



Leseprobe

Erste Schritte
Umlenkung, Pipelines und Filter
Trainingsunterlage

soluzione Script GmbH
Karl-Theodor-Straße 25
80803 München

Telefon: 089 38 99 70-50
Telefax: 089 38 99 70-77
script@soluzione.de
www.soluzione.de

© Copyright 2003 Alle Rechte vorbehalten. Kein Teil dieser Unterlage darf in irgendeiner Form (Druck, Fotokopie, Mikrofilm oder anderes Reproduktionsverfahren) ohne schriftliche Genehmigung der soluzione Script GmbH reproduziert oder unter Anwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Diese Unterlage wurde mit großer Sorgfalt erstellt und geprüft. Trotzdem können Fehler nicht vollkommen ausgeschlossen werden. Die Autoren, und soluzione Script können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in dieser Unterlage berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, daß solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

Inhaltsverzeichnis

1	Erste Schritte	3
1.1	Lernziele	3
1.2	Anmelden am System	3
1.3	Online-Hilfe verwenden	5
1.3.1	Manual-Pages	5
1.3.2	GNU info	9
1.3.3	HOWTOS, Paketdokumentation, Kerneldokumentation	10
1.4	Tastaturfunktionen der Shell	11
1.5	Abmelden vom System	12
1.6	Struktur eines UNIX-Befehls	12
1.7	Herunterfahren des Systems	14
1.8	Übungen	19
1.9	Lösungen	20
2	Umleitung, Pipelines und Filter	23
2.1	Lernziele	23
2.2	Das Kanalkonzept von UNIX	23
2.3	Umleitungen	24
2.3.1	Umleiten der Ausgabe in eine Datei	24
2.3.2	Umleiten von Kanal 1 und 2 in dieselbe Datei	26
2.3.3	Bestehende Dateien erweitern	26
2.3.4	Eingabeumleitung bis Dateendezeichen	26
2.4	Der Pipe-Mechanismus	27
2.4.1	Die Pipe zur Verbindung von Befehlen	27
2.4.2	tee zum Erstellen von Protokollen	29
2.4.3	xargs — Pipe-Datenstrom als Kommandoargumente	29
2.4.4	Named Pipes	31
2.5	Der Einsatz von Filtern in der Shell	32
2.5.1	Was ist ein Filter?	32
2.5.2	Suchen mit regulären Ausdrücken — grep	33

INHALTSVERZEICHNIS

2.5.3	Suchen, ersetzen, Text bearbeiten: sed	37
2.5.4	tr — Zeichenersetzung	39
2.5.5	Spalten ausschneiden (cut, awk)	43
2.5.6	Sortieren einer Datei — sort	45
2.5.7	Zählen von Zeilen und Wörtern — wc	50
2.5.8	Anfang und Ende einer Datei anzeigen — head, tail	50
2.6	Das Wichtigste in Kürze	52
2.7	Tips für die Praxis	53
2.8	Übungen	55
2.9	Lösungen	57
3	Stichwortverzeichnis	59

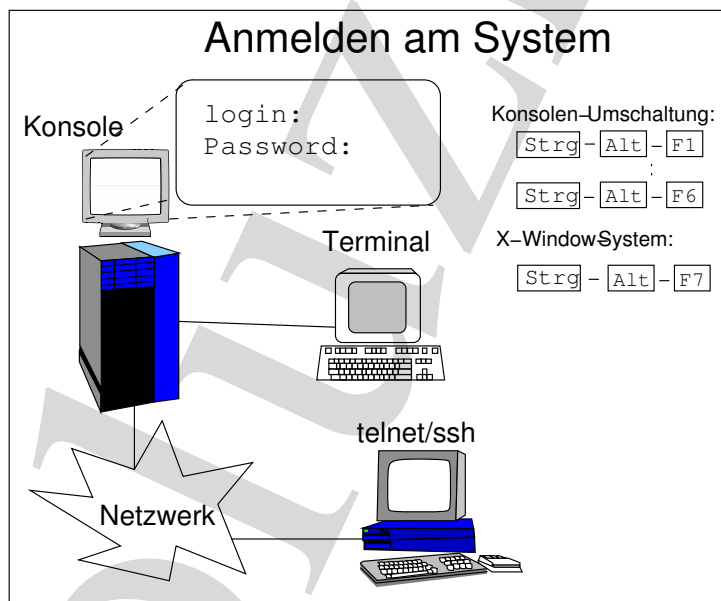
1 Erste Schritte

1.1 Lernziele

In diesem Kapitel lernen Sie Folgendes:

- Sich am System an- und abmelden
- Die Tastaturkürzel der Shell beherrschen
- Ihr persönliches Paßwort ändern
- Die Online-Hilfefunktionen und die Dokumentation verwenden
- Das System sauber herunterzufahren

1.2 Anmelden am System



Stellen Sie sich einmal vor, wie es möglich geworden ist, daß mehrere Menschen in dem selben Haus wohnen: Jeder Bewohner hat ein eigenes *Appartement*, in dessen Grenzen er tun und lassen kann, was er will, vorausgesetzt, er stört seine Mitbewohner nicht wesentlich. Um Zugang zu einem Appartement zu bekommen, benötigt man den Namen des Bewohners, um das richtige Appartement ausfindig zu machen, und den

Erste Schritte

Schlüssel. Um sicherzugehen, daß kein Unbefugter in ein Appartement eindringt, sind die Schlüssel alle verschieden und so kompliziert geschliffen, daß die Schlösser kaum zu knacken sind.

Da Linux, wie alle UNIX-Varianten, ein klassisches Mehrbenutzer-System ist, benötigt man zum Arbeiten auf dem Rechner einen *Benutzerzugang* oder *Account*. Dazu gehört ein *Benutzername* und ein *Paßwort*, mit deren Hilfe man sich beim System anmelden muß. Greift man auf den Vergleich mit dem Haus zurück, so legt der Benutzername fest, auf *welches* Appartement Zutritt gewünscht wird, und das Paßwort entspricht dem *Schlüssel*. Außerdem hat jeder Benutzer ein *Homeverzeichnis*, in dem er sich nach dem sogenannten *Login-Vorgang*, also der Eingabe von Benutzername und Paßwort, wiederfindet. In diesem Heimatverzeichnis kann ein Benutzer seine persönlichen Daten ablegen, und nur hier hat ein Benutzer uneingeschränkte Schreibrechte. Der verfügbare Speicherplatz kann jedoch, um übermäßigen Wildwuchs zu vermeiden, mit dem *Quota-System* begrenzt werden.

Nur der *Systemverwalter*, der den Benutzernamen `root` hat, kann solche Benutzerkonten — oder kurz: Benutzer — einrichten. `root` ist in einem UNIX-System allmächtig, und darf *alle* Dateien im System lesen und schreiben. Entsprechend ist `root` auch der einzige Benutzer, der ein UNIX-System kaputtmachen kann, also:

*Seien Sie als `root` immer sehr, sehr vorsichtig. Melden Sie sich nur als `root` an, wenn Sie **wirklich** Administrationsaufgaben durchzuführen haben. Für den täglichen Gebrauch legen Sie sich lieber einen normalen Benutzer an!*

Eine Beispielsitzung

```
1 Red Hat Linux release 6.1 (Cartman)
2 Kernel 2.2.12-21 on an i586
3 login: foo
4 Password:
5 Last login: Tue Apr 25 14:47:50 from localhost.localdomain
6
7 Bitte beachten Sie: am 30. Am Samstag, den 29. April 2000
8 wird das System zu Upgrade-Zwecken heruntergefahren!
9
10 [foo@johannes foo]$ date
11 Tue Apr 25 14:51:34 CEST 2000
12 [foo@johannes foo]$ cal
13     April 2000
14 Su Mo Tu We Th Fr Sa
15                   1
```

```
16  2  3  4  5  6  7  8
17  9 10 11 12 13 14 15
18 16 17 18 19 20 21 22
19 23 24 25 26 27 28 29
20 30
21 [foo@johannes foo]$ exit
```

Erläuterung: Zunächst präsentiert einem das System vor der Login-Aufforderung in den Zeilen 1-2 den Inhalt der Datei `/etc/issue`. Nach der Login-Aufforderung in Zeile 3 gibt man seinen Benutzernamen und das zugehörige Paßwort ein (Zeile 4), das sinnvollerweise nicht am Bildschirm angezeigt wird. In den Zeilen 6-9 wird der Inhalt der Datei `/etc/motd` (Message of the Day) angezeigt. Die Kommandos **date** und **cal** zeigen Datum und Uhrzeit an, schließlich beenden wir mit dem Kommando **exit** diese Sitzung.

1.3 Online-Hilfe verwenden

1.3.1 Manual-Pages

Der Unterschied zwischen einem Normalbenutzer und einem Guru besteht in der virtuoson Benutzung der Online-Dokumentation. In einem Linux-System können durchaus mehr als 2000 ausführbare Programme installiert sein. Da es unmöglich ist, alle diese Tools zu kennen, oder gar deren Eigenschaften, muß man notwendigerweise irgendwie herausfinden können, ob es schon ein Tool gibt, das eine gewünschte Aufgabe erfüllt. Wenn das der Fall ist, dann sollten wir auch herausfinden können, wie das Programm funktioniert.



Achtung: Als Linux-Einsteiger neigt man leicht dazu, diesen Abschnitt auf die leichte Schulter zu nehmen. Für die Praxis ist die Verwendung der Online-Dokumentation aber absolut unentbehrlich, wenn man sich nicht ganz verloren vorkommen will. Der geübte Umgang mit der Dokumentation ist *sehr, sehr, sehr, wichtig!!!!* (Wirklich!)

Die Standard-Dokumentation eines jeden UNIX-Systems, also auch Linux besteht aus den sogenannten *manual-pages*, oder, kurz: *man-pages*.

Beispiel:

```
man date
```

Resultat:

Die Ausgabe von `man date`

DATE (1)

FSF

DATE (1)

Erste Schritte

NAME

date - print or set the system date and time

SYNOPSIS

```
date [OPTION]... [+FORMAT]
date [OPTION] [MMDDhhmm[[CC]YY][.ss]]
```

DESCRIPTION

Display the current time in the given FORMAT, or set the system date.

```
-d, --date=STRING
    display time described by STRING, not 'now'

-f, --file=DATEFILE
    like --date once for each line of DATEFILE

-I, --iso-8601[=TIMESPEC]
    output an ISO-8601 compliant date/time string.
```

In diesem Fall haben wir unseren Befehl schon gekannt, wir schlagen aber in der Man-Page nach, um mehr über seine Funktionen zu erfahren. Für das schnelle Nachschlagen gibt es bei den meisten Kommandos noch eine Alternative, zum Beispiel falls in einem Notfall keine Man-Pages verfügbar sind: Alle Kommandos zeigen mit der Option `--help`, `-h` oder `-?` eine Kurz-Hilfe an. *Beispiel:*

```
[jack@johannes jack]$ man --help
man, version 1.5f
```

```
usage: man [-adfhktwW] [section] [-M path] [-P pager] [-S list]
          [-m system] [-p string] name ...
```

```
a : find all matching entries
c : do not use cat file
d : print gobs of debugging information
D : as for -d, but also display the pages
f : same as whatis(1)
h : print this help message
k : same as apropos(1)
K : search for a string in all pages
t : use troff to format pages for printing
```

1.3 Online-Hilfe verwenden

w : print location of man page(s) that would be displayed
 (if no name given: print directories that would be searched)
 W : as for -w, but display filenames only

C file : use 'file' as configuration file
 M path : set search path for manual pages to 'path'
 P pager : use program 'pager' to display pages
 S list : colon separated section list
 m system : search for alternate system's man pages
 p string : string tells which preprocessors to run
 e - [n]eqn(1) p - pic(1) t - tbl(1)
 g - grap(1) r - refer(1) v - vgrind(1)

Achtung: Eckige Klammern um ein Argument herum bedeuten, daß das Argument unter Umständen angegeben werden *kann*, aber *nicht* angegeben werden *muß*. Stehen *keine eckigen Klammern* um ein Argument, so ist es *zwingend erforderlich*.

Liest man nun eine *Man-Page*, so wird sie unter Linux mit dem Pager **less** dargestellt. Die wichtigsten Tastenbefehle für less sind:

Funktion	Tastenkombination, (Alternative)
Einen Bildschirm weiter	Leertaste, Bild↓
Einen Bildschirm zurück	b, Bild↑
Eine Zeile weiter	↓
Eine Zeile zurück	↑
Suche vorwärts nach <i>text</i>	/ <i>text</i>
Suche rückwärts nach <i>text</i>	? <i>text</i>
Suche wiederholen	n
Suche in umgekehrter Richtung wiederholen	N
Beenden	q

Nun waren wir gerade in der glücklichen Lage, den obigen Befehl schon zu kennen. Jedoch steht man nicht selten vor dem Problem, eine bestimmte Aufgabe erledigen zu müssen, *ohne* den Befehl zu kennen, und bei mehr als 1000 UNIX-Tools ist das nicht verwunderlich. Genau für diese Situation gibt es eine Wunderwaffe, nämlich den Befehl **apropos**. Damit lassen sich die *Kurzbeschreibungen* aller Befehle nach *Schlüsselwörtern* durchsuchen:

apropos *Schlüsselwort*

Beispiel: Wir wollen herausfinden, mit welchem Befehl wir unser Paßwort wechseln können:

```
$ apropos password
```

Erste Schritte

change (1)	- change user password expire information
crypt (3)	- password and data encryption
dpasswd (8)	- change dialup password
endpwent (3)	- get password file entry
fgetpwent (3)	- get password file entry
getpass (3)	- get a password
getpw (3)	- Re-construct password line entry
getpwent (3)	- get password file entry
getpwnam (3)	- get password file entry
getpwuid (3)	- get password file entry
passwd (1)	- change user password
passwd (4)	- The password file
passwd (5)	- password file
putpwent (3)	- write a password file entry
pwck (1)	- verify integrity of password files
pwconv (8)	- convert and update shadow password files
setpwent (3)	- get password file entry
shadow (3)	- encrypted password file routines
shadow (4)	- encrypted password file

Auffällig daran ist die Zahl hinter jedem Befehl. Der Ordnung halber sind alle Man-Pages in *sections* gegliedert. Die Zahl ist die Nummer der *section*. Diese hat unter Linux folgende Bedeutung:

section	Bedeutung	Informationen für
1	user level commands	Benutzer
2	system calls	Programmierer
3	library functions	Programmierer
4	device drivers	Programmierer und Systembetreuer
5	file formats	Programmierer und Systembetreuer
6	games	alle
7	miscellaneous stuff	alle
8	system maintenance and operation commands	Systembetreuer

Manchmal kommt es vor, daß ein Befehl in mehreren *sections* auftritt. Dann kann man mit dem **whatis**-Befehl herausfinden, in welchen:

```
$ whatis passwd
passwd (1)      - update a user's authentication tokens(s)
passwd (5)      - password file
```

In *section 1* (Benutzerkommandos) haben wir also den Befehl zum Wechseln des Paßworts. In *section 5* (Dateiformate) dagegen die Beschreibung des Dateiformats der Benutzerdatenbank `/etc/passwd`. Wollen wir die Man-Page dieser Datei betrachten, so geben wir ein:

```
$ man 5 passwd
```

1.3.2 GNU info

Bei allen Kommandos des GNU-Projektes sind meist zwar Man-Pages mit dabei. Jedoch hat das GNU-Projekt ein eigenes Online-Hilfe-System entwickelt: **info**

Dabei handelt es sich um eine Art Hypertextformat, das entweder mit dem Editor **emacs** oder mit dem Befehl **info** gelesen werden kann.

Funktion	Tastenkombination, (Alternative)
Einen Bildschirm weiter	Leertaste, Bild↓
Einen Bildschirm zurück	Bild↑
Eine Zeile weiter	↓
Eine Zeile zurück	↑
Suche vorwärts nach <i>text</i>	Ctrl-s <i>text</i>
Suche rückwärts nach <i>text</i>	Ctrl-r <i>text</i>
Suche wiederholen	Ctrl-s bzw. Ctrl-r
Ins Untermenü	RETURN
Untermenü verlassen („up“)	u
Hilfe	h
Beenden	q

Die Ausgabe von info

```
File: dir      Node: Top      This is the top of the INFO tree
```

```
This (the Directory node) gives a menu of major topics.
Typing "q" exits, "?" lists all Info commands, "d" returns here,
"h" gives a primer for first-timers,
"mEmacs<Return>" visits the Emacs topic, etc.
```

```
In Emacs, you can click mouse button 2 on a menu item or cross reference
to select it.
```

* Menu:

Erste Schritte

Texinfo documentation system

- * Info: (info). Documentation browsing system.
- * Texinfo: (texinfo). The GNU documentation format.
- * install-info: (texinfo) Invoking install-info. Update info/dir entries.
- * makeinfo: (texinfo) makeinfo Preferred. Translate Texinfo source.
- * texi2dvi: (texinfo) Format with texi2dvi. Print Texinfo documents.
- * texindex: (texinfo) Format with tex/texindex. Sort Texinfo index files.

Miscellaneous

- * As: (as). The GNU assembler.
- Info: (dir) Top, 217 lines--Top-----
Welcome to Info version 3.12h. "C-h" for help, "m" for menu item.

Jeder Menüpunkt oder Link ist mit einem Stern gekennzeichnet.

Da **info** genauso wie der Editor **emacs** und die Kommandozeile **bash** direkt aus dem GNU-Projekt stammen, kann man diese Programme (und alle Programme die die GNU Readline-Bibliothek¹ zur Eingabeverarbeitung verwenden) mit denselben Tastaturkürzeln bedienen. Es lohnt sich daher, sich mit diesen Tastenkürzeln auseinanderzusetzen, da es die Arbeitsgeschwindigkeit deutlich beschleunigt. In der Shell kann man mit den Befehlen **set -o emacs** oder **set -o vi** den Zeileneditor wechseln.

Ohne Argument zeigt **info** alle auf dem System zur Verfügung stehenden **info**-Seiten, mit **info Befehl** springt man direkt zur **info**-Seite des jeweiligen Befehls.

1.3.3 HOWTOS, Paketdokumentation, Kerneldokumentation

Unter Linux existieren noch außer **man** und **info** eine ganze Menge von Textdateien, die eine sehr wertvolle Hilfe bieten:

Die **HOWTOS**, die, wie der Name schon sagt, eine Anleitung für eine bestimmte Aufgabe liefern. Man findet sie in folgenden Verzeichnissen:

RedHat: `/usr/share/doc/HOWTO/`, falls die HOWTOS installiert wurden, ansonsten sind sie auf der Dokumentations-CD zu finden.

SuSE: Filesystem-Standard-konform unter `/usr/share/doc/howto/`

Debian: `/usr/share/doc/HOWTO/`

In diesen Verzeichnissen und (gegebenenfalls Unterverzeichnissen) findet sich für jedes Thema eine Datei, die Anleitung zu diesem Thema bietet.

¹Test: `ldd Programm`



Beispiel: Wir betrachten das Verzeichnis `/usr/share/doc/HOWTO` und lassen uns alle Dateien, die mit einem 'N' beginnen, anzeigen:

```
# cd /usr/share/doc/HOWTO/  
/usr/doc/HOWTO/ # ls N*  
NC-HOWTO    NET3-4-HOWTO  NIS-HOWTO    Networking-Overview-HOWTO  
NCD-HOWTO   NFS-HOWTO     Net-HOWTO
```

Dies sind in diesem Fall einfache Textdateien, die sich etwa mit dem Betrachter **less** ansehen lassen.



Tip: Zum Durchsuchen *der Inhalte* aller Dateien im aktuellen Verzeichnis und dessen Unterverzeichnissen nach einem Begriff leistet **grep -r Begriff .** wertvolle Dienste.

Paketdokumentation zu den individuellen Paketen ist oft eine unschätzbare Hilfe, wenn sie vorhanden ist. Sie ist in folgenden Verzeichnissen zu finden:

RedHat: `/usr/share/doc/Paketname`

SuSE: `/usr/share/doc/packages/Paketname`

Debian: `/usr/share/doc/Paketname`

Kernel-Dokumentation in `/usr/src/linux/Documentation/` — vorausgesetzt, die Kernel-Dokumentation ist installiert — betrifft eigentlich fast ausschließlich Treiberfragen, wie etwa für Netzwerkkartentreiber oder SCSI-Treiber. Für Grafiktreiber konsultiere man die Dokumentation in `/usr/X11R6/lib/X11/doc/`.

1.4 Tastaturfunktionen der Shell

Benutzung der bash Die **bash** ist eine sehr komfortable Shell. Beherrscht man deren Tricks und Kniffe, so kann man unschlagbar schnell mit dem System umgehen, schneller als jeder andere mit der Maus.

Funktion	Tastenkombination
Laufendes Programm abbrechen	Ctrl-C
Dateiendezeichen senden (bewirkt unter anderem das Beenden der aktuellen Shell)	Ctrl-D
Bildschirmausgabe anhalten	Ctrl-S
Bildschirmausgabe fortsetzen	Ctrl-Q
History-Funktion der Shell	↑, ↓
Rückwärtssuche in der History	Ctrl-R
letztes Kommando wiederholen, das einen <i>Teilstring</i> enthält	! <i>Teilstring</i>
Ein Wort vorwärts („forward“)	Alt-F
Ein Wort rückwärts („backward“)	Alt-B
Wort rechts vom Cursor löschen („delete“)	Alt-D
Wort links vom Cursor löschen	Alt-←
Datei-/Verzeichnisnamen automatisch ergänzen	Tab

Fast alle diese Tastenkombinationen findet man im Editor `emacs` wieder. Die Wort-Bearbeitungsfunktionen erlauben ein sehr schnelles Arbeiten mit der Shell, etwa bei der Bearbeitung von Dateipfaden.

1.5 Abmelden vom System

Zum Abmelden vom System benutzen wir bequemerweise folgende Möglichkeiten:

- `exit`
- Ctrl-D

1.6 Struktur eines UNIX-Befehls

Ein UNIX-Befehl hat im Normalfall folgenden Aufbau:

Befehl	Option(en)	Argument(e)
<code>ls</code>	<code>-la</code>	<code>/home/lustig</code>
<i>Was soll getan werden?</i>	<i>Wie soll es getan werden?</i>	<i>Was soll bearbeitet werden?</i>

Befehl (im Beispiel: `ls`) Bei dem Befehl kann es sich entweder um einen internen, im Befehlsinterpreter der Shell enthaltenen, oder einen externen, als eigenes Programm

in einem Systemdirectory abgelegten, Befehl handeln.

Option (im Beispiel `-la`) Optionen werden meistens an einem vorangestellten Minuszeichen erkannt, wobei darauf zu achten ist, daß zwischen dem Minuszeichen und dem Optionskürzel kein Leerzeichen steht. Die meisten Programme aus dem GNU-Projekt unterstützen aber auch lange Optionen, die aus ganzen Wörtern bestehen und durch zwei Minuszeichen eingeleitet werden (`ls --all` statt `ls -a`). Einige wenige Programme beschreiben ihre Optionen auf andere Weise (z.B. `dd`).

Argument (im Beispiel `/home/lustig`) Das Argument eines Befehls bezeichnet normalerweise das zu bearbeitende Objekt. Üblicherweise handelt es sich hierbei um Dateien, Directories oder Geräte. Werden weder Optionen noch Argumente angegeben, werden meist Default-Werte verwendet.

Variationen Es besteht auch die Möglichkeit, daß die Optionen ihrerseits Argumente besitzen:

```
$ cc -o myprogram -l /usr/lib/mylib myprogram.c
```



Hinweis: `cc` ist in diesem Beispiel der C-Compiler, der durch Kompilieren von `myprogram.c` die Ausgabedatei `myprogram` erzeugt und die Bibliotheken aus dem (fiktiven) Verzeichnis `/usr/lib/mylib` mit einbindet.

Diese Struktur der UNIX-Befehle ist der Normalfall. Es gibt jedoch eine Reihe von *Ausnahmen*, wo diese Struktur nicht konsequent durchgehalten wird. Die exakte Struktur eines bestimmten Befehls entnehmen Sie deshalb am besten aus den `manual pages`.

Trennung von Befehlen in einer Befehlszeile Es können mehrere Befehle in einer Befehlszeile eingegeben werden, wenn sie mit einem Semikolon voneinander getrennt werden. Die Befehle werden ausgeführt, als ob sie in jeweils einer Befehlszeile abgesetzt worden wären.

Kommando1 ; Kommando2

Beispiel:

```
$ echo "Das Datum kommt: "; date
Das Datum kommt:
Fri Sep 25 11:17:59 MET 1998
```

ist ähnlich wie:

```
$ echo "Das Datum kommt: "  
Das Datum kommt:  
$ date  
Fri Sep 25 11:17:59 MET 1998
```

1.7 Herunterfahren des Systems

Wenn Sie den Rechner ausschalten wollen, dann müssen Sie vorher den Rechner herunterfahren. Dies ist notwendig, damit alle Programme ordnungsgemäß beendet werden und die Dateisysteme auf der Festplatte sich in einem definierten Zustand („clean“) befinden. Ansonsten müssen beim erneuten Hochfahren erst die Gesundheit aller Linux-Partitionen überprüft werden, was, je nach Partitionsgröße, recht lange dauern kann.

Es gibt folgende Möglichkeiten, den Rechner herunterzufahren:

- Als `root` angemeldet, geben Sie ein:

```
# init 0
```

Tip: Will man die modernen ATX-Computer dabei gleich automatisch ausschalten, so verwende man den Befehl **halt**. 

- Auf einer der virtuellen Konsolen, egal, ob angemeldet oder nicht, drücken Sie folgende (bekannte) Tastenkombination:

```
Strg-Alt-Entf
```

Der Rechner fährt dann herunter und startet unmittelbar danach neu. Direkt nach dem Neustart können Sie ausschalten, sofern der Linux-Kernel nicht schon wieder losgelaufen ist.

Das Wichtigste in Kürze

- Um an einem UNIX-System arbeiten zu können, muß man sich als erstes anmelden:
 - Eventuell eine Verbindung zum Rechner aufbauen
 - Benutzername und Paßwort eingeben

- Danach befindet sich der Benutzer in seiner Arbeitsumgebung
- Hilfe erhält man über die UNIX-Dokumentation oder über die Man-Pages mit den Befehlen **man** und **info**
- Das Paßwort wird mit dem Befehl **passwd** geändert
- Die wichtigsten Kontrollsequenzen:
 - Ctrl-C** Programmabbruch
 - Ctrl-D** Dateneingabe beenden
 - Ctrl-S** Bildschirmausgabe anhalten
 - Ctrl-Q** Bildschirmausgabe weiterlaufen lassen
- Abmelden vom Rechner:** `exit` oder **Ctrl-D** am Terminal
- Eine *Befehlszeile* besteht aus Befehl, Option und Argument. Der Befehl muß immer eingegeben werden; es ist entweder ein interner oder ein externer Befehl. Die meisten Befehle kennen Optionen. Diese Optionen werden durch ein Minuszeichen kenntlich gemacht und dienen der genaueren Spezifizierung eines Befehls. Alle Befehle die auf verschiedene Objekte anwendbar sind, haben üblicherweise Argumente. Die Argumente spezifizieren die zu bearbeitenden Objekte. Auf einer Kommandozeile können mehrere Befehle stehen. Sie sind durch Semikolon voneinander getrennt.
- Mit **init 0** können Sie als Systemverwalter den Rechner sauber herunterfahren.

Tips für die Praxis

Konsolen-Umschaltung Linux beschert einem *virtuelle Terminals*, die einen gleichzeitig unter mehreren Sitzungen am Rechner arbeiten lassen, als ob man an mehreren Terminals gleichzeitig säße. Bei fast allen gängigen Distributionen gibt es sechs Textkonsolen, die mit **Alt-F1** bis **Alt-F6** zu erreichen sind.

Ist überdies die grafische Oberfläche gestartet worden, so ist diese mit **Alt-F7** zu erreichen.



Hinweis: Will man jedoch wieder aus der grafischen Oberfläche zurück zu den virtuellen Textkonsolen, so verwendet man die Tastenkombinationen **Strg-Alt-F1** bis **Strg-Alt-F6**

Terminal blockiert Wenn das Terminal völlig blockiert scheint, überprüfen Sie die Taste **Ctrl-Q**. Möglicherweise ist nur die Bildschirmanzeige angehalten worden.

Groß-, Kleinschreibung Bei UNIX ist es wichtig, daß die Groß- und Kleinschreibung berücksichtigt wird. Dies gilt bereits beim Anmelden am Rechner.

Paßwörter Unter UNIX müssen Paßwörter mindestens sechs Zeichen lang sein. Um das Erraten Ihres Paßworts zu erschweren, sollten Sie Groß-, Kleinschreibung, Ziffern und Sonderzeichen, wie „!“ oder „“ , verwenden. Die Verwendung von Umlauten sollte unterbleiben, da englische Tastaturen keine Umlaute kennen. **Achtung: Die meisten Einbrüche auf UNIX-Systemen erfolgen aufgrund schwacher Benutzerpaßwörter. Paßwörter, die irgendeinem Wörterbuch entnommen sind, etwa der Name des Ehepartners, oder Geburtsdaten sollte man meiden.** Am besten bildet man sich eine kryptische Sequenz, die man sich obendrein nun gut merken kann. Beispiel: Sie denken an Goethes Faust:

Habe nun, ach! Philosophie. . .

und bilden daraus:

Hbn,8!Phil

Dann brauchen Sie nur noch an Faust zu denken, und schon fällt Ihnen das Paßwort wieder ein. Der Phantasie sind dabei keine Grenzen gesetzt.

Sprachumschaltung, deutsche Man-Pages In der Shell gibt es die Umgebungsvariable LANG, die dem System sagt, welche Sprache der Benutzer spricht, und aus welchem Kulturkreis er stammt. Hier eine Auswahl möglicher Sprachen:

Sprache	Land	LANG=
Deutsch	Deutschland	de_DE
Deutsch	Schweiz	de_CH
Deutsch	Österreich	de_AT
Englisch	England	en_UK
Englisch	United States	en_US

Alle verfügbaren Sprachen findet als Unterverzeichnisse von /usr/share/locale/.

Wenn man sich bei SuSE oder Debian als Systemadministrator anmeldet, so hat man standardmäßig *keine* deutsche Sprache eingestellt (Fachsprache: keine deutsche *Locale*). So stellt man sich eine deutsche Locale ein:

vorübergehend: Indem man in der aktuellen Shell die LANG-Variable setzt:

```
# export LANG=de_DE
```

Dauerhaft: Indem man in seiner Login-Startdatei diese Einstellung verankert:

```
# echo export LANG=de_DE >> ~/.profile
```

Damit hängen wir schlicht und ergreifend das Kommando von oben an diese Datei an. Die Tilde steht dabei für unser Homeverzeichnis.

Unterbrochenes Programm Wenn Sie aus Versehen ein Programm mit `Ctrl-Z` unterbrochen haben, können Sie es durch unmittelbare Eingabe des Befehls `fg` (foreground) wieder in den Vordergrund zurückholen. Wenn Sie sich abmelden und die Meldung „You have stopped jobs“ erhalten, deutet dies darauf hin, daß Sie ein oder mehrere Programme mit `Ctrl-Z` unterbrochen haben. Weiteres hierzu finden Sie in Kapitel „Prozeßverwaltung unter UNIX“.

SOLUZIONE

1.8 Übungen

1. Besorgen Sie sich Informationen über folgende Befehle und probieren Sie sie aus. Verwenden Sie hierzu die Man-Pages und ggf. **info**-Seiten:
 - **who**
 - **who am i**
 - **finger**
 - **date**
 - **pwd**
2. Lassen Sie sich alle Einträge der Man-Pages auflisten, die etwas mit den Benutzern (`user`) zu tun haben.
3. Setzen Sie sich ein neues Paßwort. Probieren Sie hierbei verschiedene Möglichkeiten aus, z.B. ein Paßwort, das nur aus einem Zeichen besteht.
4. Können Sie auch das Paßwort für einen anderen Benutzer ändern? Wenn ja, versuchen Sie, Ihrem Nachbarn ein neues Paßwort zu setzen.
5. Finden Sie heraus, wie man bei UNIX die Systemzeit neu setzt ... und versuchen Sie dann, die Zeit exakt neu zu setzen. Was dürfte hier Probleme bereiten?
Hinweis: Das Kommando **date** kann als Anhaltspunkt dienen ...

1.9 Lösungen

1. Besorgen Sie sich Informationen über folgende Befehle und probieren Sie sie aus. Verwenden Sie hierzu die Man-Pages:

who Wer arbeitet im Moment im System?

who am i Wer bin ich selbst? - falls man das vergessen haben sollte...

finger oder `finger user` Genauere Informationen über einen bestimmten oder alle Benutzer.

date Zeigt Datum und Uhrzeit an

pwd Zeigt das aktuelle Directory an (print working directory)

2. Lassen Sie sich alle Einträge der Man-Pages auflisten, die etwas mit den Benutzern (user) zu tun haben².

```
$ apropos user # Berkeley
$ man -k user # System V
```

3. Setzen Sie sich ein neues Paßwort. Probieren Sie hierbei verschiedene Möglichkeiten aus, z.B. ein Paßwort, das nur aus einem Zeichen besteht.

```
$ passwd
Changing password for ul
Old password:
Enter new password:
Password must be at least 6 characters long, password unchanged.
$
```

4. Können Sie auch das Paßwort für einen anderen Benutzer ändern?

Natürlich darf das nicht möglich sein. Nur der Superuser darf das Paßwort jedes beliebigen Benutzers verändern - aber auch er kann nicht das aktuelle Paßwort lesen.

```
$ passwd foo
passwd: Only root can specify a username
```

5. Finden Sie heraus, wie man bei UNIX die Systemzeit neu setzt ...

Für den Superuser geht das ganz einfach:

²# ist das Kommentarzeichen unter UNIX, das heißt alle Zeichen danach werden von der Shell ignoriert

```
$ date 07011200
```

stellt die Systemzeit z.B. auf 1. Juli 12 Uhr mittags. Normale Benutzer dürfen die Systemzeit nicht neu setzen.

Die Manpage von **date** ist übrigens auf die nötigsten Informationen beschränkt. Sie eignet sich sehr gut, wenn man den Befehl schon kennt und schnell mal eine bestimmte Information nachschlagen will. Zu **date** gibt es aber auch eine sehr ausführliche und gut strukturierte **info**-Seite, die es einem leicht macht den Befehl zu erlernen. Ähnliches gilt auch für viele andere Programme und Befehle.

SOLUZIONE

2 Umleitung, Pipelines und Filter

In der Regel muß ein UNIX-Befehl über die Tastatur eingegeben werden. Informations- und Fehlermeldungen erscheinen am Bildschirm. Dieses Kapitel beschreibt das Konzept von Ein- und Ausgabekanälen und wie Sie deren Arbeitsweise Ihren Bedürfnissen anpassen.

2.1 Lernziele

In diesem Kapitel lernen Sie Folgendes:

- Das Konzept von Standardeingabe, Standardausgabe und Standardfehlerausgabe kennen
- Die Ausgabe von Befehlen in Dateien umzulenken
- Die Eingabe für Befehle aus einer Datei zu holen
- Mit Pipelines mehrere Kommandos hintereinanderschalten
- Mit **more** und **less** Ausgaben seitenweise anzuzeigen
- Mit **grep** die Inhalte von Dateien zu durchsuchen
- Mit **sed** Datenströme nichtinteraktiv zu bearbeiten (Löschen, Ersetzen von Textmustern, etc.)
- Mit **tr** Zeichen zu ersetzen und zu löschen
- Mit **cut** und **awk** Spalten aus einer Tabelle ausschneiden
- Mit **sort** Dateien zeilenweise zu sortieren
- Mit **wc** die Zeilen, Wörter oder Zeichen einer Datei zu zählen
- Mit **head** bzw. **tail** den Dateianfang, bzw. das Dateiende auszugeben

2.2 Das Kanalkonzept von UNIX

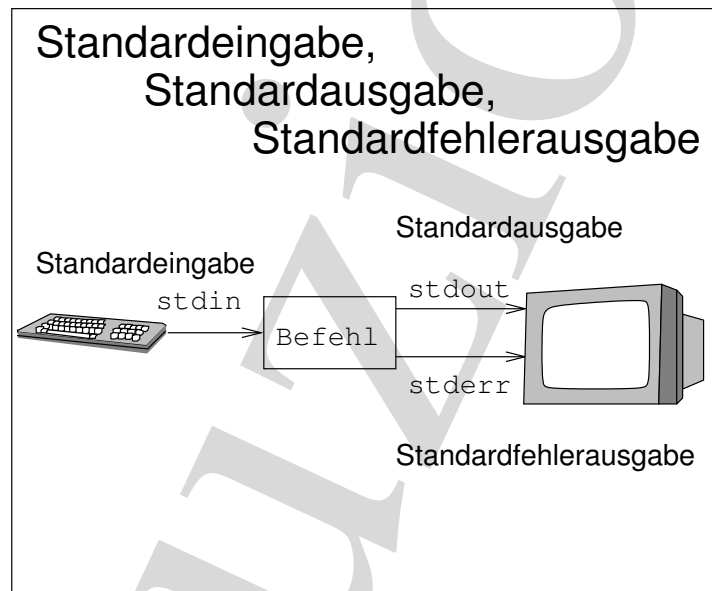
Alle UNIX-Befehle werden normalerweise über die Tastatur eingegeben. Ergebnisse sowie Fehlermeldungen werden am Bildschirm angezeigt. Es gibt unter UNIX also bei jedem Befehl drei Kanäle, die standardmäßig bestimmten Geräten zugeordnet sind:

Standardeingabe (*stdin*) *stdin* ist standardmäßig der Tastatur zugeordnet. Die Kanalnummer ist 0.

Standardausgabe (*stdout*) `stdout` ist standardmäßig dem Bildschirm zugeordnet.
Die Kanalnummer ist 1.

Standardfehlerausgabe (*stderr*) `stderr` ist ebenfalls dem Bildschirm zugeordnet.
Die Kanalnummer ist 2.

Die Kanalnummern 0, 1 und 2 sind eigentlich *Datei-Deskriptoren*, über die in Programmen offene Dateien (und auch Geräte) angesprochen werden. Für C-Programme sind diese Deskriptoren nichts anderes als eine Hausnummer einer geöffneten Datei. Jede Datei, die geöffnet wird, bekommt so einen Deskriptor. Die eben erwähnten Standard-Dateideskriptoren 0, 1 und 2 haben dadurch eine Sonderrolle, da sie immer zur Verfügung stehen.



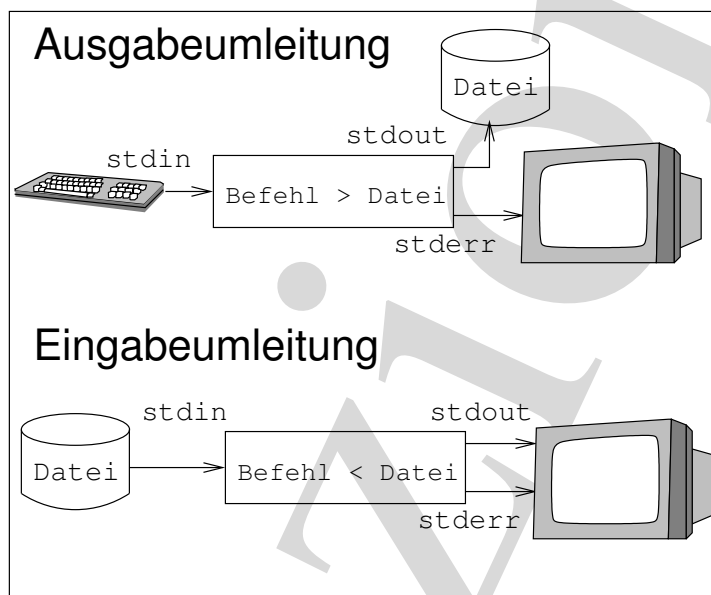
2.3 Umleitungen


2.3.1 Umleiten der Ausgabe in eine Datei

Die Ein/Ausgabekanäle eines Befehls können bei Bedarf in Dateien umgeleitet werden. Diesen Vorgang bezeichnet man auch als *I/O-Redirection*. Die Umleitung geschieht durch Eingabe einer spitzen Klammer (>) und der Zielfeile bzw. (<) und der Quelldatei.

Syntax:

Umleiten von	Befehl
stdin	<code>kdo < datei</code>
stdin	<code>kdo << EOF</code>
stdout	<code>kdo > datei</code> (überschreiben) <code>kdo >> datei</code> (anhängen)
stderr	<code>kdo 2> datei</code> (überschreiben) <code>kdo 2>> datei</code> (anhängen)



 *Beispiel:*

```
$ cd /home/lustig/asterix/figuren
$ ls > asterix_fig
$ cat asterix_fig
asterix
asterix_fig
idefix
miraculix
```

 *Beispiel:* Ablegen einer Textzeile in die Datei `status.list`:

```
$ echo "Datei der_kupferkessel ist fertig" > status.list
$ cat status.list
Datei der_kupferkessel ist fertig
```

2.3.2 Umleiten von Kanal 1 und 2 in dieselbe Datei

Man kann die Standardausgabe und die Fehlerausgabe durch folgende Konstruktion in dieselbe Datei umleiten:

```
kdo > datei 2>&1
```

Die Standardausgabe wird dabei zunächst nach `datei` umgeleitet. Dann wird die Fehlerausgabe in denselben Kanal geleitet wie die Standardausgabe, also nach `datei`.


2.3.3 Bestehende Dateien erweitern

Wenn Sie eine bestehende Datei nicht überschreiben, sondern die Bildschirmausgabe anhängen wollen, so verwenden Sie folgenden Befehl:

```
kdo >> datei
```

Beispiel: Anfügen weiterer Zeilen zur Datei `figuren_liste`. 

```
$ cd /home/lustig/asterix
$ cat >> figuren_liste
caesar
zenturix
[Ctrl]-[D]
```

Ein weiteres Anwendungsbeispiel: 

```
$ echo "Datei asterix_und_kleopatra ist auch fertig" >> status.list
```

Hierbei wird der in Anführungszeichen stehende Text an die Datei `status.list` angehängt.

2.3.4 Eingabeumleitung bis Dateiondezeichen

In einer Shell kann man Kommandos interaktiv per Tastatur in die Standardeingabe schreiben. Will man das nichtinteraktiv gestalten, ohne extra eine Datei anlegen zu müssen, so verwendet man den Doppelpfeil nach links:

Beispiel:

```
$ cat << DATEIENDE
> Dies sind
> einige Zeilen,
> die der Standardeingabe
> von cat per Eingabeumleitung
> uebergeben wurden.
> DATEIENDE
Dies sind
einige Zeilen,
die der Standardeingabe
von cat per Eingabeumleitung
uebergeben wurden.
```

Der Vorteil dieser Form der Eingabeumleitung ist, daß sie sich ohne weiteres in einem Shell-Skript verwenden läßt, ohne daß eine extra Datei angelegt werden muß, in der die Eingaben abgelegt werden.

2.4 Der Pipe-Mechanismus

2.4.1 Die Pipe zur Verbindung von Befehlen

Soll die Bildschirmausgabe eines Befehls mit einem anderen Befehl weiterverarbeitet werden, kann man die Ausgabe des ersten Befehls direkt an die Eingabe des zweiten Befehls weiterleiten.

Beispiel: Sie möchten gerne wissen, wieviele Benutzer sich momentan im System befinden. Dieses Problem könnte folgendermaßen gelöst werden:

```
$ who > user.dat
$ wc -l user.dat
6 user.dat
$ rm user.dat
```

wc -l ist ein Befehl, mit dem die Anzahl der Zeilen von der Standardeingabe oder aus einer Datei gezählt wird. Mehr Informationen hierzu weiter hinten in diesem Kapitel.

Umleitung, Pipelines und Filter

Lösung mit Pipe Eleganter ist es, wenn Sie die Aufgabe lösen, ohne eine Hilfsdatei (user.dat) zu erzeugen, die Sie danach doch wieder löschen müssen. Dies geschieht mit der Anweisung:

```
kdo1 | kdo2
```

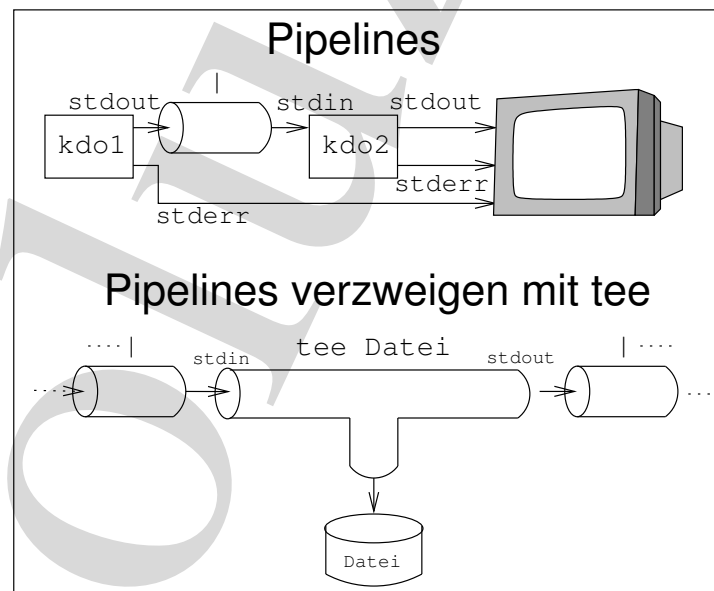
Das Symbol | wird *Pipe* genannt und verbindet die beiden Befehle auf folgende Weise:

- Die Standardausgabe des ersten Befehls wird mit der Standardeingabe des zweiten Befehls verknüpft. Das heißt, daß der zweite Befehl erst dann fertig ist, wenn die Ausgabe des ersten Befehls abgeschlossen ist.
- Dies funktioniert nur dann, wenn **kdo1** tatsächlich Ausgaben auf Kanal 1 produziert und **kdo2** Eingaben von Kanal 0 einliest. Sonst ist eine Pipe sinnlos.

Beispiel: Anzahl der eingeloggtten Benutzer

```
$ who | wc -l  
6
```

Dieses Verfahren kann auch für mehrere Befehle angewandt werden, wenn der Datenstrom mehrere Verarbeitungsschritte durchlaufen soll.



2.4.2 tee zum Erstellen von Protokollen

Der **tee**-Mechanismus unter UNIX stellt eine Kombination von Umleitung und Piping dar.

Er bewirkt, daß die Ausgabe eines Befehls standardmäßig am Bildschirm angezeigt und zusätzlich in einer Datei abgelegt wird. Dies wird mit dem Befehl **tee** erreicht:

```
tee datei
```

tee verdoppelt also den auf Kanal 0 eingelesenen Datenstrom, sowohl auf Kanal 1 wie in die angegebene Datei.



Beispiel:

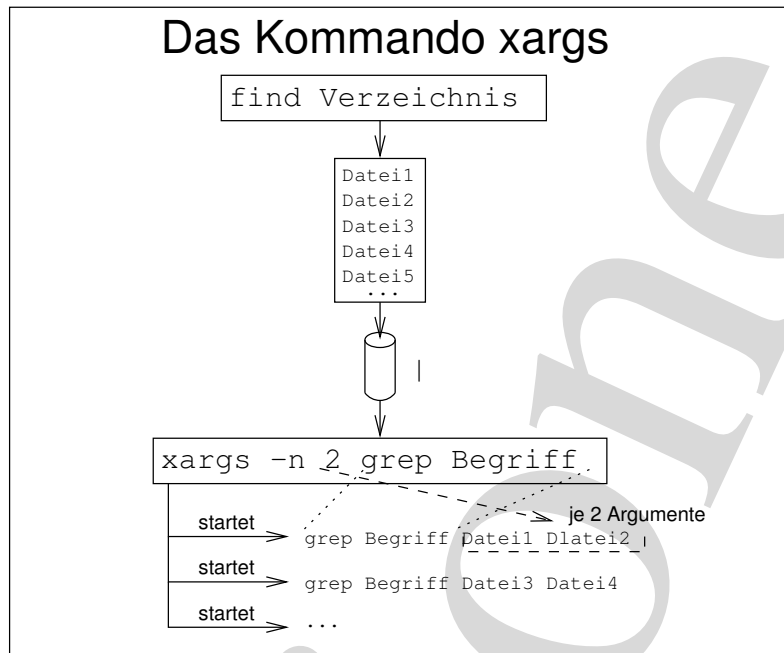
```
$ ls | tee dateien.lis
asterix
bin
dateien.lis
charly_brown
tim
$ cat dateien.lis
asterix
bin
dateien.lis
charly_brown
tim
```



Tip: Will man das Zwischenergebnis eines **tee**-Befehls nicht auf die Standardausgabe, sondern auf das *aktuelle Terminal* schreiben, so gebe man als Ausgabedatei die Gerätedatei `/dev/tty` an, die immer das gerade aktuelle Terminal repräsentiert. Dies ist immer dann sinnvoll, wenn das Endergebnis in eine Datei gespeichert wird, Sie aber Zwischenergebnisse in der Pipeline auf dem Terminal mitverfolgen wollen.

2.4.3 xargs — Pipe-Datenstrom als Kommandoargumente

In manchen Fällen will man die Daten aus der Pipe als Argumente für ein weiteres Kommando benutzen. In diesem Fall hilft der Befehl **xargs** weiter.




Syntax:

xargs [*Optionen*] [*Kommando*] [*Kommandooptionen*]

Wichtige xargs-Optionen:

Option	Bedeutung
-n <i>maxargs</i>	Maximale Anzahl von Argumenten pro Kommandoaufruf.
-p	Interaktiver Modus, Der Benutzer wird vor jeder Kommandoausführung gefragt.

Das Kommando **xargs** wird häufig in Verbindung mit dem Befehl **find** eingesetzt, um auf eine Anzahl gefundener Dateien ein bestimmtes Kommando anzuwenden.

Beispiel: Betrachten Sie zum Vergleich eine Pipeline mit und eine ohne den Befehl **xargs**, um sich den Unterschied zu verdeutlichen: 

```
$ cd /etc/pam.d
$ ls
chfn  ftp    other  ppp    rlogin  samba  sshd   sudo   xlock
chsh  login  passwd  rexec  rsh     squid  su     xdm    xscreensaver
$ ls | wc -l
```

```
18
$ ls | xargs wc -l
  6 chfn
  6 chsh
 13 ftp
 12 login
  ...
  2 xscreensaver
114 total
```

Bei der Eingabe `ls | wc -l` zählt `wc` die Anzahl der von der Pipe übergebenen Dateinamen. Bei der Eingabe `ls | xargs wc -l` hingegen werden die Dateinamen dem Befehl `wc` als Argumente übergeben, so als hätten Sie geschrieben `wc -l chfn chsh ftp login ...`, oder `wc -l `ls``.



Beispiel: Bei Programmabstürzen entstehen auf manchen Linux/UNIX-Systemen, die entsprechend konfiguriert sind, sogenannte `core`-Dateien mit einem Speicherabzug des Programms zum Zeitpunkt des Absturzes. Da diese Dateien eigentlich nur für Entwickler interessant sind und für den Benutzer und Administrator eher lästig, löscht man diese gelegentlich. Da wir in diesem Beispiel die volle Kontrolle darüber behalten wollen, *welche* Dateien gelöscht werden und was genau geschieht, wählen wir mit der `xargs`-Option `-p` den interaktiven Modus:

```
$ find . -name core | xargs -p rm -f
rm -f ./texwork/LPI/chapters/overview/core ./core ?...y
```

2.4.4 Named Pipes

Named Pipes sind Pipelines in Dateiform, und werden immer dann verwendet, wenn die zu verbindenden Prozesse nicht innerhalb derselben Shell arbeiten sollen. Was ein Prozeß in die Pipe schreibt, kann der andere lesen.



Beispiel: Zwei Shells kommunizieren über eine Named Pipe:

Shell 1

```
$ mknod Rohr p # Jetzt den Befehl in Shell 2 eingeben!
$ cat > Rohr
Dies ist ein Test,
ob die Rohrpost funktioniert
AAA
```

```
BBB
CCC
Strg-D
$
```

Shell 2

```
$ cat Rohr
Dies ist ein Test,
ob die Rohrpost funktioniert
AAA
BBB
CCC
$
```

Tip: **mkfifo** ist eine Abkürzung für den obigen **mknod**-Befehl. Argument von **mkfifo** ist lediglich der Name der zu erstellenden Named Pipe. ☆

2.5 Der Einsatz von Filtern in der Shell

2.5.1 Was ist ein Filter?

Ein Filter ist ein UNIX-Programm, das Daten einliest, bearbeitet und wieder ausgibt. Ein Filter ist für die Verwendung in einer Pipe besonders geeignet, da er - wenn man keine Datei als Argument angibt - die Eingabe von `stdin` erwartet und die Ausgabe auf `stdout` schreibt. Ein typisches Filterprogramm haben Sie bereits mit dem Kommando **more** kennengelernt.

Beispiel:

```
$ ls -R | more
```

Die Ausgabe des Befehl **ls -R** wird an den Befehl **more** übergeben und bildschirmweise ausgegeben.

Weitere Filterprogramme sind:

grep Suchen nach Textmustern (*reguläre Ausdrücke*)

sed Suchen und Ersetzen, Zeilen löschen

tr (translate) ersetzt einzelne Zeichen anhand einer Übersetzungsliste.

cut und **awk** Spalten ausschneiden

sort Sortieren von Daten

wc Zählen von Buchstaben, Wörtern und Zeilen

2.5.2 Suchen mit regulären Ausdrücken — **grep**

Nun ein Filter, der in der Praxis von höchstem Nutzen ist. Mit **grep** kann man *Inhalte* von Textdateien nach Textmustern durchsuchen. Die Standardsprache zur Darstellung von Textmustern nennt man *reguläre Ausdrücke*.

Will man etwa herausfinden, in welcher Dokumentations-Textdatei ein gewisser Chipsatz einer Netzwerkkarte vorkommt, so kann man dies mit Hilfe von **grep** erledigen.

Syntax:

```
grep [ Option(en) ] regulärer Ausdruck [ Datei(en) ]
```

Die Datei *datei* enthält den zu durchsuchenden Text, *regulärer Ausdruck* ist das Muster, wonach gesucht wird. Wird **grep** in einer Pipe verwendet, fehlt die Angabe der Quelldatei. Der Befehl **grep** gibt *alle Zeilen* aus, in denen der reguläre Ausdruck gefunden wurde.

Der einfachste Fall eines regulären Ausdrucks besteht aus dem gewünschten Suchbegriff.



Beispiele:

1. Sie wollen prüfen, ob der Benutzer *lustig* am System arbeitet:

```
$ who | grep lustig  
lustig tty04 Aug 11 12:14
```

2. Sie wollen die Benutzerdaten des Benutzers *foo* anzeigen lassen:

```
$ grep foo /etc/passwd  
foo:x:504:504:::/home/foo:/bin/bash
```

3. Sie wollen herausfinden, welche *HOWTOS* etwas mit Token-Ring zu tun haben:

Umleitung, Pipelines und Filter

```
$ cd /usr/doc/HOWTO/
$ grep -r token . | grep ring
./BootPrompt-HOWTO: there is a '%s' token in the string, the
./Ethernet-HOWTO: support token ring frames. It 'seemed lik
./Ethernet-HOWTO: since it would enable token ring shops to
./Ethernet-HOWTO: frame types couldn't coexist on a network,
./Ethernet-HOWTO: To support token ring requires more than o
./Ethernet-HOWTO: it also requires writing the source routin
./Ethernet-HOWTO: and has worked with IBM ISA and MCA token
./Ethernet-HOWTO: The present token ring code has been inclu
./Hardware-HOWTO: · Any IBM tokenring card not using DMA
./NET-3-HOWTO: token ring do require the gateway to be speci
./NET-3-HOWTO: Configuration of token ring is identical to t
./NET3-4-HOWTO: token ring do require the gateway to be spec
./NET3-4-HOWTO: Configuration of token ring is identical to
./PCMCIA-HOWTO: In order to use a PCMCIA token ring adapter,
./PCMCIA-HOWTO: · The token-ring client requires that the k
./PCMCIA-HOWTO: · The driver used by the IBM and 3Com token
./Spanish-HOWTO: tarjetas RDSI, Frame Relay, redes locales e
./mini/INDEX: How to use token ring cards
./mini/INDEX.html:<DD>How to use token ring cards
./mini/Token-Ring: This adapter will, in fact, work fi
./mini/Token-Ring: These instructions are for patching
./mini/Token-Ring: token ring support.
./mini/Token-Ring: 2.0.33. There have been many improvement
grep: ./Filesystems-HOWTO.txt: Permission denied
./mini/Token-Ring: Also, you will not have to patch a Linux
...
```

Erklärung: Zuerst wird `/usr/doc/HOWTO` zum aktuellen Verzeichnis gemacht. Anschließend werden die *Inhalte* aller Dateien im aktuellen Verzeichnis `.` und dessen Unterverzeichnissen (Option `-r`) nach dem Wort `token` durchsucht. *Achtung:* Hier ist ein Verzeichnis als Argument übergeben worden, das ist aber nur mit der Option `-r` sinnvoll! Nochmals: **grep** durchsucht *Dateiinhalte*!

Ein regulärer Ausdruck kann auch Metazeichen enthalten, die den Suchbegriff entweder allgemeiner oder konkreter definieren. Folgende Tabelle enthält die wichtigsten Metazeichen. Durch eine beliebige Zusammensetzung dieser Metazeichen und ASCII-Zeichen bildet man reguläre Ausdrücke, die auf das gewünschte Textmuster passen, das es zu finden gilt.

Achtung: Die Regulären Ausdrücke sind ein fundamentales Grundprinzip! Sie werden nicht nur von **grep**, sondern auch von den UNIX-Editoren **sed**, **vi** und **emacs**



2.5 Der Einsatz von Filtern in der Shell

verwendet, sowie von Programmiersprachen wie **awk**, **python** und **perl**. Es lohnt sich daher, diese regulären Ausdrücke zu beherrschen, da sie für die automatische Textbe- und -verarbeitung äußerst nützlich sind!

Die wichtigsten Metazeichen in einfachen regulären Ausdrücken

Zeichen	Bedeutung	Beispiel
.	Ein einzelnes beliebiges Zeichen	r.x
[]	Genau ein Zeichen aus der Klammer	[0-9], [0123456789]
[^]	Genau ein Zeichen außer denen in der Klammer	[^0-9],[^A-Za-z]
*	Beliebige Wiederholung des voranstehenden Zeichens oder Musters	.*, a*, [0-9]*
^	Als erstes Zeichen eines Musters: Das Muster steht am Anfang der Zeile	^drwx
\$	Als letztes Zeichen eines Musters: Das Muster steht am Ende der Zeile	ix\$

☞ **Hinweis:** Wird ein **grep**-Befehl mit Metazeichen im Suchmuster eingegeben, die auch in der Shell als Metazeichen existieren, muß durch *Maskierung* dafür gesorgt werden, daß nicht bereits die Shell diese Zeichen interpretiert. Maskieren Sie reguläre Ausdrücke am besten immer mit einfachen Anführungszeichen. Der **grep**-Befehl kann in seiner Arbeitsweise durch Optionen beeinflusst werden. Folgende Optionen stehen zur Verfügung:

Wichtige grep-Optionen:

Option	Bedeutung
-i	Klein- bzw. Großschreibung ignorieren (ignore case)
-c	Nur die Anzahl der entsprechenden Zeilen ausgeben (count)
-l	Nur den Namen der Dateien ausgeben (list)
-v	Alle Zeilen ausgeben, die nicht dem Suchmuster entsprechen (vice versa)
-n	Ausgabe der Zeile mit Zeilennummer (number)

Beispiele:

1. Sie wollen den **passwd**-Eintrag des Benutzers **lustig** nachsehen:

```
$ grep '^lustig:' /etc/passwd
lustig:iddk7Z.k15sS:210:15:Comixfreak:/home/lustig:/bin/bash
```

Umleitung, Pipelines und Filter

2. Sie suchen alle Namen, die in `figuren_liste` mit `m` beginnen und mit `ix` enden:

```
$ grep '^m.*ix$' asterix/figuren_liste
majestix
miraculix
methusalix
```

Nachdem wir die grundlegendsten Metazeichen in regulären Ausdrücken kennen gelernt haben, wollen wir im Folgenden eine Zusammenfassung der Syntax regulärer Ausdrücke betrachten.

Achtung: Leider verwenden die genannten Programme verschiedene Versionen regulärer Ausdrücke, so daß es leicht zu Fehlern kommen kann, wenn man die jeweilige Syntax nicht richtig beherrscht!



Übersicht über reguläre Ausdrücke:

UNIX-Filter	ed, vi, sed, grep	awk, egrep	Shell
Zeilenanfang	<code>^</code>	<code>^</code>	
Zeilenende	<code>\$</code>	<code>\$</code>	
ein beliebiges Zeichen	<code>.</code>	<code>.</code>	<code>?</code>
beliebige Zeichenfolge	<code>.*</code>	<code>.*</code>	<code>*</code>
eines der Zeichen a, b, c	<code>[a-c]</code>	<code>[a-c]</code>	<code>[a-c]</code>
ein Zeichen, jedoch nicht a, b oder c	<code>[^a-c]</code>	<code>[^a-c]</code>	<code>[!a-c]</code>
0 oder mehr x	<code>x*</code>	<code>x*</code>	
1 oder mehr x	<code>x\+</code>	<code>x+</code>	
0 oder 1-mal x	<code>x\?</code>	<code>x?</code>	
genau n-mal	<code>\{n\}</code>	<code>{n}</code>	
mindestens n-mal	<code>\{n,\}</code>	<code>{n,}</code>	
mindestens n-mal, höchstens h-mal	<code>\{n,h\}</code>	<code>{n,h}</code>	
gruppiert mehrere Zeichen	<code>\(...\)</code>	<code>(...)</code>	
Wiederverwendung n-te Gruppe (n=0..9)	<code>\n</code>	<code>\n</code>	
ODER-Verknüpfung	<code>...\ ...</code>	<code>... ...</code>	
maskiert Metazeichen x	<code>\x</code>	<code>\x</code>	<code>\x</code>
maskiert gesamten String			<code>'...'</code>

- ☞ *Hinweis:* Die GNU-Versionen (Linux) der oben genannten Programme beherrschen von Haus aus die **egrep**-Syntax, jedoch mit vorangestellten Backslashes. Das Kommando **egrep** ist die *erweiterte* Form von **grep**, die mächtigere Suchmuster unterstützt, die Backslashes vor den Metazeichen können weggelassen werden.

2.5.3 Suchen, ersetzen, Text bearbeiten: sed

sed steht für Stream Editor, und ist ein Editor, der als Filter arbeitet. Glücklicherweise ist seine Syntax **grep** oder **vi** sehr, sehr ähnlich:

Syntax:

```
sed -e 'Ausdruck 1' [ -e 'Ausdruck 2' ... ] [Datei]
```

Läßt man die Datei weg, so arbeitet **sed** als reiner Filter, sonst wird der Inhalt der Datei verarbeitet, und auf die Standardausgabe ausgegeben. Sind mehrere Ausdrücke angegeben, so werden sie nacheinander von links nach rechts abgearbeitet.

- ✎ *Beispiel:* Löschen der dritten Zeile:

```
$ sed -e '3d'
eins
eins
zwei
zwei
drei
vier
vier
Strg D
```

Der Befehl **3d** ist identisch mit dem **vi**-Befehl **:3d** im Last-Line-Modus


- ✎ *Beispiel:* Löschen der Zeile 1 bis zum ersten Auftreten des Wortes STOP:

```
$ sed -e '1,/STOP/d'
eins
zwei
drei
STOP
vier
```

Umleitung, Pipelines und Filter

```
vier
fuenf
fuenf
Strg-D
```

Zwischen den beiden Schrägstrichen kann ein beliebiger regulärer Ausdruck (=grep-Suchmuster) stehen.


Beispiel: Löschen der Zeilen 2-4: 

```
$ sed -e '2,4d'
eins
eins
zwei
drei
vier
fuenf
fuenf
Strg-D
```

Beispiel: Bereiche ausgeben: In unserem Beispiel wollen wir alle Benutzer mit einer dreistelligen Benutzernummer oder Gruppennummer größer 500 ausgeben lassen. 

```
$ sed -e '/:[5-9][0-9][0-9]:/!d' /etc/passwd
foo:x:501:501::/home/foo:/bin/bash
bar:x:502:502::/home/bar:/bin/bash
anonymous:x:504:504::/home/anonymous:/bin/bash
postfix:x:505:505::/var/spool/postfix:/bin/false
```

Zwischen den beiden Schrägstrichen befindet sich ein regulärer Ausdruck, das Suchmuster. Dieses hat dieselbe Syntax wie die Suchmuster des **grep**. Das Ausrufezeichen danach bewirkt, daß die nachfolgende Operation **d** (Delete: Zeile löschen) für alle Zeilen ausgeführt wird, auf die das Muster *nicht* paßt.

Beispiel: Suchen und Ersetzen: Wir wollen alle Doppelpunkte aus /etc/passwd durch Tabulatorzeichen ersetzen: 

```
$ sed -e 's/:/Strg-V↔/g' /etc/passwd
root  x      0      0      root  /root  /bin/bash
bin   x      1      1      bin   /bin
daemon x      2      2      daemon /sbin
```

2.5 Der Einsatz von Filtern in der Shell

```
adm    x      3      4      adm    /var/adm
lp     x      4      7      lp     /var/spool/lpd
sync  x      5      0      sync  /sbin /bin/sync
...
```

Dieser Befehl ist dem **vi**-Befehl gleich: **s** steht für **substitute** — auf Deutsch: ersetzen — und dann kommt das Suchmuster, das ein beliebiger regulärer Ausdruck sein darf. Die Tastenkombination **Strg-V** braucht man hier, um das Tabulatorzeichen vor der Shell zu verstecken. Die Option **g** (global) bewirkt, daß alle Vorkommnisse *in einer Zeile* ersetzt werden. Steht statt dessen eine Zahl *n*, so wird nur das *n*-te Vorkommnis ersetzt.



Beispiel: Suchen und Ersetzen: Nehmen wir an, unsere Eingabe bestehe aus Namen von Ehefrauen und deren Ehemännern vor der Hochzeit, also z.B.:

```
Gabi Wallenstein Heiner Mueller
```

Und wir wollen diese Beiden nun mit **sed** verheiraten, so daß folgendes herauskommt:

```
Gabi und Heiner Mueller
```

Dann benutzen wir die Möglichkeiten der Klammerung in regulären Ausdrücken:

```
$ sed -e 's/\([a-zA-Z]*\) [a-zA-Z]* \([a-zA-Z]* [a-zA-Z]*\)*/\
> \1 und \2/'
```

```
Gabi Wallenstein Heiner Mueller
```

```
Gabi und Heiner Mueller
```

```
Strg-D
```

Die mit einem Backslash maskierten runden Klammern fassen einen Teilausdruck zu einer *Gruppe* zusammen. Im Ersetzungstext kann dann die *n*-te Gruppe von links mit $\backslash n$ referenziert werden.

Man beachte hier den Backslash am Ende der ersten Zeile, der den Zeilenumbruch markiert. Will man das Kommando in eine Zeile schreiben, so läßt man einfach den Backslash weg.



Tip: Die Zeichenklasse „alle Buchstaben“ $[a-zA-Z]$ läßt sich vereinfacht auch $[a-Z]$ schreiben.

2.5.4 tr — Zeichenersetzung

Das Kommando **tr** ist ein Filter zur zeichenorientierten Textmanipulation. Es wandelt Zeichen ineinander um, löscht Zeichen oder reduziert mehrere Vorkommnisse eines

Zeichens auf ein Zeichen.

Syntax:

```
tr Zeichenmenge1 Zeichenmenge2  
tr -s Zeichenmenge1  
tr -d Zeichenmenge1  
tr -ds Zeichenmenge1 Zeichenmenge2
```

Das Filterprogramm **tr** ist im wahrsten Sinne des Wortes ein Filter. Es liest stur von der Standardeingabe und gibt die Ergebnisse auf die Standardausgabe aus. Man benutzt es folglich in der Praxis fast immer zusammen mit Ein/Ausgabeumleitungen oder Pipelines.

Wichtige Optionen:

- ohne Option Übersetzt die Zeichen der Zeichenmenge 1 in die entsprechenden Zeichen der Zeichenmenge 2
- s (squeeze) Reduziert mehrere im Eingabedatenstrom vorkommende Zeichen aus Zeichenmenge 1 auf ein solches Zeichen
- d (delete) Entfernt alle Zeichen aus Zeichenmenge 1
- ds (delete squeeze) Löscht zuerst alle Zeichen aus Zeichenmenge 1 und reduziert dann alle mehrfach vorkommenden Zeichen aus Zeichenmenge 2 durch ein Zeichen.
- c (complement) Ersetzt die Zeichenmenge 1 durch alle Zeichen, die sich *nicht* in dieser Zeichenmenge befinden

Darstellung der Zeichenmengen:

Beispiel	Beschreibung
'abcABC123'	Einzelne Zeichen. Im Beispiel die Buchstaben A, B und C in Groß- und Kleinschreibung sowie die Ziffern 1 bis 3. Für das Tabulatorzeichen schreibe man <code>\t</code> , und für das Newline-Zeichen <code>\n</code> . Weitere Sonderzeichen siehe man tr .
'a-zA-Z0-9'	ASCII-Zeichenbereiche entsprechend der ASCII-Tabelle. In unserem Beispiel alle Groß- und Kleinbuchstaben, sowie die Ziffern 0 bis 9.
'[X*5]'	Wiederholte Zeichen; kann nur in Zeichenmenge 2 vorkommen. Das Beispiel steht für fünf mal das Zeichen X.
'[X*]'	Wiederholte Zeichen; kann nur in Zeichenmenge 2 vorkommen. Dieser Ausdruck steht für so viele Vorkommnisse des Zeichens X, wie es Zeichen in Zeichenmenge 1 gibt.
'[:alnum:]'	Zeichenklasse. Das Beispiel steht für alle alphanumerischen Zeichen und ist gleichwertig mit <code>a-zA-Z0-9</code> . Die restlichen Zeichenklassen lese man unter man tr oder man grep nach.

★ *Tip:* Die ASCII-Tabelle kann man mit **man ascii** nachschlagen.

✎ *Beispiele:*

Zeichenersetzung:

```
$ tr abc xyz
bach-konzert
yxzh-konzert
```

Großbuchstaben in Kleinbuchstaben umwandeln:

```
$ tr A-Z a-z
Dies ist ein Normaler Text
dies ist ein normaler text
DIES SIND GROSSBUCHSTABEN
dies sind grossbuchstaben
```

Dasselbe läßt sich mit

```
$ tr '[:upper:]' '[:lower:]'
```

oder aber mit

Umleitung, Pipelines und Filter

```
$ tr ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
```

erreichen.

Alle Nullbytes entfernen:

```
$ tr -d '\000'
```

Alle nicht druckbaren Zeichen entfernen:

```
$ tr -cd '[:print:]'
```

Leerzeilen (=mehrfach vorkommende Newline-Zeichen) entfernen:

```
$ cat << ENDE | tr -s '\n'
> Dies ist ein Text
>
>
>
> mit ein paar Leerzeilen
> ENDE
```

Dies ist ein Text
mit ein paar Leerzeilen

Jedes Wort in einem Text in eine einzelne Zeile umbrechen:


```
$ tr -cs 'a-zA-Z0-9' ' [\n*] '
Dies ist ein normaler Text. Strg D
Dies
ist
ein
normaler
Text
```

„Geheimsprache“ kodieren (umgekehrtes Alphabet):

```
$ tr a-z zyxwvutsrqponmlkjihedbca
dies ist eine verschleierte nachricht
wrvh rhe vrmv bvihxsovrview mzxsirxse
```

„Geheimsprache“ dekodieren (umgekehrtes Alphabet):

```
$ tr zyxwvutsrqponmlkjihedbca a-z
wrvh rhe vrmv bvihxsovrview mzxsirxse
dies ist eine verschleierte nachricht
```

Hinweis: Dies ist nur Verschleierung, keine Verschlüsselung. Solche Verfahren sind leicht zu knacken. Für ernsthaftere Fälle benutze man richtige Verschlüsselungsverfahren! 

Fragen

1. Welche **tr**-Optionen erledigen Folgendes?

(a) Löschen der in Zeichenmenge 1 angegebenen Zeichen?

(b) Wiederholte Vorkommnisse der in Zeichenmenge 1 angegebenen Zeichen auf ein Zeichen reduzieren?

(c) Löschen der in Zeichenmenge 1 angegebenen Zeichen?

(d) Komplement der Zeichenmenge 1, d.h. alle Zeichen *außer* denen in Zeichenmenge 1?

2. Formulieren Sie folgende **tr**-Zeichenmengen:

(a) Die Großbuchstaben von A bis K und von L bis Z sowie die Ziffern 1,3,5 und 7 bis 9

(b) Geben Sie Zeichenmenge 2 an, so daß Sie jedes Zeichen aus Zeichenmenge 1 durch ein Newline-Zeichen ersetzen:

2.5.5 Spalten ausschneiden (**cut**, **awk**)

Möchte man aus einer Datei oder Befehlsausgabe nur bestimmte Teile jeder Zeile sehen, eignet sich das Kommando **cut**.

```
cut -f feld(er) [datei(en)]
```

schneidet die angegebenen *Felder* aus.

Achtung: Als Feldtrenner dienen *einzelne Zeichen*! Werden etwa die Spalten einer Tabelle mit mehreren Leerzeichen aufgefüllt, wird die Spaltentrennung mit **cut** zu einem frustrierenden Unterfangen. In diesem Falle verwendet man entweder **tr**, um

Umleitung, Pipelines und Filter

mehrfache Leerzeichen durch ein einziges zu ersetzen, oder man verwendet **awk**, das beliebige reguläre Ausdrücke als Feldtrenner verwenden kann. Falls man nichts anderes einstellt, wird *whitespace*, also eine beliebig lange Kette aus Leerzeichen und Tabulatoren, als Feldtrenner verwendet.

Wie **sed** und **perl** ist **awk** eine Programmiersprache, die zur textmusterbasierten Textverarbeitung geschaffen ist. **awk** ist mächtiger als **sed**, aber weniger flexibel als **perl**, von dem es zunehmend verdrängt wird. **awk** führt normalerweise für jede Zeile, in der ein bestimmtes *Muster* gefunden wird (es werden reguläre Ausdrücke wie bei **egrep** verwendet), eine *Aktion* aus. Wird kein Muster angegeben, so wird die Aktion für jede Zeile ausgeführt. Um einzelne Felder zu extrahieren, genügt uns folgender Aufruf:

```
awk '{ print [$Spalte] ["Text"] ... }' [Datei(en)]
```

Will man jedoch einzelne Bildschirmspalten ausschneiden, so verwendet man:

```
cut -c Spalte(n) [Datei(en)]
```

Wird **cut** in einer Pipe verwendet, fehlt die Angabe der Quelldatei. Der Befehl **cut** gibt alle angegebenen Teile jeder Zeile aus.

Wichtige Optionen des cut-Befehls:

Option	Bedeutung
<code>-c Spalte(n)</code>	Schneidet Spalten aus: Bereichsangaben 2-4 und Aufzählung 2, 4, 7 erlaubt
<code>-f Feld(er)</code>	Schneidet Felder aus: Bereichsangaben 2-4 und Aufzählung 2, 4, 7 erlaubt
<code>-d Trenner</code>	Felder sind durch Feldtrennzeichen <i>trenner</i> getrennt (Default: <code>TAB</code>) Bei Whitespace-Zeichen bitte quotieren!

Beispiele:

1. Sie wollen im Longlisting nur die Zugriffsrechte und den Dateinamen sehen:

```
$ ls -l | cut -c2-10,54-
```

oder

```
$ ls -l | awk '{ print $1 " " $9 }'
```

Hier erspart einem **awk** die Mühe, Zeichen abzuzählen.

2. Sie möchten nur die Uhrzeit aus der Datumsausgabe herausfischen:

```
$ date
Fri Oct 01 12:56:31 MET 1998
$ date | cut -d" " -f4
12:56:31
```

2.5.6 Sortieren einer Datei — `sort`

Sortieren ist im EDV-Alltag eine häufige Aufgabe:

```
sort [optionen] [datei(en)]
```

Das Dateiarargument wird wiederum weggelassen, wenn der Befehl in einer Pipe verwendet wird. Der `sort`-Befehl unterteilt jede Zeile in Felder, die durch Trennzeichen voneinander getrennt sind (Trennzeichen sind per Voreinstellung Leerzeichen und Tabulatoren). Wenn weiter nichts angegeben wird, sortiert `sort` nach dem ersten Feld in aufsteigender alphabetischer Reihenfolge. Diese Arbeitsweise kann jedoch mit Optionen verändert werden.



Beispiel:

Alphabetisches Sortieren der am System angemeldeten Benutzer:

```
$ who | sort
u1      ttyt4      Aug 10 9:10
u3      ttyt2      Aug 10 9:17
u7      ttyt3      Aug 10 9:15
```

Diese Beispielausgabe von `who` enthält folgende Felder:

- Benutzername
- Terminal
- Monat
- Tag
- Uhrzeit

Umleitung, Pipelines und Filter

Wichtige Optionen des `sort`-Befehls

Option	Bedeutung	Voreinstellung (ohne Option)
-f	Ignoriert Groß-/Kleinschreibung (fold)	Berücksichtigt Groß-/Kleinschreibung
-r	Sortiert absteigend (reverse)	Sortiert aufsteigend
-t <code>zeichen</code>	<code>zeichen</code> ist Trennzeichen	Blanks und Tabulatoren
-o <code>datei</code>	Das Ergebnis wird nach <code>datei</code> geschrieben. (Kann die gleiche Datei wie die Eingabedatei sein!)	Ergebnis wird auf Standardausgabe geschrieben
+num	Es werden <code>num</code> Felder nicht berücksichtigt, und nach dem darauf folgenden sortiert	Sortierschlüssel beginnt am Zeilenanfang
-num	<code>num</code> ist das letzte Feld, das bei der Sortierung berücksichtigt wird (nur sinnvoll, wenn durch folgende +num-Optionen weitere Sekundärschlüsselfelder spezifiziert werden)	Sortierschlüssel reicht bis zum Ende der Zeile
-b	Ignoriert führende Leerzeichen („Blanks“) am Feldanfang	Bewertet führende Blanks
-n	Sortiert Gleitpunktzahlen numerisch	Sortiert nach dem ASCII-Zeichensatz

Beispiel: Die Datei `comics_allg` im Homedirectory des Benutzers `lustig` soll sortiert werden:

vorher:

```
$ cat comics_allg
#Name Autor Anzahl des Auftretens pro Heft (Durchschnitt)
Schultze Herge 12
Haddock Herge 48
Bienlein Herge 25
Rastapopulos Herge 3
Asterix Uderzo 188
Obelix Uderzo 170
Idefix Uderzo 53
Struppi Herge 158
Majestix Uderzo 18
Ganzbaff Uderzo 4
Voelligbaff Uderzo 3
Tim Herge 175
```


Umleitung, Pipelines und Filter

```
bin:x:1:1:bin:/bin:
daemon:x:2:2:daemon:/sbin:
ftp:x:14:50:FTP User:/var/ftp:
games:x:12:100:games:/usr/games:
gopher:x:13:30:gopher:/usr/lib/gopher-data:
htdig:x:101:233:./var/lib/htdig:
lp:x:4:7:lp:/var/spool/lpd:
mail:x:8:12:mail:/var/spool/mail:
named:x:25:25:Bind User:/var/named:
news:x:9:13:news:/var/spool/news:
nobody:x:99:99:Nobody:./:
operator:x:11:0:operator:/root:
postfix:x:102:234:postfix:/var/spool/postfix:
uucp:x:10:14:uucp:/var/spool/uucp:
foo:x:501:504:./home/foo:/bin/bash
gdm:x:42:235:./home/gdm:/bin/bash
jack:x:500:505:./home/jack:/bin/bash
root:x:0:0:root:/root:/bin/bash
sympa:x:89:89:Sympa Mailing list manager:/var/lib/sympa:/bin/bash
xfs:x:100:102:X Font Server:/etc/X11/fs:/bin/false
sync:x:5:0:sync:/sbin:/bin/sync
halt:x:7:0:halt:/sbin:/sbin/halt
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
```

Hinweis: Als Feldtrenner dient hier der Doppelpunkt, was **sort** mit der Option **-t** beigebracht wurde. 

Beispiel: Wir wollen die Ausgabe des **ls**-Befehls nach der Dateigröße (Feld 4-5, numerisch) sortieren: 

```
$ ls -l /bin | sort -n +4 -5
...
lrwxrwxrwx 1 root root 8 Feb 7 20:07 nisdomainname -> hostname
lrwxrwxrwx 1 root root 8 Feb 7 20:07 ypdomainname -> hostname
lrwxrwxrwx 1 root root 14 Feb 7 20:32 zsh -> ../usr/bin/zsh
lrwxrwxrwx 1 root root 17 Feb 7 20:13 linuxconf -> ../sbin/linuxconf
-rwxr-xr-x 1 root root 2548 Okt 6 15:40 doexec
-rwxr-xr-x 1 root root 2712 Okt 3 10:25 arch
-rwxr-xr-x 1 root root 2990 Okt 3 15:07 igawk
-rwxr-xr-x 1 root root 4172 Okt 3 23:22 mktemp
-rwxr-xr-x 1 root root 4252 Okt 3 10:25 dmesg
-rwxr-xr-x 1 root root 4736 Okt 4 00:08 false
```

```
-rwxr-xr-x 1 root root 4736 Okt  4 00:08 true
-rwxr-xr-x 1 root root 5684 Okt  2 17:49 sync
-rwxr-xr-x 1 root root 6304 Okt  4 00:08 sleep
-rwxr-xr-x 1 root root 6396 Okt  4 00:08 basename
-rwxr-xr-x 1 root root 6772 Okt  4 00:08 uname
...
```

★ *Tip:* Leichter ginge das natürlich mit:

```
$ ls -S /bin
```

Falls man gar nach Datum oder Zugriffszeit sortieren will, so fährt man mit den **ls**-Optionen besser. Aber nicht jedes Kommando, das seine Ergüsse in Spaltenform ausgibt, hat eingebaute Sortierbefehle!

Verschiedene Sortiermethoden in verschiedenen Feldern Will man in einem Feld numerisch, gleichzeitig in einem anderen Feld aber lexikographisch sortieren, so reicht die oben dargestellte traditionelle Schreibweise nicht aus. Man verwendet die neue POSIX-Schreibweise mit der Option **-k** (Key=Sortierschlüssel). In dieser Schreibweise werden die Felder nicht wie oben von Null an, sondern von Eins an gezählt:

Syntax:

```
sort -k VonSpalte,BisSpalte[Art] [...]
```

VonSpalte und *BisSpalte* sind dabei die Spaltennummern der gewünschten Spalte (von Eins an gezählt!). Standard wird lexikographisch aufsteigend sortiert. Mit *Art* dagegen läßt sich angeben, wie die angegebenen Felder sortiert werden sollen:

Art	Bedeutung
b	Führende Leerzeichen („Blanks“) ignorieren
n	numerisch sortieren

✎ *Beispiel:* Wir wollen im Longlisting zuerst lexikographisch nach Benutzer- und Gruppennamen (Feld 3 und 4), und dann numerisch nach der Dateigröße (Feld 5) sortieren:

```
$ ls -al | sort -k 3,4 -k 5,5n
```

```
total 792
```

```
-rw-r--r--  1 foo      users          174 Sep 11 10:22 .bash_logout
```

Umleitung, Pipelines und Filter

```
-rw-r--r-- 1 foo users 266 Sep 11 10:22 .alias
-rw-r--r-- 1 foo users 373 Sep 11 10:22 .bash_profile
-rw-r--r-- 1 foo users 375 Sep 11 10:22 .cshrc
-rw-r--r-- 1 foo users 504 Sep 11 10:22 .bashrc
-rw-r--r-- 1 jack jack 57 Sep 11 14:57 dnetc.ini
-rw-r--r-- 1 jack jack 256 Dec 4 12:27 buff-out.rc5
-rw-r--r-- 1 jack jack 504 Dec 10 15:38 buff-in.rc5
drwxr-xr-x 2 jack jack 4096 Sep 11 10:23 docs
-rw-r--r-- 1 jack jack 6494 Feb 8 2000 dnetc.1
-rwxr-xr-x 1 jack jack 750780 Feb 8 2000 dnetc
drwxr-xr-x 3 jack users 4096 Sep 11 10:33 .
drwxr-xr-x 5 root root 4096 Dec 5 07:22 ..
```

2.5.7 Zählen von Zeilen und Wörtern — **wc**

Ein Filter, der früher im Kapitel schon behandelt wurde, ist **wc**. Dieses Kommando zählt die Zeichen, Worte und Zeilen einer Datei oder eines Datenstroms.

Syntax:

```
wc [ Option(en) ] [ Datei(en) ]
```

Die Optionen des Befehls **wc**

Option	Bedeutung
-l	Zählt nur Zeilen (lines)
-w	Zählt nur Wörter (words)
-c	Zählt nur Zeichen (characters)

Beispiel: Sie möchten wissen, wieviele Einträge sich im aktuellen Directory befinden:

```
$ ls | wc -w
7
```

2.5.8 Anfang und Ende einer Datei anzeigen — **head**, **tail**

Möchte man nur den Anfang oder das Ende einer Datei oder eines Datenstroms anzeigen, benutzt man die Kommandos

head [-zeilen] [datei]

bzw.

tail [-zeilen] [datei]

tail [+beginn] [datei]

head und **tail** zeigen die ersten bzw. letzten 10 Zeilen einer Datei oder einer Pipe an, sofern nicht über Optionen die Anzahl definiert wird. Bei **tail** kann mit Option + festgelegt werden, ab welcher Zeile bis zum Dateiende die Ausgabe erfolgen soll.

Beispiele:

1. Anzeige des root-Accounts aus der Paßwortdatei:

```
$ head -1 /etc/passwd  
root:x:0:1:Superuser:/:/bin/bash
```

2. Longlisting ohne Total-Titelzeile:

```
$ ls -l | tail +2  
drwxr-xr-x  2  u1  system   512 Aug 17 08:17:31 asterix  
-rw-r--r--  1  u1  system   367 Aug 17 08:31:39 figuren_liste
```

2.6 Das Wichtigste in Kürze

- Jeder Befehl besitzt drei Ein- und Ausgabekanäle: `stdin`, `stdout` und `stderr`. Ihnen sind die Nummern 0,1 und 2 zugeordnet. `stdin` ist normalerweise mit der *Tastatur* assoziiert, `stdout` und `stderr` mit dem *Bildschirm*.
- Mit dem Umleitungszeichen `>` kann `stdout` bzw. `stderr` wie folgt umgeleitet werden:

 `> datei` leitet die Standardausgabe um
 `2> datei` leitet die Fehlerausgabe um
- Existiert die Datei, in die umgelenkt wird, wird sie überschrieben.
- Soll die Ausgabe an eine bestehende Datei durch Umleitung angehängt werden, benutzen Sie den Doppelpfeil (`>> datei`). Die entsprechende Datei wird dann nicht überschrieben, sondern ergänzt.
- Wenn die Ausgabedaten einzelner Befehle durch andere Befehle weiterverarbeitet werden sollen, verwenden Sie die Pipe (`kdo1 | kdo2`). Das Verfahren kann in einer Kette mehrfach hintereinander angewendet werden.
- Soll die Befehlsausgabe gleichzeitig am Bildschirm angezeigt und in einer Datei abgelegt werden, verwenden sie den Befehl `tee datei`
- Das Kommando `xargs` verwandelt alle Wörter aus der Standardeingabe in Argumente eines beliebigen Kommandos. Es bietet sich besonders in Kombination mit dem Befehl `find` zur Weiterverarbeitung gefundener Dateien an.
- Besonders geeignet zur Verwendung in Pipes sind die Filterbefehle. Sie erhalten ihre Eingabe von `stdin` (sofern keine Quelldateien als Argument angegeben werden) und geben ihre Ausgabe an `stdout` aus.
- `more datei` Gibt `stdin` oder `datei` seitenweise aus und ermöglicht das Suchen innerhalb des Textes.
- `less datei` Wie oben, jedoch ist auch Zurückblättern möglich.
- `grep Optionen 'Textmuster' Datei` Durchsucht `stdin` oder `Datei` nach `Textmuster` und gibt die Zeilen aus, in denen das Muster vorkommt. Wichtige Optionen sind `-i`, `-v`, `-n`.
- `tr Menge1 Menge2` ersetzt im Eingabedatenstrom das erste Zeichen aus `Menge1` durch das erste Zeichen aus `Menge2`, das zweite Zeichen aus `Menge1` durch das zweite Zeichen aus `Menge2`, usw. Wichtige Optionen sind `-d` (delete), `-s` (squeeze) und `-c` (complement).

- **cut *felder/spalten datei*** Schneidet die angegebenen Felder oder Spalten aus *stdin* oder einer Datei aus. Wichtige Optionen sind `-f`, `-c`, `-dtrennzeichen`
- **sort *optionen datei*** Sortiert *stdin* oder *datei*. Es wird ohne Angabe von Argumenten nach dem ersten Feld in alphabetischer Reihenfolge aufsteigend sortiert. Wichtige Optionen sind `+num`, `-ttrennzeichen`, `-f`, `-r`.
- **wc *optionen datei*** Zählt in *stdin* oder *datei* Zeichen, Wörter und Zeilen. Optionen sind `-c`, `-w`, `-l`.

2.7 Tips für die Praxis

- Oft erzeugen Befehle umfangreiche Daten, die am Bildschirm angezeigt werden. Die weitaus häufigste Anwendung der Pipe ist es, eine solche Ausgabe mit **more**, **less** oder **grep** zu filtern.
- In der **bash**-Shell läßt sich mit dem Befehl **set -o noclobber** verhindern, daß existierende Dateien beim Umleiten überschrieben werden. Wird der obige Befehl abgesetzt, so wird beim Versuch, existierende Dateien zu überschreiben, eine Fehlermeldung ausgegeben.
- Manchmal will man entweder die Standardausgabe oder die Fehlerausgabe (oder beide) weder auf den Bildschirm noch in eine Datei leiten. In diesem Fall kann man die jeweilige Ausgabe in die Gerätedatei `/dev/null` umleiten. Bei dem Befehl

```
$ ls -lR 2> /dev/null
```

wird die Fehlerausgabe „weggeworfen“.

- Hilfreiche Filterbefehle sind auch **head** und **tail**, um den Anfang oder das Ende einer Datei anzuzeigen. Gerade ständig wachsende Protokolldateien lassen sich mit **tail** gut auf die letzten *n* relevanten Zeilen kürzen.
- Um Protokolldateien wie etwa `/var/log/messages` „live“ mitzuverfolgen, benutze man:

```
$ tail -f /var/log/messages
```

- Erstellen Sie sich bei der Installation von Software oder anderen langwierigen Arbeiten ein Protokoll, um zu einem späteren Zeitpunkt den Verlauf prüfen zu können. Das ermöglicht der Befehl **tee**.

- Man kann die leere Ausgabe eines leeren Kommandos benutzen, um mit Ausgabeumleitung eine Datei auf die Länge Null zu beschneiden. Dies ist etwa bei Log-Dateien sinnvoll:

```
# head /var/log/cron
CRON (08/07-23:54:20-432) STARTUP (fork ok)
CRON (08/08-08:22:04-436) STARTUP (fork ok)
root (08/08-08:30:00-969) CMD ( /sbin/rmmod -as)
CRON (08/08-16:48:40-452) STARTUP (fork ok)
root (08/08-16:50:00-786) CMD ( /sbin/rmmod -as)
root (08/08-17:00:00-939) CMD ( /sbin/rmmod -as)
root (08/08-17:01:00-943) CMD (run-parts /etc/cron.hourly)
root (08/08-17:10:00-970) CMD ( /sbin/rmmod -as)
root (08/08-17:20:00-1022) CMD ( /sbin/rmmod -as)
CRON (08/08-17:26:16-447) STARTUP (fork ok)
# >/var/log/cron
# head /var/log/cron
#
```

2.8 Übungen

1. Lassen Sie sich die Man-Pages für den **grep**-Befehl in die Datei `hilfe.dat` schreiben.
2. Erweitern Sie die Datei `hilfe.dat` um Informationen zum Befehl **sort**. Schauen Sie sich die erzeugte Datei einmal mit dem **vi** an, dann wissen Sie, weshalb man heutzutage die UNIX-Formatiertools **nroff** und **troff** nur noch in Ausnahmefällen benutzt! Testen Sie auch das Kommando

```
$ more hilfe.dat
```

3. Suchen Sie spaßeshalber, in wievielen Zeilen der Befehl **grep** in der Datei `hilfe.dat` vorkommt. Ist das Ergebnis nicht etwas ungewöhnlich?
4. Wieviele Dateien stehen im Directory `/bin`? Es soll nur die Anzahl der Dateien am Bildschirm ausgegeben werden.
5. Legen Sie den Inhalt Ihres aktuellen Directories in einer Datei ab und lassen Sie sich gleichzeitig die Anzahl der Dateien am Bildschirm ausgeben.
6. Sortieren Sie das Longlisting Ihres Homesdirectories nach der Dateigröße.
7. Lassen Sie sich nur die erste Zeile des Longlistings anzeigen.
8. Beim Longlisting **ls -lisa** wird im zweiten Feld eine Zahl angezeigt. Diese bezeichnet die Zahl der von der Datei beanspruchten Datenblöcke. Lassen Sie sich nun von einem Longlisting nur diese Zahl und diese auch nur aus der ersten Zeile anzeigen.
9. Erkundigen Sie sich nach den Benutzern, die am System arbeiten. An welchem Terminal und seit wann sind sie angemeldet? Leiten Sie die Ausgabe des entsprechenden Befehls, nach dem Terminal sortiert, in die Datei `user.dat` um.
10. Sofern Ihr UNIX eine Datei `/usr/share/dict/words` kennt: Wieviele Wörter, welche mit **b** oder **B** beginnen, enthält die Datei? Lassen Sie sich diese Wörter, umgekehrt sortiert, in eine Datei schreiben und die Anzahl dieser Wörter am Bildschirm anzeigen.
11. Welche Kommandozeile geben Sie ein, um in `/home` und all dessen Unterverzeichnissen alle Dateien mit der Endung `.bak` zu löschen? Wie heißt diese Kommandozeile, wenn:
 - Bei jeder einzelnen gefundenen Datei

- Nur einmal für den gesamten Löschbefehl

bestätigt werden soll? Machen Sie dabei ruhig von den Man-Pages Gebrauch.

12. Schreiben Sie eine Kommandozeile, in der Sie die Anzahl der Großbuchstaben in der Man-Page von **bash** zählen lassen. Gehen Sie, wie im vorigen Beispiel, wieder stufenweise vor!
13. Wieviele Kleinbuchstaben, Ziffern, Doppelpunkte und Punkte hat die Man-Page von **bash** (Tip: **tr** und **wc**)?

2.9 Lösungen

1. Man-Pages für **grep**-Befehl in die Datei `hilfe.dat` schreiben:

```
$ man grep > hilfe.dat
```

2. Datei `hilfe.dat` um Informationen zum Befehl **sort** erweitern:

```
$ man sort >> hilfe.dat
$ vi hilfe.dat
...
$ more hilfe.dat
...
```

Die Man-Pages sind mit den UNIX-Formatiertools **nroff** etc. formatiert. Die Steuerzeichen zum Fettdrucken werden im **vi** anders dargestellt als bei **more**.

3. Wie oft kommt der Befehl **grep** in der Datei `hilfe.dat` vor?

```
$ grep grep hilfe.dat | wc -l
```

Nur einmal steht das Wort `grep` als solches in der Datei (in Zeile 1), ansonsten immer mit den Steuerzeichen für Fettdruck versehen, wie zum Beispiel

```
g^Hgr^Hre^Hep^Hp
```

4. Wieviele Dateien stehen im Directory `/bin`? Es soll nur die Anzahl der Dateien am Bildschirm ausgegeben werden.

```
$ ls /bin | wc -l
```

5. Inhalt Ihres aktuellen Directories in einer Datei ablegen. Anzahl der Dateien am Bildschirm anzeigen:

```
$ ls -l | tee alle_meine_dateien | wc -l
```

6. Longlisting des Homedirectories nach Größe sortieren:

```
$ ls -l ~ | sort +4n
```

7. Nur erste Zeile vom Longlisting anzeigen:

Umleitung, Pipelines und Filter

```
$ ls -l ~ | head -1
```

8. Nur erste Zeile vom Longlisting, dort nur die Größe in Blöcken anzeigen:

```
$ ls -l ~ | head -1 | cut -d" " -f2
```

9. Erkundigen Sie sich nach den Benutzern, die am System arbeiten. Leiten Sie die Ausgabe, nach dem Terminal sortiert, in die Datei `user.dat` um:

```
$ who | sort -b +1 > user.dat
```

10. Wieviele Wörter, die mit `b` oder `B` beginnen, enthält die Datei `/usr/dict/words`? (Der Beginn einer Zeile kann durch das Zeichen `^` markiert werden also: `grep '^b' ...`). Diese Wörter, umgekehrt sortiert, in eine Datei schreiben und die Anzahl dieser Wörter am Bildschirm anzeigen lassen:

```
$ grep -i '^b' /usr/dict/words | sort -r | \
tee b_woerter.sort | wc -l
```

11. Einmal den gesamten `rm`-Befehl bestätigen:

```
# find /home -name ~.bak | xargs -p rm
```

Zweite Variante: Jede einzelne Datei bestätigen:

```
# find /home -name ~.bak | xargs rm -i
```

12. `$ man bash | tr -cd 'A-Z' | wc -c`

13. `$ man bash | tr -cd 'a-z' | wc -c`

```
$ man bash | tr -cd '0-9' | wc -c
```

```
$ man bash | tr -cd ':' | wc -c
```

```
$ man bash | tr -cd '.' | wc -c
```

3 Stichwortverzeichnis

Symbols	E	Q
/dev/null 53	emacs 9	Quota-System 4
/dev/tty 29	emacs 12	
/etc/issue 5	exit 5, 12	R
/etc/motd 5		reguläre Ausdrücke . 32, 33
/etc/passwd 9	F	
/usr/X11R6/lib/X11/doc/ 11	fg 17	S
/usr/share/dict/words 55	G	sections 8
/usr/share/doc/ ... 11	grep 33, 34	sed 37
/usr/share/doc/HOWTO/ 10	H	sort 45
/usr/share/doc/howto/ 10	halt 14	Standardausgabe 24
/usr/share/doc/packages/ 11	head 51	Standardeingabe 23
/usr/share/locale/ . 16	Homeverzeichnis 4	Standardfehlerausgabe . 24
/usr/src/linux/ Documentation ... 11	HOWTOS 10, 33	stderr 24
/var/log/messages .. 53	I	stderr 24
	I/O-Redirection 24	stdin 23
A	info 9	stdin 23
Account 4	L	stdout 24
Anmelden 3	LANG 16	stdout 24
apropos 7	less 7	Systemverwalter 4
awk 44	Locale 16	T
	Login 3	tail 51
B	Login-Vorgang 4	tee 29
Benutzername 4	ls 48	tr 39
Benutzerzugang 4	M	U
	man 5	Umleitung
C	man-pages 5	Ausgabe- 25
cal 5	manual-pages 5	Eingabe- 25
core 31	mkfifo 32	V
cut 43	more 32	virtuelle Terminals ... 15
	N	W
D	Named Pipes 31	wc 50
date 5	P	wc -l 27
Datei-Deskriptoren ... 24	Paßwort 4	whatis 8
	Pipe 28	X
		xargs 29