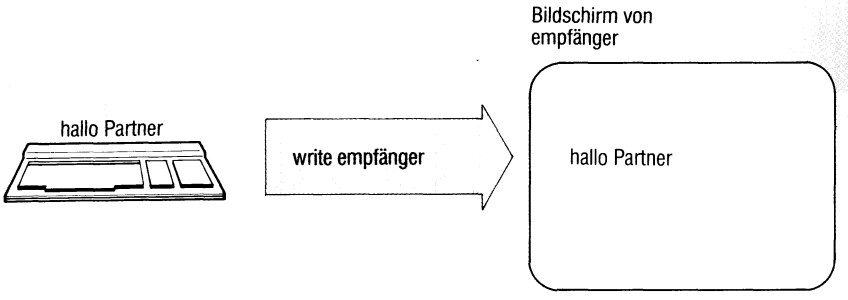
*Bolte* ist der Rechnername, *moritz* der Benutzername.

## SIEHE AUCH

*mesg*(1)

## NAME

**write** - Dialog mit anderem Benutzer (write to another user)




## DEFINITION

**write**␣empfänger[␣station]

## BESCHREIBUNG

Mit *write* können Sie mit anderen Benutzern kommunizieren. *write* schreibt Ihren Text, d.h. die Standard-Eingabe auf die Datensichtstation eines anderen Benutzers. Unmittelbar nach dem Aufrufen sendet *write* an den Empfänger Ihren Text als Nachricht.

Der Empfänger der Nachricht kann z.B. antworten, indem er beim Empfang der Nachricht von seiner Datensichtstation aus eingibt: *write empfänger*, wobei hier die Kennung des ursprünglichen Senders für *empfänger* anzugeben ist. Alle Eingaben der beiden Benutzer mit Ausnahme der Kommandoeingaben (siehe unten) werden bei Betätigen der Tasten END oder ↵ auf die Datensichtstation des anderen Benutzers geschrieben; Abbrechen können Sie den Dialog mit END am Beginn einer neuen Zeile oder DEL. *write* schreibt dann

```
EOF
```

auf die Datensichtstation des anderen Benutzers und beendet sich. Wünscht der Empfänger keinen weiteren Dialog, so kann er das Kommando *mesg␣n* eingeben.

Ist ein Benutzer an mehreren Datensichtstationen gleichzeitig angemeldet, so muß über das Argument *station* angegeben werden, an welche Datensichtstation die Nachricht geschickt werden soll (z.B. *tty00*); andernfalls wird versucht, an die Datensichtstation zu schreiben, die als erste für diesen Benutzer in der Datei */etc/utmp* steht und eine entsprechende Meldung an den Sender geschickt.

Mit dem Kommando *mesg* kann ein Benutzer einem anderen Benutzer die Schreiberlaubnis bewilligen oder verweigern. Bei einigen Kommandos sind Nachrichten nicht erlaubt, da es sonst Probleme mit der eigenen Ausgabe geben kann. Hat der Benutzer, der die Nachricht schicken will, jedoch die Privilegien des Systemverwalters, so wird die Nachricht auf jeden Fall auf der Datensichtstation des Empfängers angezeigt, unabhängig davon, ob sie schreibgeschützt ist, oder nicht.

Steht zu Beginn einer Zeile das Zeichen *!*, so wird der Rest der Zeile von *write* an die Shell als Kommando übergeben. Das Kommando wird ausgeführt, ohne den Dialog zu unterbrechen.

## FEHLER

In folgenden Fällen werden Fehler gemeldet:

- Der Empfänger ist nicht angemeldet.
- Die Datensichtstation des Empfängers ist schreibgeschützt (siehe *mesg*(1)).
- Die eigene Datensichtstation ist für die Nachrichten anderer Benutzer gesperrt.
- Der Empfänger ändert nach dem Aufruf von *write* die Zugriffsrechte (*mesg n*).

## DATEIEN

*/etc/utmp*
Datei, in der alle Anmeldungen registriert sind.

## BEISPIEL

Max an Datensichtstation 1

```
$ write moritz
morgen lassen wir die
Sache steigen!
Message from Bolte!moritz on
tty02 at 18:00
einverstanden!
EOF
$
```

Moritz an Datensichtstation 2

```
Message from Bolte!max on tty01 at 17:58
morgen lassen wir die
Sache steigen!
$ write moritz

einverstanden!
EOF
$
```

## SIEHE AUCH

*mesg*(1), *who*(1)

## NAME

**xargs** - Argumentenliste(n) aufbauen und Kommando ausführen

## DEFINITION

**xargs**[␣schalter...][␣kommando[anfangs-argument...]]

## BESCHREIBUNG

*xargs* verknüpft *anfangs-argument* bzw. die Anfangs-Argumente mit Argumenten, die es von der Standard-Eingabe liest und führt das angegebene *kommando* ein- oder mehrere Male durch. Wieviele Argumente für jeden Kommandoaufruf eingelesen werden und wie sie verknüpft werden, hängt davon ab, welche *schalter* angegeben sind.

Fehlt *kommando*, so wird *echo* ausgeführt.

Die von der Standard-Eingabe gelesenen Argumente müssen zusammenhängende Folgen von Zeichen sein, die von einem oder mehreren Leer-, Tabulator- oder einem Neue-Zeile-Zeichen abgeschlossen werden; leere Zeilen werden in jedem Fall gelöscht. In einem Argument dürfen keine Leer- oder Tabulatorzeichen enthalten sein, es sei denn, sie sind entwertet oder apostrophiert. Zeichen, die in Anführungszeichen oder Hochkommas eingeschlossen sind, werden "wörtlich" genommen; die Apostrophen werden gelöscht. Außerhalb einer nicht apostrophierten Zeichenfolge entwertet ein Gegenschrägstrich das darauffolgende Sonderzeichen.

Zu Beginn jeder Argumentenliste stehen die *anfangs-argumente*, gefolgt von Argumenten, die von der Standard-Eingabe eingelesen wurden (Ausnahme: bei Schalter *-i*). Mit den Schaltern *-i*, *-l* und *-n* wird festgelegt, wie die Argumente für jeden Kommandoaufruf ausgewählt werden. Ist keiner dieser Schalter gesetzt, so werden zunächst die *anfangs-Argumente*, dann die Argumente von der StandardEingabe eingelesen, bis ein interner Puffer voll ist. Daraufhin wird das *kommando* unter Berücksichtigung aller Argumente ausgeführt. Dieser Vorgang wird so lange wiederholt, bis alle Argumente abgearbeitet sind. Heben Schalter sich gegenseitig auf (z.B. *-l* und *-n*), so hat der letzte Schalter Priorität.

## SCHALTER

**-l**anzahl

*kommando* wird für jede *anzahl* nicht-leerer Argumentenzeilen, die *xargs* von der Standard-Eingabe liest, ausgeführt. Bleiben beim letzten Aufruf von *kommando* weniger als *anzahl* Zeilen übrig, so wird *kommando* für diese Zeilen ausgeführt. Eine Zeile wird bei ihrem ersten Neue-Zeile-Zeichen als abgeschlossen betrachtet, es sei denn, das letzte Zeichen in der Zeile ist ein Leer- oder Tabulatorzeichen; im letzteren Fall wird die Zeile in der nächsten nicht-leeren Zeile fortgesetzt. Fehlt *anzahl*, so wird der Standard-Wert 1 eingesetzt. Der Schalter *x* wird in jedem Fall gesetzt.

**-i**ersatz

Einfüge-Modus: *kommando* wird für jede von der Standard-Eingabe eingelesene Zeile ausgeführt, wobei jede Zeile als ein Argument interpretiert und für jedes Vorkommen von *ersetzungszeichenfolge* in die Liste der *anfangs-Argumente*s eingefügt wird. Maximal 5 Argumente in der Liste mit den *anfangs-Argumenten* können jeweils ein oder mehrmals *ersetzungszeichenfolge* enthalten. Leer- und Tabulatorzeichen zu Beginn jeder Zeile werden ignoriert. Die erstellten Argumente dürfen aus maximal 255 Zeichen bestehen; auch hier wird der Schalter *-x* in jedem Fall gesetzt. Fehlt *ersetzungszeichenfolge*, so wird *{}* verwendet.

**-n**n

*kommando* wird unter Verwendung von möglichst vielen (maximal *n*) von der Standard-Eingabe eingelesenen Argumenten ausgeführt; Übersteigt die Gesamtgröße der Argumente die Obergrenze von *größe* Zeichen, so werden weniger Argumente verwendet; dies gilt auch dann, wenn beim letzten Aufruf von *kommando* weniger als *n* Argumente übrig sind. Wird auch der Schalter *-x* gesetzt, so darf die Länge der einzelnen Argumentenlisten die mit *größe* festgelegte Obergrenze nicht überschreiten, unabhängig vom Wert von *n*. andernfalls wird *xargs* beendet.

**-t** Protokoll:

Das *kommando* und jede erstellte Argumentenliste werden unmittelbar bevor sie abgearbeitet werden auf die Standard-Fehlerausgabe geschrieben.

**-p** Interaktiver Modus:
Der Benutzer wird bei jedem Aufruf von *kommando* gefragt, ob *kommando* ausgeführt werden soll. Der Protokollierungs-Modus (Schalter *-t*) wird eingeschaltet, und das aufgerufene Kommando wird, gefolgt von der Eingabeaufforderung *?...* angezeigt. Beginnt die Antwort mit *y* (die übrigen Buchstaben spielen keine Rolle), so wird das Kommando ausgeführt; jede andere Antwort bewirkt, daß daß der Kommando-Aufruf ignoriert wird (als Antwort genügt hier bereits die Betätigung der Wagenrücklauf-Taste.

**-x**
*xargs* wird beendet, wenn die Länge einer Argumentenliste die angegebene Obergrenze *größe* übersteigen würde. Dieser Schalter wird bei den Schaltern *-i* und *-l* in jedem Fall gesetzt. Ist keiner der Schalter *-i*, *-l* oder *-n* gesetzt, so darf die Gesamtlänge aller Argumente *größe* nicht überschreiten.

**-s**größe
Mit *größe* wird die maximale Länge einer Argumentenliste angegeben. *größe* muß eine positive ganze Zahl kleiner gleich 470 sein. Ist der Schalter *-s* nicht gesetzt, so wird der Standard-Wert 470 verwendet. Zu beachten ist, daß in *größe* ein zusätzliches Zeichen für jedes Argument und die Zeichen im Kommandonamen bereits enthalten sind.

**-e**dateiende
*dateiende* wird als logisches Dateiende-Zeichen interpretiert. Ist dieser Schalter nicht gesetzt, so wird standardmäßig das Unterstreichungszeichen (_) verwendet. Fehlt *dateiende*, so verliert das Unterstreichungszeichen seine spezielle Bedeutung. *xargs* liest die Standard-Eingabe entweder bis zum Erreichen des tatsächlichen Dateiendes oder bis zur Erkennung des angegebenen *dateiende*-Zeichens.

*xargs* wird beendet, wenn das aufgerufene *kommando* entweder den Ende-Status -1 liefert oder nicht ausgeführt werden kann. (Das Kommando sollte explizit mit *exit* und einem geeigneten Wert beendet werden, um einen zufälligen Rückgabewert von -1 zu vermeiden).

**BEISPIEL**

1. In der folgenden Shell-Prozedur werden alle Dateien im Dateiverzeichnis *$1* in das Dateiverzeichnis *$2* übertragen; zuvor wird vor der Durchführung jedes *move*-Kommandos eine Meldung ausgegeben:

```
ls $1 | xargs -i -t mv $1/{} $2/{}
```

2. In der folgenden Shell-Prozedur erfolgt die Ausgabe der geklammerten Kommandos in einer Zeile, die dann an das Ende der Datei *log* angehängt wird:

```
(logname; date; echo $0 $*) | xargs >>log
```

3. Im folgenden Shell-Prozedur werden aufeinanderfolgende Argumentenpaare, die ursprünglich als Shell-Argumente eingegeben wurden, abgearbeitet.

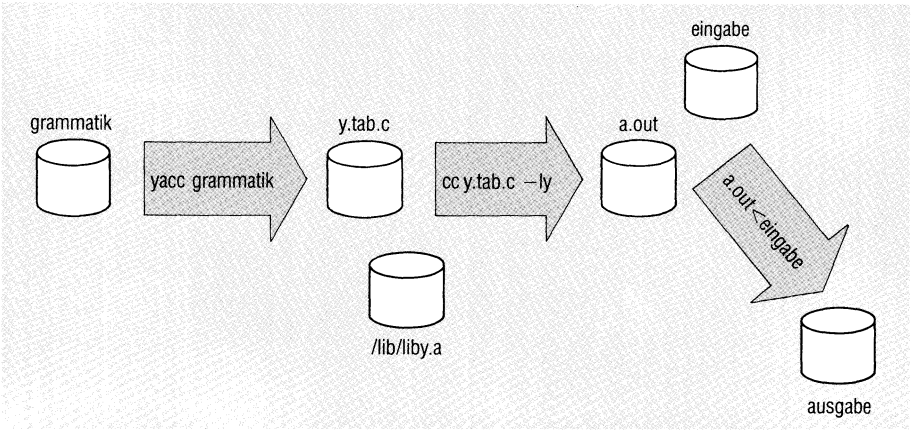
```
echo $* | xargs -n2 diff
```

**SIEHE AUCH**

*echo*(1)

## NAME

**yacc** - Parser generieren (yet another compiler-compiler)




## DEFINITION

**yacc**[␣schalter...]␣datei

## BESCHREIBUNG

*yacc* generiert C-Programme, die einen Eingabetext analysieren und bearbeiten. *yacc* kann u.a. den Parser für einen Übersetzer konstruieren. Zusammen mit einem *lex*-Programm für die lexikalische Analyse werden damit die wichtigsten Hilfsmittel bereitgestellt, die zum Übersetzerbau nötig sind.

### Wie arbeitet yacc?

*yacc* erwartet als Eingabe ein *yacc*-Quellprogramm, das in einem bestimmten Format geschrieben sein muß (siehe unten). Im *yacc*-Quellprogramm legen Sie fest:

- eine kontextfreie Grammatik (genauer: eine LALR(1)-Grammatik), die die Struktur der Eingabe(-sprache) beschreibt.
  Die Grammatik darf Mehrdeutigkeiten enthalten. Mit Präzedenzregeln können Sie Mehrdeutigkeiten auflösen.

- Aktionen, die ausgeführt werden, wenn der Parser, den *yacc* erzeugt, eine bestimmte Struktur findet.
- eine Funktion *yylex*, die die Eingabezeichen liest, nach syntaktischen Einheiten (Token) vorgruppiert und für die Strukturbestimmung vorbereitet.
  *yylex* führt also die lexikalische Analyse aus. *lex* kann diese Funktion erstellen.
  (Sie können die Funktion *yylex* auch beim Übersetzen des C-Quellprogramms, das *yacc* erstellt, dazubinden, anstatt sie im *yacc*-Quellprogramm zu definieren.)

*yacc* erstellt aus den Angaben im *yacc*-Quellprogramm ein C-Quellprogramm, in dem ein Parser (*yyparse*) definiert wird. Genauer gesagt, erzeugt *yacc* die Übergangstabellen (Parsertabellen) für einen Kellerautomaten, der nach einem LR(1)-Parsing-Algorithmus arbeitet.

*yacc* schreibt das C-Quellprogramm in die Datei *y.tab.c*.

Neben *yylex* müssen Sie noch folgende Funktionen im *yacc*-Quellprogramm definieren oder beim Übersetzen des C-Programms *y.tab.c* dazubinden:

– *main*, eine Hauptfunktion, die *yyparse* aufruft,

– *yyerror*, eine Funktion zur Fehlerbehandlung.

Einfache Versionen von *main* und *yyerror* stehen in der *yacc-Bibliothek /lib/liby.a*:

```
main()

yyerror(s)
char *s;
```

*main()* ruft lediglich *yyparse* auf.
*s* ist die Adresse einer Zeichenfolge, die den Fehler beschreibt.
*yyerror()* gibt einfach die Zeichenfolge *s* aus, wenn ein Syntaxfehler entdeckt wird.

Bei größeren Projekten ist es besser, eigene Versionen der Bibliotheksfunktionen zu verwenden.

Das C-Quellprogramm, das *yacc* generiert hat, sollten Sie mit der *yacc*-Bibliothek */lib/liby.a* übersetzen und laden; geben Sie also beim *cc*-Aufruf den Schalter *-ly* an:

```
$ cc y.tab.c -ly
```

**Wie arbeitet der Parser?**

Wenn das *yacc*-Programm *y.tab.c* und die oben genannten anderen Funktionen richtig übersetzt und gebunden wurden, dann steht in *a.out* ein ausführbares Programm. Das Programm behandelt einen Text, der auf der Standard-Eingabe bereitgestellt wird (den **Eingabestrom**), folgendermaßen:

- Das Hauptprogramm *main* ruft *yyparse* auf. *yyparse* ruft *yylex* auf.
- *yylex* liest so viele Zeichen des Eingabestroms ein, bis diese Zeichen eine Zeichenreihe bilden, die *yylex* erkennt.

  Die Zeichenreihen oder die Namen der Zeichenreihen, die *yylex* erkennt, sind die **Token**, **Terminalzeichen** oder **Terminalsymbole**. Ein Token kann der Name einer Zeichenreihe oder ein Einzelzeichen sein. Die Token stellen die Basiselemente der Eingabesprache dar. *yylex* meldet an *yyparse*, welches Terminalsymbol gefunden wurde bzw. ob ein Fehler aufgetreten ist.
- *yyparse* verarbeitet das Terminalsymbol, d.h. es sucht die Struktur, zu der das Terminalsymbol (eventuell zusammen mit vorher aufgetretenen Terminalsymbolen oder Strukturen) paßt und führt die Aktionen aus, die bei der gefundenen Struktur angegeben sind.

  Die Namen der Strukturen sind die **Nicht-Terminalzeichen**, **Nicht-Terminalsymbole** oder **syntaktischen Variablen**.

  Die Aktionen bestimmen auch, ob das Programm *a.out* eine Ausgabe erstellt und auf welchem Ausgabemedium ausgegeben wird.

  *yyparse* veranlaßt *yylex*, das nächste Terminalsymbol einzulesen.
- Wenn ein syntaktischer Fehler auftritt, wird er behandelt.
- Um das Ende eines Eingabestroms zu erkennen und dem Parser zu melden, muß *yylex* wissen, welches Terminalsymbol das "Ende der Eingabe" signalisiert. "Ende der Eingabe" kann z.B. "Dateiende" sein.

Ein Eingabestrom wird akzeptiert, wenn er (ohne das Token "Ende der Eingabe" mitzubehandeln) die Struktur des **Startsymbols** hat. Der Benutzer muß genau ein Startsymbol definieren.
Wird dem Parser "Ende der Eingabe" gemeldet, dann kehrt er zum Programm zurück, das ihn aufgerufen hat. In den meisten Fällen dürfte das die *main*-Funktion sein.

*yyparse* hat den return-Wert 0, wenn die Eingabe grammatikalisch richtig beendet und vom Parser akzeptiert wird. Falls ein Fehler in der Eingabe auftritt und nicht erfolgreich behoben werden kann, dann hat *yyparse* den return-Wert 1.

**lex und yacc**

Wenn Sie *lex* alleine verwenden, dann wird die Funktion *yylex*, die *lex* erstellt hat, von einer *main*-Funktion aufgerufen, die in der *lex*-Bibliothek steht. Wenn *lex* die lexikalische Analyse bei *yacc* übernimmt, dann erfolgt der Aufruf von *yylex* in dem Parser, den *yacc* erzeugt.

Bei Zusammenarbeit von *lex* und *yacc* sollte jede *lex*-Regel mit der Aktion

```
return(TOKEN);
```

enden und einen passenden Wert für *TOKEN* zurückgeben.

Das C-Programm *lex.yy.c*, das *lex* erstellt, übersetzen Sie am besten als Teil des Programms, das *yacc* erstellt. Dies geschieht, indem Sie im letzten Abschnitt eines *yacc*-Quellprogramms (Programmteil) die Zeile

```
#include "lex.yy.c"
```

einfügen.

Steht das *yacc*-Quellprogramm z.B. in der Datei *grammatik* und das *lex*-Quellprogramm in der Datei *regeln*, dann kann ein Aufruf so aussehen:

```
$ yacc grammatik
$ lex regeln
$ cc y.tab.c -ly -ll
```

Die *yacc*-Bibliothek muß vor der *lex*-Bibliothek gebunden werden, um ein Hauptprogramm zu haben, das den Parser aufruft.

Sie müssen das *lex*- und das *yacc*-Programm einzeln übersetzen und dann zusammenbinden, wenn die Programme zu groß sind, um zusammen übersetzt zu werden.

**Format eines yacc-Quellprogramms**

Ein *yacc*-Quellprogramm hat die folgende Form:

```
Deklarationen
%%
Grammatikregeln
%%
Programme
```

Die Deklarationen und Programme können entfallen. Geben Sie keine Programme an, dann dürfen Sie das zweite *%%* weglassen.

Deklarationen und Regeln müssen im entsprechenden Format geschrieben sein (siehe unten).

Ein Teil der Regeln kann auch die Fehlerbehandlung beschreiben, z.B. ob und wie ein Eingabestrom nach einem Fehler weiter eingelesen und behandelt werden soll.

Im Programmteil können Sie C-Funktionen definieren,

- die im *yacc*-Programm aufgerufen werden (z.B. im Aktionsteil, siehe unten),
- die Teilfunktionen eines Übersetzers sind (z.B. *yylex*),
- die als Hauptfunktion fungieren sollen (z.B. *main*).

Allgemein gilt:

- *yacc* ignoriert Leer-, Tabulator- und Neue-Zeile-Zeichen. Diese Zeichen dürfen jedoch nicht in Namen oder in aus mehreren Zeichen bestehenden, reservierten Symbolen enthalten sein.
- Kommentare dürfen überall dort vorkommen, wo auch Namen stehen dürfen (siehe unter "Deklarationen in yacc-Quell-Programmen" und "Regeln in yacc-Quell-Programmen").
  Die Kommentare müssen, wie in C, zwischen den Zeichen `/*` und `*/` stehen.
- Die Namen von Terminal- und Nicht-Terminalzeichen können beliebig lange Zeichenreihen sein. Sie dürfen aus Buchstaben, Ziffern, Punkten und dem Unterstrich _ bestehen. Sie dürfen aber nicht mit einer Ziffer beginnen. Zwischen Groß- und Kleinschreibung wird unterschieden. Namen, die mit *yy* beginnen, sollten Sie wegen der Verwechslungsgefahr mit den Parser-Symbolen vermeiden.

- Einzelzeichen müssen Sie in einfache Hochkommas einschließen, z.B. '+'. Auch C-Escapesequenzen sind zulässig (z.B. '\n').

- Namen, die *yacc* intern benutzt (z.B. *error* bei der Fehlerbehandlung) oder die in C reserviert sind (z.B. *if*, *while*), können zu Schwierigkeiten führen.
  Reservierte Namen als Token(namen) bereiten u.a. Probleme bei der lexikalischen Analyse und sollten vermieden werden.
  *yacc*-interne Namen sollten Sie nur so verwenden, wie *yacc* es erwartet.

**Deklarationsteil**

Im Deklarationsteil eines *yacc*-Quellprogramms können Sie folgende Anweisungen, Deklarationen und Definitionen angeben:

```
%{
text
%}
```

Der gesamte *text*, der zwischen Zeilen steht, die nur %{ und %} enthalten, wird (wie bei *lex*) in das C-Programm kopiert, das *yacc* erstellt. Dabei gilt der kopierte Text als global zu allen sonstigen Funktionen im C-Programm. Aktionen und Benutzerfunktionen können auf Funktionen und Variablen zugreifen, die hier definiert und deklariert werden.

Wie bei *lex* können Sie auf diese Weise Deklarationen, Definitionen, #include-Anweisungen usw. an das C-Programm übergeben. Die Begrenzungszeilen werden nicht kopiert.

**%token** NAME1 [nummer1] NAME2 [nummer2]...
*NAME1*, *NAME2* usw. sind Tokennamen, also Namen für Zeichenreihen, die *yylex* erkennen und deren Tokennummer *yylex* an den Parser übergeben soll.
*nummer1*, *nummer2* usw. sind nicht-negative ganze Zahlen.
*nummer1* bezeichnet die Tokennummer, die *NAME1* haben soll usw.

Alle Namen, die für Token stehen, müssen Sie so deklarieren. Jeder Name, der nicht im Deklarationsteil definiert wird, wird als Nicht-Terminalsymbol angesehen. Jedes Nicht-Terminalsymbol muß muß mindestens einmal auf der linken Seite einer Grammatikregel auftauchen.

Üblicherweise schreibt man bei *yacc* die Namen von Token in Großbuchstaben, die von Nicht-Terminalsymbolen in Kleinbuchstaben.

Die Tokennummer gibt an, was für ein Token eingelesen wurde. Wenn Sie im Deklarationsteil keine Tokennummern festlegen, dann teilt *yacc* die Tokennummern zu. *yacc* vergibt dann bei Einzelzeichen den numerischen Wert des Zeichens im jeweiligen Maschinenzeichensatz, Tokennamen erhalten Tokennummern, die bei 257 beginnen.
Die Tokennummer, die einem Token zugewiesen wird, muß beim Parser und bei *yylex* dieselbe sein. Die C-Anweisung *#define* ermöglicht, daß *yylex* die Tokennummern symbolisch zurückgibt. Die Tokennummern von verschiedenen Token müssen verschieden sein.

*Beispiel*

```
%token VORNAME 301 FAMILIENNAME 302 STRASSE HAUSNUMMER
POSTLEITZAHL ORT
```

Der Parser soll die Struktur einer Adresse in einem Eingabestrom erkennen. Der Scanner soll die Bestandteile der Adresse finden. Er liest jeweils soviele Zeichen des Eingabestroms, bis die Zeichenreihe zu einem der angegebenen Token paßt und gibt die entsprechende Tokennummer zurück. *VORNAME* und *FAMILIENNAME* haben Tokennummer 301 bzw. 302. *yacc* bestimmt die Tokennummern von den restlichen Token.

**%start** symbol
*symbol* ist der Name einer Struktur, also ein Nicht-Terminalsymbol.

Sie können genau ein Nicht-Terminalsymbol als Startsymbol deklarieren. Das Startsymbol beschreibt die größte, allgemeinste Struktur, die in der Grammatik vorkommt. Fehlt eine Start-Deklaration, dann nimmt *yacc* die linke Seite der ersten Grammatikregel (das ist der Name der ersten Struktur) als Startsymbol.

```
%left TOKEN...
%right TOKEN...
%nonassoc TOKEN...
```

*TOKEN* ist ein Tokenname oder ein Einzelzeichen.
Ein Einzelzeichen muß zwischen einfachen Hochkommas stehen, z.B. '+'.

Mit den drei Schlüsselwörtern *%left, %right* und *%nonassoc* definieren Sie Präzedenz- und Assoziativitätsregeln für Token (z.B. Operatoren). Die Definitionen schreiben Sie in mehrere Zeilen. Jede Zeile muß mit einem der Schlüsselwörter beginnen. Hinter die Schlüsselwörter schreiben Sie eine Liste von Token.
Token, die in derselben Zeile stehen, haben denselben Vorrang und dieselbe Assoziativität. Die Zeilen ordnen Sie nach aufsteigendem Vorrang.
*%left* bedeutet, daß die Operatoren in dieser Zeile linksassoziativ sind; *%right* bedeutet, daß sie rechts-assoziativ sind. *%nonassoc* bedeutet, daß sie nicht assoziativ sind.

Token, die Sie mit einem dieser drei Schlüsselwörter deklarieren, müssen Sie nicht zusätzlich mit *%token* deklarieren.

Für einstellige Operatoren müssen Sie i.a. einen Vorrang definieren. Wenn ein einstelliger und ein zweistelliger Operator (z.B. das Minuszeichen -) durch dasselbe Zeichen dargestellt werden, jedoch unterschiedlichen Vorrang erhalten müssen, verwenden Sie das Schlüsselwort *%prec* im Grammatikregelteil (siehe unten).

```
%union {
        unionkomponente...
        }
```

*unionkomponente* ist eine Komponentenvereinbarung, wie sie normalerweise in einem C-Quell-Programm vorkommt, also z.B.

```
double value;
```

Standardmäßig liefern die Aktionen und die Funktion *yylex* Integer-Werte zurück. Jedoch sind auch andere Typen (auch Strukturen) zulässig. Mit *%union*-Vereinbarungen können Sie die Typen der Elemente des Wertekellers festlegen.

Sie können die Union kann auch in einer Include-Datei deklarieren; in diesem Fall definieren Sie die Union in einer *typedef*-Anweisung mit der Variablen *YYSTYPE*. Die entsprechende *#include*-Anweisung schreiben Sie in den Deklarationsteil des *yacc*-Quellprogramms zwischen *%{* und *%}* (siehe oben).

Den Namen eines Token oder eines Nicht-Terminalzeichens verknüpfen Sie folgendermaßen mit dem Namen der Unionkomponente, von deren Typ das Token bzw. Nicht-Terminalzeichen sein

soll:
Der Name einer Unionkomponente sei z.B. *value*. Dann

- ist das Token *NAME* von demselben Typ wie *value*, wenn Sie im Deklarationsteil *NAME* mit

  ```
  %token␣<value>␣NAME        oder
  %left␣<value>␣NAME         oder
  %right␣<value>␣NAME        oder
  %nonassoc␣<value>␣NAME
  ```

  definieren.

- ist das Nicht-Terminalzeichen *symbol* von demselben Typ wie *value*, wenn Sie im Deklarationsteil

  **%type**␣ < value > ␣symbol

  angeben (siehe unten).

**%type**␣ < value > ␣symbol...
Das Nicht-Terminalzeichen (linke Seite einer Regel) *symbol* soll vom selben Typ wie *value* sein.

*value* ist der Name einer Unionkomponente, die Sie mit *%union* im Deklarationsteil vereinbart haben.

### Grammatikregeln

Die Grammatikregeln sind Produktionsregeln der Grammatik, die die Eingabesprache beschreibt. In diesen Grammatikregeln werden Namen für Terminalzeichen (Token) und Namen für Nicht-Terminalzeichen (syntaktische Variablen) in Beziehung gesetzt. Jede Regel beschreibt eine erlaubte Struktur und gibt dieser Struktur einen Namen (in diesem Fall also den Namen einer syntaktischen Variablen). Der Scanner *yylex* erkennt die Token im Eingabestrom und meldet sie dem Parser. Dieser stellt anhand der Grammatikregeln fest, welche Struktur vorliegt.

Bei den Regeln können Sie Aktionen angeben, die auszuführen sind, wenn die entsprechende Struktur oder ein Teil der Struktur im Eingabestrom erkannt wird.

Die Regeln und die Aktionen schreiben Sie jeweils in verschiedene Zeilen, um das *yacc*-Quellprogramm leichter lesen und ändern zu können.

Eine Regel hat das folgende Format:

```
symbol : rumpf;
```

*symbol* ist der Name der Struktur, die nach dem Doppelpunkt in der Regel steht. *symbol* ist also ein Name für ein Nicht-Terminalzeichen. *symbol* heißt auch "linke Seite der Regel".

*rumpf* besteht aus

- der Struktur, die *symbol* heißt, und aus
- Aktionen, die auszuführen sind, wenn die Struktur oder ein Teil der Struktur im Eingabestrom vorkommt.

Eine *Struktur* ist eine Folge, die aus beliebig vielen

- Tokennamen,
- Namen für Nicht-Terminalzeichen,
- Einzelzeichen des Zeichenvorrats

bestehen kann. Die Folge kann aber auch leer sein; das entsprechende Nicht-Terminalzeichen paßt dann zur leeren Zeichenfolge. Zwischen den einzelnen Namen bzw. Einzelzeichen, die eine Struktur bilden, können beliebig viele Leerzeichen stehen. Die Leerzeichen ignoriert *yacc*.

Die *Einzelzeichen* müssen zwischen einfachen Hochkommas stehen (z.B. '+'). Die Einzelzeichen sind Zeichen, die explizit im Eingabestrom vorkommen müssen und von *yylex* an den Parser übergeben werden. Sie sind also auch Token, haben aber keinen Namen und müssen nicht als Token im Deklarationsteil ausgewiesen werden.

Sie können eine Eingabezeichenreihe entweder als Token deklarieren; *yylex* meldet dann den Tokennamen (die Tokennummer), wenn die entsprechende Zeichenreihe im Eingabestrom vorkommt. Oder Sie beschreiben die Zeichenreihe als Folge von Einzelzeichen im Rumpf einer Regel. *yylex* übergibt die Einzelzeichen (genauer die Tokennummer der Einzelzeichen) an den Parser.

Das ASCII-Zeichen NUL (0 oder '\0') dürfen Sie nicht in Regeln verwenden, da die Tokennummer 0 für das Token "Ende der Eingabe" reserviert ist.

Haben mehrere Regeln dieselbe linke Seite, dann müssen Sie die linke Seite nicht jedes Mal wiederholen. Stattdessen können Sie das Wiederholungszeichen | (senkrechter Strich) verwenden:

```
symbol : rumpf1
       | rumpf2
       ...
       | rumpfn
       ;
```

Vor dem Wiederholungszeichen | darf kein Strichpunkt stehen, nur nach der letzten Regel.

Um *yacc*-Quellprogramme leichter lesen und ändern zu können, sollten Sie

- alle Regeln mit derselben linken Seite nacheinander auflisten,
- die linke Seite nur einmal angeben und
- den Strichpunkt in eine extra Zeile am Ende schreiben.

**Aktionen**

Bei jeder Regel können Sie Aktionen angeben, die auszuführen sind, wenn die Struktur (oder ein Teil der Struktur) der entsprechenden Regel im Eingabestrom erkannt wird.

Die Aktionen sind C-Programmteile. Sie können Werte zurückgeben und Werte von früheren Aktionen entgegennehmen. Falls gewünscht, kann *yylex* auch Werte für Token zurückgeben.

Eine Aktion besteht aus beliebigen C-Anweisungen, die Ein- und Ausgabe ausführen, Unterprogramme aufrufen und externe Variablen ändern können. Sie müssen die Aktionen in geschweifte Klammern schreiben:

```
{ C-Programmteil }
```

Definitionen und Deklarationen, die global für alle Aktionen gelten sollen, können Sie im Deklarationsteil angeben (siehe oben).

Aktionen müssen nicht am Ende einer Regel stehen. Sie können sie auch zwischen Struktur-Komponenten einfügen. Sie werden dann ausgeführt, bevor die Struktur ganz vom Parser bearbeitet wird.

Da der Parser, den *yacc* generiert, nur Variable benutzt, die mit *yy* beginnen, sollten Sie solche Variablen- und Funktionsnamen vermeiden oder so verwenden, wie *yacc* es erwartet.

*Werte*

Um die Kommunikation zwischen den Aktionen, *yylex* und dem Parser zu erleichtern, gibt es Pseudovariable, die mit dem Zeichen *$* beginnen und bestimmte Werte an den Parser zurückgeben.

In einer Regel haben sowohl die linke Seite als auch die Strukturkomponenten und Aktionen auf der rechten Seite einen Wert. Normalerweise ist der Wert eine ganze Zahl. Für diese Werte gilt:

- Der Wert einer Aktion ist
  - der Wert, der innerhalb derselben Aktion an die Pseudovariable *$$* übergeben wird,
  - 0, wenn in der Aktion der Variablen *$$* kein Wert zugewiesen wird.
- Der Wert eines Token (nicht zu verwechseln mit der Tokennummer) ist
  - der Wert, den die externe Variable *yylval* hat, wenn *yylex* die Tokennummer an den Parser übergibt.

    *yylval* hat den Anfangswert 0 und behält einen festen Wert, bis ein anderer Wert an *yylval* zugewiesen wird.
    Wenn *yylex* ein Token im Eingabestrom erkennt und (z.B. in einer Aktion im *lex*-Quellprogramm) an *yylval* einen Wert zuweist, dann hat das Token diesen Wert.
- Der Wert einer syntaktischen Variablen (der linken Seite einer Regel) ist
  - der Wert von *$$*, wenn auf der rechten Seite als letztes eine Aktion steht, in der an *$$* ein Wert zugewiesen wird,
  - der Wert von *$1* sonst.

*Die Bedeutung der Pseudovariablen*

Eine Aktion kann an die Pseudovariable *$$* einen Wert zuweisen. *$$* hat dann innerhalb der Aktion diesen Wert.
Wenn innerhalb einer Aktion kein Wert an *$$* zugewiesen wird, dann hat *$$* innerhalb der Aktion

- den Wert von *$1*, wenn die Aktion als letztes auf der rechten Seite steht,
- den Wert 0, sonst.

Innerhalb einer Regel kann auf die Werte von früheren Aktionen und Strukturkomponenten derselben Regel zugegriffen werden. *$1* hat den Wert der Komponente oder der Aktion, die als erste rechts vom Doppelpunkt steht, *$2* den Wert der nächsten Komponente oder Aktion usw.

*Beispiel*

1. Bei der Regel

```
a : B C D
                {$$ = $2;}
                ;
```

haben die Pseudovariablen folgende Werte:

$$ hat den Wert von $2 (den Wert von C),
$1 hat den Wert von B,
$2 hat den Wert von C,
$3 hat den Wert von D,
$4 hat den Wert der Aktion, also von $$.

Die Token *B*, *C*, und *D* haben den Wert, den die Variable *yylval* hat, wenn *yylex* die jeweilige Tokennummer an den Parser übergibt.

Die syntaktische Variable *a* hat den Wert von *$$*.

Würde die Regel

```
a : B C D
          ;
```

lauten, dann hätte *a* den Wert von *$1*, also von *B*.

2. Wenn Sie folgende Regel im *yacc*-Quellprogramm angeben:

```
a : '(' B ')'
            { printf("Hallo!\n"); }
```

dann wird auf der Standard-Ausgabe

```
Hallo!
```

ausgegeben, sobald der Parser die Struktur '(' *B* ')' erkennt.

$1 hat den Wert von '(',
$2 hat den Wert von *B*,
$3 hat den Wert von ')',
$4 hat den Wert der Aktion, also 0, da keine Zuweisung an $$ erfolgt,
*a* hat den Wert von $1.

*Fehlerbehandlung und spezielle Anweisungen*

Der Tokenname *error* ist für die Fehlerbehandlung reserviert. Diesen Namen können Sie in Grammatikregeln verwenden; er gibt dann Informationen über Stellen aus, an denen Fehler zu erwarten sind, die gelöst werden könnten. Liegt ein Fehler vor, so verhält der Parser sich so, als ob das Token *error* das aktuelle Vorausschau-Token wäre und führt die Aktion aus, die bei der Regel mit *error* angegeben ist. Das Vorausschau-Token wird dann auf das Token zurückgesetzt, das den Fehler verursacht hat. Geben Sie keine speziellen *error*-Regeln an, so bricht der Parser nach dem ersten Fehler ab.

Um zu verhindern, daß mehrere Fehlermeldungen dargestellt werden, verbleibt der Parser bei Auftreten eines Fehlers im Fehlermodus, bis drei Token erfolgreich eingelesen und verarbeitet worden sind. Befindet sich der Parser beim Auftreten eines Fehlers bereits im Fehlermodus, so wird keine Meldung ausgegeben, und das Eingabe-Token wird stillschweigend gelöscht.

Ist in einer Aktion die Anweisung

```
yyerrok;
```

enthalten, so wird der Parser wieder in den Normalmodus zurückgesetzt. Sie können diese Anweisung immer dann verwenden, wenn der Parser den Eindruck haben soll, daß ein Fehler vollständig behoben worden ist.

Ist in einer Aktion die Anweisung

```
yyclearin;
```

enthalten, so wird das vorhergehende Vorausschau-Token gelöscht. Sie können diese Anweisung immer dann verwenden, wenn Sie mit Hilfe einer Benutzerroutine die Stelle im Eingabestrom festlegen wollen, an der fortgefahren werden soll.

**Auflösung von Mehrdeutigkeiten**

Zur Auflösung von Mehrdeutigkeiten gibt es folgende interne Regeln:

1. Bei einem lies/reduziere-Konflikt wählt der Parser die lies-Aktion.
2. Bei einem reduziere/reduziere-Konflikt wählt der Parser die Reduktion, die im *yacc*-Quellprogramm als erstes angegeben ist.

Außerdem werden zur Auflösung von Konflikten die deklarierten Präzedenz- und Assoziativitätsregeln (siehe Abschnitt *Deklarationsteil*) folgendermaßen benutzt:

1. Jeder Grammatikregel ist eine Präzedenz und Assoziativität zugeordnet (Standard). Mit dem Schlüsselwort *%prec* können Sie diesen Standard aufheben. Für einige Grammatikregeln gibt es möglicherweise keinen Vorrang und keine Assoziativität.
2. Tritt ein reduziere/reduziere-Konflikt oder ein lies/reduziere-Konflikt auf, bei dem entweder das Eingabesymbol oder die Grammatikregel keinen Vorrang und keine Assoziativität haben, so werden die beiden obengenannten Regeln benutzt.
3. Wenn ein lies/reduziere-Konflikt auftritt und sowohl die Grammatikregel als auch das Eingabezeichen einen Vorrang und eine Assoziativität haben, so führt der Parser die Aktion mit der höchsten Priorität aus. Bei gleicher Priorität führt der Parser eine Aktion in Abhängigkeit von der Assoziativität aus; bei Links-Assoziativität

wird eine reduziere-, bei Rechts-Assoziativität eine lies-Aktion durchgeführt; eine Nicht-Assoziativität wird als Fehler interpretiert.

Konflikte, die mit dem Vorrang gelöst werden, werden bei der Meldung von lies/reduziere- und reduziere/reduziere-Konflikten unterdrückt.

**Programme**

Im Programm-Teil kann die Definition der Funktion zur lexikalischen Analyse des Eingabestroms *yylex* sowie anderer Funktionen enthalten sein. Hierzu können z.B. Funktionen gehören, die in den Aktionen verwendet werden (siehe Abschnitt *Grammatikregeln*).

*yylex* ist eine Integer-Funktion, die die Tokennummer mit einer return-Anweisung zurückgibt. Die Tokennummer repräsentiert ein Token eindeutig.
Soll *yylex* außer der Tokennummer noch einen Wert zurückgeben, dann müssen Sie diesen Wert der externen Variablen *yylval* zuweisen. Die Zuweisung muß von *yylex* ausgeführt werden. Sie geben z.B. in einer Aktion im *lex*-Quellprogramm an, welcher Wert mit *yylval* jeweils zurückgegeben werden soll.
*yylval* ist mit 0 vorbelegt und behält einen festen Wert, bis ein anderer zugewiesen wird. Der Wert eines Token ist der Wert, den *yylval* hat, wenn *yylex* an den Parser die Tokennummer übergibt (siehe Abschnitt *Aktionen*).

Das Ende des Eingabestroms ist durch das spezielle Token *endmarker* (Ende der Eingabe) gekennzeichnet. Dieses Token muß die Tokennummer 0 oder eine negative Zahl haben. Alle Funktionen zur lexikalischen Analyse sollten beim Ende ihres Eingabestroms den Wert 0 oder einen negativen Wert als Tokennummer zurückgeben. Stellt das Token bis zum *endmarker* (jedoch ohne *endmarker*) eine Struktur dar, die zum Startsymbol paßt, so akzeptiert der Parser den Eingabestrom. In jedem anderen Kontext führt der *endmarker* zu einer Fehlermeldung.

## SCHALTER

**-v**

Zusätzlich zum C-Programm *y.tab.c*, gibt *yacc* in der Datei *y.output* eine Beschreibung der Übergangs-Tabellen des Parsers (Parser-Tabellen) aus. Dabei berichtet *yacc* auch über alle Mehrdeutigkeiten, die in der Grammatik auftreten.

**-d**

*yacc* erstellt eine Datei *y.tab.h*, die *#define*-Anweisungen enthält. In den *#define*-Anweisungen werden die Token mit ihrer Tokennummer, die Sie oder *yacc* definiert haben, in Verbindung gesetzt. Damit können auch andere Quellcode-Dateien als *y.tab.c* auf die Tokennummern zugreifen.

**-l**

Das Programm in *y.tab.c* enthält keine *#line*-Anweisungen. Sie sollten diesen Schalter nur dann angeben, wenn die Fehlersuche in Bezug auf die Grammatik und die zugehörigen Aktionen abgeschlossen ist.

**-t**

Debugging-Code zur Suche von Laufzeitfehlern wird beim Übersetzen von *y.tab.c* mitübersetzt.

Dieser Debugging-Code wird immer in *y.tab.c* erzeugt, aber nur bedingt übersetzt. Die Übersetzung des Debugging-Code wird durch das Präprozessor-Symbol *YYDEBUG* gesteuert. Hat *YYDEBUG* einen Wert ungleich 0, dann wird der Debugging-Code mitübersetzt; hat es den Wert 0, dann wird er nicht übersetzt.
Ein Programm, das Debugging-Code enthält, ist umfangreicher und etwas langsamer.

## OPERANDEN

datei

*datei* ist der Name der Datei, in der das *yacc*-Quellprogramm steht. Nach den Angaben im *yacc*-Quellprogramm schreibt *yacc* ein C-Programm in die Datei *y.tab.c*. In dem C-Programm (*yacc*-Programm) wird u.a. eine Funktion *yyparse* definiert.

## FEHLER

Die Anzahl der reduziere/reduziere- und lies/reduziere-Konflikte wird auf der Standard-Fehlerausgabe ausgegeben. Ausführlichere Informationen sind in der Datei *y.output* enthalten. Auch wenn einige Regeln nicht vom Startsymbol erreicht werden können, werden Fehlermeldungen ausgegeben.

## DATEIEN

*y.output*
enthält eine Beschreibung der Parser-Tabellen und der Konflikte (siehe Schalter *-v*).

*y.tab.c*
enthält das Programm, das *yacc* generiert.

*y.tab.h*
enthält Definitionen von Tokennummern für die Token (siehe Schalter *-d*).

*/lib/liby.a*
*yacc*-Bibliothek

## BEISPIEL

1. *yacc* aufrufen

   In der Datei *grammatik* stehe ein *yacc*-Quellprogramm. *yylex* wird im *yacc*-Quellprogramm definiert. Sie verwenden die einfachen Versionen von *main* und *yyerror* in der *yacc*-Bibliothek. Der Eingabestrom steht in der Datei *eingabe*. Dann geben Sie ein:

   ```
   $ yacc grammatik
   $ cc y.tab.c -ly
   $ a.out < eingabe
   ```

2. Ein *yacc*-Quellprogrammteil, der die Struktur des Datums beschreibt:

   Die folgenden Grammatikregeln erkennen bestimmte Datumsangaben in der Eingabesprache, wie z.B. "5.7.1973" oder "7.Februar1930":

   ```
   %token JAHR
   %start datum
   %%
   datum  :  tag '.' zmonat '.' JAHR
   ```

```
        |  tag '.' wmonat JAHR
        ;

tag     :  '1'
        |  '2'
        ...
        |  '3' '1'
        ;

zmonat  :  '1'
        |  '2'
        ...
        |  '1' '2'
        ;

wmonat  :  'J' 'a' 'n' 'u' 'a' 'r'
        |  'F' 'e' 'b' 'r' 'u' 'a' 'r'
        ...
        |  'D' 'e' 'z' 'e' 'm' 'b' 'e' 'r'
        ;
```

Im Deklarationsteil wird *JAHR* als Terminalsymbol deklariert, *datum* als Startsymbol. *yylex*, der Scanner, übergibt an den Parser die Meldung, welches Terminalsymbol er findet. *tag* ist ein Nicht-Terminalsymbol, und *yylex* übergibt die einzelnen Ziffern als Einzelzeichen an den Parser. Sie können die Regeln noch so erweitern, daß auch geprüft wird, ob ein bestimmter Tag überhaupt existiert, wie z.B. der 30. Februar 1978.

Es liegt bei Ihnen, ob Sie z.B. auch *tag*, *zmonat* und *wmonat* als Token deklarieren möchten. Der Scanner übernimmt dann Aufgaben des Parsers.

3. Grammatikregeln mit Pseudovariablen

   In der Datei *yacc.b* steht das folgende *yacc*-Quellprogramm:

```
%token B C
%%
b : a C
        { printf("%d %d\n", $1, $2);}

a : '('
        { $$ = 5;}
    B
        { printf("%d\n", $$);}
  ')'
        { printf("%d %d %d %d %d %d\n", $1, $2, $3, $4, $5, $$);}
    B
        { printf("%d %d %d\n", $6, $7, $$);}
%%
#include "lex.yy.c"
```

In der Datei *lex.b* steht das folgende *lex-Quellprogramm*:

```
extern int yylval;

%%

B       return(B);

"("     {yylval = 9; return('(');}

")"     {yylval = 7; return(')');}

C       return(C);

.       ;

\n      ;
```

Geben Sie die folgenden Kommandos ein:

```
$ lex lex.b
$ yacc yacc.b
$ cc y.tab.c -ly -ll
$ a.out
```

und geben Sie dem Programm in *a.out* die Zeichenreihe

```
(B)BC
```

als Eingabe, dann erhalten Sie die folgende Ausgabe:

```
0
9 5 9 0 7 0
0 7 9
9 7
```

In der ersten Zeile steht der Wert, den *$$* bei der zweiten Aktion von *a* annimmt. In der zweiten Zeile stehen die Werte von *$1* bis *$5* und der Wert von *$$*, den *$$* in der dritten Aktion von *a* hat. Die nächste Zeile zeigt die Werte von *$6*, *$7* und von *$$* bei der letzten Aktion. In der letzten Zeile stehen die Werte der Variablen *$1* und *$2* innerhalb der Struktur *b*.

4. Beispiel für Präzedenz- und Assoziativitätsregeln

   Die folgenden Grammatikregeln beschreiben arithmetische Ausdrücke, in denen

   – die zweistelligen Operatoren * und / Vorrang vor den zweistelligen Operatoren + und - haben,

- die vier Operatoren +, -, * und / linksassoziativ sind,
- die Operatoren +, -, * und / Vorrang vor dem Operator = haben,
- der Operator = rechtsassoziativ ist,
- der einstellige Operator - denselben Vorrang wie * haben soll.

```
%token NAME
%right  '='
%left   '+' '-'
%left   '*' '/'

%%
ausdruck  :  ausdruck '=' ausdruck
          |  ausdruck '+' ausdruck
          |  ausdruck '-' ausdruck
          |  ausdruck '*' ausdruck
          |  ausdruck '/' ausdruck
          |  '-' ausdruck %prec '*'
          |  NAME
          ;
```

Die Eingabezeile

```
a = b = c*d - e - f*g
```

wird vom Parser so strukturiert:

```
a = ( b = (((c*d) - e) - (f*g)))
```

5. Ein Eingabestrom soll nach folgenden Kriterien untersucht werden:

   Anzahl der Zeilen,
   der Worte,
   der Zahlen,
   der Operatoren,
   der sonstigen Zeichen.

Ein *yacc*-Quellprogramm für diese Aufgabe steht in der Datei *grammatik*:

```
%{
#include <stdio.h>
long lcounter ;
long wcounter ;
long ncounter ;
long scounter ;
long opcounter ;
%}

%token WORT ZAHL 301 WSPACE 302 OPERATOR NEWLINE OTHER

%%

text        :  /* leer */
            |  text zeile zeilenende
            ;

zeile       :  /* leer */
                                {
                                  printf("Zeile %ld: ", ++lcounter);
                                }
            |  zeile WORT
                                {
                                  printf("%s", yytext);
                                  ++wcounter;
                                }
            |  zeile ZAHL
                                {
                                  printf("%s", yytext);
                                  ++ncounter;
                                }
            |  zeile OPERATOR
                                {
                                  printf("%s", yytext);
                                  ++opcounter;
                                }
            |  zeile WSPACE
                                {
                                  printf("%s", yytext);
                                }
            |  zeile OTHER
                                {
                                  printf("%s", yytext);
                                  ++scounter;
                                }
            ;

zeilenende  :  NEWLINE
                                {
                                   printf("%s", yytext);
                                }
            ;
%%
#include "lex.yy.c"
```

Das *lex*-Quellprogramm in der Datei *regeln* lautet:

```
%%
[a-zA-Z][a-zA-Z0-9]*        return (WORT);
[0-9]+                      return (ZAHL);
[ \t]+                      return (WSPACE);
[-+*/%=]                    return (OPERATOR);
[\n]                        return (NEWLINE);
```

```
[^-+*/%=0-9a-zA-Z\t \n]+   return (OTHER);
%%
yywrap()
{
    printf("Ihr Text hat %ld Zeile(n)\n",lcounter);
    printf("             %ld Wort(e)\n",wcounter);
    printf("             %ld Zahl(en)\n",ncounter);
    printf("             %ld Operator(en)\n",opcounter);
    printf("             %ld sonstige(s) Zeichen\n",scounter);
    return(1);
}
```

In der Datei *text* steht folgender Text:

```
7 kleine Negerlein,
die gingen zu der Hex'.
Eines hat sie aufgefressen -
da waren's nur noch 6.
```

Die Kommandos

```
$ yacc grammatik
$ lex regeln
$ cc y.tab.c -ly -ll
$ a.out < text
```

führen zu der Ausgabe

```
Zeile 1: 7 kleine Negerlein,
Zeile 2: die gingen zu der Hex'.
Zeile 3: Eines hat sie aufgefressen -
Zeile 4: da waren's nur noch 6.
Ihr Text hat 4 Zeile(n)
             16 Wort(e)
             2 Zahl(en)
             1 Operator(en)
             4 sonstige(s) Zeichen
```

6. Ein Tischrechner

Es soll ein einfacher Tischrechner realisiert werden. Er hat 26 Register, die *a, b,..., z* heißen. Der Tischrechner kann die Operationen +, -, *, /, %, & (bit-weise Konjunktion), | (bit-weise Disjunktion) und Zuweisung ausführen. Er kann ganze Zahlen behandeln, die Sie auch oktal angeben können (mit 0 davor).

Das Beispiel zeigt, wie Präzedenzen behandelt werden und eine einfache Fehlerbehandlung erfolgt. Das Ergebnis von Ausdrücken wird auf der Standard-Ausgabe aufgezeigt. Nur Anweisungen werden nicht ausgedruckt. Die lexikalische Analyse ist sehr einfach realisiert.

In der Datei *grammatik* steht folgendes *yacc*-Quellprogramm:

```
%{
#include <stdio.h>
#include <ctype.h>

int rregister[26];
int basis;
%}

%start liste

%token ZIFFER BUCHSTABE

%left '|'
%left '&'
%left '+' '-'
%left '*' '/' '%'
%left EMINUS

%%                                  /* Regeln */

liste       : /* leer */
            | liste  anweisung '\n'
            | liste error '\n'
                  { yyerrok; }
            ;

anweisung   :  ausdruck
                  { printf("%d\n", $1); }
            |  BUCHSTABE '=' ausdruck
                  { rregister[$1] = $3; }
            ;

ausdruck    :  '(' ausdruck ')'
                  { $$ = $2; }
            |  ausdruck '+' ausdruck
                  { $$ = $1 + $3; }
            |  ausdruck '-' ausdruck
                  { $$ = $1 - $3; }
            |  ausdruck '*' ausdruck
                  { $$ = $1 * $3; }
            |  ausdruck '/' ausdruck
                  { $$ = $1 / $3; }
            |  ausdruck '%' ausdruck
                  { $$ = $1 % $3; }
            |  ausdruck '&' ausdruck
                  { $$ = $1 & $3; }
            |  ausdruck '|' ausdruck
                  { $$ = $1 | $3; }
            |  '-' ausdruck %prec EMINUS
                  { $$ = - $2; }
            |  BUCHSTABE
                  { $$ = rregister[$1]; }
            |  zahl
            ;

zahl        :  ZIFFER
                  { $$ = $1; basis = ($1 == 0)? 8 : 10; }
            |  zahl ZIFFER
```

```
                    { $$ = basis * $1 + $2; }
            ;

%%

yylex()     {

            int c;

            while ((c = getchar()) == ' ')
                ;
            if ( islower (c)) {
                 yylval = c - 'a';
                 return (BUCHSTABE);
                 }
            if ( isdigit (c)) {
                 yylval = c - '0';
                 return (ZIFFER);
                 }
            return (c);
          }
```

Nach den Kommandos

```
$ yacc grammatik
$ cc y.tab.c -ly
```

steht der ablauffähige Tischrechner in der Datei *a.out*.

```
$ a.out
a = 8*9 - (6-7)
a
73
5+7 - 8*9
-60
b = 6
c = a+b
c
79
f = 0345
f
229
$
```

**SIEHE AUCH**

*cc*(1D), *lex*(1D).

# A Anhang

## A.1 Reguläre Ausdrücke

Ein *regulärer Ausdruck* (rA) beschreibt ein Muster, das für eine Menge von Zeichenfolgen steht. Jede dieser Zeichenfolgen *paßt* zu dem regulären Ausdruck.
Reguläre Ausdrücke werden dazu verwendet, Texte nach Zeichenfolgen zu durchsuchen und die gefundenen Zeichenfolgen evt. zu verarbeiten.

Ein regulärer Ausdruck besteht aus einer Folge von einfachen Zeichen und Metazeichen. Ein einfaches Zeichen steht für das ihm entsprechende ASCII-Zeichen, ein Metazeichen hat eine andere definierte Bedeutung.

Neben den (einfachen) regulären Ausdrücken gibt es *erweiterte reguläre Ausdrücke*.

*Einfache reguläre Ausdrücke* werden verarbeitet von den Kommandos: *ed ex grep pg vi csplit expr sed*

*Erweiterte reguläre Ausdrücke* werden verarbeitet von den Kommandos: *awk egrep lex*

## A.1.1 Einfache Reguläre Ausdrücke

| | Syntax | Bedeutung | Beispiel | |
|---|---|---|---|---|
| | Ein regulärer Ausdruck ist: | Der linksstehende reguläre Ausdruck steht für: | regulärer Ausdruck | passende Zeichenfolge(n) |
| 1 | c | c<br>wobei c jedes Zeichen sein kann außer: \ ^<br>$ . / [ ] * + - | a | a |
| 2 | \c | c<br>wobei c jedes Zeichen sein kann außer:<br>( ) 1 2 3 4 5 6 7 8 9 0 | \a<br>* | a<br>* |
| 3 | . | ein beliebiges Zeichen | . | m oder * oder 3 usw. |
| 4a | [s] | ein Zeichen aus s, wobei s eine Zeichenfolge ist, in der das Zeichen ] nur als erstes Zeichen auftreten darf.<br>In s ist das Zeichen \ kein Metazeichen. | [mz] | m oder z |
| | [a-b] | ein Zeichen aus dem Bereich a bis b, wobei nach der ASCII-Tabelle a<b sein muß. | [a-d] | a oder b oder c oder d |
| 4b | [^s] | ein Zeichen, das nicht in s vorkommt, wobei s eine Zeichenfolge ist, in der das Zeichen ] nur als erstes Zeichen auftreten darf.<br>In s ist das Zeichen \ kein Metazeichen. | [^\.5a&] | jedes Zeichen außer:<br>\ . 5 a & |
| | [^a-b] | ein Zeichen, das nicht im Bereich a-b vorkommt wobei nach der ASCII-Tabelle a<b sein muß. | [^a-z] | jedes Zeichen außer Kleinbuchstaben |
| 5 | r* | null, eine oder mehrere zu r passende Zeichenfolgen, wobei r ein rA der Form 1-4 ist. | p* | Leerzeichen oder p oder pp oder ppp usw. |
| 6 | xy | eine zu x passende Zeichenfolge, gefolgt von einer zu y passenden Zeichenfolge, wobei x und y rA sind. | .1<br>[124]r | zl oder 4l oder &l usw<br>1r oder 2r oder 4r |

| | Syntax | Bedeutung | Beispiel | |
|---|---|---|---|---|
| | Ein regulärer Ausdruck ist: | Der linksstehende reguläre Ausdruck steht für: | regulärer Ausdruck | passende Zeichenfolge(n) |
| 7a | ^r | am Zeilenanfang eine zu r passende Zeichenfolge, wobei r ein rA der Form 1-9 ist. | ^[aA]pfel | am Zeilenanfang: apfel oder Apfel |
| 7b | r$ | am Zeilenende eine zu r passende Zeichenfolge, wobei r ein rA der Form 1-9 ist. | birne$ | am Zeilenende: birne |
| 8a | r\{m,n\} | mindestens m, höchstens n Wiederholungen einer zu r passenden Zeichenfolge. | heu\{2,4\} | heu heu oder heu heu heu oder heu heu heu heu |
| 8b | r\{m\} | genau m Wiederholungen einer zu r passenden Zeichenfolge. | heu\{3\} | heu heu heu |
| 8c | r\{m,\} | mindestens m Wiederholungen einer zu r passenden Zeichenfolge. | heu\{3,\} | heu heu heu oder heu heu heu heu ... |
| 9a | \(r\) | eine zu r passende Zeichenfolge, wobei r ein rA ist, in dem höchstens viermal die Zeichenfolge \( vorkommt.<br><br>Die öffnenden \( in einem rA werden mit 1 beginnend von links nach rechts durchnumeriert (siehe 9b). | \(Haus\)<br><br>\(haus\(tür\)\) | Haus<br><br>haustür |
| 9b | x\n | eine zu r passende Zeichenfolge, wobei r ein mit dem n-ten \( \)-Paar geklammerter Teilausdruck des rA x ist. Für die Anzahl n der \( \)-Paare gilt: $1 \leq n \leq 5$. | \(haus\(tür\)\)\2 | tür |

## A.1.2 Erweiterte Reguläre Ausdrücke

| | Syntax | Bedeutung | Beispiel | |
|---|---|---|---|---|
| | Ein regulärer Ausdruck ist: | Der linksstehende reguläre Ausdruck steht für: | regulärer Ausdruck | passende Zeichenfolge(n) |
| 10a | r+ | eine oder mehrere zu r passende Zeichenfolgen, wobei r ein rA ist. | (ok)+ | ok oder okokok usw. |
| 10b | r? | null oder eine zu r passende Zeichenfolge, wobei r ein rA ist. | (ok)? | Leerzeichen oder ok |
| 11 | u\|v | eine zu u oder zu v passende Zeichenfolge, wobei u und v reguläre Ausdrücke sind. | tür(schloß\|riegel) | türschloß oder türriegel |
| 12 | (r) | eine zu r passende Zeichenfolge, wobei r ein rA ist. | (aber)<br>(bla)* | aber<br>Leerzeichen oder bla oder blabla usw. |

### A.1.3 Prioritäten

Die folgende Tabelle zeigt die Priorität der Operatoren in regulären Ausdrücken:

| Operator | Priorität |
|---|---|
| [...] | höchste Priorität |
| *? + | . |
| Verkettung | . |
| \| | niedrigste Priorität |

## A.2 Sonderzeichen der Shell

Die folgenden Zeichen sind Sonderzeichen der Shell. Sie haben für die Shell eine besondere Bedeutung.

### Steuerzeichen

**;** Kommandos, die Sie in einer Zeile durch ein ; (Strichpunkt) getrennt eingeben, werden nacheinander ausgeführt, z.B. `cd dir; pwd`

**&** Wenn Sie eine Kommandoeingabe mit dem Zeichen & beenden, wird das Kommando als Hintergrundprozeß gestartet; z.B. `cc pgm.c&`

**|** Pipezeichen. Kommandos, die Sie jeweils durch ein Pipezeichen getrennt eingeben, bilden eine Pipeline: Die Standard-Ausgabe des Kommandos vor einem Pipezeichen bildet die Standard-Eingabe für das Kommando hinter dem Pipezeichen.

*Beispiel*

```
cat datei1 datei2 | grep muenchen | sort
```

**&&** Trennen Sie zwei Kommandolisten durch &&, dann wird die zweite Kommandoliste nur dann ausgeführt, wenn die erste Kommandoliste den Endestatus 0 liefert.

*Beispiel*

```
cc prog.c && a.out
```

**||** Trennen Sie zwei Kommandolisten durch ||, dann wird die zweite Kommandoliste nur dann ausgeführt, wenn die erste Kommandoliste einen Endestatus ungleich 0 liefert.

**(...)**

Für die Kommandoliste zwischen den runden Klammern wird eine Subshell erzeugt.

*Beispiel*

```
name=nepomuk;  (name=edel;  echo  $name);
echo $name
```

{␣...;}

Die Kommandos, die in den geschweiften Klammern stehen, werden zusammengefaßt. Auf die Klammern folgende Pipeverbindungen oder Ein-/Ausgabeumlenkungen gelten für alle Kommandos, die in den Klammern stehen.
Beachten Sie das Leerzeichen nach der öffnenden Klammer! Vor der schließenden Klammer muß ein Strichpunkt oder ein Neue-Zeile-Zeichen stehen.

*Beispiel*

```
{ ls -l; ls -l . .; } | grep meier
```

name() {␣...;}

Der Funktion *name* wird die Kommandofolge zugeordnet, die in den geschweiften Klammern steht. Soll die Kommandofolge ausgeführt werden, müssen Sie dann nur noch *name* eingeben.
Beachten Sie das Leerzeichen nach der öffnenden geschweiften Klammer! Vor der schließenden geschweiften Klammer muß ein Strichpunkt oder ein Neue-Zeile-Zeichen stehen. Geben Sie nur ein einziges Kommando an, dann können Sie die geschweiften Klammern und den Strichpunkt weglassen.

*Beispiel*

```
wo() { who am i; pwd; }
```

# kommentar

Kommentarzeichen. Die Shell ignoriert alle Zeichen, die zwischen dem Kommentarzeichen und dem nächsten Neue-Zeile-Zeichen stehen.

Neue-Zeile-Zeichen

Mit einem Neue-Zeile-Zeichen schließen Sie eine Eingabe ab.

Leerzeichen
Tabulatorzeichen

Mit Leer- und Tabulatorzeichen trennen Sie beim Aufruf eines Kommandos den Kommandonamen und die übergebenen Argumente voneinander.

### Generierung von Dateinamen

`*` steht für eine beliebige Zeichenfolge, auch für die leere Zeichenfolge.

`?` steht für ein beliebiges einzelnes Zeichen.

`[...]`
steht für ein beliebiges der in den Klammern eingeschlossenen Zeichen. Zwei Zeichen, die durch - voneinander getrennt sind, stehen für ein beliebiges Zeichen, das in der ASCII-Tabelle (siehe auch im Anhang) zwischen diesen beiden Zeichen steht, z.B. `[a-dw-x]`.

`[!...]`
steht für ein beliebiges Zeichen, das nicht in den Klammern steht. Auch hier können Sie einen Bereich angeben, z.B. `[!a-z]`.

### Entwertung von Sonderzeichen und Apostrophierung

`\` Der Gegenschrägstrich \ entwertet das darauffolgende Sonderzeichen. Das Sonderzeichen hat dann keine besondere Bedeutung für die Shell mehr.
Die Shell ignoriert das Zeichenpaar \ < Neue Zeile >.

*Beispiel*
`echo \$name \$adresse` liefert: `$name $adresse`

`'...'`
Zeichen, die zwischen Hochkommas stehen, haben keine besondere Bedeutung.

*Beispiel*
`echo '$name $adresse'` liefert: `$name $adresse`

`"..."`
Steht eine Zeichenfolge zwischen Anführungszeichen, dann behalten nur die Sonderzeichen $, ` (Accent grave) und \ (Gegenschrägstrich) ihre besondere Bedeutung. Die übrigen Sonderzeichen werden entwertet. Vorhandene Variablen (Stellungs- oder Kennwortparameter) werden ersetzt, Kommandos zwischen Accents graves `` `...` `` werden ausgewertet.

`...`

Steht eine Zeichenfolge zwischen Accents graves, dann interpretiert die Shell diese Zeichenfolge als Kommando, führt das Kommando aus und setzt die Ausgabe des Kommandos an die Stelle dieser Zeichenfolge. Vorhandene Variablen (Stellungs- oder Kennwortparameter) werden ersetzt.

**Ein-/Ausgabe-Umlenkung**

> datei

Die Standard-Ausgabe soll die Datei *datei* sein. Existiert die Datei bereits, so wird ihr Inhalt überschrieben.

>> datei

Die Standard-Ausgabe wird an die Datei *datei* angehängt. Existiert die Datei noch nicht, wird sie neu angelegt.

**2>** datei

Die Standard-Fehlerausgabe soll in die Datei *datei* umgelenkt werden.

**>&**n Die Standard-Ausgabe soll in die Datei umgelenkt werden, die zur Dateikennzahl *n* gehört.

**>&-** Die Standard-Ausgabe-Datei wird geschlossen.

< datei

Die Standard-Eingabe soll die Datei *datei* sein.

<< wort

Die Shell-Prozedur wird bis zu einer Zeile, die nur aus *wort* besteht, bzw. bis zum Dateiende-Zeichen gelesen. Das so entstandene Dokument ("Here Document") wird zur Standard-Eingabe.
Ist ein Zeichen in *wort* apostrophiert, dann werden die im Dokument enthaltenen Zeichen nicht interpretiert. Andernfalls führt die Shell eine Parameter- und Kommandosubstitution durch und ignoriert (nicht entwertete) Neue-Zeile-Zeichen; in diesem Fall müssen Sie den Gegenschrägstrich \ verwenden, um die Zeichen \ (Gegenschrägstrich), $, ` (Accent grave) und das erste Zeichen von *wort* zu entwerten.

<<-wort

Folgt auf ein << ein -, dann werden in *wort* und allen Zeilen des Here Documents alle führenden Tabulatorzeichen entfernt.

<&n Die Standard-Eingabe soll die Datei sein, die zur Dateikennzahl *n* gehört.

<&- Die Standard-Eingabe-Datei wird geschlossen.

### Shell-Variablen (Parameter)

${name}

Für *${name}* wird der Wert der Variablen *name* (falls vorhanden) eingesetzt. Die geschweiften Klammern sind nur dann notwendig, wenn auf *name* ein oder mehrere Buchstaben, Ziffern oder Unterstriche folgen, die der nicht als Teil des Variablennamens interpretiert werden soll. Durch *name=wert* bzw. *name="wert"* weisen Sie der Variablen *name* einen Wert zu.

${name:-wort}

Besitzt die Variable *name* einen nichtleeren Wert, dann wird der Ausdruck durch diesen Wert ersetzt. Ist die Variable *name* nicht gesetzt oder hat sie einen leeren Wert, dann wird *wort* eingesetzt.

${name:=wort}

Ist die Variable *name* nicht gesetzt oder hat sie einen leeren Wert, so wird ihr der Wert *wort* zugewiesen. Danach wird der Ausdruck durch den Wert von *name* ersetzt.
Stellungsparametern kann auf diese Weise kein neuer Wert zugewiesen werden.

${name:?wort}

Ist die Variable *name* gesetzt und besitzt sie einen nichtleeren Wert, dann wird der Ausdruck durch diesen Wert ersetzt. Andernfalls wird *wort* als Fehlermeldung ausgegeben und die Shell beendet. Fehlt *wort*, dann wird stattdessen die Meldung *parameter null or not set* ausgegeben.

**${name:+wort}**

Ist die Variable *name* gesetzt und besitzt sie einen nichtleeren Wert, dann wird der Ausdruck durch diesen Wert ersetzt. Andernfalls wird er durch eine leere Zeichenfolge ersetzt.

**$n**

*n* kann eine Ziffer zwischen 0 und 9 sein. Für *$0* wird der Name der aufgerufenen Shell-Prozedur eingesetzt, für *$1* der erste Parameter des Aufrufs usw.

*Beispiel*

Wenn Sie eine Shell-Prozedur *proc* folgendermaßen aufrufen:

```
proc Huber Meier
```

dann wird für *$0 proc* eingesetzt, für *$1* der erste Parameter *Huber* und für *$2* der zweite Parameter *Meier*.

**$***

Für *$** werden alle vorhandenen Stellungsparameter eingesetzt, angefangen bei *$1* und getrennt durch Leerzeichen. Sie bilden dann eine einzige Zeichenfolge: "$1 $2 ..."

**$@**

Für *$@* werden alle vorhandenen Stellungsparameter eingesetzt, angefangen bei *$1* und getrennt durch Leerzeichen. Sie bilden dann n einzelne Zeichenfolgen: "$1", "$2", ...

**$#**

Die Anzahl der beim Prozeduraufruf übergebenen oder durch das eingebaute Shell-Kommando *set* erzeugten Stellungsparameter.

**$-**

Die Schalter, die der Shell beim Prozeduraufruf oder über das eingebaute Shell-Kommando *set* übergeben wurden.

**$?**

Der Endestatus des zuletzt synchron ausgeführten Kommandos.

**$$**

Die Prozeßnummer der aktuellen Shell.

**$!**

Die Prozeßnummer des zuletzt angestoßenen Hintergrundprozesses.

Weitere Informationen finden Sie in der Beschreibung des Kommandos *sh*(1) sowie im Buch SINIX Einführung [1].

# A.3 Die ASCII-Zeichen

| dezi-mal | oktal | hexa-dez. | | Bedeutung | Control |
|---|---|---|---|---|---|
| 0 | 00 | 00 | NUL | Null, keine Operation | @ |
| 1 | 01 | 01 | SOH | Start of Heading<br>Vorspannanfang | A |
| 2 | 02 | 02 | STX | Start of Text<br>Textanfang | B |
| 3 | 03 | 03 | ETX | End of Text<br>Textende | C |
| 4 | 04 | 04 | EOT | End of Transmission Taste END<br>Übertragungsende | D |
| 5 | 05 | 05 | ENQ | Enquiry<br>Stationsanruf | E |
| 6 | 06 | 06 | ACK | Acknowledge<br>Bestätigung | F |
| 7 | 07 | 07 | BEL | Bell<br>Klingel | G |
| 8 | 10 | 08 | BS | Backspace<br>Korrekturtaste | H |
| 9 | 11 | 09 | HT | Horizontal Tabulation<br>Tabulatorzeichen | I |
| 10 | 12 | 0A | LF | Line Feed<br>Zeilenvorschub, neue Zeile | J |
| 11 | 13 | 0B | VT | Vertical Tabulation | K |
| 12 | 14 | 0C | FF | Form Feed<br>Formularvorschub | L |
| 13 | 15 | 0D | CR | Carriage Return<br>Wagenrücklauf | M |
| 14 | 16 | 0E | SO | Shift Out<br>Umschalten Zeichensatz | N |
| 15 | 17 | 0F | SI | Shift In<br>Zurückschalten Zeichensatz | O |
| 16 | 20 | 10 | DLE | Data Link Escape<br>Austritt aus der Datenverbindung | P |
| 17 | 21 | 11 | DC1 | Device Control 1<br>Gerätesteuerung 1, Ausgabe fortsetzen | Q |
| 18 | 22 | 12 | DC2 | Device Control 2 | R |
| 19 | 23 | 13 | DC3 | Device Control 3 Ausgabe anhalten | S |
| 20 | 24 | 14 | DC4 | Device Control 4 | T |
| 21 | 25 | 15 | NAK | Negative Acknowledge<br>Fehlermeldung | U |
| 22 | 26 | 16 | SYN | Synchronous Idle<br>Synchronisierung | V |
| 23 | 27 | 17 | ETB | End of Transm. Block<br>Datenblockende | W |
| 24 | 30 | 18 | CAN | Cancel<br>ungültig, Zeilenlöscher | X |
| 25 | 31 | 19 | EM | End of Medium<br>Datenträgerende, quit (Signal3) | Y |
| 26 | 32 | 1A | SUB | Substitute Character<br>Zeichen ersetzen | Z |

| dezi-mal | oktal | hexa-dez. | | Bedeutung | | Control |
|---|---|---|---|---|---|---|
| 27 | 33 | 1B | ESC | Escape | Rücksprung | |
| 28 | 34 | 1C | FS | File Separator | Dateitrennung | \ oder \| |
| 29 | 35 | 1D | GS | Group Separator | Gruppentrennung | ] |
| 30 | 36 | 1E | RS | Record Separator | Satztrennung | ^ |
| 31 | 37 | 1F | US | Unit Separator | Einheitentrennung | _ oder DEL |
| 32 | 40 | 20 | SP | SPACE | Leerzeichen | |
| 33 | 41 | 21 | ! | | | |
| 34 | 42 | 22 | " | | | |
| 35 | 43 | 23 | # | | | |
| 36 | 44 | 24 | $ | | | |
| 37 | 45 | 25 | % | | | |
| 38 | 46 | 26 | & | | | |
| 39 | 47 | 27 | ' | | | |
| 40 | 50 | 28 | ( | | | |
| 41 | 51 | 29 | ) | | | |
| 42 | 52 | 2A | * | | | |
| 43 | 53 | 2B | + | | | |
| 44 | 54 | 2C | , | | | |
| 45 | 55 | 2D | - | | | |
| 46 | 56 | 2E | . | | | |
| 47 | 57 | 2F | / | | | |
| 48 | 60 | 30 | 0 | | | |
| 49 | 61 | 31 | 1 | | | |
| 50 | 62 | 32 | 2 | | | |
| 51 | 63 | 33 | 3 | | | |
| 52 | 64 | 34 | 4 | | | |
| 53 | 65 | 35 | 5 | | | |
| 54 | 66 | 36 | 6 | | | |
| 55 | 67 | 37 | 7 | | | |
| 56 | 70 | 38 | 8 | | | |
| 57 | 71 | 39 | 9 | | | |
| 58 | 72 | 3A | : | | | |
| 59 | 73 | 3B | ; | | | |
| 60 | 74 | 3C | < | | | |
| 61 | 75 | 3D | = | | | |
| 62 | 76 | 3E | > | | | |
| 63 | 77 | 3F | ? | | | |
| 64 | 100 | 40 | @ | (kaufmännisches "at" oder §) | | |
| 65 | 101 | 41 | A | | | |
| 66 | 102 | 42 | B | | | |
| 67 | 103 | 43 | C | | | |
| 68 | 104 | 44 | D | | | |
| 69 | 105 | 45 | E | | | |
| 70 | 106 | 46 | F | | | |
| 71 | 107 | 47 | G | | | |
| 72 | 110 | 48 | H | | | |
| 73 | 111 | 49 | I | | | |
| 74 | 112 | 4A | J | | | |
| 75 | 113 | 4B | K | | | |
| 76 | 114 | 4C | L | | | |
| 77 | 115 | 4D | M | | | |

| dezi-mal | oktal | hexa-dez. | | Bedeutung | Control |
|---|---|---|---|---|---|
| 78 | 116 | 4E | N | | |
| 79 | 117 | 4F | O | | |
| 80 | 120 | 50 | P | | |
| 81 | 121 | 51 | Q | | |
| 82 | 122 | 52 | R | | |
| 83 | 123 | 53 | S | | |
| 84 | 124 | 54 | T | | |
| 85 | 125 | 55 | U | | |
| 86 | 126 | 56 | V | | |
| 87 | 127 | 57 | W | | |
| 88 | 130 | 58 | X | | |
| 89 | 131 | 59 | Y | | |
| 90 | 132 | 5A | Z | | |
| 91 | 133 | 5B | [ | oder Ä | |
| 92 | 134 | 5C | \ | oder Ö | |
| 93 | 135 | 5D | ] | oder Ü | |
| 94 | 136 | 5E | ^ | oder ↑ | |
| 95 | 137 | 5F | _ | oder → | |
| 96 | 140 | 60 | ` | | |
| 97 | 141 | 61 | a | | |
| 98 | 142 | 62 | b | | |
| 99 | 143 | 63 | c | | |
| 100 | 144 | 64 | d | | |
| 101 | 145 | 65 | e | | |
| 102 | 146 | 66 | f | | |
| 103 | 147 | 67 | g | | |
| 104 | 150 | 68 | h | | |
| 105 | 151 | 69 | i | | |
| 106 | 152 | 6A | j | | |
| 107 | 153 | 6B | k | | |
| 108 | 154 | 6C | l | | |
| 109 | 155 | 6D | m | | |
| 110 | 156 | 6E | n | | |
| 111 | 157 | 6F | o | | |
| 112 | 160 | 70 | p | | |
| 113 | 161 | 71 | q | | |
| 114 | 162 | 72 | r | | |
| 115 | 163 | 73 | s | | |
| 116 | 164 | 74 | t | | |
| 117 | 165 | 75 | u | | |
| 118 | 166 | 76 | v | | |
| 119 | 167 | 77 | w | | |
| 120 | 170 | 78 | x | | |
| 121 | 171 | 79 | y | | |
| 122 | 172 | 7A | z | | |
| 123 | 173 | 7B | { | oder ä | |
| 124 | 174 | 7C | \| | oder ö | |
| 125 | 175 | 7D | } | oder ü | |
| 126 | 176 | 7E | ~ | oder ß | |
| 127 | 177 | 7F | DEL | Delete<br>Löschzeichen, Interrupt (Signal2) | |

# Fachwörter deutsch - englisch

| | |
|---|---|
| Abbildung | map |
| Ablauf verfolgen | trace |
| ablauffähiges Programm | executable program |
| absolutes Symbol | absolute symbol |
| Administrator | super user |
| Adresse | address |
| Adreßraum | address space |
| Änderungsgrund | MR, modification request |
| Änderungswunsch | MR, modification request |
| Aktion | action |
| aktuell | current |
| aktuelles Dateiverzeichnis | working directory, local directory |
| aktuelles Umfeld | current environment |
| Anfangszeile | head line |
| Anfrage | request |
| Apostrophier-Mechanismus | quoting mechanism |
| Assembler | assembler |
| Assembler-Quellcode | assembly source program |
| assembliertes Programm | object program |
| aufrufen | call |
| Ausführberechtigung | execute permission |
| ausführen, durchsuchen | execute |
| Ausführung | execution |
| Ausführerlaubnis | execute permission |
| Ausgabe | output |
| Automat | automaton |
| automatisch | automatic |
| Basisadresse | base address |
| Baum | tree |
| Befehl | instruction |
| Benutzer | user |
| Benutzerkennung | user identification |
| Benutzernummer | user ID (UID) |
| Benutzerzeit | user time |
| Bereinigung | cleanup |
| Bereitzeichen | prompt |
| beschreibender Text | descriptive text |

| | |
|---|---|
| Bibliothek | library, archive |
| Bildschirm | screen |
| Bildschirm, Datensichtstation | terminal |
| Binder | loader |
| Binärdatei | binary file |
| break-Anweisung | break |
| bss-Segment | bss |
| cast | cast |
| cast-Anweisung | cast |
| Datei | file |
| Dateideskriptor | file descriptor |
| Dateiende | end-of-file |
| Dateikennzahl | file descriptor |
| Dateiname | file name |
| Dateisystem | file system |
| Dateiverzeichnis | directory |
| Datensegment | data area, data segment |
| Datensegment, nicht-initialisierte Daten | bss |
| Datensichtstation | terminal |
| Datensymbol | data symbol |
| Delta | Delta, Version |
| Deltanummer | SID |
| Diskette | floppy disk, floppy |
| doppelte Gleitpunktzahl | double floating point |
| Dummy | dummy |
| effektive Benutzernummer | effective UID |
| effektive Gruppennummer | effective GID |
| Eigentümer | owner |
| einfügen | insert |
| Eingabe | input |
| Eingabeaufforderungszeichen | prompt |
| Eingabestrom | input stream |
| eingebaute Shell-Kommandos | built-in commands |
| einstelliger Operator | monadic operator |
| Endestatus | exit status |
| endlicher Automat | finite automaton |
| externe Referenz | external references |
| externes Symbol | external symbol |
| Fehler | error, bug |
| Fehlerbehandlung | error handling |

| | |
|---|---|
| Fehlercode | error code |
| Fehlermeldung | error diagnostic |
| Fehlersuche | debugging |
| Feld | array, field |
| Feldtrenner | field separator |
| Festplatte | disk |
| Flucht | escape |
| Freispeicherliste | free list |
| Gedächtnisstütze | reminder |
| Gegenschrägstrich | backslash |
| Gerät, externer Datenträger | device |
| Gerätedatei | special file |
| geschweifte Klammern | braces, curly braces |
| Gleitkommazahl | floating point number |
| Gleitpunktzahl | floating point number |
| Gruppenname | group name |
| Gruppennummer | group ID (GID) |
| Haltepunkt | breakpoint |
| Hintergrundprozeß | background-process |
| Home-Dateiverzeichnis | home-directory |
| Identifikations-Schlüsselwort | id keyword, identification keyword |
| Include-Datei | include file |
| Indexeintrag | inode |
| Indexnummer | inumber |
| Keller | stack (LIFO) |
| Kennwort | password |
| Kennwortparameter | user-defined variable, keyword parameter |
| Knoten | node |
| Konsole | console |
| Kontextabhängigkeit | context sensitivity |
| kontextfrei | context-free |
| Kontrollanweisung | control statement |
| Kontrollzeichen | control character |
| Korrekturtaste | back space key |
| Laufzeit | elapsed time |
| Laufzeit-Startfunktion | runtime startoff |
| Laufzeitinkonsistenz | phase error |
| Leeres Kommando | null command |
| Leerzeichen | blank |

| | |
|---|---|
| Leseerlaubnis | read permission |
| Login-Dateiverzeichnis | login-Directory |
| Login-Name | login name |
| LR(1)-Parsing-Algorithmus | LR(1) parsing algorithm |
| löschen | delete |
| mehrdeutig | ambiguous |
| Mehrdeutigkeiten auflösen | disambiguate |
| Metazeichen | meta character |
| Muster | pattern |
| Neue Zeile | new line |
| Nicht-Terminalsymbol | nonterminal symbol |
| Nicht-Terminalzeichen | nonterminal symbol |
| Objektcode | object code |
| Objektdatei | object program, object file |
| Objektmodul | object program |
| Offset | offset |
| Optimierer | optimizer |
| Originaldatei | initial delta |
| Parameter | flag |
| Parser | parser |
| Parsergenerator | compiler-compiler |
| passen | match |
| Pfad | path |
| Pfadname | pathname |
| Phasenfehler | phase error |
| Pipe, Einwegkanal | pipe |
| Pipeline | pipeline |
| portierbar | portable |
| Portierbarkeit | portability |
| Priorität | priority |
| Prozedur | procedure,script |
| Prozeß | process |
| Prozeßbeendigung | process termination |
| Präprozessor | preprocessor |
| Präzedenz | precedence |
| (auf Konsistenz) prüfen | validate |
| Prüfprogramm | checker, verifier |
| Prüfsumme | check-sum |
| Puffer | buffer |
| Punkt-Kommando | dot command |
| Quellprogramm | source program |

| | |
|---|---|
| Quellcode | source code |
| reale Benutzernummer | real UID |
| reale Gruppennummer | real GID |
| reduzieren | reduce |
| Regel | rule |
| Regelteil | rule section |
| regulärer Ausdruck | regular expression |
| Relokationsbit | relocation bit |
| Root-Dateiverzeichnis | root directory |
| runde Klammern | parentheses |
| Scanner | lexical analyzer, scanner |
| Schalter | flag, option |
| Schleife | loop |
| Schreiberlaubnis | write permission |
| Schreibmarke | cursor |
| Schrägstrich | slash |
| Schutzbit | protection bit |
| Segment | segment |
| Shell | shell |
| Shell-Umgebung | environment |
| Sonderzeichen | special character |
| Speicherabzug | core, core image, core dump, corefile |
| Speicherplatz | memory |
| sperren | lock |
| Standard-Ausgabe | standard output |
| Standard-Eingabe | standard input |
| Standard-Fehlerausgabe | standard error |
| Standardwert | default value |
| Startadresse | entry point |
| Startsymbol | start symbol |
| statisch | static |
| statischer Datenbereich | static data area |
| Stellungsparameter | positional parameter |
| steuern | schedule |
| Suchzeichenfolge | search string |
| Symboltabelle | symbol table, name-list |
| syntaktische Variable | nonterminal symbol |
| Syntaxfehler | syntax error |
| Systemaufruf | system call |
| Systemkern | kernel |

| | |
|---|---|
| Systemverwalter | super user, administrator |
| Systemzeit | system time |
| Tabulatorzeichen | tab |
| Tastatur | keyboard |
| Taste | key |
| Terminalsymbol | token, terminal symbol |
| Terminalzeichen | token, terminal symbol |
| Testhilfe | debugger |
| Textmuster | scanning pattern |
| Textsegment | text image, text segment |
| Textsymbol | text symbol |
| Textzeichen | text character |
| Token | token, terminal symbol |
| Topdelta | top delta |
| Übergangstabelle | parsing table |
| Übersetzer | compiler |
| Umgebung | environment |
| Umgebungsvariable | environmental variable |
| umleiten | redirect |
| Unterbrechung | interrupt |
| Unterstrich | underscore |
| Vergleichsausdruck | relational expression |
| Verkettung | concatenation |
| Version | delta, version |
| Versionsnummer | SID |
| Verweis | link |
| Vorausschau | lookahead |
| Vorrang | precedence |
| Vorzeichen | sign |
| Wagenrücklauf | carriage return |
| wahlfreier Zugriff | random access |
| Warteschlange | queue (FIFO) |
| Wert | value |
| Zeichen | character |
| Zeichenfolge | string |
| Zeichenkette | string |
| Zeichenreihe | string |
| Zeiger | pointer |
| Zeilentrenner | record separator |
| Zeittabelle | profile data |
| Zentraleinheit | CPU |

| | |
|---|---|
| Ziffer | digit |
| Zufallszahl | random number |
| Zugriff | access |
| Zugriffsart | access mode |
| Zugriffsberechtigung | access permission |
| Zugriffsrecht | mode |
| Zustand | state |
| Zuweisung | assignment |
| zweistelliger Operator | binary operator |
| Zwischendatei | temporary file |
| Zwischenraum | space |

# Fachwörter englisch - deutsch

| | |
|---|---|
| absolute symbol | absolutes Symbol |
| access | Zugriff |
| access mode | Zugriffsart |
| access permission | Zugriffsberechtigung |
| action | Aktion |
| address | Adresse |
| address space | Adreßraum |
| administrator | Systemverwalter, Administrator |
| ambiguous | mehrdeutig |
| archive | Bibliothek |
| array | Feld, Vektor |
| assembler | Assembler |
| assembly source program | Assembler-Quellcode, Assembler-Quellprogramm |
| assignment | Zuweisung |
| automatic | automatisch |
| automaton | Automat |
| back space key | Korrekturtaste |
| background-process | Hintergrundprozeß |
| backslash | Gegenschrägstrich |
| base address | Basisadresse |
| binary file | Binärdatei |
| binary operator | zweistelliger Operator |
| blank | Leerzeichen |
| braces, curly braces | geschweifte Klammern |
| break | break, break-Anweisung |
| breakpoint | Haltepunkt |
| bss | Datensegment, bss-Segment, nicht-initialisierte Daten |
| buffer | Puffer |
| bug | Fehler |
| built-in commands | eingebaute Shell-Kommandos |
| call | aufrufen |
| carriage return | Wagenrücklauf |
| cast | cast, cast-Anweisung |
| character | Zeichen |
| check-sum | Prüfsumme |
| checker | Prüfprogramm |

| | |
|---|---|
| cleanup | Bereinigung |
| compiler | Übersetzer |
| compiler-compiler | Parsergenerator |
| concatenation | Verkettung |
| console | Konsole |
| context sensitivity | Kontextabhängigkeit |
| context-free | kontextfrei |
| control character | Kontrollzeichen |
| control statement | Kontrollanweisung |
| core | Speicherabzug |
| core dump | Speicherabzug |
| core image | Speicherabzug |
| corefile | Speicherabzug |
| CPU | Zentraleinheit |
| curly braces | geschweifte Klammern |
| current | aktuell |
| current environment | aktuelles Umfeld |
| cursor | Schreibmarke |
| data area | Datensegment |
| data segment | Datensegment |
| data symbol | Datensymbol |
| debugger | Programm zur Fehleranalyse, Testhilfe |
| debugging | Fehlersuche |
| default value | Standardwert |
| delete | löschen |
| delta | Version, Delta |
| descriptive text | beschreibender Text |
| device | Gerät, externer Datenträger |
| digit | Ziffer |
| directory | Dateiverzeichnis |
| disambiguate | Mehrdeutigkeiten auflösen |
| disk | Festplatte |
| dot command | Punkt-Kommando |
| dummy | Dummy |
| effective GID | effektive Gruppennummer |
| effective PID | effektive Prozeßnummer |
| effective UID | effektive Benutzernummer |
| elapsed time | Laufzeit |
| end-of-file | Dateiende |
| entry point | Startadresse |

| | |
|---|---|
| environment | Shell-Umgebung, Umgebung |
| environmental variable | Umgebungsvariable |
| error | Fehler |
| error code | Fehlercode |
| error diagnostic | Fehlermeldung |
| error handling | Fehlerbehandlung |
| escape | Flucht |
| executable program | ablauffähiges Programm |
| execute | ausführen, durchsuchen |
| execute permission | Ausführerlaubnis |
| execute permission | Ausführberechtigung |
| execution | Ausführung |
| exit status | Endestatus |
| external reference | externe Referenz |
| external symbol | externes Symbol |
| field | Feld |
| field separator | Feldtrenner |
| file | Datei |
| file descriptor | Dateikennzahl, Dateideskriptor |
| file system | Dateisystem |
| finite automaton | endlicher Automat |
| flag | Schalter, Parameter |
| floating point number | Gleitkommazahl, Gleitpunktzahl |
| floppy | Diskette |
| floppy disk | Diskette |
| free list | Freispeicherliste |
| group ID (GID) | Gruppennummer |
| group identification (GID) | Gruppennummer |
| groupname | Gruppenname |
| head line | Anfangszeile |
| home-directory | Home-Dateiverzeichnis |
| id keyword | Identifikations-Schlüsselwort |
| identification keyword | Identifikations-Schlüsselwort |
| include file | Include-Datei |
| initial delta | Originaldatei |
| inode | Indexeintrag |
| input | Eingabe |
| input stream | Eingabestrom |
| insert | einfügen |
| instruction | Befehl |
| interrupt | Unterbrechung |

| | |
|---|---|
| inumber | Indexnummer |
| kernel | Systemkern |
| key | Schlüssel, Taste |
| keyboard | Tastatur |
| keyword parameter | Kennwortparameter |
| lexical analyzer | Scanner |
| library | Bibliothek |
| link | Verweis |
| loader | Binder |
| local directory | aktuelles Dateiverzeichnis |
| lock | sperren |
| login name | Login-Name |
| login-directory | Login-Dateiverzeichnis |
| lookahead | Vorausschau |
| loop | Schleife |
| LR(1) parsing algorithm | LR(1)-Parsing-Algorithmus |
| map | Abbildung |
| match | passen |
| memory | Speicherplatz |
| meta character | Metazeichen |
| mode | Zugriffsrecht |
| modification request | Änderungsgrund, Änderungswunsch |
| monadic operator | einstelliger Operator |
| MR | Änderungsgrund, Änderungswunsch |
| name-list | Symboltabelle |
| new line | Neue Zeile |
| node | Knoten |
| nonterminal symbol | Nicht-Terminalsymbol, Nicht-Terminalzeichen, syntaktische Variable |
| null command | Leeres Kommando |
| object code | Objektcode, assembliertes Programm |
| object file | Objektdatei |
| object program | Objektmodul, Objektdatei |
| offset | Offset, Abstand |
| optimizer | Optimierer |
| option | Schalter |
| output | Ausgabe |

| | |
|---|---|
| owner | Eigentümer |
| parentheses | runde Klammern |
| parser | Parser |
| parsing table | Übergangstabelle |
| password | Kennwort |
| path | Pfad |
| pathname | Pfadname |
| pattern | Muster |
| phase error | Laufzeitinkonsistenz, Phasenfehler |
| pipe | Pipe, Einwegkanal |
| pipeline | Pipeline |
| pointer | Zeiger |
| portability | Portierbarkeit |
| portable | portierbar |
| positional parameter | Stellungsparameter |
| preprocessor | Präprozessor |
| priority | Priorität |
| procedure | Prozedur |
| process | Prozeß |
| process termination | Prozeßbeendigung |
| profile data | Zeittabelle |
| prompt | Bereitzeichen, Eingabeaufforderungszeichen |
| protection bit | Schutzbit |
| queue (FIFO) | Warteschlange |
| quoting mechanism | Apostrophier-Mechanismus |
| random access | wahlfreier Zugriff |
| random number | Zufallszahl |
| read permission | Leseerlaubnis |
| real GID | reale Gruppennummer |
| real PID | reale Prozeßnummer |
| real UID | reale Benutzernummer |
| record separator | Zeilentrenner |
| redirection | umleiten |
| regular expression | regulärer Ausdruck |
| relational expression | Vergleichsausdruck |
| relocation bit | Relokationsbit |
| reminder | Gedächtnisstütze |
| request | Anfrage |
| root directory | Root-Dateiverzeichnis |

| | |
|---|---|
| rule | Regel |
| rule section | Regelteil |
| runtime startoff | Laufzeit-Startfunktion |
| scanner | Scanner |
| scanning pattern | Textmuster |
| schedule | steuern |
| screen | Bildschirm |
| script | Prozedur |
| search string | Suchzeichenfolge |
| segment | Segment |
| shell | Shell |
| SID | Versionsnummer, Deltanummer |
| sign | Vorzeichen |
| size | Größe |
| slash | Schrägstrich |
| source code | Quellcode |
| source programm | Quellprogramm |
| space | Zwischenraum |
| special character | Sonderzeichen |
| special file | Gerätedatei |
| stack (LIFO) | Keller |
| standard error | Standard-Fehlerausgabe |
| standard input | Standard-Eingabe |
| standard output | Standard-Ausgabe |
| start symbol | Startsymbol |
| state | Zustand |
| static | statisch |
| static data area | statischer Datenbereich |
| string | Zeichenreihe, Zeichenfolge, Zeichenkette |
| super user | Systemverwalter, Administrator |
| symbol table | Symboltabelle |
| syntax error | Syntaxfehler |
| system call | Systemaufruf |
| system time | Systemzeit |
| tab | Tabulatorzeichen |
| temporary file | Zwischendatei, temporäre Datei |
| terminal | Bildschirm, Datensichtstation |
| terminal symbol | Token, Terminalzeichen, Terminalsymbol |
| text character | Textzeichen |

| | |
|---|---|
| text image | Textsegment |
| text segment | Textsegment |
| text symbol | Textsymbol |
| token | Token, Terminalzeichen, Terminalsymbol |
| top delta | Topdelta |
| trace | Ablauf verfolgen |
| underscore | Unterstrich |
| user | Benutzer |
| user ID (UID) | Benutzernummer |
| user identification (UID) | Benutzernummer |
| user time | Benutzerzeit |
| user-defined variable | Kennwortparameter |
| validate | (auf Konsistenz) prüfen |
| value | Wert |
| verifier | Prüfprogramm |
| version | Version, Delta |
| working directory | aktuelles Dateiverzeichnis |
| write permission | Schreiberlaubnis |

# Literatur

## SINIX-Handbücher

[1] Betriebssystem SINIX V5.2
Einführung

[2] Betriebssystem SINIX V5.2
CES C-Entwicklungssystem

[3] Betriebssystem SINIX V5.2
Systemverwaltung

[4] Betriebssystem SINIX V5.0
Kommandos
Englischsprachige Ausgabe

[5] Betriebssystem SINIX
Buch 1

[6] Betriebssystem SINIX
Buch 2
Menüs

[7] Betriebssystem SINIX
CES Buch 1
Werkzeuge zur C-Programmierung
Grundlagen und Kommandos

[8] Betriebssystem SINIX
CES Buch 2
Werkzeuge zur C-Programmierung
Systemaufrufe, C-Funktionen und Makros

## X/OPEN Guide

[9] X/OPEN Portability Guide
Elsevier Science Publishing Company Amsterdam, 1987

## Literatur zu UNIX

[10] M.J. Bach
The Design of the UNIX System
Prentice-Hall Englewood Cliffs, 1986

[11] M. Banahan, A. Rutter
UNIX lernen, verstehen, anwenden
Deutsche Ausgabe von Prof. Dr. A.T. Schreiner, T.Mandry
Carl Hanser Verlag München, 1984

[12] M. Banahan, A. Rutter
UNIX - the book
Wilmslow, UK, Sigma Technical Press, 1982

[13] S.R. Bourne
Das UNIX System
Addison-Wesley Verlag (Deutschland) GmbH, 1985

[14] Kaare Christian
The UNIX Operating System
John Wiley & Sons, 1983

[15] Jürgen Gulbins
UNIX
Springer-Verlag Berlin, 1985

[16] B.W. Kernighan, R. Pike
Der UNIX-Werkzeugkasten
Carl Hanser Verlag München, 1987

[17] D.M. Ritchie
The UNIX Time-Sharing System: A Retrospective
Bell System Technical Journal,
Vol. 57, No. 6, 1947-69
American Telephone and Telegraph Company, 1978

[18] R. Thomas und J. Yates
A User Guide to the Unix System
Berkeley: Osborne/McGraw-Hill, 1982

[19] R. Thomas, L.R. Rogers, J. Yates
Advanced Programmer's Guide to UNIX System V
Berkeley: Osborne/McGraw-Hill, 1986

**Literatur zur Sprache C**

[20] H. Herold, W. Unger
Das C-Buch
te-wi Verlag München, 1986

[21] B.W. Kernighan, D.M. Ritchie
Programmieren in C
Deutsche Ausgabe von Prof. Dr. A.T. Schreiner, Dr. Ernst Janich
Carl Hanser Verlag München, 1983

[22] T. Plum
Das C-Lernbuch
Carl Hanser Verlag München Wien, Prentice-Hall London, 1985

# Stichwörter

# Nennen Sie uns Ihren »Stolperstein«

Buch: Kommandos V5.2, U3903-J-Z95-1

| Seite | Mein »Stolperstein«: | Datum: |
|---|---|---|
| | | |

Das Handbuch benutze ich: ☐ sehr häufig ☐ gelegentlich zum Nachschlagen
☐ als Programmierer ☐ Systemverwalter
☐ Sachbearbeiter ☐ _____

Buch: Kommandos V5.2, U3903-J-Z95-1

| Seite: | Mein »Stolperstein«: | Datum: |
|---|---|---|
| | | |

Das Handbuch benutze ich: ☐ sehr häufig ☐ gelegentlich zum Nachschlagen
☐ als Programmierer ☐ Systemverwalter
☐ Sachbearbeiter ☐ _____

# Kritik – Anregungen – Korrekturen

Schreiben Sie uns bitte, was Ihnen an diesem Buch gefällt und was Ihnen nicht gefällt:

- vor allem, wo Sie Fehler entdeckt haben,
- wo etwas unklar beschrieben ist.

Absender:

Name

Firma/Dienststelle

Straße, Postfach

PLZ Ort

Telefon ( )

betreut von ZN

von Hrn./Fr.

Antwort

Bitte mit 60 Pf. frankieren, falls Marke zur Hand.

Siemens AG
K D ST QM2
Manualredaktion
Otto-Hahn-Ring 6
Postfach 83 09 51

D-8000 München 83

Absender:

Name

Firma/Dienststelle

Straße, Postfach

PLZ Ort

Telefon ( )

betreut von ZN

von Hrn./Fr.

Antwort

Bitte mit 60 Pf. frankieren, falls Marke zur Hand.

Siemens AG
K D ST QM2
Manualredaktion
Otto-Hahn-Ring 6
Postfach 83 09 51

D-8000 München 83