

Betriebssystem UNIX/Linux

Table of Contents

<u>Betriebssystem UNIX</u>	1
<u>Betriebssystem UNIX/Linux</u>	1
<u>Inhalt</u>	1
<u>1. Überblick über UNIX</u>	1
<u>2. Einführung in die Shell</u>	2
<u>3. Wichtige Kommandos</u>	2
<u>4. Kommandos zur Prozeß- und Systemverwaltung</u>	2
<u>5. Zeileneditoren und reguläre Ausdrücke</u>	3
<u>6. Bildschirmeditor vi</u>	3
<u>7. Kommunikationsdienste und Netze</u>	3
<u>8. Die Shell-Programmierung</u>	3
<u>9. Die Scriptsprache awk</u>	3
<u>10. Referenztabellen</u>	3
<u>11. Anhang</u>	4
<u>Vorlesung "Unix"</u>	5
<u>1. Überblick über Unix</u>	5
<u>1.1 Was ist Unix, was ist Linux</u>	5
<u>Was ist Linux?</u>	7
<u>1.2 Unix-Historie</u>	10
<u>1.3 Die Struktur von Unix</u>	12
<u>1.4 Der Kernel</u>	12
<u>1.4.1 Aufgaben</u>	12
<u>1.4.2 Erste Schritte</u>	15
<u>1.5 Das Dateisystem</u>	15
<u>1.5.1 Struktur und Zugriffsrechte</u>	24
<u>1.5.2 Unix-Dateistruktur</u>	26
<u>1.5.3 Sonderzeichen der Tastatur</u>	26
<u>1.5.4 Dateikommandos</u>	28
<u>1.6 Das Prozesskonzept</u>	29
<u>1.6.1 Der Scheduler</u>	31
<u>1.6.2 Swapping und Paging</u>	31
<u>1.6.3 Speicheraufteilung eines Prozesses im Arbeitsspeicher</u>	31
<u>1.6.4 Prozesskommunikation und -Synchronisation</u>	32
<u>1.6.5 fork(), exec() und wait()</u>	33
<u>1.6.6 Die Programme init und getty</u>	34
<u>1.6.7 Vorbemerkung zur Shell</u>	34
<u>1.6.8 Einflußnahme des Benutzers auf Prozesse</u>	34
<u>1.6.9 Das /proc-Verzeichnis</u>	35
<u>1.7 Grundlagen der Benutzerverwaltung</u>	35
<u>1.7.1 Was geschieht beim Login?</u>	36
<u>1.7.2 Aufbau der Datei /etc/passwd</u>	37
<u>1.7.3 Aufbau der Datei /etc/group</u>	37
<u>1.7.4 Aufbau der Datei /etc/shadow</u>	38
<u>1.7.5 Voreinstellungsverzeichnis /etc/default</u>	38
<u>1.7.6 Anlegen eines Benutzers</u>	39
<u>1.7.7 Löschen eines Benutzers</u>	39
<u>1.8 Starten und Stoppen des Systems</u>	45
<u>1.9 Sicherheitsmaßnahmen bei Arbeiten mit Root-Rechten</u>	45
<u>1.10 Das X Window System</u>	45
<u>Der Name</u>	55

Table of Contents

1. Überblick über Unix	
<u>Vorlesung "UNIX"</u>	56
2 Einführung in die Shell	56
2.1 Aufgaben der Shell.....	56
<u>Anmerkungen:</u>	58
2.2 Ein-und Ausgabeumleitung.....	59
<u>Eingabeumleitung</u>	62
2.3 Pipes.....	63
2.4 Metazeichen zur Expansion von Dateinamen.....	65
2.5 String-Ersetzungen (Quoting).....	65
2.6 Bash - Die Linux-Shell.....	66
Der Prompt.....	70
2.7 Schlußbemerkung.....	71
Copyright © FH München, FB 04, Prof. Jürgen Plate.....	71
<u>Vorlesung "UNIX"</u>	72
3. Einige Kommandos	72
3.1 Kommandos zur Steuerung der Zugriffsberechtigung.....	72
<u>chmod Modus Dateiliste</u>	73
<u>umask Modus</u>	73
<u>chown Username Dateiliste</u>	73
<u>chown markus dat1</u>	73
<u>chgrp Gruppe Dateiliste</u>	74
3.2 Dateidienste.....	74
<u>cat[-usvte] datei [datei]</u>	74
<u>cd [directory]</u>	74
<u>cp dat1 [dat2 ... datx] directory</u>	74
<u>ln [-fs] dat.alt dat.neu</u>	75
<u>mkdir Dirname</u>	75
<u>more</u>	75
<u>less</u>	75
<u>mv [-f] dat.alt dat.neu</u>	75
<u>rm [-fri] Dateiliste</u>	75
<u>rmdir directory</u>	76
<u>stat [Optionen] Datei(en)</u>	77
3.3 Steuerung des Druckers.....	77
<u>lp [-cdmwns] Dateiliste</u>	77
<u>lpstat [-acdoprstuv] [Auftragsnummer]</u>	78
<u>cancel [Auftragsnummer(n)] [Druckername(n)]</u>	78
<u>disable Druckername</u>	78
<u>enable Druckername</u>	78
<u>lpr Dateiliste</u>	78
<u>lprm</u>	78
<u>lpq</u>	79
3.4 Informationsdienste.....	79
<u>date [mmddhhmm[yy]] [+format]</u>	79
<u>touch [-em] [mmddhhmm[yy]] Dateiliste</u>	80
<u>df [-tfs] [Dateisystem] (Disk Free)</u>	80
<u>du [-arsk] [Dateiname]</u>	80
<u>file [-c] [-f Dateiname] [-m Dateiname] [Dateinamen]</u>	80
<u>find Pfadname(n) Bedingung(en) Aktion(en)</u>	83

Table of Contents

3. Einige Kommandos

<u>ps [adeflnptu] (Report Process Status)</u>	83
3.5 Die Kommandos stty und tput	84
Termcap.....	86
stty [Optionen].....	87
tput Parameter.....	87
3.6 Weitere Kommandos	87
xargs Programm [Parameter].....	88
wc [-lwc] [Datei(en)].....	88
pr [+n] [-h Header] [-ftF] [-wü] [-oü] [-lü] [Datei(en)].....	89
sleep zeit.....	89
tr Ersetzungsliste.....	89
cut -cSpalten [dateien].....	89
cal [[monat] jahr].....	90
calendar.....	90
basename string [suffix].....	90
dirname.....	90
logname.....	90
time [kommando].....	90
bc [-lmath].....	90
last [-Zahl] [-f datei] [namen].....	91
script [-a] [datei].....	91
3.7 Exkurs über Parameterersetzung	91
Copyright © FH München, FB 04, Prof. Jürgen Plate.....	91
Vorlesung "UNIX".....	92

4. Kommandos zur Prozeß- und Systemverwaltung.....92

4.1 Hintergrundprozesse	92
<u>Eigenschaften von Hintergrundprozessen</u>	93
nice [-increment] Kommando &.....	93
nohup Kommando &.....	93
4.2 Löschen von Prozessen	93
kill [-Signalnummer] Prozeßnummer(n).....	94
4.3 Kommandos zur Zeitsteuerung	94
at zeit [datum] [+incr].....	94
at [-r] [nummer].....	95
crontab [-er].....	96
4.4 Wechsel der Benutzeridentität	96
su [-] [Name].....	96
4.5 Dateisystem einbinden	96
mount [-a] [Gerätename] [Directory].....	97
umount Gerätename.....	97
Disk-Quotas.....	97
4.6 Weitere Informationskommandos	97
lsof.....	100
strace, ltrace.....	100
ldd.....	101
uname.....	101
free.....	101
uptime.....	101
<u>Weitere Informationskommandos unter Linux</u>	101
Systeminfos.....	103

Table of Contents

4. Kommandos zur Prozeß- und Systemverwaltung	
Sysstat-Paket.....	105
Nützliche Programme.....	106
Vorlesung "UNIX".....	107
Zeileneditoren und reguläre Ausdrücke.....	107
5.1 Der Editor ed.....	107
ed [-Optionen] [Dateiname].....	108
5.1.1 Editor-Anweisungen:.....	109
5.1.2 Textersetzung (s-Anweisung), Teil 1.....	110
5.1.3 Reguläre Ausdrücke.....	111
5.1.4 Textersetzung (s-Anweisung), Teil 2.....	112
5.2 grep.....	113
grep [Optionen] reg. Ausdruck [Dateiname(n)].....	114
sort [Optionen] [Dateiname(n)].....	114
3.4 Der Stream-Editor sed.....	115
sed [-n] [-e 'sed-Anweisungen'] [-f SKriptdatei] [eingabedatei(en)].....	118
Vorlesung "UNIX".....	119
6. Bildschirmeditor vi.....	119
vi Dateiname.....	120
Umschalten in den Eingabemodus.....	125
Vorlesung "UNIX".....	126
7. Kommunikationsdienste.....	126
wall.....	126
7.1 Das UNIX Mailsystem.....	126
Post versenden mail Login-Name(n).....	127
mail [-ehpqr] [-f datei] [-F login-name].....	128
Weiterleiten von Mails.....	129
7.2 Verbindung zu anderen UNIX-Rechnern.....	129
7.2.1 Am Anfang war das Modem.....	131
7.2.2 Die Internet-Protokollfamilie.....	139
7.2.3 Netzwerk-Kommandos.....	153
7.2.4 Binärdaten per Mail (oder News) versenden.....	155
7.3 NFS - Network File System.....	156
Copyright © FH München, FB 04, Prof. Jürgen Plate.....	156
Vorlesung "UNIX".....	157
8 Shell-Programmierung.....	157
8.1 Testen von Shell-Scripts.....	157
8.2 Kommentare in Shellscripits.....	158
8.3 Shell-Variable.....	158
8.3.1 Allgemeines.....	159
8.3.2 Quoting von Variablen.....	160
8.3.3 Vordefinierte Variablen:.....	160
8.3.4 Parameterzugriff in Shell-Scripten:.....	162
8.3.5 Namens- und Parameterersetzung:.....	163
8.3.6 Bearbeitung einer beliebigen Anzahl von Parametern.....	164
8.3.7 Gültigkeit von Kommandos und Variablen.....	165
8.4 Interaktive Eingaben in Shellscripits.....	165
8.5 Hier-Dokumente.....	166

Table of Contents

8 Shell-Programmierung

<u>8.6 Verkettung und Zusammenfassung von Kommandos</u>	166
<u>Hintereinanderausführung</u>	168
<u>8.7 Strukturen der Shell</u>	168
<u>8.7.1 Bedingungen testen</u>	170
<u>8.7.2 Bedingte Anweisung (if - then - else)</u>	172
<u>8.7.3 case-Anweisung</u>	173
<u>8.7.4 for-Anweisung</u>	174
<u>8.7.5 Abweisende Wiederholungsanweisung (while)</u>	176
<u>8.7.6 until-Anweisung</u>	176
<u>8.7.7 select-Anweisung</u>	177
<u>8.7.8 Weitere Anweisungen</u>	179
<u>8.7.9 Arithmetik in Scripts</u>	181
<u>8.7.10 exec [Kommandozeile]</u>	181
<u>8.7.11 eval [Argumente]</u>	182
<u>8.7.12 trap 'Kommandoliste' Signale</u>	183
<u>8.7.13 xargs</u>	184
<u>8.7.14 dialog</u>	186
<u>8.8 Beispiele für Shellscrip</u> t.....	186
<u>Disk usage</u>	191
<u>8.9 Shell-Funktionen</u>	195
<u>8.10 Fallen und Stolpersteine</u>	195
<u>Variablenamen</u>	198
<u>8.11 Weitere Beispiele</u>	198
<u>Eingabe ohne RETURN-Taste</u>	211
<u>Vorlesung "UNIX"</u>	212

9 Die Skriptsprache awk.....212

<u>9.1 Überblick</u>	212
<u>9.2 Aufruf und Optionen</u>	213
<u>9.3 Grundzüge der Sprache</u>	213
<u>9.3.1 Programmablauf</u>	216
<u>9.3.2 Konstante, Variable, Sätze und Felder</u>	218
<u>9.3.3 Eingebaute Variable</u>	219
<u>9.3.4 Sonderzeichen in regulären Ausdrücken</u>	219
<u>9.3.5 Zusammengesetzte reguläre Ausdrücke</u>	219
<u>9.4 Suchmuster und Aktionen</u>	220
<u>9.4.1 Suchmuster</u>	221
<u>9.4.2 Operatoren</u>	222
<u>9.4.3 Aktionen (Steueranweisungen)</u>	223
<u>9.4.4 Ein- und Ausgabe-Anweisungen</u>	225
<u>9.4.5 Stringfunktionen</u>	225
<u>9.4.6 System-Funktionen</u>	226
<u>9.4.7 Arithmetische Funktionen</u>	226
<u>9.5 Benutzerdefinierte Funktionen</u>	227
<u>9.6 awk-Aufruf</u>	227
<u>9.6.1 Variablen in der Kommandozeile setzen</u>	227
<u>9.6.2 Zugriff auf Shell-Parameter</u>	227
<u>9.6.3 Die Kommandozeilenoptionen</u>	227
<u>9.7 Beispiele</u>	227
<u>9.7.1 Notendurchschnitt</u>	228
<u>9.7.2 Wortliste</u>	228

Table of Contents

9 Die Skriptsprache awk	
<u>9.7.3 Spielen mit dem Mailspool</u>	229
<u>9.7.4 Rechner</u>	230
Vorlesung "UNIX".....	231
10 Kommandoreferenz	231
<u>10.1 vi Short Quick Reference Guide</u>	231
<u>vi-Kurzreferenz</u>	233
<u>10.2 Liste der Meta-Zeichen in regulären Ausdrücken</u>	233
<u>Wiederholungs-Operatoren</u>	235
<u>Vordefinierte Zeichenklassen</u>	235
<u>10.3 Tabellen zur Shell-Programmierung</u>	235
<u>Befehlsmöglichkeiten</u>	235
<u>Wildcards</u>	236
<u>Quotings</u>	236
<u>Eingabeumleitung</u>	237
<u>pipes</u>	237
<u>Ausdrücke beim test-Kommando</u>	238
<u>sed (stream editor)</u>	238
<u>tr</u>	239
<u>10.4</u>	239
<u>Copyright © FH München, FB 04, Prof. Jürgen Plate</u>	239
<u>Vorlesung "UNIX"</u>	240
11 Literatur und Anhang	240
<u>11.1 Literatur</u>	240
<u>Allgemeines</u>	242
<u>Open Books</u>	243
<u>Deutschsprachige Online-Bücher</u>	243
<u>Print</u>	245
<u>11.2 Erfinder von UNIX und C geben zu: Alles Quatsch!</u>	246
<u>11.3 The UNIX Acronym List</u>	248
<u>11.4 The ABCs of Unix</u>	249
<u>11.5 An amusing photo</u>	250
<u>11.6 Manpage BABY</u>	251
<u>11.7 UNIX-Universum</u>	252
<u>11.8 UNIX-Camping</u>	252
<u>Copyright © FH München, FB 04, Prof. Jürgen Plate</u>	title

Betriebssystem UNIX/Linux

Linux supports the notion of a command line or a shell for the same reason that only children read books with only pictures in them.

Language, be it English or something else, is the only tool flexible enough to accomplish a sufficiently broad range of tasks.

Bill Garrett

Inhalt

1. Überblick über UNIX

- 1.1 Was ist UNIX
- 1.2 Historie
- 1.3 Die Struktur von UNIX
- 1.4 Der Kernel
 - 1.4.1 Aufgaben
 - 1.4.2 Erste Schritte
- 1.5 Das Dateisystem
 - 1.5.1 Struktur und Zugriffsrechte
 - 1.5.2 UNIX-Dateistruktur
 - 1.5.3 Sonderzeichen der Tastatur
 - 1.5.4 Dateikommandos - die ersten Kommandos
- 1.6 Das Prozesskonzept
 - 1.6.1 Der Scheduler
 - 1.6.2 Swapping und Paging
 - 1.6.3 Speicheraufteilung eines Prozesses im Arbeitsspeicher
 - 1.6.4 Prozesskommunikation und -Synchronisation
 - 1.6.5 FORK, EXEC und WAIT
 - 1.6.6 Die Programme INIT und GETTY
 - 1.6.7 Vorbemerkung zur shell
 - 1.6.8 Einflußnahme des Benutzers auf Prozesse
- 1.7 Grundlagen der Benutzerverwaltung
 - 1.7.1 Was geschieht beim Login?
 - 1.7.2 Aufbau der Datei /etc/passwd
 - 1.7.3 Aufbau der Datei /etc/group
 - 1.7.4 Aufbau der Datei /etc/shadow
 - 1.7.5 Voreinstellungsverzeichnis /etc/default
 - 1.7.6 Anlegen eines Benutzers
 - 1.7.7 Löschen eines Benutzers
- 1.8 Starten und Stoppen des Systems
- 1.9 Sicherheitsmaßnahmen beim Arbeiten mit Root-Rechten
- 1.10 Das X-Window-System

2. Einführung in die Shell

- 2.1 Aufgaben der Shell

[2.2 Ein- und Ausgabeumleitung](#)

[2.3 Pipes](#)

[2.4 Metazeichen zur Expansion von Dateinamen](#)

[2.5 String-Ersetzungen \(Quoting\)](#)

[2.6 Bash - die Linux-Shell](#)

[2.7 Schlußbemerkung](#)

Die PDP-8 von DEC war der erste UNIX-Rechner



3. Wichtige Kommandos

[3.1 Kommandos zur Steuerung der Zugriffsberechtigung](#)

[3.2 Dateidienste](#)

[3.3 Steuerung des Druckers](#)

[3.4 Informationsdienste](#)

[3.5 Die Kommandos stty und tput](#)

[3.6 Weitere Kommandos](#)

[3.7 Exkurs über Parameterersetzung](#)

4. Kommandos zur Prozeß- und Systemverwaltung

[4.1 Hintergrundprozesse](#)

[4.2 Löschen von Prozessen](#)

[4.3 Zeitsteuerung](#)

[4.4 Wechsel der Benutzeridentität](#)

[4.5 Dateisystem einbinden](#)

[4.6 Weitere Informationskommandos](#)

5. Zeileneditoren und reguläre Ausdrücke

[5.1 Der Editor ed](#)

[5.1.1 Editor-Anweisungen](#)

[5.1.2 Textersetzung \(1\)](#)

[5.1.3 Reguläre Ausdrücke](#)

[5.1.4 Textersetzung \(2\)](#)

[5.2 Suchen mit grep](#)

[5.3 Sortieren mit sort](#)

[5.4 Der Stream-Editor sed](#)

6. Bildschirmditor vi

7. Kommunikationsdienste und Netze

7.1 Das UNIX Mailsystem

7.2 Verbindung zu anderen UNIX-Rechnern

7.2.1 Am Anfang war das Modem

7.2.2 Die Internet-Protokoll-Familie

7.2.3 Netzwerk-Kommandos

7.2.4 Binärdaten per Mail versenden

7.3 NFS - Network File System

8. Die Shell-Programmierung

8.1 Testen von Shellskripts

8.2 Kommentare in Shellskripts

8.3 Shell-Variable

8.4 Interaktive Eingaben in Shellskripts

8.5 Hier-Dokumente

8.6 Verkettung und Zusammenfassung von Kommandos

8.7 Strukturen der Shell

8.8 Beispiele für Shell-Skripts

8.9 Shell-Funktionen

8.10 Weitere Beispiele

Zum Weiterlesen:

B.H.: Einzeiler für die Shell

B.H.: Kurze Skripte

9. Die Scriptsprache awk

9.1 Überblick

9.2 Aufruf und Optionen

9.3 Grundzüge der Sprache

9.4 Suchmuster und Aktionen

10. Referenztabellen

10.1 vi Short Quick Reference Guide

10.2 Liste der Meta-Zeichen in regulären Ausdrücken

10.3 Tabellen zur Shell-Programmierung

Debian Linux Referenz (PDF, 265 Seiten)

11. Anhang

11.1 Literatur

11.2 Erfinder von UNIX und C geben zu: Alles Quatsch!

11.3 The UNIX Acronym List

11.4 The ABCs of Unix

[11.5 An amusing Photo](#)

[11.6 Manpage BABY](#)

[11.7 UNIX-Universum](#)

[11.8 UNIX Camping](#)

Artikel

[The UNIX Time-Sharing System*](#)

[The Development of the C Language*](#)

[The Evolution of the Unix Time-sharing System*](#)

[Ritchie and Thompson Receive National Medal of Technology](#)

[UNIX Advertisement](#)

[UNIX-Patent, 1. Seite \(gif\)](#)

[UNIX-Patent komplett \(PDF\)](#)

[Interview mit Brian Kernighan](#)

Diverses

[Rechnerbenutzung](#)

[Übungsaufgaben](#)

[Sind Sie fit in UNIX?](#)

[vi-Howto](#)

[Dialog-Howto](#)

[Linux-Daemon-Howto](#) und [dummy_daemon.c](#)

[Skript als PDF](#) (aus HTML konvertiert)

[Download](#) des gesamten Skripts

"Linux is a cancer that attaches itself in an intellectual property sense to everything it touches.

- Steve Ballmer, Juni 2001



Will we interoperate with products that come, like Linux, from the Open Source world? Yes, we will.

Will we encourage people who want to do opensource development to do it on top of Windows? Yes.

- Steve Ballmer, Juli 2008

Copyright © FH München, FB 04, Prof. Jürgen Plate

Letzte Aktualisierung: 14. Apr 2012

Vorlesung "Unix"

von Prof. Jürgen Plate

1. Überblick über Unix

Unix is user friendly, it just happens to be very particular about who it makes friends with.

1.1 Was ist Unix, was ist Linux

Das Betriebssystem Unix blickt auf eine lange Vergangenheit zurück. Ursprünglich wurde es für PDP-Rechner der Firma DEC (Digital Equipment Corporation) entwickelt und sollte hauptsächlich die Bedürfnisse von professionellen Softwareentwicklern befriedigen. Heute ist Unix auf Workstations mit 64-Bit-Prozessoren zugeschnitten. Zu den Neuerungen gehören die Einführung von Threads, mit denen ein Programm mehrere Funktionen gleichzeitig ausführen kann. Ebenfalls neu sind Echtzeitfunktionen mit denen Programmlaufzeiten vorhersagbar sind. Insbesondere für Datenbanken ist die Möglichkeit, sehr große Dateien verarbeiten zu können, von nicht unerheblicher Bedeutung. Ein Schritt in Richtung einer völligen Unabhängigkeit von der Hardware-Architektur ist die Normierung ganzer Zahlen mit einer Länge von mehr als 64-Bit. Ebenfalls neu ist die Einbeziehung der Benutzeroberfläche in die Spezifikation. Aber schon seit den Anfängen gilt:

- Unix ist ein portables, einfach aufgebautes Betriebssystem
 - ◆ Multitasking-BS (Multiprocessing-BS)
 - ◆ Multiuser-BS (Mehrbenutzer-BS)
 - ◆ dialogorientiert
- Unix ist ein Werkzeugkasten
 - ◆ viele hundert Dienstprogramme (dadurch wird es so kompliziert)
 - ◆ flexibel: kleine Tools sind schnell erstellt: "small is beautiful"
- Unix ist geeignet für
 - ◆ Mikrocomputer der Oberklasse
 - ◆ Mini-Computer
 - ◆ Großrechner
- Unix ist schrecklich
 - ◆ Befehlsname sind kryptisch (ls, pwd, cat, awk, grep, ...)
 - ◆ Motto 1: "Keine Nachricht ist eine gute Nachricht!"
 - ◆ Motto 2: "Wer will schon schlechte Nachrichten?"
- Das Betriebssystem Unix ist aus diesem Grunde ein System von Programmierern für Programmierer, die wissen was sie tun.

Unix gehört historisch gesehen zu den älteren Betriebssystemen. Dennoch ist es gleichzeitig ein modernes Betriebssystem, das von Anfang an mit Merkmalen ausgestattet war, die von anderen Systemen erst viel später in einer vergleichbaren Form angeboten wurden. Unter Unix gab es von Anfang an echtes Multitasking, eine Trennung der Prozesse voneinander (d. h. hohe Stabilität), klar definierte Zugriffsrechte für Dateien (d. h. hohe Sicherheit im Multiuser-Betrieb), ausgereifte Netzwerkfunktionen etc. Allerdings bot Unix vor ein bis zwei Jahrzehnten nur eine kommandozeilenorientierte Benutzeroberfläche und stellte hohe Hardware-Anforderungen. Deshalb wurde es fast ausschließlich auf teuren Workstations im wissenschaftlichen und industriellen Bereich eingesetzt.

Was ist Linux?

Linux ist ein Unix-ähnliches Betriebssystem. Der wichtigste Unterschied gegenüber herkömmlichen Unix-Systemen besteht darin, dass Linux zusammen mit dem vollständigen Quellcode frei weitergegeben werden darf. Details zu den Bedingungen, unter denen Linux und die dazugehörigen Programme weitergegeben werden dürfen, folgen an Ende dieses Kapitels.

Linux ist im Prinzip nichts anderes als eine neue Unix-Variante. Zu den Besonderheiten von Linux zählen die freie Verfügbarkeit des gesamten Quelltexts und die große Hardware-Unterstützung. Genauer genommen bezeichnet der Begriff Linux nur den **Kernel**: Der Kernel ist der innerste Teil (Kern) eines Betriebssystems mit ganz elementaren Funktionen wie Speicherverwaltung, Prozessverwaltung und Steuerung der Hardware.

Als **Linux-Distribution** wird die Einheit bezeichnet, die aus dem eigentlichen Betriebssystem (Kernel) und seinen Zusatzprogrammen besteht. Eine Distribution ermöglicht eine rasche und bequeme Installation von Linux. Distributionen werden zumeist in Form von CD-ROMs oder DVDs verkauft. Viele Distributionen sind darüber hinaus auch zum Download im Internet verfügbar. Wegen der riesigen Datenmengen (oft mehrere GByte) ist das Kopieren einer Distribution via Internet bzw. eine direkte Installation über das Netz aber nur bei einer ausgezeichneten Internet-Anbindung möglich. Manche behandeln die Distributionen sogar so, als seien sie eigene Betriebssysteme. Die Frage, welche Distribution die beste sei, welche wem zu empfehlen sei etc., artet leicht zu einem Glaubenskrieg aus. Wer sich einmal für eine Distribution entschieden und sich an deren Eigenheiten gewöhnt hat, steigt nicht so schnell auf eine andere Distribution um. Ein Wechsel der Distribution ist nur durch eine Neuinstallation möglich, bereitet also einige Mühe. Um Linux herum existieren auch etliche Behauptungen und Vorurteile:

Linux ist schneller/langsamer als Windows: Diese Aussage ist so weder in der einen noch in der anderen Form richtig. Tatsächlich gibt es einzelne Programme, die unter Linux oder unter Windows schneller laufen. Daraus lassen sich aber keine allgemein gültigen Schlussfolgerungen ziehen. Das Ergebnis hängt unter anderem davon ab, für welches Betriebssystem das Programm optimiert wurde, welche Linux- und Windows-Versionen miteinander verglichen werden, welche Hardware für den Vergleich verwendet wurde etc.

Linux benötigt weniger Ressourcen als Windows: Grundsätzlich stimmt es, dass Sie Linux auf einem 486-er PC mit einigen MByte RAM betreiben können. In dieser Konfiguration läuft Linux zwar nur im Textmodus, bietet ansonsten aber sicher viel mehr Funktionen als eine alte Windows-Version, die auf einem derartigen Rechner ebenfalls noch läuft. Wenn Sie dagegen eine aktuelle Linux-Distribution von Red Hat oder Suse mit einer aktuellen Windows-Version vergleichen, sind die Unterschiede weniger deutlich. Für ein komfortables Arbeiten in einer grafischen Benutzeroberfläche (KDE oder Gnome) stellt Linux ähnliche Hardware-Ansprüche wie Windows.

Linux ist sicherer als Windows: Leider kranken alle zurzeit populären Betriebssysteme an Sicherheitsproblemen. Linux schneidet in den meisten Vergleichen relativ gut ab, dennoch finden sich immer wieder neue Sicherheitslücken. Wie sicher Linux ist, hängt aber auch von seiner Verwendung ab:

- ◆ In Desktop-Anwendungen ist Linux fast vollständig virensicher. Es hat bis jetzt keinen einzigen nennenswerten Virenbefall unter Linux gegeben, während Windows-Viren in regelmäßigen Abständen ganze Firmen tagelang lähmen. Der Hauptgrund besteht darin, dass die Zugriffsverwaltung unter Linux verhindert, dass gewöhnliche Anwender großen Schaden am System anrichten können. Außerdem sind Webbrowser und E-Mail-Programme unter Linux generell viel sicherer als die entsprechenden Windows-Programme.
- ◆ Bei der Anwendung von Linux als Netzwerk- oder Internet-Server hängt die Sicherheit sehr stark von der Wartung des Systems ab. Beinahe zu allen Sicherheitsproblemen der vergangenen Jahre gab es bereits Updates, bevor diese Sicherheitsrisiken allgemein bekannt und von Hackern ausgenutzt wurden. Wenn Sie also die auf Ihrem Rechner eingesetzte Software regelmäßig aktualisieren, haben Angreifer wenig Chancen, in Ihr System einzudringen.

Die Sicherheit von Linux-Systemen hängt schließlich sehr stark von Ihrem eigenen Wissen

ab. Wenn Sie als Linux-Einsteiger rasch einen Internet-Server konfigurieren und ins Netz stellen, ist nicht zu erwarten, dass dieser Server bereits optimal abgesichert ist. Es mangelt aber nicht an Literatur zu diesem Thema!

Linux ist stabiler als Windows: Mittlerweile hat Microsoft mit Windows 7 durchaus respektable und stabile Windows-Versionen zustande gebracht. Der Linux-Kernel an sich ist außerordentlich stabil. Wenn Sie mit Linux aber das Gesamtsystem der mitgelieferten Software meinen (also eine ganze Distribution), dann sieht es mit der Stabilität gleich erheblich schlechter aus. Insbesondere relativ neue Programme stürzen immer wieder ab. Server-Programme laufen dagegen meist vollkommen fehlerfrei. Je stärker Sie sich anwendungsorientierten Programmen zuwenden und Linux als Desktop-System einsetzen, desto eher werden Sie die negativen Seiten kennen lernen.

Linux ist billiger als Windows: Diese Aussage ist leicht zu untermauern - Linux ist schließlich kostenlos erhältlich. Bei Microsoft hat man mit dieser Argumentation natürlich keine Freude - dort weist man darauf hin, dass auch Schulungskosten etc. berücksichtigt werden müssen. (In solchen Rechenbeispielen wird Windows-Wissen meist als gottgegeben vorausgesetzt, Linux-Kenntnisse natürlich nicht.) Außerdem ist nicht jede Linux-Distribution tatsächlich kostenlos.

Linux ist kompliziert zu installieren: Wenn man einen PC kauft, ist Windows meist schon vorinstalliert. Insofern stellt es natürlich einen Mehraufwand dar, Linux zusätzlich zu installieren. Wie Sie im nächsten Kapitel feststellen werden, ist eine Linux-Installation aber mittlerweile kinderleicht - und sicher nicht schwieriger als eine Windows-Installation. Problematisch ist lediglich die Unterstützung neuer Hardware, die unter Windows besser ist: Jeder Hersteller von Computer-Komponenten stellt selbstverständlich einen Windows-Treiber zur Verfügung. Vergleichbare Treiber für Linux müssen dagegen oft von der Open-Source-Gemeinschaft programmiert werden. Das dauert natürlich eine gewisse Zeit.

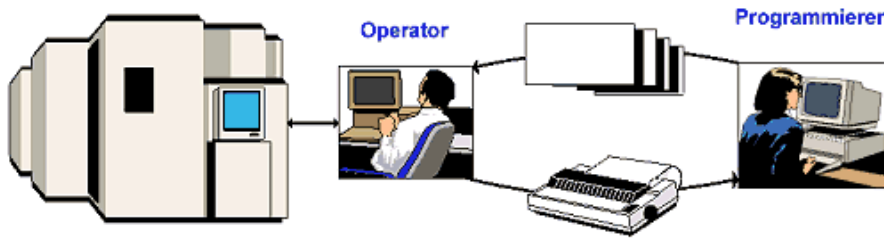
Linux ist kompliziert zu bedienen: Dieses Vorurteil ist alt, aber nicht mehr bzw. nur noch in einem sehr geringen Maß zutreffend. Linux ist einfach anders zu bedienen als Windows, so wie auch Apples Mac OS anders zu bedienen ist. Wirklich schwieriger ist die Handhabung von Linux zumeist nicht, lediglich die Umgewöhnung von Windows kann manchmal mühsam sein.

1.2 Unix-Historie

Unix ist aus der Notwendigkeit entstanden, für die Programmierstellung ein Rechnerbetriebssystem zu entwickeln. Die Portierbarkeit von Unix beruht auf der Entwicklung der Programmiersprache C (als Nachfolger von BCPL und B), die es möglich machte, ca. 90% des Betriebssystems in einer höheren Programmiersprache zu entwickeln.

Als Ken Thompson 1969 bei Bell Laboratories, die Entwicklung eines neuen Betriebssystems begann, waren die meisten der vorhandenen Systeme ausgesprochene Batch-Systeme. Der Programmierer gab seine Lochkarten oder Lochstreifen beim Operator ab, diese wurden in den Rechner eingelesen und ein Rechenauftrag nach dem anderen abgearbeitet. Der Programmierer konnte dann nach einiger Zeit seine Ergebnisse abholen.

Betriebssystem UNIX/Linux



Ziel von Ken Thompsons Entwicklung war es deshalb, ein System zu schaffen, auf welchem mehrere Programmierer im Team und im Dialog mit dem Rechner arbeiten, Programme entwickeln, korrigieren und dokumentieren konnten, ohne von einem Großrechner mit allen seinen Restriktionen abhängig zu sein. Dabei standen Funktionalität, strukturelle Einfachheit und Transparenz sowie leichte Bedienbarkeit im Vordergrund der Entwicklung. Dieses erste System mit dem Namen Unix lief auf einer DEC PDP-7. Unix ist vom Betriebssystem "Multics" inspiriert, so auch der Name. Brian Kernighan schlug vor, das neue Betriebssystem "Unics" zu nennen. Irgendwer hat es dann mit "X" am Schluss buchstabiert und dabei blieb es dann.

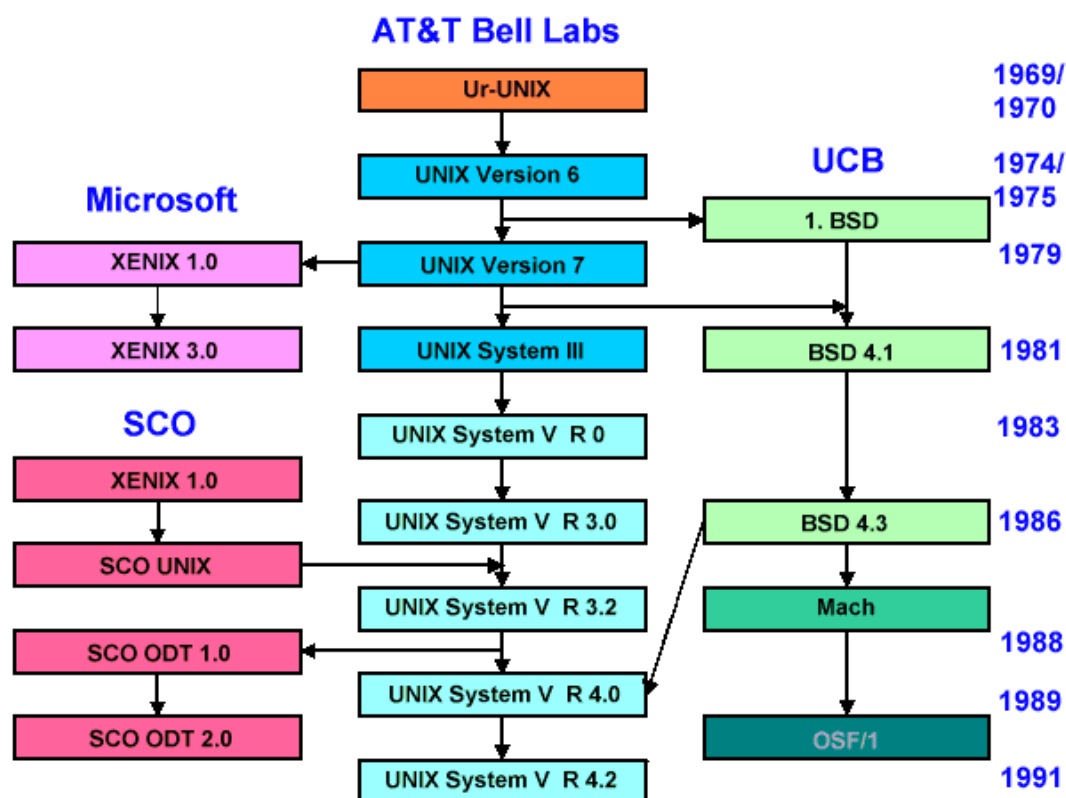
Schon bald findet sich eine kommerzielle Anwendung: Die Patentabteilung der Bell Labs sucht ein System zum Erstellen, Bearbeiten und Formatieren von Patentformularen. Mit dem auf die PDP-11 portierten und erweiterten Text-Prozessor "roff" (heute noch als "nroff" für die Formatierung der Manual-Pages zuständig) nimmt Unics Mitte 1971 seinen ersten kommerziellen Dienst auf. Währenddessen arbeiten die Entwickler an Erweiterungen und neuen Programmen auf derselben Maschine. Am 3. November 1971 ist schließlich das "Unix Time-Sharing System", First Edition fertig.

Die erste Version von Unix war in der Assemblersprache der PDP-7 geschrieben. Von Anfang an wollte Thompson das System in einer hardwareunabhängige Hochsprache programmieren. Zuerst entwickelt er eine einfache Sprache auf Basis von BCPL (Basic Combined Programming Language), die er schlicht "B" nennt. B ist als Interpreter-Sprache für ein Betriebssystem aber nicht schnell genug. Dennis Ritchie erweitert die Syntax um strukturierte Typen und schreibt 1971 den fehlenden Compiler. Brian Kernighan, späterer Koautor von AWK (Aho, Weinberger, Kernighan) steuert die Dokumentation zur neuen Sprache bei, die "C" genannt wird.

Unix wurde 1972 in C umgeschrieben und auf die PDP-11 übertragen. Von nun an erfolgte die Weiterentwicklung des Systemkerns sowie der meisten Dienstprogramme in dieser Sprache. Die Kompaktheit und strukturelle Einfachheit des Systems ermunterte viele Benutzer zur eigenen Aktivität und Weiterentwicklung des Systems, so daß Unix recht schnell einen relativ hohen Reifegrad erreichte. Dies ist deshalb bemerkenswert, da kein Entwicklungsauftrag hinter diesem Prozess stand und die starke Verbreitung von Unix nicht auf den Vertrieb oder die Werbung eines Herstellers, sondern primär auf das Benutzerinteresse zurückzuführen ist. Eine ähnliche Entwicklung zeigt sich seit einigen Jahren bei den freien Unix-Varianten.

Mitte 1973 erscheint Unix V4, fast vollständig in C geschrieben. Damit ist das System auch sehr portabel. Wo immer ein C-Compiler existiert, kann Unix implementiert werden. Schon 1973 erhält die UCB (University of California, Berkeley) eine Kopie von Unix V4 und beginnt umgehend mit Erweiterungen - die Wurzel des heute noch aktiven BSD-Unix.

Betriebssystem UNIX/Linux



Da das 1956 erlassene Consent Decree AT&T kommerzielle Aktivitäten außerhalb des Telefonmarktes verbietet, gibt die Firma Unix-Quellen und -Binaries zum geringen Preis an Universitäten ab. 1977 veröffentlicht der Student Bill Joy die Berkeley-Erweiterungen als Berkeley Software Distribution (BSD), 1978 folgt 2BSD und der legendäre Editor "vi". 3BSD und 4BSD werden zur Grundlage für die Entwicklung von TCP/IP - und damit des Internets. Die für Universitäten erschwinglichen PDP-Nachfolgerechner VAX und das darauf portierte 4.xBSD steigern die Verbreitung. Im Lauf der Zeit sind zwei Entwicklungszweige entstanden, da Unix sowohl bei Bell Labs (AT&T) als auch an der Universität von Berkeley weiterentwickelt wurde: "BSD" und "System V". Kommerzielle Unix-Versionen von Sun (SunOS/Solaris), DEC (Ulrix) und andere folgen.

Als 1979 nach der Freigabe von Unix V7 das Consent Decree fällt, will AT&T Unix kommerziell nutzen. Als eine der ersten Firmen lizenziert Microsoft 1979 den Code von Unix V7. Seine Portierungen auf Intels 8086 erscheinen 1980 unter dem Namen "Xenix OS". Obwohl Microsoft bereits 1987 Xenix an SCO verkauft, die es weiterführt, hat dieses kurze Intermezzo Folgen: Die Unix-Erfahrungen bilden schon den Hintergrund des DOS-2.0-Dateisystems und die Grundlage für Windows NT. Auch die TCP/IP-Implementierung erinnert stark an BSD-Unix.

AT&T entwickelt Unix weiter, es folgen die Systeme III, IV und V. Ihnen entstammen viele Derivate diverser Hersteller. Auch entwickelt AT&T die freien UNIXe V8, V9 und V10 weiter, doch ohne sich um die Verbreitung zu bemühen. Die Nachfolge tritt 1987 Plan 9 an. Daneben wurden zahlreiche weitere Unix-Derivate entwickelt, z. B. die frei erhältlichen Systeme Net-BSD; Free-BSD, Minix und "Linux".

Die allerersten Teile des **Linux-Kernels** wurden von **Linus Torvalds** (Helsinki) entwickelt, der den Programmcode im September 1991 über das Internet freigab. In kürzester Zeit fanden sich weltweit Programmierer, die an der Idee Interesse hatten und Erweiterungen dazu programmierten: ein verbessertes System zur Dateiverwaltung, Treiber für diverse Hardware-Komponenten, Zusatzprogramme wie den DOS-Emulator etc. All diese Einzelkomponenten wurden ebenfalls kostenlos zur Verfügung gestellt. Das Gesamtsystem wuchs mit einer atemberaubenden Geschwindigkeit. Die Entstehung dieses neuen Betriebssystems wäre ohne die weltweite Kommunikation der Programmierer via Internet unmöglich gewesen.

Betriebssystem UNIX/Linux

Ein wesentlicher Faktor dafür, dass Linux frei von den Rechten der großen Software-Firmen ist und dennoch derart schnell entwickelt werden konnte, war die zu diesem Zeitpunkt schon frei verfügbare Software. Linux ist nicht aus dem Nichts aufgetaucht, wie das manchmal fälschlich dargestellt wird, sondern baut auf einer breiten Basis freier Software auf. Für die ersten Schritte war das freie (aber im Funktionsumfang sehr eingeschränkte) Minix eine praktische Grundlage. So verwendeten die ersten Linux-Versionen noch das Dateisystem von Minix.

In ihrer Bedeutung wohl noch wichtiger für Unix und Linux waren und sind die zahlreichen GNU-Programme (*GNU is Not Unix*). GNU-Programme wurden auf vielen Unix-Systemen als Ersatz für diverse (teure) Originalkomponenten verwendet - etwa der GNU-C-Compiler, der Texteditor Emacs und viele andere GNU-Utilities. Sobald der Kernel von Linux dann so weit entwickelt worden war, dass der GNU-C-Compiler darauf zum Laufen gebracht werden konnte, stand praktisch mit einem Schlag die gesamte Palette der GNU-Tools zur Verfügung. So wurde aus einem Kernel plötzlich ein recht vollständiges System, das dann für eine noch größere Entwicklergemeinde zu einer attraktiven Umgebung wurde.

GNU-Programme sind ebenso wie Linux (unter gewissen Einschränkungen) frei kopierbar - und zwar nicht nur als Binärprogramme, sondern mit sämtlichen Codequellen. Das ermöglicht es allen GNU-Anwendern, die Programme bei Problemen oder Fehlern selbst zu erweitern oder zu korrigieren. Aus diesen Änderungen resultieren immer bessere und ausgereifere Versionen der diversen GNU-Programme. Nicht zuletzt aufgrund der freien Verfügbarkeit des Programmcodes stellt der GNU-C-Compiler *den* Standard in der Unix-Welt dar: Der Compiler ist praktisch auf jedem Unix-System verfügbar - und nicht nur dort.

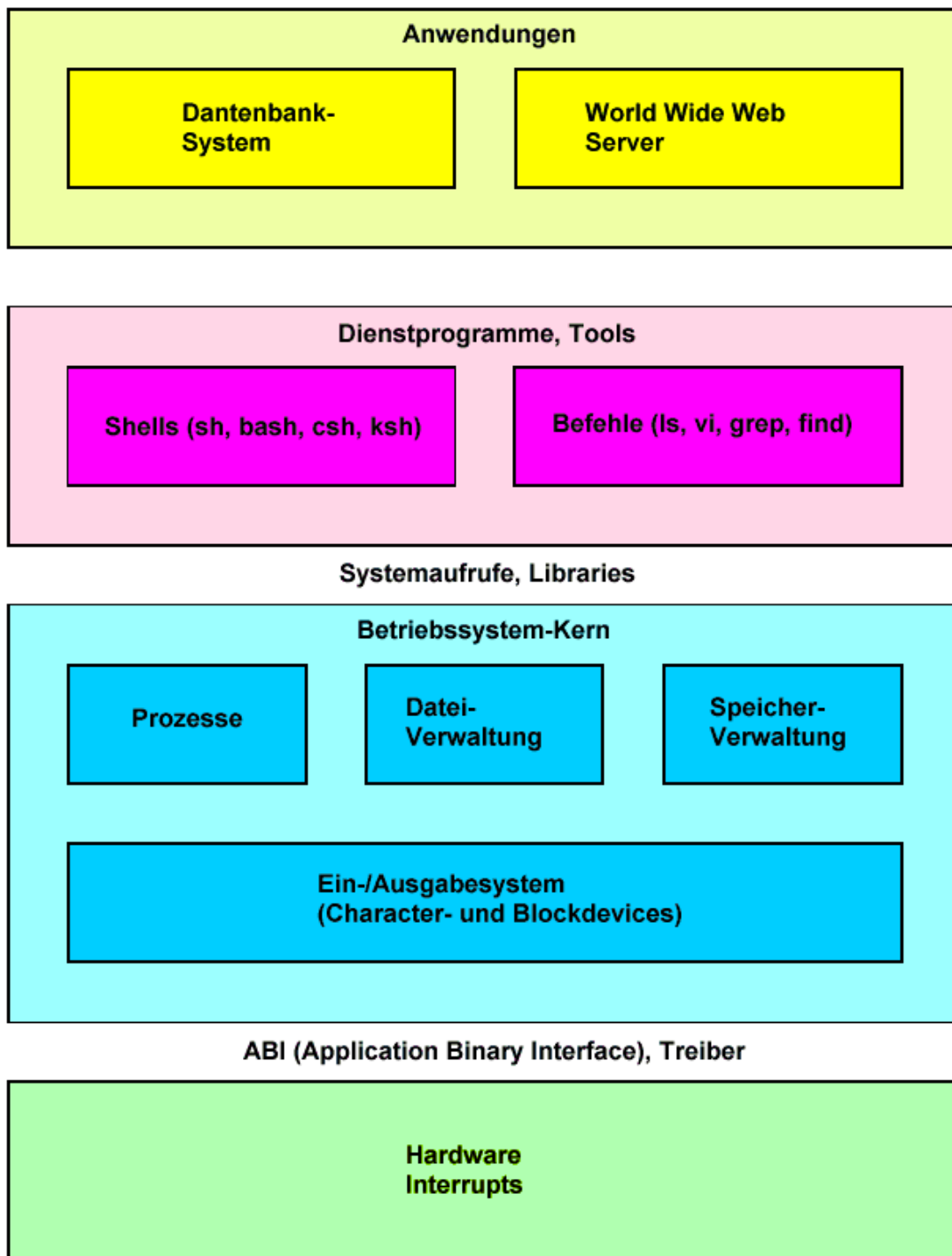
Erst die Kombination aus dem Linux-Kernel, den zahlreichen GNU-Komponenten, der Netzwerk-Software des BSD-Unix, dem ebenfalls frei verfügbaren X Window System des MIT (Massachusetts Institute of Technology) und dessen Portierung XFree86 für PCs mit Intel-Prozessoren sowie aus zahlreichen weiteren Programmen macht eine Linux-Distribution zu einem kompletten Unix-System.

Das Ziel der Entwickler von GNU und Linux war es also, ein System zu schaffen, dessen Quellen frei verfügbar sind und es auch bleiben. Um einen Missbrauch auszuschließen, ist Software, die im Sinne von GNU entwickelt wurde und wird, durch die *GNU General Public License* (kurz GPL) geschützt. Hinter der GPL steht die *Free Software Foundation* (FSF). Diese Organisation wurde von Richard Stallmann (der unter anderem auch Autor des Editors Emacs ist) gegründet, um qualitativ hochwertige Software frei verfügbar zu machen.

1.3 Die Struktur von Unix

Zur Verdeutlichung der Zusammenarbeit verschiedener Komponenten in einem Rechnersystem wird allgemein ein Schalen- oder Schichtenmodell verwendet. Dabei werden die einzelnen Komponenten in Form von Schalen oder Schichten dargestellt. Die Grenze zwischen den einzelnen Schalen werden dabei als Schnittstellen bezeichnet. In der folgenden Abbildung wird das Unix Schalenmodell gezeigt. Die Funktionen des Betriebssystemkerns sind dabei speziell hervorgehoben.

Betriebssystem UNIX/Linux



Die Dialogschnittstelle zur Kommunikation mit dem Benutzer (zeichenorientiert) wird dabei als Shell bezeichnet. Diese Shells unter Unix haben dabei zwei Funktionen, sie werden

- als Kommandointerpreter und
- als Programmiersprache

verwendet. Die zweite Funktion ist unter Unix deshalb so bedeutend, da die gesamte Verwaltung (Administration) des Betriebssystems mit Skripten in dieser "Shell-Programmiersprache" erfolgt. Diese Möglichkeit wird in den Kapiteln 2 und 9 behandelt. Es gilt also:

Kommandointerpreter = Shell:

- sh: Bourne Shell an die Programmiersprache ALGOL 68 angelehnt

- csh: C-Shell an die Programmiersprache C angelehnt
- ksh: Korn-Shell vereinigt Bourne- und C-Shell
- bash: Bourne-Again-Shell (Erweiterung der Bourne Shell)
- ...

1.4 Der Kernel

1.4.1 Aufgaben

- Prozess-Scheduling
- Prozess-Umschaltung
- Prozess-Kommunikation
- Dateisystem verwalten
- Ein-/Ausgabesteuerung
- Gerätesteuerung (device driver)
- Zugangskontrolle und Abrechnung
- alle Systemdienste für Programmier-Schnittstellen
- der Kernel ist relativ klein (ladbare Treiber)
- alle Interrupts und I/O-Operationen werden über den Kernel abgewickelt

1.4.2 Erste Schritte

Unix ist ein Multiuser- und Multitasking-System. Das bedeutet, daß in einem bestimmten Augenblick sowohl mehrere Benutzer auf einer Unix-Maschine gleichzeitig arbeiten können, als auch, daß jeder einzelne dieser Benutzer mehrere Programme aufrufen kann, die alle zur gleichen Zeit ausgeführt werden.

Damit jeder dieser Benutzer seine Daten vor dem Zugriff der anderen Benutzer schützen kann, muß man sich, bevor man mit einem Unix-System arbeiten kann, erst einmal *anmelden*, das heißt, einen speziellen Benutzernamen und ein Passwort eingeben. Dadurch erfährt das System, welcher Benutzer da gerade die Arbeit aufnehmen möchte, und kann diesem Benutzer seine persönliche Arbeitsumgebung (inclusive aller privater Daten) zur Verfügung stellen.

Nach dem Einschalten des Terminals bzw. nach Aufnahme der Verbindung mit dem Unix-Rechner meldet sich das BS mit der Aufforderung, sich zu identifizieren:

```
login:
```

Der Benutzer gibt darauf den ihm zugewiesenen Login-Namen ein. Dann erscheint die Abfrage des Paßwortes:

```
password:
```

Nun muß der Benutzer sein Passwort eingeben. Das Passwort wird im Gegensatz zu den üblichen Eingaben nicht auf dem Bildschirm ausgegeben. Wenn alles gutgeht, sind Sie jetzt beim System angemeldet (man sagt auch: *eingeloggt*). Sie erkennen die erfolgreiche Anmeldung daran, daß die Eingabeaufforderung des Systems, der sog. *Prompt* erscheint. Der Prompt sieht etwa so aus:

```
benutzername@sun1-lbs$
```

und dahinter ist ein Cursor sichtbar (die Texteingabemarkierung) und das System erwartet nun die Eingabe eines Kommandos.

Hat sich der Benutzer vertippt, erscheint die Meldung:

```
login incorrect
```

und die o. g. Prozedur muß wiederholt werden. Neue Benutzer haben noch kein Passwort, sie drücken nur die RETURN-Taste bei der Frage nach dem Passwort. Bei vielen Systemen wird der Benutzer beim ersten Login zur Eingabe des Passwortes aufgefordert. Zum Ändern und Eingeben des Passwortes gibt es ein eigenes Kommando:

```
passwd
```

Das Passwort muß einigen Bedingungen genügen:

- Mindestlänge 6 Zeichen
- Je nach BS-Version muß mindestens eine Zahl und/oder ein Sonderzeichen darin enthalten sein

User und Superuser

Unix kennt zwei Benutzerklassen. Die normalen Benutzer sind voneinander abgeschottet (siehe Kap. 1), jeder besitzt seinen eigenen Arbeitsbereich. Der Superuser, normalerweise der Systemadministrator, kann und darf alles. Er kann auf jede Datei zugreifen und Benutzer eintragen oder löschen. Er kann auch Systemdienste aufrufen, die normalen Benutzern verwehrt sind. Normalerweise hat der Superuser den login-Namen "root". Der login-Name spielt jedoch eine untergeordnete Rolle, der Superuser ist vielmehr derjenige Benutzer, der die Usernummer (User Id, UID) 0 hat (Genaueres später).

Beenden der Arbeit am Computer

Zum Beenden der Arbeit am Rechner muß sich der Benutzer abmelden (logoff). Bei Unix geschieht dies (je nach System) durch den Befehl exit oder die Tastenkombination Control-D.

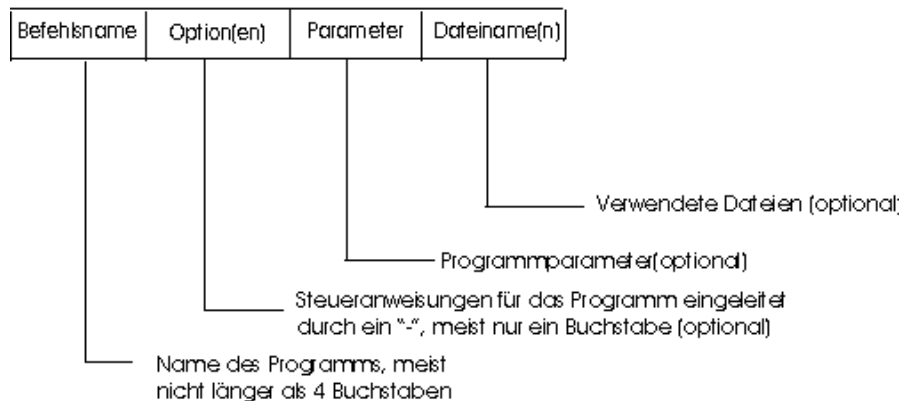
Start und Stop des Systems

Bei Unix-Rechnern kann der Rechner nicht einfach ein- und ausgeschaltet werden, denn speziell beim Ausschalten müssen alle Benutzerprozesse zuvor abgeschlossen und alle Dateien ordnungsgemäß geschlossen sein. In der Regel laufen Unix-Systeme auch Tag und Nacht durch.

Beim Einschalten des Rechners werden zunächst Systeminitialisierungsroutinen durchlaufen und die einzelnen Platten des Systems in das Dateisystem eingebunden ("mount"). Je nach BS-Version gelangt das BS dann gleich in den Mehrbenutzerbetrieb oder in den Einzelbenutzerbetrieb (single user mode). Dieser Modus ist speziell für die Systemwartung notwendig, wenn kein anderer Benutzer den Rechner verwenden darf (z. B. Generieren einer neuen Systemversion, Benutzerverwaltung, Datensicherung, etc.). Vom Einzelbenutzerbetrieb wird dann der normale Mehrbenutzerbetrieb gestartet. Beim Abschalten des Systems wird umgekehrt verfahren. Alle noch laufenden Prozesse werden gestoppt (normalerweise mit vorheriger Warnung der noch aktiven Benutzer, damit diese ihre Arbeit in Ruhe beenden können), das Dateisystem aktualisiert (Schließen offener Dateien, Wegschreiben von Pufferbereichen) und in den Einzelbenutzerbetrieb übergegangen. Danach kann abgeschaltet werden.

Kommandosyntax (Befehlsaufbau)

Alle Befehle (= Dienstprogramme) von Unix haben einen einheitlichen Aufbau; die Eingabe wird durch Drücken der RETURN-Taste abgeschlossen:



Fast jedem Kommando kann eine Liste von Dateien mitgegeben werden, auf die das Kommando dann angewendet wird. Fehlt die Dateiliste, wird in der Regel die Standardeingabe - normalerweise die Tastatur - als Eingabedatei verwendet. Wie später noch genauer gezeigt wird, verwendet Unix die Jokerzeichen (Wildcards) Stern (*) und Fragezeichen (?), um beliebige Zeichen innerhalb eines Dateinamens zu kennzeichnen. DOS und Windows haben diese Methode übernommen. Im Gegensatz zu DOS und Windows werden diese Wildcards jedoch **nicht vom Programm, sondern von der Shell zu Dateinamen expandiert**. Deshalb "sieht" jedes Unix-Programm nur eine mehr oder weniger lange Dateiliste.

Grundlagen des X-Window-Systems

Puristen arbeiten direkt auf der Kommandozeile und auch bei Telnet-Logins wird Kommandozeilenorientiert gewerkelt. Am besten startet man aber gleich nach dem Einloggen eine grafische Benutzeroberfläche (vergleichbar etwa dem Windows-System von Microsoft). Das erreicht man durch Eingabe des Befehls *startx* oder, falls man an einem X-Terminal sitzt, durch direktes Einloggen am Grafikbildschirm. Es dauert jetzt eine Weile (eine Sanduhr ist solange sichtbar) bis das X-Window-System (oder "X"), wie diese Oberfläche heißt, hochgefahren ist. Sobald das Starten von X beendet ist, nimmt der Mauszeiger die übliche Form eines Pfeils an und man kann mit der Arbeit beginnen.

Nahezu jedes Fenster hat unter X einen Rahmen. Mit Hilfe dieses Rahmens kann man die Dimensionen und die Position des Fensters verändern. Zunächst kann man durch "Klicken und Ziehen" auf die *Titelleiste* des Fensters das Fenster als Ganzes bewegen und an einer anderen Stelle des Bildschirms "loslassen". "Klicken und Ziehen" bedeutet, daß man mit dem Mauszeiger auf die Titelleiste zeigt, dann die linke Maustaste drückt, festhält, und bei gedrückter Taste die Maus bewegt. Wenn das Fenster die erwünschte Position erreicht hat, läßt man die linke Maustaste wieder los.

Wenn man jetzt mit der Maus die *untere rechte Ecke* des Fensters ansteuert, verwandelt sich der Mauszeiger selber in eine "Ecke" (ausprobieren!). Wenn der Mauszeiger so aussieht, dann kann man durch klicken+ziehen (s.o.) die Fenstergröße verändern. Es ist *nicht* ratsam, die Größe der zwei Textfenster zu ändern, die gleich am Anfang erscheinen, weil viele Programme davon ausgehen, daß diese Fenster eine feste Größe haben (nämlich 80 Zeichen Breite und 25 Zeilen Höhe).

Weiterhin sind am oberen Rand des Fensters, direkt rechts neben der Titelleiste, zwei Knöpfe zu sehen: einer enthält ein großes und einer ein kleines Quadrat. Durch einmaliges, kurzes Klicken auf den Knopf mit dem großen Quadrat bewirkt man, daß das Fenster seine volle Größe annimmt, d.h. es wird in der Regel über den ganzen Bildschirm vergrößert. Ein weiteres Klicken auf diesen Knopf setzt

die Fenstergröße wieder auf die Normalgröße zurück.

Der zweite Knopf, der mit dem kleinen Quadrat, bewirkt, daß das Fenster zum Symbol verkleinert wird. Das Symbol landet dann in der "Icon-Box", die in der Abbildung des gesamten X-Window-Bildschirms (oben) in der linken unteren Ecke zu sehen ist. Ein Doppelklick auf ein Icon führt dazu, daß das entsprechende Fenster wieder geöffnet und im Vordergrund angezeigt wird (d.h. ohne daß es durch andere Fenster überdeckt wird). Man kann also diese Icons auch dazu benutzen, unsichtbare (weil verdeckte) Fenster wieder in den Vordergrund zu holen.

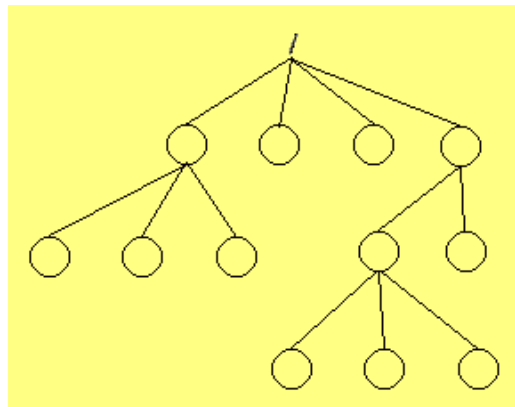
Den Inhalt einiger Fenster (xterm) kann man mit Hilfe der Rollbalken auf der rechten Seite bewegen, und so auch bereits nach oben weggerollte Zeilen wieder sichtbar machen. Dazu plaziert man den Mauszeiger auf den schwarzen Bereich des Rollbalkens und hält die *mittlere* Maustaste gedrückt, während man die Maus nach oben oder unten bewegt.

Das X-Window-System wird verlassen, indem man mit der Maus auf den *Hintergrund* des Bildschirms klickt, dabei die linke Maustaste aber nicht losläßt, sondern bei gedrückter Taste die Maus nach unten zieht. Es erscheint ein Menü und man kann den Punkt "Exit" oder "Quit" ansteuern und dann die Maustaste loslassen. Nach einer weiteren Abfrage ("OK") ist man dann wieder auf der grünen Kommandozeile und kann mit dem Kommando "exit" die Sitzung beenden. Bei Linux geht das auch durch Drücken von Ctrl-Alt-Backspace.

1.5 Das Dateisystem

1.5.1 Struktur und Zugriffsrechte

Das Dateisystem ist hierarchisch strukturiert in Form eines (umgedrehten) Baumes.



Da Unix ein Multiuser-System ist, muß der Zugriff auf einzelne Dateien und Verzeichnisse vom System schon so geregelt werden, daß kein Benutzer die Daten der anderen Benutzer manipulieren oder vertrauliche Daten unbefugterweise einsehen kann. Das wird dadurch gewährleistet, daß jede Datei und jedes Verzeichnis einen *Besitzer* (Owner) hat, und dieser Besitzer kann mit Hilfe bestimmter Befehle festlegen, welcher der anderen Benutzer auf welche Weise auf seine Dateien zugreifen darf. Die meisten Dateien und Verzeichnisse in einem Unix-System gehören natuergemäß dem Systemverwalter (der den Usernamen "root" trägt) und jeder Benutzer kann auf diese Dateien lesend zugreifen, sie aber nicht verändern (zu dieser Art von Dateien gehören zum Beispiel alle Anwendungsprogramme, die das System zur Verfügung stellt).

Man unterscheidet grob drei Dateitypen (weitere Typen weiter unten):

Betriebssystem UNIX/Linux

- Normale Dateien (normal files)
 - ◆ Dateien im üblichen Sinn: Text- oder Binärdateien
 - ◆ Das System unterstützt keine Dateistruktur (keine Datei-Header)
 - ◆ Dateien sind Bytefolgen (Strukturierung nur durch das Programm)
 - ◆ Zugriffsarten:
 - ◇ sequentiell
 - ◇ random access (Bytepositionierung)
- Verzeichnisse (directories)
 - ◆ Strukturierung des Dateisystems
 - ◆ Zugriff auf normale Dateien auch über mehrere Namen
 - ◆ Verzeichnisse werden auf der Platte wie Dateien gespeichert
- Spezialdateien (special files)
 - ◆ E/A-Geräte werden als Spezialdatei eingetragen
 - ◆ Zugriff auf Gerät <-> Zugriff auf eine Datei
 - ◆ gleicher Schutzmechanismus wie für normale Dateien
 - ◆ es gibt blockorientierte und zeichenorientierte Geräte
 - ◇ Platte /dev/hda1
 - ◇ Drucker /dev/lp
 - ◇ Speicher /dev/mem
 - ◇ Terminal /dev/tty

Unix arbeitet mit einem Filesystem, das auf den ersten Blick dem von DOS sehr ähnlich ist (nur auf den ersten). Also gibt es eine Baumstruktur von verschiedenen Verzeichnissen (Directories), in der sich jede Datei irgendwo befindet. Beachten Sie aber, daß ein Directory beim Anzeigen zunächst genauso aussieht, wie eine Datei!. Die einzelnen Verzeichnisnamen werden durch normale Schrägstriche (/) getrennt, NICHT durch Backslashes (\) wie bei DOS/Windows!

Nach dem Einloggen landen Sie in Ihrem sogenannten Homedirectory (Meist /home/username'). Es gehört Ihnen ganz alleine, damit können Sie machen, was Sie wollen. Zum Beispiel können Sie dort Dateien oder weitere Verzeichnisse anlegen. An dieser Stelle gleich einige wichtige Punkte:

Dateinamen

Dateinamen können bei Unix aus Groß und Kleinbuchstaben bestehen, wobei die unterschiedliche Schreibweise auch unterschiedliche Dateinamen bezeichnet. Die Datei 'test' ist also eine andere als 'Test' oder gar 'TEST'.

Dateilisten

Wo Dateieingaben erwartet werden, können in der Regel beliebig viele Dateinamen stehen.

Sonderzeichen

Dateinamen können ausser dem '/' so ziemlich alle Zeichen enthalten. Auch Leerzeichen oder Steuerzeichen.

Pfadnamen

Dateien werden über ihren Pfad spezifiziert: Angabe von Verzeichnissen und zum Schluß des Namens, getrennt durch Schrägstrich.

- vollständiger Pfadname, beginnend bei root (absoluter Pfad) Angabe beginnt mit dem Schrägstrich. Zum Beispiel:

`/usr/projekt1/meier/test/steuerung`

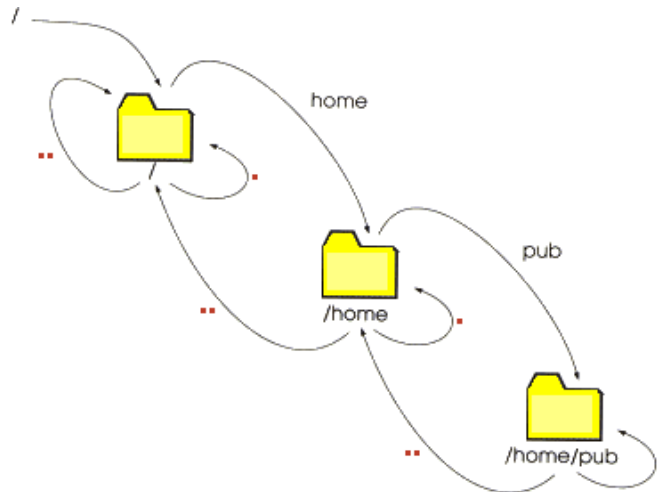
- relativer Pfadname, beginnend beim aktuellen Verzeichnis (kurzer Pfad), zum Beispiel:

`steuerung` (akt. Verz.: `/usr/projekt1/meier/test`)

Betriebssystem UNIX/Linux

test/steuerung (akt. Verz.: /usr/projekt1/meier)

Das *aktuelle* Verzeichnis ist immer jenes, dessen Inhalt wir gerade bearbeiten. Das aktuelle Verzeichnis läßt sich jederzeit wechseln, aber ein Verzeichnis ist zu einem bestimmten Zeitpunkt immer das aktuelle und alle unsere Kommandos beziehen sich dann auf dieses eine, aktuelle, Verzeichnis.



Eine besondere Erwähnung verdienen die beiden Verzeichnis-Einträge "." und "..". Das sind Stellvertreter für Verzeichnisnamen, die man statt der realen Namen (abkürzend) benutzen kann. Und zwar bezeichnet "." das jeweils aktuelle Verzeichnis, und ".." das dem aktuellen Verzeichnis übergeordnete Verzeichnis.

Schutzmechanismen

- Jeder Unix-Benutzer hat eine Benutzerkennung (user id, kurz: uid) mit der er sich gegenüber dem BS identifizieren kann.
- Jeder Unix-Benutzer gehört einer Gruppe an und besitzt damit eine Gruppen-ID, kurz:gid)
- Jede Datei hat einen Eigentümer und eine Gruppe, die bei der Erzeugung der Datei eingetragen werden
- Jeder Benutzer kann seine Dateien explizit einem anderen Benutzer (bzw. einer anderen Gruppe) "schenken".
- Jede Datei besitzt 12 voneinander unabhängige Schutzbits

SUI	SGI	STI	R	W	X	R	W	X	R	W	X
			User			Group			Others		

Bedeutung der drei Schutzbits SUID, SGID und STICKY

Wenn das SUID-Bit (Set User ID) gesetzt ist, behält das Programm für die Dauer der Ausführung die Rechte des Programmeigentümers und nicht jene dessen, der die Programme aufruft. Setzen durch das Kommando: "chmod u+s datei". Anzeige: "s" statt "x" bei den User-Rechten. Dazu ein Beispiel:

Alle Benutzer sind in einer speziellen Datei gespeichert, die nur der Superuser ändern darf - sonst könnte ja jeder einen neuen Benutzer eintragen.

Jeder Benutzer kann aber sein Passwort ändern, das auch in dieser Datei steht. Dazu muß er schreibend auf die Datei zugreifen - obwohl er dazu keine Berechtigung besitzt. Das Programm "passwd" gehört dem Superuser, hat das SUID-Bit gesetzt und kann so auf die User-Datei schreibend

zugreifen.

Wenn das SGID-Bit (Set Group ID) gesetzt ist, hat das Programm die Rechte der Gruppe, zu der es gehört. Dieses Feature wird z. B. beim Drucker-Spooling verwendet. Bei Dateien ohne Ausführungsrecht sorgt dieses Bit dafür, daß die Datei nur von einem Prozess geöffnet werden kann (Vermeiden von Verklemmungen).

Bei Verzeichnissen hat das SGID-Bit eine andere Aufgabe. Dateien, die in ein SGID-Verzeichnis kopiert werden, erhalten automatisch die Gruppe des Verzeichnisses (man muß also nicht mehr explizit die Gruppe setzen, um den Mitgliedern einer Gruppe Zugriff zu ermöglichen). Setzen durch das Kommando: "chmod g+s datei". Anzeige: "s" statt "x" bei den Gruppen-Rechten ("l" bei Daten-Dateien).

Das STICKY-Bit sollte früher den Systemdurchsatz verbessern. Programme, bei denen dieses Bit gesetzt ist, verbleiben nach dem ersten Aufruf im Speicher und starten bei den folgenden Aufrufen schneller. Heute ist das nicht mehr nötig.

Bei Verzeichnissen dient dieses Bit der Systemsicherheit. Auch wenn im Verzeichnis für alle User Schreibrecht existiert (= Löschen und Anlegen von Dateien), können bei gesetztem Sticky-Bit nur Dateien gelöscht werden, die einer folgenden Bedingungen genügen:

- die Datei gehört dem Benutzer, der sie löschen will
- das Verzeichnis, in dem die Datei liegt, gehört dem Benutzer
- der Benutzer hat Schreibrecht für die Datei
- der Superuser will die Datei löschen

Setzen durch das Kommando : "chmod +t datei". Anzeige: "t" statt "x" bei den "Others"-Rechten.

Historisches: Das System der Zugriffsrechte, insbesondere SUID, SGID und STICKY, wurden von den Entwicklern patentiert. Interessant an der Patentschrift ([Bild der ersten Seite](#)) war, das alles als Digitalschaltung dargestellt wurde, denn Software war damals noch nicht patentierbar. Das Patent wurde ein Jahr nach Erteilung von den Entwicklern freigegeben.

Die Implementierung des Dateisystems

Ein physisches Dateisystem ist eine dateiorientierte Struktur auf einem logischen Datenträger (Partition, Slice). Unix bietet die Möglichkeit, mehrere dieser logischen Datenträger auf einem physischen Datenträger (Festplattenlaufwerk) zu verwalten. Desweiteren können neben den permanent vorhandenen physischen Dateisystemen (Dateisysteme auf dem Systemlaufwerk) auch weitere Dateisysteme auf montierbaren Datenträgern (Festplattenlaufwerke anderer Rechnersysteme, Wechselpplatten, Disketten, usw.) in den Dateibaum eingehängt bzw. entfernt werden.

Schaut man sich ein physisches Dateisystem genauer an, so erkennt man den Betriebssystemblock als kleinste Einheit (im Bereich von 512 Byte bis 16 kByte). Das physische Dateisystem wird in in vier Bereiche aufgespalten:

- Boot-Block:
Er enthält im root-Dateisystem ein Umlader-Programm, welches das eigentliche Unix-System (Betriebssystemkern) in den Arbeitsspeicher lädt. Dieser Bereich ist bei heutigen Systemen leer.
- Super-Block:
Er enthält alle relevanten Verwaltungsinformationen zu einem physischen Dateisystem:
 - ◆ Name und Größe des physischen Dateisystems
 - ◆ Größe der nachfolgenden Bereiche des physischen Dateisystems (Inodeliste, Nutzdatenbereich)
 - ◆ Verweise auf die Liste der freien Datenblöcke und die Liste der freien Inodes

- ◆ Datum der letzten Sicherung und Modifikation
- ◆ und weitere Angaben
- Inodeliste:
 - Sie stellt ein Inhaltsverzeichnis aller in dem Dateisystem existierenden Dateien dar. Sie besteht aus einer Folge von Inodes (Dateiköpfen), die die Verwaltungsdaten zu jeder Datei enthalten.
- Nutzdatenbereich: Hier befinden sich die freien Blöcke, Datenblöcke (Inhalte von Dateien und Verzeichnissen) und Referenzblöcke, die zur Adressierung der Datenblöcke eingesetzt werden.

Die Größe der einzelnen Bereiche wird bei der Initialisierung eines physischen Dateisystems festgelegt und kann im nachhinein nicht mehr dynamisch verändert werden (außer im Unix-System AIX von IBM). Das dazu notwendige Kommando ist 'mkfs' (make file system). Die Bereiche eines physischen Dateisystems sind auf die Kapazität eines logisch Datenträgers und damit maximal auf die Gesamtkapazität eines Festplattenlaufwerks beschränkt, d. h. festplattenübergreifende physische Dateisysteme sind nicht möglich.

Die Verwaltungsinformation über Dateien steht also nicht wie bei DOS oder Windows in den Dateien selbst, sondern in den Inodes. Jede Datei besitzt einen eigenen Inode (eine eigene Datei-Nummer). Der Zugriff erfolgt entweder sequentiell (Datei ist ein Strom von Bytes ohne weitere Strukturierung) oder wahlfrei (random access) durch Positionierung auf ein bestimmtes Byte in der Datei. Das BS hat einen Cache-Mechanismus implementiert, der einen Teil der Datei im Speicher hält. Ziel: Reduzierung der Plattenzugriffe.

Problem: Dateien müssen regelmäßig aktualisiert werden (Cache-Inhalt auf die Platte schreiben), Gefahr der Inkonsistenz von Daten bei Prozess-Abbruch oder Stromausfall.

Inodes (I-Knoten)

Da alle Inodes die identische Länge von 128 Byte haben, erübrigt sich die Abspeicherung der Knotennummer innerhalb des Inodes. Ein Inode enthält die relevanten Verwaltungsattribute einer Datei:

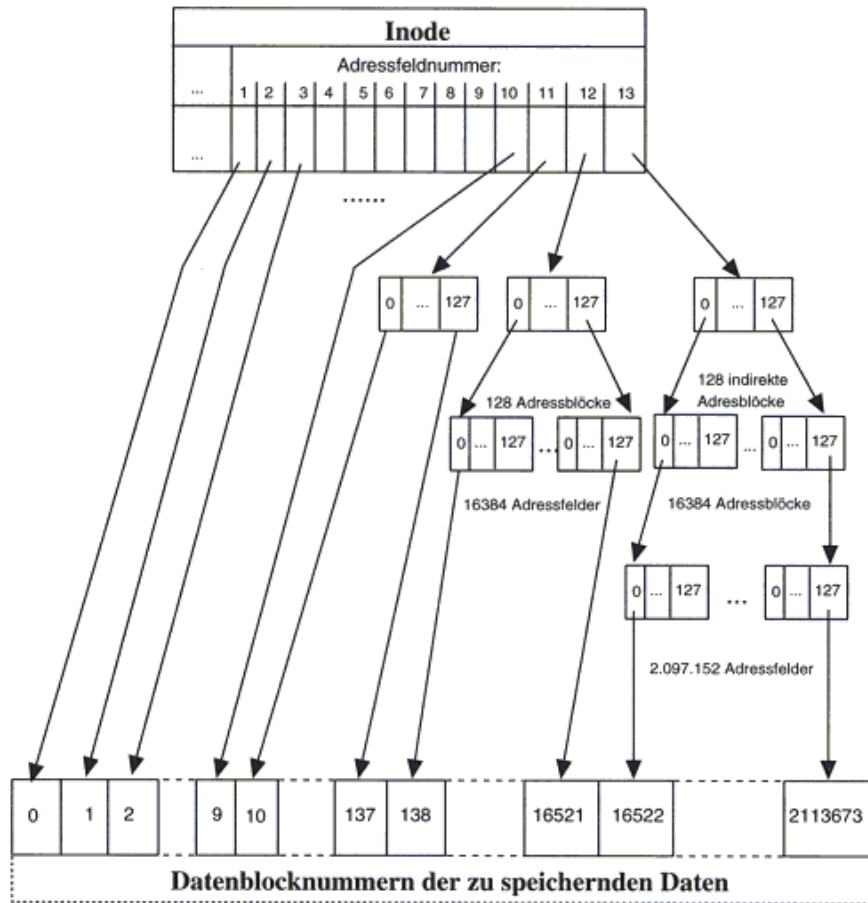
- Dateityp und Zugriffsrechte
- Eigentümer, Gruppe
- Link-Zähler (siehe später)
- Zeitstempel
 - ◆ Datum der letzten Änderung am i-Node
 - ◆ Datum der letzten Modifikation
 - ◆ Datum des letzten Zugriffs
- Dateigröße in Byte
- 10 Verweise auf Plattenblöcke
- Einfach indirekter Verweis
- Zweifach indirekter Verweis
- Dreifach indirekter Verweis

Zu beachten ist, daß der Name der Datei nicht aufgeführt wird. Diese Systemarchitektur ermöglicht es, unter mehreren (verschiedenen) Namen als Einträge von Verzeichnissen die gleiche Datei (genauer gesagt Inode und Dateiiinhalt) anzusprechen. Der Inode fungiert somit als Bindeglied zwischen dem Namen und dem Inhalt einer Datei. Für die Adressierung der Inhalte einer Datei ergibt sich eine bestimmte Verweisstruktur.

Eine Datei mit bis zu 10 Plattenblöcken (Blockgröße 512 oder 1024 Bytes) kann also direkt angesprochen werden. Größere Dateien haben zusätzlich einen Verweis (einfach indirekt) auf einen

Betriebssystem UNIX/Linux

Datenblock, der seinerseits 128 Verweisfelder enthält. Reicht das noch nicht, wird zweimal (128 Blöcke mit je 128 Verweisen) oder dreimal indiziert (Dateigröße bis 2 GByte bei 512-Byte-Blöcken, bis 16 GByte bei 1KByte-Blöcken).



Links

Dies sind Verweise auf Dateien, d. h. über ein Link kann eine Datei unter einem zweiten Namen angesprochen werden. Eine Datei mit mehreren Links hat also mehrere Namen, ist jedoch nur einmal vorhanden. Beim Löschen eines Links - das für den Benutzer wie eine "normale" Datei wirkt - wird nur der Link und nicht die Datei gelöscht. erst beim Löschen des letzten Links verschwindet auch die Datei. Links sind nicht an den Eigentümer der Datei gebunden, d. h. Benutzer A kann ein Link auf die Datei von Benutzer B haben. Die Dateizugriffsrechte bleiben davon unberührt. **Problem:** Benutzer A löscht eine Datei, auf die Benutzer B ein Link hat. Dann bleibt die Datei zwar wegen des zweiten Links erhalten, wenn nun aber Benutzer A gelöscht wird, dann gibt es eine Datei ohne Eigentümer.

Das System der Links wird generell verwendet. So gibt es unter Unix keinen Systemaufruf zum Löschen einer Datei, sondern lediglich einen Unlink-Call. Wenn der Link-Zähler einer Datei auf Null gesunken ist (durch entsprechend viele Unlink-Aufrufe), wird deren Inode und der durch die Datei belegte Plattenplatz freigegeben.

Der Superblock

Der Superblock bildet die Verwaltungseinheit für eine Platte oder Plattenpartition (= Dateisystem). Er enthält folgende Felder:

- Größe des Dateisystems
- Anzahl und Liste der freien Blöcke
- Index auf den nächsten freien Block der Liste
- Größe der Inode-Liste - Anzahl und Liste der freien Inodes
- Index auf den nächsten freien Inode der Liste
-

Geräte

Alle Peripheriegeräte des Unix-Rechners (Platte, Floppy-Disk, Magnetband, Terminals, Drucker, usw.) werden über das Dateisystem, also prinzipiell wie Dateien angesprochen. Dabei handelt es sich um speziell gekennzeichnete Dateien, in denen keine Daten gespeichert werden, sondern die vielmehr eine Verbindung zum Linux-Kernel herstellen. Diese "Spezialdateien" ("special files", normalerweise im Verzeichnis /dev) stellen die Verbindung zwischen Dateisystem und den sogenannten Geräte-Treibern (device drivers) her. Die Gerätedateien sorgen also für den Anschluß der Geräte an das Unix-Dateisystem. Devices ermöglichen den Zugriff auf viele Hardware-Komponenten des Rechners, also etwa auf Festplatten, Diskettenlaufwerke, serielle und parallele Schnittstellen, den Arbeitsspeicher (RAM) etc.

Gerätedateien dienen zur Abwicklung des Datenverkehrs zwischen den Programmen und der Peripherie. Da sie, wie die abstrakten Komponenten (normale Dateien, Verzeichnisse, usw.) einheitlich in den Systembaum integriert werden, sind auch für sie die Zugriffsschutzmechanismen und Ein- /Ausgabeumleitung gültig. Die Geräte werden durch logische Namen angesprochen, die die Gerätetreiber der Peripheriegeräte bezeichnen. Intern wird jedes Gerät durch eine Treiber-Nummer (major device number) und eine Geräte-Nummer (minor device number) markiert. Diese Nummern sind im `ls -l` Eintrag in dem Feld enthalten, das die Dateigröße angibt.

- **Major Device Number:** Angabe des Gerätetyps (Platte, Terminal, Drucker, etc.)
- **Minor Device Number:** Angabe über die E/A-Schnittstelle oder nähere Umstände der Ansteuerung (z.B. Band zurückspulen/nicht zurückspulen)

Die *Major Device Number* gibt an, welcher Treiber des Linux-Kernels für die Verwaltung zuständig ist. Bei vielen Treibern kann durch die *Minor Device Number* zwischen verschiedenen (verwandten) Einzelgeräten unterschieden werden, etwa beim Treiber für die seriellen Schnittstellen. Der Zugriffstyp gibt an, ob die Geräte gepuffert sind (das ist bei allen blockorientierten Geräten wie Festplatten etc. der Fall) oder nicht (zeichenorientierte Geräte wie serielle oder parallele Schnittstellen).

Ein Gerät kann durchaus auch über verschiedene Treiber angesprochen werden. Normalerweise werden die Geräte vom Benutzer nicht direkt, sondern über Dienstprogramme angesprochen. Wenn jemand z. B. den Drucker direkt anspricht, während ein anderer Benutzer über den Spooler ausdruckt, gibt es "Datenmüll" auf dem Papier. Zwei interessante Geräte können jedoch alle Benutzer ansprechen:

- `/dev/null`
Daten, die an dieses Gerät geschickt werden, landen in einem "schwarzen Loch" - ideal für Ausgaben eines Programms. Wenn von diesem Gerät gelesen wird, liefert es immer "End Of File".
- `/dev/zero` ist das zweite "Spezialgerät". Beim Lesen von diesem Gerät werden beliebig viele Nullbytes geliefert. Man verwendet es z. B. zum Füllen von Dateien oder wenn man eine "Dummy-Engabe" benötigt.

Es gibt bei Unix prinzipiell zwei Typen von Treibern:

- **Zeichentreiber Für zeichenorientierte Geräte** (Terminal, Drucker, etc.). Eingehende Daten werden in der Regel gepuffert und dann vom Programm aus dem Puffer gelesen, beim Schreiben erfolgt analog die Pufferung in Ausgangsrichtung.
- **Blocktreiber für blockorientierte Geräte** (z. B. Platte). Hier ist meist noch ein Software-Cache (Schreib-Lesepuffer mit entsprechenden Zugriffsroutinen) zur Beschleunigung der E/A-Operationen implementiert. Zur Einrichtung von Treibern und Zuweisung der "device numbers" existiert ein passendes Kommando (`mknod`), das nur vom Superuser aufgerufen werden kann. Die Gerätedateien enthalten selbst keine Treiber, sondern legen nur fest, welcher Treiber im Kernel über die Gerätedatei angesprochen werden soll.

Wenn Sie mit `ls -l` das Inhaltsverzeichnis von `/dev` betrachten, werden statt der Dateigröße die Device-Nummern (major und minor) ausgegeben. Das erste Zeichen der Zugriffsbits lautet `b` oder `c` (block- oder zeichenorientiert). Neue Device-Dateien können mit dem Kommando `mknod` eingerichtet werden. Manche Dateien im `/dev`-Verzeichnis sind in Form von Links realisiert. So zeigt beispielsweise `/dev/mouse` auf die Device-Datei, die für die Schnittstelle zuständig ist, an der die Maus tatsächlich angeschlossen ist (`/dev/ttyS0` oder `/dev/psaux` bei PS/2-Mäusen).

Um gezielt steuern zu können, welcher Benutzer auf welche Devices zugreifen darf, sind den Devices unterschiedliche Benutzergruppen zugeordnet. Beispielsweise sind die Devices `/dev/ttyS*` für die seriellen Schnittstellen üblicherweise der Gruppe `uucp` zugeordnet. Wenn der Systemadministrator möchte, dass die Userin `gundel` über die serielle Schnittstelle direkt auf ein Modem zugreifen darf, fügt er `gundel` zur Gruppe `uucp` hinzu (mit dem Kommando `usermod -G` oder durch Bearbeiten der Datei `/etc/group`).

Eine vollständige Beschreibung aller unter Linux zurzeit definierten Devices samt der dazugehörigen Device-Nummern finden Sie in der Datei `/usr/src/linux/Documentation/devices.txt` (nur, wenn der Kernel-Quellcode installiert ist).

Die Zukunft des Device-Systems

Das gegenwärtige System von Device-Dateien ist aus zwei Gründen unflexibel und unpraktisch:

- Um jede nur denkbare Hardware des Benutzers zu unterstützen, erzeugen die meisten Distributionen bei der Installation eine riesige Anzahl von Device-Dateien. (Bei Red Hat 9 gab es beispielsweise beinahe 8000 derartige Dateien!) Tatsächlich genutzt werden auf jedem Rechner aber höchstens ein paar Prozent der Dateien.
- Aus historischen Gründen handelt es sich bei der Device-Nummer bis einschließlich Kernel 2.4 um eine 16-Bit-Zahl. Da jedes Jahr neue Hardware-Komponenten hinzukommen, wird der verfügbare Zahlenraum für neue Devices immer enger.

Die größte Neuerung in Kernel 2.6 ist die Vergrößerung der Device-Nummern auf 64 Bits. Bereits vorhandene Device-Nummern sollten sich dadurch nicht ändern; ganz sind Kompatibilitätsprobleme aber nicht auszuschließen. Offen ist noch, ob sich am prinzipiellen Umgang mit Device-Dateien in Zukunft etwas ändern wird. Salopp gesagt erstellt und entfernt es Einträge in `/dev`, basierend auf der aktuellen Systemkonfiguration. Prinzipiell ist `udev` ein modulares System zur automatischen Erstellung von Gerätedateien in `/dev`.

Im Verzeichnis `/dev` befinden sich normalerweise Gerätedateien (Device Nodes), die, wie oben beschrieben, den Zugriff auf Geräte (z. B. Festplatten, Maus, Soundkarte) erlauben. Diese Gerätedateien werden normalerweise mit `MAKEDEV` angelegt, denn ohne Gerätedatei ist kein Zugriff auf ein Gerät möglich. Aus diesem Grunde existieren normalerweise Unmengen von Gerätedateien für größtenteils nicht vorhandene Geräte. Das `udev`-Dateisystem legt diese Gerätedateien dynamisch an, der Einsatz von `MAKEDEV` ist nicht mehr nötig und es existieren nur noch Gerätedateien für existierende Geräte mit Treiber. Mit anderen Worten: nie wieder nach nicht existierenden Geräten

suchen oder Device Nodes anlegen müssen.

Dies wurde erreicht durch Überwachen der vom Programm `hotplug` generierten Ereignisse im System und Auslesen von Informationen zu diesen Ereignissen aus dem Dateisystem `sysfs`. `udev` arbeitet unter Benutzung von `hotplug`-Aufrufen des Kernels, wann immer ein Gerät zum Kernel hinzugefügt oder daraus entfernt wird. Die Namensgebung und Zugangsberechtigungen werden im User-Space ausgeführt. Die Ziele des `udev` Projekts sind:

- Läuft im User-space
- Erstellt/entfernt dynamisch Gerätedateien
- Liefert konsequente Benennung
- Liefert ein User-space API

Um diese Funktionen zu liefern, wird `udev` in drei unterschiedlichen Projekten entwickelt: `namedev`, `libsysfs` und natürlich `udev`.

- `namedev`
`namedev` erlaubt es, Geräte unabhängig vom `udev`-Programm zu benennen. Dies ermöglicht flexible Benennungsrichtlinien und Namensschemata. Das Subsystem zur Gerätebenennung liefert ein Standardinterface, das `udev` benutzen kann. Derzeit gibt es nur ein einzelnes Benennungsschema von `namedev`, `LANANA`, das auch von der Mehrheit der Linux-Systeme verwendet wird. `namedev` verwendet eine fünfstufige Prozedur, um den Namen eines bestimmten Gerätes herauszufinden. Wenn in einem dieser Schritte der Gerätename gefunden wird, verwendet es diesen Namen. Diese Schritte sind:

1. Beschriftung oder Seriennummer
2. Bus-Gerätenummer
3. Bus-Topologie
4. Statisch vergebener Name
5. Vom Kernel gelieferter Name

Der erste Schritt überprüft, ob das Gerät ein einzigartiges Identifikationsmerkmal hat. Zum Beispiel haben USB-Geräte eine einzigartige USB-Seriennummer und SCSI-Geräte eine einzigartige UUID. Wenn `namedev` eine Übereinstimmung zwischen dieser einzigartigen Nummer und einer gegebenen Konfigurationsdatei findet, dann wird der von der Konfigurationsdatei gelieferte Name verwendet.

Der zweite Schritt überprüft die Bus-Gerätenummer. Für nicht-hot-swappable Umgebungen ist diese Prozedur ausreichend, um ein Hardware-Gerät zu identifizieren. Zum Beispiel verändern sich PCI-Busnummern selten in der Lebenszeit eines Systems. Findet `namedev` eine Übereinstimmung mit dieser Position und einer gegebenen Konfigurationsdatei, wird der von der Konfigurationsdatei gelieferte Name verwendet.

Auch die Bus-Topologie ist ein eher statischer Weg zur Definition von Geräten. Wenn die Position des Gerätes zu einer vom Benutzer gelieferten Einstellung passt, wird der beiliegende Name verwendet.

Der vierte Schritt ist eine einfache Stringersetzung. Wenn der Kernel-Name (der Standardname) des Device zu einem gegebenen Ersatzstring passt, wird der Ersatzname stattdessen verwendet.

Der letzte Schritt nimmt den vom Kernel gelieferten Standardnamen. In den meisten Fällen ist dies ausreichend, da es zur augenblicklich üblichen Gerätebenennung passt.

- `libsysfs`

Betriebssystem UNIX/Linux

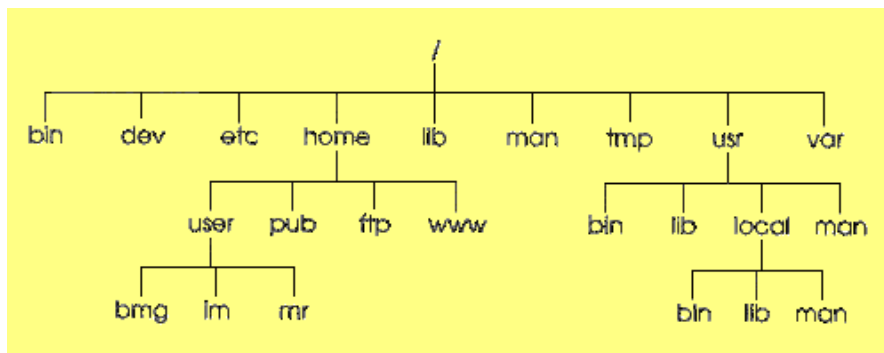
udev interagiert mit dem Kernel durch das `sysfs`-Pseudodateisystem. Das `libsfs`-Projekt liefert ein Standard-API, um auf die durch das `sysfs`-Dateisystem gegebenen Informationen zuzugreifen. Dies erlaubt eine Abfrage aller Art von Hardware, ohne dass man Vermutungen über die Art der Hardware anstellen muss.

- udev

Jedes Mal, wenn der Kernel ein Update in der Gerätestruktur feststellt, ruft er das Programm `hotplug` auf. `Hotplug` führt die Anwendung aus, welche im Verzeichnis `/etc/hotplug.d/default` verlinkt ist. `Hotplug` übergibt die Informationen vom Kernel an das `udev`-Programm, das die notwendigen Aktionen (Erstellen oder Entfernen von Gerätedateien) in `/dev` ausführt.

1.5.2 Unix-Dateistruktur

So oder so ähnlich stellt sich die Dateistruktur von Unix dar:



Stimmt, das Bild war weiter oben schon zu sehen. Wenn wir uns ein Verzeichnis mit dem Kommando `"ls -l"` ansehen, erhalten wir eine solche Liste:

```
total 1093
-rw-r--r--  1 root   root   116547 May 25  1997 System.map
drwxr-xr-x  2 root   root    1024 Sep 23  1996 bin/
drwxr-xr-x  2 root   root    1024 May 25  1997 boot/
drwxr-xr-x  2 root   root    1024 Oct 27  1996 cdrom/
drwxr-xr-x  3 root   root   20480 May  4  15:28 dev/
drwxr-xr-x  7 root   root    2048 May  4  16:05 etc/
drwxr-xr-x  5 root   root    1024 Dec  7  1997 home/
drwxr-xr-x  3 root   root    1024 Sep 23  1996 lib/
drwxr-xr-x  5 root   root    1024 Sep 23  1996 local/
drwxr-xr-x  2 root   root   12288 Sep 23  1996 lost+found/
drwxr-xr-x  2 root   root    1024 Sep 23  1996 mnt/
dr-xr-xr-x  5 root   root      0 May  4  1999 proc/
drwx----- 5 root   root    1024 Sep 21  1997 root/
drwxr-xr-x  4 root   root    2048 Sep 23  1996 sbin/
drwxrwxrwx  4 root   root    1024 Apr  6  09:18 tmp/
drwxr-xr-x 18 root   root    1024 Apr 25  1997 usr/
drwxr-xr-x 14 root   root    1024 Apr 25  1997 var/
-rw-r--r--  1 root   root  407793 May 25  1997 vmlinuz
```

Betrachten wir diese Ausgabe von **rechts nach links**. Ganz rechts in jeder Zeile steht der Dateiname, jede Zeile ist der Eintrag für eine Datei. Links davon steht in drei Feldern Datum und Uhrzeit der letzten Modifikation der Datei. Im fünften Feld (links vom Monat) steht die Größe der Datei in Bytes (1 Byte entspricht einem Zeichen, also z.B. einem Buchstaben). Die weiteren zwei Felder links von der Dateigröße geben den Besitzer der Datei und dessen Gruppe an. Alle Dateien in diesem Verzeichnis gehören dem Benutzer "root", der Mitglied der Gruppe "root" ist. Schließlich stehen ganz links (in der ersten Spalte) die Zugriffsrechte. Das erste Zeichen dieser ersten Spalte zeigt an, ob es

Betriebssystem UNIX/Linux

sich beim entsprechenden Eintrag um eine normale Datei ("-"), um ein Verzeichnis ("d"), um ein Character-Device ("c"), um ein Blockdevice ("b") oder um einen Link ("l") handelt.

Verzeichnisstruktur

Verzeichnis	Beschreibung / Funktion
/	Hier sollte mit Ausnahme des Default-Kernels und der zugehörigen System-Map keine Dateien liegen
/bin	Systemverwaltungsprogramme, die nicht spezifisch für den Superuser sind (cp, ls oder more)
/boot	Lilo-Dateien, die nicht ausführbar und keine Konfigurationsdateien sind; hier liegen auch alternative Kernel
/dev	Geräte dateien
/etc	Konfigurationsdateien des Basissystems und von Lilo sowie die Benutzerdatenbank
/home	Heimatverzeichnisse der Benutzer
/lib	Shared Libraries des Basissystems
/lost+found	Nimmt Dateien auf, die beim Plattencheck anfallen
/mnt	Hier mountet man Laufwerke wie CD-ROM (/mnt/cdrom) und Diskette (/mnt/floppy)
/opt	Erweiterungspakete (in Unterverzeichnissen); hier sollte beispielsweise KDE oder Netscape Communicator installiert sein
/proc	Dateien des proc-Dateisystems - ein virtuelles Dateisystem, über das Informationen über die Hardware ermittelt werden können
/root	Home-Verzeichnis des Superusers (root)
/sbin	Die ausführbaren Dateien für Systemadministration und Startskripten
/tmp	Temporäre Dateien
/usr	Zweites Hauptverzeichnis für Programme (Unix System Resources)
/usr/X11R6	X-Window-System
/usr/dict	Wörterbücher zu ispell
/usr/doc	Doku
/usr/include	Header-Dateien für den Präprozessor
/usr/local	Lokale Anwendungen + Dateien (analog /usr)
/usr/info	Textdateien des Textinfo-Dokumentationssystems
/usr/man	Manuals (Online-Hilfe)
/usr/share	Architekturunabhängige Daten in verteilten Systemen mit verschiedenen Rechnerarchitekturen (Sparc, RS 6000 ...)
/usr/src	Quellcode für die Programme des Standardsystems
/usr/src/linux	Kernel-Sourcen; meist handelt es sich hier nur um einen symbolischen Link auf ein aussagekräftigeres Verzeichnis

/usr/spool	Wird aus Kompatibilitätsgründen als symbolischer Link auf /var/spool beibehalten
/var	Verzeichnis für die variablen Daten, falls /usr über NFS readonly gemountet ist; in /var/log liegen System-Logfiles

1.5.3 Sonderzeichen der Tastatur

Da die ersten Unix-Terminals Fernschreiber (Teletypes) waren, gab es keine Möglichkeit - wie am Bildschirm - einzelne Zeichen oder die ganze Eingabezeile mit dem Cursor zu löschen. Daher gibt es immer noch druckbare Löschrzeichen. Außerdem existieren einige andere Spezialzeichen:

Funktion	Standard Unix	Berkely Unix
Zeilenende	Return-Taste	Return-Taste
Lösche Zeichen	#	Backspace
Lösche Zeile	@	Ctrl-U
Programm abbrechen	Delete	Ctrl-C
Dateiende Eingabeende	Ctrl-D	Ctrl-D
Ausgabe anhalten	Ctrl-S	Ctrl-S
Ausgabe fortsetzen	Ctrl-Q	Ctrl-Q
Bildschirm wiederherstellen	Ctrl-L	Ctrl-L

Ctrl-S und Ctrl-Q waren früher bei langsamen, seriellen Terminals noch als Reaktionstest brauchbar. Heute sind sie nicht mehr zum Steuern der Ausgabe verwendbar, weil die Bildschirmanzeige zu schnell durchläuft. Trotzdem kann Ctrl-S Ärger machen, wenn Sie versehentlich auf die Taste kommen. Dann bleibt die Ausgabe stehen und man hat das Gefühl, der Rechner reagiert nicht mehr. Also erstmal versuchsweise Ctrl-Q drücken.

1.5.4 Dateikommandos

Die ersten Kommandos

Die meisten Unix-Befehle (= Programme) haben sehr kurze, kryptische Namen - und dafür zahllose Parametereinstellungen.

pwd (Print Working Directory)

Gibt das aktuelle Verzeichnis aus - damit man weiß, wo man überhaupt rumpfuscht.

cd (Change Directory)

Navigieren im Dateibaum - Wechsel des aktuellen Verzeichnisses. Wechsel nur in Verzeichnisse mit Execute-Permission möglich. Es gibt zwei Aufrufformen:

- cd Wechsel ins Home-Directory
- cd [Verzeichnis] Wechsel in das angegebene Verzeichnis

Es können beliebige Pfade angegeben werden, z. B.: cd /usr/hans/daten

ls (List)

ls [-Parameter] [pfadname] Auflisten von Dateien und Verzeichnissen mit der zugehörigen Information (Eselsbrücke: "laß sehen"). Durch Parameter wird die Ausgabe gesteuert. ls ohne Dateispezifikation listet das aktuellen Verzeichnis, sonst muß der Pfad angegeben werden. ls / listet z. B. das Wurzelverzeichnis auf Parameter: ls kennt sehr viele Parameter, die wichtigsten sind:

- **-d** nur Verzeichnisse auflisten
- **-a** alle Einträge auflisten
- **-l** Langform (alle Infos)
- **-i** Inode-Nummer anzeigen
- **-n** UID-, GID-Nummer anzeigen
- **-R** alle Unterverzeichnisse auch anzeigen
- **-C** mehrspaltige Ausgabe (sinnvoll bei Kurzform)
- **-F** Kurzformat
- **-p** / nach Verzeichnisnamen anzeigen (passt zu -C)

ls **-al** listet z. B. alle wichtige Informationen

ls **-CF** liefert eine übersichtliche Kurzliste

man (Manual)

Brauchen Sie zu einem Befehl eine Erläuterung, geben Sie 'man Befehl' ein, statt 'Befehl' natürlich den Namen des zu erklärenden Befehls. Diese Manual-Seiten sind für fast alle Kommandos vorhanden. Je nach Implementierung ist seitenweises Blättern mit der Leertaste möglich - oder ein Reaktionstest mit CTRL-S und CTRL-Q. Sehen Sie sich z. B. einmal die Infos über den Befehl 'ps' ('Prozess Status') an. Der ist auch sehr nützlich. Auch die hier vorgestellten Befehle können noch viel mehr - alles steht in dem Manual-Seiten.

Es ist auch unmöglich, in diesem Skript bei jedem Kommando alle Parameter aufzuzählen. Auch werden sicher nicht alle wichtigen Kommandos besprochen. Aus diesem Grund ist für den Anfänger wie für den Profi das man-Kommando wichtig und Voraussetzung für die Arbeit.

passwd (Password)

Interaktives Ändern des Paßworts. Zunächst muß das bisherige Paßwort eingegeben werden. Bei der Eingabe der Paßwörter werden diese auf dem Bildschirm nicht angezeigt. Damit Eingabefehler abgefangen werden können, ist das neue Passwort zweimal einzugeben. Programmablauf: \$

```
passwd
old passwd: Josef1
newpasswd: Maria2
retype new passwd: Maria2
```

who (Who)

Das Kommando gibt aus, wer alles im System aktiv ist. Es werden Login-Name, Terminal und Datum/Uhrzeit der Anmeldung angezeigt. Zum Beispiel:

```
$ who
hans  tty11    Juli 15 09:15
karl  tty12    Juli 15 09:55
kathi tty18     Juli 15 10:03
```

Wer nur wissen will, an welchem Terminal er sitzt, tippt: who am i Das Kommando kennt etliche Parameter, die wichtigsten sind: -p Anzeige der Prozesse, die gerade laufen -T Terminal-Status (mit "write" erreichbar?)

tty (Teletype)

gibt den Namen des aktuellen Terminals (eigentlich jenen des Device-Treibers) aus. Direkte Ausgabe auf dem Bildschirm können auf dieses Device geleitet werden. Sie gelangen dann auf jeden Fall aus

den Bildschirm, auch wenn die Eingabe umgeleitet wird (näheres später).

finger Info über User

Informationen über einen User erhält man mit 'finger user' - einfach mal ausprobieren. Unter der Idle-Time versteht man übrigens die Zeit, seit der der Mensch nichts gemacht hat.

write (Write Message)

Mit diesem Kommando kann man einem anderen Benutzer, der gerade am Rechner arbeitet, eine Nachricht auf den Bildschirm schicken (Nachrichten an nicht eingeloggte Benutzer später). Die Nachricht erscheint sofort auf dem Bildschirm des Partners. Als Parameter wird der Benutzername des Partners angegeben. Anschließend kann der Text zeilenweise eingegeben werden. Abgeschlossen wird die Eingaben mit den EOF-Zeichen CTRL-D. Zum Beispiel:

```
$ write karl
Lieber Karl,
dieGeburtstagsfeier für den Boss fängt schon in einer
halben Stunde an! Mach Dich rechtzeitig auf die Socken!
Gruss, Markus

<CTRL-D-Taste>
```

Bei Karl erscheint dann auf dem Bildschirm:

```
Message from markus tty16 [Tue Jul 23 14:05:00]

Lieber Karl, die Geburtstagsfeier für den Boss fängt schon in einer
halben Stunde an! Mach Dich rechtzeitig auf die Socken!
Gruss,Markus
```

mesg (Message) Solche Nachrichten sind zwar recht nützlich, können aber auch stören - wenn man z. B. gerade Texte schreibt. Denn die Nachricht zerstört ja das Bild auf dem Schirm. Mit dem mesg-Kommando kann man das Terminal gegen Nachrichten von außen sperren. *mesg n* sperrt das Terminal *mesg y* gibt das Terminal frei.

1.6 Das Prozesskonzept

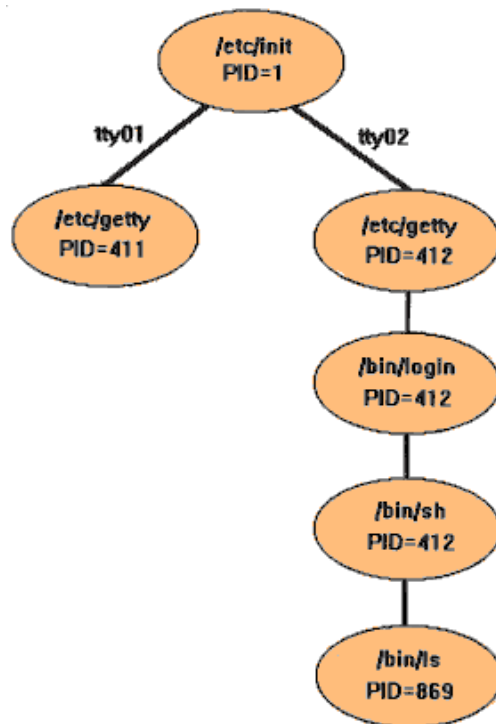
Jeder Prozess, der im Betriebssystemkern aktiviert wird, erhält eine eindeutige Kennzeichnung, die sogenannte Prozessnummer (PID). Über diese kann jeder Prozess eindeutig identifiziert werden. Da die Prozessnummer eine positive Integer-Zahl ist, gibt es eine maximale Prozessnummer. Wird diese erreicht, beginnt die Zählung wieder von vorne, wobei noch existierende Prozesse mit niedriger Nummer übersprungen werden.

Ein neuer Prozess kann nur von einem bereits laufenden Prozess erzeugt werden. Dadurch werden, ähnlich wie beim Dateibaum, die einzelnen Prozesse im Betriebssystemkern in einer baumartigen, hierarchischen Struktur verwaltet. Jeder Kind-Prozess ist genau einem Eltern-Prozess untergeordnet. Ein Eltern-Prozess kann jedoch beliebig viele Kind-Prozesse besitzen. Die Wurzel der Prozessstruktur wird durch den Systemstart geschaffen und als init-Prozess (PID 1) bezeichnet.

In der Regel wartet der Eltern-Prozess auf die Beendigung seiner Kind-Prozesse. Diese Art der Prozesssynchronisation wird als synchrone Ausführung bezeichnet, der Kind-Prozess wird als Vordergrundprozess ausgeführt. Bezogen auf einen Benutzer ist die Shell (Login-Shell) der Eltern-Prozess. Alle Kommandos, die der Benutzer startet, sind Kind-Prozesse. Während diese abgearbeitet werden ruht der Eltern-Prozess. Als asynchroner Prozess oder **Hintergrundprozess** werden solche Prozesse bezeichnet, bei denen der Eltern-Prozess nicht auf das Ende seines Kind-Prozesses wartet, sondern parallel (quasiparallel auf einer Ein-Prozessor-Maschine) asynchron

Betriebssystem UNIX/Linux

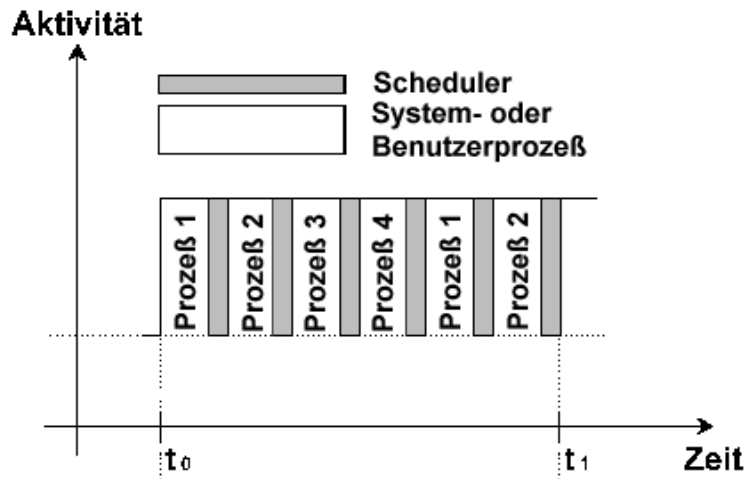
weiterläuft. Auf der Shell-Ebene kann jeder Prozess durch Anfügen von '&' (kaufm. UND) in der Kommandozeile als Hintergrundprozess gestartet werden.



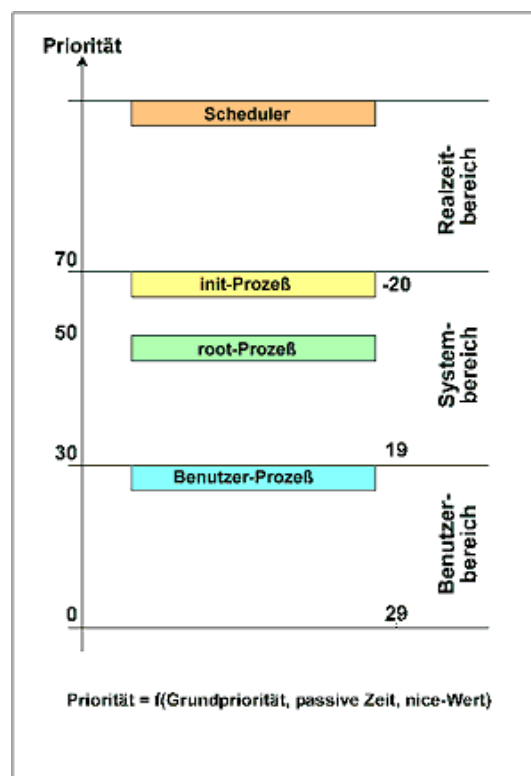
- Jeder Prozess kann beliebig viele neue Prozesse erzeugen
- Die Kindprozesse erben die gesamte Prozess-Umgebung (Home-Directory, Standard-Pfadnamen, Terminal-Typ, etc.)
- Eltern- und Kindprozesse laufen asynchron (Elternprozess kann auf Ende des/der Kindprozess(e) warten --> Synchronisation)
- Eltern- und Kindprozesse können miteinander kommunizieren
- Prozess-Generierung mit fork()
- Zwei Prozesse können miteinander kommunizieren und sich untereinander synchronisieren
- Ein Prozess kann einen Nachfolgeprozess starten: exec()
- 40 externe Prioritätsebenen, 2 interne Prioritätsebenen (System/User)

1.6.1 Der Scheduler

Unix ist ein Multitasking-Betriebssystem, d.h. mehrere Prozesse eines oder mehrerer Benutzer konkurrieren um die Vergabe der Rechenzeit des Prozessors. Wie viele andere Systeme auch, arbeitet Unix nach dem Zeitscheiben-Prinzip.



Über einen Scheduling-Algorithmus zur Berechnung der Priorität erhält jeder einzelne Prozess einen bestimmten Teil der Rechenzeit zugewiesen. D.h. der Prozess mit der zur Zeit höchsten Priorität erhält die CPU, wird nach einem Zeitintervall suspendiert und, falls noch nicht beendet, zu einem späteren Zeitpunkt wieder reaktiviert. Die aktuelle Priorität eines Prozesses setzt sich aus dem Produkt des CPU-Faktors und der Grundpriorität zusammen.



Die Prozessverwaltung und Prioritätssteuerung ist recht komplex. In Stichpunkten:

- Round Robin mit Multilevel Feedback (siehe Kap. 1)
- Neuberechnung aller Prioritäten einmal pro Sekunde - Hohe CPU-Auslastung: Priorität sinkt
- länger nicht gerechnet: Priorität steigt

--> einfach, effizient, gerecht

1.6.2 Swapping und Paging

- Wächst ein laufender Prozess dynamisch, werden andere Prozesse auf Platte ausgelagert; zunächst blockierte P., dann erst bereite P.
- Steht wieder Speicher zur Verfügung, werden erst die bereiten P. und dann die blockierten P. zurückgeholt.
- Je nach Hardware kann auch virtuelle Speicherverwaltung verwendet werden (Paging); bei anderen Maschinen gibt es nur Swapping.

1.6.3 Speicheraufteilung eines Prozesses im Arbeitsspeicher

Der Speicher zu einem Prozess wird in verschiedene Segmente unterteilt:

- Environment: Programmumgebung (Eigentümer, Home-Dir., usw.)
- Text: Programmcode (schreibgeschützt) - kann von mehreren Prozessen gleichzeitig verwendet werden
- Daten: Benutzerdaten des Prozesses. Unterteilt in einen initialisierten und einen nicht init. Datenbereich
- Stack: Benutzerstack und Verwaltungsinfo

1.6.4 Prozesskommunikation und -Synchronisation

- **Dateien** Ein oder mehrere Prozesse schreiben in Dateien, die von anderen Prozessen gelesen werden (wenig effizient).
- **Pipes (Datenkanäle)** Eine Pipe ist ein FIFO-organisierter Speicher, in den ein Prozess schreibt und aus dem ein anderer Prozess liest (FIFO = first in, first out). Pipes werden wie Dateien angesprochen, sind jedoch in der Regel als Pufferbereich im Hauptspeicher organisiert (auf Datei muß erst ausgewichen werden, wenn der Puffer überläuft).
- **Signale** sind Software-Interrupts, die asynchron auftreten. Hauptsächlich zur Kommunikation Benutzerprogramm <--> BS. Auslösung z. B. durch:
 - ◆ Aktionen des Benutzers (CTRL-C)
 - ◆ Programmfehler
 - ◆ durch andere Prozesse Im Prozess muß explizit festgelegt werden, welche Aktion auf ein bestimmtes Signal erfolgen soll:
 - ◆ Ignorieren
 - ◆ Bearbeiten durch eine Service-RoutineIst nichts entsprechendes definiert, wird der Prozess abgebrochen.
- **Named Pipes** Bei einfachen Pipes gelten zwei Einschränkungen:
 - ◆ Lebensdauer an die Lebensdauer der beteiligten Prozesse gebunden
 - ◆ Kommunikation nur für Prozesse mit gemeinsamen Elternprozess oder zwischen Eltern- und KindprozessNamed Pipes werden als Spezialdateien angelegt (siehe Dateitypen), die sich wie Gerätedateien verhalten und beliebig lange leben können. Named Pipes können nur einmal gelesen werden und natürlich arbeiten sie als FIFO.
- **Message Queues** dienen dem Austausch von strukturierten Nachrichten über eine Dienstleistung des BS.
- **Semaphore** sind Zustandsvariablen als elementarer Mechanismus zur Synchronisation von Prozessen.
- **Shared Memory** ist ein gemeinsamer Datenbereich im Hauptspeicher, der von zwei Prozessen genutzt werden kann (sehr viel effizienter als Dateien).

1.6.5 fork(), exec() und wait()

Diese Systemaufrufe haben mit der Generierung von Kindprozessen zu tun und erlauben die Synchronisation zwischen Eltern- und Kindprozessen. An dieser Stelle wird nur soweit darauf eingegangen, wie es zum Verständnis der folgenden Abschnitte nötig ist.

fork() erzeugt einen Kindprozess, der ein vollständiges Abbild des Elternprozesses ist und der beim gleichen Stand des Befehlszählers fortgesetzt wird. Eltern- und Kindprozess wird jedoch die Möglichkeit geboten, festzustellen, ob es sich um Eltern- oder Kindprozess handelt: Der Kindprozess bekommt als Rückgabewert 0, der Elternprozess die PID des Kindprozesses. Durch bedingte Verzweigung nach dem Schema (".. if Elternprozess then ... else ...") können beide Prozesse dann unterschiedlich weiterarbeiten.

- Etliche Systemprozesse schließen stdin, stdout und stderr, treten also in den Hintergrund. Solche Prozesse nennt man 'daemon' (Daemonprozesse).
- Terminiert der Elternprozess vor dem Kindprozess, wird dieser zum 'Waisenkind'. Normalerweise wird er dann vom Init-Prozess 'adoptiert'.
- Hat der Kindprozess dann auch noch den Kontakt zum Terminal (Standardausgabe und -eingabe) wird er zum 'Zombie'.

Einzelsschritte beim Aufruf von fork():

1. Prozesstabelle überprüfen (Platz frei?)
2. Speicher für Kindp. allozieren
3. Elternprozess-Speicher --> Kindprozess-Speicher kopieren
4. Prozesstabelleneintrag es Elternprozesses aktualisieren
5. PID für Kindp. wählen, Kindp. in Prozesstabelle eintragen
6. Kernel und Dateisystem über Kindprozess informieren
7. Fertigmeldung an Eltern- und Kindprozess senden

wait() ermöglicht dem Elternprozess das Warten auf die Beendigung des/der Kindprozess(e). Der Elternprozess wird verdrängt und erst durch das Ende eines Kindprozesses wieder "aufgeweckt". Zur Unterscheidung mehrerer Kindprozesse liefert die Funktion wait() die PID des "gestorbenen" Kindprozesses zurück.

Gibt es keinen Kindprozess, ist das Ergebnis -1. Beheben des Waisenkind/Zombie-Problems:

1. Elternprozess ruft wait() zu spät auf:
Beim Wait-Aufruf wird zuerst die Prozesstabelle nach terminierten Kindprozessen durchsucht und diese dann gelöscht.
2. Elternprozess ist terminiert:
Beim Terminieren des Elternprozesses werden dessen Kindprozesse zu Kindprozessen des Systemprozesses Init.

Bei **exec()** wird der ursprüngliche Prozess durch einen neuen Prozess ersetzt (eine Rückkehr zum aufrufenden Prozess ist daher logischerweise nicht möglich). exec() ist der komplizierteste Aufruf, da der komplette Prozessadreibraum ersetzt werden muß. Dieser Aufruf ist auch als Kommando verfügbar (exec [Programmname]). Der Ablauf im Schema:

1. Zugriffsrechte prüfen (Datei ausführbar?)
2. Größe der Speichersegmente feststellen
3. Aufrufparameter und Umgebung des Aufrufers festhalten
4. Speicher des Aufrufers freigeben, neuen Speicher allozieren
5. Neues Programm in den Speicher laden

6. UID-, GID-Bits bearbeiten
7. Prozesstabelle aktualisieren
8. Bereitmeldung an den Kernel senden

Schließlich gibt es noch eine Systemfunktion, welche die zeitweise Blockierung eines Prozesses erzwingt: Mit **sleep()** kann ein Prozess für eine definierte Zeit "eingeschläfert" werden. Nach Ablauf der vorgegebenen Zeit, wird der Prozess wieder auf "bereit" gesetzt und kann weiterlaufen. Auch **sleep()** ist als Kommando verfügbar (**sleep [Zeit in Sekunden]**).

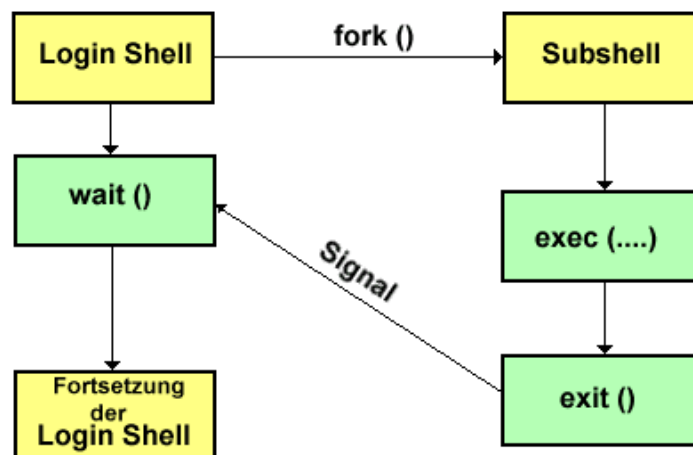
1.6.6 Die Programme **init** und **getty**

Nach dem Start des Rechners und Laden von Unix starten als erstes:

- der Scheduler mit der Prozessnummer 0
- und das Programm **init** mit der Prozessnummer 1

Alle Prozesse, die auf dem Rechner laufen sind Kindprozesse von **init** (wobei "Kind" hier auch für alle weiteren Nachkömmlinge steht). Im Single-User-Modus werden von **init** nur noch einige Prozesse gestartet, im Multi-User-Modus sind wesentlich mehr Prozesse zu aktivieren (z. B. der Drucker-Spooler, die Abrechnung, etc.). Alle zu startenden Prozesse sind in der Datei **/etc/inittab** aufgeführt. **init** holt sich die Informationen über zu startende Prozesse aus der Datei **/etc/inittab**. Für uns ist eigentlich nur das Programm **getty** interessant, den dieses Programm sorgt für den Kontakt mit den Terminals. Was geschieht nun weiter?

- **getty** wird von **init** für jedes angeschlossene Terminal gestartet.
- **getty** wartet, bis das Terminal eingeschaltet wird, liest den Usernamen ein und ruft dann mittels **exec** das **login**-Programm auf, das seinerseits das Passwort einliest.
- Wenn sich nun ein Benutzer anmeldet, wird das in der Benutzerliste spezifizierte Programm per **exec** aktiviert (in der Regel der Kommandointerpreter, die Shell).



- Die Shell (oder das anstelle der Shell aufgerufene Programm) arbeitet, bis es terminiert (bei der Shell durch Eingabe von EOF = CTRL-D). Da auch die Shell immer noch Kindprozess von **init** ist, erhält nun **init** ein Signal, daß der Kindprozess terminiert ist (siehe **fork**).
- Nun "wacht **init** auf", stellt fest, welcher seiner Kindprozesse terminiert ist und durchsucht die Datei **/etc/inittab** - uns interessiert wieder nur **getty**.
- In **inittab** steht beim Prozess **getty** die Steuerinformation **respawn** (**spawn** = "laichen"). Das heißt für **init**, daß dieser Prozess nach seiner Terminierung neu gestartet werden soll - und genau das geschieht.

1.6.7 Vorbemerkung zur Shell

Der Kommandointerpreter heißt bei Unix 'Shell' (er umgibt das System wie eine Schale die Muschel). Die Shell ist sehr leistungsfähig und man kann über eine eigene Programmiersprache Kommandoprozeduren, sogenannte Shell-scripts, erstellen, die sich wie Programme aufrufen lassen. Darauf wird später noch genau eingegangen. Zunächst geht es um die grundlegenden Funktionen. Dabei ist immer von der StandardShell, der Bourne-Shell die Rede. Analoges gilt auch für andere Shells (bash = bourne again Shell, C-Shell, Korn-Shell); es werden dort aber teilweise andere Startdateien verwendet.

- Nach dem Start sucht die Shell nach der Datei **/etc/profile** und führt die darin enthaltenen Kommandos aus. **/etc/profile** ist also für alle Benutzer gleich.
- Dann erfolgt der gleiche Vorgang mit der benutzereigenen Datei **.profile** im Home-Directory.
- Nun macht die Shell nichts anderes, als Eingaben vom Benutzer anzunehmen und die entsprechenden Programme aufzurufen.
- Zeigt der Benutzer durch Eingabe von EOF = CTRL-D an, daß keine Kommandos mehr folgen, terminiert die Shell.

1.6.8 Einflußnahme des Benutzers auf Prozesse

- Der Benutzer kann Prozesse erzeugen (siehe vorher)
- Der Benutzer kann eigene Prozesse jederzeit terminieren
- Prozesse können "im Hintergrund" laufen (mit hoher oder niedriger Priorität)
- Prozesse können als Batch (d.h. ohne Bindung an ein Terminal) laufen
- Prozesse können regelmäßig zu bestimmten Zeiten gestartet werden (z. B. jede Nacht, jede Woche, etc.)
- Prozesse können zu einem bestimmten Zeitpunkt gestartet werden

Alles wird später im Detail behandelt!

1.6.9 Das /proc-Verzeichnis

Sie können eine Menge interner Kernel-Informationen sowie Daten zu allen gerade laufenden Prozessen aus den Dateien des `/proc`-Verzeichnis entnehmen. Unter anderem ist dort jedem Prozess ein eigenes Unterverzeichnis zugeordnet. Innerhalb des Prozessverzeichnisses befinden sich dann einige Dateien mit diversen Verwaltungsdaten (z. B. die zum Start verwendete Kommandozeile). Diese Daten werden von diversen Kommandos zur Prozessverwaltung (z. B. `top`, `ps` etc.) ausgewertet.

Die Dateien im `/proc`-Verzeichnis sind keine echten Dateien und beanspruchen daher auch keinen Platz auf der Festplatte. Das gilt auch für die scheinbar sehr große Datei `/proc/kcore`, die den Arbeitsspeicher abbildet. Vielmehr handelt es sich bei diesem Verzeichnis um ein virtuelles Dateisystem, das der Linux-Kernel zur Verfügung stellt.

Die meisten der `/proc`-Dateien liegen im Textformat vor. Um die Dateien zu lesen, müssen Sie unter Umständen `cat` statt `less` verwenden. (Manche `less`-Versionen kommen mit den virtuellen Dateien des `/proc`-Verzeichnisses nicht zurecht.)

Ein paar interessante `/proc`-Dateien `/proc/bus/pccard/*` - `/proc/n/*` Informationen zum Prozess mit der PID=`n`

`/proc/bus/usb/*` USB-Informationen

`/proc/bus/pci/*` PCI-Informationen

`/proc/cmdline` LILO/GRUB-Boot-Parameter
`/proc/cpuinfo` CPU-Informationen
`/proc/devices` Nummern von aktiven Devices
`/proc/ide/*` IDE-Laufwerke und -Controller
`/proc/interrupts` Nutzung der Interrupts
`/proc/iomem` Nutzung des IO-Speichers
`/proc/ioports` Nutzung der IO-Ports
`/proc/modules` Aktive Module
`/proc/mounts` Aktive Dateisysteme
`/proc/net/*` Netzwerkzustand und -nutzung
`/proc/partitions` Partitionen der Festplatten
`/proc/scsi/*` SCSI-Laufwerke und -Controller
`/proc/sys/*` System- und Kernel-Informationen
`/proc/uptime` Zeit in Sekunden seit dem Rechnerstart
`/proc/version` Kernel-Version

über das `/proc/sys`-Verzeichnis können Sie sogar Kernel-Parameter im laufenden Betrieb verändern. Eine sehr ausführliche Beschreibung aller `/proc`-Dateien finden Sie im Buch zur Linux-Kernel-Architektur.

1.7 Grundlagen der Benutzerverwaltung

Die Benutzerverwaltung ist bei Unix ebenso offen aufgebaut, wie der Rest des Systems. Es existieren zwei Text-Dateien, die sämtliche Benutzerinformation aufnehmen:

- **`/etc/passwd`** nimmt Benutzerdaten und Paßwort auf
so war es jedenfalls bei den früheren Systemen (siehe unten). Deshalb gibt es seit einiger Zeit:
- **`/etc/shadow`** nimmt das Passwort auf und ist nur von root lesbar
- **`/etc/group`** nimmt die Gruppenzugehörigkeit auf

Beim Login (und beim Wechsel des Benutzerkennzeichens während einer Terminalsitzung) wird auf diese Dateien zugegriffen - sie sind übrigens für alle Benutzer lesbar. Schreiben darf jedoch nur der Superuser und das Programm `passwd`. Da `passwd` von jedem Benutzer aufgerufen werden kann, ergibt sich hier eigentlich ein Widerspruch. Gelöst wird das Problem durch das UID-Bit von `passwd`: während das Programm läuft, nimmt der Prozess die Identität seines Eigentümers an - und der darf schreiben. Anmerkung: Ab der Unix-Version System V, Version 3 steht das Paßwort nicht mehr in `/etc/passwd`, sondern in einer eigenen Datei `/etc/shadow`. Da `/etc/passwd` für alle lesbar sein muß (z. B. für die Anzeige des Benutzernamens im `ls`-Kommando), kann man mit entsprechenden Programmen versuchen, die Paßwörter zu 'knacken'. Durch die Verlagerung der Paßwortinfo in `/etc/shadow`, die nur von Superuser-Prozessen gelesen werden kann, wird diese Sicherheitslücke geschlossen.

1.7.1 Was geschieht beim Login?

- Das Programm `login` erhält vom `getty`-Prozess den eingegeben Benutzernamen (Login-Namen) und sucht nun in der Datei `/etc/passwd` nach einer passenden Zeile. Wird nichts gefunden --> Benutzer nach Paßwortabfrage abweisen.
- Ist der Benutzer gefunden, wird überprüft, ob ein Paßwort eingetragen ist. Falls vorhanden, wird das Paßwort abgefragt und bei fehlerhafter Eingabe der Benutzer abgewiesen.
- Danach wird das im Benutzereintrag spezifizierte Programm gestartet. In der Regel ist dies die Shell.

Wichtig: Das Paßwort ist in der Datei `/etc/passwd` verschlüsselt gespeichert. Die Verschlüsselung erfolgt beim Ändern des Passworts mit dem Programm `passwd` oder bei der Eingabe im Login-Programm. Verglichen werden immer nur die verschlüsselten Passwörter. Auch der Superuser kann das Paßwort nicht entschlüsseln. Wenn Sie Ihr Paßwort vergessen haben, kann er nur ihr altes Paßwort löschen, damit Sie dann ein neues eintragen können. Um das "knacken" von Login-Name und Paßwort zu erschweren, fragt das Login-Programm auch dann das Paßwort ab, wenn schon der Benutzername falsch war. Außerdem wird beim Eingeben des Passworts nichts auf dem Bildschirm angezeigt.

1.7.2 Aufbau der Datei `/etc/passwd`

Die Datei ist eine normale Textdatei und sie enthält für jeden Benutzer genau eine Zeile mit 7 Feldern, die jeweils durch einen Doppelpunkt voneinander getrennt sind. Die Zeilenlänge darf 511 Zeichen nicht überschreiten, wobei die Länge der einzelnen Felder variabel ist. Der Aufbau der Zeile ist:

```
Login-Name:Paßwort:UID:GID:Kommentar:Home-Directory:Programm
```

- Der Login-Name (3 .. 6 Zeichen) ist der Name, unter dem der Benutzer dem BS bekannt ist. Er kann Großbuchstaben enthalten. Besteht er vollständig aus Großbuchstaben, schaltet Unix auf Großschreibung um (historisch bedingt).
- Das Paßwort ist verschlüsselt und es ist stets 13 Zeichen lang. Ist das Feld leer muß kein Paßwort eingegeben werden (nur RETURN-Taste). Der Superuser kann zusätzlich ein Komma und zwei weitere Zeichen eintragen, die die Gültigkeitsdauer festlegen (später mehr). Anmerkung: Ab Version 5.3 steht hier nur ein "x" und das Paßwort in `/etc/shadow`. Das Paßwort darf nicht mit dem Login-Namen übereinstimmen, muß mindestens 6-8 Zeichen lang sein und muß mind. zwei Buchstaben und eine Ziffer enthalten.
- UID = User Ident: In diesem Feld wird die Benutzernummer festgehalten (Wertebereich 0 bis 50000). Jeder Benutzer muß eine individuelle UID besitzen. Die 0 ist für den Superuser reserviert, die UIDs 1 - 99 für interne Zwecke. Reguläre Benutzer beginnen ab UID 100.
- GID = Group ID: In diesem Feld wird festgehalten, zu welcher Gruppe der Benutzer gehört (Wertebereich 0 bis 50000). I. A. wird die GID 100 als Sammelgruppe verwendet (für alle Benutzer die keiner anderen Gruppe zugeordnet werden).
- Kommentar: In diesem Feld (max. Länge 30 Stellen) werden allgemeine Hinweise zum Benutzer eingetragen (normalerweise der vollständige Name des Benutzers, Abteilung, etc). Dieses Feld wird beispielsweise von Mail- und News-Programmen abgefragt, um automatisch den Absender einzutragen.
- Home Directory: Beim Eintragen eines Benutzers wird ihm auch ein Arbeitsverzeichnis, das Home Directory, eingerichtet. In dieses Verzeichnis wird beim Login verzweigt. Der Pfad muß vollständig eingegeben werden.
- Programm: Hier wird das Programm angegeben (vollständiger Pfadname, max. 256 Zeichen), das nach Beendigung von login gestartet werden soll. Bei normalen Benutzern ist dies der Kommandointerpreter, die Shell. Sie wird auch gestartet, wenn dieser Eintrag fehlt. Je nach Anwendung des Rechners können hier aber auch andere Programme eingetragen werden, z. B. für die Buchhaltung gleich das Buchhaltungsprogramm. Es gibt auch die Auswahl zwischen den verschiedenen Shells (C-Shell, Bourne-Shell, Korn-Shell) oder der restricted Shell, die einen eingeschränkten Befehlsumfang besitzt.

Beispiel für einen Eintrag in `/etc/passwd`:

```
karl:,:235:100:Karl Müller:/usr/karl:/bin/sh
```

Gültigkeitsdauer des Paßworts:

Die Gültigkeitsdauer des Paßworts wird im Paßwortfeld durch Anfügen von Komma und zwei Zeichen vom Superuser eingeschränkt. Bei Verwendung der Shadow-Datei stehen diese Angaben dort.

- Das erste Zeichen legt die minimale Gültigkeitsdauer fest, d. h. in dieser Zeit kann das Paßwort nicht geändert werden.
- Das zweite Zeichen legt die maximale Gültigkeitsdauer fest, d. h. nach Ablauf dieser Zeit muß das Paßwort geändert werden.

Die Zeitdauer wird in Wochen gezählt; der Punkt "." bedeutet 0 Wochen, der Schrägstrich "/" 1 Woche, dann folgen Ziffern und Buchstaben:

- 0..9: 2 bis 11 Wochen
- A..Z: 12 bis 37 Wochen
- a..z: 38 bis 63 Wochen

Sonderfälle:

- .. (zwei Punkte) Der Benutzer muß beim nächsten Login sein Paßwort ändern - danach lebt es ewig (für neu eingerichtete Benutzer).
- ./(Punkt, Schrägstrich) Der Benutzer kann sein Paßwort nicht mehr ändern. Sinnvoll bei einer "Gast"-Kennung, denn der Gast soll das Paßwort nicht ändern können.

1.7.3 Aufbau der Datei /etc/group

Auch diese Datei kann von den Benutzern nur gelesen werden, das Schreiben ist dem Superuser vorbehalten. Sie legt die Gruppenzugehörigkeit der Benutzer fest. Die Datei besteht aus Textzeilen mit 4 Feldern, die durch Doppelpunkte getrennt sind:

Gruppenname:Paßwort:GID:Benutzernamen

- Gruppenname: Maximal 8 Buchstaben/ziffern, die den Namen der Gruppe festlegen.
- Paßwort: Zwar ursprünglich konzipiert, bleibt dieses Feld leer, denn es gibt keine Möglichkeit, das Paßwort verschlüsselt einzutragen.
- GID = Group ID: In diesem Feld wird festgehalten, zu welcher Gruppe der Benutzer gehört (Wertebereich 0 bis 50000). Die Nummern 0 bis 99 sind für interne Zwecke reserviert, GID 100 wird als Sammelgruppe verwendet.
- Benutzernamen: Hier werden alle Login-Namen der zur Gruppe gehörenden Benutzer, getrennt durch Komma, eingetragen. Einträge für die Standardgruppe, die in der Datei /etc/passwd festgelegt ist, sind nicht nötig.

Beispiel für einen Eintrag in /etc/group:

```
dozenten::200:kristl,plate,rother,ries,thomas
```

1.7.4 Aufbau der Datei /etc/shadow

Diese Datei kann von den Benutzern nicht gelesen werden, das Lesen und Schreiben ist dem Superuser vorbehalten. Sie enthält das Benutzerpaßwort und Angaben über die Gültigkeitsdauer von Paßwort und Benutzeraccount. Die Datei besteht aus Textzeilen mit 9 Feldern, die durch Doppelpunkte getrennt sind:

Name:Paßwort:letzte

Änderung:Min:Max:Vorwarnzeit:Inaktiv:Verfall:Kennzeichen

- Name: Derselbe Benutzername, wie er in /etc/passwd steht
- Paßwort: Normalerweise 13-stelliges, verschlüsseltes Paßwort, in dem die ersten beiden Stellen die Verschlüsselungsmethode (eine aus 4096) kennzeichnen. Anstelle des Paßworts kann hier auch ein Sperrvermerk stehen.
- letzte Änderung: Der Tag der letzten Änderung des Paßworts. Gezählt wird in Tagen ab dem 1.1.1970 (offizieller Entstehungstag von Unix).
- Min: Minimale Gültigkeitsdauer des Paßworts in Tagen (vorher ist kein Ändern möglich).
- Max: Maximale Gültigkeitsdauer des Paßworts in Tagen. Der Benutzer muß vor Ablauf dieser Frist sein Paßwort ändern.
- Vorwarnzeit: Anzahl der Tage, wie lange der Benutzer vor Verfall seines Paßworts auf die notwendige Änderung hingewiesen wird.
- Inaktiv: Anzahl der Tage, die es dem Benutzer gestattet ist, seinen Account unbenutzt zu lassen.
- Verfall: Absolutes Datum, an dem die Verwendung des Accounts gesperrt wird.
- Kennzeichen: Reserviert für künftige Verwendung. Derzeit auf 0 gesetzt. Damit nun nicht alle Diese Daten beim Anlegen eines Benutzers von Hand eingegeben werden müssen, wird - so vorhanden- die Datei /etc/default/passwd herangezogen, in der Standardwerte gespeichert sind.

1.7.5 Voreinstellungsverzeichnis /etc/default

In diesem Verzeichnis werden unter Unix V.4 die Standardwerte für verschiedene Programme abgelegt. Das Verzeichnis enthält Dateien, deren Namen sich auf das zugehörige Programm oder die zugehörige Funktion des Betriebssystems beziehen, z. B. "passwd", "tar", "boot", "init", etc. Die Einträge in den Daten bestehen in der Regel aus Zeilen mit jeweils einer Zuweisung der Form 'Schlüsselwort'='Wert'. Dazu ein Beispiel:

```
/etc/default/passwd:
```

Standardwerte für das Paßwort und seine Gültigkeitsdauer. Einige wichtige Werte sind:

- MAXWEEKS='maximale Zeit, bis das PW geändert werden muß'
- MINWEEKS='minimale Zeit, in der das PW nicht geändert wird'
- WARNWEEKS='Vorwarnzeit'
- PASSLENGTH='minimale Passwortlänge'

1.7.6 Anlegen eines Benutzers

Erstaunlicherweise gab es ursprünglich kein Programm zum Anlegen von Benutzern - bei den meisten Systemen hat sich jedoch der Systemverwalter (= Superuser) ein Shell-script dafür angelegt. Bei vielen Systemen wird heute ein Standard-Tool mitgeliefert ('useradd' oder 'adduser'). Anmerkung: In diesem Abschnitt werden einige Kommandos erwähnt, die erst zu einem späteren Zeitpunkt näher erläutert werden. Um einen Benutzer neu einzutragen sind folgende Schritte notwendig:

- Benutzer in der Datei /etc/passwd eintragen. Dabei ist darauf zu achten, daß der Login-Name und die UID eindeutig, d. h. nur einmal vorhanden, sind - sonst kommt es zu Schwierigkeiten bei der Zuordnung des Homedirectory und anderer Dateien.
- In der Datei /etc/group wird der neue Login-Name im entsprechenden Gruppeneintrag ergänzt.

Betriebssystem UNIX/Linux

- Nun wird das Home-Directory des neuen Benutzers eingerichtet und - falls erwünscht - eine Standardversion der Datei `.profile` (bei C-Shell `.login`) dorthin kopiert.
- Home-Directory und `.profile` sind immer noch "Eigentum" des Superusers, mit dem `chown`-Befehl (Change Owner) werden beide an den neuen Benutzer übergeben. An sich reicht das aus. Jetzt können noch von der Arbeitsumgebung abhängende Maßnahmen vorgenommen werden.

In der Regel wird das Anlegen und Löschen eines Users durch passende Systemprogramme oder -Skripten erledigt.

Einen Benutzer sperren

Soll ein Benutzer zeitweise (z. B. bei längerem Urlaub) oder auf Dauer gesperrt werden, seine Dateien aber noch erhalten bleiben, trägt der Superuser im Paßwortfeld einen Zeichenfolge ein, die als Ergebnis der Verschlüsselung nicht vorkommen kann, z. B. `'*LCK*`. Überprüfung von `/etc/passwd` und `/etc/group`.

1.7.7 Löschen eines Benutzers

Die hierfür notwendigen Schritte sind wesentlich gefährlicher, da hier u. U. wichtige Informationen gelöscht werden. In der Regel wird der Benutzer zunächst deaktiviert (er kann sich also nicht mehr einloggen) und erst nach einer gewissen Zeit vollständig gelöscht. Dazu sind dann folgende Schritte nötig:

- Löschen des Home-Directories mit allen darunter liegenden Dateien und Verzeichnissen (`rm`-Kommando)
- Löschen des mail-Directory und aller Spool-Dateien
- Löschen der zum Benutzer gehörenden Zeile aus `/etc/passwd`
- Löschen des Login-Namens aus `/etc/group` Anmerkung:

Um alle Dateien (nicht Directories) eines Benutzers zu löschen, kann das Kommando `find` (siehe später) verwendet werden (für UID wird die User-ID des Benutzers eingesetzt):

```
find / -user UID -type f -exec rm -f {} ";"
```

Probleme können durch Dateien des zu löschenden Benutzers verursacht werden, auf die andere Benutzer ein Link gesetzt haben. Man kann diese Dateien z. B. den betroffenen Benutzern zuordnen.

1.8 Starten und Stoppen des Systems

Wie schon anfangs gesagt, sind Start (Bootstrap) und Stop (Shutdown) des Systems bei Unix wesentlich komplexer, als bei einfachen Betriebssystemen. Es gibt - abhängig von den jeweiligen Aufgaben - mehrere "Runlevels" des Systems; hier nur eine Auswahl:

0	Power-Down: Ausschalten des Rechners
1	Administrativer Level: z. B. Einrichten neu eingebauter Platten oder andere Hardware-Initialisierungen. Oft auch "s" oder "S" (Singleuser = Einzelbenutzer-Modus).
2	Multiuser-Modus ohne Netzwerkanbindung

3	Multiuser-Modus mit Netzwerkanbindung (Normal-Level)
4	Frei für benutzerdefinierten Modus
5	Firmware-Modus: z. B. Diagnose und Wartung; oft nur mit spezieller Floppy zu starten
6	Shutdown und Reboot: Wechsel zu Level 0 und dann sofortiges Hochlaufen

Die Zuordnung der Level kann auch von der oben angeführten abweichen. Der Wechsel des Levels wird durch spezielle Kommandos erreicht, z. B. shutdown, telinit, (re)boot oder halt. So ist beispielsweise auch ein Bootstrap von einem bestimmten Datenträger (Floppy, CD-ROM, ...) möglich.

Beispiel: Reboot des Systems nach 2 Minuten (Solaris): **shutdown -g120 -i6 -y** Shutdown sendet eine Nachricht an alle eingeloggtten Benutzer, bevor der eigentliche Prozess beginnt.

Nach dem Einschalten des Rechners und dem Durchlaufen des Power On Self Test wird normalerweise der Bootsektor der ersten Festplatte ausgeführt. Unter Linux wird hierfür meistens der Bootmanager GRUB verwendet. Dieser Bootmanager kann auch unterschiedliche Betriebssysteme von beliebigen Festplatten oder anderen Speichermedien (CD-ROM, DVD etc.) starten.

Bei Linux ist der Ablauf noch etwas komplexer. Der Bootmanager lädt den zu bootenden Kernel. Dieser braucht zum Fortsetzen des Bootvorgangs ein Root-Dateisystem. Deshalb legt er eine "initiale Ramdisk" an, indem die Datei `initrd` in den Speicher geladen und als Dateisystem montiert wird. (Ja, Linux kann eine Datei als "Hülle" eines Dateisystems verwenden und wie eine Platte bzw. Plattenpartition verwenden.) Damit läßt ein minimales Linux, das nun alle folgenden Aufgaben ausführen kann.

Egal, ob der Reboot-Vorgang durch einen Shutdown oder durch Einschalten des Rechners ausgelöst wurde, sind die Systemaktivitäten im Prinzip immer gleich:

- Hardware-Erkennung (CPU, Grafikinterface, Schnittstellen, Platten usw.)
- Testen der Dateisysteme (Platten)
- Montieren (mount) der Platten (Info aus `/etc/fstab` bzw. `/etc/vfstab`).
- Säuberungsaktionen (z.B. Löschen von temporären Dateien, Aufheben von eventuell beim Shutdown gesetzten Sperren, etc.
- Starten der Systemprozesse (Scheduler, init, getty, cron, Printer-Daemon, Mail, Accounting, etc.
- Start der Netzwerk-Programme, montieren von Remote-Platten (NFS)
- User-Login freigeben

Diese doch relativ komplexen Aktionen werden wieder über spezielle Shell-Scripts gesteuert. Bei BSD-Unix war der Aufbau dieser Scripts relativ einfach. Die Datei `/etc/rc` enthält alle beim Systemstart auszuführenden Kommandos. Innerhalb von `rc` werden eventuell weitere `rc`-Dateien aufgerufen, z. B.:

<code>/etc/rc.local</code>	Start lokaler Software
<code>/etc/rc.net</code>	Start der Netzwerksoftware
<code>/etc/rc.single</code>	Start im Single-User-Modus

Später wurde das System dahingehend erweitert, daß es für jeden Runlevel eine eigene `rc`-Datei gab (`rc0`, `rc1`, `rc2`, usw.). Ab System V ist das System der `rc`-Dateien vereinheitlicht worden. Für jeden Runlevel existiert ein Verzeichnis unter `/etc`, wobei der Name der Verzeichnisse einheitlich `/etc/rcx.d` ist (`x` steht für den Runlevel, es gibt also `rc0.d`, `rcs.d`, `rc2.d`, usw.). Im Verzeichnis

Betriebssystem UNIX/Linux

`/etc/init.d` (manchmal auch `/sbin/init.d`) sind alle Programme (oder Shell-Scripts) gespeichert, die beim System-Boot aufgerufen werden könnten. In den Verzeichnissen `rcx.d` sind nun nur noch Links auf diese Programme enthalten. Alle Links folgen ebenfalls einer festen Namenskonvention:

- der erste Buchstabe ist entweder ein "S" oder ein "K"
- danach folgt eine zweistellige Zahl
- zum Schluß folgt der Name des Programms in `/etc/init.d`

Die so entstandenen rc-Scripts werden in lexikalischer Reihenfolge aufgerufen und zwar zuerst die K-Dateien, dann die S-Dateien. Die Zahl im Namen legt also die Reihenfolge innerhalb der K- oder S-Gruppe fest. Die K-Dateien dienen zum Löschen (Kill) von Prozessen, die S-Dateien zum Starten von Prozessen. Angenommen, es existieren folgende Dateien in `/etc/rc.d`:

- K30tcp
- S20syssetup
- S85lp
- K40nfs
- S40nfs
- S75cron.

Dann ist die Aufruf-Reihenfolge:

```
K30tcp K40nfs S20syssetup S30tcp S40nfs S75cron S85lp
```

Dabei sind K- und S-Dateien mit ansonsten gleichem Namen lediglich Hinweise darauf, dasselbe Programm aufzurufen. So wird z. B. bei den Dateien K30tcp und S30tcp das Programm oder Script `/etc/init.d/tcp` einmal mit dem Parameter "stop" und einmal mit dem Parameter "start" aufgerufen. Man kann also durch Anlegen von Links das Hochfahren des Systems sehr gezielt steuern. Das entsprechende rc-Script wird dann auch sehr einfach, es läßt sich folgendermaßen skizzieren:

```
#!/bin/sh
# Wenn Directory /etc/rc2.d vorhanden ist
if [ -d /etc/rc2.d ] ; then
# K-Files bearbeiten
for f in /etc/rc2.d/K* ; do
    if [ -s $f ] ; then
        /bin/sh $f stop
    fi
done
# S-Files bearbeiten
for f in /etc/rc2.d/S* ; do
    if [ -s $f ] ; then
        /bin/sh $f start
    fi
done
fi
```

Ein von der rc-Datei aufgerufenens Script in `/etc/init.d` könnte dann z. B. so aussehen:

```
#!/bin/sh
case $1 in
'start')
# aufgerufen als "Kxxcron"
# Lockfilelöschen
rm -f /var/spool/cron/FIFO
if [ -x /etc/cron ] ; then
    /etc/cron
```

```
fi
;;
'stop')
# aufgerufen als "Sxxcron"
pid=`/bin/ps -e | grep 'cron$' | sed -e 's/^ *//' -e 's/ .*//'\`
if [ "$pid" != "" ] ; then
    /bin/kill -9 $pid
fi
;;
esac
```

Natürlich muss es einen Prozess geben, der die Ausführung der oben beschriebenen Scriptdateien vornimmt. Dazu wird vom Kernel die Root-Partition der Platte im Read-Only-Modus montiert und dort nach dem Programm `/sbin/init` gesucht und ausgeführt (andere Aufgaben von `init` wurden ja schon besprochen). Das `Init`-Programm liest seine Konfiguration aus der Textdatei `/etc/inittab`. In dieser Datei werden der Standard-Runlevel und das Script festgelegt, das beim Starten des Rechners ausgeführt wird:

```
id:3:initdefault:
si::sysinit:/etc/init.d/rcS
```

Dieses Script sorgt für Basisaktionen, z. B. das Überprüfen der Dateisysteme, das Montieren der Platten, das Stellen der Systemuhr usw. Anschließend folgen die Script-Aufrufe für die einzelnen Runlevel, der jeweils in der zweiten Spalte angegeben ist. Dabei können auch mehrere Runlevel bei einem Startscript angegeben werden.

Die `inittab` enthält auch die Startbefehle für die sechs virtuellen Konsolen:

```
1:2345:respawn:/sbin/getty 38400 tty1
2:23:respawn:/sbin/getty 38400 tty2
...
```

Sie erinnern sich, "respawn" heißt für `init`, daß der `getty`-Prozess nach seiner Terminierung neu gestartet werden soll.

In der Datei `/etc/inittab` können prinzipiell beliebige Programme gestartet oder auch das Verhalten beim Drücken der als "Affengriff" bekannten Tastenkombination `[Strg]+[Alt]+[Entf]` geregelt werden:

```
ca:12345:ctrlaltdel:/sbin/shutdown -h now
```

Es geht sogar noch weiter. Bei Embedded Systems (etwa einem DSL-Router) kann anstelle von `init` gleich die (einzige) Anwendung gestartet werden. Das Booten erfolgt in diesem Fall aus dem ROM und das ganze System läuft in der Ramdisk ab. Es sind auch alle Zwischenstufen zwischen diesem Minimalsystem und dem Standard-Desktop-System möglich, etwa ein System, das zwar mit `init` startet, aber dann nicht als Multiusersystem weiterbootet.

Mit den Kommandos `init <Runlevel>` oder `telinit <Runlevel>` kann der Runlevel gewechselt werden. Dazu fährt `init` den aktuellen Runlevel herunter (Ausführen aller K-Scripten) und dann den neuen Runlevel hoch (Ausführen aller S-Scripten). Der aktuelle Runlevel kann mit dem Befehl `/sbin/runlevel` abgefragt werden.

Aufgrund seiner Konzeption startet der SysV-Init die Prozesse immer in einer vorgegebenen Reihenfolge und er startet einen Prozess fast immer erst, wenn der vorherige Prozess fertig abgeschlossen wurde. Dieses schrittweise Vorgehen macht den Bootvorgang zwar zuverlässig, aber auch relativ langsam. Inzwischen werden beim Booten eines Unix- oder Linux-Systems so viele Dienste gestartet, dass der Bootvorgang relativ lange dauert. Man kann natürlich die nicht

notwendigen Dinge lahmlegen, indem man einfach das entsprechende Script in `/etc/init.d` umbenennt (z. B. durch Anhängen von `".inaktiv"`).

Schon seit einiger Zeit gibt es Versuche, dieses Konzept zu überarbeiten oder wenigstens zu modifizieren. So werden zum Beispiel bei einigen Linux-Distributionen alle Prozesse in einem Runlevel gleichzeitig gestartet. Alternativen zu `init` waren u. a. `InitNG`, `eINIT` und `Launchd` in Mac OS X. All diese Konzepte haben eines gemeinsam: sie sind zielorientiert. Es wird also vorher festgelegt, welche Dienste am Ende des Startvorganges laufen sollen. Die Abhängigkeiten werden durch eine sinnvolle Startreihenfolge definiert.

Upstart ist ein Ubuntu-Projekt, das den Systemstart von Ubuntu (bzw. Linux) beschleunigt. Upstart verwendet dabei einen neuen Ansatz. Programme und Dienste werden nicht mehr zielorientiert, sondern **ereignisbasiert** geladen und gestartet, wodurch mehrere Dienste parallel und voneinander unabhängig gestartet werden können. Hierdurch wird die Startgeschwindigkeit verbessert. Dabei ersetzt Upstart den klassischen Init-Prozess. Mittel- bis langfristig soll Upstart zum zentralen System von Ubuntu zur Verwaltung von Ereignissen ausgebaut werden. Dann könnte Upstart auch andere ereignisbasierte Dienste wie `at` oder `cron` ablösen. Sowohl Upstart als auch SysV-Init werden vom Kernel als erster Prozess mit der Id 1 gestartet, sobald dieser gebootet und etwaige Boot-Skripte aus der Initial Ramdisk abgearbeitet hat.

Viele Dienste hängen vom Funktionieren bestimmter Hardware ab. So können manche Dienste erst gestartet werden, wenn die dazu nötige Hardware initialisiert wurde. Oder nes können, um ein anders Beispiel zu nehmen, Netzdienste erst gestartet werden, wenn die Verbindung zum Netzwerk steht. Die Entwicklung steht noch am Anfang, aber Ubuntu verwendet Upstart schon seit einiger Zeit und auch die "testing"-Version von Debian setzt es ein. Um den reibungslosen Betrieb der Distributionen zu ermöglichen, erfolgen etliche Teile der Initialisierung mit Hilfe des Kompatibilitätsscriptes in `/etc/init.d/rc` durch die üblichen Init-Dateien. Die Ausweitung auf Teile des Systemstarts erfolgt kontinuierlich in jeder neuen Ubuntu-Versionen, wobei temporale Events noch nicht integriert sind.

Upstart wartet also auf bestimmte Events. Sobald ein Ereignis eintritt, führt Upstart alle passenden Aktionen in Form spezieller Scripten aus. Diese wecken wiederum alle notwendigen Dienste oder richten die Hardware ein. Die einzelnen Scripte, die beim Eintreffen eines Events ausgeführt werden sollen, werden bei Upstart "Jobs" genannt und enden in der Regel auf `".conf"`. Die Scripte befinden sich alle im Verzeichnis `/etc/init` (anfänglich in `/etc/event.d`). Wie beim Init wird auch bei Upstart das Programm `/sbin/init` als erster Prozess gestartet. Die Datei `/etc/inittab` gibt es bei Upstart nicht mehr. Viele der Upstart-Scripte sind noch im Kompatibilitätsmodus geschrieben. So wird man bei vielen Job-Scripte (Runlevel-Scripte), derzeit feststellen, dass sie nichts anderes tun, als das entsprechende rc-Skript aufzurufen. Das zentrale Werkzeug ist nun `initctl`, das Init-Jobs startet oder stoppt, Signale verschickt und den Status abfragt. So gibt beispielsweise der Befehl `initctl list` eine Liste aller Init-Jobs und deren Status aus. Mit `initctl [start | stop] Job` werden Init-Jobs gestartet bzw. beendet.

Wie sieht nun ein solcher Job aus? Im Prinzip handelt es sich natürlich wieder um eine Art Shellscript - jedoch mit einigen Besonderheiten. Das sieht man am Besten an einem Beispiel:

```
# /hallo_welt.conf:
# Experimente mit Upstart
# Eine Beschreibung mitgeben:
description      "Popliges Upstart-Beispiel"

# wann starten bzw. stoppen (das "!" bedeutet "NOT"):
start on runlevel [23]
stop on runlevel [!23]
env enabled=1
PATH_BIN=/bin/bash
```

Betriebssystem UNIX/Linux

```
exec echo "Bonjour Monde" > /var/log/Bonjour.log
```

Hiermit wird einfach der Text "Bonjour Monde" in die Datei `/var/log/HalloWelt.log` geschrieben. Die Bedingungen für den Event sind hier sehr einfach, bei Runlevel 2 und 3 wird es gestartet und beim Wechsel in einen anderen Runlevel beendet. Beachten Sie, dass in die eigentlich Aufgabe (im Beispiel `echo`) per `exec` gewechselt wird. In einer Upstart-Job-Description steht hinter den Schlüsselworten "start on" das Ereignis, bei dem das hinter `exec` eingetragene Programm startet. Die Prozesse laufen allesamt im Vordergrund und nicht wie bei SysV-Init im Hintergrund. Umgekehrt werden die Prozesse gestoppt, wenn die Bedingung hinter "stop on" erfüllt ist.

Es lassen sich auch mehrere Events, auf die der Job reagieren soll, logisch miteinander verknüpfen, z. B.:

```
start on (runlevel [123] and (stopped kdm or stopped gdm))
```

Vor bzw. nach der eigentlichen Ausführung des Scripts lassen sich noch beliebige Befehle einfügen, die zwischen "pre-start script" und "end script" hinterlegt werden. Auch für die Ausführung nach dem Starten des Haupt-Scripts lassen sich auf die gleiche Weise Befehle zwischen "post-start script" und "end script" unterbringen. Fall nun jemand einwirft, dass nach einem `exec` das Script verlassen wurde: Das `exec` läuft unter der Kontrolle von Upstart ab.

Mit Hilfe der Schlüsselwörter "pre-stop script" und "post-stop script" lassen sich Befehle angeben, die Upstart vor und nach dem Beenden des Dienstes ausführen soll - etwa für Aufräumarbeiten, z. B.:

```
post-stop exec rm -f /var/run/hello.pid
```

Für unser Bonjour-Script hätte ich beispielsweise noch für das Vorhandensein der Datei sorgen können:

```
pre-start script
  touch /var/log/Bonjour.log
end script
```

Beim Stoppen eines Jobs bearbeitet Upstart nur den per `exec` gestarteten Prozess. Dieser erhält zuerst ein Terminate-Signal (SIGTERM). Beendet sich der Prozess nicht, wird er kurz danach per Kill-Signal (SIGKILL) abgebrochen.

Da es `/etc/inittab` nicht mehr gibt, müssen auch die Terminalkonsolen per Upstart eingerichtet werden. Deshalb gibt es auch hier ein "respawn". Da der Job `rc`, nach dem Aufruf der Init-Skripte des jeweiligen Runlevels terminiert, starten die tty-Jobs nach Beenden des `rc`-Jobs und nicht etwa bei dessen Start, z. B.:

```
start on stopped rc
stop on runlevel [!2345]
respawn
exec /sbin/getty -8 19200 tty1
```

Das Programm `getty` läuft im Vordergrund und wird (wie früher von Init) von Upstart überwacht. Durch das Schlüsselwort "respawn" weiss Upstart, dass der Prozess immer wieder neu gestartet werden muss, wenn er sich beendet. Um eine Überlastung des Systems zu verhindern, wenn ein Prozess amok läuft und immer wieder startet, kann die "Startgeschwindigkeit" beschränkt werden:

```
start on stopped rc
stop on runlevel [!2345]
respawn
```

```
respawn limit 10 120  
exec /sbin/getty -8 19200 tty1
```

Das Limit für den Neustart wird hier mit "respawn limit" auf 10 Versuche innerhalb von 120 Sekunden festgelegt.

1.9 Sicherheitsmaßnahmen bei Arbeiten mit Root-Rechten

Grundsätzlich sollten nur diejenigen Arbeiten mit Superuser-Rechten vorgenommen werden, die wirklich Root-Rechte benötigen. Da 'Root' wirklich alles darf, können Fehler massive Probleme bringen (z. B. das Löschen von Dateien oder das Ändern von Zugriffsrechten). Alle anderen Arbeiten können als 'Normaluser' oder mit Privilegien durch besondere Gruppenrechte durchgeführt werden. Durch Setzen von Gruppenrechten können auch bestimmte Benutzer mit Administrativen Aufgaben für bestimmte Bereiche betraut werden. Anmerkung: Durch die Vergabe von Gruppenrechten kann man sogar bestimmte Benutzergruppen "aussperren". Man ordnet den Benutzern und den zu vor ihnen zu schützenden Verzeichnissen und Dateien dieselbe Gruppe zu. Danach nimmt man bei den Dateien und Verzeichnisse die Gruppenzugriffsrechte weg. Bei Änderungen an "lebenswichtigen" Dateien, zu denen auch die oben genannten gehören, sollte man noch einen Superuser-Login auf einen zweiten Terminal haben laufen haben. Nach den Änderungen wird dann getestet, ob beim root-Login noch alles in Ordnung ist. Im Notfall kann man dann vom zweiten Terminal aus eingreifen. Auch hier gilt grundsätzlich, daß man sich sicher darüber sein sollte, was man eigentlich tut. Von zu verändernden Dateien werden zunächst Sicherheitskopien angefertigt. Bei Shell-Scripts und Konfigurationsdateien kann man vor dem Ändern einzelner Zeilen die betreffenden Zeilen zuerst kopieren und dann die "alte" Version der Zeile durch eine Kommentar-Markierung deaktivieren (bei Unix sind alle Zeilen, die mit '#' beginnen, Kommentare). Das gilt auch für zu löschende Einträge, die ebenfalls "auskommentiert" werden. So kann man leicht zum alten Stand zurückkehren, wenn nach der Änderung etwas nicht klappt. Als Systemadministrator sollte man auch ein Tagebuch führen. Das kann ein einfaches Heft sein, in das man einträgt, was wann geändert wurde. Manche Fehler zeigen sich erst später und nur auf sein Gedächtnis sollte man sich nicht verlassen.

1.10 Das X Window System

Der Name

Fast jeder sagt "X-Windows", dieser Name ist jedoch nicht korrekt. Das X Consortium hat die folgenden Namen als richtig festgelegt:

- X
- X Window System
- X Version 11
- X Window System, Version 11
- X11

Das X Window System

Das X-Window-System ist ein netzbasiertes graphisches Fenstersystem, das auf einer Client-Server-Architektur basiert:

Betriebssystem UNIX/Linux

- Programme, die grafische Ausgaben tätigen, sind die Clients, die den Service in Anspruch nehmen,
- der Display-Server bringt die Ausgabe auf dem Bildschirm, verarbeitet Eingaben von Tastatur und Maus und sendet diese an den Client.

Dabei können Client und Server über ein Netzwerk miteinander kommunizieren. Es ist also unerheblich, ob die Informationen, die auf einem Bildschirm dargestellt werden, auf dem lokalen Rechner laufen oder auf einem anderen Rechner im Netz.

Client und Server verständigen sich mit einem Protokoll, dem sogenannten *X-Protokoll*. Dieses Protokoll nutzt als Transportbasis meist TCP/IP, aber der Betrieb auf reinen Unix-Sockets ist ebenso möglich (lokaler Betrieb). Technisch gesehen ist das X-Protokoll die eigentliche Definition des X-Window-System.

Zur Verdeutlichung noch ein Hinweis: bei den meisten anderen Client-Server Systemen (beispielsweise Datenbanksystem, Mailsystem usw.) befindet sich der Client näher am Benutzer als der Server. Bei X ist das naturgemäß umgekehrt, da der Server Tastatur und Bildschirm verwaltet und den Clients zur Verfügung stellt.

X begann auch als Protokollspezifikation. Nachdem vor einigen Jahren immer schnellere Rechner mit Bitmap-Grafikdisplays erhältlich waren, ging das MIT mit der Unterstützung von einigen Firmen (dem X Consortium) daran, das Windowsystem für die Zukunft zu spezifizieren. Es wurde dabei zuerst ein Protokoll festgelegt. Danach begann das MIT mit einer Beispielimplementierung des Protokolls, um zu zeigen, wie es funktioniert und welche Möglichkeiten es bietet. Nach einigen Jahren begannen dann Firmen mit dem Vertrieb von kommerziellen Implementierungen des X Protokolls.

Da das Protokoll allerdings recht aufwendig war - es teilt sich der Klarheit wegen in mehrere streng getrennte Schichten - waren die anfänglichen Implementierungen relativ langsam. Wesentlich langsamer jedenfalls als solche Windowsysteme, die direkt auf das Bitmap-Display zugreifen. Es folgten dann immer mehr Programme, die auf dem X Window System aufbauen.

Das X Window System ist jedoch keine einheitliche Benutzeroberfläche, die ein bestimmtes "Look-and-Feel" bietet. X könnte aussehen wie der Macintosh Finder oder wie Microsoft Windows. Dieses deshalb, weil das X-Protokoll sehr einfach ist. Man kann lediglich grafische Elemente (Linien, Kreise, Widgets etc.) und Zeichen auf dem Bildschirm anzeigen. Das Protokoll enthält keine komplexeren Grafikelemente wie Buttons oder Menus. Deshalb gibt es auch keine Aussagen über Aussehen von Anwendungsprogrammen (Style Guide), so daß sich mehrere Standards gebildet haben.

Möchte man zum Beispiel einen Knopf (Button) mit einer Aufschrift, so muß dieser aus Linien und Text selbst zusammengesetzt werden. Dies kann dem Programmierer aber auch durch ein Toolkit abgenommen werden. Diese Toolkits bestimmen dann hauptsächlich das Look-and-Feel.

Komponenten von X

X benötigt hardwareseitig ein Bitmap-Display, eine Tastatur und ein Pointing-Device (Maus, Grafiktablett, etc.)

Das Pointing-Device muß nur zum Zeigen auf Punkte fähig sein. Es könnte zum Beispiel auch ein Touch-Screen sein. Oder eine Maus mit nur einer Taste. Soll ein Programm konform zur X-Spezifikation sein, müssen alle Funktionen mit nur einer Maustaste ausführbar sein. Dies kann man z. B. erreichen, indem man beim Drücken der Maustaste auch noch gleichzeitig gedrückte Tasten (Control, Meta, etc.) abfragt.

Softwareseitig gibt es folgende Prozesse:

- X Server** Der X-Server ist - wie gesagt - das Programm, das alle Bildschirmausgaben übernimmt und alle Eingaben von der Tastatur und der Maus verarbeitet. Daher ist ein Teil des X-Servers sehr an die Hardware des Rechners gebunden (Farb- oder Schwarzweiß-Bildschirm, Art der Tastatur, Anzahl der Maustasten, Bildschirmgröße ...). Ein Programm, das etwas auf dem Bildschirm ausgeben will, schickt einen diesbezüglichen Auftrag an den X-Server, der daraufhin eine Linie zeichnet, einen Text ausgibt oder tut, was immer das Programm von ihm verlangt. In der anderen Richtung gibt der X-Server Meldungen an die X-Clients, wann immer der Benutzer eine Eingabe getätigt hat, sei es das Bewegen der Maus, das Drücken einer Maustaste oder eine Eingabe über die Tastatur. Die Programme können dann entscheiden, was sie mit dieser Eingabe anfangen und wie (oder ob überhaupt) sie darauf reagieren. Vorteil dieser Konfiguration ist, daß nur der X-Server über die Möglichkeiten der vorhandenen Hardware informiert sein muß. Die Clients können diese Information vom Server erfragen, wenn sie sie brauchen, müssen sich ansonsten aber nicht darum kümmern.
- X Clients** Jedes Programm, das auf einem X-Bildschirm ein Fenster darstellen will, ist ein X-Client. Der X-Client bittet den Server, gewisse Aufgaben (eben das Zeichnen des Fensters) für ihn zu übernehmen. Sie müssen dazu Aufträge im X Protokoll an den Server schicken.
- Window-Manager** Ein Client hat den anderen gegenüber einen gewissen Sonderstatus: der Window-Manager, hier heißt er `ctwm`. Er stellt dem Benutzer Mittel zur Verfügung, mit deren Hilfe dieser das Aussehen seiner Benutzeroberfläche bestimmen kann. Insbesondere kann der Window-Manager die Größe und die Position der Fenster anderer Clients beeinflussen. Die Clients können dem Window-Manager Hinweise geben, wo und in welcher Größe sie ihre Fenster plaziert haben wollen. Der Window-Manager muß diese Hinweise zwar nicht berücksichtigen, die meisten gängigen Systeme tun dies aber. Durch die Funktion des Window-Managers ergibt sich, daß es zumindest problematisch ist, zwei Window-Manager für einen Bildschirm zu starten. Daher prüfen Window-Manager beim Start in der Regel, ob bereits ein anderer Window-Manager für den Bildschirm existiert und brechen in diesem Fall die Arbeit ab.

Die Rahmen und Titelbalken, die die einzelnen Fenster verzieren, sind nicht Teil des jeweiligen Clients, sondern werden vom Windowmanager um die Fenster herumgezeichnet, damit durch ihre Betätigung Funktionen des Windowmanagers ausgelöst werden können. Der Windowmanager bestimmt somit auch "look-and-feel" der Benutzeroberfläche. Beachten Sie aber, daß der Window-Manager aber auch nur ein ClientProzess ist. Es gibt alle möglichen Window-Manager:

`vtwm` Virtual twm (virtueller Screen mit Window)
`gwm` Generic Window Manager (gut konfigurierbar)
`olwm` OPEN LOOK Window Manager
`mwm` Motif Window Manager
... Und viele andere

Auf dem Window Manager kann dan noch ein sogenanntes Desktop Environment aufsetzen, das nicht nur das heute gewohnte und typische

Betriebssystem UNIX/Linux

Erscheinungsbild bietet, sondern auch Funktionen wie "Drag-and-Drop", "Cut-and-Paste" usw. Bekannte Vertreter dieser Gattung sind "Gnome" oder "KDE".

Anmerkung: "Cut-and-Paste" konnte X schon immer. Einfach in einem Fenster den Text markieren und dann im anderen Fenster die *mittlere* Maustaste drücken - geht immer und überall.

Was ist das Besondere an X?

X ist ein portables Windowsystem. X benötigt nur Benutzerprozesse und keine Veränderungen am Betriebssystemkern. X-Server gibt es auch auf anderen Betriebssystemen (MS-DOS, MacOS, Atari TOS, Windows etc.) und es gibt sogenannte X-Terminals, intelligente Grafikterminals, die einen integrierten X Server haben.

X ist netzwerkfähig. Clients können ihre Grafikausgabe auch auf Server machen, die auf anderen Rechnern im Netz laufen. Als Netzwerkprotokolle können dabei verschiedene Protokolle eingesetzt werden. Bei Unix ist dies meist TCP/IP.

Wie benutzt man X über ein Netzwerk

Das X Protokoll unterstützt ja auch Rechner, die vernetzt sind. Das bedeutet, man kann Ausgaben von einem Client auf Rechner A auch auf einem Server auf Rechner B ausgeben. Dafür ist folgendes notwendig:

Auf dem Zielrechner muß dem Server mitgeteilt werden, daß er Requests vom Senderechner zulassen darf. Das geschieht mit dem Kommando:

```
xhost +senderechner
```

Auf dem Senderechner muß man dem Client mitteilen, daß die Ausgabe nicht auf dem eigenen Display erscheinen soll, sondern beim Zielrechner. Dazu setzt man entweder die Environment-Variable

```
export DISPLAY=zielrechner:0
```

oder man schreibt beim Aufruf des Programms

```
clientprogramm -display zielrechner:0
```

Zum Beispiel startet man auf einem fernen Rechner in einem Terminal mit dem Kommando

```
firefox -display myhost:0.0 &
```

den Browser und leitet dessen Ausgabe auf den eigenen Bildschirm (myhost:0.0).

Die Zahl hinter dem Rechnernamen gibt die Nummer des Displays an. Sie kann auch eine Nachkommastelle haben, z.B. 0.0. Normalerweise haben aber die Rechner nur ein Display mit der Nummer 0.

Noch einfacher geht es mit **SSH**. Ein xterm mit einer Shell auf einem Remote-Rechner erhält man mit dem Kommando: `ssh -X [SSH_OPTIONS] [USER@]HOST` SSH regelt automatisch die Verwendung von X-Window. Man muß also auf dem Remote-Rechner weder eine X-Window-Authorisierung noch die Environment-Variable DISPLAY setzen.

Die X Session

Entweder sofort nach dem Einloggen oder durch das Kommando `startx` wird für den Benutzer eine Session gestartet. Dabei werden automatisch verschiedene Clients gestartet (`xterm`, `twm`, `xclock`, etc.). Die Session beendet sich, wenn der Windowmanager beendet wird.

Der Mechanismus funktioniert folgendermaßen: von Systemprozessen wird eine Prozedur aufgerufen, welche die Session steuert. Wenn sich die Prozedur beendet, übernehmen wieder die Systemprozesse die Steuerung.

```
process xdm is
  while (true)
  do
    xlogin;
    if (Benutzer hat eigene Session-Prozedur)
      then fuehre Benutzer-Session-Prozedur aus
    else if (Benutzer hat zusätzliche Session-Prozedur)
      then fuehre zusätzliche Session-Prozedur im Hintergrund aus
    fi;
    starte xterm im Hintergrund;
    starte xclock im Hintergrund;
    starte twm;
  fi
done
endprocess
```

Bei `xdm` heißt die Benutzer-Session-Prozedur `".xsession"` und die zusätzliche Session-Prozedur `".xsession+"`. Man sollte darauf achten, daß diese Dateien ausführbar sind. Außerdem wird die Session beendet, wenn sich die Prozedur beendet. In der Prozedur sollten also alle Kommandos im Hintergrund gestartet werden, außer dem letzten, das während der gesamten Session laufen muß. Dies ist meist der Window-Manager. Wird dieser dann mit `Exit` beendet, wird auch die Session mit allen anderen Clients beendet.

X und Sicherheit

Ein Problem von X ist noch die Sicherheit. Zum Beispiel kommt es vor, daß `xterm`-Fenster beim Ausloggen nicht geschlossen werden, sondern beim nächsten Benutzer wieder auf dem Bildschirm erscheinen.

Ein anderes Problem liegt am Konzept des TCP/IP-Protokolls. Da nicht alle Rechnertypen, die am Internet hängen, das Konzept einer Benutzer-ID haben, ist diese auch bei TCP/IP nicht vorgesehen. Auf den X-Server kann also nicht nur der augenblickliche Benutzer der Konsole zugreifen, sondern alle Benutzer des Rechners. Und das in beliebiger Form. Ein anderer Benutzer kann zum Beispiel den gesamten Bildschirminhalt überschreiben.

Bei jedem Anmelden an einem Rechner generiert das Programm `xdm`, das die Anmeldemaske zur Verfügung stellt, einen Schlüssel und legt ihn in der Datei `.Xauthority` im HOME-Verzeichnis des Benutzers ab. Jedes X-Programm, das der Benutzer dann startet, sucht in dieser Datei nach dem Schlüssel und gibt ihn dem Server beim Aufbau der Verbindung an. Nur wenn dieser Schlüssel mit dem übereinstimmt, der beim Login generiert wurde, wird die Verbindung tatsächlich aufgebaut, ansonsten wird der Aufbauversuch vom Server zurückgewiesen. Auf diese Art wird verhindert, daß jeder Benutzer seine Fenster auf den Bildschirm eines anderen Benutzers legen und diesen dadurch bei seiner Arbeit behindern kann. Voraussetzung für die Sicherheit des Systems ist natürlich, daß die Rechte für die Datei `.Xauthority` richtig gesetzt sind. Nur der Eigentümer der Datei darf dafür das Lese- und Schreibrecht haben, alle anderen Benutzer nicht einmal das Leserecht, da sie sonst den Schlüssel aus der Datei herauslesen könnten.

X ermöglicht unter anderem jedem Programm, das eine Verbindung zum X-Server aufbauen kann, das Mitlesen von Tastatureingaben, die auf dem Rechner vorgenommen werden. Darunter können natürlich auch Paßworteingaben sein (zum Beispiel bei einem `rlogin`). Daher darf der Zugriff auf den Server auf keinen Fall unkontrolliert freigegeben werden.

Der Terminalemulator `xterm`

Um Nicht-X-Programme unter X laufen lassen zu können, gibt es den Terminalemulator `xterm`. Er erzeugt ein Fenster, das sich wie der normale Textbildschirm an einem Unix-Rechner verhält. Um die Nutzung optimal zu gestalten, können mit dem Aufruf mehrere Optionen eingestellt werden, von denen einige hier aufgeführt werden (Vollständig sind sie in der Manualseite aufgeführt):

```
xterm [Optionen]
```

oder

```
xterm [Optionen] & (als Hintergrundprozess)
```

Optionen:

`-e name [parameter]`: Das Programm `name` wird sofort ausgeführt, wenn sich das Fenster öffnet. Am Ende des Programms wird das Fenster wieder geschlossen. Die angegebenen Parameter werden dem Programm übergeben. Der Fenstertitel und Iconname heißen dann auch `name` (soweit keine Option `-T` oder `-n` verwendet wurde).

Die Option muß als letzte in der Optionenfolge eingegeben werden, da alles, was der gesamte Zeilenrest als Teil des auszuführenden Befehls betrachtet wird! Beispiel:

```
xterm -e vi main.c
```

- `-l`: Jede Ausgabe in `xterm` wird in der Datei `XtermLog.nnnn` (`nnnn` entspricht der Prozessnummer) abgespeichert und gleichzeitig im Fenster angezeigt.
- `+l`: Kein Logging.
- `-lf Dateiname`: Die Logging-Ausgabe wird in der Datei `Dateiname` abgelegt.
- `-sb`: Erzeugt den Scroll-Balken am linken Fensterrand.
- `+sb`: Kein Scroll-Balken.
- `-sl nn`: `nn` gibt an, wieviele Zeilen nach dem Herausrollen aus dem Anzeigebereich gespeichert werden sollen. Voreinstellung: 64 Zeilen.
- `-T string`: setzt `string` als Fenstertitel.
- `-n string`: setzt `string` als Fenstername für den Iconmanager.
- `-rv`: Tausche Vordergrund- und Hintergrundfarbe.
- `-geometry expression`: gibt an, wie groß das Fenster sein soll und wo es plziert werden soll. Das Format für `expression`:

```
breite x höhe +xkoord +ykoord
```

'breite' und 'höhe' geben die Fenstergröße (in Buchstaben) an, 'xkoord' und 'ykoord' die Position (in Pixeln). Ein `xterm`-Fenster hat gemäß Voreinstellung 24 Zeilen mit 80 Zeichen Breite.

- `-iconic`: Das aufgerufene Fenster wird sofort iconifiziert. Es erscheint also nur als Eintrag im Iconmanager (falls vorhanden) und/oder als einzelnes Icon.

Die Datei `.xsession`

Diese Datei wird nach dem Anmelden vom System abgearbeitet. Es handelt sich bei ihr um ein Shellscript, das von der Standardshell des Benutzers. Die Sitzung eines Benutzers dauert so lange, bis die Datei `.xsession` vollständig abgearbeitet wurde. Daraus folgt, daß `.xsession` mindestens ein Programm nicht als Hintergrundprozess starten darf. Das Ende dieses Programms stellt dann auch das Ende der Sitzung dar. Es ist sinnvoll, Programmen, die man in `.xsession`

startet, eine Positionsangabe mitzugeben, entweder direkt oder über `.Xdefaults`. Tut man das nicht, muß man das Fenster jeweils per Hand auf dem Bildschirm positionieren (siehe unten).

Die Datei `.xxxwmrc`

Diese Datei konfiguriert den Window-Manager (wobei "xxx" für den gewählten Window-Manager steht. Dieser stellt er die Menüs zur Verfügung, die erscheinen, wenn man auf dem *root-Fenster* (dem Bildschirmhintergrund) die mittlere oder die rechte Maustaste drückt. Diese Datei dient dazu, die Benutzeroberfläche frei nach seinen eigenen Wünschen zu gestalten.

Die Datei `.rhosts`

Diese Datei gibt an, wer sich von welchem Rechner aus ohne Angabe eines Paßwortes am Rechner anmelden darf. Damit kann man vermeiden, daß bei jedem Aufruf von `rlogin` das Paßwort eingegeben werden muß. `.rhosts` darf nur für den User selbst schreibbar sein. Sie muß jedoch für alle lesbar sein. Jede Zeile in `.rhosts` muß folgenden Aufbau haben:

Rechnername [*Benutzer*]

Die Datei `.Xdefaults`

Viele X-Clients ermöglichen es, ihr Aussehen und ihre Konfiguration mittels sogenannter *Ressourcen* festzulegen. Ressourcen sind im Grunde nur Variablen, die der X-Server bereithält und die der X-Client bei Bedarf abfragen kann. Meist erkundigt sich der Client beim Server, ob für ihn Ressourcen definiert sind und verwendet diese, falls es welche gibt. Ansonsten benutzt er Standardwerte. Jeder Benutzer kann seine eigenen Ressourcendefinitionen in der Datei `.Xdefaults` in seinem HOME-Verzeichnis ablegen. Diese wird, falls vorhanden, automatisch beim Anmelden dem X-Server bekanntgemacht.

Konfiguration von X-Clients über die Kommandozeile

Wie bei allen Kommandos kann man auch beispielsweise dem `xterm` Parameter auf der Kommandozeile mitgeben. Der Unterschied besteht darin, dass hier das Aussehen des Terminalfensters festgelegt wird. Was im Folgenden am Beispiel `xterm` gezeigt wird, geht natürlich auch mit anderen Programmen.

Die Auswahl der Schriftart erfolgt mit dem Parameter `-fn`. Die Schriftnamen unter X11 sind etwas komplex; weiter unten wird diese Nomenklatur noch etwas erläutert. Momentan genügt das Wissen, dass es ein Programm gibt, das alle im System vorhandenen Schriften auflistet: `xlsfonts`. Sein Aufruf listet alle Schriften auf:

```
-adobe-courier-bold-o-normal--10-100-75-75-m-60-iso8859-1
-adobe-courier-bold-o-normal--11-80-100-100-m-60-iso8859-1
-adobe-courier-bold-o-normal--12-120-75-75-m-70-iso8859-1
...
```

Mit `xfontsel` kann man sich die Schriften sogar ansehen.

Um ein `xterm` mit einer bestimmten Schrift zu starten, ruft man es mit Parameter auf:

```
xterm -fn adobe-courier-bold-o-normal--10-100-75-75-m-60-iso8859-1 &
```

X11 verwaltet den Bildschirm als zweidimensionale Fläche mit kartesischem Koordinatensystem. Die obere linke Ecke entspricht der Position `0, 0`. Die untere, linke Ecke wird durch die maximale Auflösung bestimmt. Ein X-Client kann mit dem Parameter `-geometry` Größe und Position des

Betriebssystem UNIX/Linux

Fensters auf dem Schirm angeben. Beginnen wir mit der Größe. Das `xterm`-Fenster soll anstatt der Standardgröße von 80 Spalten und 24 Zeilen nun 90 Spalten und 30 Zeilen besitzen:

```
xterm -geometry 90x30 &
```

Nach dem `-geometry` folgt die Angabe von Spalten und Zeilen (mit einem "x" dazwischen). Wem das Wort "geometry" zu lang ist, kann es auch mit "g" abkürzen (meistens).

Achtung: Die Angabe von Zeilen und Spalten bezieht sich beim `xterm` auf Zeichen, nicht auf Pixel. Graphikorientierte Programme rechnen hier mit Pixeln, zeichenorientierte Programme mit Zeichen. Um also etwa ein Programm wie `xclock` in der Größe zu ändern, muss in Pixeln gerechnet werden.

Die zweite Aufgabe von `-geometry` ist die Positionierung eines Fensters auf dem Schirm. Dazu wird der Offset zum Rand des Schirms angegeben, ein Pluszeichen meint den oberen bzw. den linken Rand, ein Minuszeichen den unteren bzw. rechten. Je nach verwendeten Zeichen + oder - ergibt sich die Ecke des Bildschirms und des Fensters, auf die sich die Koordinaten jeweils beziehen. Es gibt vier Möglichkeiten:

- +0+0 obere, linke Ecke
- -0+0 obere, rechte Ecke
- -0-0 untere, rechte Ecke
- +0-0 untere, linke Ecke

Dazu ein Beispiel:

```
xterm -geometry +10+50 &
```

positioniert die obere linke Ecke des Terminalfensters 10 Pixel vom linken Rand und 50 Pixel vom oberen Rand entfernt. Bei dieser Angabe sind übrigens **immer** Pixel die Masseinheit. Um sich z. B. auf die untere linke Ecke zu beziehen schreiben Sie:

```
xterm -geometry -10-50 &
```

Es ist sogar möglich, die Bezugskanten zu mischen. Die Zeile

```
xterm -geometry +10-100 &
```

positioniert das Fenster so, dass sein linker Rand 10 Pixel vom linken Rand des Bildschirms entfernt ist und seine untere Begrenzung 100 Pixel vom unteren Rand.

Die Angaben von Größe und Position lassen sich kombinieren, wobei immer die Größe zuerst angegeben wird, danach folgt (ohne Leerzeichen aber mit Vorzeichen + oder -) die Positionsangaben. Um ein `xterm` mit 90 Spalten und 30 Zeilen an der Position 10,100 zu erhalten schreiben Sie:

```
xterm -geometry 90x30+10+100 &
```

Wird keine Angabe zur Position gemacht, so übernimmt der Window-Manager die Positionierung.

Zur Einstellung der Farbe dienen in der Regel die Parameter `-fg` für die Vordergrundfarbe (foreground) und `-bg` für den Hintergrund (background). Das Farbmodell von X wird weiter unten besprochen. Auch hier existiert ein Programm, das die Namen aller bekannten Farben auflistet: `showrgb`. Es liest einfach nur die Datei `/var/X11R6/lib/rgb.txt`, in der die Farbnamen und Farben definiert sind. Um ein `xterm` mit blauem Hintergrund und gelber Schrift zu erhalten,

geben Sie folgendes Kommando ein:

```
xterm -bg blue -fg yellow &
```

Mit Font, Geometrie und Farben kann man schon sehr viel Individualität in seine X-Oberfläche bringen. Die Festlegung der Programmeigenschaften über die Kommandozeile kann zu recht langen Kommandos führen, wie das folgende Beispiel zeigt:

```
xterm -geometry 90x30-10-10 \  
-fn -adobe-courier-bold-o-normal--10-100-75-75-m-60-iso8859-1 \  
-fg blue -bg Yellow &
```

Deshalb gibt es noch eine andere Möglichkeit:

Konfiguration von X-Clients über Ressourcen

Um oft gebrauchte Einstellungen festzulegen steht das oben erwähnte Ressourcen-System zur Verfügung. Ressourcen sind Variablen, die ein Programm benutzt und die von außerhalb des Programms verändert werden können. Die Grundeinstellungen dieser Ressourcen stehen im Verzeichnis `/var/X11R6/lib/app-defaults`. Jeder User kann aber diese Werte für sich verändern, um Programme mit anderen Werten zu starten. Das geschieht in der Regel über die Datei `.Xresources` im jeweiligen Heimatverzeichnis des Users. Dort können Zeilen wie die folgenden eingetragen werden:

```
XTerm*font: -adobe-courier-bold-o-normal--10-100-75-75-m-60-iso8859-1  
XTerm*Background: yellow  
XTerm*Foreground: blue  
XTerm*geometry: 90x40
```

Die Analogie zu der Kommandozeile im Beispiel oben ist offensichtlich. Die Ressource-Werte werden dann beim nächsten Start des X11-Servers in den Speicher geladen und aktiviert. Jedesmal, wenn Sie jetzt `xterm` ohne Parameter aufrufen erscheint ein Fenster mit dem entsprechenden Aussehen.

Ressourcen sind hierarchisch aufgebaut. Jedes Programm greift auf eine Klasse von Ressourcen zu (`xterm` zum Beispiel auf `Xterm`). Darunter können sich Komponenten des Programms befinden und am Ende der Hierarchie stehen Eigenschaften. Teile der Hierarchie können auch durch Wildcards (*) ersetzt werden.

Was kann man nun überhaupt mit Ressourcen festlegen? Eigentlich alles! Welche Ressourcen ein Client versteht, kann in der Regel in seiner Manualpage nachgelesen werden. Die grundsätzliche Form eines Eintrags ist immer die Gleiche:

```
Clientname*Resource: Wert
```

Es gibt noch einen netten Trick, für ein Programm mehrere Alternativ-Ressourcen zu erstellen: die Verwendung von symbolischen Links. Sie können beispielsweise folgende Zeilen in der Datei `.Xresources` aufnehmen:

```
gelbterm*font: -adobe-courier-bold-o-normal--10-100-75-75-m-60-iso8859-1  
gelbterm*Background: yellow  
gelbterm*Foreground: black  
gruenterm*font: -adobe-courier-bold-o-normal--10-100-75-75-m-60-iso8859-1  
gruenterm*Background: green  
gruenterm*Foreground: black
```

Jetzt erstellen Sie noch zwei symbolische Links auf `/usr/X11R6/bin/xterm`:

Betriebssystem UNIX/Linux

```
ln -s /usr/X11R6/bin/xterm /usr/X11R6/bin/gelbterm
ln -s /usr/X11R6/bin/xterm /usr/X11R6/bin/gruenterm
```

Damit gibt es nun zwei neue Programme, `gelbterm` und `gruenterm`. Nachdem die Datei `.Xresources` neu geladen wurde (Neustart des X-Servers oder Aufruf von `xrdb -merge .Xresources`) können Sie die Programme aufrufen.

In der Regel haben Clients sehr viele Ressourcen, wesentlich mehr als mögliche Kommandozeilenparameter. So kann einem Xterm beispielsweise durch die Resource `XTerm*scrollbar:true` eine Scrollbar hinzugefügt werden.

Damit alle diese Ressourcen auch über die Kommandozeile möglich sind, gibt es den Parameter `-xrm`, dem ein Ressourcenstring folgen kann. Damit ist es möglich, Programmeinstellungen über die Kommandozeile vorzunehmen, für die es zwar Ressourcen gibt aber keine Parameter.

Das Farbmodell von X11

Farben werden in X immer als analoge Werte von Rot-, Grün- und Blau-Anteil gewertet. Die numerische Darstellung der Farben ist einfach ein Doppelkreuz gefolgt von Zahlenwerten für rot, grün und blau wird unter X11 als Farbwert interpretiert. Dabei kann jede Farbe entweder eine, zwei, drei oder vier hexadezimale Stellen haben (auch wenn die meisten Grafikinterfaces nur Werte zwischen 0 und FF "verkräften"). Schematisch dargestellt:

```
#RGB
#RRGGBB
#RRRGGGBBB
#RRRRGGGGBBBB
```

wobei R, G und B die hexadezimalen Werte für die Farben Rot, grün und blau bedeuten. An jeder Stelle, an der Farbangaben gemacht werden können können auch diese Werte statt Farbnamen stehen - zum Beispiel würde `xterm -bg #B7BB6E` ein Terminalfenster öffnen, dessen Hintergrundfarbe ein hässliches Grün wäre. Es gibt auch die Möglichkeit, Farbnamen zu definieren. In der Datei `/var/X11R6/lib/rgb.txt` sind 738 fest definierte Farben enthalten, jeweils mit ihren numerischen Werten und einem Namen.

Diese Datei hat ein leicht zu überblickendes Format. Jede Zeile repräsentiert eine Farbe, die ersten drei Felder der Zeile enthalten die **dezimalen** Werte für die Rot-, Grün- und Blauanteile, das vierte Feld den Namen der Farbe. Jeder Wert für die Farbanteile kann von 0 bis 255 reichen (24-Bit-Darstellung mit 16 Millionen Farben). Bei Grafikmodi mit weniger möglichen Farben werden die Werte umgerechnet. Zum Beispiel:

```
250 235 215  AntiqueWhite
255 239 213  papaya whip
255 239 213  PapayaWhip
255 235 205  blanched almond
255 235 205  BlanchedAlmond
255 228 196  bisque
```

Um eigene Farben zu definieren, müssen Sie nur diese Datei entsprechend erweitern. Nach einem Neustart des X-Servers existiert die neue Farbe. Zum Beispiel fügen Sie die folgende Zeile hinzu:

```
183 187 110  KotzeGruen
```

Nach einem Neustart des Display-Servers kann nun mit `xterm -bg KotzeGruen` ein X-Terminal aufgerufen werden, dessen Hintergrund in dieser Farbe gestaltet ist. Im gesamten X11-System steht diese Farbe jetzt zur Verfügung. Aber Vorsicht: Sollte einmal eine Anwendung auf einem anderen Server laufen, dessen Datei `rgb.txt` diese Zeile nicht enthält, kann er mit der

Farbe nichts anfangen.

Fontdefinition bei X11

Schriftnamen haben eine etwas komplexe Form, die am folgenden Beispiel erläutert werden soll:

Hersteller	Gewicht	Breite	Pixels	XAufl	Sp	Zeichensatz
<code>-b&h-lucida-medium-r-normal-sans-18-180-75-75-p-106-iso8859-1</code>						
Schriftart	Neigung	Stil	Punkte	YAufl	Durchschn. Zeichenbreite	Optionen des Zeichensatzes

Dabei bedeuten im Einzelnen:

Hersteller: Meist die Herstellerfirma der Schriftart, manchmal aber auch nur eine weitere Schublade wie misc oder bitstream.

Schriftart: Die "Schriftfamilie" also etwa Times, Helvetica, Palatino, Utopia, ...

Gewicht: Das "Gewicht" der Schwärze wie black, bold, demibold oder medium.

Neigung: Die Schräge der Schrift. "i" steht für "italic" (kursiv), "r" für "roman" (aufrecht).

Breite: Die Breite einer Schrift wie normal, narrow oder condensed.

Stil: Zusätzliche Stilangaben

Pixels: Das Höhenmaß der Schrift in Pixel (in einer bestimmten Punktgröße und Auflösung)

Punkte: Die Größe der Schrift in typographischen Punkten (1/72 Zoll) angegeben in Zehntel-Punkten

X-Auflösung: Horizontale Auflösung in Dots per Inch (DPI) in der die Schrift ursprünglich erstellt wurde.

Y-Auflösung: Vertikale Auflösung in Dots per Inch (DPI) in der die Schrift ursprünglich erstellt wurde.

Spacing: Die Beschreibung der Raumaufteilung einer Schriftart. Mögliche Werte sind hier "p" für "proportional", "m" für "monospaced" und "c" für "charactercell" (nicht proportional).

Durchschnittsbreite: Die durchschnittliche Breite eines Zeichens dieser Schrift.

Zeichensatz: Die ISO-Standard, die diese Schrift enthält.

Optionen des Zeichensatzes: Die Nummer der ausgewählten Zeichensatzseite des Standards.

Oft werden statt einzelner Angaben auch ein Sternchen angegeben, dann wird die erste passende Einstellung genommen. So wird ein * bei der Angabe der Neigung automatisch zu einem "i".



[Zum Inhaltsverzeichnis](#)



[Zum nächsten Abschnitt](#)

Copyright © FH München, FB 04, Prof. Jürgen Plate

Letzte Aktualisierung: 20. Sep 2011



Vorlesung "UNIX"

von Prof. Jürgen Plate

2 Einführung in die Shell

Die Shell ist der Kommandointerpreter von UNIX. Es haben sich drei Typen entwickelt:

- Bourne Shell (die erste Shell, angelehnt am ALGOL68), sh
- C-Shell (angelehnt an C), csh
- Korn Shell (das "Beste" aus Bourne- und C-Shell), ksh

Aus diesen sind weitere Shells entwickelt worden, z. B. die Bourne again shell, bash, die ähnliche Funktionen hat wie die ksh oder die tcsh als Abkömmling der csh.

2.1 Aufgaben der Shell

- Kommandozeile vom Terminal annehmen, das Kommando entschlüsseln und in seine Komponenten (Kommandoname, Optionen, Parameter) aufteilen
- Gewünschtes Programm suchen und starten. Der Suchweg wird durch eine Shellvariable PATH vorgegeben.
- Steuerung der Ein- und Ausgabe des Terminals. UNIX ist als Dialogsystem konzipiert und war ursprünglich für den Betrieb mit Fernschreibern (Teletype) und Datensichtgerät konzipiert. Diese sogenannten "Terminals" findet man heute noch in den *logischen* Terminals, z. B.:

- ◆ den Konsolen, die dem Bildschirm und der Tastatur des PCs zugeordnet sind,
- ◆ den Pseudo-Terminals, die einer Telnet-Verbindung zugeordnet werden oder
- ◆ den X-Terminal-Fenstern auf der grafischen Benutzeroberfläche.

Daher gibt es drei Standarddateien, die dem aktuellen Terminal zugeordnet sind:

- ◆ Standardeingabe (Tastatur, *stdin*)
- ◆ Standardausgabe (Bildschirm, *stdout*)
- ◆ Standard-Fehlerausgabe (Bildschirm, *stderr*)

Die Shell kann angewiesen werden, die Ein- und Ausgaben umzuleiten.

- Ersetzung von Sonderzeichen und Befehlssubstitution. Kommandoeingaben können durch Sonderzeichen erweitert und der Kommandostring kann selbst durch die Shell expandiert werden.
- Ablaufsteuerung. Die Shell beherrscht viele Strukturen, wie man sie von Programmiersprachen her kennt (Test, bedingte Anweisung, Schleifen, Variablen, Rechenanweisungen).
- Erzeugen von Kind- und Hintergrundprozessen.

Es gibt weiterhin die Möglichkeit, Programme "im Hintergrund" zu starten und am Bildschirm weiterzuarbeiten, während das Programm läuft. Sobald die Shell bereit ist, Kommandoeingaben anzunehmen, meldet sie sich mit einem Bereitschaftszeichen (Prompt). Im einfachsten Fall ist dies ein "\$"-Zeichen als normales Prompt, dem "#" als Prompt für Superuser und dem ">"-Prompt, wenn noch weitere Eingaben erwartet werden. Wie vieles in der Shell, kann der Prompt beliebig modifiziert werden.

Anmerkungen:

1. In diesem Kapitel geht es um grundlegende Eigenschaften der Shell, später wird das Thema noch vertieft.
2. Die Shell ist ein ganz normales Programm, das von der Standardeingabe (Tastatur) liest und auf die Standardausgabe (Bildschirm) ausgibt. Wenn das Dateiende erreicht wird (End Of File, EOF), z. B. durch Eingabe von `CTRL-D`, terminiert sie. Ist es die Login-Shell, erfolgt ein Logoff des Benutzers.

3. Die Shell kann wie ein Programm als Subshell aufgerufen werden (Schachtelung). Dies wird beispielsweise benötigt, wenn man Shell-Programme (shell skripts) testen will.
4. Suchpfade für Kommandos: Es gibt zwei Möglichkeiten, den Pfad für ein Kommando festzulegen:
 1. Direkt als absoluter oder relativer Pfadname
 2. Indirekt über den Pfad, der in der Shell-Variablen PATH gespeichert ist.

Kommandobeschreibung

In der Literatur (auch beim Kommando "man") werden die Kommandos in Kurzform beschrieben. Dabei werden Werte, die optional sind, in eckige Klammern gesetzt. Die (meist einbuchstabigen) Optionen werden als Kette aufgelistet, z.B. [-aeGhlz]. Das bedeutet nichts anderes, als daß eine beliebige Kombination dieser Optionen möglich ist (ob sie sinnvoll ist, wird hier nicht berücksichtigt).

UNIX-Benutzer sind "mündig"! Was heißt das? Wenn Sie ein Kommando eingeben, das die gesamte Platte löscht, fragt das Programm nicht noch einmal nach, ob Sie das auch wirklich wollen, sondern löscht die Platte sofort.

Beim Testen von Shell-Programmen hilft das Kommando

```
echo [Argumente]
```

Dieses Kommando gibt die Argumente auf dem Bildschirm aus. Für den Einsteiger ist das Kommando wichtig, weil er so die Kommandobearbeitung der Shell recht gut verfolgen und studieren kann (Einstreuen von echo-Kommandos in die Befehlsfolge).

Bearbeitung der Kommandozeile durch die Shell

Ein Kommando wird erst ausgeführt, wenn der Benutzer am Ende der Kommandozeile die RETURN-Taste drückt. Eine genauere Kenntniss des dann vonstatten gehenden Ablaufs erlaubt es, zu verstehen, warum etwas nicht so klappt, wie man es sich vorgestellt hat. Daher sollen diese Schritte hier kurz beschrieben werden. (Anmerkung: Einige der Kommandotrenner, `...`, Jokerzeichen und Variablen werden erst später behandelt.):

1. Die Shell liest bis zum ersten Kommandotrenner (& && || ; > <) und stellt fest, ob Variablenzuweisungen erfolgen sollen oder die Ein-Ausgabe umgelenkt werden muß/ztlig.
2. Die Shell zlegt die Kommandozeile in einzelne Argumente. Sie trennt die einzelnen Argumente durch eines der Zeichen, die in der Shell-Variablen IFS (Internal Field Separator) stehen, normalerweise Leerzeichen, Tabs und Newline-Zeichen.
3. Variablenreferenzen, die in der Kommandozeile stehen, werden durch ihre Werte ersetzt.
4. Kommandos, die in ` . . . ` oder bei der bash in \$ (. . .) stehen, werden ausgeführt und durch ihre Ausgabe ersetzt.
5. *stdin*, *stdout* und *stderr* werden auf ihre "Zieldateien" umgelenkt.
6. Falls in der Kommandozeile noch Zuweisungen an Variablen stehen, werden diese ausgeführt.
7. Die Shell sucht nach Jokerzeichen und ersetzt diese durch passende Dateinamen.
8. Die Shell führt das Kommando aus.

Login-Shell und Subshell

Eine Shell kann in zwei verschiedenen Modi laufen: als Login-Shell und als Subshell (Nicht-Login-Shell). Das Login-Programm setzt ein Flag, um der Shell mitzuteilen, dass sie eine Login-Shell ist. Eine Subshell wird auch durch den Aufruf eines Shell-Skripts, durch Aufruf des Shell-Kommandos (z.B. bash) oder durch den Aufruf einer Shell aus einem Anwendungsprogramm heraus) gestartet. Eine Subshell gibt keinen Prompt aus und hat normalerweise nur eine kurze

Lebensdauer. Die Login-Shell richtet Dinge wie z. B. den Terminal-Typ, Suchpfad-Kommandos usw. ein. Die einzelnen Shells haben verschiedene Mechanismen, beim ersten Shell-Aufruf bestimmte Tätigkeiten auszuführen. Dazu werden sogenannte Shell-Setupdateien abgearbeitet, die Kommandos und Definitionen enthalten.

Shell Setup-Dateien (z. B. ".profile") führen typischerweise mindestens folgende Tätigkeiten aus:

- Setzen/Erweitern des Suchpfades.
- Setzen des Terminaltyps und verschiedener Terminalparameter.
- Setzen von Umgebungsvariablen.
- Ausführen von Kommandos für Dinge, die bei jedem Login durchgeführt werden sollen.

Shell Setup-Dateien (Profile)

Hinweis: Bevor Sie eine Setup-Datei ändern, sollten Sie immer eine Sicherheitskopie von ihr erstellen, damit bei Problemen wieder auf das Original zurückgegangen werden kann und die Unterschiede zwischen der Originaldatei und der geänderten Version zum Finden von Fehlern untersucht werden können. Anstelle einer Backup-Datei kann man auch Zeilen auskommentieren anstatt sie zu löschen und neue Zeilen entsprechend markieren.

Je nachdem, welche Shell eingesetzt wird, gibt es unterschiedliche Setup-Dateien.

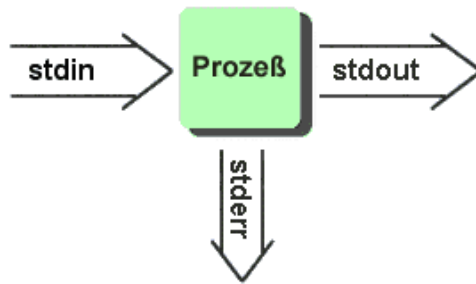
- Die Bourne-Shell liest beim Einloggen zwei Dateien ein, die für alle User global zuständige Datei "/etc/profile" und ".profile" im Heimatverzeichnis eines jeden Benutzers. Wird die Bourne-Shell als Subshell gestartet, übernimmt sie alle Setup-Infos aus den Umgebungs-Variablen, die beim ersten Login oder durch Eingabe von Kommandos gesetzt wurden.
- Die C-Shell verwendet drei Setup-Dateien: Die Datei ".cshrc" (C-Shell resource) wird bei jedem Start einer C-Shell gelesen. Die Datei ".login" wird bei jedem Start einer Login-Shell gelesen. Die Datei ".logout" wird ausgeführt, wenn eine Login-Shell beendet wird. Bitte beachten, dass beim Login ".cshrc" vor ".login" gelesen wird.

Die TC-Shell verhält sich bis auf eine Ausnahme wie die C-Shell. Lediglich, wenn die Datei ".tcshrc" im Home-Verzeichnis des Benutzers steht, wird sie anstelle von ".cshrc" ausgeführt.

- Die Korn-Shell entspricht fast der Bourne-Shell. Eine Login-Korn-Shell liest zuerst die Dateien "/etc/profile" und ".profile". Sie kann die Umgebungsvariable ENV (environment) auf den Pfadnamen einer Datei (meist ".kshrc") setzen. Jede Korn-Shell (auch jede Subshell) führt bei ihrem Start die durch \$ENV bezeichnete Datei aus.
- Die Bash (Bourne again shell) bearbeitet beim Login zuerst die Datei "/etc/profile" und dann eine der Dateien ".bash_profile", ".bash_login" oder ".profile" (die Suche erfolgt in dieser Reihenfolge). Eine Bash-Subshell führt dagegen die Datei ".bashrc" im Home-Verzeichnis des Benutzers aus. Beim Beenden einer Login-Bash wird die Datei ".bash_logout" ausgeführt. Siehe auch [Bash - Die Linux-Shell](#) weiter unten.

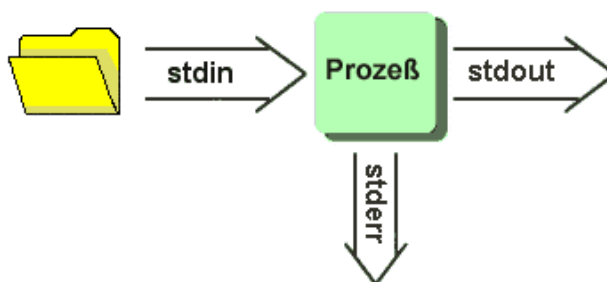
2.2 Ein-und Ausgabeumleitung

Die drei dem Terminal zugeordneten Dateikanäle *stdin*, *stdout* und *stderr* können jederzeit auf Dateien umgeleitet werden. Den drei Standarddateien sind die Filehandles 0 (*stdin*), 1 (*stdout*) und 2 (*stderr*) zugeordnet.



Eingabeumleitung

Das Programm liest nun nicht mehr von der Tastatur (*stdin*), sondern aus einer Datei, bis das Dateiende erreicht ist.



Die Eingabeumleitung erfolgt durch das Zeichen "<", gefolgt von einem Dateinamen.

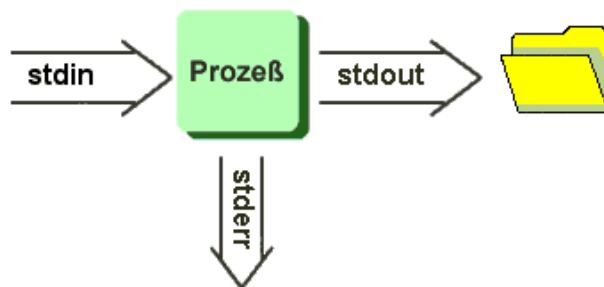
Kommando < Dateiname

Statt z. B. beim write-Kommando den Text direkt einzugeben, kann auch eine Datei an ein anderes Terminal gesendet werden:

```
write markus < Nachricht
```

Ausgabeumleitung

Die Ausgabe des Programms wird nicht auf dem Bildschirm (*stdout*) ausgegeben, sondern in eine Datei geschrieben. Die Ausgabeumleitung erfolgt durch das Zeichen ">", gefolgt von einem Dateinamen.



Falls die Datei noch nicht vorhanden war, wird sie automatisch angelegt. Falls die Datei schon vorhanden ist, wird sie überschrieben, d. h. es wird immer ab dem Dateianfang geschrieben.

Kommando > Dateiname

Fehlermeldungen (*stderr*) erscheinen nach wie vor auf dem Bildschirm. Beispiel: Ausgabe der Verzeichnisbelegung in einer Datei:

```
ls -l > info
```

Umlenkung der Fehlerausgabe (*stderr*)

Die Umleitung der Fehlerausgabe erfolgt genauso, wie die Ausgabeumleitung, jedoch wird hier die Zeichenfolge "2>" verwendet, da *stderr* die Handlungszahl 2 hat.

```
Kommando 2> Fehlerdatei
```

(Die Umleitung der Standardausgabe ist nur die Kurzform von Kommando 1> Dateiname). Natürlich ist eine beliebige Kombination von Ein- und Ausgabeumleitung möglich, z. B.

```
Kommando < Eingabedatei > Ausgabedatei 2> Fehlerdatei
```

Anhängen von Infos an eine Datei

Es ist auch möglich, die Ausgabe des Programms an eine bereits vorhandene Datei anzuhängen. Dazu wird das ">" doppelt geschrieben.

```
Kommando >> Sammeldatei
```

Dazu ein paar Beispiele:

Dateiliste und aktive Benutzer in eine Datei schreiben:

```
ls -l > liste  
who >> liste
```

Durch Umleitung von Ein- und Ausgabe lässt sich auch unterdrücken. Für die Ausgabe schreibt man

```
Kommando > /dev/null
```

oder für die Fehlerausgabe

```
Kommando 2> /dev/null.
```

Beides lässt sich auch kombinieren:

```
Kommando > Ergebnisdatei 2> /dev/null.
```

Will man ein Programm mit einem beliebigen Eingabedatenstrom versorgen, schreibt man

```
Kommando < /dev/zero.
```

Die Umleitung von *stdout* und *stderr* in dieselbe Datei würde prinzipiell eine zweimalige Angabe der Datei (eventuell mit einem langen Pfad) erfordern. Für die Standarddateien werden in solchen Fällen spezielle Platzhalter verwendet:

&0 Standardeingabe

&1 Standardausgabe

&2 Standard-Fehlerausgabe

Kommando > `ausgabe 2>&1`

Wenig bekannt ist der Bourne-Shell-Operator `<>`. Er öffnet eine Datei zum Lesen und Schreiben und verbindet sie mit der Standardeingabe. Fehlende Dateien legt er automatisch an, im Gegensatz zu `>` löscht er jedoch nicht den Inhalt bestehender Dateien. Gut eignet sich `<>` vor allem für den Zugriff auf Geräte, die eine bidirektionale Verbindung voraussetzen - etwa Terminals oder Modems.

Die Bourne-Shell kann noch mehr: Man kann einen beliebigen Kommunikationskanal umlenken, indem er dessen Kennzahl direkt vor das Größer oder Kleiner-Zeichen schreibt. Wie schon erwähnt, steht '0' für die Standardeingabe, '1' und '2' stehen für die Standard- beziehungsweise Fehlerausgabe.

Allgemein gilt also:

Mit den Bourne-Shell-Operatoren `<&` und `>&` lassen sich Ein- und Ausgabekanäle miteinander verbinden. Vor dem Operator darf die Nummer des umgeleiteten Kanals stehen, dahinter muß die des Quell- beziehungsweise Zielkanals folgen. Die häufigste Konstruktion `>datei 2>&1` leitet Standard- und Fehlerausgabe in dieselbe Datei um.

Michael Riepe beschreibt in *iX 10/2002*, im Artikel "Kreuz und quer - Ein- und Ausgabeumleitung in der Shell", S. 142 weitergehende Anwendungen von Umleitungen: Mehrere Umleitungen innerhalb eines Befehls führt die Shell nacheinander aus, und zwar in Schreibrichtung von links nach rechts. Stehen Umleitungen hinter einem zusammengesetzten Befehl, etwa einer Schleife, gelten sie für das gesamte Konstrukt; sie lassen sich jedoch innerhalb der Schleife durch weitere Umleitungen zeitweilig außer Kraft setzen. Es ist auch möglich, denselben Kanal in einem Befehl mehrmals umzuleiten. Dabei gilt, daß die jeweils zuletzt ausgeführte Ein- oder Ausgabeumleitung Vorrang hat. Das Kommando

```
cat <a <b >c >d
```

kopiert den Inhalt der Datei b in die Datei d. Als Seiteneffekt legt es eine leere Datei c an oder löscht den Inhalt von c, falls die Datei schon existierte.

Mitunter genügen die drei Standard-Kommunikationskanäle nicht, um eine Aufgabe zu erfüllen. Soll ein Skript etwa den Inhalt dreier Dateien miteinander vermengen, benötigt es zusätzliche Eingabekanäle. In der Bourne-Shell lassen sich weitere Kanäle mit den normalen Umleitungsoperatoren öffnen; der Benutzer muß lediglich die Nummer des gewünschten Kanals angeben. Mit den Umleitungsoperatoren `<&` und `>&` kann er die neuen Kanäle für einzelne Befehle öffnen, zum Beispiel:

```
# Dateimischer
while read X <&3 && read Y <&4 && read Z <&5
do
  echo $X $Y $Z
done 3<Datei-1 4<Datei-2 5<Datei-3
```

Die Anzahl der offenen Kanäle ist durch die Shell oder das Betriebssystem begrenzt. Portable Skripte dürfen nur die Kanäle 0 bis 9 verwenden. Bourne-Shell-Benutzer verwenden eine Spezialform des Befehls `exec`: Stehen hinter dem Befehlswort nur Umleitungen, aber keine Argumente, leitet die Shell die gewünschten Kanäle permanent um. Ist man zum Beispiel an Fehlermeldungen nicht interessiert, kann man sie vernichten mit

```
exec 2>/dev/null
```

Soll eine Umleitung später aufgehoben werden, kann man den ursprünglichen Kanal duplizieren:

```
exec 3<&0 <datei
```

verbindet beispielsweise die Standardeingabe mit einer Datei,

```
exec <&3
```

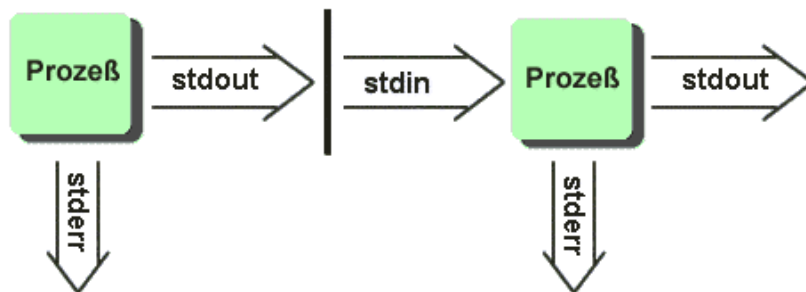
stellt die alte Verbindung wieder her. Der Ordnung halber schließt man mit `exec 3<&` den zusätzlichen Eingabekanal, wenn man ihn nicht mehr braucht. Analog dazu schließt der Operator `x>&` – einen Ausgabekanal.

2.3 Pipes

Eine Pipe verbindet zwei Kommandos über einen temporären Puffer, d. h. die Ausgabe vom ersten Programm wird als Eingabe vom zweiten Programm verwendet. Alles, was das erste Programm in den Puffer schreibt, wird in der gleichen Reihenfolge vom zweiten Programm gelesen. Pufferung und Synchronisation werden vom Betriebssystem vorgenommen. Der Ablauf beider Prozesse kann verschränkt erfolgen. In einer Kommandofolge können mehrere Pipes vorkommen. Der Pipe-Mechanismus wird durch das Zeichen "|" (senkrechter Strich) aktiviert:

```
Kommando 1 | Kommando 2
```

Beispiel: Ausgabe der Dateien eines Verzeichnisses mit der Möglichkeit, zu blättern:



Natürlich können auch mehrere Kommandos hintereinander durch Pipes verbunden werden:

```
Kommando 1 | Kommando 2 | Kommando 3 | Kommando 4 | ...
```

Pipelines haben Vorrang vor anderen Formen der Ein- und Ausgabeumleitung. Bevor die Shell mit der Ausführung der Einzelbefehle und der dazugehörigen Umleitungen beginnt, baut sie die gesamte Pipeline zusammen: Sie erzeugt für jeden Abschnitt einen neuen Prozeß und verbindet die Standardausgabe jedes Prozesses mit der Standardeingabe des Nächsten. Will der Anwender die Fehlerausgabe eines Programms ebenfalls durch die Pipeline schicken, kann er in Bourne-Shell-Skripten

```
foo 2>&1 | bar
```

schreiben. Da die Shell die Pipeline-Verbindung zuerst herstellt, landen die Fehlermeldungen vom Programm foo auf der Standardeingabe von bar.

Kommandofolgen, die durch Pipes verbunden sind, werden auch als "Filter" bezeichnet. Einige nützliche Filter sind in jedem UNIX-System verfügbar. Zum Beispiel:

```
head [-n] [datei(en)]
```

Ausgabe der ersten n Zeilen aus den angegebenen Dateien. Voreinstellung ist 10 Zeilen. Wird keine Datei angegeben, liest head von der Standardeingabe.

```
tail [-/+n] [bc[f|r]] [datei]
```

Ausgabe der letzten n Zeilen einer Datei. Voreinstellung für n ist 10. Wird keine Datei angegeben, liest tail von der Standardeingabe.

Optionen

+n ab der n. Zeile ausgeben

-n die letzten n Zeilen ausgeben Wird hinter die Zahl n ein 'b' gesetzt (z. B. -15b), werden nicht n Zeilen, sondern n Blöcke ausgegeben. Wird hinter die Zahl n ein 'c' gesetzt (z. B. -200c), werden nicht n Zeilen, sondern n Zeichen (characters) ausgegeben.

-r Zeilen in umgekehrter Reihenfolge ausgeben (letzte zuerst). Geht nicht bei GNU-tail - stattdessen kann man das Programm `toc` verwenden.

-f tail am Dateiende nicht beenden, sondern auf weitere Zeilen warten. (Ende des Kommandos mit der CTRL-C-Taste). Damit kann man z. B. Logfiles beobachten, die ständig wachsen.

```
tee [-i] [-a] [datei]
```

Pipe mit T-Stück: Kopiert von *stdin* nach *stdout* und schreibt die Daten gleichzeitig in die angegebene Datei.

Optionen

-i Ignorieren von Interrupts (Unterbrechungs-Taste)

-a Anhängen der Info an die angegebene Datei (Voreinstellung: Überschreiben der Datei)

```
wc [-lwc] [Datei(en)]
```

Dieses Kommando zählt Zeilen, Worte oder Zeichen in einer Datei. Wird kein Dateiname angegeben, liest wc von der Standardeingabe. Normalerweise zählt man damit in Skripten irgendwelche Ergebnisse. Optionen:

-l Zähle Zeilen

-w Zähle Worte

-c Zähle Zeichen

Weitere Filter sind `more`, `less`, `tr`,

2.4 Metazeichen zur Expansion von Dateinamen

Damit man beim Angeben von z. B. Dateinamen nicht alle Namen eintippen muß, sondern die Dateien auch alle oder nach bestimmten Kriterien auswählen kann, gibt es Metazeichen (Jokerzeichen,

Wildcards). Im Gegensatz zu anderen Systemen (z. B. MS-DOS) werden diese von der Shell ersetzt. Dies ist eine ganz wichtige Tatsache, die zur Folge hat, daß nahezu jedes UNIX-Kommando als Datei-angabe immer eine (im Rahmen der BS-Parameter) beliebige Menge von Dateien als Parameter haben kann. Im Programm sind daher auch keine Systemaufrufe nötig, die auf die Verzeichnisinformation zugreifen; es wird lediglich eine Schleife benötigt, welche die einzelnen Dateien nacheinander bearbeitet. Metazeichen sind Zeichen mit erweiterter Bedeutung. Die Shell ersetzt die Metazeichen durch alle Dateinamen des aktuellen Verzeichnisses, die auf das Muster passen. Dabei können die Metazeichen beliebig oft an beliebiger Stelle im Dateinamen stehen (z. B.: *abc*def*). Es gibt folgende Metazeichen:

*	Der Stern steht für eine beliebige Zeichenfolge - oder für überhaupt kein Zeichen. Dazu ein Beispiel: "ab*" steht für alle Dateinamen, die mit "ab" anfangen, auch für "ab" selbst ("ab", "abc", "abcd", "abxyz", usw.).
?	Das Fragezeichen steht für genau ein beliebiges Zeichen. Zum Beispiel: "?bc" steht für alle Dateinamen mit 3 Zeichen, die auf "bc" enden ("abc", "bbc", "1bc", "vbc", "xbc", usw.), nicht jedoch für "bc".
[]	Die eckige Klammer wird ersetzt durch eines der in der Klammer stehenden Zeichen. Auch ein Bereich ist möglich, z. B. [a-k] = [abcdefghijk]. Beispiel: "a[bcd]" wird ersetzt durch "ab", "ac" und "ad". Soll das Minuszeichen selbst in die Zeichenmenge aufgenommen werden, muß es an erster Stelle stehen (gleich nach der öffnenden Klammer).
[!]	Die eckige Klammer mit Ausrufezeichen wird ersetzt durch eines der nicht in der Klammer stehenden Zeichen, zum Beispiel: "[!abc]" wird ersetzt durch ein beliebiges Zeichen außer a, b oder c. Soll das Ausrufezeichen selbst in die Zeichenmenge aufgenommen werden, muß es an letzter Stelle stehen.
\	Der Backslash hebt den Ersetzungsmechanismus für das folgende Zeichen auf. Beispiel: "ab\?cd" wird zu "ab?cd" - das Fragezeichen wird übernommen. Wichtig: Bei der Umleitung von Ein- und Ausgabe werden Metazeichen in den Dateinamen hinter dem Umleitungszeichen nicht ersetzt.

Beispiele für die Anwendung:

```
ls -l a* listet alle Dateien, die mit "a" anfangen
```

```
ls test? listet alle Dateien die mit "test" anfangen und 5 Zeichen lang sind ("test1", "test2", "testa")
```

```
ls /dev/tty1[1-9] listet alle Terminalbezeichnungen mit einer 1 in der Zehnerstelle ("tty11", "tty12", ... , "tty19")
```

Lebenswichtig:

Der * ist ein gefährliches Zeichen, Tippfehler könne zum Fiasko führen, wenn aus Versehen ein Leerzeichen zuviel getippt wird.

```
rm a* löscht beispielsweise alle Dateien, die mit "a" anfangen.
```

```
rm a * löscht dagegen erst die Datei "a" und dann alle Dateien im Verzeichnis.
```

Anmerkungen:

- Der Punkt am Anfang von Dateinamen stellt eine Ausnahme dar, er muß explizit angegeben werden (wegen der Verzeichnisreferenzen "." bzw. ".." und der Tatsache, daß Dateien, die mit einem Punkt beginnen, normalerweise nicht angezeigt werden).
- Der "\ " am Zeilenende unterdrückt auch das Return-Zeichen - das Kommando kann in der folgenden Zeile fortgesetzt werden (es erscheint dann der Prompt ">" anstelle von "\$").

2.5 String-Ersetzungen (Quoting)

Um bestimmte Sonderzeichen (z. B. *, ?, [], Leerzeichen, Punkt) zu übergeben, ohne daß sie von der Shell durch Dateinamen ersetzt werden, werden Anführungszeichen verwendet, die auch ineinander geschachtelt werden können. Dabei haben die drei verschiedenen Anführungszeichen Doublequote ("), Quote (') und Backquote (`) unterschiedliche Bedeutung:

"	Keine Ersetzung der Metazeichen * ? [], jedoch Ersetzung von Shellvariablen (siehe unten) und Ersetzung durch die Ergebnisse von Kommandos (Backquote). Auch \ funktioniert weiterhin. Dazu ein Beispiel: <pre>echo Der * wird hier durch alle Dateinamen ersetzt echo "Der * wird hier nicht ersetzt"</pre>
'	Das einfache Anführungszeichen unterdrückt jede Substitution. Zum Beispiel: <pre>echo 'Weder * noch `pwd` werden ersetzt'</pre>
`	Zwischen Backquote (Accent Grave) gesetzte Kommandos werden ausgeführt und das Ergebnis wird dann als Parameter übergeben (d. h. die Ausgabe des Kommandos landet als Parameter in der Kommandozeile). Dabei werden Zeilenwechsel zu Leerzeichen. Braucht dieses Kommando Parameter, tritt die normale Parameterersetzung in Kraft. Zum Beispiel: <pre>echo "Aktuelles Verzeichnis: `pwd`"</pre> Weil die verschiedenen Quotes manchmal schwer zu unterscheiden sind, wurde bei der <i>bash</i> eine weitere Möglichkeit eingeführt. Statt in Backquotes wird die Kommandofolge in \$(...) eingeschlossen., z. B.: <pre>echo "Aktuelles Verzeichnis: \$(pwd)"</pre>

2.6 Bash - Die Linux-Shell

Die Bash (Bourne Again SHell) ist vollständig kompatibel zu der originalen Bourne-Shell aber sie hat etliche Erweiterungen erfahren. Sie ist bei fast allen Linux-Systemen die Standard-Shell.

Beim Start liest die Bash eine Reihe von Konfigurationsdateien ein, interpretiert deren Inhalt und übernimmt bestimmte Einstellungen. Was dabei genau geschieht, hängt von mehreren Faktoren ab.

Ist die Bash als Login-Shell eingestellt, wird zunächst die Login-Konfigurationsdatei *.bash_profile* im Home-Verzeichnis des Anwenders gesucht. Ist sie vorhanden, führt die Bash die darin enthaltenen Befehle aus. Fehlt diese Datei, sucht die Bash im selben Verzeichnis nach einer Konfigurationsdatei mit dem Namen *.bash_login*. Fehlt auch diese, wird versucht, die systemweit gültige Konfigurationsdatei */etc/profile* einzulesen und auszuwerten. Als letzte Möglichkeit versucht die Login-Bash, die Datei *.profile* im Home-Verzeichnis des Anwenders auszuführen.

Als interaktive (Sub-) Shell liest die Bash ihre Konfiguration aus der Datei *.bashrc* (im Home-Verzeichnis des Anwenders). Eine interaktive Bash verwendet für den Befehlszeileneditor zusätzlich die Einstellungen aus der Konfigurationsdatei *.inputrc*. Diese Datei wird zunächst im Home-Verzeichnis des Anwenders gesucht, ist sie dort nicht vorhanden, wird im Konfigurationsverzeichnis */etc/* nach der Datei *inputrc* gesucht.

Eine nicht interaktive (Sub-) Shell, wie sie zum Start von Skripten verwendet wird, liest keine Konfigurationsdateien ein, sofern dies nicht explizit gefordert wird.

Einige Versionen der Bash (z. B. die von SuSE Linux verwendete) suchen als interaktive Shell zunächst im Konfigurationsverzeichnis */etc/* nach systemweit gültigen Einstellungen in der Datei *bash.bashrc*. Diese ist aber normalerweise nicht vorhanden bzw. leer. Anschließend und unabhängig

davon, ob diese Datei gefunden wurde, wird die persönliche Konfigurationsdatei *.bashrc* im Home-Verzeichnis des Anwenders eingelesen.

Damit bei einer Login-Shell auch Funktionen einer interaktiven Shell zur Verfügung stehen, wird am Ende der Login-Konfigurationsdatei oft auch die persönliche Konfigurationsdatei im Home-Verzeichnis des Anwenders eingelesen.

SuSE Linux löst die Konfiguration der Bash in folgender Weise: Die systemweite Konfiguration erfolgt in der Datei */etc/profile*, an deren Ende weitere Konfigurationsdateien ausgeführt werden:

- */etc/SuSEconfig/profile*
- alle Skripte in */etc/profile.d*
- */etc/profile.local*

Auch beim Verlassen der Shell können von der Bash noch automatisch Befehle ausgeführt werden. Die Bash führt beim Eintreffen eines *exit*-Befehls die Datei *.bash_logout* im Home-Verzeichnis des Anwenders aus. Vorsicht: Nicht alle Versionen der Bash werten diese Datei korrekt aus.

Die Ausführung der Dateien läßt sich über zwei Kommandozeilen-Parameter steuern. Mit dem Parameter *-noprofile* veranlassen Sie, daß die Bash keine der oben genannten Startdateien ausführt, mit *-norc* erreichen Sie, daß die persönliche Konfigurationsdatei *~/ .bashrc* ignoriert wird.

Der Prompt

Die Bash gibt einen Eingabeprompt aus, häufig in der Form `Username: Pfad>`. Der Prompt ist über die Umgebungsvariable `PS1` konfigurierbar. Ein Prompt in der o. a. Form resultiert aus der Einstellung `PS1=\u:\w\$`.

Um andere Einstellungen auszuprobieren, müssen Sie die Variable neu belegen. Innerhalb dieser neuen Einstellungen können Sie einige spezielle Codes verwenden:

Codes für die Konstruktion des Prompts	
Code	Wirkung
<code>\a</code>	Das <i>Bell</i> -Zeichen, wie es durch die Tastenkombination [Ctrl][g] erzeugt wird
<code>\d</code>	das aktuelle Datum im Format <i>Thu Jan 18</i>
<code>\e</code>	das Escape-Zeichen
<code>\H</code>	der gesamte (Host) -Name
<code>\h</code>	der Rechner (Host) -Name bis zum ersten Punkt
<code>\n</code>	ein <i>Newline</i> (LineFeed) -Zeichen
<code>\r</code>	ein <i>Return</i> (Carriage Return) -Zeichen
<code>\s</code>	der Programmname der Bash, also <i>bash</i>
<code>\t</code>	die aktuelle Systemzeit im 24-Stundenformat <i>HH:MM:SS</i>
<code>\T</code>	die aktuelle Systemzeit im 12-Stundenformat <i>HH:MM:SS</i>
<code>\@</code>	die aktuelle Systemzeit im 12-Stundenformat mit <i>am/pm</i> (<i>01:39am</i>)
<code>\u</code>	der Username

<code>\v</code>	die Version der ausgeführten Bash (2.03)
<code>\V</code>	Das Release der Bash, bestehend aus der Versionsnummer und dem Patchlevel (2.03.1)
<code>\w</code>	das aktuelle Arbeitsverzeichnis in ausführlicher Darstellung, beispielsweise <code>~/LinuxMagazin/bash/teil6</code>
<code>\W</code>	der letzte Teil des aktuellen Verzeichnisses, etwa <code>teil6</code>
<code>\!</code>	die (History-) Nummer der aktuellen Befehlszeile
<code>\#</code>	die Nummer der Befehlszeile in der aktuellen Bash-Sitzung
<code>\\$</code>	mit diesem Schlüssen wird der Rootaccount gekennzeichnet. Wenn die effektive UID gleich Null ist, stellt die Bash das Hashmark dar, sonst ein Dollar-Zeichen
<code>\NNN</code>	Jedes beliebige ASCII-Zeichen kann durch Eingabe des oktalen Codes nach einem Backslash erzeugt werden
<code>\</code>	der Backslash selbst wird durch zwei Backslash-Zeichen erzeugt
<code>\</code>	eine Folge von Steuerzeichen wird so eingeleitet
<code>\</code>	die Folge von Steuerzeichen wird so beendet

Edieren der Kommandozeile

Die Hauptaufgabe einer Shell ist die Entgegennahme und Ausführung von Kommandos. Zum Bearbeiten der aktuellen Kommandozeile stehen sowohl Emacs- als auch vi-kompatible Editiermodi zur Verfügung - voreingestellt ist der Emacs-Modus. Wenn Sie lieber im vi-Modus arbeiten, stellen Sie ihn durch den Befehl `set -o vi` ein, zurück in den Emacs-Modus geht's mit `set -o emacs`. Im Emacs-Modus gibt es unter anderem folgende Edierfunktionen:

- `[Cursor links]` beziehungsweise `[Cursor rechts]` bewegen den Cursor in der Kommandozeile
- `[Esc]`, `[F]` springt bis zum nächsten Wortende
- `[Esc]`, `[B]` springt bis zum vorigen Wortanfang
- `[Strg]+[E]` springt zum Zeilenende
- `[Strg]+[A]` springt zum Zeilenanfang
- `[Strg]+[K]` löscht den Text ab der Cursorposition bis zum Ende der Zeile
- `[Strg]+[Y]` fügt den zuletzt gelöschten Text nach der Cursorposition ein
- `[Tab]` ergänzt ein Eingabefragment zum passenden Dateinamen. Ist die Ergänzung nicht eindeutig möglich, ertönt ein Signalton und Sie erhalten mit der Kombination
- `[Tab]`, `[Tab]` eine Liste der in Frage kommenden Dateinamen

Abhängig von der eingesetzten Terminalemulation können Sie Kombinationen mit der `[Esc]`-Taste oft auch mit der `[Alt]`-Taste nachgebildet. Statt also nacheinander `[Esc]` und `[F]` zu drücken, funktioniert meist auch die Kombination `[Alt]+[F]`.

History-Mechanismus

In der Bash können Sie nicht nur die aktuelle Kommandozeile editieren, sie merkt sich auch alle einmal eingegebenen Befehle in einer Datei, der Kommandozeilen-History. Auch diese Datei befindet sich im Home-Directory des Benutzers und heißt `.bash_history`. Mit den Cursor Tasten auf/ab kann man in dieser Liste blättern. Darüber hinaus stehen folgende History-Befehle zur Verfügung:

- `[Strg]+[R]` sucht nach dem letzten Kommando anhand der Anfangsbuchstaben
- `[Esc]` oder "Cursor links" springt an den Anfang der History-Liste

- [Esc] oder "Cursor rechts" springt ans Ende der History-Liste

Die Anzahl der Befehlszeilen wird mit der Variablen HISTSIZE eingestellt. Wächst die History darüber hinaus, verwirft die Bash die ältesten Zeilen.

Wichtige interne Kommandos

- `alias` weist einem Befehl oder einer Befehlsfolge einen neuen Namen zu. Beispiele für MS-DOS-Umsteiger sind:

```
alias dir='ls -l'
alias cd.='cd ..'
alias md='mkdir'
alias rd='rmdir'
alias del='rm -i'
```

Ohne Parameter gibt dieser Befehl eine Liste der aktuell definierten Aliasnamen aus. Zum Löschen eines Alias-Eintrags verwendet man den Befehl `unalias Name`.

- `fg`, `bg` und `jobs` sind Befehle für die interne Jobkontrolle der Bash. Ein laufendes Programm kann mit `[Strg]+[Z]` in den Hintergrund verschoben werden; dort wird es zunächst angehalten ("suspend"). Das Kommando `jobs` gibt eine Liste aller derzeit im Hintergrund schlafenden Programme aus. Dabei wird für jedes Programm die interne Jobnummer angegeben. Es ist wichtig anzumerken, daß die vergebene Jobnummer in keinem Zusammenhang zu der - etwa von `ps` - angezeigten Prozeßnummer steht. Mit `fg Jobnummer` holen Sie diese Task wieder in den Vordergrund. Soll ein angehaltenes Kommando dagegen im Hintergrund weiterlaufen, verwendet man den Befehl `bg Jobnummer`.
- `dirs`, `pushd`, `popd` verwalten den Stapel der internen Verzeichnisse. Mit `pushd Verzeichnis` wird das angegebene Verzeichnis auf dem Stapel abgelegt. Mit `popd` gelangen Sie anschließend zum zuletzt abgelegten Verzeichnis zurück. Den aktuellen Stapelinhalt zeigt der Befehl `dirs` an.
- `echo` wurde ebenfalls erweitert. Soll der Zeilenumbruch am Ende der Ausgabe unterdrückt werden, verwendet man den Parameter `-n`. Wer die von der C-Funktion `printf` bekannten Ausgabesteuerzeichen verwenden will, gibt den Parameter `-e` an.
- `hash` (ohne Parameter) zeigt die Liste der gemerkten Pfade zu den Programmen an. Um die Zugriffe auf Programme zu beschleunigen, verwaltet die Bash einen internen Cache der Pfade auf bereits gestartete Programme. Wird ein Programm erneut gestartet, kann die zeitaufwendige Suche entlang des Pfades entfallen. Bei der Gelegenheit wird auch gleich angezeigt, wie oft das Programm gestartet wurde. Wer am Abend wissen will, womit er sich den ganzen Tag über beschäftigt hat, kann mit diesem Kommando zumindest Hinweise bekommen (oder er wirft einen Blick auf die `.bash_history`).
`hash -r` verwirft alle gespeicherten Pfade.
- `help` ist die Hilfefunktion der Bash. Ohne Parameter aufgerufen, listet sie alle eingebauten Befehle auf. Mit einem Parameter wird ein Hilfetext über den angegebenen Befehl ausgegeben.
- `logout` führt das Skript `.bash-logout` aus und beendet die Login-Shell.

Mittels `set` einstellbare Optionen der Bash

Option	Name	Funktion
-a	allexport	neu definierte oder veränderte Variablen werden automatisch exportiert
-b	notify	bewirkt, dass Meldungen von Hintergrundjobs sofort ausgegeben werden (voreingestellt wartet die Bash bis zur Ausgabe des nächsten Prompts)
-B	braceexpand	Klammerexpandierungen erlauben (entspricht der Voreinstellung)

Betriebssystem UNIX/Linux

-C	noclobber	Setzen dieser Option verhindert, dass bestehende Dateien durch Ausgabeumleitungen (Redirections) zerstört werden
-e	errexit	In diesem Modus beendet sich die Shell immer dann automatisch, wenn ein Befehl einen Fehlercode erzeugte
-f	noglob	Deaktiviert die Kompletterungsfunktion für Dateinamen
-h	hashall	Deaktiviert das Speichern der Pfade bereits einmal ausgeführter externer Befehle; ein Abschalten bewirkt längere Ausführungszeiten bei Skripten
-H	histexpand	Erlaubt erweiterte Ersetzungen aus dem Historybuffer (voreingestellt: on)
-k	keyword	Zuweisungen werden in das Environment des Befehls übernommen
-m	monitor	Aktiviert die Job-Kontrollfunktionen (Voreinstellung: on bei interaktiven Shells)
-n	noexec	Verhindert die Ausführung von Befehlen; die Syntax wird aber überprüft und gegebenenfalls Fehlermeldungen erzeugt (deaktiviert bei interaktiven Shells)
-o	<i>Option</i>	setzt die im Argument übergebene Option, siehe das Beispiel oben
-p	privileged	Aktiviert den privilegierten Modus
-P	physical	Unterdrückt die Darstellung von symbolischen Links, statt dessen wird das physikalische Verzeichnis verwendet
-t	onecmd	Die Shell terminiert nach dem Ausführen des ersten Befehls
-u	nounset	Bewirkt, dass ungesetzt Variablen Fehlermeldungen erzeugen; ohne diese Option wird ihnen ein leerer Inhalt zugewiesen
-v	verbose	Befehlszeilen werden angezeigt, bevor sie ausgeführt werden
-x	xtrace	Alle Befehlszeilen werden mit expandierten Argumenten angezeigt, bevor sie ausgeführt werden

Mit der Einführung der Bash-Version 2.0 gibt es viele neue Konfigurations-Features. Sie werden durch den "erweiterten Konfigurationsbefehl" *shopt* gesetzt oder angezeigt. Die aktuellen Optionen sind über die Variable *SHELLOPTS* zugänglich. Der Befehl verfügt über folgende Optionen: *-p* (print) gibt eine Liste der aktuellen Einstellungen aus, *-s* (set) setzt die im Argument angegebene Option, *-u* (unset) löscht sie. Durch *-q* (quiet) wird die Ausgabe des Befehls unterdrückt, wie dies in Skripten praktisch sein kann. Durch *-o* beschränkt sich die Wirkung von *shopt* auf die durch *set -o* gesetzten Optionen. In der folgenden Tabelle sind nur die wichtigsten mittels *shopt* einstellbaren Bash-Optionen aufgeführt, eine vollständige Liste enthält die Manual-Page.

Option	Funktion
cdable_vars	bewirkt, dass die Bash Argumente des <i>cd</i> -Befehls als Variablen interpretiert, wenn es sich um keine Verzeichnisse handelt
cdspell	einfache Schreibfehler (vertauschte oder fehlende Buchstaben) in Verzeichnisnamen werden durch diese Option automatisch korrigiert
checkhash	die Bash sucht einen externen Befehl zunächst in der Hashtabelle, bevor er anhand des Suchpfades gefunden wird
checkwinsize	wenn diese Option aktiviert ist, prüft die Bash nach jedem ausgeführten Befehl, ob sich die Terminal-Abmessungen geändert haben.

<code>cmdhist</code>	zusammengehörige Befehlszeilen werden in der History in Form einer Zeile abgelegt, dadurch vereinfacht sich ihre Bearbeitung
<code>dotglob</code>	durch das Setzen dieser Option werden auch die mit einem Punkt beginnenden Dateinamen beim automatischen Kompletieren berücksichtigt
<code>execfail</code>	verhindert in Skripten, dass die Shell nach einem Fehler in einem <i>exec</i> -Befehl terminiert
<code>histexpand</code>	bewirkt, dass Historydateien nicht mehr überschrieben, sondern an bestehende Dateien angehängt werden
<code>lithist</code>	zusammen mit <code>cmdhist</code> bewirkt sie das Zusammenfassen mehrzeiliger Befehle
<code>sourcepath</code>	der <i>source</i> -Befehl kann auf die <code>PATH</code> -Variable zugreifen, wenn diese Option gesetzt ist
<code>expand_aliases</code>	erlaubt das Expandieren von Alias-Definitionen
<code>nocaseglob</code>	bei der automatischen Dateinamenkompletierung berücksichtigt die Bash Groß-/Kleinschreibung nicht, wenn diese Option aktiv ist
<code>huponexit</code>	allen von einer interaktiven Shell ausgeführten Hintergrundjobs wird ein <code>SIGHUP</code> -Signal gesendet, wenn sie terminiert
<code>restricted_shell</code>	die Bash wird im Restricted-Modus mit eingeschränkter Funktionalität betrieben

2.7 Schlußbemerkung

Es gibt zwei Möglichkeiten, einem UNIX-Kommando beim Aufruf Informationen mitzugeben:

- Auf der Kommandozeile
- Über die Standardeingabe

Für die letztere Möglichkeit gibt es das Kommando `cat Datei` (concatenate): Lesen der angegebenen Datei und Kopieren auf die Standardausgabe. Mit `cat Datei | Kommando` bekommt man den Dateininhalt in die Standardeingabe des Kommandos.

Auch für die Ausgabe gibt es zwei Möglichkeiten:

- Die Standardausgabe eines Programms kann durch die Backquotes (`` ``) oder durch `$ ()` als Parameter an ein Kommando übergeben werden.
- Umgekehrt erlaubt das Kommando `echo` die Ausgabe der Kommandoparameter auf der Standardausgabe oder per Umleitung in eine Datei oder Pipe.

Wichtig werden diese Erkenntnisse erst bei Erstellen von Shell-Skripten, wo von allen geschilderten Möglichkeiten Gebrauch gemacht wird.



[Zum Inhaltsverzeichnis](#)



[Zum nächsten Abschnitt](#)

Letzte Aktualisierung: 15. Sep 2011



Vorlesung "UNIX"

von Prof. Jürgen Plate

3. Einige Kommandos

In diesem Kapitel wird eine Reihe wichtiger UNIX-Kommandos besprochen. Wie so oft, liegt hier der Nutzen in der Beschränkung. Weitere Kommandos kann sich nach Kenntnis des "Grundwortschatzes" jeder selbst aneignen. Es gibt für fast jeden Zweck schon passende UNIX-Kommandos. Denken Sie daran, bevor Sie anfangen selbst etwas zu programmieren. Es werden hier auch nicht alle Kommandos aus der Vorlesung beschrieben. Zunächst aber ein etwas "seltsames" Kommando.

: (Doppelpunkt)

Der Doppelpunkt ist das leere Kommando der Unix-Shell. Es liefert immer "true" zurück und hat keine Wirkung. Es werden jedoch alle Shell-Ersetzungen der nach dem Doppelpunkt angegebenen Kommandos durchgeführt. Es eignet sich daher auch zum Testen von Shell-Skripts.

3.1 Kommandos zur Steuerung der Zugriffsberechtigung

Die Zugriffsberechtigungen wurde in Kapitel 1 besprochen; hier nun die Kommandos zum Setzen und Rücksetzen der Schutzbits. Die Bits UID und GID nehmen übrigens eine Sonderstellung ein - sobald eine Datei an einen anderen Benutzer übertragen wird, werden sie zur Sicherheit automatisch gelöscht. Bei den folgenden Kommandos erfolgen die Eingaben oktal (d. h. zur Basis 8). Das ist deshalb sinnvoll, weil es jeweils 3 Schutzbits für jede Benutzerklasse (Eigentümer, Gruppe, Andere) gibt, die sich zu einer Oktalstelle zusammenfassen lassen.

chmod Modus Dateiliste

Kommando zum Einstellen der Zugriffsrechte einer Datei, wobei zwei Modi existieren, symbolisch oder direkt. **Symbolischer Modus:** = Bereich Operand Berechtigung

Bereich	u Eigentümer (user) g Gruppe (group) o Übrige Benutzer (others) keine Angabe = ugo
Operand	+ Recht hinzufügen - Recht wegnehmen = Recht absolut setzen
Berechtigung	r Read w Write x eXecute s UID/GID (je nachdem, ob u+s oder g+s) t Sticky Bit (ohne Angabe von ugo)

Beispiele:

```
chmod +x           Ausführungsrecht für alle hinzufügen
chmod go-w dat2    Schreibrecht nur noch für User
chmod g+rx dat3    Volle Rechte für Gruppe
chmod ugo-rwx dat4 Allen alle Rechte entziehen
chmod +t verzl     Sticky-Bit von verzl setzen
chmod u+s admintool SUID-Bit setzen
```

Direktmodus: Modus = Oktalzahl (3 - 4 Stellen)

Für user, group und others wird jeweils eine Oktalzahl angegeben, welche die jeweiligen Rechte

wiedergibt. Dabei gilt folgende Wertigkeit:

read	4
write	2
execute	1

Falls die Zugriffsrechte SUID (4), SGID (2) und STICKY (1) vergeben werden sollen, kommt vorne noch eine vierte Stelle hinzu. Die Anwendung dieses Modus zeigen die folgenden Beispiele:

```

User  Group  Others
chmod 754 dat1      rwx   r-x   r--
chmod 740 dat2      rwx   r--   ---
chmod 440 wichtig   r--   r--   ---
chmod 000 geheim    ---   ---   ---
chmod 4711 systool  rws   --x   --x
    
```

umask Modus

Dieses Kommando vereinbart eine Standardeinstellung der Dateirechte für alle Dateien, die nach Aufruf von umask erzeugt werden - man braucht also nicht jedesmal des chmod-Kommando aufrufen, wenn eine Datei kreiert wurde. Auf bestehende Dateien hat das Kommando keinen Einfluß. Bedauerlicherweise ist der dreistellige Zahlenwert (für User, Group und Others) genau das Komplement der Angabe von chmod, d. h. hier wird festgelegt, welches Bit nicht gesetzt werden soll (für den Anfänger eine Pest!). Stellen Sie sich einfach vor, sie ziehen den Wert des umask-Modus vom Maximalwert 777 ab. Sie erhalten dann die Zugriffsrechte. Beispiele:

```

umask 022          7 7 7          rwx r-x r-x
                   - 0 2 2
                   = 7 5 5

umask 027          7 7 7          rwx r-x ---
                   - 0 2 7
                   = 7 5 0

umask 177          7 7 7          rwx --- ---
                   - 0 7 7
                   = 6 0 0
    
```

chown Username Dateiliste

Übertragen einer Datei an einen anderen Benutzer. Der Benutzer wird vollständiger Eigentümer der Datei. Der alte Eigentümer verliert sie. Beispiel:

chown markus dat1

Datei dat1 wird an markus übertragen

chgrp Gruppe Dateiliste

Ändern der Gruppenzugehörigkeit einer Datei. Es kann entweder die Gruppennummer oder der Gruppenname angegeben werden.

3.2 Dateidienste

cat [-usvte] datei [datei]

(concatenate) Lesen der angegebenen Dateien und Kopieren auf die Standardausgabe. Zusammenkopieren mehrerer Dateien oder Ansehen der Dateien. Optionen:

- e \$-Zeichen am Ende der Zeile ausgeben
- s (silent) Keine Fehlermeldung bei nicht existierenden Dateien
- t Tabulatoren als "^I" ausgeben
- v Nicht druckbare Zeichen als "^Z" ausgeben, wobei Z der ASCII-Wert des Zeichens + 64 ist (SOH = ^A, usw.)
- u ungepufferte Ausgabe

Beispiele:

cat dat1 gibt dat1 auf dem Bildschirm aus

cat dat1 dat2 > dat3 kopiert dat1+dat2 auf dat3

cd [directory]

(change directory) Wechsel des Arbeitsverzeichnis. cd ohne Parameter wechselt ins Home-Directory.

cp dat1 [dat2 ... datx] directory

(copy) Kopiert eine oder mehrere Dateien in das angegebene Directory. Existiert die Datei schon im Zieldirectory, wird sie überschrieben. Existiert sie noch nicht, wird sie neu angelegt. Dateien, die mit einem Punkt beginnen, müssen explizit angegeben werden ("cp * ziel" kopiert nur diejenigen Dateien, die nicht mit einem Punkt beginnen; "cp .* ziel" kopiert alle Dateien).

ln [-fs] dat.alt dat.neu

(link) Trägt für eine bereits existierende Datei "dat.alt" ein (Hard-)Link unter dem Namen "dat.neu" ein. Die Datei ist physikalisch nur einmal vorhanden, aber unter zwei Namen ansprechbar. Beim Dateinamen wird der Link-Count auf 2 gesetzt. Beide Directoryeinträge müssen auf dem gleichen Datenträger (= Dateisystem) liegen. Dateien mit mehreren Links werden erst beim Löschen des letzten Links physikalisch gelöscht. Optionen:

- f verhindert eine Nachfrage des Kommandos, falls die Datei schreibgeschützt ist.
- s erzeugt ein "Symbolisches Link". Hier wird eine Datei angelegt, die einen Verweis auf ihr Link enthält (Dateityp 'l'). Solche Links werden hauptsächlich für Verzeichnisse verwendet, sie wirken auch über die Grenzen von Dateisystemen hinweg.

Beispiele:

```
ln /usr/bin/ls /home/meier/bin/dir
ln -s /home/local /usr/local
```

mkdir Dirname

(make directory) Anlegen eines neuen Verzeichnisses, sofern der Schreibschutz des übergeordneten Verzeichnisses dies zuläßt. Voreinstellung für das Zugriffsrecht ist (umask) 755. Die Standardeinträge "." und ".." werden automatisch erzeugt. Die Zugriffsrechte lassen sich mit chmod ändern.

more

Wie cat eine weitere Möglichkeit zum Ansehen von Dateien. Jedoch stoppt hier die Ausgabe nach einer Bildschirmseite. Durch Tastendruck kann weitergeblättert oder abgebrochen werden. Außerdem kann auch im Text gesucht werden:

- Leertaste: Seitenweise weiterblättern
- Return-Taste: Zeilenweise weiter
- h: Hilfsmenü
- b: zurückblättern
- /: Suchen
- q: Beenden (quit)

Die Daten werden entweder über die Standardeingabe gelesen oder aus (einer) Datei(en), die als Parameter angegeben wird.

less

Wie more, jedoch erweiterter Befehlssatz und man kann auch bei Pipes zurückblättern. Siehe man-pages.

mv [-f] dat.alt dat.neu

(move) Umbenennen einer Datei unter Einbeziehung des gesamten Dateibaums. D. h. neben reiner Namensänderung auch ein Verschieben in ein anderes Verzeichnis möglich (Änderung der Verzeichniseinträge, kein Kopieren). Ist die Datei "dat.neu" bereits vorhanden, wird sie überschrieben. Liegt die Datei auf einem anderen Datenträger, wird sie physikalisch kopiert und an der alten Stelle gelöscht. Die Option -f unterdrückt eine Rückfrage bei Schreibschutz für dat.neu.

rm [-fri] Dateiliste

(remove) Löschen von Dateien und Verzeichnissen. Die angegebenen Dateien werden gelöscht. Optionen:

- i interaktiv: vor dem Löschen wird nachgefragt, ob die Datei gelöscht werden soll. Antwort "y" oder "n".
- f Löschen ohne Nachfrage bei schreibgeschützten Dateien
- r rekursives Löschen; es werden auch alle Dateien in darunterliegenden Verzeichnissen gelöscht (Diese Option ist gefährlich, wenn man nicht aufpasst!).

rmdir directory

Löschen eines leeren Verzeichnisses

stat [Optionen] Datei(en)

Status einer Datei oder eines Dateisystems anzeigen. Folgende Optionen sind möglich:

-L, --dereference	Verknüpfungen folgen
-f, --file-system	Dateisystemstatus anstelle eines Dateistatus anzeigen
-c	für die Anzeige das angegebene Format (siehe unten) verwenden. Nach jedem Eintrag wird ein Newline ausgegeben.
--format=FORMAT	
--printf=FORMAT	wie --format, aber es werden Sonderzeichen wie "\n" oder "\r" akzeptiert. Es wird KEIN Newline nach dem Eintrag ausgegeben (muss man im Format mittels "\n" selbst machen).
-t, --terse	Ausgabe erfolgt einzeilig in kompakter Form (ideal für die Weiterverarbeitung)
--help	Hilfe anzeigen
--version	Versionsinformation anzeigen

Die gültigen Formatangaben für Dateien sind:

%a	Zugriffsrechte im Oktalformat
%A	Zugriffsrechte in menschenlesbarer Form
%b	Anzahl der beanspruchten Blöcke (siehe %B)
%B	die Größe in Bytes jedes mit -%b- gemeldeten Blocks
%C	SELinux-Sicherheitskontext-Zeichenkette
%d	Gerätenummer dezimal
%D	Gerätenummer hexadezimal
%f	"raw"-Modus hexadezimal
%F	Dateityp
%g	Gruppen-ID des Eigners
%G	Gruppenname des Eigners
%i	Inode-Nummer
%h	Anzahl der Hardlinks
%n	Dateiname
%N	"Quoted File Name" mit Dereferenzierung bei symbolischen Links
%o	E/A-Blockgröße
%s	Gesamtgröße in Bytes
%t	Major-Gerätetyp hexadezimal
%T	Minor-Gerätetyp hexadezimal
%u	Nutzer-ID des Eigners
%U	Nutzername des Eigners
%x	Zeit des letzten Zugriffs
%X	Zeit des letzten Zugriffs als Timestamp
%y	Zeit der letzten Modifikation
%Y	Zeit der letzten Modifikation als Timestamp
%z	Zeit der letzten Änderung
%Z	Zeit der letzten Änderung als Timestamp

Die gültigen Formatangaben für Dateisysteme sind:

%a	Freie Blöcke, die Nicht-Superusern zur Verfügung stehen
%b	Gesamt-Datenblöcke im Dateisystem
%c	Gesamt-Inodes im Dateisystem
%d	Freie Inodes im Dateisystem
%f	Freie Blöcke im Dateisystem
%C	SELinux-Sicherheitskontext-Zeichenkette
%i	Dateisystem-ID in Hex
%l	Maximale Länge von Dateinamen
%n	Dateiname
%s	Optimale Transfer-Blockgröße

Betriebssystem UNIX/Linux

%S grundlegende Blockgröße (für Blockzahlen)
%t Typ hexadezimal
%T Typ in lesbarer Form

Die verwendete Shell besitzt möglicherweise eine eigene Version von stat mit anderen Optionen. Gegebenenfalls müssen Sie die Dokumentation der Shell zu Rate ziehen.

Beispiel:

```
stat .bash_history
  File: ".bash_history"
  Size: 4097          Blocks: 16          IO Block: 4096   reguläre Datei
Device: 801h/2049d   Inode: 245776       Links: 1
Access: (0600/-rw-----)  Uid: ( 777/   plate)   Gid: ( 777/   plate)
Access: 2011-09-19 17:27:09.000000000 +0200
Modify: 2011-09-19 15:57:28.000000000 +0200
Change: 2011-09-19 15:57:28.000000000 +0200
```

3.3 Steuerung des Druckers

lp [-cdmns] Dateiliste

Line Printer Spooler bei System V

Die angegebenen Dateien werden in die Druckerwarteschlange übergeben und dann vom Druckerspooler auf dem Drucker ausgegeben. Wird keine Datei angegeben liest lp von der Standardeingabe. Dem Druckauftrag wird eine eindeutige Auftragsnummer zugeteilt unter der er später angesprochen werden kann (siehe lpstat, cancel).

In der Regel wird nur ein Link auf die Datei(en) gesetzt. Will man sofort nach dem Aufruf von lp die Datei(en) ändern, muß eine Kopie erzeugt werden (siehe -c). Optionen:

- c Es wird eine Kopie im lp-Verzeichnis angelegt (kein Link)
- n# Es werden # Kopien ausgedruckt (z.B. -n4 für 4 Kopien) -w Nach Beendigung des Druckauftrags wird eine Meldung ans Terminal geschickt (wie bei write). Hat sich der Benutzer abgemeldet, erfolgt die Meldung per mail-Kommando.
- >-w w Nach Beendigung des Druckauftrags wird eine Meldung ans Terminal geschickt (wie bei write). Hat sich der Benutzer abgemeldet, erfolgt die Meldung per mail-Kommando.
- m Nach Beendigung des Druckauftrags Benachrichtigung des Benutzer per mail-Kommando.
- s Die Ausgabe der Auftragsnummer durch lp wird unterdrückt.
- d Explizite Angabe eines Druckers DRUCKER, falls sich mehrere Drucker im System DRUCKER befinden.

Beispiel: Ausgabe der Dateien vorles.kap1 und vorles.kap2 auf dem Drucker, Da die Datei weiter bearbeitet werden soll, wird eine Kopie erzeugt. Die Nachricht über Druckende soll per mail erfolgen:

```
lp -cm vorles.kap1 vorles.kap2
```

lpstat [-acdoprstuv] [Auftragsnummer]

(Line Printer Status) Gibt den Status der mit lp abgesetzten Druckaufträge aus. Fehlt die Auftragsnummer, wird der Status aller Druckaufträge des Benutzers ausgegeben. Bei den Optionen a,

c, o, p, u und v kann anschließend an die Option eine Liste von Druckern oder Druckerklassen angehängt werden. Die Information erfolgt dann nur zu den angegebenen Geräten. Fehlt die Liste, wird die Statusinformation zu allen Druckern ausgegeben. Optionen siehe man-pages. Beispiel:
lpstat -u gibt den Status aller Durckaufträge des Benutzers aus.

cancel [Auftragsnummer (n)] [Druckername (n)]

Abbrechen eines mit lp gestartete Druckauftrags. Angabe von entweder Auftragsnummer(n) oder Druckername(n). Wird ein Druckername angegeben, wird der gerade auf diesem Drucker bearbeitete Auftrag abgebrochen. Werden Auftragsnummern angegeben, können auch Aufträge aus der Warteschlange gelöscht werden.

disable Druckername

Stoppen des angegebenen Druckers (z. B. um Papier nachzulegen).

enable Druckername

Erneuter Start des Druckers.

lpr Dateiliste

BSD-Style Printer Spooler. Analog zu dem lp-Programm unter System V gibt es bei BSD das Programm lpr, um Dateien über den Spooler auszudrucken. Ohne weitere Optionen wird die Datei wieder zum Standarddrucker geschickt, z. B.: lpr text

Möchte man explizit angeben, auf welchem Drucker der Ausdruck erfolgen soll, verwendet man bei BSD die Option -P. Auch hier folgt direkt hinter dem "P" der Druckername, z. B.: lpr

-Plaserjet3 text

Auch unter BSD gibt es meist das Programm lpstat, um sich mit der Option -s die angeschlossenen Drucker anzeigen zu lassen. Falls das nicht geht, kann man sich die Datei /etc/printcap mit more anschauen. In dieser Datei sind alle angeschlossenen Drucker mit allen für den Drucker-Spooler wichtigen Einstellungen eingetragen. Abgesehen von den Namen, unter denen ein Drucker ansprechbar sein soll, und der Zuordnung zu einer bestimmten Schnittstelle, d. h. also einer Gerätedatei /dev/lp?, können hier auch Filterprogramme und Optionen für den Ausdruck angegeben werden. Die Einstellungen kann allerdings nur der Systemverwalter ändern.

Das mehrmalige Ausdrucken einer Datei erreicht man mit -# gefolgt von der Zahl der gewünschten Kopien. Wenn das UNIX-System nach erfolgtem Ausdruck eine entsprechende Bestätigung als Mail schicken soll, steht dafür die Option -m zur Verfügung.

Anders als bei System V kopiert das lpr-Programm unter BSD standardmäßig die auszudruckenden Dateien immer automatisch in den Pufferbereich des Drucker-Spoolers. Sollen nur Verweise angelegt werden, muß man die Option -s bei lpr angeben. Das ist vor allem dann sinnvoll, wenn eine Druckdatei sehr groß ist.

lprm

Druckaufträge wieder löschen. Es erwartet als Parameter die Kennnummer des Druckauftrags, der gelöscht werden soll. Leider gibt lpr beim Aufruf nicht die Kennnummer aus. Dazu muß man zunächst mit lpq nachschauen, welche Aufträge in der Warteschlange stehen.

lpq

Druckerwarteschlange auflisten. Zum Beispiel: lpq -Plaserjet3. Die Ausgabe sieht dann etwa so aus:

Betriebssystem UNIX/Linux

```
laserjet3 is ready and printing
Rank      Owner      Job      Files      Total Size
active    markus     3        bericht.ps 41984 bytes
lst       plate      17       brief      3213 bytes
```

Mit der Option -Plaserjet3 wird im Beispiel die Ausgabe von lpq auf den Laserdrucker eingeschränkt.

3.4 Informationsdienste

Einige Informationsdienste sind schon bekannt: who, ls, man, tty. Hier folgen einige weitere Kommandos, die den Benutzer mit Informationen versorgen:

date [mmddhhmm[yy]] [+format]

Beim Aufruf ohne Parameter werden Datum und Uhrzeit ausgegeben. Der Superuser kann mit diesem Kommando die Werte auch neu eingeben:

```
date mmddhhmmyy (Monat, Tag, Stunde, Minute, Jahr)
z. B.: date 10091245 (9.10., 12 Uhr 45)
```

Über den Parameter `+format` ("`+`"-Zeichen beachten!) kann die Ausgabe nach eigenen Wünschen gestaltet werden. Die Ausgabeformatierung erfolgt gemäß den Konventionen der Programmiersprache C (`printf`). Als Parameter werden verwendet:

<code>%m</code> : Monat (1 - 12)	<code>%D</code> : Datum im Format mm/dd/yy
<code>%d</code> : Tag (1 - 31)	<code>%H</code> : Stunde
<code>%y</code> : Jahr (00 - 99)	<code>%M</code> : Minute
<code>%n</code> : neue Zeile	<code>%S</code> : Sekunde
<code>%w</code> : Wochentag (0-6)	<code>%T</code> : Zeit im Format HH:MM:SS
<code>%a</code> : Wochentag (Sun-Sat)	<code>%h</code> : Monat (Jan-Dec)

Beispiel: Datum und Uhrzeit zweizeilig:

```
date '+Datum: %w, %d.%m.%y%nUhrzeit: %T'
```

Erstellungs- und Zugriffszeit/Datum von Dateien werden intern in Sekunden ab dem Offiziellen Geburtstag von UNIX, dem 1.1.1970, 0 Uhr UTC, gespeichert. Vor einiger Zeit habe ich mal das UNIX-Verfallsdatum ausgerechnet:

Auf 32-bit-Systemen wird `time_t` genau 2038-01-19 03:14:08 UTC ueberlaufen, d. h. dieses Datum hat die gleiche Darstellung wie 1970-01-01 00:00:00 UTC = (`time_t`) 0.

touch [-em] [mmddhhmm[yy]] Dateiliste

Setzt Datum und Uhrzeit der Datei. Parameter `-m` setzt die "modification time" (Voreinstellung), `-a` die "access time". Wird keine Zeit angegeben, nimmt touch die aktuelle Zeit (`mmddhhmmyy` = Monat, Tag, Stunde, Minute, Jahr).

df [-tfk] [Dateisystem] (Disk Free)

Liefert die Anzahl freie Blöcke und freier i-nodes des angegebenen Datenträgers. Fehlt die Angabe des Datenträgers, wird der aktuelle Datenträger verwendet. Optionen:

- t Gesamtkapazität wird mit angegeben
- f Die Angabe über i-nodes entfällt
- k Angabe in KByte statt in Blöcken

df /dev/dsk/0s1 Daten über einen bestimmte Platte

df -f /usr freie Blöcke des Dateisystems, das unter /usr eingehängt ist.

du [-arsk] [Dateiname]

(Disk Usage) Liefert die Anzahl der belegten Blöcke von Dateien und Verzeichnissen (fehlt der Dateiname, wird das aktuelle Directory untersucht). Bei Verzeichnissen werden die Blöcke der darin enthaltenen Dateien aufsummiert. Bei Link-Dateien wird nur der i-node gezählt. Es werden auch die indirekten Blöcke gezählt. Optionen:

- a (Voreinstellung) Daten für jede Datei ausgeben
- r Dateien melden, für die kein Leserecht existiert
- s Nur die Summe der belegten Blöcke ausgeben
- k Angabe in KByte statt in Blöcken

du -a dat1 Größe der Datei dat1 ausgeben

du -s /bin Summe der belegten Blöcke im Verzeichnis /bin

du -s /usr/* ausprobieren

file [-c] [-f Dateiname] [-m Dateiname] [Dateinamen]

(Determine File Type) Überprüfung der angegebenen Dateien auf ihren Inhalt (C-Programm, ASCII-Datei, ausführbares Programm, etc.). Um den Inhalt der Datei zu analysieren, verwendet das file-Kommando die Datei /etc/magic, in der Angaben über die Dateieigenschaften stehen (welche Bytes zu analysieren sind). Optionen:

- f Dateiname Die zu durchsuchenden Dateinamen stehen nicht in der Kommandozeile, sondern in der angegebenen Datei.
- m Die Unterscheidungskriterien stehen nicht in /etc/magic, sondern in der Dateiname angegebenen Datei.

find Pfadname(n) Bedingung(en) Aktion(en)

(Find Files) Durchsuchen der Platte nach bestimmten Dateien. Dieses Kommando ist sehr mächtig und

Betriebssystem UNIX/Linux

besitzt zahlreichen Optionen, welche die Bedingungen für die Dateiauswahl (= Treffer) festlegen. Die angegebenen Pfade werden rekursiv durchsucht, d.h. auch Unterverzeichnisse und Unter-Unterverzeichnisse, usw. Aufrufschema:

```
find Pfadname(n) Bedingung(en) Aktion(en)
```

Bei den nachfolgenden Optionen (= Bedingungen) steht das N für eine Zahlenangabe (ganze Zahl). Das Vorzeichen dieser Zahl bestimmt die Bedingung:

- N: genau N
- +N: mehr als N
- -N: weniger als N

Bedingungen

(Es werden nur die wichtigsten Optionen aufgeführt.):

`-name Dateiname`

Suche nach bestimmten Dateien (bei Verwendung von Metazeichen wie "*" oder "?" den Namen unbedingt in '.' einschließen).

`-type T`

Suche nach einem bestimmten Dateityp T:

f: normale Datei

d: Directory

b: Block Device

c: Character Device

p: Named Pipe

`-perm onum`

Suche nach Dateien mit den durch die Oktalzahl onum angegebenen Zugriffsrechten. Steht ein "-"-Zeichen vor onum, werden nicht alle, sondern nur die spezifizierten Rechte geprüft (z. B. Test SUID: `-perm -4000`).

`-links N`

Suche nach Dateien mit einer bestimmten Anzahl N von Links

`-user Kennung`

Suche nach Dateien eines bestimmten Users; es kann der Login-Name oder die UID angegeben werden.

`-group Kennung`

Wie `-user`, jedoch für Gruppen

`-size N`

Suche nach Dateien mit N Blöcken

`-mount`

Suche nur auf dem aktuellen Datenträger (wichtig, falls weitere Platten über NFS eingebunden sind).

`-newer Dateiname`

Suche nach Dateien, die jünger sind, als die angegebene.

Aktionen

(auch hier nur die wichtigsten):

`-print`

Ausgabe der gefundenen Dateinamen auf die Standardausgabe.

`-exec Kommando`

Ausführen eines Kommandos. Wird innerhalb des Kommandos die leere geschweifte Klammer `{}` aufgeführt, so wird anstelle der geschweiften Klammern der jeweils gefunden absolute Pfad der Datei eingesetzt. Das per `-exec` aufgerufene Kommando wird immer mit einem geschützten Strichpunkt (`\;` oder `;"` oder `;'`) abgeschlossen.

`-ok Kommando`

Wie `-exec`, jedoch mit Sicherheitsabfrage.

Beispiele:

```
find / -name '*.c' -print
```

Beginne beim Wurzelverzeichnis, alle C-Quellen (.c) zu suchen und gib die Namen (und Pfade) auf dem Bildschirm aus. Die Apostrophe verhindern die Ersetzung der Angabe `"*.c"` durch die Shell.

```
find -user markus -print
```

Suche alle Dateien von User markus.

```
find .-name dat1 -print
```

Suchen nach allen Vorkommen der Datei `dat1` ab dem aktuellen Verzeichnis.

```
find .-print -name dat1
```

Gibt *alle Dateinamen* ab dem aktuellen Verzeichnis aus. **Vorsicht Falle:** da die Bedingungen von links nach rechts ausgewertet werden und `-print` immer wahr ist, hat der Teil `-name dat1` keine Wirkung.

```
find . -name '*.bak' -exec rm {} \;
```

Suche alle `.bak`-Dateien ab dem aktuellen Verzeichnis und lösche sie.

```
find /usr -size +2000 -print
```

Gib alle Dateien der Benutzer mit mehr als 2000 Blöcken aus (z. B. um die Platzverschwender zu mahnen).

```
find . -name '*.bak' -ok rm {} \;
```

Suche alle `.bak`-Dateien ab dem aktuellen Verzeichnis und lösche sie nur, wenn die Nachfrage mit `"y"` beantwortet wurde.

```
find / -user markus -exec rm {} \;
```

Lösche alle Dateien von User markus, wo auch immer sie stehen.

```
find . -name 'buch.kap*' -exec lpr {} \;
```

Drucke alle Dateien eines Buch-Manuskripts. Verwendet man `xargs`, sieht das Kommando so aus:

```
find . -name 'buch.kap*' -print | xargs lpr
```

Betriebssystem UNIX/Linux

`xargs` (siehe unten) in Verbindung mit `find` kann die Effizienz des Programms erhöhen, da nicht für jede Datei ein neuer Prozeß gestartet werden muß. Aber es kann auch Gefahren in sich bergen. Stellen Sie sich vor, ein Bösewicht führt folgende Kommandos aus:

```
mkdir "/tmp/boese "  
mkdir "/tmp/boese /etc"  
touch "/tmp/boese /etc/passwd"
```

Er legt also - was er auch durchaus darf - zwei Verzeichnisse und eine Datei im allgemeinen Temporär-Verzeichnis an. Die Gemeinheit liegt darin, daß das Verzeichnis "boese" mit einem Leerzeichen endet. Bei `xargs` werden daraus möglicherweise auf der Kommandozeile zwei Verzeichnisse: `/tmp/boese` (ohne Leerzeichen am Schluß) und `/etc/passwd`. Führt nun der Superuser das Kommando

```
find /tmp -mtime +30 -print | xargs rm -f
```

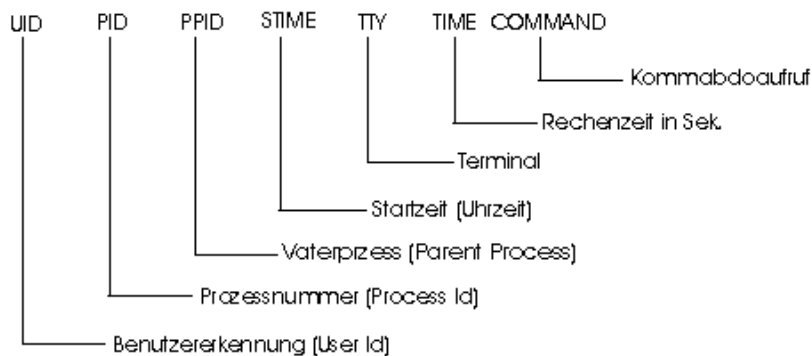
aus, um ältere Dateien zu löschen, schießt er sich die Passwortdatei weg. mit der Option `-exec` kann so etwas nicht passieren. Neuere Versionen der Programme verhindern diesen Fall.

ps [adeflnptu] (Report Process Status)

Liefert Statusinformationen über die aktiven Prozesse. Ohne Angabe von Optionen wird Information zu den Prozessen ausgegeben, die vom jeweiligen Terminal aus gestartet wurden. Zu jedem Prozeß wird eine Zeile ausgegeben, die Prozeßnummer, Terminalname, verbrauchte Rechenzeit und das Kommando angibt. Für den Alltag reichen zwei Optionen:

- e Informationen über alle laufenden Prozesse ausgeben
- f Vollständige Informationen über die Prozesse ausgeben.

Beispiel: `ps -ef`In jeder Zeile werden folgende Informationen ausgegeben:



Anmerkung: Bei Linux erreicht man Ähnliches mit `ps aux` (schlimmstenfalls muss man "man ps" eingeben).

3.5 Die Kommandos stty und tput

UNIX arbeitet terminalorientiert, d. h. der Benutzer sitzt an einen wie auch immer ausgestatteten Text-Terminal. Auch der PC-Bildschirm bei Linux ist ein solches Text-Terminal. Ebenso ist ein "xterm" auf dem Grafikbildschirm nichts anderes als ein emuliertes Text-Terminal. Sobald Kontakt zu einem Terminal hergestellt ist, greift die Terminalsteuerung auf zwei Dateien zurück: `/etc/ttydefs` oder `/etc/gettydefs`, in der hauptsächlich Steuerungswerte für die

Datenübertragung gespeichert sind (z. B. Baudrate, Datenformat, Handshake, etc.). Das ist natürlich nur noch dann interessant, wenn das Terminal über eine serielle Schnittstelle oder per Modem angeschlossen ist. Bei den Pseudo-Terminals, die über das Netz mit dem UNIX-Rechner verbunden sind, spielen die Inhalte der Dateien nur noch eine untergeordnete Rolle (z. B. bei der Definition einiger Steuertasten wie Backspace).

Beim Programmieren keimt recht schnell der Wunsch auf, den Bildschirm anzusteuern, z. B. den Bildschirm zu löschen, den Cursor zu positionieren etc. Es existieren verschiedene Möglichkeiten, dies zu realisieren. In den frühen Jahre von UNIX gab es zahlreiche Anbieter von Bildschirmterminals und jedes dieser Terminals wurde mit anderen Kombinationen von ASCII-Zeichen angesteuert. Selbst unter MS-DOS wurde diese Tradition noch gepflegt - in Form des ANSI-Treibers.

Welche Möglichkeiten der Bildschirmansteuerung gibt es? Ein Terminal reagiert normalerweise auf Zeichenfolgen, die vom Programm aus geschickt werden. Normale Zeichen werden als Buchstaben dargestellt, bestimmte Sonderzeichen oder Kombinationen bewirken aber bestimmte Funktionen. Da diese Kombinationen meist bestimmte Sonderzeichen enthalten (z.B. ESCAPE), um sie von Text zu unterscheiden, werden die Steuerfolgen auch Escape-Sequenzen genannt. Wenn man nun z. B. den Bildschirm löschen möchte, dann sucht man sich im Manual des Terminals die entsprechende Sequenz für "Bildschirm löschen" heraus und schickt sie vom Programm aus an das Terminal. Weil es auch nichtdruckbare Zeichen sein können, kann es sein, daß man ihre oktale oder sedezimale Verschlüsselung angeben muß. Dazu ein Beispiel:

Bei einem VT100-Terminal ist die Sequenz für "Bildschirm löschen" die Zeichenfolge

```
[ ;H[2J
```

Man schickt das beispielsweise aus einem C-Programm heraus mit `printf("\033[;H\033[2J")` ans Terminal. Dabei ist "\033" die oktale Schreibweise für das Escape-Zeichen.

Bei bildschirmorientierten Programmen wie z. B. einem Editor muß es möglich sein, den Cursor auf dem Bildschirm frei hin und her bewegen zu können. Das geht nicht ohne weiteres, weil die Terminals dafür spezielle Steuercodes, meist ESC gefolgt von einer bestimmten Zeichenkette, erwarten. Auch Fett- oder Kursivschrift lassen sich oft über solche Steuersequenzen ein- bzw. ausschalten. Andere Funktionen sind z. B. das Löschen des gesamten Bildschirms oder bis zum Ende der Zeile. Leider sind die Steuersequenzen für die einzelnen Funktionen nicht einheitlich. Auch haben die vielen verschiedenen Terminals unterschiedliche Möglichkeiten.

Deshalb gibt es eine Datenbank, in der, bei BSD-UNIX in der Datei `etc/termcap` im Textformat und bei System V in kompilierter Form in einer Reihe von Dateien unter dem Verzeichnis `/usr/lib/terminfo`, die Steuersequenzen zur Bildschirmansteuerung bei verschiedenen Terminaltypen gespeichert sind. Die Termcap und Terminfo-Datenbanken sind im wesentlichen äquivalent zueinander. Der Vorteil der Termcap-Variante ist die leichte Lesbarkeit und Änderbarkeit. Dafür kann auf die kompilierte Terminfo-Datenbank schneller zugegriffen werden.

Termcap

Je nachdem mit welchem Terminal man nun gerade arbeitet, sucht sich das UNIX-System die richtigen Steuersequenzen für dieses Terminal aus der Datenbank, wenn ein Anwendungsprogramm Funktionen wie "Bildschirm löschen" oder "Cursor nach oben" aufruft. Auf diese Art sind auch bildschirmorientierte Programme ohne Veränderung mit praktisch jedem Terminal einsatzfähig, vorausgesetzt natürlich, daß ein Termcap/Terminfo-Eintrag für das Terminal existiert. Im allgemeinen sollte das für alle gängigen Terminaltypen der Fall sein. Falls der Eintrag für ein verwendetes Terminal fehlt, können die Datenbanken auch um eigene Einträge ergänzt werden.

Betriebssystem UNIX/Linux

Der Termcap-Eintrag für das oft verwendete VT100-Terminal von DEC sieht z. B. folgendermaßen aus:

```
vt100|vt100-am|vt100am|dec vt100:\
:do=^J:co#80:]i#24:c1=50\l[;H\E[2J:sf=5\ED:\
:le=^H:bs:am:ce5\E[%i%d:%dH:nd=2\EEC:up=2\E[A:\
:ce=3\E[K:cd=50\E[J:so=2\E[7m:se=2\E[m:us=2\E[4m:ue=2\E[m:\
:md=2\E[lm:mr=2\E[7m:mb=2\E[5m:me=2\E[m:is=\E[1;24r\E[24;IH:\
:rf=/usr/share/lib/tabset/vt100:\
:rs=\E>\E[?31\E[?41\E[?51\E[?7h\E[?8h:ks=\E[?lh\E=:ke=\E[?11\E>:\
:ku=\EOA:kd=\EOB:kr=\EOC:kl=\EOD:kb=^H:\
:ho=\E[H:kl=\EOP:k2=\EOO:k3=\EOR:k4=\EOS:pt:sr=5\EM:vt#3:xn:\
:sc=\E7:rc=\E8:cs=\E[%i%d;%dr:
```

In der ersten Zeile stehen die Namen, unter denen dieser Termcap-Eintrag angesprochen werden kann. Danach folgen durch Doppelpunkte getrennt die zur Verfügung stehenden Funktionen und entsprechenden Steuersequenzen. Die Steuercodes, um den Cursor auf den Bildschirm bewegen, findet man beispielsweise in den ersten beiden Zeilen. `do = ^J` bedeutet, daß der Cursor mit CTRL-J eine Zeile nach unten, und `up=2\E[A`, daß er mit ESC-[A eine Zeile nach oben bewegt wird. Kombinationen mit der Control-Taste werden also in der Form `^x` angegeben, wobei `x` ein beliebiger Buchstabe sein kann. `\E` ist die Umschreibung für das Escape-Zeichen. Die Zahl 2 hinter dem Gleichheitszeichen bei der Angabe der Steuersequenz für die Bewegung des Cursors nach oben hat eine besondere Bedeutung. Grundsätzlich kann bei jeder Funktion an dieser Stelle eine Wartezeit in Millisekunden angegeben werden, die nach dem Abschicken der Steuersequenz verstreichen muß, bis der UNIX-Rechner weitere Zeichen an das Terminal schicken darf. Das ist notwendig, weil die Terminals für die Verarbeitung eines Steuercodes eine bestimmte Zeit brauchen, in der alle ankommenden Zeichen im - meist sehr kleinen - internen Puffer zwischengespeichert werden müssen, bis die Verarbeitung der Steuersequenz beendet ist. Bei einer schnellen Leitung zwischen Terminal und Computer kann es so zum Verlust von Zeichen kommen, weil der interne Puffer schnell überlaufen würde, wenn das Terminal beispielsweise damit beschäftigt ist, den Bildschirminhalt zu löschen. Dieses Warten nach dem Abschicken einer Steuersequenz an das Terminal nennt man englisch `Padding`.

Andere Angaben in dem Termcap-Eintrag sind die Zeilen- und Spaltenzahl des Terminals in der zweiten Zeile mit den Kürzeln `co#80` bzw. `li#24`. Sollte man tatsächlich einmal in die Verlegenheit kommen, einen Termcap-Eintrag verändern oder gar neu schreiben zu müssen, findet man weitere Informationen in den `man-pages` oder in weiterführender Literatur.

Auch in Programmen werden symbolische Bezeichnungen für die Terminalfunktionen verwendet, die in den entsprechenden Headerdateien der C-Bibliothek festgelegt sind. Zur Änderung der Werte in `ttydefs/gettydefs` dient das Kommando `stty`, zur Steuerung der Ausgabe kann das Kommando `tput` verwendet werden.

Terminfo

Mittlerweile gibt es einen Nachfolger für Termcap, die sogenannte Terminfo-Datenbank. Sie ist ziemlich ähnlich, nur daß die Datenbank hier nicht im ASCII-Format abgespeichert ist, sondern in einem Binärformat, weil das geht. Im Normalfall haben die meisten verwendeten Unix-Varianten noch beides, `termcap` und `terminfo`.

Curses

Mit `termcap` kann man flexible Programme schreiben, aber es könnte noch einfacher gehen. Deshalb gibt es eine weitere Bibliothek, die auf `termcap` aufbaut: "`curses`". Die `curses`-Bibliothek ermöglicht sämtliche Terminal-E/A-Funktionen. Zusätzliches Ziel von "`curses`" ist es, den Update des Bildschirms in optimaler Form durchzuführen, also möglichst schnell mit möglichst wenig Zeichen,

die geschickt werden müssen.

Curses geht dabei folgendermaßen vor: es gibt einen virtuellen Bildschirm im Hauptspeicher. Ausgaben werden nun zuerst in diesem internen Speicherbereich aufgebaut. Und erst beim Aufruf der Funktion `refresh()` werden alle seit dem letzten `refresh()` angefallenen Änderungen an den realen Bildschirm geschickt. Zusätzlich ist es möglich, den Bildschirm in Bereiche (Windows, hat aber nichts mit X Windows zu tun) einzuteilen und diese getrennt zu bearbeiten. Am Anfang gibt es für den gesamten zur Verfügung stehenden Bildschirm das vordefinierte Window `stdscr`.

Wie sieht ein curses-Programm aus? Hier ein Beispielprogramm, das in der Mitte des Bildschirms den String "Hello, world!" invers ausgibt.

```
#include

int main(void)
{
    char *text = "Hello, world!";

    initscr();           /* Curses-Paket initialisieren */
    clear();             /* Bildschirm löschen */
    move(LINES/2-1, (COLS-strlen(text))/2-1);
                        /* Cursor zur richtigen Position */
                        /* LINES und COLS sind vordefiniert */
    standout();         /* schalte auf Invers */
    printw("%s\n",text); /* gib den Text aus */
    standend();         /* wieder auf normal */
    refresh();          /* reale Ausgabe */
    endwin();           /* Ende Curses-Nutzung */
    return 0;
}
```

Die Verwendung von Bibliotheken wie `termcap` oder `curses` hat natürlich Auswirkungen auf die Portabilität des Programms. Schon verschiedene Unix-Systeme haben verschiedene Curses-Bibliotheken.

stty [Optionen]

(set tty) Änderung der Standard-tty-Parameter. Die Einstellungen sind nur solange gültig, wie der Shellprozeß mit dem Terminal verbunden ist. Das Kommando kann alleine oder mit diversen Parametern verwendet werden. `stty` ohne Parameter gibt die wichtigsten Einstellungen aus; um alle Parameter zu sehen, verwendet man `stty -a`. Meist kann man die Wirkung der entsprechenden Option durch ein "-"-Zeichen davor umkehren (siehe Beispiel `stty -echo`). Weitere Info: man `stty`.
Beispiele:

```
stty erase ^h
```

Setzen der Backspace-Funktion (in diesem Fall Control-H). Bei direkter Eingabe kann man die entsprechende Taste direkt betätigen (Steuertasten werden als zwei Zeichen "geecho", Backspace z. B. als "^h"). Bei Eingabe in Skripten mit dem `vi` kann man mit `Ctrl-D` erreichen, daß das folgende Zeichen unverändert übernommen wird. Manche Shells akzeptieren auch die Kombination aus zwei Zeichen, "^" + Buchstabe.

```
stty eb
```

Einschalten der akustischen Fehleranzeige (error bell).

```
stty echo stty -echo
```

Echo auf dem Bildschirm ein- und ausschalten. Bei der folgenden Kommandosequenz wird die Eingabe des Paßworts auf dem Bildschirm nicht wiedergegeben.

```
echo "Paßwort eingeben: \c"
stty -echo
read PASS
stty echo
```

tput Parameter

(terminal put) Senden von Steuerbefehlen an das Terminal, wobei die "echte" Steuerinfo der terminfo-Datenbasis entnommen wird. Als Parameter dient der symbolische Name der entsprechenden Steuerfunktion. Je nach Terminaltyp sind sehr viele Steuerfunktionen möglich. Derzeit akzeptiert tput nur einen Parameter, mehrere Steuerbefehle müssen also nacheinander gegeben werden. Die in der Praxis am häufigsten gebrauchten Parameter sind:

bel	Signalton
blink	Text auf Blinken schalten
bold	Text heller darstellen (hohe intensität)
clear	Bildschirm löschen,Cursor in die linke obere Ecke
cr	Carriage Return
cup	Cursor positionieren. Hinter cup folgen Zeilennummer und Spaltennummer der Cursorposition, z. B. cup 10 40
ed	Bildschirm von Cursorposition bis Ende löschen
el	Von Cursorposition bis Zeilenende löschen
home	Cursor in linke obere Ecke
ind	Schirm nach oben rollen
rev	Text auf Inversdarstellung schalten
ri	Schirm nach unten rollen rmso Wieder auf normale Zeichendarstellung schalten

Beispiel:

```
tput bel
tput bold
echo "\nAchtung! Alle Daten werden gelöscht!\n"
tput rmso
```

3.6 Weitere Kommandos

xargs Programm [Parameter]

Das xargs-Programm fällt etwas aus dem Rahmen der übrigen in diesem Kapitel behandelten Programme. xargs übergibt alle aus der Standardeingabe gelesenen Daten einem Programm als zusätzliche Argumente. Der Programmaufruf wird einfach als Parameter von xargs angegeben. Ein Beispiel soll die Funktionsweise klarmachen:

```
$ ls *.txt | xargs echo Textdateien:
```

Hier erzeugt ls eine Liste aller Dateien mit der Endung .txt im aktuellen Verzeichnis. Das Ergebnis wird über die Pipe an xargs weitergereicht. xargs ruft echo mit den Dateinamen von ls als zusätzliche Parameter auf. Der Output ist dann:

Betriebssystem UNIX/Linux

Textdateien: kap1.txt kap2.txt kap3.txt h.txt

Durch Optionen ist es möglich, die Art der Umwandlung der Eingabe in Argumente durch xargs zu beeinflussen. Mit der Option `-n <Nummer>` wird eingestellt, mit wievielen Parametern das angegebene Programm aufgerufen werden soll. Fehlt der Parameter, nimmt xargs die maximal mögliche Zahl von Parametern. Je nach Anzahl der Parameter ruft xargs das angegebene Programm einmal oder mehrmal auf. Dazu ein Beispiel. Vergleich einer Reihe von Dateien nacheinander mit einer vorgegebenen Datei:

```
ls *.dat | xargs -n1 cmp compare Muster
```

Die Vergleichsdatei "Muster" wird der Reihe nach mittels `cmp` mit allen Dateien verglichen, die auf ".dat" enden. Die Option `-n1` veranlaßt xargs, je Aufruf immer nur einen Dateinamen als zusätzliches Argument bei `cmp` anzufügen.

Mit der Option `-i <Zeichen>` ist es möglich, an einer beliebigen Stelle im Programmaufruf, auch mehrmals, anzugeben, wo die eingelesenen Argumente einzusetzen sind. In diesem Modus liest xargs jeweils ein Argument aus der Standardeingabe, ersetzt im Programmaufruf jedes Vorkommen des hinter `-i` angegebenen Zeichens durch dieses Argument und startet das Programm. In dem folgenden Beispiel wird das benutzt, um alle Dateien mit der Endung ".txt" in ".bak" umzubenennen.

```
ls *.txt | cut -d. f1 | xargs -iP mv P.txt P.bak
```

Das Ganze funktioniert allerdings nur, wenn die Dateien nicht noch weitere Punkte im Dateinamen haben.

wc [-lwc] [Datei(en)]

Dieses Kommando zählt Zeilen, Worte oder Zeichen in einer Datei. Wird kein Dateiname angegeben, liest `wc` von der Standardeingabe. Optionen:

- l Zähle Zeilen
- w Zähle Worte
- c Zähle Zeichen

pr [+n] [-h Header] [-ftF] [-wü] [-oü] [-lü] [Datei(en)]

(print) Ausgeben und Formatieren von Dateien. Ist keine Datei angegeben, wird von der Standardeingabe gelesen. Optionen:

- +n Beginne bei Seite n der Dateien (Default: 1)
- h Definiere Seitenüberschrift (Default: Dateiname)
Kopf
- f Verwende beim Seitenwechsel das Formfeed-Zeichen anstelle von Leerzeichen
- t Kopf- und Fußzeilen unterdrücken
- F Zu lange Zeilen nicht abschneiden sondern umbrechen
- w# Zeilenbreite festlegen (Default: 72)
- o# ü führende Leerzeichen vor jede Zeile setzen

-# Seitenlänge festlegen (Default: 66). Davon gehen ab 5 Zeilen für Header und 2 Zeilen für Seiten-Nummer am Ende der Seite.

sleep zeit

Legt einen Prozeß für zeit Sekunden schlafen. Erst nach Ablauf dieser Zeit wird er wieder als "bereit" in die Warteschlange eingereiht. Anwendung in der Regel in Kommandoprozeduren (Shellskripts).

tr Ersetzungsliste

tr ist ein typischer Filter (tr = translate). Es wird benutzt, um in einer Datei bestimmte Zeichen durch andere zu ersetzen. tr liest immer von der Standardeingabe und schreibt auf die Standardausgabe. Als Parameter gibt man bei tr an, welche Zeichen durch welche anderen zu ersetzen sind. Die einfachste Variante ist, ein bestimmtes Zeichen in ein anderes umgewandelt werden soll. Dazu gibt man zunächst das zu ersetzende Zeichen und danach - durch Leerzeichen getrennt - das neue Zeichen als Parameter an. Beispiel:

```
tr : . < foo Ersetzt ":" durch ".".  
tr abc def < foo Ersetzt "a" durch "d", "b" durch "e" und "c" durch "f".  
tr a-z A-Z < foo Ersetzt Kleinbuchstaben durch Großbuchstaben.
```

Leider unterscheiden sich die Versionen von tr auf System V und BSD-UNIX-Varianten etwas, was die Verarbeitung dieser Zeichenfolgen anbetrifft. Bei System V müssen die Bereichs- und Wiederholungsangaben in eckigen Mammern eingeschlossen sein. Bei BSD gibt es nur Bereichsangaben, allerdings ohne Klammern. Dafür wird das letzte Zeichen der zweiten Zeichenfolge bei der ESD-Variante automatisch so oft wiederholt, bis die zweite Zeichenfolge genauso lang ist wie die erste.

cut -cSpalten [dateien]

```
cut -fFelder [-dZeichen] [-s] [dateien]
```

Extrahieren bestimmter Spalten aus einer Datei. Werden keine Dateien angegeben, liest cut von der Standardeingabe. Die Ausgabe erfolgt auf die Standardausgabe. Optionen:

- cSpalten Die angegebenen Spalten werden herausgenommen, dabei sind mehrere Zahlenangaben, getrennt durch Kommata möglich, ebenso Bereiche, z.B. 5-9.
- c-10,15,20,30- 1. bis 10. Spalte, Spalte 15, Spalte 20 und alles ab Spalte 30
- fFelder Angaben von Feldern durch die Feldnummern analog -c. Als Feldtrenner ist das Tabzeichen vorgegeben.
- dZeichen Vorgabe des Feldtrenners. Das Zeichen wird als Feldtrenner verwendet. z. B. -c:
- s Zeilen, in denen der Feldtrenner vorkommt, nicht ausgeben (Voreinstellung: Zeile ausgeben)

cal [[monat] jahr]

Kalender zu einem Monat oder Jahr auf der Standardausgabe ausgeben.

- ohne Parameter: aktueller Monat
- mit einem Parameter (jahr): Jahreskalender (1-9999)
- Mit 2 Parametern (monat): Monat (1-12) des angegebenen Jahres (1-9999)

calendar

Minimalversion eines Terminkalenders. Sucht im aktuellen Verzeichnis nach einer Textdatei namens "calendar" und gibt alle Zeilen aus, die irgendwo das aktuelle oder morgige Datum enthalten. Datumsangabe amerikanisch: Monat, Tag (z. B. Jan 24, 1/25, January 25). Je nach Installation auch Funktion von cal implementiert.

basename string [suffix]

Extrahieren des Dateinamens aus einer Pfadangabe, die als String übergeben wird. Wird ein Suffix angegeben, wird dieser am Ende des Dateinamens entfernt. Zum Beispiel:

basename /home/plate/dings.c	dings.c
basename /home/plate/dings	dings
basename \$0	Programmname

dirname

Extrahieren des Pfadnamens aus einer Pfadangabe, die als String übergeben wird. Beispiel: `dirname /home/plate/dings.c` ergibt `/home/plate` und `dirname dingsda` ergibt `.`

logname

Gibt den Loginnamen auf der Standardausgabe aus. Im Gegensatz zum Kommando `id`, das immer den aktuellen Loginnamen ausgibt, wird hier der Name des Login geliefert, auch wenn mit `su` die Benutzeridentität gewechselt wurde.

time [kommando]

Berechnet die Zeit, die das angegebene Kommando zur Ausführung braucht. Es werden drei Zeiten ausgegeben:

- real Insgesamt verbrauchte Zeit (elapsed time)
- user Im User-Modus verbrauchte Zeit
- sys Im System-Modus verbrauchte Zeit (Systemdienste)

bc [-lmath]

Einfacher Taschenrechner hoher Genauigkeit. Um mit Gleitpunktzahlen zu rechnen, muß der Parameter angegeben werden. Liest Formeln von der Standardeingabe und gibt das Ergebnis auf der Standardausgabe aus. End mit EOF = CTRL-D.

last [-Zahl] [-f datei] [namen]

Ausgabe der An- und Abmeldezeiten von Usern. Optionen:

- Zahl Anzahl der Zeilen, die ausgegeben werden sollen
- f Datei Liest aus der Datei statt aus `/var/adm/wtmpx`

namen Benutzer- oder Terminalnamen

script [-a] [datei]

script erlaubt es, eine Terminal-Sitzung (oder einen Teil davon) auf einer Datei aufzuzeichnen. Dabei wird alles, was in dieser Zeit auf der alphanumerischen Dialogstation (oder dem xterm-Fenster bei graphischen Oberflächen) erscheint, festgehalten - auch das Eingabeecho. Ist 'datei' im Aufruf angegeben, so wird die Sitzung in dieser Datei protokolliert. Fehlt die Angabe einer Ausgabedatei, so wird in die Datei `typescript` des aktuellen Katalogs geschrieben. Ctrl-C (Unterbrechung) beendet die Protokollierung. Die Option `-a` veranlaßt script, das Protokoll an die Datei `datei` (oder `typescript`) anzuhängen, anstatt eine neue Datei zu eröffnen. script ist immer dann hilfreich, wenn alle Aktivitäten einer längeren Sitzung dauerhaft protokolliert werden sollen: z. B. bei der Fehlersuche, bei einer längeren Netzwerk-Sitzung auf einem entfernten Rechner oder beim Ausprobieren eines chat-skripts für eine PPP-Verbindung. Man kann auch eine Kommandofolge interaktiv ausprobieren und dann die richtigen Kommandozeilen aus der Protokolldatei per Editor schnell in ein Shellskript umsetzen.

3.7 Exkurs über Parameterersetzung

Es wurden mehrere Möglichkeiten vorgestellt, Parameter für ein Kommando zu erzeugen. Am Kommando `find` sollen alle Möglichkeiten kritisch betrachtet werden. Es geht in allen Fällen darum, bestimmte Dateien (*.txt) in allen Verzeichnissen zu suchen und die gefundenen Dateien auszudrucken.

1. `lpr `find / -name '*.txt' -print` /home/texte`
Zuerst wird das `find`-Kommando ausgeführt und dessen Output bildet die Parameterliste des `lpr`-Kommandos. Nachteil: Das geht nur solange gut, solange nicht die systemspezifische Länge der Kommandozeile überschritten wird.
2. `find / -name '*.txt' -exec lpr {} /home/texte \;`
Für jede gefundene Datei wird das `lpr`-Kommando aufgerufen. Vorteil: Klappt immer. Nachteil: `lpr` wird u. U. sehr oft aufgerufen.
3. `find / -name '*.txt' -print | xargs lpr`
Das Ergebnis von `find` wird unter Berücksichtigung der Systembegrenzungen an `lpr` übergeben. Effizienteste Lösung.



[Zum Inhaltsverzeichnis](#)



[Zum nächsten Abschnitt](#)

Copyright © FH München, FB 04, Prof. Jürgen Plate

Letzte Aktualisierung: 20. Sep 2011



Vorlesung "UNIX"

von Prof. Jürgen Plate

4. Kommandos zur Prozeß- und Systemverwaltung

Bei UNIX muß man Prozessen, die vom Terminal aus gestartet wurden, nicht unbedingt warten, bis der Prozeß terminiert, bis man das nächste Programm starten kann. Der Prozeß kann vielmehr im Hintergrund ablaufen, während sich der Benutzer anderen Arbeiten zuwendet. Mit dem Druckerspooler haben wir schon einen Hintergrundprozeß kennengelernt - wenn auch von spezieller Art.

Jetzt erscheint auch die Notwendigkeit des ps-Kommandos unter anderem Licht - man kann sich so über laufende Hintergrundprozesse informieren. Bei neueren Versionen greift das ps-Kommando nicht mehr auf die Prozeßtabelle des Schedulers zu, sondern es gibt ein Pseudo-Dateisystem /proc, in dem alle Infos über die Prozesse abgelegt sind. In der Regel werden Hintergrundprozesse abgebrochen, wenn sich der Benutzer am Terminal abmeldet (Logoff), denn sie sind ja Kindprozesse des aktuellen Shell-Prozesses. Es gibt jedoch Möglichkeiten, Prozesse auch nach dem Logoff "am Leben" zu erhalten. UNIX kennt recht mächtige Möglichkeiten für Hintergrundprozesse:

- Der "normale" Hintergrundprozeß
- Hintergrundprozeß mit reduzierter Priorität
- Batch-Prozesse (ohne Terminalanbindung)
- Zeitlich versetzte Prozesse (Start zum festgelegten Zeitpunkt)
- Sich regelmäßig wiederholende Prozesse

4.1 Hintergrundprozesse

Der Start eines Prozesses im Hintergrund erfolgt durch anhängen eines "&" an die Befehlszeile. Alle Kommandoeingaben bleiben wie vorher - auch der Ersetzungsmechanismus der Shell wirkt nach wie vor. Ebenso möglich sind Pipes. Zum Beispiel:

```
find / -user markus -print &  
2354
```

Es wird die Nummer (PID) des neu erzeugten Prozesses ausgegeben (2354) und die Shell meldet sich sofort wieder mit dem Eingabeprompt. So wie das Kommando oben eingegeben wurde, hat es aber noch einen Nachteil:

Alle Ausgaben von find, auch die Fehlerausgaben, gelangen nach wie vor auf den Bildschirm, was u. U. die Arbeit im Vordergrund empfindlich stört. Durch Ein-/Ausgabeumleitung kann der Hintergrundprozeß zum Schweigen gebracht werden:

```
find / -user markus -print > liste 2> /dev/null &  
2354
```

Eigenschaften von Hintergrundprozessen:

- Hintergrundprozesse sind Vordergrundprozessen gleichberechtigt, sie laufen gleich schnell ab.
- Jedes Programm kann im Hintergrund ablaufen.
- Die Standardeingabe und -ausgabe ist weiterhin an das Terminal gebunden (Umleitung nötig).
- Hintergrundprozesse (übrigens auch Vordergrundprozesse) können mit dem kill-Kommando (siehe später) zwangsterminiert werden.
- Die Shell wartet nicht auf den Hintergrundprozeß, sondern ist sofort wieder eingabebereit.

- Beim Logoff werden auch alle Hintergrundprozesse gekillt, sofern nicht besondere Maßnahmen getroffen werden.
- Mit dem ps-Kommando kann man den Status der Prozesse ansehen.

Weitere Kommandos für den Start von Hintergrundprozessen sind:

nice [-increment] Kommando &

Kommando zum Ausführen eines Hintergrundprozesses mit niedrigerer Priorität. `increment` gibt an, um wieviel die Priorität des Prozesses herabgesetzt werden soll (Voreinstellung: 10). Der Superuser kann Prozesse auch mit höherer Priorität laufen lassen, indem er ein negatives Increment angibt.

nohup Kommando &

(No Hang Up) Dieses Kommando ermöglicht es, einen Prozeß nach dem Logoff weiterlaufen zu lassen. Wenn die Ausgaben nicht explizit umgelenkt wurden, werden Standardausgabe und Standard-Fehlerausgabe in die Datei `nohup.out` im zuletzt aktuellen Verzeichnis geschrieben. Sollte dies nicht möglich sein (Zugriffsrechte nicht ausreichend), wird `nohup.out` im Home-Directory des Benutzers angelegt. Bei einigen Anlagen gibt es das Kommando "batch" mit analogen Eigenschaften.

4.2 Löschen von Prozessen

kill [-Signalnummer] Prozeßnummer (n)

Mit diesem Kommando kann der Benutzer eigene Prozesse löschen (auch im Vordergrund laufende Prozesse - von einem anderen Terminal aus oder aus einem anderen X-Fenster heraus). Der Superuser kann jeden Prozeß löschen. Dem Prozeß wird das Signal mit der beim kill-Aufruf angegebenen Nummer gesendet (Voreinstellung: 15). Fängt der Prozeß das Signal nicht ab, wird er terminiert. Über die in der Parameterzeile angegebenen Nnummer(n) werden die Prozeßnummer(n) der zu löschenden Prozesse angegeben. Wird hier eine **0** angegeben, so sind **alle** Prozesse des Benutzers zu löschen.

Neben anderen können für kill folgende Signalnummern verwendet werden:

- | | | |
|----|---------|--|
| 0 | SIGKILL | Terminate (beim Beenden der shell) |
| 1 | SIGHUP | Hangup (beim Beenden der Verbindung zum Terminal oder Modem) |
| 2 | SIGINT | Interrupt (wie Ctrl-C-Taste am Terminal) |
| 3 | SIGQUIT | Abbrechen (Beenden von der Tastatur aus) |
| 9 | SIGKILL | Kann nicht abgefangen werden - Beendet immer den empfangenden Prozeß |
| 15 | SIGTERM | Terminate (Software-Terminate, Voreinstellung) |

Die Datei `/usr/include/signal.h` enthält eine Liste aller Signale. Später wird das Kommando "trap" besprochen, mit dem man innerhalb von Shell-Skripts gezielt auf einzelne Signale reagieren kann.

4.3 Kommandos zur Zeitsteuerung

at zeit [datum] [+incr]

at [-rl] [nummer]

Veranlaßt die Ausführung von Kommandos, die über die Standardeingabe eingegeben werden, zu einem angegebenen Zeitpunkt, auch wenn der Benutzer zu diesem Zeitpunkt nicht angemeldet ist. `at` stützt sich dabei auf das Programm `atrun`, das in regelmäßigen Zeitabständen gestartet wird (in der Regel alle 5 Minuten). Das Kommando gibt die Auftragsnummer und den Zeitpunkt der Ausführung aus. Die Ausgabe der Kommandos (sofern nicht umgeleitet wurde) wird dem Benutzer per `mail`-Kommando zugestellt. Die auszuführenden Kommandos werden von der Standardeingabe gelesen (d. h. in den folgenden Zeilen, abgeschlossen mit `CTRL-D` oder per Eingabeumleitung). Die Benutzung dieses Kommandos kann explizit erlaubt oder verboten werden. Die aktuelle Umgebung wird für die `at`-Prozesse übernommen. Es gibt zwei besondere Aufrufe:

```
at -r Auftragsnummer(n) Löscht den genannten Auftrag
at -l [Auftragsnummer(n)] Listet die noch auszuführenden Aufträge
```

Fehlt bei `at -l` die Nummer, werden alle Aufträge gelistet.

Optionen:

`zeit`

Ausführungszeitpunkt in der Form `SSMM`. Werden nur zwei Ziffern angegeben, sind dies die Stunden. Ist der Zeitpunkt bereits überschritten, erfolgt der Start des Prozesses zum angegebenen Zeitpunkt des nächsten Tages. Für die Zeit sind auch folgende Angaben möglich: `"noon"` für Mittag, `"midnight"` für Mitternacht und `"now"` für "jetzt" (siehe `"increment"`).

`datum`

Datumsangabe für den Start des Prozesses in der Form `"Monat Tag"`, oder `"Monat Tag, Jahr"`. Für den Monat werden dreibuchstabile Kürzel verwendet: `Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec`. Für das Datum können auch die Angaben `"today"` oder `"tomorrow"` verwendet werden. Beispiele: `"Feb 28"`, `"Oct 15"` oder `"Feb 28,1991"`.

`incr`

Relative Zeitangabe, bestehend aus einer Zahl, gefolgt von einem der Begriffe: `"minutes"`, `"hours"`, `"days"`, `"weeks"`, `"years"`. Um zum Beispiel einen Prozeß in 3 Stunden starten, gibt man an: `at now +3 hours`

Einschränkung des `at`-Kommandos

Die Verwendung des Kommandos kann auf einen bestimmten Benutzerkreis eingeschränkt werden. Dazu dienen zwei Dateien, `usr/lib/cron/at.allow` und `usr/lib/cron/at.deny`.

- Existiert nur `at.allow`, wird nur den dort eingetragenen Benutzern die Verwendung des `at`-Kommandos erlaubt.
- Existiert nur `at.deny`, können alle dort *nicht* eingetragenen Benutzer das `at`-Kommando verwenden. Ist die Datei leer, dürfen es alle Benutzer verwenden.
- Existiert keine der beiden Dateien, darf nur der Superuser das `at`-Kommando verwenden.

Ein Beispiel für das `at`-Kommando: Am 1. Oktober um 22 Uhr wird das Ausdrucken einer langen Datei gestartet.

```
at 2200 Oct 1
lp -m superlangedatei.txt
CTRL-D
```

job 545900100.a at Tue Oct 1 22:00:00 1991

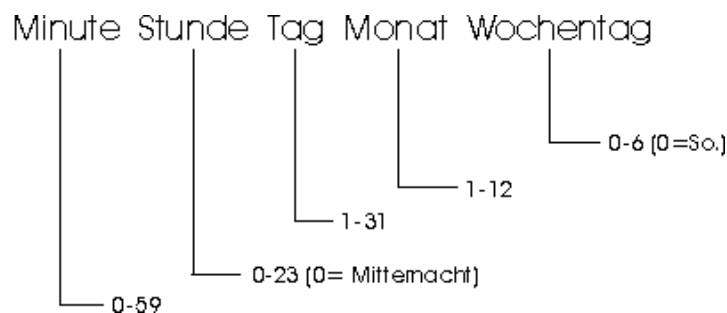
Mit `at -r 545900100.a` könnte man den o. g. Job wieder löschen.

crontab [-erl]

Das Crontab-Kommando erstellt bzw. bearbeitet eine Auftragsdatei für den cron-Daemon. Der cron wird nur einmal beim Systemstart aufgerufen. Er prüft in regelmäßigen Abständen (normalerweise jede Minute) den Inhalt der crontab-Dateien, in denen Zeitpunkte für automatisch ablaufende Programme festgelegt sind. Mit dem Kommando `crontab` kann jeder Benutzer eine crontab-Datei anlegen und ändern (-e), löschen (-r) oder auflisten (-l). Zum Ändern wird der voreingestellte Editor (ed oder vi) gestartet (die Festlegung erfolgt durch die Variable EDITOR). Die Ausgabe der Kommandos, bei denen keine Ausgabeumleitung erfolgte, wird über mail an den Benutzer gesendet.

Format der crontab-Datei:

Jede Zeile der crontab-Datei enthält sechs Felder (durch Leerzeichen getrennt). Die ersten fünf Felder sind eine Zeitangabe, das sechste enthält das auszuführende Kommando. Die fünf Zeit-Felder sind folgendermaßen aufgebaut:



Statt einer expliziten Zahlenangabe sind auch folgende Angaben möglich:

- Ein Teilbereich, z. B. 1-5
- Eine durch Komma getrennte Liste, z. B. 10,20,30,40,50
- Ein * (deckt den gesamten Bereich ab)

Einige Beispiele dazu:

<code>0 0 * * *</code>	Jeden Tag um Mitternacht
<code>0 9 * * 1</code>	Jeden Montag um 9 Uhr
<code>30 10 1 * 1</code>	Jeden Monatsersten und jeden Montag um 10 Uhr
<code>0,30 8-18 * * *</code>	Täglich alle halbe Stunde aber nur zwischen 8 und 18
<code>0,5,10,15,20,25,30,35,40,45,50,55 * * * *</code>	Alle 5 Minuten

Wie beim `at`-Kommando kann auch die Benutzung von `crontab` für bestimmte Benutzer gesperrt werden. Dazu dienen die Dateien `/usr/lib/cron/cron.allow` und `/usr/lib/cron/cron.deny`, bei denen das gleiche Schema angewendet wird, wie es schon beim `at`-Kommando besprochen wurde.

Die Tabellen (crontabs) stehen je nach UNIX-Variante in `/etc/cron.d` oder `/usr/lib/cron`. Beim Erstellen von cron- und at-Jobs sollte man sich nicht darauf verlassen, daß außer ein paar Standardvariablen irgendwelche Voreinstellungen verwendbar sind. Benutzen Sie entweder nur absolute Pfade oder eine explizite Pfad-Definition. Will man keinen Output per E-Mail erhalten, muß der Output (stdout und stderr) umgeleitet werden (Man kann den Job sehr schnell von E-Mail auf Dateiausgabe umstellen, indem man alle Kommandozeilen mit geschweiften Klammern einrahmt und dahinter die Ausgabeumleitung setzt: `{ } > Datei 2>&1`). Will man nur wenige Kontrollausgaben erzeugen, lassen sich auch mittels `echo >> logfile` cron-Jobs loggen. Bei root-cron-Jobs kann dafür auch die allgemeine Logdatei `/var/adm/messages` (manchmal auch `/var/log/messages`) verwendet werden.

4.4 Wechsel der Benutzeridentität

Um die Benutzeridentität zu wechseln, ist nicht unbedingt ein Logoff mit nachfolgendem Logon nötig. Speziell der Superuser als Systemverwalter sollte üblicherweise als normaler Benutzer arbeiten und seine Sonderrechte nur dann einsetzen, wenn es nötig ist - schon um bei Fehlern den Schaden einzugrenzen.

su [-] [Name]

(Substitute User) Der Benutzer arbeitet temporär unter einem neuen Benutzernamen. Er muß immer das Paßwort des neuen Benutzerkennzeichens angeben. Fehlt die Namensangabe, wird "root" (Superuser) angenommen (weshalb das Kommando irrtümlich oft als "SuperUser" interpretiert wird). Es wird eine neue Shell eröffnet. Zurück zur alten Identität gelangt man durch beenden der Shell mit CTRL-D oder dem Kommando `exit`. Der Wechsel betrifft nur die effektive UID, die "reale" UID bleibt unverändert, ebenso das aktuellen Verzeichnis, Shell-Variablen, etc. Wird zusätzlich die Option `'-'` angegeben, dann wird auch die komplette Login-Prozedur (z. B. Ausführen von `.profile`) durchlaufen.

4.5 Dateisystem einbinden

UNIX erlaubt selbstverständlich die Verwendung mehrere Platten, Bandgeräte, Diskettenlaufwerke, etc., die alle in das Dateisystem eingebunden werden können.

- Jede Platte enthält ein vollständiges Dateisystem.
- Die Koppelung erfolgt durch Ersetzen eines vorhandenen Verzeichnisses durch das Dateisystem der eingebundenen Platte (der ursprüngliche Inhalt des Verzeichnisses ist nicht mehr zugänglich).
- Austauschbare Datenträger dürfen nicht gewechselt werden, solange das Dateisystem dieses Datenträgers eingebunden ist.
- Das ursprüngliche Dateisystem erweitert sich um die Dateien und Verzeichnisse des neuen Datenträgers.
- Durch Abmelden wird das Dateisystem wieder aus dem Verzeichnisbaum entfernt.
- Es lassen sich auch Dateisysteme anderer Rechner einbinden (NFS = Network File System).
- Dateisysteme, die beim Booten des Rechners eingebunden werden sollen, stehen in der Datei `/etc/fstab` (oft auch `/etc/vfstab`).

mount [-a] [Gerätename] [Directory]

Anmelden (Montieren) eines Gerätes (Platte, Band, etc.). Das Directory (Mountpoint) muß als absoluter Pfadname angegeben werden. Ohne Angabe von Parametern erhält man eine Liste der

montierten Platten. Die zum Zeitpunkt des Bootvorgangs einzubindenden Dateisysteme (in `/etc/fstab` (`/etc/vfstab`) aufgeführt) lassen sich mit `mount -a` einbinden. Lassen sich diese Systeme nicht automatisch einbinden (z. B. Diskette, Wechselplatte, CD-ROM, usw.), können sie später durch "mount Directory", z. B. `mount /cdrom` eingebunden werden, ohne daß eine Geräteangabe nötig ist, sofern sie in `/etc/fstab` aufgeführt sind. Je nach UNIX-Variante hat das `mount`-Kommando noch weitere Parameter, z. B. den Typ des Dateisystems oder Nur-Lese/Schreibzugriff. So lassen sich auch andere Typen von Dateisysteme "mounten", etwa unter Linux auch MS-DOS- oder Windows-Partitionen.

umount Geräteiname

Abmelden einer Platte aus dem Dateisystem. Es darf kein Benutzer oder Prozeß mehr offene Dateien auf dieser Platte haben (am besten einmal 'cd' eingeben, bevor man die Platte abmeldet). Beim Abmelden werden auch alle Dateipuffer auf die Platte geschrieben. Wechseldatenträger (Diskette, Wechselplatte) unbedingt vor dem Herausnehmen abmelden. Übrigens: Das Kommando schreibt sich wirklich 'umount' - ohne 'n' hinter dem 'u'.

Natürlich gibt es noch weitere Kommandos für Dateisysteme, z. B. zum Partitionieren der Platte (bei Linux heißt es "fdisk", bei Sun-OS "format"). Eine neue Platte muß zuerst mit einem Dateisystem versehen ("formatiert") werden. Das Kommando dazu heißt "mkfs". Gebrauch und Parameter muß man der jeweiligen Dokumentation entnehmen. Zum Überprüfen der Platten gibt es das Kommando "fsck". Es wird normalerweise beim Booten automatisch ausgeführt. Für den Fall des manuellen Aufrufs sollte auch hier die Dokumentation des jeweiligen Systems konsultieren.

Disk-Quotas

Bei einem System mit vielen Benutzern sollte man sich als Administrator überlegen, ob man nicht Disk-Quotas einführt. Einzelne Benutzer können sonst die gesamte Festplatte, oder zumindest die Home-Partition mit Daten füllen und so die Arbeit aller anderen Anwender blockieren. Wenn Sie für das Home-Verzeichnis eine eigene Partition angelegt haben, so läuft das System zwar weiter, aber die User können es nicht mehr wie gewohnt nutzen. UNIX erlaubt Quotas für einzelne Benutzer oder für Gruppen. Die Beschränkungen gelten jeweils für eine einzelne Partition. Gruppenquotas geben die Summe des Speicherplatzes an, den alle Mitglieder dieser Gruppe gemeinsam belegen dürfen. Es lassen sich Obergrenzen für den belegten Plattenplatz und für die Anzahl der Dateien festlegen. Bei beiden Möglichkeiten können Sie zwei unterschiedliche Grenzen setzen:

- Das Hard-Limit ist eine Grenze, die der Benutzer auf keinen Fall überschreiten kann.
- das Soft-Limit darf der Benutzer eine bestimmte Zeit lang überschreiten, aber nur bis zum Hard-Limit.

4.6 Weitere Informationskommandos

lsof

Dieses Kommando zeigt geöffnete Dateien, Verzeichnisse, Unixsockets und IP-Sockets (siehe Netzwerk-Kapitel) sowie Pipes an. Mit den passenden Optionen aufgerufen, zeigt es z.B.

- alle Dateien und Netzverbindungen, die ein bestimmter Prozess geöffnet hat,
- alle Prozesse, die eine bestimmte Datei oder Netzverbindung geöffnet haben oder
- die Namen aller Prozesse, die auf eine Netzverbindung warten.

Betriebssystem UNIX/Linux

Wenn der User "root" das Kommando ohne Parameter aufruft, zeigt es alles, was von sämtlichen Prozessen geöffnet ist - und das ist eine Menge (oft über 1000 Zeilen):

```
COMMAND    PID      USER    FD     TYPE    DEVICE    SIZE      NODE NAME
init        1        root    cwd    DIR      3,2      4096        2 /
init        1        root    rtd    DIR      3,2      4096        2 /
init        1        root    txt    REG      3,2     27844     1782394 /sbin/init
init        1        root    mem    REG      3,2     90210     196307 /lib/ld-2.2.5.so
init        1        root    mem    REG      3,2    1153784     196310 /lib/libc-2.2.5.so
init        1        root    10u    FIFO     3,2                2339761 /dev/initctl
keventd     2        root    cwd    DIR      3,2      4096        2 /
keventd     2        root    rtd    DIR      3,2      4096        2 /
keventd     2        root    10u    FIFO     3,2                2339761 /dev/initctl
ksoftirqd   3        root    cwd    DIR      3,2      4096        2 /
ksoftirqd   3        root    rtd    DIR      3,2      4096        2 /
ksoftirqd   3        root    10u    FIFO     3,2                2339761 /dev/initctl
kswapd      4        root    cwd    DIR      3,2      4096        2 /
kswapd      4        root    rtd    DIR      3,2      4096        2 /
kswapd      4        root    10u    FIFO     3,2                2339761 /dev/initctl
bdflush     5        root    cwd    DIR      3,2      4096        2 /
bdflush     5        root    rtd    DIR      3,2      4096        2 /
bdflush     5        root    10u    FIFO     3,2                2339761 /dev/initctl

...

lsuf       27430    root    2u    CHR      136,1                3 /dev/pts/1
lsuf       27430    root    3r    DIR       0,3                0 1 /proc
lsuf       27430    root    4r    DIR       0,3                0 1797652488 /proc/27430/fd
lsuf       27430    root    5w    FIFO     0,6                7473007 pipe
lsuf       27430    root    6r    FIFO     0,6                7473008 pipe
lsuf       27431    root    cwd    DIR      3,2      4096     3662907 /home/plate
lsuf       27431    root    rtd    DIR      3,2      4096        2 /
lsuf       27431    root    txt    REG      3,2     108956    1831546 /usr/sbin/lsuf
lsuf       27431    root    mem    REG      3,2     90210     196307 /lib/ld-2.2.5.so
lsuf       27431    root    mem    REG      3,2    1153784     196310 /lib/libc-2.2.5.so
lsuf       27431    root    4r    FIFO     0,6                7473007 pipe
lsuf       27431    root    7w    FIFO     0,6                7473008 pipe
```

Mit den passenden Parametern kann man die Ausgabe auf die interessierenden Zeilen eingrenzen. Gibt man mehrere Filterkriterien an, so zeigt `lsuf` diejenigen Zeilen an, die entweder mindestens eine oder (wenn man zusätzlich `-a` angibt) alle Bedingungen erfüllen.

Datei(en)

Zugriffe auf diese Datei(en) anzeigen. Beispiel: Wer benutzt gerade `bash`?

```
# lsuf /bin/bash
COMMAND    PID      USER    FD     TYPE    DEVICE    SIZE      NODE NAME
bash       19041    holzmann txt    REG      3,2    511400    2322044 /bin/bash
bash       21620    root    txt    REG      3,2    511400    2322044 /bin/bash
bash       26921    plate   txt    REG      3,2    511400    2322044 /bin/bash
bash       26937    root    txt    REG      3,2    511400    2322044 /bin/bash
```

+D Verzeichnis

Zugriffe auf die Dateien unterhalb des Verzeichnisses anzeigen. **Beispiel:** Wer arbeitet mit Dateien im `/home`-Verzeichnis?

```
# lsuf +D /home
COMMAND    PID      USER    FD     TYPE    DEVICE    SIZE      NODE NAME
bash       26921    plate   cwd    DIR      3,2     4096     3662907 /home/plate
bash       26937    root    cwd    DIR      3,2     4096     3662907 /home/plate
lsuf       27890    root    cwd    DIR      3,2     4096     3662907 /home/plate
lsuf       27891    root    cwd    DIR      3,2     4096     3662907 /home/plate
```

Betriebssystem UNIX/Linux

+p PIDs

Alles anzeigen, was die Prozesse mit diesen PIDs geöffnet haben. Um mehrere PIDs anzugeben, diese mit Komma trennen. Will man die Prozesse namentlich angeben, verwendet man `lsof -c [name]`. **Beispiel:** Welche Dateien benötigt meine Shell?

```
# ps
  PID TTY          TIME CMD
26937 pts/1    00:00:00 bash
27899 pts/1    00:00:00 ps

# lsof +p 26937
COMMAND  PID USER   FD   TYPE DEVICE   SIZE   NODE NAME
bash    26937 root    cwd   DIR    3,2     4096 3662907 /home/plate
bash    26937 root    rtd   DIR    3,2     4096      2 /
bash    26937 root    txt   REG    3,2    511400 2322044 /bin/bash
bash    26937 root    mem   REG    3,2     90210 196307 /lib/ld-2.2.5.so
bash    26937 root    mem   REG    3,2    248132 196280 /lib/libncurses.so.5.2
bash    26937 root    mem   REG    3,2     8008 196313 /lib/libdl-2.2.5.so
bash    26937 root    mem   REG    3,2   1153784 196310 /lib/libc-2.2.5.so
bash    26937 root    mem   REG    3,2     40152 196316 /lib/libnss_compat-2.2.5.so
bash    26937 root    mem   REG    3,2     69472 196315 /lib/libnsl-2.2.5.so
bash    26937 root     0u   CHR   136,1          3 /dev/pts/1
bash    26937 root     1u   CHR   136,1          3 /dev/pts/1
bash    26937 root     2u   CHR   136,1          3 /dev/pts/1
bash    26937 root    255u  CHR   136,1          3 /dev/pts/1
```

-u user[user...]

Alles anzeigen, was die Prozesse geöffnet haben, die einem der angegebenen Usernamen oder User-ID gehören. Um mehrere User anzugeben, diese mit Komma trennen. **Beispiel:** Welche Dateien werden zurzeit von Plate benutzt?

```
# lsof -u plate
COMMAND  PID USER   FD   TYPE DEVICE   SIZE   NODE NAME
bash    26921 plate   cwd   DIR    3,2     4096 3662907 /home/plate
bash    26921 plate   rtd   DIR    3,2     4096      2 /
bash    26921 plate   txt   REG    3,2    511400 2322044 /bin/bash
bash    26921 plate   mem   REG    3,2     90210 196307 /lib/ld-2.2.5.so
bash    26921 plate   mem   REG    3,2    248132 196280 /lib/libncurses.so.5.2
bash    26921 plate   mem   REG    3,2     8008 196313 /lib/libdl-2.2.5.so
bash    26921 plate   mem   REG    3,2   1153784 196310 /lib/libc-2.2.5.so
bash    26921 plate   mem   REG    3,2     40152 196316 /lib/libnss_compat-2.2.5.so
bash    26921 plate   mem   REG    3,2     69472 196315 /lib/libnsl-2.2.5.so
bash    26921 plate     0u   CHR   136,1          3 /dev/pts/1
bash    26921 plate     1u   CHR   136,1          3 /dev/pts/1
bash    26921 plate     2u   CHR   136,1          3 /dev/pts/1
bash    26921 plate    255u  CHR   136,1          3 /dev/pts/1
...
```

-i [TCP|UDP][@host][: ports]

Die Netzverbindungen des übergebenen Hosts und Ports anzeigen. Dabei kann der Host wahlweise als Hostname oder IP-Adresse und die Ports als Portnummern oder Servicenamen angegeben werden. Um mehrere Ports zu betrachten, kann man Listen (z.B. `ssh`, `www`) oder Bereiche (z.B. `1-1024`) angeben. **Beispiel:** Welche Prozesse kommunizieren da auf Port 80 miteinander?

```
menetekel:/home/plate# lsof -i :80
COMMAND  PID USER   FD   TYPE DEVICE   SIZE   NODE NAME
apache   11299 root    18u  IPv4  7063707      TCP *:www (LISTEN)
apache   26057 root    18u  IPv4  7063707      TCP *:www (LISTEN)
apache   26960 root    18u  IPv4  7063707      TCP *:www (LISTEN)
apache   26961 root    18u  IPv4  7063707      TCP *:www (LISTEN)
apache   27026 root    18u  IPv4  7063707      TCP *:www (LISTEN)
apache   27037 root    18u  IPv4  7063707      TCP *:www (LISTEN)
apache   27038 root    18u  IPv4  7063707      TCP *:www (LISTEN)
```

Betriebssystem UNIX/Linux

```
apache 27383 root 18u IPv4 7063707 TCP *:www (LISTEN)
apache 27402 root 18u IPv4 7063707 TCP *:www (LISTEN)
apache 27427 root 18u IPv4 7063707 TCP *:www (LISTEN)
apache 27919 root 18u IPv4 7063707 TCP *:www (LISTEN)
```

strace, ltrace

`strace` schaltet sich zwischen einen Benutzerprozeß und den Kernel und protokolliert alle Aktivitäten an dieser Schnittstelle. Wenn der verfolgte Prozeß einen Systemaufruf macht, gibt `strace` den Namen, die Argumente des Aufrufs und den Rückgabewert dieses Systemaufrufs aus. Wenn der verfolgte Prozeß ein Signal erhält, gibt `strace` den Namen des Signals aus.

Falls ein Systemaufruf mit einem Fehler zurückkehrt, wird, wenn vorhanden, die dem Fehlerstatus zugeordnete Fehlermeldung angezeigt. Zeichenketten als Argumente eines Systemaufrufs werden nur bis zu einer bestimmten Länge ausgegeben. Standardmäßig ist dieser Wert auf 32 Zeichen eingestellt. Wenn eine Zeichenkette länger als der eingestellte Wert ist, werden die fehlenden Zeichen durch zwei Punkte angedeutet.

Es ist möglich, einen einzelnen Prozeß oder eine ganze Prozeßfamilie zu verfolgen. Wenn die von einem verfolgten Prozeß erzeugten Kindprozesse auch verfolgt werden, setzt die Protokollierung erst ein, nachdem der das Kind erzeugende Systemaufruf zurückkehrt und die ProzeßID des Kindes an den Elternprozeß zurückgibt. Zu diesem Zeitpunkt kann der Kindprozeß bereits Systemaufrufe gemacht haben, die dann nicht protokolliert sind.

Wenn `strace` zu einem bereits laufenden Prozeß hinzugeschaltet wird, können nur solche Kindprozesse verfolgt werden, die nach dem Hinzuschalten erzeugt werden.

Sie können das Protokoll auch in eine Datei schreiben lassen. Wenn Sie die Kinder des verfolgten Prozesses auch verfolgen, werden die Protokolle für die Kindprozesse in separaten Dateien gespeichert, deren Namen mit dem für den Stammprozeß gewählten übereinstimmen und auf die ProzeßID des jeweiligen Kindes enden. Optionen:

- d
(debug) veranlaßt `strap`, zusätzliche Informationen über eigene Systemaufrufe und Signale auf den Standardfehlerkanal zu schreiben
- t
(time) läßt am Anfang jeder Protokollzeile die Zeit (Std:Min:Sec) ausgeben
- f
(follow) veranlaßt die Verfolgung von Kindprozessen des verfolgten Prozesses
- o *datei*
veranlaßt `strace`, das Protokoll in die Datei zu schreiben
- s *länge*
verändert die standardmäßig auf 32 Zeichen beschränkte Länge der Ausgabe von Zeichenketten als Argument eines Systemaufrufs
- p *prozess-id*
schaltet `strace` zum laufenden Prozeß mit der Prozeßnummer *prozess-id* hinzu; ohne Rootprivilegien können Sie nur Ihre eigenen Prozesse verfolgen.

`ltrace` leistet Ähnliches wie `strace` in Bezug auf die Verwendung von dynamischen Bibliotheken (Libraries) durch Prozesse.

ldd

Dieses Kommando listet alle Bibliotheken auf, die ein Programm benötigt (also im Gegensatz zu `ltrace` statisch). Zum Beispiel:

```
$ ldd /bin/bash
libncurses.so.5 => /lib/libncurses.so.5 (0x40017000)
```

Betriebssystem UNIX/Linux

```
libdl.so.2 => /lib/libdl.so.2 (0x40055000)
libc.so.6 => /lib/libc.so.6 (0x40059000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

uname

gibt Auskunft über Betriebssystem und Hardware. Optionen:

```
-s          zeigt den Betriebssystemnamen (Voreinstellung, wenn keine Option angegeben wird)
-n          zeigt den Netzwerknamen des Systems
-r          zeigt die Release (Versionsnummer) des Betriebssystems
-v          zeigt das Erstellungsdatum des Betriebssystems
-m          zeigt den Prozessortyp
-a          zeigt alle der oben genannten Daten
```

Beispiel:

```
$ uname -a
Linux menetekel 2.4.18-bf2.4 #1 Son Apr 14 09:53:28 CEST 2002 i686 unknown
```

free

Zeigt die Speicherbelegung an. Zum Beispiel:

```
$ free
              total        used         free       shared    buffers     cached
Mem:          512960        506016         6944           0         51296     387396
-/+ buffers/cache:      67324      445636
Swap:         497972          1740       496232
```

uptime

Zeigt die Zeit seit dem letzten reboot und die aktuelle Last an, z. B.:

```
$ uptime
17:09:18 up 143 days, 59 min,  3 users,  load average: 0.07, 0.07, 0.07
```

Weitere Informationskommandos unter Linux

Systeminfos

Die Hardware eines Rechners ist für manchen Anwender ein Buch mit sieben Siegeln. Zwar liefert Linux eine Fülle von Informationen über die verbauten Komponenten - allerdings nicht unbedingt in einer leicht verständlichen Form. Unter `/proc` befinden sich beim klassischen Unix nur Informationen zu Prozessen. Bei Linux findet man jedoch noch einiges mehr. Die Informationen können ganz einfach z. B. per `cat`-Kommando auf der normalen Konsole angezeigt werden, da es reine (Pseudo-)Textdateien sind. Bei `/proc` handelt es sich nämlich um ein virtuelles Dateisystem, das der Kernel anlegt und in dem die laufenden Prozesse in Unterverzeichnisse abgebildet werden. Es beansprucht keinerlei Platz auf der Festplatte, auch wenn manche Dateien scheinbar eine enorme

Betriebssystem UNIX/Linux

Größe aufweisen. Allerdings erleichtert das Proc-Verzeichnis durch eine Vielzahl von Unterverzeichnissen und teilweise auch mit vieldeutigen Verzeichnisnamen die Suche nach spezifischen Informationen nicht gerade. Auch sind nicht alle Dateien bei jeder Hardware zu finden. Unter anderem finden Sie:

<code>/proc/cpuinfo</code>	Informationen zur CPU (u.a. Name, Taktfrequenz, Cache-Größe)
<code>/proc/meminfo</code>	Informationen über den Arbeitsspeicher (u.a. Gesamtgröße, Cache, Swap)
<code>/proc/partitions</code>	Informationen über Partitionen des Massenspeichers (u.a. Partitionsname, Anzahl der Blöcke)
<code>/proc/modules</code>	Übersicht über die geladenen Module (u.a. Name, Speicheradresse)
<code>/proc/devices</code>	Übersicht über vorhandene Char- und Block-Devices
<code>/proc/version</code>	Informationen über die Kernelversion
<code>/proc/loadavg</code>	aktuelle Systemauslastung
<code>/proc/stat</code>	Prozesse
<code>/proc/interrupts</code>	Übersicht über Interrupts
<code>/proc/net/dev</code>	Statistik der Netzwerkgeräte (u.a. übertragene Bytes, Fehler)
<code>/proc/bus</code>	Informationen und Übersichten über die Busse im Rechner.
<code>/proc/bus/*/devices</code>	Übersichten über die jeweiligen Geräte des Bus
<code>/proc/ide</code>	Hier finden sich die IDE-Geräte
<code>/proc/ide/*/model</code>	Die Modellbezeichnung des Geräts
<code>/proc/acpi/thermal-zone/THRG/temperature</code>	CPU-Temperatur
<code>/proc/acpi/battery/BAT?/state</code>	Batteriezustand
<code>/proc/acpi/ibm/fan</code>	Lüfterstatus
<code>/proc/<PID>/cmdline</code>	Kommandozeile des Prozesses mit der Nummer PID

Das Programm "Hardinfo" bereit dem Durcheinander durch eine ansprechende grafische Oberfläche und eine übersichtliche Präsentation der Hardware-Informationen ein Ende. Hardinfo steht unter der GPL und kann von <http://freshmeat.net/projects/hardinfo> heruntergeladen werden. Danach entpacken Sie das Archiv zunächst in ein temporäres Verzeichnis:

```
tar xjvf hardinfo-w.x.y.z.tar.bz2
```

beziehungsweise

```
tar xzvf hardinfo-w.x.y.z.tar.tar.gz
```

(w.x.y.z steht für die gerade aktuelle Version). Dann wechseln Sie in das neu angelegte Verzeichnis und richten das Programm mit dem übliche Dreisprung

```
./configure && make && make install
```

ein. Versuchen Sie es vorher auf jeden Fall mit dem Paketmanager Ihrer Distribution - das geht einfacher und schneller. Bei Debian ist es beispielsweise schon verfügbar und kann mittels `apt-get install hardinfo` eingerichtet werden.

Sysstat-Paket

Das Paket `sysstat` ist bei Linux und einigen anderen UNIX-Derivaten eine Sammlung von Kommandozeilen-Tools, die einen schnellen Überblick des Systems verschaffen. Die Kommandos bilden ein Frontend zum Linux-Kernel, und können daher auch nur Daten ausgeben, die der Kernel liefert - sie sammeln keine Daten darüber hinaus!

iostat

`iostat` liefert Status-Informationen über den Datendurchsatz der Festplatten, der angeschlossenen NFS-Laufwerke und der CPUs. Zum Beispiel:

```
# iostat
Linux 2.6.32-5-686 (info)      19.09.2011      _i686_ (4 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0,02    0,00    0,01    0,01    0,00   99,97

Device:            tps    Blk_read/s    Blk_wrtn/s    Blk_read    Blk_wrtn
sda                  0,26         0,45         3,97       281194     2507152
```

`iostat` besitzt zahlreiche Optionen, unter anderem:

- d Anzeige nur der Festplatten-Daten.
- c Anzeige der reinen CPU-Daten.
- p Anzeige der IO-Daten je Partition.
- n Anzeige der NFS-IO-Daten.
- x Erweiterte Anzeige der Festplatten-Daten.
- t Zahl Angabe, nach wie vielen Sekunden die Anzeige aktualisiert werden soll.

mpstat

`mpstat` dient der Analyse der Prozessor-Auslastung und liefert eine aussagekräftigere Info als `uptime`. Ohne Option wird eine Standardübersicht gezeigt, die an die Ausgabe von `iostat` angelehnt ist:

```
# mpstat
Linux 2.6.32-5-686 (info)      19.09.2011      _i686_ (4 CPU)

18:08:03   CPU   %usr   %nice   %sys %iowait   %irq   %soft   %steal   %guest   %idle
```

Betriebssystem UNIX/Linux

```
18:08:03    all    0,01    0,00    0,01    0,01    0,00    0,00    0,00    0,00    99,97
```

Mit der Option `-A` wird die Ausgabe erweitert: die Statistiken werden pro Prozessor/Core angezeigt, ausserdem werden die Interrupts pro Sekunde pro Prozessor angezeigt. Gibt man am Ende der Kommandozeile eine Zahl `N` an, läßt `mpstat` dauernd und aktualisiert die Anzeige alle `N` Sekunden.

pidstat

Dieses Kommando dient der Analyse einzelner Prozesse. Der Aufruf ohne Optionen zeigt eine Liste aller Prozesse an, z. B.:

```
# pidstat
Linux 2.6.32-5-686 (info)          19.09.2011    _i686_ (4 CPU)

18:14:04      PID    %usr  %system  %quest    %CPU   CPU  Command
18:14:04        1     0,00   0,00   0,00     0,00    1   init
18:14:04        4     0,00   0,00   0,00     0,00    0  ksoftirqd/0
18:14:04        9     0,00   0,00   0,00     0,00    2  migration/2
18:14:04       10     0,00   0,00   0,00     0,00    2  ksoftirqd/2
18:14:04       12     0,00   0,00   0,00     0,00    3  migration/3
18:14:04       15     0,00   0,00   0,00     0,00    0  events/0
18:14:04       16     0,00   0,00   0,00     0,00    1  events/1
18:14:04       17     0,00   0,00   0,00     0,00    2  events/2
18:14:04       18     0,00   0,00   0,00     0,00    3  events/3
18:14:04       24     0,00   0,00   0,00     0,00    0  sync_supers
18:14:04       25     0,00   0,00   0,00     0,00    1  bdi-default
18:14:04       37     0,00   0,00   0,00     0,00    0  kseriod
18:14:04       46     0,00   0,00   0,00     0,00    3  khungtaskd
18:14:04      217     0,00   0,00   0,00     0,00    0  scsi_eh_1
18:14:04      306     0,00   0,00   0,00     0,00    0  kjournald
18:14:04      356     0,00   0,00   0,00     0,00    2  udevd
18:14:04      494     0,00   0,00   0,00     0,00    2  edac-poller
17:02:32     2889     0,00   0,00   0,00     0,00    0  kdeinit4
...
```

Die Option `-C` gestattet die Spezifizierung der zu untersuchenden Prozesse. Mit `-p` kann statt des Prozess-Namens die PID definiert werden. Beispiel:

```
# pidstat -C bash
Linux 2.6.32-5-686 (info)          19.09.2011    _i686_ (4 CPU)

18:16:12      PID    %usr  %system  %quest    %CPU   CPU  Command
18:16:12     3675     0,00   0,00   0,00     0,00    2   bash
18:16:12     3726     0,00   0,00   0,00     0,00    1   bash
```

Mit der Option `-d` werden I/O-Statistiken zu den Prozessen angezeigt und `-r` verschafft einen Überblick der Speicherauslastung. Auch hier kann durch eine Zahl am Ende ein Dauerbetrieb initiiert werden.

sar

Da die oben beschriebenen Kommandos jeweils nur einen Schnappschuss der Systemleistung wiedergeben, kann mit `sar` ein Status-Logging erreicht werden. Das Programm nimmt via Cron-Job z. B. alle 5 Minuten verschiedene Leistungsdaten des Systems auf und speichert diese ab. `sar` kennt wieder zahllose Optionen, einen Überblick gibt die Manual-Seite. Aös Konfigurationsdatei dient `/etc/default/sysstat`.

Nützliche Programme

Die folgenden Programme geben Auskunft über verschiedene Systemdaten, laufende Prozesse, die Rechnerauslastung, das Netzwerk usw.

free	Arbeitsspeicherverwendung, s. o.
df	Speicherplatz auf gemounteten Geräten
uptime	Anzeige der Systemlast und der Uptime, s. o.
hostname	Anzeige des Hostname
date	Systemzeit anzeigen
uname	OS-Namen anzeigen
ifconfig	Informationen über Netzwerkgeräte (nur als root)
iwconfig	desgleichen für WLAN-Benutzer
top	kurze Systeminformationen (Last, Speicher, etc.) und Anzeige der Prozesse z. B. <code>top -b -n 1</code>
htop	die noch komfortablere version von top
ps	Anzeige der Prozesse
pstree	Anzeige der Prozesse in Baumform
hddtemp	Zeigt die Temperatur von SMART-fähigen Festplatten an
acpi	Bietet eine komfortable Schnittstelle zu ACPI
dmesg	Alle Systemstart-Informationen des Kernels (u.a. auch die Erkennung der Hardware)
iptraf	Viele Möglichkeiten zur Überwachung des IP-Verkehrs
lsof	list open files, siehe oben
lsmod	Module auflisten
lspci	PCI-Bus auflisten
lsusb	USB-Devices auflisten
lsattr	Dateisystem-Attribute auflisten
netstat	Informationen über Netzwerkverbindungen, Routingtabelle, etc. z. B. <code>netstat -nt</code> (alle tcp-Verbindungen), <code>netstat -nlt</code> (alle Serverprozesse)
arp -nav	aktuelle ARP-Tabelle
nmap	Portscanner
route	eingetragene Routen
mount	eingebundene Dateisysteme
last	Tabelle der letzten Logins
lastb	Tabelle der letzten fehlgeschlagenen Logins (nur als root)



[Zum Inhaltsverzeichnis](#)



[Zum nächsten Abschnitt](#)

Copyright © FH München, FB 04, Prof. Jürgen Plate

Letzte Aktualisierung: 20. Sep 2011



Vorlesung "UNIX"

von Prof. Jürgen Plate

Zeileneditoren und reguläre Ausdrücke

5.1 Der Editor ed

ed [-Optionen] [Dateiname]

Der ed ist ein stilles Programm. Nach dem Aufruf tut sich kaum etwas. Er ist so richtig etwas für Risikofreudige, es gibt normalerweise nicht einmal ein Prompt-Zeichen. ed arbeitet immer mit einem Puffer im Hauptspeicher, auf die Platte wird erst durch ein ed-Kommando geschrieben. Ist die im Kommandoaufruf angegebene Datei vorhanden, wird die Länge der Datei in Bytes ausgegeben. Ist noch keine Datei vorhanden wird ein Fragezeichen und der Dateiname ausgegeben. Wurde der Dateiname weggelassen, tut sich gar nichts. Optionen:

- s Unterdrückt die Anzeige der gelesenen/geschriebenen Bytes.
- p Nach dem p wird ein Promptzeichen angegeben, das der ed dann im Kommandomodus ausgibt (z. B. -p#).

Der ed kennt zwei Verarbeitungsmodi, den Kommandomodus und den Eingabemodus. Nach dem Aufruf befindet er sich immer im Kommandomodus und, falls schon eine Text vorhanden war, am Ende der Datei. Um in den Eingabemodus zu gelangen, muß eines der folgenden Kommandos gegeben werden:

- a (append) Anhängen des Textes an die aktuelle Zeile
- i (insert) Einfügen des Textes vor der akt. Zeile
- c (change) Ersetzen von Text durch die Eingabe

Die maximale Zeilenlänge beträgt 512 Zeichen. Zur Korrektur gibt es nur "Backspace" (#, CTRL-H) und "Zeile löschen" (@, CTRL-X). **Zurück in den Kommandomodus gelangt man durch eine Zeile, die als einziges Zeichen einen Punkt enthält.** ed arbeitet zeilenorientiert, wobei ein interner Zeiger immer auf eine bestimmte Zeile zeigt (zu Beginn auf die letzte Zeile). Ein ed-Kommando besteht aus Zeilenadresse und Editier-Anweisung, wobei bei fehlender Adresse die aktuelle Zeile verwendet wird:

[Anfangsadresse [,Endadresse]] Anweisung

Man kann also eine einzige Zeile oder einen Bereich definieren:

Bereich	Beispiel	Bedeutung
Zeilenr.	5	setze Zeiger auf die angegebene Zeile
Z-Nr , Z-Nr	1,5	bearbeite Zeilen 1 bis 5
\$	\$	letzte Zeile
.	.	aktuelle Zeile
.,\$.,\$	von der aktuellen Zeile bis Dateiende
1,.	1,.	von der ersten Zeile bis zur aktuellen Zeile
1,\$	1,\$	gesamter Text

Z-Nr,\$ 20,\$ bearbeite Zeile 20 bis Ende

keine Angabe Return-Taste nächste Zeile

Außerdem ist relative Adressierung mit **+n** oder **-n** (n = Zahl) möglich, z. B. adressiert **.,+10** die folgenden 10 Zeilen. Eine weitere Möglichkeit der Adressierung ist ein regulärer Ausdruck (dazu unten noch mehr):

/reg.Ausdr./ Sucht ab der aktuellen Zeile (bis \$) nach einer Zeile, die den regulären Ausdruck erfüllt. Wird keine solche Zeile gefunden, wird ab Zeile 1 bis zur aktuellen Zeile gesucht - also gewissermaßen "rundherum".

?reg.Ausdr.? sucht genauso, jedoch von der aktuellen Position aus rückwärts.

Man kann die regulären Ausdrücke genauso wie die Zeilennummern zum Adressieren von Textzeilen verwenden. Um beispielsweise die erste Zeile zu löschen, in der das Wort "Festplatte" enthalten ist, genügt es, den Befehl `/Festplatte/d` einzugeben.

5.1.1 Editor-Anweisungen:

In einer Zeile darf immer nur eine Anweisung gegeben werden, allerdings darf an alle Anweisungen (außer e, f, r, w) eine der folgenden Ausgabeanweisungen angehängt werden:

l (list) Auflisten der adressierten Zeilen

n (number) Auflisten der Zeilen mit Zeilennummern

p (print) Auflisten der Zeilen (Voreinstellung, wenn Anweisung fehlt)

Für alle Anweisungen gibt es eine Voreinstellung (default) für die Zeilenadressierung, die dann genommen wird, wenn keine Zeilenadressierung vor der Anweisung erfolgt. Die Kommandos a, i und c werden, wie schon erwähnt, mit einer Zeile abgeschlossen, die nur einen Punkt enthält.

Anweisung	default	Funktion
a	.	(append) Anhängen von Text an die adressierte Zeile
c	.,.	(change) Ersetzen der adressierten Zeilen durch neuen Text
d	.,.	(delete)Löschen der adressierten Zeilen
e Datei		(edit) Angegebene Datei laden
f		(file) Dateinamen ausgeben
f Datei		(file) Dateinamen festlegen(merken)
g/ra/x	1,\$	(global) Editoranweisung x für alle Zeilen ausführen, die den regulären Ausdruck ra erfüllen
g		(global) g kann auch an eine Anweisung angehängt werden; die Anweisung wird global ausgeführt.
h		(help) Erklärung zur letzten Fehlermeldung (die nur aus '?' besteht)
H		(Help) Help-Funktion auf Dauer einschalten
i	.	(insert) Wie append, jedoch Einfügen vor der aktuellen Zeile
j	.,.	+1 (join) Zeilen aneinanderhängen
l	.,.	(list) Zeilen komplett auflisten
m Adr	.,.	

		(move) Zeilen hinter Zeile Adr verlagern n .. (number) Zeilen mit Nummern anzeigen
p	..	(print) Zeilen anzeigen
P		(Prompt) Promptzeilen ein-/ausschalten
q		(quit) Editor verlassen. ed warnt mit '?', wenn nicht gespeichert wurde
Q		(Quit) Editor ohne Warnung verlassen (Text ist weg)
r Datei	\$	(read) Text aus der angegebenen Datei einlesen (hinter die adress. Zeile)
s/ra/x/	..	(substitute) Vom regulären Ausdruck abgedeckten String durch x ersetzen
t Adr	..	(transfer) Den adressierten Textbereich hinter die Zeile Adr kopieren
u		(undo) Letzte Anweisung zurücknehmen
w	1,\$	(write) Text auf Datei schreiben
w Datei	1,\$	(write) Text auf Datei mit dem angegebenen Namen schreiben
!Kdo		UNIX-Kommando Kdo ausführen und in den ed zurückkehren

5.1.2 Textersetzung (s-Anweisung), Teil 1

Die s-Anweisung erlaubt in ihrer einfachsten Form das Ersetzen einer Zeichenkette durch eine andere:

```
s/alter Text/neuer Text/
```

Es wird das erste Vorkommen des alten Textes in der Zeile durch den neuen Text ersetzt. Zum

Beispiel:

```
Diese Zeile enthält einen Tuppfehler
```

```
s/Tu/Ti/p
```

```
Diese Zeile enthält einen Tippfehler
```

Kommt der alte Text mehrfach in der Zeile vor, wird er nur einmal ersetzt. Um in der gesamten Zeile zu ersetzen, wird das g angehängt:

```
s/Tu/Ti/g
```

Will man eine Zeichenkette im gesamten Text ersetzen, wird dazu der gesamte Text (oder auch nur ein Bereich) adressiert:

```
1,$s/Tu/Ti/g
```

Achtung! Es werden u. U. auch Zeichenketten ersetzt, die gar nicht ersetzt werden sollen. Im o. g. Beispiel würde auch "Tulpe" zu "Tilpe". Weitere Möglichkeiten:

- Der Ersetzungsstring darf natürlich auch leer sein.
- Es gibt keine Möglichkeit, das Zeilenende zu ersetzen.
- Anstelle der Zeilenadressierung für die gewünschte(n) Zeile(n) können auch reguläre Ausdrücke verwendet werden.
- Das &-Zeichen dient als Abkürzung für die alte Zeichenkette. Will man "UNIX" ersetzen durch "UNIX-Vorlesung", kann man schreiben:

```
s/UNIX/&-Vorlesung/
```

- Reicht eine Zeile nicht zur Eingabe, kann die Befehlszeile mit \ beendet und in der folgenden Zeile weitergeschrieben werden (Gilt auch für andere Anweisungen).

5.1.3 Reguläre Ausdrücke

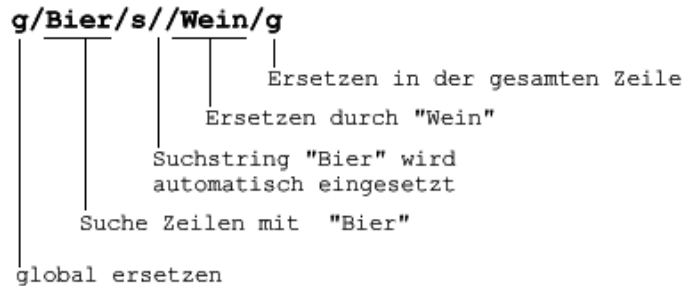
Beim Adressieren einer Zeile und beim Ersetzen wird eine sehr mächtige Möglichkeit durch die sogenannten regulären Ausdrücke geboten. Hier wird ein ähnlicher Mechanismus geboten, wie er schon bei der Auswahl von Dateien in der Shell besprochen wurde - die Möglichkeiten gehen aber sehr viel weiter. Die einfachste Form ist eine Zeichenkette als Suchmuster.

/string/	adressiert die nächste Zeile, die 'string' enthält (rückwärts suchen mit ?string?)
^	steht für den Zeilenbeginn. /^Meier/ adressiert Zeile, die mit "Meier" beginnt
\$	steht für das Zeilenende. /Meier\$/ adressiert Zeile, die mit "Meier" endet. /^\$/ adressiert die nächste Leerzeile (Achtung! Doppelbedeutung! Im Suchmuster eines reg. Ausdrucks Zeilenende, als Adresse im ed-Kommando Dateiende).
[]	definiert einen Buchstaben aus dem Bereich in []. Beginnt der Bereich mit ^, wird nach einem Zeichen gesucht, das nicht im Breich enthalten ist (Bei Dateinamen war dies das !-Zeichen). [ABC]: einer der Buchstaben A, B oder C [A-Z]: Großbuchstaben [A-Za-z]: alle Buchstaben [^0-9]: keine Ziffer
.	der Punkt steht für ein beliebiges Zeichen (wie ? in der Shell),
*	der Stern "*" steht für eine beliebige Folge des vorhergehenden Zeichens (auch Null Zeichen!). a*: Leerstring oder beliebige Folge von a's aa*: eine beliebige Folge von a's (mindestens eines) [a-z]*: Leerstring oder eine beliebige Folge von Kleinbuchstaben [a-z][a-z]*: eine beliebige Folge von Kleinbuchstaben (mindestens einer) . *: jede beliebige Zeichenfolge
\	hebt den Metazeichen-Charakter für das folgende Zeichen auf a* steht für Leerstring oder beliebige Folge von a's a* steht für die Zeichenfolge "a*"
\(\)	Reguläre Ausdrücke können mit Klammern gruppiert werden. Damit die Klammern nicht als Zeichenkette aufgefaßt werden, muß ein \ davor stehen, also schreibt man in der Regel immer "\(" und "\)". \([A-Za-z]*\) gruppiert beispielsweise ein Wort aus beliebig vielen Buchstaben, wobei auch ein leeres Wort (0 Buchstaben) dazugehört. Soll das Wort mindestes einen Buchstaben enthalten, muß man \([A-Za-z][A-Za-z]*\) schreiben. Die Anwendung solcher Gruppen wird weiter unten gezeigt - es kann in den Editor-Befehlen nämlich Bezug auf die Gruppen genommen werden.
\i	Referenzieren des i-ten Klammersausdrucks (siehe Beispiele weiter unten)

5.1.4 Textersetzung (s-Anweisung), Teil 2

Reguläre Ausdrücke können an zwei Stellen verwendet werden, wie schon erwähnt zur Adressierung von Zeilen und beim Ersetzen von Zeichenketten. Mit regulären Ausdrücken läßt sich der Ersetzungsbefehl erweitert verwenden. Zuerst ein einfaches Beispiel:

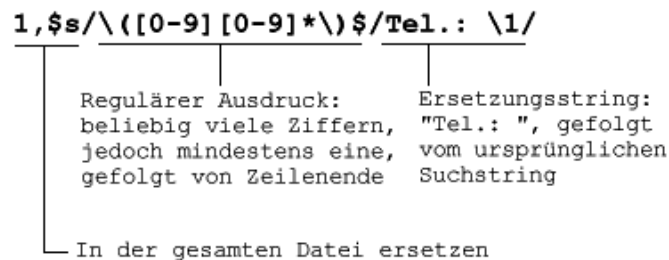
Ersetzen des Wortes "Bier" durch "Wein" im gesamten Text:



In den folgenden Beispielen wird von einer Telefonliste ausgegangen, die aus Namen, Vornamen und Telefonnummer besteht:

```
1,$p
Anders Helga 781
Huber Karl 123
Meier Hans 231
Schulze Maria 256
Weber Klaus 400
```

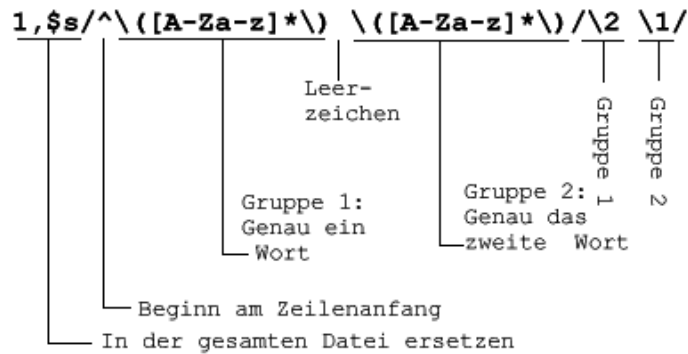
Die Telefonnummern sollen um dem Text "Tel.: " ergänzt werden. Dazu wird eine weiteres Feature der s-Anweisung verwendet. Auf den regulären Ausdruck kann durch \n Bezug genommen werden. Dabei ist n die Nummer der Gruppe (runde Klammern) im regulären Ausdruck, der den Suchstring definiert.



Das Ergebnis sieht dann so aus:

```
Anders Helga Tel.: 781
Huber Karl Tel.: 123
Meier Hans Tel.: 231
Schulze Maria Tel.: 256
Weber Klaus Tel.: 400
```

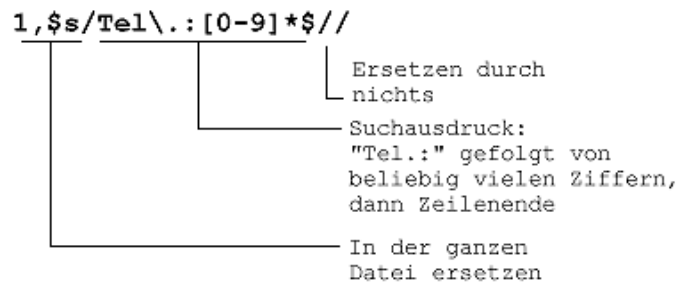
Jetzt sollen Nachname und Vorname vertauscht werden (beachten Sie die Leerzeichen zwischen den Gruppen)



Das Ergebnis sieht dann so aus:

```
Helga Anders  Tel.: 781
Karl Huber   Tel.: 123
Hans Meier   Tel.: 231
Maria Schulze Tel.: 256
Klaus Weber  Tel.: 400
```

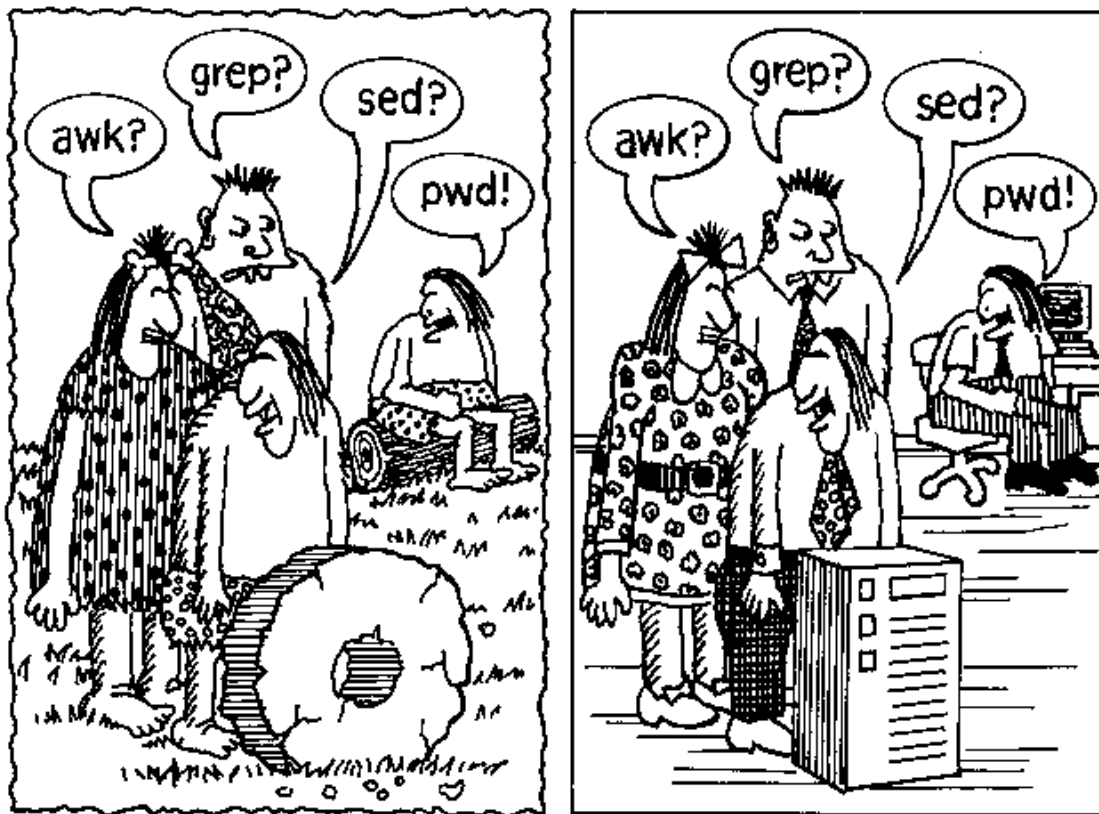
Zum Schluß werden die Telefonnummern aus der Liste entfernt:



Das Ergebnis sieht dann so aus:

```
Helga Anders
Karl Huber
Hans Meier
Maria Schulze
Klaus Weber
```

5.2 grep



grep [Optionen] reg. Ausdruck [Dateiname(n)] .

'grep' steht für 'global regular expression print'. Das Kommando ist ein Tool zum Suchen von Strings, die mit regulären Ausdrücken definiert sind, in Dateien. Die gefundenen Zeilen werden auf die Standardausgabe ausgegeben. Das Kommando arbeitet wie die ed-Anweisung g/reg. Ausdruck/p.

Wegen der Angabe des regulären Ausdrucks in der Befehlszeile muß der reguläre Ausdruck in " " oder ' ' eingeschlossen werden, wenn er Leerzeichen oder Sonderzeichen enthält, die von der Shell ersetzt werden. Wurde keine Datei angegeben, liest grep von der Standardeingabe. Optionen:

- q (quiet) Nur Return-Wert zurückgeben.
- n Zeilennummern mit ausgeben.
- c Nur die Anzahl der Zeilen ausgeben, für die der reguläre Ausdruck erfüllt ist.
- l Nur die Namen der Dateien ausgeben, in denen etwas gefunden wurde.
- i Groß-/Kleinschreibung nicht unterscheiden.
- v Alle Zeilen ausgeben, in denen der reguläre Ausdruck **nicht erfüllt ist**.

Beispiele:

Suchen nach allen Meiers, Maieres, Meyers, Mayers in einer Datei: `grep 'M[ae][iy]er'`
Namen

Suchen nach allen Zeilen, die mit dem Wort "Beispiel" beginnen; Ausgabe der Zeilennummern: `grep -n '^Beispiel'` Vorlesung

Wichtig ist grep für den Systemverwalter, z. B. einen Login-Namen in der Benutzerdatei suchen (damit nicht derselbe Name zweimal verwendet wird):

```
grep "^Klaus:" /etc/passwd
```

Bei grep wird auch vielfach nur der Rückgabewert des Programms (nicht die Ausgabe!) verwendet. In Skript-Konstruktionen (siehe später) kann das beispielsweise so aussehen:

```
if grep -q "^Klaus:" /etc/passwd ; then .....
```

5.3 sort

sort [Optionen] [Dateiname (n)]

Sortieren von Text-Dateien. Die Ausgabe geht an die Standardausgabe. Wenn mehrere Dateien angegeben sind, werden diese Dateien gemeinsam sortiert, ist keine Datei angegeben, wird von der Standardeingabe gelesen. Optionen:

- c nur prüfen ob die Dateien sortiert sind
- u reduziert mehrere identische Zeilen auf eine einzige (wie `sort ... | uniq`)
- d lexikographisch sortieren (Buchstaben, Ziffern, Leerzeichen, Tabulatorzeichen)
- f Groß-/Kleinschreibung ignorieren
- i nichtdruckbare Zeichen ignorieren
- r absteigend sortieren
- n numerisch sortieren

Beim Aufruf von sort können Sortierschlüssel angegeben werden, die festlegen, nach welchem Feld oder welchen Feldern sortiert wird. Als Feld wird dabei eine Zeichenkette betrachtet, die von Leerzeichen, Tabulator, etc. begrenzt ist. Die Sortierschlüssel sind folgendermaßen aufgebaut:

- +m[n] Beginn des Sortierbereichs. m Felder und n Zeichen überspringen - es wird also beim n+1. Zeichen im m+1. Feld begonnen (Voreinstellung für n: 0).
- m[n] Ende des Sortierbereichs. Nicht mehr dazu gehören alle Zeichen ab dem n. Zeichen (einschl. Trennzeichen) nach den m. Feld (Voreinstellung für n: 0).

Beispiele zum Sortierschlüssel:

- +2 UNIX ist *ein einfaches Betriebssystem.*
- +3.2 UNIX ist ein *einfaches Betriebssystem.*
- +2 -3 UNIX ist *ein einfaches Betriebssystem.*

3.4 Der Stream-Editor sed

sed liest aus der angegebenen Eingabedatei (normalerweise Standardeingabe) Zeile für Zeile in seinen Eingabepuffer, führt die Edieranweisungen auf die Zeile aus und schreibt sie auf die Standardausgabe. sed beendet seine Arbeit, wenn das Ende der Eingabedatei erreicht ist oder explizit eine Beendigungsanweisung erreicht wurde. Ein zweiter Puffer, der Haltepuffer, dient zum

Zwischenspeichern von Ergebnissen. Aufruf:

```
sed [-n] [-e 'sed-Anweisungen'] [-f SKriptdatei]
[eingabedatei (en) ]
```

Der sed ist ein nicht-interaktiver Editor zur Dateibearbeitung. Da der sed nicht interaktiv arbeitet, muß die Befehlsfolge zum Editieren bereits beim Aufruf festliegen. Die angegebene Textdatei ist die Eingabedatei, deren Inhalt durch die Editieranweisungen (Skript) modifiziert und dann auf die Standardausgabe ausgegeben wird. Wird keine Eingabe-Datei angegeben, liest sed seine Eingabe von der Standardeingabe. Mit der Option `-f Skriptdatei` entnimmt der sed die Editieranweisungen der Skriptdatei. Beachten Sie, daß die Eingabe-Datei selbst nicht verändert wird! Soll das Ergebnis der sed-Anweisung in einer (anderen) Datei gespeichert werden, dann muß die Standardausgabe in diese Datei umgelenkt werden. Optionen:

- n Die Standardausgabe wird unterdrückt (sinnvoll im Zusammenhang mit Pipes).
- e Es werden die auf die Option `-e` folgenden Anweisungen für die Bearbeitung der Eingabedatei verwendet. Bei mehr als einer Anweisung müssen die Anweisungen durch Semikolon voneinander getrennt werden. Das Semikolon muß dabei ohne Leerzeichen direkt hinter der Anweisung stehen! Die `-e` Option kann mehrfach angegeben werden. Um Fehlinterpretationen der Shell zu vermeiden, sollte die Editieranweisung grundsätzlich in Hochkomma oder Gänsefüßchen eingeschlossen werden.
- f Wird diese Option angegeben, dann liest sed seine Editieranweisungen aus der angegebenen Datei. Die Anweisungen müssen entweder durch Semikolon getrennt werden oder jede Anweisung muß in einer eigenen Zeile stehen. Leerzeichen hinter einer Anweisung sind nicht erlaubt. Die Anweisungen dürfen nicht in Hochkommata eingeschlossen werden. Durch mehrfache Angabe der `-f` Option können mehrere Skriptdateien zugewiesen werden.

sed kennt dieselben Anweisungen wie der interaktive Editor ed. Auch die Adressierung (Zeilennummern, reguläre Ausdrücke) erfolgt auf die gleiche Weise. Editieranweisungen haben die folgende Form:

```
[Adresse1 [,Adresse2 ] ] Funktion [Argumente]
```

Für jede Zeile der Eingabedatei werden alle Anweisungen ausgeführt. Durch Angabe einer Zeile (Adresse1) oder eines Zeilenbereichs (Adresse1, Adresse2) kann man deren Wirkung jedoch einschränken. Anstelle von Zeilennummern können auch reguläre Ausdrücke verwendet werden:

```
[/Muster/] Funktion [Argumente]
[/Muster1/ [/Muster2/]] Funktion [Argumente]
```

Im ersten Fall wird jede Zeile, die die Zeichenkette Muster enthält, von der Anweisung bearbeitet, im zweiten Fall alle Bereiche der Eingabedatei, die mit einer Zeile, die Muster1 enthält, beginnen und mit einer Zeile, die Muster2 enthält, enden.

Zusätzlich hat der sed noch weitere Funktionen (z.B. Test- und Sprungfunktionen, Klammerung, Wiederholungen) und er eignet sich daher besonders für Shell-Skripts. Interessant sind beim sed die Klammern:

- **() runde Klammern:**

Die runden Klammern müssen mit `\` geschützt werden. Sie legen die Gruppierung der einzelnen Komponenten fest. Zum Beispiel:

`AB*C` deckt die Strings "AC", "ABC", "ABBC" usw. ab.

`\(AB)*C` deckt die Strings "C", "ABC", "ABABC" usw. ab.

• { } geschweifte Klammern

Die geschweiften Klammern müssen mit \ geschützt werden. Sie legen Wiederholungen fest. Im folgenden stehen Z für ein Zeichen und M und N für Zahlen zwischen 0 und 253.

1. genau M Wiederholungen: Z\{M\} z.B. alle Namen mit genau 8 Zeichen Länge:
`sed -n '/^\.{8\}/p' namen`
2. mindestens M Wiederholungen: Z\{M, \}
3. zwischen N und M Wiederholungen: Z\{N, M\}

Die folgende Tabelle faßt die sed-Kommandos knapp zusammen.

a\ (append lines) text	text wird nach der aktuellen Eingabezeile auf die Standardausgabe geschrieben. Besteht der text aus mehreren Zeilen, so muß das Fortsetzungszeichen \ am Zeilenende vor RETURN angegeben werden.
b [marke] (branch to label)	Es wird zu der marke (in der Form :marke angegeben) des sed-Skripts gesprungen und dort die Abarbeitung des Skripts fortgesetzt. Fehlt die Angabe der Marke, so wird an das Skriptende gesprungen, was bewirkt, daß eine neue Eingabezeile gelesen und das sed-Skript von Beginn an mit dieser neuen Eingabezeile wieder ausgeführt wird.
c \ (change lines) text	Der Inhalt des Eingabepuffers wird durch text ersetzt. Besteht text aus mehreren Zeilen, so muß das Fortsetzungszeichen \ am Zeilenende vor RETURN angegeben werden.
d (delete lines)	Der Inhalt des Eingabepuffers wird gelöscht (nicht ausgegeben), und das sed-Skript wird sofort wieder von Beginn an mit dem Lesen einer neuen Eingabezeile gestartet.
D (Delete lines part of pattern space)	Der erste Teil des Eingabepuffers (bis zum ersten Zeilenende) wird gelöscht (nicht ausgegeben), und das sed-Skript wird sofort wieder von Beginn an mit dem restlichen Eingabepuffer gestartet. Pattern space ist der Eingabepuffer.
g (get contents of hold area)	Der Inhalt des Eingabepuffen wird durch den Inhalt des Haltepuffers (hold area) überschrieben.
G (Get contents of hold area)	Der Inhalt des Haltepuffers wird am Ende des Eingabepuffers (mit Newline-Zeichen getrennt) angehängt.
h (hold pattern space)	Der Inhalt des Haltepuffers wird durch den Inhalt des Eingabepuffers überschrieben.
H (Hold pattern space)	Der Inhalt des Eingabepuffers wird am Ende des Haltepuffers (mit Newline-Zeichen getrennt) angehängt.
ì\ (insert lines) text	Der text wird vor der aktuellen Eingabezeile auf die Standardausgabe geschrieben. Besteht der text aus mehreren Zeilen, muß das Fortsetzungszeichen \ am Zeilenende vor RETURN angegeben werden.
l (list pattern space on the standard output)	Der Inhalt des Eingabepuffers wird auf die Standardausgabe geschrieben, wobei nicht druckbare Zeichen durch ihren ASCII-Wert (2-Ziffer) ausgegeben werden. Überlange Zeilen werden als mehrere einzelne Zeilen ausgegeben.
n (next line)	Der Eingabepuffer wird auf die Standardausgabe ausgegeben, dann wird die nächste Eingabezeile in den Eingabepuffer gelesen.
N (Next line)	Die nächste Eingabezeile wird an den Eingabepuffer (mit Newline-Zeichen getrennt) angehängt; die aktuelle Zeilennummer wird

Betriebssystem UNIX/Linux

	hierbei weitergezählt.
p (print)	Der Eingabepuffer wird auf die Standardausgabe ausgegeben.
P (Print first part of the pattern space)	Der erste Teil des Eingabepuffers (einschließlich des ersten Newline) wird auf die Standardausgabe ausgegeben.
q (quit)	Nach Ausgabe des Eingabepuffers (nicht bei Option -n) wird zum Skriptende gesprungen und die Skriptausführung beendet.
r datei (read the contents of a file)	Es wird die Datei datei gelesen und ihr Inhalt auf die Standardausgabe ausgegeben, bevor die nächste Eingabezeile gelesen wird. Zwischen r und datei muß genau ein Leerzeichen sein.
s/regulärer Ausdruck/text/[flags] (substitute)	<p>Im Eingabepuffer werden die Textstücke, die durch den regulären Ausdruck abgedeckt sind, durch den String text ersetzt. Anstelle des Trennzeichens / kann jedes beliebige Zeichen verwendet werden. Die <i>flags</i> legen fest, wie der Ersetzungsprozeß durchgeführt werden soll. Es kann folgendes angegeben werden:</p> <ul style="list-style-type: none"> • n: Es wird nur das n-te Teilstück ersetzt; n muß zwischen 1 und 512 liegen. • g (global): Es werden alle passenden Textstücke, die sich nicht überlappen, ersetzt. • p (print the pattern space if a replacement was made): Falls eine Ersetzung stattfand, dann wird der veränderte Eingabepuffer ausgegeben. • w datei (write the pattern space to a file if a replacement was made): Falls eine Ersetzung stattfand, dann wird der veränderte Eingabepuffer ans Ende der Datei <i>datei</i> geschrieben. Fehlt die Angabe von flags, so wird nur das erste passende Textstück des Eingabepuffers ersetzt. Es dürfen mehrere der vorgestellten flags-Angaben gleichzeitig verwendet werden, wobei dann g (falls verwendet) an erster Stelle stehen muß.
t [marke] (test substitutions):	Falls seit dem letzten Lesen einer Eingabezeile oder der Ausführung einer t-Funktion eine Ersetzung stattfand, dann wird zu der marke (in der Form :marke angegeben) des sed-Skripts gesprungen und dort die Abarbeitung des Skripts fortgesetzt; im anderen Fall hat die Angabe dieser Funktion keine Auswirkung. Fehlt die Angabe der Marke, so wird an das Ende des Skripts gesprungen.
w datei (write to a file)	Es wird der Eingabepuffer an das Ende der Datei datei geschrieben. Zwischen w und datei muß genau ein Leerzeichen sein.
x (exchange)	Der Inhalt des Eingabepuffers wird mit dem Inhalt des Haltepuffers vertauscht.
y/string1/string2/	Im Eingabepuffer werden alle Zeichen, die in string1 vorkommen, durch die an gleicher Stelle in string2 stehenden Zeichen ersetzt. string1 und string2 müssen gleich lang sein.
!funktion	Die angegebene Funktion (oder Gruppe von Funktionen bei Klammerung mit { }) wird nur für Zeilen ausgeführt, für die die angegebene Adreßangabe nicht zutrifft.
:marke	Definiert eine Sprungmarke für die Funktionen b und t.
"	

	Schreibt die aktuelle Zeilennummer als eigene Zeile auf die Standardausgabe.
{...}	Klammert eine Gruppe von Funktionen, die nur ausgeführt werden, wenn der Eingabepuffer gefüllt ist.
#	Dieses Zeichen leitet, wenn es als erstes Zeichen in einer Zeile angegeben wird, mit einer Ausnahme einen Kommentar ein. Die Ausnahme ist, wenn direkt nach # das Zeichen n angegeben wird; in diesem Fall wird die automatische Ausgabe, wie bei der Kommandozeilen-Option -n, ausgeschaltet. Der Rest der Zeile nach #n wird ebenfalls ignoriert. Eine Skript-Datei muß mindestens eine Nicht-Kommentarzeile enthalten. Eine leere Funktion wird ignoriert.



[Zum Inhaltsverzeichnis](#)



[Zum nächsten Abschnitt](#)

Copyright © FH München, FB 04, Prof. Jürgen Plate

Letzte Aktualisierung: 15. Jan 2010



Vorlesung "UNIX"

von Prof. Jürgen Plate

6. Bildschirmeditor vi

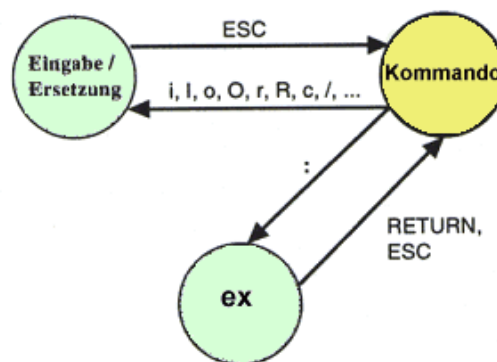
Der Editor vi ("vi" steht für "visual") ist ein bildschirmorientierter Editor, d. h. der Text ist in seiner aktuellen Version auf dem Bildschirm zu sehen. Es erfolgt hier nur ein Überblick der Möglichkeiten des vi. Programmaufruf:

vi Dateiname

Ist die Datei vorhanden, wird sie in den Editorpuffer geladen, andernfalls wird sie neu angelegt. Nach dem Aufruf des vi befindet sich der Benutzer im Kommandomodus. Die Arbeit spielt sich wie beim ed immer in zwei Ebenen ab:

- Kommandomodus: Freies Positionieren innerhalb des Textes, Umsetzen von Textblöcken, Schreiben und Lesen von Dateien, Löschen von Textblöcken und Aufruf von UNIX-Kommandos.
- Eingabemodus: Einfügen von Text oder Überschreiben vorhandener Textpassagen.

Das Umschalten zwischen den Modi erfolgt mit der ESC-Taste ([ESC]). Die Handhabung des Kommandomodus des vi macht die Bedienung für den Anfänger etwas ungewohnt. Die Cursorpositionierung erfolgt mit verschiedenen Tasten, für die Eingabe von Dateibefehlen oder anderen Funktionen (z. B. Suchen/ersetzen) ist die Eingabe eines Doppelpunktes nötig. Man befindet sich dann in ex-Modus, der ähnliche Eigenschaften hat, wie der ed.



Bleiben wir gleich bei den ex-Befehlen. Nach der Eingabe des `:` springt der Cursor in die letzte Zeile und erlaubt die Eingabe eines ex-Befehls, der mit der Return-Taste abgeschlossen wird. Es sind nach dem Doppelpunkt alle ed-Befehle möglich. Für den Einstieg genügen:

<code>:w</code>	Text auf die Platte schreiben
<code>:w!</code>	Text auf die Platte schreiben, Schreibschutz ignorieren
<code>:wq</code>	Text auf die Platte schreiben und vi beenden
<code>[ESC] ZZ</code>	wie <code>:wq</code>
<code>:x</code>	wie <code>:wq</code>
<code>:w Datei</code>	Text auf die angegebene (andere) Datei schreiben
<code>w >> Datei</code>	Text an die angegebene Datei anhängen
<code>:q</code>	Abbrechen, ohne den Text auf Platte zu schreiben. Bei Änderungen im Text erfolgt eine Warnung und kein Abbruch.
<code>:q!</code>	Abbrechen, ohne den Text auf Platte zu schreiben.
<code>:r Datei</code>	

Betriebssystem UNIX/Linux

	Text aus der angegebenen Datei nach der momentanen Zeile einfügen.
:e Datei	Neue Datei bearbeiten (der Textpuffer wird überschrieben). Ist die aktuelle Datei noch nicht gespeichert worden, erfolgt eine Warnung.
:e! Datei	Neue Datei bearbeiten (der Textpuffer wird immer überschrieben).
:n	wechselt zur nächsten geladenen Datei (falls beim Aufruf mehr als eine Datei angegeben wurde).

Umschalten in den Eingabemodus

a	(append) Eingabe rechts vom Cursor
A	Eingabe am Zeilenende
i	(insert) Eingabe links vom Cursor
I	Eingaben am Zeilenanfang
o	Eingabe ab der aktuellen Zeile, d.h. in der folgenden Zeile, Spalte 1.
O	Eingabe in der vorhergehenden Zeile, Spalte 1.

Wenn es die Hardware ermöglicht, ändert sich die Form des Cursors beim Wechsel zwischen Eingabe- und Kommandomodus. Bei vielen vi-Versionen wird der Eingabemodus unten rechts am Bildschirm angezeigt ('append mode', 'input mode');

Umschalten in den Kommandomodus

Der Eingabemodus wird durch Drücken der ESC-Taste verlassen und somit der Kommandomodus aktiv.

Blättern auf dem Bildschirm

CTRL-F	Eine Bildschirmseite vorwärts
CTRL-B	Eine Bildschirmseite rückwärts
CTRL-D	Eine halbe Bildschirmseite vorwärts
CTRL-L	Bildschirm neu aufbauen

Cursorpositionierung (im Kommandomodus)

Normalerweise ist der vi an die Pfeiltasten der Tastatur richtig angepaßt. Das muß aber nicht immer so sein. Deshalb helfen manchmal nur die folgenden Möglichkeiten (Auswahl):

h	Zeichen links (auch Backspace)
l	Zeichen rechts (auch blank)
k	Spalte höher
j	Spalte tiefer
b	Wortanfang

Betriebssystem UNIX/Linux

e	Wortende
w	Anfang nächstes Wort
H	Anfang erste Zeile des Bildschirms
L	Anfang letzten Zeile des Bildschirms
0	(Null) Zeilenanfang
\$	Zeilenende
RETURN	nächste Zeile
nG	(Go) Gehe zu Zeile n. n ist eine Zeilennummer. Fehlt die Zahl, wird zur letzten Zeile der Datei gesprungen.
%	sucht zur aktuellen Klammer die korrespondierende öffnende bzw. schließende Klammer

Löschen von Text (im Kommandomodus)

x	Zeichen unter dem Cursor löschen
X	Zeichen vor dem Cursor löschen
dw	ab Cursorposition bis Wortende löschen
db	ab Cursorposition bis Wortanfang lö
[n] dd	n ganze Zeilen löschen. Falls n fehlt, wird eine Zeile gelöscht.
D	ab Cursorposition bis Zeilenende löschen

Text ändern:

Änderungskommandos wechseln automatisch in den Eingabemodus, der mit der ESC-Taste verlassen werden muß.

r	Zeichen ersetzen (kein Abschluß mit ESC)
~	Wechsel Groß-/Kleinbuschst. (kein Abschluß mit ESC)
R	Mehrere Zeichen ersetzen
s	Ein Zeichen ersetzen und zusätzliche Zeichen anschließend einfügen
cc	Gesamte Zeile ändern
cw	Wort ändern
C	ab Cursorposition bis Zeilenende ändern

Suchen nach Zeichen oder Strings

fx	Zeichen "x" in der momentanen Zeile suchen (vorwärts)
Fx	Zeichen "x" in der momentanen Zeile suchen (rückwärts)
Die folgenden Suchkommandos müssen mit der Return-Taste abgeschlossen	

werden	
/str	String "str" vorwärts in der Datei suchen
?str	String "str" rückwärts in der Datei suchen
//	Letzten Suchbefehl wiederholen (vorwärts)
??	Letzten Suchbefehl wiederholen (rückwärts)

Sonstige Kommandos

.	Letztes Kommando wiederholen
J	Zeilen verbinden (nächste Zeile anhängen)
u	Letzten Befehl rückgängig machen
U	Aktuelle Zeile wiederherstellen
CTRL-g	Zeilennummer und Dateiinfo zeigen
CTRL-v	Nächstes Zeichen transparent eingeben (z. B. Steuerzeichen)
!: <i>cmd</i>	Verläßt vi temporär und führt das Kommando <i>cmd</i> aus (ggf. Rückkehr mit exit).
:r! <i>cmd</i>	Fügt die Ausgabe von <i>cmd</i> ab Cursorposition ein.
!! <i>cmd</i>	benutzt aktuelle Zeile als Eingabe für <i>cmd</i> und ersetzt durch dessen Ausgabe.

Zum Kopieren dient der **y-Befehl** (yank), mit dem Textteile in einen Puffer kopiert werden. "y" wird in der Regel mit einem zweiten Steuerbuchstaben kombiniert, der angibt, wieviel Text gepuffert wird (z. B. "yw", "y\$"). Eine ganze Zeile wird mit "yy" oder "Y" kopiert. Vor "yy" kann noch die Anzahl der zu kopierenden Zeilen stehen (z. B. 12yy). Das Einfügen des Kopierten Textes geht dann mit **p** (vor dem Cursor) oder **P** (nach dem Cursor).

Zusätzlich bietet der vi weitere Steuerungsfunktionen für den Editor und einen Makromechanismus. Kommandofolgen können damit durch ein einziges Befehlswort abgerufen werden. Weitere Infos auf den vi-Manualpages.

Weitere Techniken:

- Zeile verschieben:
 Zeile mit dd löschen
 Zur neuen Position wechseln
 Zeile mit p (vor dem Cursor)
 oder P (nach dem Cursor) einfügen
- Zeile kopieren:
 Zeile mit yy in den Puffer kopieren
 Zur neuen Position wechseln
 Zeile mit p (vor dem Cursor)
 oder P (nach dem Cursor) einfügen
- Ein paar einfache Tricks:
 zwei aufeinanderfolgende Zeichen vertauschen: xp
 zwei aufeinanderfolgende Zeilen vertauschen: ddp
 eine Zeile duplizieren: yyp

Betriebssystem UNIX/Linux

- Mehrere Zeilen verschieben oder kopieren:
Mehrere Zeilen werden durch Voranstellen der Zeilenzahl vor "dd" oder "yy" in den Puffer gebracht. Alles weitere wie oben.
- Weitere Puffer:
Neben dem anonymen Puffer können 26 weitere Pufferbereiche verwendet werden, die durch einen Buchstaben ("a" - "z") bezeichnet werden. Dazu wird ein Doppelpostroph und der Puffername vor den Befehl gestellt, z. B.:

"ayy	kopiere Zeile in Puffer a
"b6yy	kopiere die nächsten 6 Zeilen in Puffer b
"ap	Inhalt Puffer a nach dem Cursor einfügen

- Blöcke von einer Datei in die andere übertragen:
Mit dem Kommando `:e <dateiname>` kann eine neue Datei bearbeitet werden, ohne den vi zu verlassen.
vi mit der ersten Datei aufrufen, z. B.:
`vi foo.txt`
Textblock mit den Yank-Kommando in einen Puffer schreiben:
`"f10yy` (10 Zeilen in Puffer f)
neue Datei holen, z. B.:
`:e bar.txt`
an die gewünschte Stelle gehen und Puffer einfügen:
`"fp`
- Weitere Steuerfunktionen für reguläre Ausdrücke:
Im vi gibt es noch zwei Steuerfunktionen für RA, die sich auf den Anfang oder das Ende eines Wortes beziehen (Wortanfang/-Ende wird durch Leerzeichen oder Satzzeichen identifiziert).

\<	Markiert den Wortanfang bei der Mustersuche
\>	Markiert das Wortende bei der Mustersuche

- Blöcke selektieren:
Da bei allen ex-Kommandos reg. Ausdrücke als Positionsbestimmung möglich sind, können auch Blöcke über Suchbegriffe selektiert werden. Sei beispielsweise folgender Text gegeben:

```
Das Betriebssystem UNIX blickt auf eine lange Vergangenheit zurück. Ursprünglich wurde es für PDP-Rechner der Firma DEC entwickelt und sollte hauptsächlich die Bedürfnisse von professionellen Softwareentwicklern befriedigen. Heute ist UNIX auf Workstations mit 64-Bit-Prozessoren zugeschnitten. Zu den Neuerungen gehören die Einführung von Threads, mit denen ein Programm mehrere Funktionen gleichzeitig ausführen kann. Ebenfalls neu sind Echtzeitfunktionen mit denen Programmlaufzeiten vorhersagbar sind. Insbesondere für Datenbanken ist die Möglichkeit, sehr große Dateien verarbeiten zu können, von nicht unerheblicher Bedeutung.
```

Die beiden letzten Zeilen sollen nach vorne verschoben werden.
`:/Echtzeitfunktionen/,/Bedeutung/ mo /UNIX/`

Betriebssystem UNIX/Linux

- Weitere Beispiele für globales Suchen und Ersetzen:
Das Leerzeichen wird in den Beispielen als "_" dargestellt. Das Prozentzeichen (%) ist eine Abkürzung für !,\$ - also für die Bearbeitung der ganzen Datei.
 - a. Ersetzen Punkt durch Semikolon in den ersten 10 Zeilen:
`:1,10s/\./;/g`
 - b. Alle Worte "Hilfe" oder "hilfe" durch "HILFE" ersetzen:
`:%s/[Hh]ilfe/HILFE/g`
 - c. aufeinanderfolgende Leerzeichen durch ein einziges ersetzen
`:%s/_ _*/_/g`
 - d. alle Leerzeilen löschen:
`:g/^$/d`
 - e. alle Zeilen löschen, die entweder leer sind oder nur Leerzeichen enthalten:
`:g/^_*$$/d`
 - f. Wie wir gesehen haben, können geklammerte Terme durch Backslash und die Nummer des Ausdrucks referiert werden. Der folgende Ausdruck ersetzt alle führenden Leerzeichen aller Zeilen:
`%s/^_ _*\ (.*) \1/`
 - g. Einfügen von 3 Leerzeichen am Anfang jeder Zeile:
`:%s/^/_ _ _/`
 - h. Die Reihenfolge zweier Texte in einer Zeile vertauschen, die durch einen Gedankenstrich getrennt sind; so soll aus `more - Dateien anzeigen` die Zeile `Dateien anzeigen - more` entstehen. Zur besseren Verständlichkeit ist der Ausdruck durch Leerstellen strukturiert, die man bei der Eingabe natürlich weglassen muß (das Leerzeichen wird hier durch "_" dargestellt):
`%s / \ (.*) _ _ \ (.*) / \2 _ _ \1 /`

Makros

Makros sind nichts weiter, als die Zusammenfassung von vi-Kommandos unter einer Taste. Man kann so häufig benutzte Tastenbetätigungen auf eine Taste legen. Bei der Definition von Makros sollten Sie immer davon ausgehen, daß sich der vi im Kommandomodus befindet. Die Makros lassen sich auch dauerhaft in der Datei `.exrc` im eigenen Homedirectory speichern. Sie werden dann immer beim Aufruf des vi geladen. In `.exrc` lassen sich auch andere vi-Voreinstellungen vornehmen (siehe vi-Manualpage, `set`-Kommando). Die Definition eines Makros ist recht einfach (entweder als Zeile in `.exrc` oder im `ex`-Modus, also eingeleitet durch Doppelpunkt:

```
map <Makroname> <Kommandofolge>
```

Mit `map` ohne Parameter gibt es eine Liste der Makros und mit `unmap` kann ein Makro gelöscht werden. Steuerzeichen (RETURN, ESC, BACKSPACE, etc.) können natürlich nicht direkt eingegeben werden, sondern als `CTRL-V <Stuerzeichen>`. In den Beispielen unten werden die Steuerzeichen beispielsweise als `<CTRL-X>` dargestellt. Zum Beispiel kann man den Text "Mit freundlichen Grüßen" auf den Tastencode "fg" legen, indem man mit "i" in den Eingabemodus wechselt, den Text angibt und dann wieder in den Kommandomodus geht:

```
map fg iMit freundlichen Grüßen<ESC>
```

Um das ESC-Zeichen einzugeben, tippt man `<CTR-V><ESC>`. Man könnte auf diese Weise auch die gewohnten PC-Tasten (in der entsprechenden Terminal-Emulation) auf die vi-Kommandos legen.

```
" Taste einf  
map <ESC>[2~ i  
" Taste del
```

```
map <ESC>[3~ x
" Taste pos1
map <ESC>[1~ ^
" Taste end
map <ESC>[4~ $
" Taste PgUp
map <ESC>[5~ <CTRL-B>
" Taste PgDown
map <ESC>[6~ <CTRL-F>
```

Literaturhinweise

L. Lamb:
Learning the vi Editor
O'Reilly & Ass. (1990)

S. von Bomhard:
Hartes Brot - vi-Tips für Neugierige
iX, 4/92, S. 115-119, Heise Verlag

Manualpage zum vi (`man vi`)

Im Anhang finden Sie noch eine vi-Kurzreferenz.



[Zum Inhaltsverzeichnis](#)



[Zum nächsten Abschnitt](#)

Copyright © FH München, FB 04, Prof. Jürgen Plate

Letzte Aktualisierung: 21. Sep 2011



Vorlesung "UNIX"

von Prof. Jürgen Plate

7. Kommunikationsdienste

UNIX stellt eine Reihe von Diensten zur Kommunikation der Benutzer untereinander zur Verfügung. Einen Dienst haben Sie schon kennengelernt: write. Dieser Befehl hat noch einen nahen Verwandten, der jedoch meist nur vom Systemverwalter verwendet wird:

wall

Dieses Kommando liest einen Text von der Standardeingabe und sendet ihn an alle angemeldeten Benutzer. Es darf nur von root verwendet werden.

Es gibt daneben die Möglichkeit, Benutzer auch dann zu informieren, wenn sie nicht eingeloggt sind.

- Die Datei `/etc/motd` (Message of the day) wird beim Login angezeigt. Manche haben dort einen belanglosen Text stehen, der sich nie ändert. Sinnvoller ist es jedoch, diese Datei für wirklich wichtige Meldungen zu verwenden. Bei Ubuntu wird diese Datei von einigen Systemdiensten automatisch verädert.
- Bei manchen Rechnern bekommt man auch bei jedem Login einen mehr oder weniger lustigen Spruch präsentiert. Hier hat dann meist der Systemverwalter einen Programmaufruf von 'fortune' in die `/etc/profile` aufgenommen. Dieses Programm sucht sich einen zufälligen Spruch aus einer großen Datenbank heraus.
- in die Datei `/etc/profile` kann der Administrator natürlich noch beliebige andere Informationssysteme einbauen.

7.1 Das UNIX Mailsystem

Man kann aber auch Benutzer erreichen, die gerade nicht am Rechner arbeiten. Mit dem Mail-System kann elektronische Post an andere Benutzer verschickt werden. Zu diesem Zweck gibt es das Verzeichnis `/var/spool/mail` (früher `/usr/mail` oder `/var/mail`), in dem für jeden Benutzer, der Post erhalten hat, eine Datei existiert, deren Name mit dem Loginnamen des Benutzers übereinstimmt. Mit den Kommandos `mail` oder der Erweiterung `mailx` kann Post an andere Benutzer verschickt werden und die empfangene elektronische Post bearbeitet werden. An dieser Stelle wird die lokale Nutzung des Mail-Systems behandelt, später dann die Anwendung im Netz. Übrigens - mit 'Mail' meine ich immer die elektronische Mail von Computer zu Computer. Zunächst das primitivste aller Mail-Programme:

Post versenden `mail Login-Name (n)`

Versenden von Post an einen oder mehrere andere Benutzer. Bei mehreren Namen sind diese durch Leerzeichen zu trennen. Mail liest den Text von der Standardeingabe. Der Text kann nicht nur mit CTRL-D (End of File) sondern auch mit einer Zeile beendet werden, die nur einen Punkt enthält.

Das Mail-Kommando ergänzt den Brief um einen Briefkopf, in dem Absender und Absenzeitpunkt verzeichnet sind. Der Mailkopf enthält einzelne Zeilen, die sich anhand eines Schlüsselwortes identifizieren lassen ("From", "To", usw.). Zum Versand an ferne Rechner käme dann noch ein "Briefumschlag" hinzu. Normalerweise sorgt das Mail-Kommando nicht selbst für den Versand, sondern es übergibt den Brief an ein anderes, speziell für den Mailversand konzipiertes Programm.

Wir haben es hier also mit einem Frontend, dem 'Mail User Agent' (`mail`, `mailx`, `elm`, `pine`,...), und einem Backend, dem 'Mail Transport Agent' (`sendmail`, `smail`, ...), zu tun. Der MTA sorgt auch für die Weiterleitung ankommender Post an den richtigen Empfänger.

Betriebssystem UNIX/Linux

Kann der Brief nicht zugestellt werden (z. B. unbekannter Empfänger), gibt mail eine Fehlermeldung aus. Gleichzeitig erhält man den eigenen Brief, ergänzt um Zusatzinformationen per mail retour. Fehlerhafte Briefe werden auch (je nach Einstellung des MTA) an einen besonderen Pseudo-Empfänger namens 'postmaster' geschickt. Dieser Benutzer steht nicht in der Passwortdatei, sondern es ist ein sogenanntes 'Mail-Alias' für den Systemverwalter oder einen Beauftragten. An den 'postmaster' kann man sich auch wenden, wenn man Fragen im Zusammenhang mit Mail hat (z. B. wenn man einen Empfänger sucht). Beispiel für das Versenden einer Mail:

```
$ mail markus
Lieber Markus,

leider habe ich dich heute nicht getroffen. Ich brauche
dringend das UNIX-Buch, das ich Dir neulich geliehen habe.
Bitte leg es doch in mein Fach. Danke.

Gruss, Hans
.
```

Wenn der Benutzer markus mehrere Mails erhalten hat, wird er über diesen Brief nicht besonders glücklich sein, denn jeder Mailer (MUA) zeigt normalerweise erst einmal den Absender und eine Betreff-Zeile an. Letztere besitzt aber der obige Brief garnicht. Man sollte also eine "Subject:"-Zeile anbringen. Das geht recht gut mit dem verbesserten 'mail'-Kommando, 'mailx'. Hier kann man in der Kommandozeile ein "Subject" angeben:

```
mailx -s "UNIX-Buch" markus

....
```

'mail' und 'mailx' werden eigentlich nur noch verwendet, wenn nichts besseres da ist - oder wenn es darum geht in Kommandodateien (Shellskripts) Mail automatisch zu versenden, da man mit Ausgabeumleitung oder Pipe die Ausgabe eines Programms direkt an den MUA übergeben kann.

Eine Nachricht über eingegangene Post erhält der Benutzer beim Login mit der Meldung `you have mail`.

Er kann dann mit einem Mail-Kommando seine Post ansehen (und gleich antworten). Der Unterschied zum Senden besteht bei mail und mailx im Fehlen der Empfänger-Logins. mail arbeitet interaktiv - man kann so alle Briefe nacheinander bearbeiten. Die obige Ausgabe stützt sich auch auf das Mail-Kommando.

mail [-ehpqr] [-f datei] [-F login-name]

Bearbeiten empfangener Briefe. Die zuletzt eingegangenen Briefe werden zuerst angezeigt (LIFO).

Optionen:

- e Keine Bearbeitung, nur anzeigen, ob Post vorliegt. mail antwortet mit 0 oder 1 (=keine Post/Post).
- h Nur numerierte Liste der Briefköpfe anzeigen, danach in den interaktiven Modus gehen.
- p Alle Briefe ohne interaktive Steuerung ausgeben.
- q Abbruchtaste beendet mail (im Normalfall wird damit nur die Ausgabe des aktuellen Briefs abgebrochen).
- r Anzeige der Briefe nach Alter, den ältesten zuerst (FIFO).

-f mail bietet die Möglichkeit, Briefe in einer benutzereigenen Datei zu speichern. Statt des datei Standardnamens "mbox" wird der angegebene Name verwendet.

Beim Lesen der Post im interaktiven Modus (der Aufruf von mail ohne Parameter ist die Regel) meldet sich mail mit dem Fragezeichen als Prompt. Danach können zahlreiche Kommandos gegeben werden. Die Ausgabe kann mit CTRL-S angehalten und mit CTRL-Q fortgesetzt werden. Für den Anfang reichen ein paar Tasten:

CR,n,+ zum nächste Brief gehen

d löscht den angezeigten Brief (markiert als "gelöscht")

p aktuellen Brief nochmals ausgeben.

- vorherigen Brief nochmals ausgeben.

s Brief in der Datei mbox speichern. Hinter s kann auch ein Dateiname angegeben werden.

w wie s, jedoch ohne Briefkopf. m Brief weiterleiten. Hinter m muß ein Login-Name angegeben werden.

r Antworten, der Empfänger wird von mail dem Brief entnommen.

! Ausführen des hinter dem ! angegebenen UNIX-Kommandos.

q mail verlassen, als gelöscht markierte Briefe entfernen

x mail abbrechen - alles bleibt, wie es war.

Das Kommand 'mailx' zeigt beim Start zumindest eine Übersicht der eingegangenen Post (Subject und Absender) und bietet komfortablere Bearbeitung (z. B. Löschen aller Mails auf einmal oder die oben erwähnte Subject-Angabe).

Programme wie 'elm', 'pine' oder 'mutt' arbeiten interaktiv und stellen eine komfortablere Benutzer-Schnittstelle dar. Deshalb werden sie normalerweise verwendet. Eine detaillierte Beschreibung dieser Programme würde jedoch den Rahmen dieses Skripts sprengen.

Weiterleiten von Mails

Bei früheren Versionen von 'mail' konnte die Post an einen anderen Benutzer mit der Option "-F login-name" umgeleitet werden (z. B. für die Urlaubsvertretung). Vorher muß der Briefkasten vollständig geleert werden. Die Umleitung erlaubt natürlich auch das Weiterschicken an den eigenen Account auf fremden Rechnern (so laufen z. B. alle Mails an den Systemadministrator auf einem Rechner zusammen).

Bei allen sendmail- oder smail-basierten Systemen ist die Umleitung einfacher. Der Benutzer muß lediglich in seinem Home-Directory eine Datei namens ".forward" anlegen und in dieser Datei eine (korrekte) Mailadresse eintragen. Sollen mehrere Empfänger angesprochen werden, sind die Namen durch Kommas zu trennen.

Der Mechanismus geht jedoch noch weiter. Wird eine Datei mit vollständigem Pfad angegeben (die Weiterleitungszeile beginnt also mit einem '/'), dann landet die Post in der angegebenen Datei. Ein weiterer Schritt ist die Angabe einer Pipe in ein Programm oder Skript, das die Mail weiterverarbeitet (z. B. "| tuwas"). Beim Erstellen des Skripts ist zu beachten, daß keinerlei Pfade oder Voreinstellungen vorausgesetzt werden dürfen und gewisse Sicherheitsmaßnahmen zu beachten sind. Ein recht bekanntes Programm, das eingehende Mail vorsortieren oder unerwünschte Mail gleich löschen kann ist beispielsweise 'procmail'.

Mailsysteme, die auf 'smail' oder 'sendmail' basieren, bieten noch einige weitere Features. Die Datei /etc/aliases ist hier von besonderem Interesse, da sich damit einige Mail-Dienste realisieren lassen:

- Durch Eintrag eines Mail-Alias kann entweder eine Mailadresse dauerhaft umgeleitet werden (ohne .forward im Home-Verzeichnis des Users). Man trägt einfach den Usernamen und die

Zieladresse ein, z. B.

```
plate: plate@fh-muenchen.de
```

- Es lassen sich aber auch lokale Aliase eintragen, beispielsweise für die Adressen, die jedes System haben sollte:

```
postmaster: holzmann
webmaster: plate
admin: root
. . . .
```

- Es lassen sich auch Mailverteiler eintragen:

```
netmaster: plate,holzmann,root
```

Eine E-Mail an "netmaster" wird im Beispiel an drei verschiedene Accounts geschickt.

- Die Zieladressen des Mailverters lassen sich auch in einer Datei speichern. Diese Datei kann irgendeinem Benutzer gehören, der diese "Mailingliste" verwaltet. Eine Mailingliste ist schon optisch günstiger, als eine endlose Reihe von Adressaten im Mail-Header. Ausserdem erfährt so nicht jeder, wer noch auf dem Mailverteiler ist. Die Zeile in /etc/aliases verweist auf die Datei:

```
wichtel: :include:/home/plate/wichtel-mailingliste
```

Die Datei enthält einfach in jeder Zeile eine komplette Mailadresse. Durch Hinzufügen und Löschen von Zeilen kann die Liste aktualisiert werden.

7.2 Verbindung zu anderen UNIX-Rechnern

Der Datenaustausch mit anderen UNIX-Rechnern kann nicht nur über lokale Netze (LAN) sondern auch über Wähl- und Standleitungen der Post erfolgen (WAN). Es stehen folgenden Möglichkeiten zur Verfügung:

- Senden von Post an andere UNIX-Systeme
- Kopieren von Dateien auf und von anderen UNIX-Systemen
- Arbeiten auf anderen UNIX-Systemen
- Mounten von Dateisystemen anderer Systeme
- und vieles mehr ...

7.2.1 Am Anfang war das Modem

Zunächst wurde eine Möglichkeit gesucht, z. B. die Post über eine Kette mehrerer anderer Rechner weiterzuleiten und gegebenenfalls auch einmal Dateien zu transferieren. Es entstanden die Programme, die heute unter dem Oberbegriff 'uucp' zusammengefaßt sind. Die Abkürzung 'uucp' steht für 'Unix to Unix Copy'. Bei uucp handelt es sich um ein System von Programmen und Protokollen zum Offline-Datentransfer, d. h. Mail, Kopieraufträge oder die Ausführung von Kommandos wurde zunächst in Form von Auftragsdateien in einem Verzeichnis (meist /var/spool/uucp) abgelegt.

In regelmäßigen Zeitabständen wird dann der eigentliche Datentransfer gestartet (Programm 'uucio' = Unix to Unix Copy I/O). Per Modem wird Kontakt zu einem entfernten Rechner aufgenommen und wechselseitig Daten ausgetauscht. Ist alles erledigt, wird die Verbindung wieder unterbrochen.

Betriebssystem UNIX/Linux

Heute wird vieles online im Netz erledigt, aber uucp hat immer noch seine Daseinsberechtigung, z. B. zum Austausch von Mail und Usenet-News. Wenn jemand beispielsweise sowieso nur einmal am Tag seine Mail bearbeitet dann kann dies offline, d. h. lokal geschehen und der Transfer irgendwann Nachts oder im Lauf des Tages erfolgen. Uucp-Software gibt es auch für andere Betriebssysteme (z. B. das Paket 'Waffle' für DOS) und so ist auch ein platformübergreifender Datenverkehr möglich. Außerdem kommt man so auch in Gegenden an seine Mail, in denen nur eine Telefonverbindung verfügbar ist. Das Thema wird im Kapitel über das Internet nochmals aufgegriffen. Um erreichbar und von anderen Rechnern unterscheidbar zu sein, braucht jedes System einen eindeutigen Knotennamen. Mit deren Hilfe gab es dann auch die erste Möglichkeit, Benutzer auf anderen Rechnern zu adressieren.

uname [-amnrsv]

Ausgabe des Knotennamens und weiterer Info. Wichtig sind:
s System Name des Rechners
n Knotenname des Rechners

Das Versenden von Post an ein anderes System erfolgte anfangs durch Angabe von Knotennamen und Loginnamen des Empfängers, getrennt durch ein Ausrufezeichen, z. B.:

```
mailx -s "Konferenz an 12.7." werk2!hans
```

Die übrige Bedienung von mail erfolgt dann wie schon beschrieben. Es ist auch möglich, die Post über mehrere Knoten zu leiten, wenn zwischen dem eigenen Rechner und den Zielrechner keine direkte Verbindung besteht. Angenommen alle Rechner eines Unternehmens sind per uucp gekoppelt, dann könnte der mail-Aufruf so aussehen:

```
mailx -s "Konferenz am 12.7." werk1!werk2!werk4!klaus
```

Nachteil dieser Wegbeschreibung, die wegen der Ausrufezeichen als 'Bang-Path' bezeichnet wird, ist die Festlegung eben gerade auf einen Weg. Ist der Rechner "werk2" gerade nicht betriebsbereit, bleibt die Post hängen. Deshalb ist man schon sehr bald zu einer wegeunabhängigen Adressierung übergegangen, der an anderer Stelle beschriebenen Domain-Adressierung, die heute verwendet wird. Manchmal findet man noch Mischformen, bei denen sozusagen die letzte Wegstrecke noch als Bangpath notiert ist. Es gibt eine ganze Reihe von uucp-Kommandos:

uucp [Optionen] Quelldatei(en) Zieldatei

(Unix to Unix Copy) Kopieren von Dateien zwischen zwei Unix-Systemen. Für Quell- und Zieldatei kann dabei folgendes angegeben werden:

- Ein Pfadname auf dem lokalen System
- Kombination aus Knotenname(n) und Pfadname

Der Pfadname kann angegeben werden als:

- ~Loginname[Pfadname]
(absoluter Pfad) - uucp setzt für die Tilde ~ dann den Pfad zum Home-Directory des Benutzers ein.
- ~/Pfadname
uucp setzt für die Tilde ~ dem Pfad /usr/spool/uucppublic/ ein und hängt den angegebenen Pfad an.

Von den Optionen werden nur einige vorgestellt:

- j gibt die Auftragskennung aus

- m Sendet an der Aufrufer einen mail-Mitteilung über den Erfolg der Kopieraktion
- n Empfänger-Loginname Der Empfänger wird per mail über den Kopiervorgang informiert.

uux [-jmu] Kommandostring

(Unix to Unix command eXecution) Ermöglicht die Ausführung eines Kommandos auf einem anderen UNIX-Rechner. Die Dateien des Kommandos können auf eine ausgewählten Maschinen liegen. Der Kommandostring ist ähnlich aufgebaut wie bei uucp:

Knotenname [~Loginname] Kommando

Mit "uux Knotenname!login" kann man sich auf dem fremden Rechner anmelden, als ob man an einem seiner Terminals wäre. Dazu wird oft aber auch das folgende Kommando angeboten:

cu [Optionen] Knotenname

(Call Unix) Einloggen am Rechner mit dem angegebenen Knotenname. Es erfolgt dann ganz normal die Login-Aufforderung des entfernten Rechners - sozusagen die Minimalform eines Terminalprogramms für das Modem. cu wird heute fast nur noch verwendet, um angeschlossene Modems zu testen, Mit dem Parameter "-l" kann man eine Schnittstelle direkt ansprechen, z. B. cu -l /dev/ttyS0.

Die uu-Kommandogruppe kennt noch eine Reihe weiterer Kommandos für den Verkehr zwischen UNIX-Rechnern - siehe weiterführende Literatur. uucp-Verbindungen sind jedoch in jedem Fall für Mail, Netnews und gelegentliche Dateitransfers ausreichend. Sehr viel komfortabler wird es jedoch, wenn über Netzwirkabel, Standleitung, Modemverbindung oder sogar Satellitenfunk die einzelnen Systeme direkt vernetzt sind.

7.2.2 Die Internet-Protokollfamilie

Am häufigsten eingesetzt wird das Internet-Protokoll = TCP/IP (Transmission Control Protocol/Internet Protocol). Über TCP/IP lassen sich auch unterschiedliche Rechner miteinander vernetzen. Es deckt die untersten 4 Schichten des OSI-Modells ab (Bitübertragung, Datensicherung, Vermittlung, Transport). Einige Funktionen reichen jedoch bis zur Applikationsschicht hinauf. Grundsätzlich können TCP/IP-Verbindungen über die unterschiedlichsten Medien erfolgen. Durch die Verbindung zweier Netze entsteht so ein größeres Netz (Inter-Net). Dies hat zu einer weltweiten Vernetzung von Rechnern mit TCP/IP geführt, die unter dem Namen "Internet" läuft.

Dieses Kapitel geht davon aus, dass Sie Ihren Linux-Rechner an ein schon bestehendes lokales Netzwerk anschließen möchten. Für die Verbindung zum Internet von zuhause aus empfehle ich generell die Anschaffung eines passenden Routers (ISDN, DSL etc.). Derartige Geräte sind inzwischen so preiswert geworden, dass die Kosten den Gewinn an Bequemlichkeit und vor allem Sicherheit bei weitem aufwiegen. Der Router wird gemäß der Hersteller- und Providerangaben konfiguriert und ermöglicht dann jedem Rechner im Netz den Internetzugang. Gleichzeitig schützt er das Netz durch seine Firewall-Funktionen.

Die Installation und Initialisierung von TCP/IP komplett zu beschreiben, würde die Grenzen dieses Skripts sicherlich sprengen. Eine solche Beschreibung ist auch ziemlich überflüssig, da nahezu jeder Hersteller eigene Installationsroutinen zur Verfügung stellt. Leider sind diese unter Unix nicht einheitlich, jedoch läuft die Einrichtung von TCP/IP zumeist schon während der Installation des Betriebssystems ab. Aus diesem Grund beschränke ich mich hier auf eine kurze Zusammenfassung der wichtigsten Punkte und auf die allgemein wichtigen Kommandos und Konfigurationsdateien.

Für weitergehende Information zur Netzwerktechnik sei auf das Skript **Praktische Einführung in Computernetze** verwiesen. Mehr über das Internet finden Sie im Skript **Internet-Einführung**.

Was braucht man eigentlich alles, um einen Linux-Rechner ans Netz zu bringen? Eine Netzwerkkarte - klar! Diese wird in der Regel schon bei der Installation erkannt und das passende Kernel-Modul eingebunden. Feststellen lässt sich das mit dem Kommando `dmesg | more`. Dann brauchen Sie noch eine **IP-Adresse**, die **Netzmaske**, die **Netzwerkadresse** sowie **Broadcast- und Gateway-Adresse**.

Beim derzeit aktuellen IP-Protokoll V4 ist jede Netzwerk-Adresse eine 32-stellige Binärzahl. Weil aber nun 32 Nullen und Einsen etwas unübersichtlich sind, hat man daraus einfach vier Gruppen zu je acht Stellen gemacht und schreibt diese vier Gruppen als Dezimalzahlen auf. Nachdem bei jeder Zahl Werte zwischen 0 und 255 auftreten können, schreibt man noch einen Punkt zwischen die Zahlen. Heraus kommt dabei z. B. 105.22.234.1 (was besser lesbar ist als binär 01101001000101101110101000000001).

Rechner in lokalen Netzen sind für das Internet im Regelfall unsichtbar. Das bedeutet aber nicht, dass die Rechner keine Internetfunktionen nutzen können. Aber diese Rechner sind vor unkontrollierten Zugriffen aus dem Internet geschützt. Rechner, die weltweit kommunizieren sollen, bekommen vom jeweiligen Internet Service Provider eine IP-Adresse zugeteilt. Im LAN ohne direkte Internet-Verbindung braucht man aber nur IP-Adressen, die im jeweiligen Netz eindeutig sein müssen, nicht aber weltweit. Es wurden daher im IP-Zahlenraum drei Bereiche für lokale Netzwerke reserviert, die man jederzeit verwenden darf:

10.0.0.0 - 10.255.255.255 (A-Netz)
172.16.0.0 - 172.31.255.255 (B-Netze)
192.168.0.0 - 192.168.255.255 (C-Netze)

Der erste Bereich ermöglicht theoretisch ein Netz mit 16 Millionen Rechnern - das reicht auch für sehr große Firmen. Beim zweiten Bereich handelt es sich um 16 Teilnetze mit je ca. 65 000 Adressen (z. B. 172.23.0.0 bis 172.23.255.255). Der dritte Bereich besteht aus 256 kleinen Teilnetzen mit jeweils 254 Adressen. Ganz egal, in welchem Teilnetz Sie Ihr lokales Netz bilden - es ist sichergestellt, dass es zu keinen Adresskonflikten mit *richtigen* IP-Internetadressen kommt.

Meist wollen Sie freilich auch innerhalb des lokalen Netzes Internetfunktionen nutzen (beispielsweise im Web surfen). Um dies zu ermöglichen, muss innerhalb des lokalen Netzwerks ein Rechner bzw. der oben erwähnte Router als sogenanntes *Gateway* zum Internet konfiguriert werden. Dieser Rechner/Router stellt die Verbindung zum Internet her (über DSL, ISDN, Modem etc.) und leitet alle Internetanforderungen des lokalen Netzes weiter. Das Gateway hat außerdem die Aufgabe, die lokalen IP-Adressen durch eine weltweit gültige IP-Adresse zu ersetzen. Sie suchen sich also ein (Teil-)Netz aus dem oben angegebenen Nummernkreis und vergeben daraus die Rechner-IP-Adressen.

Nun kommen Netzmaske, Netzwerk- und Broadcast-Adresse ins Spiel. Die Ausdehnung eines lokalen Netzes wird durch die Netzmaske eingeschränkt. Dabei handelt es sich abermals um vierteilige Zifferngruppen, die intern als Bitmuster für IP-Adressen verwendet werden. Die Netzmaske legt fest, welcher Teil der IP-Adresse die Netzwerkadresse ist und welcher die Rechneradresse innerhalb des Netzes (das sorgt nicht nur dafür, dass mehrere lokale IP-Netze dasselbe Kabel verwenden können, ohne sich gegenseitig zu stören, sondern es macht auch den Routern das Leben leichter. Die Netzmaske hat bei allen Stellen, welche die Netzwerkadresse repräsentieren, eine 1 stehen und dahinter lauter Nullen (z. B.: 11111111111111110000000000000000 entspricht 255.255.0.0). Die Netzwerkadresse hat an den Stellen, welche den Rechner (Host) repräsentieren, Nullen stehen. Die Broadcast-Adresse (Broadcast = Rundruf, an alle) hat beim Rechneranteil lauter Einsen. Dazu ein Beispiel:

Betriebssystem UNIX/Linux

Wenn das lokale Netz alle Nummern 192.168.12.x umfasst, lautet die dazugehörige Netzmaske 255.255.255.0, die Netzwerkadresse 192.168.12.0 und die Broadcast-Adresse 192.168.12.255. (Bei manchen Konfigurationsprogrammen brauchen Sie keine Netzwerkadresse anzugeben, da sich diese aus den beiden anderen Adressen ergibt.) Das resultierende Netzwerk wird jetzt mit 192.168.12.0/255.255.255.0 oder kurz mit 192.168.12.0/24 bezeichnet. (Die Kurzschreibweise gibt die Anzahl der binären Einser der Netzmaske an.) Zwei Rechner mit den IP-Adressen 192.168.12.71 und 192.168.12.72 können sich in diesem Netzwerk also direkt miteinander verständigen (weil die IP-Adressen im Bereich der Netzmaske übereinstimmen). Die maximale Anzahl von Rechnern, die gleichzeitig in diesem Netz kommunizieren können, beträgt 254 (.1 bis .254) - die Nummern .0 und .255 sind ja reserviert.

Ein Gateway ist ein Router oder Rechner, der an der Schnittstelle zwischen zwei Netzen steht (oft zwischen dem lokalen Netz und dem Internet). Damit Ihr Unix/Linux-Rechner in einem lokalen Netz auf das Internet zugreifen kann, muss bei der Konfiguration die Gateway-Adresse angegeben werden. Die Gateway-Adresse bezeichnet also einen Rechner, der ebenfalls im lokalen Netz steht - z. B. mit der IP-Adresse 192.168.12.254. Dieser Rechner hat insofern eine Sonderstellung, als er mit dem Internet in Verbindung steht. Dort hat er eine (vom Provider zugeteilte) gültige IP-Adresse. Der Internetverkehr des gesamten lokalen Netzwerks erfolgt über den Gateway-Rechner, der jeweils die interne IP-Adresse eines jeden Datenpakets in die "offizielle" umsetzt und beim Antwortpaket umgekehrt verfährt (NAT, Network Address Translation).

Ein **Nameserver** ist ein Programm, das Rechnernamen bzw. Internetadressen (z. B. www.netzmafia.de) in IP-Adressen übersetzt. Bei kleinen Netzen erfolgt die Zuordnung zwischen Namen und Nummern oft über eine Tabelle (Datei */etc/hosts*). Im Internet übernehmen Rechner mit entsprechenden Datenbanken diese Aufgabe. Statt des Begriffs Nameserver ist auch die Abkürzung DNS für *Domain Name Server* oder *Domain Name Services* üblich. Wenn Sie in einem Webbrowser die Site www.netzmafia.de ansehen möchten, wird daher als Erstes der Nameserver des Providers kontaktiert, um die IP-Adresse des Netzmafia-Webservers herauszufinden. Erst nachdem das gelungen ist, wird eine Verbindung mit dieser IP-Adresse hergestellt.

Die Abkürzung **DHCP** steht für *Dynamic Host Configuration Protocol*. DHCP wird oft in lokalen Netzwerken verwendet, um die Administration des Netzwerks zu zentralisieren. Anstatt bei jedem Rechner getrennt die IP-Adresse, das Gateway, den Nameserver etc. einzustellen, wird der Router oder ein Rechner als DHCP-Server konfiguriert. Alle anderen Rechner im lokalen Netzwerk nehmen beim Systemstart Kontakt mit dem DHCP-Server auf und fragen diesen, welche Adressen und Einstellungen sie verwenden sollen. Damit reduziert sich die Client-Konfiguration auf ein Minimum.

Eine besondere Rolle spielt noch das **Loopback-Interface**: Diese Schnittstelle ermöglicht die Verwendung des Netzwerkprotokolls für lokale Dienste, also zur Kommunikation innerhalb des Rechners. Das klingt vielleicht widersinnig, ist aber für viele elementare Linux-Kommandos erforderlich. Der Grund: Manche Kommandos bauen ihre Kommunikation auf dem Netzwerkprotokoll auf, ganz egal, ob die Daten lokal auf dem Rechner bleiben oder über ein Netz auf einem fremden Rechner weiterverarbeitet werden. Ein Beispiel dafür ist der Druckerdämon *lpd*, der das Spooling für den Drucker übernimmt und sowohl lokal als auch von fremden Rechnern genutzt werden kann. Auch das X-Protokoll für die grafische Bedineroberfläche arbeitet netzwerkbasierend.

Als IP-Adresse für das Loopback-Interface ist immer 127.0.0.1 vorgesehen. Alle Distributionen kümmern sich automatisch um die Konfiguration des Loopback-Interface, auch wenn ansonsten keine Netzwerkkonfiguration durchgeführt wird. Dieser Adresse ist in der Regel auch der Name *localhost* zugeordnet.

Für die Konfiguration des Netzwerks eines Unix/Linux-Rechners gibt es mehrere Möglichkeiten:

Betriebssystem UNIX/Linux

- Ihr Rechner ist nicht Teil eines lokalen Netzes: Abgesehen von 127.0.0.1 für das Loopback-Interface benötigen Sie gar keine IP-Adresse. Ihre einzige Konfigurationsaufgabe besteht darin, Domain- und Hostnamen anzugeben.
- Ihr Rechner ist Teil eines bestehenden lokalen Netzes: Die IP-Adresse muss sich innerhalb der gültigen Adressen für dieses Netzwerk befinden (z. B. 192.168.67.*) und sie muss darin eindeutig sein. Falls es im Netz einen DHCP-Server gibt, brauchen Sie keine IP-Adresse anzugeben, müssen aber festlegen, dass der Rechner als DHCP-Client arbeiten soll.
- Ihr Rechner soll ein lokales Netz gründen: Entscheiden Sie sich für einen privaten IP-Adressraum (z.B.10.10.10.*) und weisen Sie dem Rechner eine IP-Adresse daraus zu.
 - ◆ Sie müssen auf jeden Fall Rechnernamen und Nameserver angeben. Optional kann der Hostname auch via DHCP eingestellt werden. Hier erfolgt auch die Nameserver-Konfiguration. Falls es einen DHCP-Server gibt, stellt dieser die IP-Adressen von mindestens einem Nameserver zur Verfügung. Nur wenn das nicht der Fall ist, müssen Sie die Adressen selbst eingeben.
 - ◆ Im Routing-Dialog müssen Sie normalerweise nur die IP-Adresse des Internet-Gateways Ihres Netzwerks eingeben. Falls es einen DHCP-Server gibt, überträgt er diese Information und das Eingabefeld "Standard-Gateway" darf leer bleiben.

Im Folgenden soll die letzte Möglichkeit näher betrachtet werden, weil sie die Interessanteste ist. Die beschriebenen Kommandos werden in der Regel von einem Start-Script in */etc/init.d* aufgerufen. Die Kenntnis der folgenden Kommandos ist jedoch trotzdem nützlich, beispielsweise, wenn sie temporär eine Netzwerk-Konfiguration ausprobieren möchten, ohne die Werte zu speichern.

Schnittstellenkonfiguration mit dem `ifconfig`-Kommando

Das Starten von TCP/IP erfolgt (unter Unix) durch Shell-Skripte, die je nach Unix-Derivat anders heißen und sich an ganz unterschiedlichen Stellen des jeweiligen Dateisystems befinden können. So unterschiedlich die Shell-Skripte auch sein mögen, die Initialisierung erfolgt in jedem Falle durch das `ifconfig`-Kommando. Hier wird auch die Initialisierung der Netzwerkschnittstellen vorgenommen. Dabei gibt es folgende Arten von Schnittstellen:

- das Loopback-Interface,
- Broadcast-Interfaces und
- Point-to-Point-Interfaces.

Initialisiert wird das Loopback-Interface durch das Kommando:

```
ifconfig lo0 127.0.0.1
```

Broadcast-Interfaces sind die üblichen Schnittstellen zu lokalen Netzwerken, über die mehrere Systeme erreichbar sind, und über die Broadcasts, also Nachrichten an alle, verschickt werden. Es handelt sich dabei um Schnittstellen zu Ethernet und TokenRing. Neben der Internet-Adresse werden bei der Initialisierung des Broadcast-Interfaces auch die Netzmaske und die Broadcast-Adresse angegeben:

```
ifconfig eth0 192.168.0.1 netmask 255.255.255.0 broadcast  
192.168.0.255
```

Neben den Broadcast-Schnittstellen gibt es noch die sogenannten Point-to-Point-Schnittstellen. Sie sind dadurch gekennzeichnet, daß man nur über sie ein anderes System erreichen kann. Beispiele sind SLIP (Serial Line IP) und das Point-to-Point-Protokoll PPP, die Verbindungen über die serielle Schnittstelle oder per Modem/ISDN-Adapter WAN-Verbindungen zulassen. Die Initialisierung

Betriebssystem UNIX/Linux

einer Point-to-Point-Schnittstelle hat z.B. die folgende Form:

```
ifconfig ppp0 192.168.1.1 192.168.1.2 netmask 255.255.255.240
```

So eine PPP-Verbindung bildet ein eigenständiges Netzwerk. Sollen mehrere Verbindungen kombiniert werden, so muß eine Unterteilung in Subnetze erfolgen. Das heißt, daß eine entsprechende Netzmaske gewählt werden muß. Wird als Argument für das ifconfig-Kommando nur der Name der Schnittstelle angegeben, so bezieht sich das auf die aktuelle Konfiguration der Schnittstelle, die dann ausgegeben wird:

```
eth0      Link encap:10Mbps Ethernet  HWaddr 00:20:18:03:0B:F5
          inet addr:10.10.10.4  Bcast:10.10.10.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0
          TX packets:0 errors:0 dropped:0 overruns:0
          Interrupt:11 Base address:0x340
```

Nun kann man mit dem *route*-Kommando überprüfen, ob auch die Routen korrekt gesetzt wurden (ggf. muss man das mit diesem Kommando nachholen), zum Beispiel:

```
# route -n

Kernel IP routing table
Destination Gateway      Genmask         Flags Metric Ref Use Iface
127.0.0.0  0.0.0.0        255.0.0.0      U         0      0   0   lo
192.168.1.0 0.0.0.0        255.255.255.0 U         0      0   0   eth0
0.0.0.0     192.168.0.10  0.0.0.0        UG        0      0   0   eth0
```

Die ersten beiden Zeilen beschreiben das Loopback-Interface und die Netzwerkkarte, die letzte Zeile die ROute ins INternet (Gateway).

Systemnamen und Internet-Adressen: /etc/hosts

In dieser Datei werden die Systeme des Netzwerks mit ihrem Systemnamen und die dazu gehörenden Internet-Adressen aufgelistet. Die Einträge in die Datei /etc/hosts haben die folgende allgemeine Form:

Internet-Adresse Name Aliase ...

Dazu ein Beispiel:

```
127.0.0.1 localhost
192.168.0.1 sun1-lbs micky
192.168.0.2 sun2-lbs minnie
192.168.0.3 sun3-lbs goofy
192.168.0.4 sun4-lbs donald
192.168.0.5 sun5-lbs dagobert
192.168.0.6 sun6-lbs daisy
192.168.0.7 sun7-lbs tick
192.168.0.8 sun8-lbs trick
192.168.0.9 sun9-lbs track
```

Nach der Internet-Adresse wird der "offizielle" Name des Systems angegeben, gefolgt von Alias-Namen für dieses System. Wird als Argument für ein Netzwerk-Kommando ein Name angegeben, so wird in dieser Datei die zugehörige Internet-Adresse ermittelt. Erst über die Adresse wird eine Verbindung zum Zielsystem aufgebaut. Die Datei /etc/hosts wird jedoch auch für den umgekehrten Vorgang benutzt. Mit einem IP-Datagramm wird nur die Internet-Adresse des sendenden Systems mitgeschickt. Soll nun der zugehörige Name ermittelt werden, so geschieht dies ebenfalls mittels dieser Datei. Das Resultat ist jedoch immer der "offizielle" Name des Systems.

Betriebssystem UNIX/Linux

Deshalb ist darauf zu achten, daß stets dieser Name verwendet werden muß, wenn ein Rechnername in weiteren Konfigurationsdateien eingetragen wird.

Natürlich reicht das System mit `/etc/hosts` höchstens für ein lokales Kleinstnetz mit einer handvoll Rechner aus, denn auf jedem Rechner muß die `/etc/hosts` auf dem aktuellen Stand gehalten werden. Diesem Problem sahen sich auch bald die Väter des Internet gegenüber und so wurde die größte weltweit verteilte Datenbank, das Domain Name System (DNS) erfunden. Es gibt eine Software, welche die Namensdatenbank verwaltet und auf Anfrage zu einem Rechnernamen die IP-Adresse liefert oder umgekehrt. Das Programm ist unter UNIX in der Regel BIND (Berkely Internet Name Daemon). Es läuft als "Nameserver" auf normalerweise mindestens zwei Rechnern einer Domain. Mehr darüber finden Sie im [DNS-Kapitel](#) des Internet-Skripts. Für den Rechner, der DNS nutzen will gibt es zwei Dateien, `/etc/hosts.conf` und `/etc/resolv.conf`, die festlegen, wie der Nameserver genutzt wird. In `/etc/hosts.conf` wird festgelegt, wie die Namenssuche erfolgen soll:

```
order hosts bind
multi on
```

Mit `order hosts bind` wird festgelegt, daß zuerst in der lokalen Datenbank `/etc/hosts` gesucht werden soll und erst dann eine Nameserveranfrage an einen fernen Rechner gestartet wird. Die Datei `/etc/resolv.conf` enthält Infos über den Nameserver:

```
search mydomain.net
nameserver 10.10.10.4
nameserver 10.10.10.1
```

Wie die Datei `/etc/hosts` enthält auch die Datei `/etc/networks` Adressen und Namen. Diesmal sind es allerdings Namen für Netzwerke. Die Funktion dieser Datei ist durchaus mit der `/etc/hosts` vergleichbar: Netzwerk-Namen werden in Netzwerk-Adressen umgesetzt und umgekehrt. Die allgemeine Form eines Eintrags sieht dann so aus:

Netzwerk-Name Netzwerk-Adresse Netzwerk-Aliase ...

Zum Beispiel:

```
loopback 127
admin-net 192.168.1
dev-net 192.168.2
```

Eine weitere Datei ist für die Zuordnung der Portnummern zu den einzelnen Diensten wie Telnet, FTP, WWW, Mail, usw. zuständig. In dieser Datei, `/etc/services`, werden der Name des Dienstes, die Portnummer, das Transportprotokoll (UDP oder TCP) und Service-Aliase angegeben. Die allgemeine Form eines Eintrags in `/etc/services` hat die Form:

Service-Name Portnummer/Protokoll Service-Aliases

Wichtig: Hier sind nur Portnummern für Server spezifiziert. Client-Programme bekommen beim Verbindungsaufbau eine beliebige, freie Portnummer zugewiesen. So kann der Server wieder auf der Standard-Portnummer aus `/etc/services` auf einen weiteren Verbindungswunsch warten. Die spezifizierten Portnummern sind auf allen Rechnern im Netz gleich. Die Server-Programme entnehmen dieser Datei, auf welchen Port sie zugreifen müssen. Die Client-Programme finden hier die entsprechenden Portnummern ihrer Server. In `/etc/services` werden die Portnummern für TCP- und UDP-Dienste spezifiziert. Die Portnummern für diese beiden Transport-Protokolle sind völlig unabhängig voneinander. Trotzdem ist es im allgemeinen üblich, gleiche Portnummern für beide Protokolle zu benutzen, wenn ein Dienst über beide Transportprotokolle verfügbar ist.

Betriebssystem UNIX/Linux

Ein Ausschnitt aus /etc/services:

```
tcpmux          1/tcp                # TCP port service multiplexer
echo            7/tcp
echo            7/udp
discard         9/tcp                sink null
discard         9/udp                sink null
sysstat         11/tcp              users
daytime         13/tcp
daytime         13/udp
netstat         15/tcp
gotd            17/tcp              quote
msp             18/tcp              # message send protocol
msp             18/udp              # message send protocol
chargen         19/tcp              ttytst source
chargen         19/udp              ttytst source
ftp             21/tcp
#               22 - unassigned
telnet          23/tcp
#               24 - private
smtp            25/tcp              mail
#               26 - unassigned
time            37/tcp              timserver
time            37/udp              timserver
rlp             39/udp              resource # resource location
nameserver      42/tcp              name # IEN 116
whois           43/tcp              nickname
domain          53/tcp              nameserver # name-domain server
domain          53/udp              nameserver
mtp             57/tcp              # deprecated
bootps          67/tcp              # BOOTP server
bootps          67/udp
bootpc          68/tcp              # BOOTP client
bootpc          68/udp
tftp            69/udp
gopher          70/tcp              # Internet Gopher
gopher          70/udp
rje             77/tcp              netrjs
finger          79/tcp
www             80/tcp              http # WorldWideWeb HTTP
www             80/udp              # HyperText Transfer Protocol
link            87/tcp              ttylink
kerberos        88/tcp              krb5 # Kerberos v5
kerberos        88/udp
supdup          95/tcp
#               100 - reserved
hostnames       101/tcp              hostname # usually from sri-nic
iso-tsap        102/tcp              tsap # part of ISODE.
csnet-ns        105/tcp              cso-ns # also used by CSO name server
csnet-ns        105/udp              cso-ns
rtelnet         107/tcp              # Remote Telnet
rtelnet         107/udp
pop2            109/tcp              postoffice # POP version 2
pop2            109/udp
pop3            110/tcp              # POP version 3
pop3            110/udp
sunrpc          111/tcp
sunrpc          111/udp
auth            113/tcp              tap ident authentication
sftp            115/tcp
uucp-path       117/tcp
nntp            119/tcp              readnews untp # USENET News Transfer Protocol
ntp             123/tcp
ntp             123/udp              # Network Time Protocol
netbios-ns      137/tcp              # NETBIOS Name Service
netbios-ns      137/udp
```

Betriebssystem UNIX/Linux

```
netbios-dgm      138/tcp          # NETBIOS Datagram Service
netbios-dgm      138/udp
netbios-ssn      139/tcp          # NETBIOS session service
netbios-ssn      139/udp
imap2            143/tcp          # Interim Mail Access Proto v2
imap2            143/udp
.....
```

Es gibt bei UNIX zwei Möglichkeiten, einen Netzdienst anzubieten:

- Starten eines eigenen Server-Daemons beim Systemstart
- Starten des Server-Daemons über den Netzwerk-Daemon `inetd`.

Die erste Möglichkeit wird man bei stark frequentierten Diensten verwenden (z. B. `http`, `smtp`), da hier gleich der Server angesprochen werden kann und nicht erst gestartet werden muß. Bei allen anderen Diensten nimmt man in der Regel den `inetd`. Dieser Prozeß hat eine Tabelle, in der steht, für welchen Port welches Programm zu starten ist - also eine recht flexible Angelegenheit. Will man beispielsweise einen neuen FTP-Server (etwa `wu-ftp` statt des Standard-`ftpd`) einsetzen, so genügt das Ändern der Tabelle in der Datei `/etc/inetd.conf` und ein Restart des `inetd` (Kommando: `telinit q`. Ja, ohne "-" vor dem "q"). Man kann durch Auskommentieren von Zeilen in der `inetd.conf` auch nicht benötigte Netzdienste sperren und so den Rechner gegen Eindringlinge sicherer machen. Hier ein Auszug aus der Datei:

```
# See "man 8 inetd" for more information.
#
# If you make changes to this file, either reboot your machine or send the
# inetd a HUP signal.
#
#
# These are standard services.
#
ftp      stream    tcp    nowait    root    /usr/sbin/wu.ftp    wu.ftp -a
# ftp    stream    tcp    nowait    root    /usr/sbin/in.ftp    in.ftp
telnet   stream    tcp    nowait    root    /usr/sbin/in.telnetd in.telnetd
# nntp   stream    tcp    nowait    root    tcpd     in.nntp
smtp     stream    tcp    nowait    root    /usr/sbin/sendmail  sendmail -v
printer  stream    tcp    nowait    root    /usr/bin/lpd        lpd -i
#
# Shell, login, exec and talk are BSD protocols.
#
shell    stream    tcp    nowait    root    /usr/sbin/in.rshd   in.rshd -L
login    stream    tcp    nowait    root    /usr/sbin/in.rlogind in.rlogind
exec     stream    tcp    nowait    root    /usr/sbin/in.rexecd  in.rexecd
# talk   dgram     udp    wait      root    /usr/sbin/in.talkd  in.talkd
# ntalk  dgram     udp    wait      root    /usr/sbin/in.talkd  in.talkd
#
# Kerberos authenticated services
#
# klogin  stream    tcp    nowait    root    /usr/sbin/tcpd      rlogind -k
# eklogin stream    tcp    nowait    root    /usr/sbin/tcpd      rlogind -k -x
# kshell  stream    tcp    nowait    root    /usr/sbin/tcpd      rshd -k
#
# Services run ONLY on the Kerberos server
#
# krbupdate stream    tcp    nowait    root    /usr/sbin/tcpd      registerd
# kpasswd  stream    tcp    nowait    root    /usr/sbin/tcpd      kpasswd
#
# Pop et al
#
# pop2    stream    tcp    nowait    root    /usr/sbin/in.pop2d  in.pop2d
pop3     stream    tcp    nowait    root    /usr/sbin/popper     popper -s
#
```

Betriebssystem UNIX/Linux

```
# Comsat - has to do with mail.
#
# comsat    dgram    udp    wait    root    /usr/sbin/tcpd    in.comsat
#
# The Internet UUCP service.
#
# uucp      stream    tcp    nowait    uucp    /usr/sbin/tcpd    /usr/lib/uucp/uucico
#
# Tftp service is provided primarily for booting.  Most sites
# run this only on machines acting as "boot servers."
#
# tftp      dgram    udp    wait    nobody    /usr/sbin/tcpd    in.tftpd
# bootps    dgram    udp    wait    root    /usr/sbin/bootpd    bootpd
#
# Finger, systat and netstat give out user information which may be
# valuable to potential "system crackers."  Many sites choose to disable
# some or all of these services to improve security.
# Try "telnet localhost systat" and "telnet localhost netstat" to see that
# information yourself!
#
finger     stream    tcp    nowait    nobody    /usr/sbin/in.fingerd    in.fingerd    -w
systat     stream    tcp    nowait    nobody    /bin/ps                /bin/ps        -auwx
netstat    stream    tcp    nowait    root     /bin/netstat            /bin/netstat   -a
ident      stream    tcp    nowait    root     /usr/sbin/in.identd    in.identd
#
# These are to start Samba, an smb server that can export filesystems to
# Pathworks, Lanmanager for DOS, Windows for Workgroups, Windows95, Lanmanager
# for Windows, Lanmanager for OS/2, Windows NT, etc.  Lanmanager for dos is
# available via ftp from ftp.microsoft.com in bussys/MScclient/dos/. Please read
# the licensing stuff before downloading. Use the TCP/IP option in the client.
# Add your server to the \etc\lmhosts (or equivalent) file on the client.
netbios-ssn  stream    tcp    nowait    root     /usr/bin/smbd          smbd
netbios-ns   dgram    udp    wait     root     /usr/bin/nmbd          nmbd
# End.
```

Als letzte der Konfigurations-Dateien soll die /etc/protocols behandelt werden. Hier werden die über IP arbeitenden Protokolle aufgelistet. Die allgemeine Form eines Eintrags hat die Form:

Protokoll-Name Protokoll-Nummer Protokoll-Aliase ...

Zum Beispiel:

```
ip    0    IP        # internet protocol, pseudo protocol number
icmp  1    ICMP     # internet control message protocol
igmp  2    IGMP     # internet group multicast protocol
ggp   3    GGP      # gateway-gateway protocol
tcp   6    TCP      # transmission control protocol
egp   8    EGP      # Exterior-Gateway Protocol
PUP   12   PUP      # PARC universal packet protocol
udp   17   UDP      # user datagram protocol
idp   22   IDP      # WhatsThis?
hello 63   HELLO    # HELLO Routing Protocol
raw   255   RAW      # RAW IP interface
```

Die Protokoll-Nummer wird im Header des Internet-Protokolls angegeben.

7.2.3 Netzwerk-Kommandos

Vorab sollen ganz kurz einige Remote-Kommandos besprochen werden. Aus Berkley-UNIX in die Release 4 übernommen wurde die Familie der sogenannten 'R-Kommandos' (weil sie alle mit "r" beginnen). Sie erlauben komfortable Kommunikation zwischen UNIX-Rechnern auf der Basis von

TCP/IP.

Jeder Rechner im Netz (= Host) besitzt eine Datei `/etc/hosts.equiv`, in der "vertrauenswürdige" Rechner eingetragen sind. Wollen sich Benutzer von einem dieser Rechner auf dem lokalen Computer einloggen, brauchen sie kein Passwort für den Rechner-Rechner-Übergang (wobei "Benutzer" auch Dämonprozesse sein können automatischer Datenaustausch möglich). Enthält die Datei nur eine Zeile mit einem '+'-Zeichen, werden alle angeschlossenen Rechner akzeptiert. Gibt es keine "vertrauenswürdigen" Rechner, ist jeweils die Eingabe des Login-Passwortes nötig. Außerdem funktioniert das Ganze nur, wenn man auf Quell- und Zielrechner die gleiche Benutzererkennung hat.

Inzwischen veraltet sind die sogenannten R-Kommandos von Unix:

- `rlogin rhost [-ec] [-8] [-l username] Rechnername`
Einloggen auf einem anderen Rechner im Netz. Nach Eingabe des Passwortes können Sie auf diesem Rechner genauso arbeiten, wie auf den lokalen Rechner (fehlt die explizite Angabe einer Benutzererkennung, wird die Benutzererkennung automatisch übertragen).
- `rcp [-r] Quellrechner:datei Zielrechner:datei`
Kopieren von Dateien zwischen zwei Rechnern. Beim eigenen Rechner kann die Angabe Quell-/Zielrechner entfallen. Die Option "-r" erlaubt das Kopieren ganzer Verzeichnisbäume.
- `rsh rhost Kommando`
Erzeugen einer Shell auf einen fernen Rechner im Netz. Die Ausgaben erfolgen über die lokale Shell. Umleitung der Ausgabe auf dem Host durch Quoten des Umleitungssymbols (>').
- `rwho rhost`

die R-Kommandos werden inzwischen von der 'Secure Shell' (ssh) und dem 'Secure Copy' (scp) abgelöst. Weitere Netzwerkprogramme, die man zum Fern-Login braucht, sind 'telnet' und 'ftp', wobei telnet hauptsächlich zum Testen im lokalen Netz dient. An dieser Stelle gibt es eine kurze Bedienungsanleitung, im Internet-Skript werden die Hintergründe erklärt.

Telnet

telnet eignet sich besonders für die Kommunikation von UNIX-Rechnern mit Computern, die andere Betriebssysteme verwenden. Um eine Telnet-Sitzung zu starten wird `telnet` aufgerufen.

```
telnet host
```

Telnet (Teletype Network) unterscheidet zwei Modi, den Kommando- und den Eingabemodus. Der Kommandomodus ist aktiv, wenn man Telnet ohne Argument aufruft. Am Bildschirm erscheint der Prompt "Telnet>". Wird Telnet mit Argument aufgerufen, so wird ein "open"-Kommando mit dem angegebenen Argument ausgeführt und man befindet sich im Eingabemodus, bei dem zwei Übertragungsmodi unterschieden werden:

- der "Zeichen für Zeichen"-Modus
In diesem Modus wird im Regelfall jedes Zeichen sofort an das entfernte System zum Zweck der Weiterverarbeitung gesendet.
- der "Zeile für Zeile"-Modus
In diesem Modus wird der eingegebene Text lokal angezeigt, aber nur volle Zeilen werden (im Regelfall) an das entfernte System gesendet.

Welcher der beiden Modi aktiviert wird, hängt vom Zielsystem ab und ist mit dem Kommando `status` abfragbar. Vom Eingabe- in den Kommandomodus kann mit dem Telnet-Fluchtsymbol (Voreinstellung "^]") (Ctrl-]) gewechselt werden. Nur im Kommandomodus können die unten

beschriebenen Kommandos eingegeben werden. Hat man eine bestehende Verbindung und wechselt in den Kommandomodus, so fällt man automatisch nach Abarbeitung eines Kommandos wieder in den Eingabemodus zurück.

Falls der `localchars`-Schalter auf EIN steht (das ist die Grundeinstellung im Zeilen-Modus), werden in beiden Modi die "quit"-, "intr"- und "flush"-Zeichen lokal abgefangen und als Telnet-Protokoll-Sequenzen an das entfernte System gesendet.

Während eine Verbindung zu einem entfernten System besteht, kann in den Telnet-Kommandomodus mittels dem Telnet-Fluchtsymbol (Voreinstellung "^]") umgeschaltet werden.

Nachfolgend werden die verfügbaren Kommandos beschrieben. Kommandos können soweit verkürzt eingegeben werden, als sie noch eindeutig erkennbar sind. Gleiches gilt auch für die Argumente der Kommandos `mode`, `set`, `toggle` und `display`. Im Kommandomodus haben die üblichen Terminal-Editier-Konventionen Gültigkeit (UNIX-Shell und "stty"-Einstellungen).

- `open host`
eröffnet eine Verbindung zu dem bezeichneten Host. Der Host kann mit seinem Namen (z.B. `ftp.UniBw-Muenchen.de`) oder mit seiner Internet-Adresse (z.B. `129.187.10.3`) angegeben werden.
- `close`
schließt eine Telnet-Sitzung und kehrt in den Kommandomodus zurück. Wenn Argumente beim Aufruf angegeben wurden, ist die Wirkung die gleiche wie bei "quit".
- `quit`
schließt jede offene Telnet-Sitzung und verlässt Telnet.
- `mode type`
Als "type" steht entweder "line" für den "Zeile für Zeile"-Modus oder "character" für den "Zeichen für Zeichen"-Modus (jeweils ohne die Anführungszeichen geschrieben). Die Umschaltung erfolgt nur mit Genehmigung des Zielsystems.
- `set argument value`
Setzt eine oder mehrere Telnet-Variablen
- `argument` auf einen bestimmten Wert `value`. Der spezielle Wert "off" schaltet die Funktion aus, die im Zusammenhang mit der Variablen steht. Der Wert von Variablen kann mit dem "display"-Kommando abgefragt werden. Mit Ausnahme des Fluchtsymbols gelten für sie die Voreinstellungen des Terminals. Es können folgende Variablen angegeben werden:
 - ◆ `escape`
Das ist das Telnet-Fluchtsymbol (Voreinstellung "^]"), das eine Umschaltung in den Kommandomodus bewirkt, wenn eine Verbindung zu einem entfernten System besteht.
 - ◆ `interrupt`
Setzen des "interrupt"-Zeichens. Telnet sendet dies als "IP"-Sequenz (Interrupt Process) an das entfernte System.
 - ◆ `quit`
Setzen des "quit"-Zeichens. Telnet sendet dies als "BRK"-Sequenz (Break) an das entfernte System.
 - ◆ `flushoutput`
Setzen des "flushoutput"-Zeichens. Telnet sendet dies als "AO"-Sequenz (Abort Output) an das entfernte System.
 - ◆ `erase`
Setzen des "erase"-Zeichens. Telnet sendet dies als "EC"-Sequenz (Erase Character) an das entfernte System.
 - ◆ `kill`
Setzen des "kill"-Zeichens. Telnet sendet dies als "EL"-Sequenz (Erase Line) an

- ◊ das entfernte System.
- ◊ eof
 - Setzen des "eof"-Zeichens. Telnet sendet dies als "eof"-Zeichen an das entfernte System.
- toggle arguments ...
 - schaltet diverse Schalter ein und aus, die bestimmen, wie Telnet auf verschiedene Ereignisse reagieren soll. Es dürfen mehrere Argumente angegeben werden. Der Zustand dieser Schalter kann mit dem "display"-Kommando abgefragt werden. Gültige Argumente sind:
 - ◊ localchars
 - Wenn dieser Schalter eingeschaltet ist, werden die "flush"-, "interrupt"-, "quit", "erase"- und "kill"- Zeichen (siehe oben) lokal erkannt und in die entsprechende Telnet- Steuersequenz umgesetzt (resp.: ao, ip, brk, ec und el). Anfangs ist dieser Schalter im "Zeile für Zeile"-Modus eingeschaltet und im "Zeichen für Zeichen"-Modus ausgeschaltet.
 - ◊ autoflush
 - Sind beide Schalter "autoflush" und "localchars" eingeschaltet und werden die "ao"-, "intr"- oder "quit"-Zeichen eingegeben, weigert sich Telnet irgendwelche Daten auf dem Benutzer-Terminal anzuzeigen, bis das entfernte System die Bearbeitung dieser Steuersequenzen bestätigt hat. Anfangs ist dieser Schalter eingeschaltet, soweit der Benutzer nicht das Kommando "stty noflsh" eingegeben hat (siehe "stty"-Kommando).
 - ◊ autosynch
 - Sind beide Schalter "autosynch" und "localchars" eingeschaltet und werden die "intr"- oder "quit"-Zeichen eingegeben, sendet Telnet zuerst die entsprechende Steuersequenz und anschließend die Telnet-"SYNCH"-Steuersequenz. Diese Vorgehensweise soll das entfernte System veranlassen, alle bisherigen Eingaben zu verwerfen, bis beide Telnet-Steuersequenzen gelesen und verarbeitet worden sind. Anfangs ist dieser Schalter ausgeschaltet.
 - ◊ crmod
 - Ist dieser Schalter eingeschaltet, werden die "carriage return"-Zeichen, die vom entfernten System empfangen werden, in ein "carriage return"-Zeichen gefolgt von einem "line feed"-Zeichen umgesetzt. Dieser Modus hat nur Wirkung auf die Zeichen, die vom entfernten System empfangen werden, jedoch nicht auf die Zeichen, die vom Benutzer eingegeben werden. Dieser Modus ist nur dann sinnvoll, wenn das entfernte System nur "carriage return"-Zeichen sendet aber keine "line feed"-Zeichen. Anfangs ist dieser Schalter bei UNIX ausgeschaltet und bei DOS eingeschaltet.
 - ◊ options
 - Wird dieser Schalter eingeschaltet, wird die Telnet-Protokoll-Verarbeitung angezeigt. Anfangs ist dieser Schalter ausgeschaltet.
 - ◊ netdata
 - Wird dieser Schalter eingeschaltet, werden die Verkehrsdaten im hexadezimalen Format angezeigt. Anfangs ist dieser Schalter ausgeschaltet.
 - ◊ ?
 - Zeige die zulässigen Schalter-Kommandos.
- status
 - zeigt den aktuellen Zustand von Telnet, die verbundenen Partner und den gegenwärtigen Modus.
- display [argument...]
 - zeigt alle oder nur ausgewählte Einstellungen und Schalter, die z.B. mit "set " oder "toggle" eingestellt wurden (siehe unten).
- ? [command]
 - Funktionsgleich mit help.

Betriebssystem UNIX/Linux

- `help [command]`
Ohne Argument gibt Telnet eine Übersicht über die Hilfe-Funktion. Wenn ein Kommando angegeben wird, gibt Telnet Auskunft über dieses Kommando.
- `send arguments`
sendet ein oder mehrere Sonderzeichen an das entfernte System. Folgende Argumente können angegeben werden (auch mehrere auf einmal):
 - ◆ `escape`
sendet das gegenwärtig definierte Telnet-Fluchtsymbol an das entfernte System (Voreinstellung "`^]`"); somit ist dieses Zeichen auch an das entfernte System sendbar.
 - ◆ `synch`
sendet das Telnet-"SYNCH"-Zeichen. Diese Sequenz veranlaßt das entfernte System, alle bisherigen, aber noch nicht verarbeiteten Eingaben zu verwerfen. Sie wird mittels TCP mit Dringlichkeit gesendet, kann aber von der Gegenseite abgelehnt werden. Wenn das entfernte System ein "4.2 BSD"-System ist, so wird die Aufforderung verworfen werden und z.B. als Kommando-Antwort ein "`r`" wie `rejected` empfangen werden.
 - ◆ `brk`
sendet die Telnet-"BRK"-Sequenz (Break), die für das entfernte System eine Bedeutung haben kann.
 - ◆ `ao`
sendet die Telnet-"AO"-Sequenz (Abort Output), die das entfernte System veranlasst, jede noch anstehende Ausgabe für das Terminal zu verwerfen.
 - ◆ `ayt`
sendet die Telnet-"AYT"-Sequenz (Are You There), worauf das entfernte System meist antwortet.
 - ◆ `ec`
sendet die Telnet-"EC"-Sequenz (Erase Character), die das entfernte System veranlasst, das zuletzt eingegebene Zeichen zu löschen.
 - ◆ `el`
sendet die Telnet-"EL"-Sequenz (Erase Line), die das entfernte System veranlasst, die aktuelle Zeile zu löschen.
 - ◆ `nop`
sendet die Telnet-"NOP"-Sequenz (No Operation).
 - ◆ `?`
zeigt Hilfe-Information für das "`send`"-Kommando.

Beispiel: Verbindung zum Rechner `lx-lbs`

```
telnet lx-lbs
```

Telnet reagiert daraufhin mit:

```
Trying to .....  
Connect to sun1-lbs  
Escape Character is '^]'
```

Jetzt folgt die normale Loginsequenz auf dem Rechner `sun1-lbs`. Um weitere Kommandos an Telnet geben zu können, muß der Befehl mit dem angegebenen Escape-Character eingeleitet werden (im Beispiel das ESC-Zeichen). Beenden der Verbindung erfolgt in diesem Fall mit "`^]` quit" oder einfach `Ctrl-D`.

telnet kann auch vorzüglich zum Testen von Verbindungen und Servern verwendet werden. Bei Kenntnis der Protokolle (nachlesbar in den entsprechenden RFCs) kann ein Protokoll wie SMTP, NNTP, HTTP, usw. auch von Hand nachgebildet werden. Man kann so durch eine

telnet-Verbindung nachsehen, ob auf einem fernen System Mail-, News-, WWW- oder andere Dienste laufen.

```
ftp [ -v ] [ -d ] [ -i ] [ -n ] [ -g ] [ host ]
```

FTP ist die Benutzerschnittstelle zum Dateiübertragungsprotokoll. Dieses Programm ermöglicht den Dateitransfer zwischen zwei Rechenanlagen. Es besitzt eine eigene Kommandooberfläche, die interaktiv bedient wird. Der Aufruf dieses Filetransferprogrammes erfolgt durch `ftp`.

FTP funktioniert aber auch, wenn man auf dem fernen Rechner keine Benutzerberechtigung hat, denn viele Rechner bieten große Dateibereiche über sogenannten 'anonymen' FTP. Man gibt in diesem Fall als Benutzernamen 'ftp' ein und als Passwort die eigene Mailadresse. Danach kann man sich im öffentlichen Dateibereich tummeln.

Wird beim Programmaufruf der gewünschte Kommunikationspartner (host) mit angegeben, so wird sofort versucht, eine Verbindung zu diesem Rechenanlagensystem aufzubauen. Ist der Versuch erfolglos, so wird in den Kommandomodus umgeschaltet. Der Prompt "`ftp>`" erscheint immer auf dem Bildschirm, wenn `ftp`-Kommandos eingegeben werden können. `ftp` verfügt über einen help-Mechanismus, über den sämtliche auf dem jeweiligen System verfügbare Kommandos mit Kurzerklärungen abfragbar sind.

Nachfolgend werden wesentliche Kommandos nach Funktionalität gruppiert vorgestellt.

Kommandos können soweit verkürzt eingegeben werden, als sie noch eindeutig erkennbar sind. Enthalten Kommandoargumente "Blanks", so sind die Argumente beidseitig mit Hochkommas eingeschlossen einzugeben.

- `help [command]`
zeigt kurze Informationen zu dem angegebenen Kommando "command". Wird "command" weggelassen, zeigt dieser Aufruf eine Liste der zulässigen Kommandos.
- `open host`
Je nach angewähltem System wird die Benutzererkennung und das zugehörige Passwort interaktiv abgefragt oder man muß diese per `ftp`-Kommando `user` aktiv senden.
- `user user-name [password]`
Eingabe von Benutzererkennung und Passwort.
- `!`
Aufruf einer Shell auf dem lokalen System mit eingeschränkter Funktionalität. Für Dateiübertragung relevante Kommandos wie `mkdir`, `mv`, `cp`, etc sind absetzbar. Verlassen wird diese Shell mit "exit".
- `lcd [directory]`
Wahl des Directories für die Dateiübertragung. Voreinstellung: Homedirectory des Benutzers.
- `pwd`
Anzeige des aktuellen Directories auf dem entfernten System.
- `cd remote-directory`
Wahl des aktuellen Directories auf dem entfernten System.
- `cdup`
Wechsel in das nächsthöhere Directory auf dem entfernten System.
- `dir [remote-directory [local-file]]`
`ls [remote-directory [local-file]]`
Ohne Optionen erfolgt eine Anzeige der Einträge des entfernten aktuellen Directories. Dabei liefert `dir` ausführliche und `ls` eine knappe Information bezüglich des Directory-Inhalts.
Bei Angabe des `remote-directory` erfolgt die Anzeige der Einträge des entfernten Directories. Wird `local-file` angegeben, erfolgt eine Umlenkung der Directory-Anzeige in die Datei `local-file` auf dem lokalen System.
- `mdir remote-files [local-file]`
`mls remote-files [local-file]`

Anzeige von Dateien aus dem entfernten aktuellen Directory und Abspeicherung in eine lokale Datei.

- `mkdir directory-name`
Einrichten eines neuen Directories `directory-name` auf dem entfernten System.
- `rmdir directory-name`
löscht das Directory `directory-name` auf dem entfernten System.
- `rename [from] [to]`
Umbenennen einer Datei auf dem entfernten System von `from` nach `to`.
- `delete remote-file`
Löschen der Datei `remote-file` auf dem entfernten System.
- `mdelete remote-files`
Löschen mehrerer Dateien `remote-files` auf dem entfernten System.
- `put local-file [remote-file]`
`send local-file [remote-file]`
Dateiübertragung der Datei `local-file` vom lokalen zum entfernten System. Wird `remote-file` nicht angegeben, so wird auch auf dem Zielsystem der Dateiname `local-file` verwendet.
- `append local-file [remote-file]`
überträgt die Datei `local-file` vom lokalen System an das entfernte System und hängt diese am Ende der Datei `remote-file` an. Wurde `remote-file` nicht angegeben, wird die Datei ans Ende der Datei `local-file` auf dem entfernten System angehängt.
- `mput local-files`
Dateiübertragung einer Dateigruppe namensgleich vom lokalen zum entfernten System.
- `get remote-file [local-file]`
`recv remote-file [local-file]`
Dateiübertragung einer Datei `remote-file` vom entfernten System zum lokalen System. Wird `local-file` nicht mitangegeben, so erhält die Datei auch auf dem lokalen System den Dateiname `remote-file`.
- `mget remote-files`
Dateiübertragung einer Dateigruppe namensgleich vom entfernten zum lokalen System.
- `ascii`
`type ascii`
Die Dateiübertragung findet im ASCII-Code statt. Gegebenfalls werden Bei Binärdateien Zeichen verändert (z. B. die Zeilenendedarstellung ans Zielsystem angepaßt) oder Zeichen verfälscht.
- `binary`
`type image`
`type binary`
Die Dateiübertragung findet transparent statt.
- `case`
Mit diesem Schalter läßt sich einstellen, ob Dateinamen beim Empfangen (`get`, `recv`, `mget`) von Großbuchstaben nach Kleinbuchstaben übersetzt werden sollen.
- `glob`
Mit diesem Schalter läßt sich einstellen, ob bei den Kommandos `mdelete`, `mget` und `mput` bei Dateinamen, die Metazeichen (`*?[]~{ }`) enthalten, diese Metazeichen übertragen werden oder nicht. ("off" keine Metazeichenbehandlung).
- `ntrans [inchars [outchars]]`
Definition und Aktivierung einer Übersetzungstabelle für Dateinamen, wenn beim Dateiübertragungsauftrag (Senden und Empfangen) keine Zieldateinamen angegeben werden. Zeichen eines Dateinamens, die in `inchars` zu finden sind, werden durch das positionsgleiche Zeichen in `outchars` übersetzt. Ist `inchars` länger als `outchars`, so werden die korrespondenzlosen Zeichen von `inchars` aus dem Zieldateinamen entfernt.
- `prompt`
Mit diesem Zeichen wird bei Mehrdateienübertragung gesteuert, ob jede zu übertragende

Betriebssystem UNIX/Linux

Datei extra quittiert werden muß oder nicht.

- `verbose`
Wenn der "verbose"-Modus eingeschaltet ist, erhält man für jede übertragene Datei den Dateinamen auf dem lokalen und entfernten Rechner, sowie die Datenmenge und die dafür benötigte Übertragungszeit angezeigt.
- `bell`
Dieser Schalter bewirkt, daß je nach Stellung am Ende jedes Dateiübertragungsauftrages ein akustisches Signal ertönt oder nicht.
- `status`
Anzeige der aktuellen logischen Schalterstellungen sowie des Verbindungszustandes.
- `close`
`disconnect`
Beendigung einer aktiven Verbindung.
- `quit`
Beendigung des Programmes `ftp`.
- `bye`
Beendigung einer aktiven Sitzung und des Programmes `ftp`.

Die optionalen Parameter beim `ftp`-Kommando setzen logische Schalter für den `ftp`-Programmablauf. Im Kommandomodus sind die Einstellungen jederzeit wieder änderbar.

- `-v` verbose-Schalter einschalten.
- `-d` debug-Schalter einschalten.
- `-i` interactive-Modus für Mehrdateiübertragung einschalten.
- `-n` verhindert, daß
- FTP zum Beginn der Sitzung einen Login-Versuch unternimmt.
- `-g` glob-Schalter einschalten.

Die Datei-Übertragung wird durch die Terminal "interrupt"-Taste (üblicherweise Ctrl-C) abgebrochen, was einen sofortigen Abbruch zur Folge haben soll. Nicht alle Kommunikationspartner verstehen die Abbruchaufforderung und dann wird dennoch die gesamte Datei übertragen ausgeführt.

Dateinamen, die als Argumente von FTP-Kommandos Verwendung finden, werden wie folgt bearbeitet: Ist "file globbing" eingeschaltet, werden bei den Kommandos `mget`, `mput` und `mdelete` die Namen lokaler Dateien folgendermaßen behandelt:

- Der `*` steht für eine beliebige Anzahl (auch Null) von Zeichen.
- Das `?` steht für ein einziges beliebiges Zeichen.
- Wird im Dateinamen eine Zeichenfolge angetroffen, die zwischen eckigen Klammern oder zwischen geschweiften Klammern steht, so sind alle Dateinamen zutreffend, die an dieser Stelle ein einziges beliebiges Zeichen aus der Zeichenfolge innerhalb der Klammern enthalten.
- Steht die Zeichenfolge `~/` (Tilde, Schrägstrich) am Beginn des Dateinamens, so wird sie durch den Home-Directory-Pfad ersetzt. Das Zeichen `~`, dem eine Benutzerkennung folgt, wird durch den Home-Directory-Pfad dieser Benutzerkennung ersetzt.

Nicht alle `ftp`-Installationen unterstützen alle `ftp`-Kommandos. Wenn sich eine Dateiübertragung nicht ordnungsgemäß abbrechen läßt, kann der lokale `ftp`-Prozess mit dem Terminal-intr-Zeichen (üblicherweise Ctrl-C) abgebrochen werden.

Die Kommandos `telnet` und `ftp` (und die `r`-Kommandos) stellen jedoch ein gravierendes Sicherheits-Problem dar, denn:

Betriebssystem UNIX/Linux

- Beim Verbindungs-Aufbau erforderliche Userkennungen und Paßwörter werden im *Klartext* übertragen. Damit sind diese Informationen im Netz abhörbar.
- Während einer bestehenden Verbindung werden auch alle Daten im Klartext übertragen.
- Beim Verbindungs-Aufbau findet keine Authentifizierung (d. h. Überprüfung der Identität) der beteiligten Rechner statt. Beide Seiten können sich also nicht sicher sein, daß der Kommunikations-Partner nicht ein Angreifer ist, der eine falsche Identität vorgibt ("man in the middle attac"). Angreifer können sich in eine bestehende Verbindung einklinken und dann den Daten-Strom nicht nur mithören, sondern auch verändern!

Deshalb wird heute in der Regel die **Secure Shell (SSH)** verwendet (die Verwendung von telnet als Test-Tool bleibt natürlich erhalten).

Die Secure Shell (SSH)

Bei der Entwicklung von SSH wurde besonderer Wert auf folgende Punkte gelegt:

- Einfache Bedienung, damit Benutzer(innen) nicht von der Verwendung von SSH abgeschreckt werden. Darüberhinaus besitzt SSH einige Funktionen, die die Nutzung im Vergleich zu den r-Kommandos sogar erleichtert (z. B. automatische X-Window-Authorisierung und automatisches Setzen der DISPLAY-Environment-Variablen).
- SSH *mißtraut* prinzipiell dem potentiell unsicheren Netz, dem Domain-Name-Service (DNS) etc.
- SSH ermöglicht eine *strenge Authentifizierung* der beteiligten Kommunikations-Partner.
- SSH garantiert *Vertraulichkeit* durch eine strenge Verschlüsselung aller Daten (speziell der Paßwörter).
- Nicht nur Dialog-Sitzungen sondern auch beliebige TCP/IP-Ports (besonders X-Window) können durch einen sicheren SSH-Kanal (sogenannter SSH-Tunnel) geleitet werden.
- Optional wird der Datenverkehr zusätzlich komprimiert, was besonders bei schlechten Kommunikations-Verbindungen nützlich ist.
- SSH kann sowohl bei der Übersetzung als auch zur Laufzeit in weitem Rahmen konfiguriert und damit an die lokalen Anforderungen angepaßt werden. SSH-Server und -Clients sind auf allen Betriebssystemen verfügbar (der bekannteste Windows-Client nennt sich "putty").

SSH speichert alle wichtigen Informationen im Unterverzeichnis ".ssh" des Heimatverzeichnisses des jeweiligen Benutzers. Das Directory sollte deshalb nur für den Besitzer der Kennung zugänglich sein (`chmod 700 .ssh`).

Mit SSH kann man:

- Sich auf einem Remote-Rechner für eine interaktive Dialog-Sitzung einloggen (Ersatz für telnet):

```
ssh [SSH_OPTIONS] [-l USER] HOST
ssh [SSH_OPTIONS] [USER@]HOST
```

- Kommandos auf einem Remote-Rechner ausführen (interaktiv oder im Batch-Betrieb):

```
ssh [SSH_OPTIONS] HOST COMMAND [COMMAND_ARGUMENTS]
```

- Dateien zwischen Rechnern kopieren:

```
scp [SCP_OPTIONS] [[USER@]HOST:]FILE [...]
```

Für Windows gibt es als äquivalent das Programm "winscp".

- `xterm` mit einer Shell auf einem Remote-Rechner:

Betriebssystem UNIX/Linux

```
ssh -X [SSH_OPTIONS] [USER@]HOST
```

SSH regelt automatisch die Verwendung von X-Window. Man muß also auf dem Remote-Rechner weder eine X-Window-Authorisierung noch die Environment-Variable `DISPLAY` setzen.

Bereits durch diese einfache Nutzung der SSH werden alle Daten einer Verbindung komplett verschlüsselt. Damit sind die Sicherheits-Probleme von telnet und ftp behoben, d. h. Passwörter und Daten werden nicht mehr im Klartext übertragen und es findet eine strenge Authentifizierung statt.

Bei jeder neuen Verbindung laufen am Anfang folgende Schritte ab:

- Der SSH-Client (`scp` setzt intern ebenfalls auf `ssh` auf) wendet sich an den SSH-Server auf einem Remote-Rechner.
- Der SSH-Server schickt sowohl seinen Host- als auch seinen Server-Public-Key (siehe oben) an den Client.
- Der SSH-Client kann nun überprüfen, ob er den Server kennt, indem er in Tabellen von bekannten Public-Keys nach dem gerade erhaltenen Host-Public-Key sucht. Ist dieser Key noch nicht bekannt, teilt der Client dies dem Benutzer mit und fragt nach, ob die Verbindung trotz der Unsicherheit (die Identität des Servers kann ja nicht überprüft werden) aufgebaut werden soll. In der Regel muss man die Frage mit einem ausgeschriebenen "yes" beantworten (nicht nur Taste "y"). Nun trägt der Client den Public Key des Servers in die Datei `$HOME/.ssh/known_hosts` ein. Bei der nächsten Verbindung ist dann der Remote-Rechner schon bekannt und der Client muß nicht mehr nachfragen.
- Der Client schickt dem Server eine 256 Bit lange Zufallszahl, die mit dem Host- und dem Server-Public-Key des Remote-Rechners verschlüsselt wurde.
- Ab diesem Zeitpunkt werden alle Daten der Verbindung verschlüsselt. Dabei dient die im vorherigen Schritt erzeugte Zufallszahl (bzw. je nach Algorithmus auch nur Teile davon) als Sitzungs-Key. Dieser Sitzungs-Key wird bei jeder Verbindung individuell und zufällig erzeugt. Hat der Remote-Rechner eine falsche Identität vorgespiegelt, kann dem Benutzer nichts passieren: Da die Zufallszahl mit dem Host-Public-Key des richtigen Rechners verschlüsselt wurde, kann auch nur der richtige Rechner diese Zahl wieder entschlüsseln.
- Client und Server führen einen Dialog, durch den der Server die Identität und Berechtigung des Clients überprüfen kann. Ist der Client unbekannt oder nicht berechtigt, wird die Verbindung an dieser Stelle abgebrochen.
- In einem weiteren Dialog wird die neue Sitzung vorbereitet. Dabei werden u. a. der Terminal-Typ und die Art der Sitzung (interaktive Dialog-Sitzung oder Ausführung eines Kommandos) festgelegt.
- Zum Schluß startet der Server eine Login-Shell bzw. führt das gewünschte Kommando aus.

Der Benutzer kann seine Identität nicht nur per Username und Passwort, sondern auch durch eine individuelle RSA-Authentifikation nachweisen (automatischer Login). Dazu ist jedoch auf dem Remote-Rechner ein entsprechender Eintrag in der Datei `$HOME/.ssh/authorized_keys` erforderlich.

Bei SSH kommen kryptographische Verfahren an mehreren Stellen zum Einsatz. Das asymmetrische RSA-Verfahren wird für die Authentifizierung der Kommunikations-Partner und für die sichere Übertragung eines zufällig erzeugten und nur einmal verwendeten Sitzungs-Schlüssel eingesetzt. Dabei werden Key-Längen zwischen 768 und 1024 Bit empfohlen. Zur Verschlüsselung der Daten während der Verbindung wird wegen des höheren Durchsatzes ein symmetrisches Verfahren verwendet. Je nach Installation der SW hat der Benutzer i.a. die Auswahl zwischen verschiedenen Algorithmen: DES (Data Encryption Standard) mit 56 Bit Schlüssellänge, 3DES (Tripple DES) mit 112 Bit Schlüssellänge, IDEA (International Data Encryption Algorithm) mit

128 Bit Schlüssellänge, Blowfish mit 128 Bit Schlüssellänge und Arcfour mit 128 Bit Schlüssellänge.

Folgende Optionen werden bei `ssh` häufiger verwendet:

- `-l USER`: Man arbeitet auf dem Remote-Rechner unter der angegebenen Kennung. Diese Option ist immer dann erforderlich, wenn die Kennungen auf den beteiligten Rechnern unterschiedliche Namen haben. Alternativ kann man die Kennung wie bei einer Mailadresse getrennt durch "@" vor den Hostnamen setzen.
- `-n` bzw. `-f`: Die Eingabe von `ssh` (d.h. 'stdin') wird nach `/dev/null` umgelenkt. Dies ist i.a. dann erforderlich, wenn `ssh` im Hintergrund laufen soll. Bei `-n` wird die Eingabe sofort umgelenkt; bei `-f` erfolgt die Umlenkung erst nach dem Verbindungsaufbau und einer dabei evtl. erforderlichen Passwort-Eingabe (bei `-n` kann kein Passwort eingegeben werden).
- `-C`: Die Daten sollen während der Übertragung komprimiert werden.
- `-v`: Durch Diagnose-Meldungen wird jeweils mitgeteilt, was `ssh` im Augenblick gerade tut. Dies ist sehr hilfreich für den Debug von Problemen beim Verbindungsaufbau, bei der Authentifikation und bei der Konfiguration.
- `-c ALGORITHMUS`: Mit dieser Option kann man den symmetrischen Algorithmus für die Verschlüsselung der Daten spezifizieren. Dabei kann der Algorithmus folgende Werte haben: `idea`, `des`, `3des`, `blowfish`, `arcfour` oder `none`. Bedenken Sie, daß je nach Konfiguration von SSH-Client bzw. -Server nicht immer alle dieser Algorithmen zur Verfügung stehen.

Folgende Optionen werden bei `scp` häufiger verwendet:

- `-l`, `-v`, `-C` und `-c` wie bei `ssh`.
- `-r`: Directories sollen mit dem gesamten Inhalt rekursiv kopiert werden.
- `-p`: Beim Kopieren sollen die Datei-Rechte und die Zeitstempel der Dateien beibehalten werden.
-

RSA-Schlüsselpaare

Die Authentifizierung des Clients kann, wie oben erwähnt, auch durch eine benutzerspezifische RSA-Authentifikation erfolgen. Dazu benötigt der Benutzer jedoch ein individuelles RSA-Key-Paar, das im Directory `$HOME/.ssh` abgelegt und mit dem Kommando `ssh-keygen` verwaltet wird:

- `ssh-keygen [-f KEY_FILE] [-C COMMENT] [-b KEY_LENGTH]`

Mit diesem Aufruf erzeugt man ein neues Key-Paar, wobei folgende Informationen interaktiv abgefragt werden:

- ◆ Datei-Name für die Speicherung des Key-Paares (default: `$HOME/.ssh/identity`). In dieser Datei wird nur der Secret-Key des Paares in einem binären Format abgelegt, der dazugehörige Public-Key wird als ASCII-Text in der Datei `FILE.pub` gespeichert.
Wurde schon ein Name über die Option `'-f KEY_FILE'` angegeben, entfällt diese Abfrage.
- ◆ Passwort, mit dem der Secret-Key verschlüsselt wird. Der Secret-Key wird i.a. zur Sicherheit verschlüsselt gespeichert. Dafür muß man bei jeder Verwendung des individuellen Secret-Keys dieses Passwort erneut eingeben. Gibt man bei `ssh-keygen` ein leeres Passwort an, wird der Secret-Key nicht verschlüsselt und

Betriebssystem UNIX/Linux

es entfällt die Passwortabfrage beim Remote-Login.

- Damit das neue Key-Paar wirksam wird, muß man noch jeweils auf allen Remote-Rechnern den neuen *Public-Key* an die Datei `$HOME/.ssh/authorized_keys` anhängen. Ab da ist ein SSH-Login ohne Angabe der Login-Daten (Username und Passwort) möglich - lediglich das Passwort für den Secret-Key muss ggf. angegeben werden. Der Secret-Key sollte jedoch *niemals* den lokalen Rechner verlassen!

Hinweise: Die Erzeugung des Key-Paares erfolgt automatisch und kann *nicht* reproduziert werden. Aus Sicherheitsgründen sollte man das Passwort für den Secret-Key *niemals* als Option auf der Kommandozeile von `ssh-keygen` angeben (das kann jeder nämlich per `ps`-Kommando sehen)! Mit der Option `'-b KEY_LENGTH'` kann man die Länge der Schlüssel festlegen (default: 1024). Mit der Option `'-C COMMENT'` kann man zur besseren Übersicht einen Kommentar bzw. einen Namen für das Schlüssel-Paar vergeben.

`ssh-keygen -p [-f KEY_FILE]` ('Passwort') dient zum Ändern des Passworts für die Aktivierung des Secret-Keys und mit `ssh-keygen -c [-f KEY_FILE] [-C COMMENT]` ('Comment') kann der Kommentar geändert werden.

Neben den Kommandos für den "Normalbenutzer" gibt es noch Testkommandos, die recht hilfreich sind, wenn man einen fernen Rechner nicht erreicht oder, um mehr über das Netz zu erfahren.

ping

Falls man mit dem Kommando `ping` zuerst einmal "Ping-Pong" assoziiert, liegt man gar nicht so falsch. Allerdings werden hier keine Zelluloidbälle, sondern Datenpakete hin und her geschickt. Man kann mit `ping` testen, ob ein Rechner im Netz erreichbar ist. Das Programm `ping` erzeugt ICMP-Echo-Request-Pakete, die mit ICMP-Echo-Response-Paketen beantwortet werden, wenn sie das angegebene System erreichen. Das Zielsystem kann durch seinen Systemnamen (falls in der `/etc/hosts` oder im Nameservice enthalten) oder durch seine Internet-Adresse angegeben werden. Durch den einfachen Aufruf

```
$ ping donald
```

erhält man je nach System die Meldung `"donald is alive."` oder es wird pro Sekunde 1 Datenpaket gesendet. Die als Echo zurückkommenden Pakete werden angezeigt. Ab gebrochen wird das Ping-Pong-Spiel durch das Interrupt-Signal (Delete, Ctrl-C). Nach dem Abruch von `ping` wird noch eine kurze Statistik ausgegeben. (die Wortkarge Variante von `ping` muß man durch die Option `"-s"` zur Dauerarbeit bringen). Besonders interessant ist die Angabe "packet loss", also der Prozentsatz der nicht beantworteten Pakete. Bei einer einwand freien Verbindung, insbesondere in einem lokalen Netz, sollte hier eigentlich immer 0% stehen. Im Falle von 100% ist definitiv etwas nicht in Ordnung. Passiert dies bei allen Systemen, so ist das Netz defekt. Beispiel:

```
$ ping www.ee.hm.edu
PING www.ee.hm.edu (129.187.206.140): 56 data bytes
64 bytes from 129.187.206.140: icmp_seq=0 ttl=242 time=48.9 ms
64 bytes from 129.187.206.140: icmp_seq=1 ttl=242 time=41.9 ms
64 bytes from 129.187.206.140: icmp_seq=2 ttl=242 time=41.3 ms
64 bytes from 129.187.206.140: icmp_seq=3 ttl=242 time=39.9 ms
64 bytes from 129.187.206.140: icmp_seq=4 ttl=242 time=44.9 ms
64 bytes from 129.187.206.140: icmp_seq=5 ttl=242 time=42.9 ms
64 bytes from 129.187.206.140: icmp_seq=6 ttl=242 time=45.4 ms
64 bytes from 129.187.206.140: icmp_seq=7 ttl=242 time=40.5 ms
64 bytes from 129.187.206.140: icmp_seq=8 ttl=242 time=41.4 ms
64 bytes from 129.187.206.140: icmp_seq=9 ttl=242 time=42.3 ms

--- www.ee.hm.edu ping statistics ---
10 packets transmitted, 10 packets received, 0% packet loss
round-trip min/avg/max = 39.9/42.9/48.9 ms
```

arp

Das "Address Resolution Protocol" dient der Zuordnung von Internet-Adressen zu Ethernet-Adressen. Zu diesem Zwecke existiert eine Adreßumwandlungstabelle (address-translation table), die normalerweise vom ARP selbständig aktualisiert wird. Mit der Option "-a" wird der aktuelle Inhalt der Tabelle ausgegeben, z.B.:

```
$ arp -a
Net to Media Table
Device  IP Address  -----          Mask          Flags    Phys Addr
-----  -
le0     brokrz.lrz-muenchen.de  255.255.255.255          00:00:a2:0f:76:97
le0     infoserv.rz.fh-muenchen.de  255.255.255.255          00:e0:29:06:18:d3
le0     flynt.rz.fh-muenchen.de  255.255.255.255          00:e0:29:08:49:f1
le0     kobra.rz.fh-muenchen.de  255.255.255.255          00:08:c7:a9:6c:cc
le0     netmon.rz.fh-muenchen.de  255.255.255.255          00:e0:29:0e:83:92
le0     linux4.rz.fh-muenchen.de  255.255.255.255          00:00:c0:93:19:d3
le0     linux5.rz.fh-muenchen.de  255.255.255.255          00:00:c0:37:19:d3
le0     door2.rz.fh-muenchen.de  255.255.255.255          00:00:c0:3f:fb:a7
le0     wapserv  255.255.255.255 SP      08:00:20:23:02:88
le0     sun10.rz.fh-muenchen.de  255.255.255.255          08:00:20:86:ce:5e
le0     kioskl.rz.fh-muenchen.de  255.255.255.255          00:00:c0:60:af:d7
le0     satellit.rz.fh-muenchen.de  255.255.255.255          08:00:20:71:77:b4
le0     kaputt.rz.fh-muenchen.de  255.255.255.255          00:50:56:82:f0:f0
```

Mit Hilfe der Option "-d" können Einträge aus dieser Tabelle gelöscht werden. Die Einträge sind jedoch nicht permanent, sondern nach einer gewissen Zeit verschwinden sie wieder. Daher ist es meistens nicht notwendig einen Eintrag manuell zu entfernen.

netstat

Mit Hilfe des Programms netstat können Status-Information über alle aktiven TCP-, UDP- und IP-Verbindungen, die Routing-Tabelle und eine detaillierte Statistik der TCP/IP-Daten ausgegeben werden. Bei der Fehlersuche kann sich dieses Programm ebenfalls als durchaus nützlich erweisen. So wird mit der Option "-i" eine Statistik über die Benutzung der Schnittstellen ausgegeben. Die Statistik zeigt die Namen der installierten Schnittstellen, die "Maximal transmission unit" als die maximale Paketgröße des Netzwerks, das Netzwerk, zu dem die Schnittstelle führt und die Adresse der Schnittstelle.

```
Kernel Interface table
Iface  MTU Met  RX-OK RX-ERR RX-DRP RX-OVR  TX-OK TX-ERR TX-DRP TX-OVR Flags
lo     3584  0     220   0       0       0     220   0       0       0 BLRU
eth0   1500  0       0     0       0       0       0     0       0       0 BRU
```

Möchte man die Angaben numerisch, so verwendet man `netstat -in`. Von besonderem Interesse sind die letzten fünf Spalten. Hier werden die Anzahl von empfangenen und gesendeten Paketen, die Anzahl der dabei auftretenden Fehler, sowie die Anzahl der Kollisionen ausgegeben, in die das System verwickelt waren. Sind die Zahlen `lpkts` und `Opkts` gleich Null oder ist `Opkts` gleich `Oerrs`, so liegt ein mehr oder weniger gravierendes Hardware-Problem vor. Die Anzahl der Kollisionen sollte bei jedem System im Netz unter 5% von `Opkts` liegen. In diesem Fall arbeiten die Netzwerkschnittstellen effizient. Eine weitere interessante Option des `netstat`-Kommandos ist die Möglichkeit, sich die aktuellen Verbindungen und aktiven Server mittels der Option "-a" anzeigen zu lassen. Bei diesem Aufruf werden zunächst die zur Zeit benutzten Verbindungen ausgegeben. Dies ist dadurch gekennzeichnet, daß in der Spalte (state) der Zustand ESTABLISHED anzeigt wird. Anschließend werden alle aktiven Server Prozesse angezeigt, d.h. alle Server, die zur Zeit erreichbar sind. Ein Auszug aus der Ausgabe von `netstat -a` könnte beispielsweise so aussehen:

```
Active Internet connections (including servers)
```

Betriebssystem UNIX/Linux

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	(State)	User
tcp	0	0	*:netbios-ssn	*:*	LISTEN	root
tcp	0	0	*:nntp	*:*	LISTEN	root
tcp	0	0	*:auth	*:*	LISTEN	root
tcp	0	0	*:sunrpc	*:*	LISTEN	root
tcp	0	0	*:pop3	*:*	LISTEN	root
tcp	0	0	*:www	*:*	LISTEN	root
tcp	0	0	*:finger	*:*	LISTEN	root
tcp	0	0	*:midinet	*:*	LISTEN	root
tcp	0	0	*:http-rman	*:*	LISTEN	root
tcp	0	0	*:btx	*:*	LISTEN	root
tcp	0	0	*:smtp	*:*	LISTEN	root
tcp	0	0	*:telnet	*:*	LISTEN	root
tcp	0	0	*:ftp	*:*	LISTEN	root
tcp	0	0	*:netstat	*:*	LISTEN	root
tcp	0	0	*:systat	*:*	LISTEN	root
tcp	0	0	*:printer	*:*	LISTEN	root
tcp	0	0	*:shell	*:*	LISTEN	root
tcp	0	0	*:login	*:*	LISTEN	root
tcp	0	0	*:exec	*:*	LISTEN	root
udp	0	0	*:rplay	*:*		
udp	0	0	*:netbios-ns	*:*		
udp	0	0	*:sunrpc	*:*		
udp	0	0	*:ntalk	*:*		
udp	0	0	*:talk	*:*		
udp	0	0	*:syslog	*:*		
raw	0	0	*:1	*:*		

Active UNIX domain sockets

Proto	RefCnt	Flags	Type	State	Inode	Path
unix	1	[ACC]	SOCK_STREAM	LISTENING	417	/dev/log
unix	2	[]	SOCK_STREAM	CONNECTED	440	
unix	2	[]	SOCK_STREAM	UNCONNECTED	441	/dev/log
unix	2	[]	SOCK_STREAM	CONNECTED	499	
unix	2	[]	SOCK_STREAM	UNCONNECTED	500	/dev/log
unix	2	[]	SOCK_STREAM	CONNECTED	517	
unix	2	[]	SOCK_STREAM	UNCONNECTED	518	/dev/log

Die erste Spalte enthält das Transportprotokoll. Die zweite und dritte Spalte sagen etwas über die Anzahl der Bytes in der Empfangs- bzw. Sende-Warteschlange aus. Die nächsten beiden Spalten geben lokale und ferne Adressen einer Verbindung an. Diese Adressen bestehen aus der Internet-Adresse und der Portnummer der Kommunikationspartner. Ist der Rechner in der `/etc/hosts` bzw. der Dienst in der `/etc/services` eingetragen, so werden statt der Adressen Rechnername bzw. der Name des Services aus gegeben. Dies läßt sich durch den Aufruf von `netstat -in` verhindern. Handelt es sich um einen Eintrag für einen aktiven Server, so wird die lokale Adresse in der Form `"*:<portnummer>"` und ferne Adressen in der Form `"*:*"` angegeben. Diese Art der Ausgabe zeigt an, daß der entsprechende Dienst bereit ist. Bei TCP-Diensten zeigt zusätzlich die letzte Spalte an, daß der Server auf LISTEN aesetzt ist. Kommt für einen speziellen Dienst keine Verbindung zustande, obwohl andere Programme (z.B. ping) funktionieren. so kann man mittels `netstat -a` auf dem Zielsystem überprüfen, ob der Server dort aktiv ist. Nur dann kann eine entsprechende Verbindung überhaupt aufgebaut werden. Der Aufruf von `netstat` mit der Option `-r` listet die Routing-Tabelle auf und `netstat -s` liefert eine detaillierte Statistik der TCP/IP-Daten. `netstat -p` zeigt zusätzlich an, welche Programme zu welcher Netzverbindung gehören (nur als User "root").

traceroute

Um festzustellen, welchen Weg die Datenpakete zu einem fernen Rechner nehmen und wie "gut" die Verbindung dorthin ist, kann man 'traceroute' einsetzen. Das Programm schickt UDP-Pakete mit unterschiedlicher "Lebensdauer" an einen unbenutzten Port und wertet so die Fehlermeldungen der einzelnen Router und Gateways aus. Dem Kommando wird wie bei Ping nur der Rechnername oder eine IP-Nummer als Parameter übergeben. Für jeden Gateway wird dann auf dem Bildschirm eine

Zeile ausgegeben:

```
Zähler Gateway-Name Gateway-IP-Nummer "round-trip"-Zeit (3 Werte)
```

Traceroute sendet jeweils drei Datenpakete. Wenn auf ein Paket keine Antwort erfolgt, wird ein Sternchen (*) ausgegeben. Ist ein Gateway nicht erreichbar, wird statt einer Zeitangabe '!N' (network unreachable) oder '!H' (host unreachable) ausgegeben. Man kann so feststellen, wo eine Verbindung unterbrochen ist, und auch, welchen Weg die Daten nehmen - wo also der Zielrechner in etwa steht. Bei grafischen Benutzerschnittstellen erfolgt die Parameterangabe über Dialogfelder und nicht in der Kommandozeile.

```
$ traceroute www.linux.org
traceroute to www.linux.org (198.182.196.56), 30 hops max, 40 byte packets
 1 space-gw2m (194.97.64.8)  2.758 ms  3.637 ms  2.491 ms
 2 Cisco-M-IV.Space.Net (195.30.0.123)  6.413 ms  4.118 ms  4.107 ms
 3 Cisco-M-Fe0-0.Space.Net (195.30.0.126)  4.826 ms  4.508 ms  5.53 ms
 4 Cisco-ECRC-H1-0.Space.Net (193.149.44.2)  5.977 ms  6.273 ms  20.832 ms
 5 munich-ebs2-s0-0-0.ebone.net (192.121.158.189)  14.415 ms  17.018 ms  8.575 ms
 6 newyork-ebs1-s5-0-0.ebone.net (195.158.224.21)  137.35 ms  139.103 ms  138.14 ms
 7 serial0-0-1.br1.nyc4.ALTER.NET (137.39.23.81)  137.132 ms  141.742 ms  141.207 ms
 8 134.ATM2-0.XR1.NYC4.ALTER.NET (146.188.177.178)  135.375 ms  128.12 ms  165.913 ms
 9 189.ATM3-0.TR1.EWR1.ALTER.NET (146.188.179.54)  141.83 ms  144.798 ms  362.469 ms
10 105.ATM4-0.TR1.DCA1.ALTER.NET (146.188.136.185)  145.321 ms  147.889 ms  152.43 ms
11 299.ATM6-0.XR1.TCO1.ALTER.NET (146.188.161.169)  354.577 ms  133.535 ms  348.647 ms
12 193.ATM8-0-0.GW2.TCO1.ALTER.NET (146.188.160.49)  152.444 ms  369.313 ms  150.106 ms
13 uu-peer.oc12-core.ai.net (205.134.160.2)  365.008 ms  509.81 ms  144.898 ms
14 border-ai.invlogic.com (205.134.175.254)  270.065 ms  341.586 ms  153.441 ms
15 router.invlogic.com (198.182.196.1)  356.496 ms  506.371 ms  532.983 ms
16 www.linux.org (198.182.196.56)  584.957 ms  300.612 ms  380.004 ms
```

7.2.4 Binärdaten per Mail (oder News) versenden

Da nach wie vor 7-Bit-ASCII als kleinster gemeinsamer Standard für News und Mail gilt, lassen sich Binärdateien nicht ohne weiteres posten. Abhilfe schaffen hier die Programme UUENCODE und UUDECODE, mit deren Hilfe binäre Daten auf den Bereich der druckbaren ASCII-Zeichen (Großbuchstaben, Ziffern und Sonderzeichen) abbilden lassen. Es werden also Bytes in 6-Bit-Worten codiert und in Zeilen umbrochen. Die mit UUENCODE erzeugte Datei ist nun zwar größer als die Ursprungsdatei, sie läßt sich aber problemlos per News oder Mail verbreiten. Die "uuencodete" Datei enthält am Anfang eine Zeile mit Zugriffsrechte und den Dateinamen:

```
begin 644 camera.wav
```

Danach folgen die codierten Daten und als Abschluß eine Zeile

```
end
```

Eine weitere Form der Codierung findet man bei der Verbreitung von Programmquellen und anderen zusammengehörenden Texten (Programmquellen bestehen in der Regel aus mehreren Quelldateien, Manualpages, Makefile, usw.). Diese Dateien kann man zu sogenannten 'Shell-Archiven' zusammenfassen. Hier wird ein komplettes Shell-Skript gepostet, das die Dateien in Form einzelner Here-Dokumente enthält. Man startet das Skript, wodurch die einzelnen Dateien ausgepackt und meist auch noch auf Korrektheit (Prüfsumme) geprüft werden. Hier als Beispiel der Anfang eines Shell-Archivs:

```
#!/bin/sh
# This is a shell archive. Remove anything before this line, then
# feed it into a shell via "sh file" or similar. To overwrite
```

Betriebssystem UNIX/Linux

```
# existing files, type "sh file -c".
# Contents: bells.sh hells-bells.au.uu
PATH=/bin:/usr/bin:/usr/ucb ; export PATH
echo If this archive is complete, you will see the following message:
echo ' "shar: End of archive."'
if test -f 'bells.sh' -a "${1}" != "-c" ; then
    echo shar: Will not clobber existing file \"'bells.sh'\"
else
    echo shar: Extracting \"'bells.sh'\" \ (581 characters\ )
sed "s/^X//" >'bells.sh' <<'END_OF_FILE'
X#!/bin/
X
XBELL=/usr/local/sounds/hells-bells.au
XPLAY=/usr/local/bin/play
XVOL=40 XDATE=`date +%H:%M`
XMINUTE=`echo $DATE
| sed -e 's/.*://'\`
XHOURL=`echo $DATE | sed -e 's/:.*//'\`
X
....
```

Um mehrere Dateien in einer einzigen zusammenzufassen verwendet man in der Regel das tar-Kommando (Tape ARchive), nur wird hier nicht auf das Band geschrieben, sondern alle Dateien und Verzeichnisse in eine einzige Datei geschrieben (Option "f"). Analog kann ein Archiv dann wieder "ausgepackt" werden.

```
tar cvf datei(en) archiv    Archiv erzeugen (c = create)
tar xvf archiv              Archiv auspacken (x = extract)
tar tvf archiv              Archiv ansehen
(das "v" steht wieder mal für "verbose")
```

Um z. B. alle Dateien des Verzeichnisses 'bin' (und dessen Unterverzeichnisse) in einem Archiv zu sichern, geben das Kommando:

```
tar cvf bin binaeres.tar
```

tar kann aber auch verwendet werden, um ganze Dateibäume auf der Platte zu verschieben:

```
(cd Quelldir ; tar cvf - ) | (cd Zieldir ; tar xvfp - )
```

Damit Übertragungszeit beim ftp gespart wird, kann man nun das Archiv noch komprimieren, wozu bei fast allen Systemen die Kommandos pack/unpack bzw. compress/uncompress existieren:

```
pack datei
```

komprimiert die Datei und ersetzt sie durch die komprimierte Version, wobei an den Namen die Endung ".z" angehängt wird. Mit

```
unpack datei.z
```

wird die Datei wiederhergestellt.

```
compress datei
```

komprimiert die Datei und ersetzt sie durch die komprimierte Version, wobei an den Namen die Endung ".Z" angehängt wird. Mit

```
uncompress datei.Z
```

wird die Datei wiederhergestellt, wobei das Ergebnis von `compress` wesentlich kompakter als bei `pack` ist.

In letzter Zeit wurde der GNU-Zip zum Standard. Er liefert bessere Ergebnisse als `compress` und kann auch mit `pack` oder `compress` bearbeitete Dateien bearbeiten. Die Aufrufe lauten:

```
gzip datei
```

zum Komprimieren und

```
gunzip datei oder gzip -d datei
```

zum Dekomprimieren.

7.3 NFS - Network File System

NFS erlaubt das Einbinden einer beliebigen Platten-Partition eines Computers in das Dateisystem eines anderen Rechners. Dazu wird entweder per `mount`-Befehl oder über die Datei `/etc/fstab` bzw. `/etc/vfstab` angegeben, welches Verzeichnis eines fernen Rechners auf welches lokale Verzeichnis gemountet werden soll.

Die einzige Konfigurationsdatei für den NFS-Dämon des NFS-Servers ist die Datei `/etc/exports` mit folgendem Format:

```
Verzeichnis-Pfad    Rechnernamen (Optionen)
```

Links steht das Verzeichnis, das der NFS-Server exportieren soll, beispielsweise `/home/public` oder `/cdrom`. In der Mitte stehen die Rechner, die Zugriff auf das Verzeichnis haben sollen, und danach in Klammern die Optionen. ACHTUNG: Beachten Sie das Leerzeichen zwischen den Rechnernamen und den Optionen! Nach jeder Änderung der Datei müssen Sie den Portmapper neu starten und dann den NFS-Dämon neu einlesen lassen.

Die zugriffsberechtigten Client-Rechner können Sie auf drei Arten definieren:

- Ein einzelner Rechnername oder eine einzelne IP-Nummer. Damit gestatten Sie nur diesem einen Rechner den Zugriff auf das Verzeichnis. Wenn Sie den Rechnernamen angeben, sollte in der Datei `/etc/hosts` seinem Namen eine IP-Adresse zugeordnet sein.
- Domain-Eintrag mit Jokerzeichen (`*` oder `?`). Damit können Sie allen Rechnern einer Domain den Zugriff gestatten, z. B.: `*.netzmafia.de`
- IP-Netzwerknummern. Durch Eingabe eines Subnetzes mit Netzmaske. Wenn Sie z. B. `192.168.253.255/255.255.255.255` angeben, haben alle Rechner im Subnetz `192.168.253.0` Zugriff auf das Verzeichnis.

Die gebräuchlichsten Optionen sind:

- `rw`: Read-Write gibt den Clients Lese- und Schreibrechte im Verzeichnis.
- `ro`: Read-Only gibt den Clients nur Leserecht (Voreinstellung).
- `noaccess`: Verbietet Clients den Zugriff auf Unterverzeichnisse.
- `root-squash`: Dateien mit User/Group `root` werden bei den Clients einem anonymen Eigentümer und einer anonymen Gruppe zugeordnet.
- `no-root-squash`: Das Gegenteil zu obiger Option.

Betriebssystem UNIX/Linux

Im folgenden Beispiel sollen folgende Zugriffe möglich sein: Auf die erste CD-ROM bekommen die Clients nur Lesezugriff. Der Rechner `boss.netzmafia.de` benötigt root-Zugriff auf `/install` `knecht.netzmafia.de` darf ebenfalls auf `/install` zugreifen, allerdings ohne daß Dateien des Benutzers `root` als solche erscheinen. Die Home-Verzeichnisse aller Benutzer auf dem Server exportiert der Server an alle Rechner im Subnetz, damit die Benutzer auf allen Clients das gleiche Home-Verzeichnis bekommen:

```
# /etc/exports
#
/cdrom      *.netzmafia.de (ro)
/install    boss.netzmafia.de (rw,no-root-squash) \
            knecht.netzmafia.de (ro,root-squash)
/home       192.168.253.255/255.255.255.255
```

Beim Client werden die Verzeichnisse unter Angabe des Servernamens (durch Doppelpunkt getrennt) eingebunden. Das kann entweder per `mount`-Kommando geschehen, z. B. durch:

```
mount nfs.netzmafia.de:/cdrom -t nfs /opt/cdrom
```

oder in der Datei `/etc/fstab`, z. B.:

```
.
.
nfs.netzmafia.de:/cdrom    /opt/cdrom  nfs  ro      0 0
nfs.netzmafia.de:/home    /home       nfs  defaults 0 0
.
.
```



[Zum Inhaltsverzeichnis](#)



[Zum nächsten Abschnitt](#)

Copyright © FH München, FB 04, Prof. Jürgen Plate

Letzte Aktualisierung: 07. Dez 2011



Vorlesung "UNIX"

von Prof. Jürgen Plate

8 Shell-Programmierung

Die Shell dient nicht nur der Kommunikation mit dem Bediener, sondern sie kennt die meisten Konstrukte einer Programmiersprache. Es lassen sich Anweisungen in einer Textdatei speichern, die dann wie ein beliebiges anderes UNIX-Kommando aufgerufen werden kann. Solche Dateien nennt man 'Shell-Script' oder 'shell-Script'. Ein Shell-Script kann auf zwei Arten aufgerufen werden:

- Über eine Sub-Shell: `sh Dateiname`
- Über den Namen, wenn Execute-Berechtigung gesetzt ist (`chmod u+x Dateiname`):
`Dateiname`

Das Shell-Script wird mit einem Editor erstellt und kann alle Möglichkeiten der Shell nutzen, die auch bei der interaktiven Eingabe möglich sind. Insbesondere kann auch die Umleitung der Ein-/Ausgabe wie bei einem Binärprogramm erfolgen. Selbstverständlich lassen sich auch innerhalb eines Scripts weitere Shell-Scripts aufrufen. Zusätzlich kennt die Shell einige interne Befehle. Dem Shell-Script können Parameter übergeben werden, es ist damit universeller verwendbar.

8.1 Testen von Shell-Scripts

Die Ersetzungsmechanismen der Shell machen es manchmal nicht leicht, auf Anhieb korrekt funktionierende Scripts zu erstellen. Zum Testen bieten sich daher einige Möglichkeiten an:

- Einfügen von echo-Kommandos anstelle der vorgesehenen Kommandos. Man kann so die Ersetzungen der Shell beim Aufruf kontrollieren.
- Aufruf über Subshell mit Optionen:
 - ◆ `-v` (verbose) Die Shell gibt alle bearbeiteten Befehle aus.
 - ◆ `-x` (execute) Die Shell gibt die Ersetzungen aus.
 - ◆ `-n` (execute) Die Shell gibt die Befehle zwar aus, sie werden jedoch nicht ausgeführt.Typischer Kommandoaufruf: `sh -vx Dateiname`
- Nur "kritische" Kommandos (z. B. `rm`) erst mal durch davorgesetzte echo-Befehle "entschärfen".
- Ein Doppelpunkt vor einer Zeile wirkt wie ein Kommentar (Kommando wird nicht ausgeführt), aber die Parametersubstitution funktioniert.

Merke: Durch Testen kann nur die Fehlerhaftigkeit von Scripts nachgewiesen, aber nicht deren Korrektheit bewiesen werden.

8.2 Kommentare in Shellscripits

Wie Programme müssen auch Shellscripits kommentiert werden. Kommentare werden durch das Zeichen '#' eingeleitet. Alles was in einer Zeile hinter dem '#' steht, wird als Kommentar betrachtet (Übrigens betrachten auch nahezu alle anderen Unix-Programme das '#' als Kommentarzeichen in Steuer- und Parameterdateien). Leer- und Tabulatorzeichen können normalerweise in beliebiger Anzahl verwendet werden. Da die Shell Strukturen höherer Programmiersprachen enthält, ist durch Einrücken eine übersichtliche Gestaltung der Scripts möglich. Auch eingestreute Leerzeilen sind fast überall erlaubt.

Bei vielen Scripts findet man eine Sonderform der Kommentarzeile zu Beginn, die beispielsweise so aussieht:

```
#!/bin/sh
```

Durch diesen Kommentar wird festgelegt, welches Programm für die Ausführung des Scripts verwendet wird. Er wird hauptsächlich bei Scripts verwendet, die allen Benutzern zur Verfügung stehen sollen. Da möglicherweise unterschiedliche Shells verwendet werden, kann es je nach Shell (sh, csh, ksh,...) zu Syntaxfehlern bei der Ausführung des Scripts kommen. Durch die Angabe der ausführenden Shell wird dafür gesorgt, daß nur die "richtige" Shell verwendet wird. Die Festlegung der Shell stellt außerdem einen Sicherheitsmechanismus dar, denn es könnte ja auch ein Benutzer eine modifizierte Shell verwenden. Neben den Shells können auch andere Programme zur Ausführung des Scripts herangezogen werden; häufig sind awk- und perl-Scripts.

8.3 Shell-Variable

8.3.1 Allgemeines

Variable sind frei wählbare Bezeichner (Namen), die beliebige Zeichenketten aufnehmen können. Bestehen die Zeichenketten nur aus Ziffern, werden die von bestimmten Kommandos als Integer-Zahlen interpretiert (z. B. `expr`). Bei Variablen der Shell sind einige Besonderheiten gegenüber anderen Programmiersprachen zu beachten. Im Umgang mit Variablen lassen sich grundlegend drei Formen unterscheiden:

- Variablendeklaration
- Wertzuweisung
- Wertreferenzierung

Beispiele zur Verdeutlichung des Sachverhaltes:

Form	Beispiel	Ausgabe
Deklaration	<code>\$ foo=</code>	
Wertzuweisung	<code>\$ foo=bar</code>	
Wertreferenzierung	<code>\$ echo \$foo</code>	bar

Im Allgemeinen werden Variablen in der Shell nicht explizit deklariert. Vielmehr ist der Wertzuweisung die Variablendeklaration implizit enthalten. Wird eine Variable dennoch ohne Wertzuweisung deklariert, so wird bei der Wertreferenzierung ein leerer String ("") zurückgegeben.

Für Variablen gilt allgemein:

- Variable sind frei wählbare Bezeichner (Namen), die mit einem Buchstaben beginnen und bis zu 200 Zeichen lang sein dürfen. Leerzeichen innerhalb von Variablen sind (normalerweise) nicht erlaubt (und schlechter Stil).
- Ist der Wert, d. h. der Inhalt einer Variablen gemeint, wird ein `$`-Zeichen vor den Namen gestellt (siehe oben).
- Mittels spezieller Funktionen kann eine Variable auch numerisch oder logisch interpretiert werden.
- Shell-Scripten dürfen ihrerseits wieder Shell-Scripten oder Programme aufrufen. Dabei muß die Vererbung einer Variablen ausdrücklich festgelegt werden (`-->` Exportieren durch den `export`-Befehl), sonst kann eine andere Shell - oder ein beliebiges anderes Programm - nicht darauf zugreifen.
- Jede Subshell läuft in einer eigenen Umgebung, d. h. Variablendefinitionen (und die Wirkung verschiedener Kommandos) in der Subshell sind nach deren Beendigung wieder "vergessen". Es ist nicht möglich, den Wert einer Variablen aus einem Unterprogramm in die aufrufende Ebene zu übergeben.

- Die Zuweisung eines Wertes an die Variable erfolgt mittels des Gleichheitszeichens:

VAR=Wert	Wert ist ein String (ohne Leerzeichen und Sonderzeichen)
VAR="Wert"	Wert ist ein String (Ersetzung eingeschränkt, Leerzeichen und Sonderzeichen dürfen enthalten sein)
VAR=`kommando` VAR=\$(kommando)	Wert der Variablen ist die Ausgabe des Kommandos (Newline --> Leerzeichen)

- Es hat sich die Konvention eingebürgert, Variablen zur Unterscheidung von Kommandos groß zu schreiben.
- Soll der Wert der Variablen mit einem String konkateniert werden, ist der Name in geschweifte Klammern einzuschließen - damit die Shell erkennen kann, wo der Variablenname endet. Bei der Zuweisung von Zahlen an Shell-Variable werden führende Leerzeichen und führende Nullen ignoriert.

Beispiele:

Kommando-Eingaben beginnen mit "\$ ".

```
$ VAR="Hello World!"
$ echo $VAR
Hello World!
$ echo '$VAR'
$VAR

$ VAR=`pwd`
$ echo "Aktuelles Verzeichnis: $VAR"
Aktuelles Verzeichnis: /home/plate

$ VAR=`pwd`
$ echo ${VAR}/bin
/home/plate/bin

$ VAR=/usr/tmp/mytmp
$ ls > $VAR
```

Das letzte Beispiel schreibt die Ausgabe von ls in die Datei /usr/tmp/mytmp

Enthält eine Variable ein Kommando, so kann dies Kommando durch Angabe der Variablen ausgeführt werden, z. B.:

```
$ VAR="ls -la"
$ $VAR
```

8.3.2 Quoting von Variablen

Es gelte: VAR=abcdef

\	entwertet nachfolgendes Metazeichen
, ,	entwertet alle dazwischenliegenden Metazeichen echo '\$VAR' liefert als Ausgabe \$VAR
" "	entwertet alle dazwischenliegenden Metazeichen, aber nicht die Variablen- und Kommandosubstitution echo "\$VAR" liefert abcdef

8.3.3 Vordefinierte Variablen:

Beim Systemstart und beim Aufruf der Dateien /etc/profile (System-Voreinstellungen) und .profile (benutzereigene Voreinstellungen), die ja auch Shellscripts sind, werden bereits einige Variablen definiert. Alle aktuell definierten Variablen können durch das Kommando `set` aufgelistet werden. Einige vordefinierte Variablen sind neben anderen:

Variable	Bedeutung
HOME	Home-Directory (absoluter Pfad)
PATH	Suchpfad für Kommandos und Scripts
MANPATH	Suchpfad für die Manual-Seiten
MAIL	Mail-Verzeichnis
SHELL	Name der Shell
LOGNAME USER	Login-Name des Benutzers
PS1	System-Prompt (\$ oder #)
PS2	Prompt für Anforderung weiterer Eingaben (>)
IFS	(internal field separator) Trennzeichen, meist CR, Leerzeichen und Tab
TZ	Zeitzone (z. B. MEZ)

Folgende spezielle Variablen sind definiert:

Variable	Bedeutung	Kommandobeispiel
\$-	gesetzte Shell-Optionen	<code>set -xv</code>
\$\$	PID (Prozeßnummer) der Shell	<code>kill -9 \$\$</code> (Selbstmord)
#!	PID des letzten Hintergrundprozesses	<code>kill -9 \$!</code> (Kindermord)
\$?	Exitstatus des letzten Kommandos	<code>cat /etc/passwd ; echo \$?</code>

8.3.4 Parameterzugriff in Shell-Scripten:

Shell-Scripts können mit Parametern aufgerufen werden, auf die über ihre Positionsnummer zugegriffen werden kann. Die Parameter können zusätzlich mit vordefinierten Werten belegt werden (später mehr). Trennung zweier Parameter durch die in IFS definierten Zeichen.

Positionsparameter	Bedeutung
\$#	Anzahl der Argumente
\$0	Name des Kommandos
\$1	1. Argument
.	.
.	.
.	.
.	.

Betriebssystem UNIX/Linux

\$9	9. Argument
\$@	alle Argumente (z. B. für Weitergabe an Subshell)
\$*	alle Argumente konkateniert (--> ein einziger String)

Zur Verdeutlichung soll ein kleines Beispiel-Shell-Script dienen:

```
#!/bin/sh
echo "Mein Name ist $0"
echo "Mir wurden $# Parameter übergeben"
echo "1. Parameter = $1"
echo "2. Parameter = $2"
echo "3. Parameter = $3"
echo "Alle Parameter zusammen: $*"
echo "Meine Prozeßnummer PID = $$"
```

Nachdem dieses Shell-Script mit einem Editor erstellt wurde, muß es noch ausführbar gemacht werden (`chmod u+x foo`). Anschließend wird es gestartet und erzeugt die folgenden Ausgaben auf dem Bildschirm:

```
$ ./foo eins zwei drei vier
Mein Name ist ./foo
Mir wurden 4 Parameter übergeben
1. Parameter = eins
2. Parameter = zwei
3. Parameter = drei
Alle Parameter zusammen: eins zwei drei vier
Meine Prozeßnummer PID = 3212
$
```

Anmerkung: So, wie Programme und Scripts des UNIX-Systems in Verzeichnissen wie `/bin` oder `/usr/bin` zusammengefaßt werden, ist es empfehlenswert, im Home-Directory ein Verzeichnis `bin` einzurichten, das Programme und Scripts aufnimmt. Die Variable `PATH` wird dann in der Datei `.profile` durch die Zuweisung `PATH=$PATH:$HOME/bin` erweitert. Damit die Variable `PATH` auch in Subshells (d. h. beim Aufruf von Scripts) auch wirksam wird, muß sie exportiert werden:

export PATH

Alle exportierten Variablen bilden das Environment für die Subshells. Information darüber erhält man mit dem Kommando:

set: Anzeige von Shellvariablen und Environmentvariablen

oder

env: Anzeige der Environmentvariablen

In Shellscripts kann es sinnvoll sein, die Variablen in Anführungszeichen ("...") zu setzen, um Fehler zu verhindern. Beim Aufruf müssen Parameter, die Sonderzeichen enthalten ebenfalls in Anführungszeichen (am besten '...') gesetzt werden. Dazu ein Beispiel. Das Script "zeige" enthält folgende Zeile:

```
grep $1 dat.adr
```

Betriebssystem UNIX/Linux

Der Aufruf `zeige 'Hans Meier'` liefert nach der Ersetzung das fehlerhafte Kommando `grep Hans Meier dat.adr`, das nach dem Namen 'Hans' in der Adreßdatei `dat.adr` und einer (vermutlich nicht vorhandenen) Datei namens 'Meier' sucht. Die Änderung von "zeige":

```
grep "$1" dat.adr
```

liefert bei gleichem Parameter die korrekte Version `grep "Hans Meier" dat.adr`. Die zweite Quoting-Alternative `grep '$1' dat.adr` ersetzt den Parameter überhaupt nicht und sucht nach der Zeichenkette "\$1". Das Script "zeige" soll nun enthalten:

```
echo "Die Variable XXX hat den Wert $XXX"
```

Nun wird eingegeben:

```
$ XXX=Test
$ zeige
```

Als Ausgabe erhält man:

```
Die Variable XXX hat den Wert
```

Erst wenn die Variable "exportiert" wird, erhält man das gewünschte Ergebnis:

```
$ XXX=Test
$ export XXX
$ zeige
Die Variable XXX hat den Wert Test
```

Das Script "zeige" enthalte nun die beiden Kommandos:

```
echo "zeige wurde mit $# Parametern aufgerufen:"
echo "$@"
```

Die folgenden Kommandoaufrufe zeigen die Behandlung unterschiedlicher Parameter:

```
$ zeige
zeige wurde mit 0 Parametern aufgerufen:

$zeige eins zwei 3
zeige wurde mit 3 Parametern aufgerufen:
eins zwei 3

$ zeige eins "Dies ist Parameter 2" drei
zeige wurde mit 3 Parametern aufgerufen:
eins Dies ist Parameter 2 drei
```

Die Definition von Variablen und Shell-Funktionen (siehe später) kann man mit **unset** wieder rückgängig machen.

8.3.5 Namens- und Parameterersetzung:

Die einfache Parameterersetzung (textuelle Ersetzung durch den Wert) wurde oben gezeigt. Es gibt zusätzlich die Möglichkeit, Voreinstellungen zu vereinbaren und auf fehlende Parameter zu reagieren. Bei den folgenden Substitutionen kann bei manchen Shell-Varianten der Doppelpunkt hinter "variable" auch fehlen.

- **`\${shellvar:-neuerwert}**: Die Variable `shellvar` ist deklariert:

- a. Die Variable `shellvar` hat einen Wert, dann wird auf diesen Wert referenziert.
- b. Die Variable `shellvar` hat keinen Wert, dann wird bei der Referenzierung der Wert `neuerwert` eingesetzt.
- **`\${shellvar}:=neuerwert`**: Die Variable `shellvar` ist deklariert:
 - a. Die Variable `shellvar` hat einen Wert, dann wird auf diesen Wert referenziert.
 - b. Die Variable `shellvar` hat keinen Wert, dann wird der Variablen `shellvar` der Wert `neuerwert` zugewiesen und bei der Referenzierung der Wert `neuerwert` eingesetzt.
- **`\${shellvar}?neuerwert`**: Die Variable `shellvar` ist deklariert:
 - a. Die Variable `shellvar` hat einen Wert, dann wird auf diesen Wert referenziert.
 - b. Die Variable `shellvar` hat keinen Wert, dann wird die Fehlermeldung `neuerwert` ausgegeben und das Shellsript abgebrochen.
- **`\${shellvar}+neuerwert`**: Die Variable `shellvar` ist deklariert:
 - a. Die Variable `shellvar` hat einen Wert, dann wird der Wert `neuerwert` referenziert.
 - b. Die Variable `shellvar` hat keinen Wert, dann bleibt dieser Zustand erhalten.

Am besten läßt sich das am Beispielen zeigen, für die folgende Vorbesetzungen gelten:

- W1=Hello
- W2 ist nicht definiert
- der Parameter \$1 habe den Wert "abc"

einfache Substitution	<code>\$W1</code>	HELLO
String-Konkatenation	<code>\${W1}HaHa</code>	HelloHaHa
bedingte Substitution	<code>\${W1-"is nich!"}</code> <code>\${W2-"is nich!"}</code>	Hello is nich!
falls Variable undefiniert ist, nimm Parameter 1	<code>\${W1-\$1}</code> <code>\${W2-\$1}</code>	Hello abc
falls Variable undefiniert, nimm \$1 und brich Script ab	<code>\${W1?\$1}</code> <code>\${W2?\$1}</code>	Hello abc <Abbruch>
falls Variable definiert, nimm \$1, sonst nichts	<code>\${W1+\$1}</code> <code>\${W2+\$1}</code>	abc

In Kommandodateien können Variablen auch Kommandonamen oder -aufrufe enthalten, da ja die Substitution vor der Ausführung erfolgt.

8.3.6 Bearbeitung einer beliebigen Anzahl von Parametern

Die Positionsparameter \$1 bis \$9 reichen nicht immer aus. Man denke nur an Scripts, die (ähnlich wie viele Kommandos) beliebig viele Dateinamen auf Parameterposition erlauben sollen. Die Shell-Sripten können mit mehr als neun Parametern versorgt werden - es wird dann mit dem Befehl `shift` gearbeitet:

shift

Eliminieren von \$1, \$2 ... \$n --> \$1 ... \$n-1

Die Prozedur "zeige" enthält folgende Befehle:

```
echo "$# Argumente:"
echo "$*"
shift
```

```
echo "Nach shift:"  
echo "$# Argumente:"  
echo "$*"
```

Der folgende Aufruf von "zeige" liefert:

```
$ zeige eins zwei drei  
3 Argumente:  
eins zwei drei  
Nach shift:  
2 Argumente:  
zwei drei
```

shift wird jedoch viel häufiger verwendet, wenn die Zahl der Parameter variabel ist. Es wird dann in einer Schleife so lange mit shift gearbeitet, bis die Anzahl der Parameter 0 ist:

```
while [ $# -gt 0 ]  
do  
  tuwas mit $1  
  shift  
done
```

8.3.7 Gültigkeit von Kommandos und Variablen

Jeder Kommandoaufruf und somit auch der Aufruf einer Befehlsdatei (Shellscript) hat einen neuen Prozeß zur Folge. Wie wir wissen, wird zwar das Environment des Elternprozesses "nach unten" weitergereicht, jedoch gibt es keine umgekehrten Weg. Auch der Effekt der Kommandos (z. B. Verzeichniswechsel) ist nur innerhalb des Kindprozesses gültig. Im Elternprozeß bleibt alles beim alten. Das gilt natürlich auch für Zuweisungen an Variablen.

Die Kommunikation mit dem Elternprozeß kann aber z. B. mit Dateien erfolgen. Bei kleinen Dateien spielt sich fast immer alles im Cache, also im Arbeitsspeicher ab und ist somit nicht so ineffizient, wie es zunächst den Anschein hat. Außerdem liefert jedes Programm einen Rückgabewert, der vom übergeordneten Prozeß ausgewertet werden kann.

- 0: O. K.
- > 0: Fehlerstatus

Es gibt außerdem ein Kommando, das ein Shell-Script in der aktuellen Shell und nicht in einer Subshell ausführt:

```
. (Dot) Script ausführen
```

Das Dot-Kommando erlaubt die Ausführung eines Scripts in der aktuellen Shell-Umgebung, z. B. das Setzen von Variablen usw.

Damit die Variable auch in Subshells (d. h. beim Aufruf von Scripts auch wirksam wird, muß sie exportiert werden:

```
export PATH
```

Alle exportierten Variablen bilden das Environment für die Subshells.

8.4 Interaktive Eingaben in Shellscrip

Es können auch Shellscrip

```
read variable [variable ...]
```

read liest eine Zeile von der Standardeingabe und weist die einzelnen Felder den angegebenen Variablen zu. Feldtrenner sind die in IFS definierten Zeichen. Sind mehr Variablen als Eingabefelder definiert, werden die überzähligen Felder mit Leerstrings besetzt. Umgekehrt nimmt die letzte Variable den Rest der Zeile auf. Wird im Shell-Script die Eingabe mit < aus einer Datei gelesen, bearbeitet read die Datei zeilenweise.

Anmerkung: Da das Shell-Script in einer Sub-Shell läuft, kann IFS im Script umdefiniert werden, ohne daß es nachher restauriert werden muß. Die Prozedur "zeige" enthält beispielsweise folgende Befehle:

```
IFS=', '
echo "Bitte drei Parameter, getrennt durch Komma eingeben:"
read A B C
echo Eingabe war: $A $B $C
```

Aufruf (Eingabe kursiv):

```
$ zeige
Bitte drei Parameter, getrennt durch Komma eingeben:
eins,zwei,drei
Eingabe war: eins zwei drei
```

8.5 Hier-Dokumente

Die Shell bietet die Möglichkeit, Eingaben für Programme direkt in das Shell-Script mit aufzunehmen - womit die Möglichkeit einer zusätzlichen, externen Datei wegfällt.

Eingeleitet werden Hier-Dokumente mit << und anschließend einer Zeichenfolge, die das Ende des Hier-Dokuments anzeigt. Diese Zeichenfolge steht dann alleine am Anfang einer neuen Zeile (und gehört nicht mehr zum Hier-Dokument). Bei Quoting der Ende-Zeichenfolge (eingeschlossen in "...", '!...'), werden die Datenzeilen von den üblichen Ersetzungsmechanismen ausgeschlossen. Dazu ein Beispiel:

Die Shell-Script "hier" enthält folgende Zeilen:

```
cat << EOT
Dieser Text wird ausgegeben, als ob er von
einer externen Datei kaeme - na ja, nicht ganz so.
Die letzte Zeile enthaelt nur das EOT und wird
nicht mit ausgegeben. Die folgende Zeile wuerde
bei der Eingabe aus einer Datei nicht ersetzt.
Parameter: $*
EOT
```

Aufruf:

```
$ hier eins zwei
Dieser Text wird ausgegeben, als ob er von
einer externen Datei kaeme - na ja, nicht ganz so.
Die letzte Zeile enthaelt nur das EOT und wird
```

Betriebssystem UNIX/Linux

nicht mit ausgegeben. Die folgende Zeile wuerde bei der Eingabe aus einer Datei nicht ersetzt.
Parameter: eins zwei

Außerden wäre bei der Eingabe aus einer Datei die Ersetzung von '\$*' durch die aktuellen Parameter nicht möglich. Hier-Dokumente bieten also weitere Vorteile. Diese Vorteile lassen sich besonders gut ausspielen, wenn man eine Datei mit `ed` bearbeitet und dabei die Editor-Kommandos als Hier-Dokument mitgibt. Durch in die Kommandos eingestreute Variablen wird die ganze Dateibearbeitung auch variabel steuerbar.

Noch ein Beispiel, diesmal die Simulation des Kommandos 'wall'.

```
for X in `who | cut -d' ' -f1`  
do  
write $X << TEXTENDE  
Hallo Leute,  
das Wetter ist schoen. Wollen wir da nicht um 17 Uhr Schluss machen  
und in den Biergarten gehen?  
TEXTENDE  
done
```

Die Variablenersetzung oder andere Substitutionen im Text finden nur statt, wenn das Wort hinter dem Umleitungsoperator keine Quotation (einfache oder doppelte Anführungszeichen, Backslash) enthält. Text und Endmarkierung lassen sich leider nicht einrücken: Führende Leer- oder Tabulatorzeichen würden Bestandteil des Textes, und eine eingerückte Endmarkierung erkennt die Shell nicht. Ohne Einrückungen hingegen sind Scripte oft schlechter lesbar. Die Bourne-Shell verfügt deshalb über den zweiten Operator `<<-`, der Tabulatorzeichen am Zeilenanfang ignoriert.

8.6 Verkettung und Zusammenfassung von Kommandos

Hintereinanderausführung

Will man mehrere Kommandos ausführen lassen, braucht man nicht jedes Kommando einzeln einzugeben und mit der Eingabe des nächsten Kommandos auf die Beendigung des vorhergehenden warten. Die Kommandos werden, getrennt durch Strichpunkt, hintereinander geschrieben:

```
kommando1 ; kommando2; kommando3
```

Sequentielles UND

Das zweite Kommando wird nur dann ausgeführt, wenn das erste erfolgreich war.

```
Kommando1 && Kommando2
```

Für die Bewertung der Abarbeitung wird folgende Wahrheitstabelle verwendet:

Exitstatus Kommando 1	Exitstatus Kommando 2	&&-Verkettung
Exitstatus gleich 0 --> ok	Exitstatus gleich 0 --> ok	Exitstatus gleich 0 --> ok
Exitstatus gleich 0 --> ok	Exitstatus ungleich 0 --> nicht ok	Exitstatus ungleich 0 --> nicht ok
	wird nicht gestartet!	

Exitstatus ungleich 0 --> nicht ok		Exitstatus ungleich 0 --> nicht ok
------------------------------------	--	------------------------------------

Sequentielles EXOR

(EXOR gleich Exklusives ODER) Das zweite Kommando wird nur dann ausgeführt, wenn das erste erfolglos war.

Kommando1 || Kommando2

Für die Bewertung der Abarbeitung wird folgende Wahrheitstabelle verwendet:

Exitstatus Kommando 1	Exitstatus Kommando 2	-Verkettung
Exitstatus gleich 0 --> ok	wird nicht gestartet!	Exitstatus gleich 0 --> ok
Exitstatus ungleich 0 --> nicht ok	Exitstatus gleich 0 --> ok	Exitstatus gleich 0 --> ok
Exitstatus ungleich 0 --> nicht ok	Exitstatus ungleich 0 --> nicht ok	Exitstatus ungleich 0 --> nicht ok

Zusammenfassung von Kommandos

Kommandofolgen lassen sich - analog der Blockstruktur höherer Sprachen - logisch klammern. Das Problem der normalen Hintereinander-Ausführung mit Trennung durch ";" ist die Umleitung von Standardeingabe und Standardausgabe, z. B.:

```
pwd > out ; who >> out ; ls >> out
```

Die Umleitung läßt sich auch auf die Fehlerausgabe erweitern

```
echo "Fehler!" # geht nach stdout
echo "Fehler!" 1>&2 # geht nach stderr
```

Kommandos lassen sich zur gemeinsamen E/A-Umleitung mit {...} klammern:

```
{ Kommando1 ; Kommando2 ; Kommando3 ; ... ; }
```

Wichtig: Die geschweiften Klammern müssen von Leerzeichen eingeschlossen werden!

Die Ausführung der Kommandos erfolgt nacheinander innerhalb der aktuellen Shell, die Ausgabe kann gemeinsam umgelenkt werden, z. B.:

```
{ pwd ; who ; ls ; } > out
```

Die schließende Klammer muß entweder durch einen Strichpunkt vom letzten Kommando getrennt werden oder am Beginn einer neuen Zeile stehen. Die geschweiften Klammern sind ein ideales Mittel, die Ausgabe aller Programme eines Shell-Scripts gemeinsam umzuleiten, wenn das Script beispielsweise mit 'nohup' oder über cron gestartet wird. Man fügt lediglich am Anfang eine Zeile mit der öffnenden Klammer ein und am Schluß die gewünschte Umleitung, z. B.:

```
{
  ...
```

```
...
...
...
} | mailx -s "Output from Foo" $LOGNAME
```

Eine Folge von Kommandos kann aber auch in einer eigenen Subshell ausgeführt werden:

```
( Kommando1 ; Kommando2 ; Kommando3 ; ... )
```

Das Ergebnis des letzten ausgeführten Kommandos wird als Ergebnis der Klammer zurückgegeben. Auch hier kann die Umleitung der Standardhandles gemeinsam erfolgen. Auch dazu ein Beispiel. Bei unbewachten Terminals "bastle" ich gerne an der Datei '.profile' des Users. Eine Zeile

```
( sleep 300 ; audioplay /home/local/sounds/telefon.au ) &
```

ist schnell eingebaut. Fünf Minuten nach dem Login rennt dann jemand zum Telefon (geht natürlich nur, wenn der Computer auch soundfähig ist). Noch gemeiner wäre

```
( sleep 300 ; kill -9 0 ) &
```

Abschließend vielleicht noch etwas Nützliches. Wenn Sie feststellen daß eine Plattenpartition zu klein geworden ist, müssen Sie nach Einbau und Formatierung einer neuen Platte oftmals ganze Verzeichnisbäume von der alten Platte auf die neue kopieren. Auch hier hilft die Kommandoverkettung zusammen mit dem Programm `tar` (Tape ARchive), das es nicht nur erlaubt, einen kompletten Verzeichnisbaum auf ein Gerät, etwa einen Streamer, zu kopieren, sondern auch in eine Datei oder auf die Standardausgabe. Wir verknüpfen einfach zwei `tar`-Prozesse, von denen der erste das Verzeichnis archiviert und der zweite über eine Pipe das Archiv wieder auspackt. Der Trick am ganzen ist, das beide Prozesse in verschiedenen Verzeichnissen arbeiten. Angenommen wir wollen das Verzeichnis `/usr/local` nach `/mnt` kopieren:

```
( cd /usr/local ; tar cf - . ) | ( cd /mnt ; tar xvf - )
```

Der Parameter "f" weist `tar` an, auf eine Datei zu schreiben oder von einer Datei zu lesen. Hat die Datei wie oben den Namen "-", handelt es sich um *stdout* bzw. *stdin*.

8.7 Strukturen der Shell

In diesem Abschnitt werden die Programmstrukturen (Bedingungen, Schleifen, etc.) besprochen. Zusammen mit den Shell-Variablen und den E/A-Funktionen 'echo', 'cat' und 'read' hat man nahezu die Funktionalität einer Programmiersprache. Es fehlen lediglich strukturierte Elemente wie z. B. Arrays und Records, die teilweise in anderen Shells (z. B. Korn-Shell) oder auch in Script-Sprachen realisiert sind.

8.7.1 Bedingungen testen

Das wichtigste Kommando ist 'test', mit dem man mannigfache Bedingungen testen kann.

test Argument

Dieses Kommando prüft eine Bedingung und liefert 'true' (0), falls die Bedingung erfüllt ist und 'false' (1), falls die Bedingung nicht erfüllt ist. Der Fehlerwert 2 wird zurückgegeben, wenn das Argument syntaktisch falsch ist (meist durch Ersetzung hervorgerufen). Es lassen sich Dateien, Zeichenketten und Integer-Zahlen (16 Bit, bei Linux 32 Bit) überprüfen.

Betriebssystem UNIX/Linux

Das Argument von Test besteht aus einer Testoption und einem Operanden, der ein Dateiname oder eine Shell-Variable (Inhalt: String oder Zahl) sein kann. In bestimmten Fällen können auf der rechten Seite eines Vergleichs aus Strings oder Zahlen stehen - bei der Ersetzung von leeren Variablen kann es aber zu Syntaxfehlern kommen. Weiterhin lassen sich mehrere Argumente logisch verknüpfen (UND, ODER, NICHT). Beispiel:

```
test -w /etc/passwd
```

mit der Kommandoverkettung lassen sich so schon logische Entscheidungen treffen, z. B.:

```
test -w /etc/passwd && echo "Du bist ROOT"
```

Normalerweise kann statt 'test' das Argument auch in eckigen Klammern gesetzt werden. Die Klammern müssen von Leerzeichen umschlossen werden:

```
[ -w /etc/passwd ]
```

Die folgenden Operationen können bei 'test' bzw. [...] verwendet werden.

Eigenschaften von Dateien

Ausdruck	Bedeutung
-e <datei >	datei existiert
-r <datei >	datei existiert und Leserecht
-w <datei>	datei existiert und Schreibrecht
-x <datei>	datei existiert und Ausführungsrecht
-f <datei>	datei existiert und ist einfache Datei
-d <datei>	datei existiert und ist Verzeichnis
-h <datei>	datei existiert und ist symbolisches Link
-c <datei>	datei existiert und ist zeichenor. Gerät
-b <datei>	datei existiert und ist blockor. Gerät
-p <datei>	datei existiert und ist benannte Pipe
-u <datei>	datei existiert und für Eigentümer s-Bit gesetzt
-g <datei>	datei existiert und für Gruppe s-Bit gesetzt
-k <datei>	datei existiert und t- oder sticky-Bit gesetzt
-s <datei>	datei existiert und ist nicht leer
-L <datei>	datei ist symbolisches Link
-t <dateikennzahl>	dateikennzahl ist einem Terminal zugeordnet

Vergleiche und logische Verknüpfungen

Vergleich von Zeichenketten	
Ausdruck	Bedeutung

-n <String>	wahr, wenn String nicht leer
-z <String>	wahr, wenn String leer ist
<String1> = <String2>	wahr, wenn die Zeichenketten gleich sind
<String1> != <String2>	wahr, wenn Zeichenketten verschieden sind
Algebraische Vergleiche ganzer Zahlen	
Operator	Bedeutung
-eq	equal - gleich
-ne	not equal - ungleich
-ge	greater than or equal - größer gleich
-gt	greater than - größer
-le	less than or equal - kleiner gleich
-lt	less than - kleiner
Logische Verknüpfung zweier Argumente	
UND	<bedingung1> -a <bedingung2>
ODER	<bedingung1> -o <bedingung2>
Klammern	\(<ausdruck> \)
Negation	! <ausdruck>

8.7.2 Bedingte Anweisung (if - then - else)

Wichtig: Als Bedingung kann nicht nur der test-Befehl, sondern eine beliebige Folge von Kommandos verwendet werden. Jedes Kommando liefert einen Errorcode zurück, der bei erfolgreicher Ausführung gleich Null (true) und bei einem Fehler oder Abbruch ungleich Null (false) ist. Zum Testen einer Bedingung dient die if-Anweisung. Jede Anweisung muß entweder in einer eigenen Zeile stehen oder durch einen Strichpunkt von den anderen Anweisungen getrennt werden. Trotzdem verhält sich eine bedingte Anweisung - oder die Schleifenkonstrukte, die weiter unten behandelt werden - wie eine einzige Anweisung. Somit ergibt sich eine starke Ähnlichkeit mit der Blockstruktur von C oder Pascal. Man kann dies ausprobieren, indem man eine if- oder while-Anweisung interaktiv eingibt. Solange nicht 'fi' bzw. 'done' eingetippt wurde, erhält man den PS2-Prompt ('>').

einseitiges if:

```
if kommandoliste
then
    kommandos
fi
```

zweiseitiges if:

```
if kommandoliste
then
    kommandos
else
    kommandos
fi
```

mehrstufiges if:

```
if kommandoliste1
then
    kommandos
elif kommandoliste2
    then
        kommandos
elif ...
    ...
fi
```

Beispiele für die if-Anweisung:

Es soll eine Meldung ausgegeben werden, falls mehr als 5 Benutzer eingeloggt sind:

```
USERS=`who | wc -l` # Zeilen der who-Ausgabe zählen
if test $USERS -gt 5
then
    echo "Mehr als 5 Benutzer am Geraet"
fi
```

Das geht natürlich auch kürzer und ohne Backtics:

```
if [ $(who | wc -l) -gt 5 ] ; then
    echo "Mehr als 5 Benutzer am Geraet"
fi
```

Man sollte bei der Entwicklung von Scripts aber ruhig mit der Langfassung beginnen und sich erst der Kurzfassung zuwenden, wenn man mehr Übung hat und die Langfassungen auf Anhieb funktionieren. Ein weiteres Beispiel zeigt eine Fehlerprüfung:

```
if test $# -eq 0
then
    echo "usage: sortiere filename" >&2
else
    sort +1 -2 $1 | lp
fi
```

Das nächste Beispiel zeigt eine mehr oder weniger intelligente Anzeige für Dateien und Verzeichnisse. 'show' zeigt bei Dateien den Inhalt mit 'less' an und Verzeichnisse werden mit 'ls' präsentiert. Fehlt der Parameter, wird interaktiv nachgefragt:

```
if [ $# -eq 0 ]
then
    # falls keine Angabe
    # interaktiv erfragen
    echo -n "Bitte Namen eingeben: "
    read DATEI
else
    DATEI=$1
fi
if [ -f $DATEI ]
then
    # wenn normale Datei
    # dann ausgeben
    less $DATEI
elif [ -d $DATEI ]
then
    # wenn aber Verzeichnis
    # dann Dateien zeigen
    ls -CF $DATEI
else
    # sonst Fehlermeldung
    echo "cannot show $DATEI"
fi
```

Das nächste Beispiel hängt eine Datei an eine andere Datei an; vorher erfolgt eine Prüfung der Zugriffsberechtigungen: `append Datei1 Datei2`

```
if [ -r $1 -a -w $2 ]
then
  cat $1 >> $2
else
  echo "cannot append"
fi
```

Beim Vergleich von Zeichenketten sollten möglichst die Anführungszeichen (" ... ") verwendet werden, da sonst bei der Ersetzung durch die Shell unvollständige Test-Kommandos entstehen können. Dazu ein Beispiel:

```
if [ ! -n $1 ] ; then
  echo "Kein Parameter"
fi
```

Ist \$1 wirklich nicht angegeben, wird das Kommando reduziert zu:

```
if [ ! -n ] ; then ....
```

Es ist also unvollständig und es erfolgt eine Fehlermeldung. Dagegen liefert

```
if [ ! -n "$1" ] ; then
  echo "Kein Parameter"
fi
```

bei fehlendem Parameter den korrekten Befehl `if [! -n ""]`

Bei fehlenden Anführungszeichen werden auch führende Leerzeichen der Variablenwerte oder Parameter eliminiert.

Noch ein Beispiel: Es kommt ab und zu vor, daß eine Userid wechselt oder daß die Gruppenzugehörigkeit von Dateien geändert werden muß. In solchen Fällen helfen die beiden folgenden Scripts:

```
#!/bin/sh
# Change user-id
#
if [ $# -ne 2 ] ; then
  echo "usage `basename $0` <old id> <new id>"
  exit
fi
find / -user $1 -exec chown $2 {} ";"
```

```
#!/bin/sh
# Change group-id
#
if [ $# -ne 2 ] ; then
  echo "usage `basename $0` <old id> <new id>"
  exit
fi
find / -group $1 -exec chgrp $2 {} ";"
```

8.7.3 case-Anweisung

Diese Anweisung erlaubt eine Mehrfachauswahl. Sie wird auch gerne deshalb verwendet, weil sie Muster mit Jokerzeichen und mehrere Muster für eine Auswahl erlauben

```
case selector in
  Muster-1) Kommandofolge 1 ;;
  Muster-2) Kommandofolge 2 ;;
```

```

    ....
    Muster-n) Kommandofolge n ;;
esac

```

Die Variable `selector` (String) wird der Reihe nach mit den Mustern "Muster-1" bis "Muster-n" verglichen. Bei Gleichheit wird die nachfolgende Kommandofolge ausgeführt und dann nach der case-Anweisung (also hinter dem `esac`) fortgefahren.

- In den Mustern sind Metazeichen (*, ?, []) erlaubt, im Selektor dagegen nicht.
- Das Muster * deckt sich mit jedem Selektor --> default-Ausgang. Muß als letztes Muster in der case-Konstruktion stehen.
- Vor der Klammer können mehrere Muster, getrennt durch | stehen. Das Zeichen | bildet eine Oder-Bedingung:

```

case selector in
    Muster1)                Kommandofolge1 ;;
    Muster2 | Muster3)     Kommandofolge2 ;;
    *)                     Kommandofolge3 ;;
esac

```

Beispiel 1: Automatische Bearbeitung von Quell- und Objekt-Dateien. Der Aufruf erfolgt mit 'compile Datei'.

```

case $1 in
    *.s) as $1 ;;                # Assembler aufrufen
    *.c) cc -c $1 ;;            # C-Compiler aufrufen
    *.o) cc $1 -o prog ;;       # C-Compiler als Linker
    *) echo "invalid parameter: $1";;
esac

```

Beispiel 2: Menü mit interaktiver Eingabe:

```

while : # Endlosschleife (s. später)
do
tput clear # Schirm löschen und Menütext ausgeben
echo " +-----+"
echo " | 0 --> Ende |"
echo " | 1 --> Datum und Uhrzeit |"
echo " | 2 --> aktuelles Verzeichnis |"
echo " | 3 --> Inhaltsverzeichnis |"
echo " | 4 --> Mail |"
echo " +-----+"
echo "Eingabe: \c" # kein Zeilenvorschub
read ANTW
case $ANTW in
    0) kill -9 0 ;; # und tschuess
    1) date ;;
    2) pwd ;;
    3) ls -CF ;;
    4) elm ;;
    *) echo "Falsche Eingabe!" ;;
esac
done

```

8.7.4 for-Anweisung

Diese Schleifenanweisung hat zwei Ausprägungen, mit einer Liste der zu bearbeitenden Elemente oder mit den Kommandozeilenparametern.

for-Schleife mit Liste:

```
for selector in liste
do
  Kommandofolge
done
```

Die Selektor-Variable wird nacheinander durch die Elemente der Liste ersetzt und die Schleife mit der Selektor-Variablen ausgeführt. Beispiele:

for X in hans heinz karl luise # vier Listenelemente do echo \$X done Das Programm hat folgende Ausgabe:

```
hans
heinz
karl
luise
```

```
for FILE in *.txt # drucke alle Textdateien
do
  # im aktuellen Verzeichnis
  lpr $FILE
done
```

```
for XX in $VAR      # geht auch mit
do
  echo $XX
done
```

for-Schleife mit Kommandozeilen-Parametern

```
for selector
do
  Kommandofolge
done
```

Die Selektor-Variable wird nacheinander durch die Parameter \$1 bis \$n ersetzt und mit diesen Werten die Schleife durchlaufen. Es gibt also \$# Schleifendurchläufe. Beispiel:

Die Prozedur 'makebak' erzeugt für die in der Parameterliste angegebenen Dateien eine .bak-Datei.

```
for FF
do
  cp $FF ${FF}.bak
done
```

8.7.5 Abweisende Wiederholungsanweisung (while)

Als Bedingung kann nicht nur eine "klassische" Bedingung (`test` oder `[]`) sondern selbverständlich auch der Ergebniswert eines Kommandos oder einer Kommandofolge verwendet werden.

```
while Bedingung
do
  Kommandofolge
done
```

Solange der Bedingungsausdruck den Wert 'true' liefert, wird die Schleife ausgeführt. Beispiele:

Warten auf eine Datei (z. B. vom Hintergrundprozeß)

Betriebssystem UNIX/Linux

```
while [ ! -f foo ]
do
sleep 10 # Wichtig damit die Prozesslast nicht zu hoch wird
done
```

Pausenfüller für das Terminal

Abbruch mit DEL-Taste

```
while :
do
tput clear # BS löschen
echo "\n\n\n\n\n" # 5 Leerzeilen
banner $(date '+ %T ') # Uhrzeit groß
sleep 10 # 10s Pause
done
```

Oder auch rückwärts:

```
#!/bin/sh
F=60
tput clear
while [ $X -gt 0 ]
do
X=`expr $X - 1`
tput clear
banner $X
tput bel
sleep 1
done
reboot # darf zum Glueck nur root
```

Umbenennen von Dateien durch Anhängen eines Suffix

```
# Aufruf change suffix datei(en)
if [ $# -lt 2 ] ; then
echo "Usage: `basename $0` suffix file(s)"
else
SUFF=$1 # Suffix speichern
shift
while [ $# -ne 0 ] # solange Parameter da sind
do
mv $1 ${1}.$SUFF # umbenennen
shift
done
fi
```

Umbenennen von Dateien durch Anhängen eines Suffix

Variante 2 mit for

```
# Aufruf change suffix datei(en)
if [ $# -lt 2 ] ; then
echo "Usage: `basename $0` suffix file(s)"
else
SUFF=$1 # Suffix speichern
shift
for FILE
do
mv $FILE ${FILE}.$SUFF # umbenennen
shift
done
fi
```

8.7.6 until-Anweisung

Diese Anweisung ist identisch zu einer `while`-Schleife mit negierter Bedingung. Als Bedingung kann nicht nur eine "klassische" Bedingung (`test` oder `[]`) sondern selbverständlich auch der Ergebniswert eines Kommandos oder einer Kommandofolge verwendet werden.

```
until Bedingung
do
    Kommandofolge
done
```

Die Schleife wird solange abgearbeitet, bis Bedingungsausdruck einen Wert ungleich Null liefert.
Beispiele:

```
# warten auf Datei foo
until [ -f foo ]
do
    sleep 10
done
```

oder Warten auf einen Benutzer:

```
# warten, bis sich der Benutzer hans eingeloggt hat
TT=`who | grep -c "hans"`
until [ $TT -gt 0 ]
do
    sleep 10
    TT=`who | grep -c "hans"`
done
```

```
# warten, bis sich der Benutzer hans eingeloggt hat
# Variante 2 - kuerzer
until [ `who | grep -c "hans"` -gt 0 ]
do
    sleep 10
done
```

8.7.7 select-Anweisung

```
select VAR in Wortliste
do
    Kommandofolge
done
```

Die Select-Kontrollstruktur bietet eine Kombination aus menügesteuerter Verzweigung und Schleife. Die Wortliste wird als numerierte Liste (Menü) auf dem Standardfehlerkanal ausgegeben. Mit dem PS3-Prompt wird daraufhin eine Eingabe von der Tastatur angefordert. Eine leere Eingabe führt zu einer erneuten Anzeige des Menüs.

Wenn ein Wort aus der Wortliste durch die Eingabe seiner Nummer bestimmt wird, führt die Shell die Kommandofolge aus und stellt dabei das ausgewählte Wort in der Variablen VAR und die die Eingabezeile ist aber in der Variablen REPLY zur Verfügung. Wird in der Eingabezeile keine passende Zahl übergeben, ist VAR leer.

Menüteil und Ausführung der Liste werden so lange wiederholt, bis die Schleife mit `break` oder `return` verlassen wird. Es ist möglich, mit `Ctrl-D` das Menü unmittelbar zu verlassen. Wenn die Wortliste fehlt (nur die Zeile `select VAR`), werden stattdessen die Positionsparameter `$0 ... $9` verwendet. Beispiel:

```
export PS3="Ihre Wahl: "
```

```
select EING in eins zwei drei fertig
do
  echo "EING=\$EING\ " REPLY=\$REPLY\ "
  if [ "$EING" = "fertig" ] ; then
    break
  fi
done
```

8.7.8 Weitere Anweisungen

exit

Wie schon bei der interaktiven Shell kann auch eine Shell-Script mit exit abgebrochen werden. Vom Terminal aus kann mit der DEL-Taste abgebrochen werden, sofern das Signal nicht abgefangen wird (siehe trap).

break [n]

Verlassen von n umfassenden Schleifen. Voreinstellung für n ist 1.

continue [n]

Beginn des nächsten Durchgangs der n-ten umfassenden Schleife, d. h. der Rest der Schleife(n) wird nicht mehr ausgeführt. Voreinstellung für n ist 1.

Interne Kommandos

Etliche der besprochenen Shell-Kommandos starten nicht, wie sonst üblich, einen eigenen Prozeß, sondern sie werden direkt von der Shell interpretiert und ausgeführt. Teilweise ist keine E/A-Umleitung möglich. Etliche Kommandos der folgenden Auswahl wurden schon besprochen. Andere werden weiter unten behandelt. Zum Teil gibt es interne und externe Versionen, z. B. 'echo' (intern) und '/bin/echo' (extern).

break	Schleife verlassen
continue	Sprung zum Schleifenanfang
echo	Ausgabe
eval	Mehrstufige Ersetzung
exec	Überlagerung der Shell durch ein Kommando
exit	Shell beenden
export	Variablen für Subshells bekannt machen
read	Einlesen einer Variablen
shift	Parameterliste verschieben
trap	Behandlung von Signalen

set

```
set [Optionen] [Parameterliste]
```

Setzen von Shell-Optionen und Positionsparametern (\$1 ... \$n). Einige Optionen:

- v Gibt die eingelesenen Shell-Eingaben auf dem Bildschirm aus.
- x Gibt alle Kommandos vor der Ausführungen aus (--> zeigt Ersetzungen).
- n Liest die Kommandos von Shell-Scripten, führt sie jedoch nicht aus.

Betriebssystem UNIX/Linux

Der Aufruf von `set` ohne Parameter liefert die aktuelle Belegung der Shell-Variablen. Außerdem kann `set` verwendet werden, um die Positionsparameter zu besetzen.

`set eins zwei drei vier` besetzt die Parameter mit `$1=eins`, `$2=zwei`, `$3=drei` und `$4=vier`. Da dabei auch Leerzeichen, Tabs, Zeilenwechsel und anderes "ausgefiltert" wird (genauer alles, was in der Variablen IFS steht), ist `set` manchmal einfacher zu verwenden, als die Zerlegung einer Zeile mit `cut`. Die Belegung der Parameter kann auch aus einer Variablen (z. B. `set $VAR`) oder aus dem Ergebnis eines Kommandoaufrufs erfolgen. Beispiel:

```
set `date`          # $1=Fri $2=Apr $3=28 $4=10:44:16 $5=MEZ $6=1999
echo "Es ist $4 Uhr"
Es ist 10:44:16 Uhr
```

Aber es gibt Fallstricke. Wenn man beispielsweise den Output von `ls` bearbeiten möchte, gibt es zunächst unerklärliche Fehlermeldungen (`set: unknown option`):

```
ls -l > foo
echo "Dateiname Laenge"
while read LINE
do
  set $LINE
  echo $9 $5
done < foo
rm foo
```

Da die Zeile mit dem Dateityp und den Zugriffsrechten beginnt, und für normale Dateien ein "-" am Zeilenbeginn steht, erkennt `set` eine falsche Option (z. B. "-rwxr-xr-x"). Abhilfe schafft das Voranstellen eines Buchstabens:

```
ls -l > foo
echo "Dateiname Laenge"
while read LINE
do
  set Z$LINE
  echo $9 $5
done < foo
rm foo
```

Weitere Beispiele: Wenn ein Benutzer eingeloggt ist, wird ausgegeben seit wann. Sonst erfolgt eine Fehlermeldung.

```
if HELP=`who | grep $1`
then
  echo -n "$1 ist seit "
  set $HELP
  echo "$5 Uhr eingeloggt."
else
  echo "$1 ist nicht auffindbar"
fi
```

Ersetzen der englischen Tagesbezeichnung durch die deutsche:

```
set `date`
case $1 in
  Tue) tag=Die;;
  Wed) tag=Mit;;
  Thu) tag=Don;;
  Sat) tag=Sam;;
  Sun) tag=Son;;
  *)   tag=$1;;
esac
```

```
echo $tag $3.$2 $4 $6 $5
```

8.7.9 Arithmetik in Scripts

Die `expr`-Anweisung erlaubt das Auswerten von arithmetischen Ausdrücken. Das Ergebnis wird in die Standardausgabe geschrieben. Als Zahlen können 16-Bit-Integerzahlen (beim Ur-UNIX) oder 32-Bit-Integerzahlen (bei LINUX) verwendet werden (bei manchen Systemen auch noch längere Zahlen mit 64 Bit).

Die Bash wurde auch an dieser Stelle erweitert. Mit der doppelten Klammerung `$ ((Ausdruck))` kann man rechnen, ohne ein externes Programm aufzurufen. `expr Ausdruck` und `$ ((Ausdruck))` beherrschen die vier Grundrechenarten:

+	Addition
-	Subtraktion
*	Multiplikation
/	Division
%	Divisionsrest (Modulo-Operator)

Die Priorität "Punkt vor Strich" gilt auch hier. Außerdem können Klammern gesetzt werden. Da die Klammern und der Stern auch von der Shell verwendet werden, müssen diese Operationszeichen immer durch den Backslash geschützt werden: `*`, `\(`, `\)`. Damit die Operatoren von der Shell erkannt werden, müssen sie von Leerzeichen eingeschlossen werden. Zum Beispiel eine Zuweisung der Summe von A und B an X durch:

```
X=`expr $A + $B`
```

oder

```
X=$((expr $A + $B))
```

(Backquotes beachten!) Außerdem sind logische Operationen implementiert, die den Wert 0 für 'wahr' und den Wert 1 für 'falsch' liefern.

<code>expr1 expr2</code>	oder
<code>expr1 & expr2</code>	und
<code>expr1 < expr2</code>	kleiner
<code>expr1 <= expr2</code>	kleiner oder gleich
<code>expr1 > expr2</code>	größer
<code>expr1 >= expr2</code>	größer oder gleich
<code>expr1 = expr2</code>	gleich
<code>expr1 != expr2</code>	ungleich

Beispiele:

```
# Mittelwert der positiven Zahlen, die von stdin gelesen werden
SUM=0
COUNT=0
while read $WERT          # lesen, bis zum ^D
do
```

Betriebssystem UNIX/Linux

```
COUNT=`expr $COUNT + 1`
SUM=`expr $SUM + $WERT`
done
AVINT=`expr $SUM / $COUNT`
echo "Summe: $SUM      Mittelwert: $AVINT"

#Nimm-Spiel, interaktiv
ANZ=0
if test $# -ne 1
then
    echo "Usage: $0 Startzahl"
else
    echo "NIM-Spiel als Shell-Script"
    echo "Jeder Spieler nimmt abwechselnd 1, 2 oder 3 Hoelzer"
    echo "von einem Haufen, dessen Anfangszahl beim Aufruf fest-"
    echo "gelegt wird. Wer das letzte Holz nimmt, hat verloren."
    echo
    ANZ=$1
    while [ $ANZ -gt 1 ]                # bis nur noch 1 Holz
    do                                  # da ist wiederholen
        echo "\nNoch $ANZ Stueck. Du nimmst (1 - 3): \c # Benutzer"
        read N
        if [ $N -lt 1 -o $N -gt 3 ] ; then # Strafe bei Fehleingabe
            N=1
        fi
        ANZ=`expr $ANZ - $N`            # Benutzer nimmt N weg
        if [ $ANZ -eq 1 ] ; then        # Computer muß letztes Holz nehmen
            echo "\nGratuliere, Du hast gewonnen"
            exit                        # Prozedur verlassen
        else
            C=`expr \( $ANZ + 3 \) % 4`  # Computerzug berechnen
            if [ $C -eq 0 ] ; then
                C=1                    # Wenn 0 Verlustposition
            fi
            echo "Es bleiben $ANZ Stueck. Ich nehme ${C}.\c"
            ANZ=`expr $ANZ - $C`        # Computerzug abziehen
            echo " Rest $ANZ"
        fi
    done                                # Dem Benutzer bleibt
    echo "\nIch habe gewonnen"        # das letzte Holz
fi
```

In der Standard-Shell von Linux, der **Bash** ist die Rechnerei noch einfacher. Hier wird der arithmetische Ausdruck einfach in doppelte Klammern eingeschlossen, z. B. `((ANZ = $ANZ - $C))`. Diese bei C entlehnte Aritmetik kann sogar noch mehr, beispielsweise Variablen inkrementieren oder dekrementieren:

```
(( a = 23 )) # Wert zu weisen, C-style,
             # diesmal mit Leerzeichen auf beiden Seiten des "=".

echo "a (initial value) = $a" # 23

(( a++ ))   # Post-increment 'a', C-style.
echo "a (after a++) = $a"    # 24

(( a-- ))   # Post-decrement 'a', C-style.
echo "a (after a--) = $a"    # 23

(( ++a ))   # Pre-increment 'a', C-style.
echo "a (after ++a) = $a"    # 24

(( --a ))   # Pre-decrement 'a', C-style.
echo "a (after --a) = $a"    # 23

# Achtung: In manchen Faellen gibt es Seiteneffekte:
```

```
n=1; (( --n )) && echo "True" || echo "False" # False
n=1; (( n-- )) && echo "True" || echo "False" # True
```

8.7.10 exec [Kommandozeile]

Ähnlich wie beim Dot-Kommando wird keine Subshell erzeugt, sondern die Kommandozeile in der aktuellen Umgebung ausgeführt. Eine erste Anwendung liegt darin, das aktuelle Programm durch ein anderes zu überlagern. Wenn Sie z. B. die Bourne-Shell als Login-Shell haben, aber lieber mit der C-Shell arbeiten, können sie die Bourne-Shell durch die Kommandozeile

```
exec /bin/csh
```

als letzte Zeile in der .profile-Datei durch die C-shell ersetzen (Wenn Sie die C-Shell nur Aufrufen, müssen Sie beide Shells beenden, um sich auszuloggen). Das Kommando entspricht also dem Systemcall exec(). Wird jedoch kein Kommando angegeben, kann die E/A der aktuellen Shell dauerhaft umgeleitet werden. Beispiel:

```
exec 2>fehler
```

leitet alle folgenden Fehlerausgaben in die Datei "fehler" um, bis die Umleitung explizit durch

```
exec 2>-
```

zurückgenommen wird. Es können bei exec auch andere DateideScriptoren verwendet werden. Ebenso kann auch die Dateiumleitung einer Eingabedatei erfolgen, z. B.:

```
exec 3< datei
```

Danach kann mit read <&3 von dieser Datei gelesen werden, bis die Umleitung mit exec 3<- wieder zurückgenommen wird. Man kann also in Shellscripts durch das Einfügen einer exec-Anweisung die Standardausgabe/-eingabe global für das gesamte Script umleiten, ohne weitere Änderungen vornehmen zu müssen (eine andere Möglichkeit wäre die oben beschriebene Verwendung von { }).

8.7.11 eval [Argumente]

Das Kommando eval liest seine Argumente, wobei die üblichen Ersetzungen stattfinden, und führt die resultierende Zeichenkette als Kommando aus. Die Argumente der Kommandozeile werden von der Shell gelesen, wobei Variablen- und Kommandoersetzungen sowie Dateinamenersetzung durchgeführt werden. Die sich ergebende Zeichenkette wird anschließend erneut von der Shell gelesen, wobei wiederum die oben genannten Ersetzungen durchgeführt werden. Schließlich wird das resultierende Kommando ausgeführt. Beispiel (Ausgaben fett):

```
$ A="Hello world!"
$ X='$A'
$ echo $X
$A
$ eval echo $X
Hello world!
```

Der Rückgabestatus von eval ist der Rückgabestatus des ausgeführten Kommandos oder 0, wenn keine Argumente angegeben wurden. Ein weiteres Beispiel:

```
$ cat /etc/passwd | wc -l
76
```

Betriebssystem UNIX/Linux

```
$ foo='cat /etc/passwd'
$ bar=`| wc -l`
$ $foo $bar                # Fehler: $bar ist Argument von cmdl
cat: | : No such file or directory
cat: wc: No such file or directory
cat: -l: No such file or directory
$ eval $foo $bar
76
```

In diesem Beispiel wird zunächst ein einfaches Kommando gestartet, das die Anzahl der Zeilen der Datei /etc/passwd bestimmt. Anschließend werden die beiden Teile des gesamten Kommandos in die zwei Shell-Variablen foo und bar aufgeteilt. Der erste Aufrufversuch \$foo \$bar bringt nicht das gewünschte Ergebnis, sondern lediglich einige Fehlermeldungen, da in diesem Fall der Wert von bar als Argument für foo interpretiert wird ('cat' wird mit den Dateien '/etc/passwd', '|', 'wc' und '-l' aufgerufen). Wird jedoch das Kommando eval auf die Argumente \$foo und \$bar angewendet, werden diese zunächst zur Zeichenkette "cat /etc/passwd | wc -l" ersetzt. Diese Zeichenkette wird dann durch das Kommando eval erneut von der Shell gelesen, die jetzt das Zeichen "|" in der Kommandozeile als Pipesymbol erkennt und das Kommando ausführt. Das Kommando eval wird üblicherweise dazu eingesetzt, eine Zeichenkette als Kommando zu interpretieren, wobei zweifach Ersetzungen in den Argumenten der Kommandozeile vorgenommen werden.

Eine andere Anwendung ist beispielsweise die Auswahl des letzten Parameters der Kommandozeile. Mit \\$\$# erhält man die Parameterangabe (bei fünf Parametern --> \$5). Das erste Dollarzeichen wird von der Shell ignoriert (wegen des '\'), \$# hingegen ausgewertet. Durch eval wird der Ausdruck nochmals ausgewertet, man erhält so den Wert des letzten Parameters:

```
eval \$$#
```

Aber Vorsicht, das klappt nur bei 1 - 9 Parametern, denn z. B. der 12. Parameter führt zu \$12 --> \${1}2. Es lassen sich mit eval sogar Pointer realisieren. Falls die Variable PTR den Namen einer anderen Variablen, z. B. XYZ, enthält, kann auf den Wert von XYZ durch eval \$PTR zurückgegriffen werden, z. B. durch eval echo \\$\$PTR.

8.7.12 trap 'Kommandoliste' Signale

Ausführen der Kommandoliste, wenn eins der angegebenen Signale an den Prozeß (= Shell) gesendet wird. Die Signale werden in Form der Signalnummern oder über ihre Namen (SIGKILL, SIGHUP, ...), getrennt durch Leerzeichen aufgeführt.

Ist die Kommandoliste leer, werden die entsprechenden Signale abgeschaltet. Bei einfachen Kommandos reichen oft auch die Anführungszeichen, um die Shell-Ersetzung zu verhindern.

Signale sind eine Möglichkeit, wie verschiedenen Prozesse, also gerade laufende Programme, miteinander kommunizieren können. Ein Prozeß kann einem anderen Prozeß ein Signal senden (der Betriebssystemkern spielt dabei den Postboten). Der Empfängerprozeß reagiert auf das Signal, z. B. dadurch, daß er sich beendet. Der Prozeß kann das Signal auch ignorieren. Das ist beispielsweise nützlich, wenn ein Shellscript nicht durch den Benutzer von der Tastatur aus abgebrochen werden soll. Mit dem trap-Kommando kann man festlegen, mit welchen Kommandos auf ein Signal reagiert werden soll bzw. ob überhaupt reagiert werden soll.

Neben anderen können folgende Signalnummern verwendet werden:

- 0 SIGKILL Terminate (beim Beenden der shell)
- 1 SIGHUP Hangup (beim Beenden der Verbindung zum Terminal oder Modem)
- 2 SIGINT Interrupt (wie Ctrl-C-Taste am Terminal)

- 3 SIGQUIT Abbrechen (Beenden von der Tastatur aus)
- 9 SIGKILL Kann nicht abgefangen werden - Beendet immer den empfangenden Prozeß
- 15 SIGTERM Termine (Software-Terminate, Voreinstellung)

Die Datei `/usr/include/Signal.h` enthält eine Liste aller Signale.

Beispiele:

```
# Script sperren gegen Benutzerunterbrechung:
trap "" 2 3
```

oder auch

```
# Script sauber beenden
trap 'rm tmpfile; cp foo fileb; exit' 0 2 3 15
```

Bitte nicht das `exit`-Kommando am Schluss vergessen, sonst wird das Script nicht beendet. Wiedereinschalten der Signale erfolgt durch `trap [Signale]`. Ein letztes Beispiel zu `trap`:

```
# Automatisches Ausführen des Shellscripts .logoff beim
# Ausloggen durch den folgenden Eintrag in .profile:
trap .logoff 0
```

Noch eine Demo für das Kommando: Der Programmabbruch wird erst nach 10 Sekunden zur Kenntnis genommen.

```
trap "sleep 10; echo 'Und wech'; exit;" 1 2 3 15
while :
do
    echo schnarch
    sleep 1
done
```

8.7.13 xargs

Das `xargs`-Programm fällt etwas aus dem Rahmen der übrigen in diesem Kapitel behandelten Programme. `xargs` übergibt alle aus der Standardeingabe gelesenen Daten einem Programm als zusätzliche Argumente. Der Programmaufruf wird einfach als Parameter von `xargs` angegeben.

```
xargs Programm [Parameter]
```

Ein Beispiel soll die Funktionsweise klarmachen:

```
$ ls *.txt | xargs echo Textdateien:
```

Hier erzeugt `ls` eine Liste aller Dateien mit der Endung `.txt` im aktuellen Verzeichnis. Das Ergebnis wird über die Pipe an `xargs` weitergereicht. `xargs` ruft `echo` mit den Dateinamen von `ls` als zusätzliche Parameter auf. Der Output ist dann:

```
Textdateien: kap1.txt kap2.txt kap3.txt h.txt
```

Durch Optionen ist es möglich, die Art der Umwandlung der Eingabe in Argumente durch `xargs` zu beeinflussen. Mit der Option `-n <Nummer>` wird eingestellt, mit wievielen Parametern das angegebene Programm aufgerufen werden soll. Fehlt der Parameter, nimmt `xargs` die maximal mögliche Zahl von Parametern. Je nach Anzahl der Parameter ruft `xargs` das angegebene Programm einmal oder mehrmal auf. Dazu ein Beispiel. Vergleich einer Reihe von Dateien nacheinander mit

einer vorgegebenen Datei:

```
ls *.dat | xargs -n1 cmp compare Muster
```

Die Vergleichsdatei "Muster" wird der Reihe nach mittels `cmp` mit allen Dateien verglichen, die auf ".dat" enden. Die Option `-n1` veranlaßt `xargs`, je Aufruf immer nur einen Dateinamen als zusätzliches Argument bei `cmp` anzufügen.

Mit der Option `-i <Zeichen>` ist es möglich, an einer beliebigen Stelle im Programmaufruf, auch mehrmals, anzugeben, wo die eingelesenen Argumente einzusetzen sind. In diesem Modus liest `xargs` jeweils ein Argument aus der Standardeingabe, ersetzt im Programmaufruf jedes Vorkommen des hinter `-i` angegebenen Zeichens durch dieses Argument und startet das Programm. In dem folgenden Beispiel wird das benutzt, um alle Dateien mit der Endung ".txt" in ".bak" umzubenennen.

```
ls *.txt | cut -d. f1 | xargs -iP mv P.txt P.bak
```

Das Ganze funktioniert allerdings nur, wenn die Dateien nicht noch weitere Punkte im Dateinamen haben.

`xargs` ist überall dann besonders notwendig, wenn die Zahl der Argumente recht groß werden kann. Obwohl Linux elend lange Kommandozeilen zuläßt, ist die Länge doch begrenzt. `xargs` nimmt immer soviele Daten aus dem Eingabestrom, wie in eine Kommandozeile passen und führt dann das gewünschte Kommando mit diesen Parametern aus. Liegen weitere Daten vor, wird das Kommando entsprechend oft aufgerufen. Insbesondere mit dem folgenden Kommando sollte `xargs` verwendet werden, da `find` immer den gesamten Dateipfad liefert, also schnell recht lange Argumente weitergibt.

8.7.14 dialog

`dialog` ist ein Programm, welches innerhalb eines Shellsriptes Interaktionen mit dem Benutzer ermöglicht. Unter einer einheitlichen Oberfläche bietet `dialog` eine Vielzahl von Interaktionsmöglichkeiten wie z. B.:

- Informations-Box / Nachrichten-Box
- Ja-Nein-Box
- verschiedene Menu-Boxen
- Text-Box
- Eingabe-Box

`dialog` bietet im einzelnen folgende Interaktionsmöglichkeiten:

1. **infobox: übermittlung einer Nachricht an den Benutzer**
Auf der Oberfläche erscheint ein Fenster mit einer Information.
2. **msgbox: übermittlung einer zu bestätigenden Nachricht an den Benutzer**
Auf der Oberfläche erscheint ein Fenster mit einer Nachricht, die vom Benutzer mit "OK" quittiert werden muß.
3. **yesno: Einfache Benutzerabfrage**
Auf der Oberfläche erscheint ein Fenster mit einer Frage, die vom Benutzer mit "Yes" oder "No" beantwortet werden muß.
4. **menu: Auswahl-Menu für den Benutzer**
Auf der Oberfläche erscheint ein Fenster mit einem Auswahl-Menu, aus dem sich der Benutzer einen Menu-Punkt auswählen kann. Die Auswahl ist auf einen einzigen Menu-Punkt beschränkt.
5. **checklist: Auswahl von mehreren Menu-Punkten**
Auf der Oberfläche erscheint ein Auswahl-Menu, aus dem sich der Benutzer einen oder

mehrere Menu-Punkte auswählen kann.

6. radiolist: Auswahl eines Menu-Punktes mit Vorauswahl

Auf der Oberfläche erscheint ein Auswahl-Menu, bei dem bereits ein Menu-Punkt ausgewählt ist. Der Benutzer kann den Menu-Punkt akzeptieren oder aber einen anderen Menu-Punkt wählen.

7. textbox: Anzeige einer Textdatei

Auf der Oberfläche erscheint ein Fenster mit Scroll-Balken, in dem sich der Benutzer eine Text-Datei ansehen kann.

8. inputbox: Eingabe von Daten

Auf der Oberfläche erscheint ein Fenster, in das der Benutzer Daten eingeben kann.

Aufruf:

```
dialog --title "Fenstertitel" \  
  --backtitle "Hintergrundtitel" \  
  --[infobox | yesno | menu | ...] \  
  Fensterinhalt u. -abmaße
```

aufgerufen. Nach Beendigung von `dialog` durch den Benutzer (Abbruch über Escape-Taste, "OK" / "Yes" bzw. "Cancel" / "No") liefert `dialog` folgende Informationen zurück:

1. exit-status von dialog

Der exit-status von `dialog` wird an `stdout` zurückgegeben. Dabei bedeutet ein Rückgabewert von 0 = Beendigung über "OK"/"Yes", 1 = Beendigung über "Cancel"/"No" und 255 = Beendigung über "Escape-Taste"

2. ausgewählte Menu-Punkte / abgefragte Benutzerdaten

Die vom Benutzer ausgewählten Menu-Punkte bzw. eingegebenen Benutzerdaten werden auf `stderr` zurückgegeben.

Beispiel:

```
dialog --backtitle "$BTITLE" --title "Auswahl-Menu" \  
  --menu "Bitte treffen Sie Ihre Auswahl:" 12 45 3 \  
  "Pizza Regina" "heute besonders köstlich zubereitet" \  
  "vino chianti" "beschränken wir uns auf das wesentliche" \  
  "grappa" "reduced to the max" \  
  2> dialog-dat.tmp
```

Bei Beendigung von `dialog` über "OK" wird der ausgewählte Menu-Punkt über den Standard-Fehlerkanal ausgegeben. Da der Standard-Fehlerkanal in die Datei `dialog-dat.tmp` umgelenkt ist, wird der ausgewählte Punkt demnach in diese Datei geschrieben und kann von dort z. B. mit `read AUSWAHL < dialog-dat.tmp` gelesen werden.

Beim Einbinden von `dialog`-Abfragen in Skripte müssen zwei Dinge besonders beachtet werden:

1. Übergabe von Variablen in Anführungszeichen

Wenn Titel, Hintergrundtitel bzw. Texte in Form von Variablen übergeben werden, so sind die Variablen in Anführungszeichen zu setzen, z. B.: `--title "$TITEL"`

2. Fenstergröße nicht größer als Bildschirmgröße

Bei jedem Aufruf muß `dialog` die Fenstergröße mitgeteilt werden: Höhe in Bildschirmzeilen, Breite in Zeichen. Die sich daraus ergebende Fenstergröße darf keinesfalls größer sein als die Bildschirmgröße!

8.8 Beispiele für Shellscrip

Disk usage

Sie erinnern sich noch an das `du`-Kommando? Mit etwas "Kosmetik" wird die Ausgabe noch aussagekräftiger:

```
#!/bin/sh
du -ks * | sort -rn | head -11
# total          reverse
# kilobyte       numerisch
# summarize
```

Nachbildung des Kommandos seq

`seq` hat zwei Parameter, einen Anfangswert und einen Endwert. Es wird vom Anfangswert bis zum Endwert gezählt.

```
#!/bin/sh
if [ $# -ne 2 -o $1 -gt $2 ]
then
    echo "usage: seq n1 n2 (n1 < n2)"
    exit 1
fi
I=$1
while [ $I -le $2 ]
do
    echo $I
    I=`expr $I + 1`
done
```

Datei verlängern

Weil immer wieder danach gefragt wird: "Wie hänge ich den Inhalt einer Variablen an eine Datei an?":

```
( cat file1 ; echo "$SHELLVAR" ) > file2
```

Telefonbuch

Telefonverzeichnis mit Hier-Dokument. Aufruf `tel Name [Name ..]`:

```
if [ $# -eq 0 ]
then
    echo "Usage: `basename $0` Name [Name ..]"
    exit 2
fi
for SUCH in $*
do
    if [ ! -z $SUCH ] ; then
        grep $SUCH << "EOT"
        Hans 123456
        Fritz 234561
        Karl 345612
        Egon 456123
    EOT
fi
done
```

Kompakte Auflistung des Inhalts eines ganzen Dateibaums

Dies ist ein rekursives Programm. Damit es klappt, muß das folgende Script `dir` über `PATH` erreichbar sein. Aufruf: `dir` Anfangspfad.

```
if test $# -ne 1
then
    echo "Usage: dir Pfad"
    exit 2
fi
cd $1
echo
pwd
ls -CF
for I in *
do
    if test -d $I
    then
        dir $I
    fi
done
```

Auflisten des Dateibaums in grafischer Form

wobei zur Dateiauswahl alle Optionen des `find`-Kommandos zur Verfügung stehen. Aufruf z.B. `tree .` für alle Dateien oder `tree . -type d` für Directories ab dem aktuellen Verzeichnis.

```
# find liefert die vollständigen Pfadnamen (temporäre Datei).
# Mit ed werden die "/"-Zeichen erst zu "|---- " expandiert
# und dann in den vorderen Spalten aus "|---- " ein Leerfeld
# "    |" gemacht.
if test $# -lt 1
then
    echo "Usage: tree Pfadname [Pfadname ...] [find-Options]"
else
    TMPDAT=$0$$
    find $@ -print > $TMPDAT 2>/dev/null
    ed $TMPDAT << "EOT" >/dev/null 2>/dev/null
1,$s/[^\\/*\\//|---- /g
1,$s/---- |/    |/g
w
q
EOT
    cat $TMPDAT
    rm $TMPDAT
fi
```

Mit dem Stream-Editor `sed` kann das sogar noch kompakter formuliert werden:

```
if [ $# -lt 1 ]
then
    echo "Usage: tree Pfadname [Pfadname ...] [find-Options]"
else
    find $@ -print 2>/dev/null | \
    sed -e '1,$s/[^\\/*\\//|---- /g' -e '1,$s/---- |/    |/g'
fi
```

Argumente mit J/N-Abfrage ausführen

Das folgende Script führt alle Argumente nach vorheriger Abfrage aus. Mit "j" wird die Ausführung bestätigt, mit "q" das Script abgebrochen und mit jedem anderen Buchstaben (in der Regel "n") ohne Ausführung zum nächsten Argument übergegangen. Ein- und Ausgabe erfolgen immer über das

Betriebssystem UNIX/Linux

Terminal(-fenster), weil `/dev/tty` angesprochen wird. Das Script wird anstelle der Argumentenliste bei einem anderen Kommando eingesetzt, z. B. Löschen mit Nachfrage durch `rm $(pick *)`

```
# pick - Argumente mit Abfrage liefern
for I ; do
  echo "$I (j/n)? \c" > /dev/tty
  read ANTWORT
  case $ANTWORT in
    j*|J*) echo $I ;;
    q*|Q*) break ;;
  esac
done </dev/tty
```

Sperrern des Terminals während man kurz weggeht

Nach Aufruf von `lock` wird ein Kennwort eingegeben und das Terminal blockiert. Erst erneute Eingabe des Kennwortes beendet die Prozedur. Die Gemeinheit dabei ist, daß sich bei jeder Fehleingabe die Wartezeit verdoppelt.

```
echo "Bitte Passwort eingeben: \c"
stty -echo # kein Echo der Zeichen auf dem Schirm
read CODE
stty echo
tput clear # BS loeschen
trap "" 2 3
banner " Terminal "
banner " gesperrt "
MATCH=""
DELAY=1
while [ "$MATCH" != "$CODE" ]
do
  sleep $DELAY
  echo "Bitte Passwort eingeben: \c"
  read MATCH
  DELAY=`expr $DELAY \* 2` # doppelt so lange wie vorher
done
echo
```

Dateien im Pfad suchen

Das folgende Script bildet das Kommando "which" nach. Es sucht im aktuellen Pfad (durch `PATH` spezifiziert) nach der angegebenen Datei und gibt die Fundstelle aus. An diesem Script kann man auch eine Sicherheitsmaßnahme sehen. Für den Programmaufruf wird der Pfad neu gesetzt, damit nur auf Programme aus `/bin` und `/usr/bin` zugegriffen wird. Bei Scripts, die vom Systemverwalter für die Allgemeinheit erstellt werden, sollte man entweder so verfahren oder alle Programme mit absolutem Pfad aufrufen.

```
#!/bin/sh
# Suchen im Pfad nach einer Kommando-Datei
OPATH=$PATH
PATH=/bin:/usr/bin
if [ $# -eq 0 ] ; then
  echo "Usage: which kommando" ; exit 1
fi
for FILE
do
  for I in `echo $OPATH | sed -e 's/^:/:/' -e 's/:::/:/g \ -e 's/:$/:./`
  do
    if [ -f "$I/$FILE" ] ; then
      ls -ld "$I/$FILE"
    fi
  done
done
```

done

Berechnung von Primfaktoren einer Zahl:

```

echo "Zahl eingeben: \c"; read ZAHL
P=2
while test `expr $P \* $P` -le $ZAHL; do
    while test `expr $ZAHL % $P` -ne 0; do
        if test $P -eq 2; then
            P=3
        else
            P=`expr $P + 2`
        fi
    done
    ZAHL=`expr $ZAHL / $P`
    echo $P
done
if test $ZAHL -gt 1; then
    echo $ZAHL
fi
echo ""

```

Berechnung des Osterdatums nach C.F. Gauss

```

if [ $# -eq 0 ] ; then
    echo "Osterdatum fuer Jahr: \c"; read JAHR
else
    JAHR="$1"
fi
G=`expr $JAHR % 19 + 1`
C=`expr $JAHR / 100 + 1`
X=`expr \( $C / 4 - 4 \) \* 3`
Z=`expr \( $C \* 8 + 5 \) / 25 - 5`
D=`expr $JAHR \* 5 / 4 - $X - 10`
E=`expr \( $G \* 11 + $Z - $X + 20 \) % 30`
if test $E -lt 0; then
    $E=`expr $E + 30`
fi
if [ $E -eq 25 -a $G -gt 11 -o $E -eq 24 ] ; then
    E=`expr $E + 1`
fi
TAG=`expr 44 - $E`
if [ $TAG -lt 21 ] ; then
    TAG=`expr $TAG + 30`
fi
TAG=`expr $TAG + 7 - \( $D + $TAG \) % 7`
if [ $TAG -gt 31 ] ; then
    TAG=`expr $TAG - 31`
    MON=4
else
    MON=3
fi
echo "Ostern $JAHR ist am ${TAG}.${MON}.\n"

```

Statt des expr-Befehls kann bei der Bash auch das Konstrukt $\$((\dots))$ verwendet werden. Das Programm sieht dann so aus:

```

if [ $# -eq 0 ] ; then
    echo "Osterdatum fuer Jahr: \c"; read JAHR
else
    JAHR="$1"
fi
G=$(( $JAHR % 19 + 1 ))
C=$(( $JAHR / 100 + 1 ))
X=$(( ( ( $C / 4 - 4 ) * 3 ))

```

Betriebssystem UNIX/Linux

```
Z=$(( ( ( $C \* 8 + 5 \ ) / 25 - 5 ))
D=$(( ( $JAHR \* 5 / 4 - $X - 10 ))
E=$(( ( ( $G \* 11 + $Z - $X + 20 \ ) % 30 ))
if test $E -lt 0; then
    $E=$(( $E + 30 ))
fi
if [ $E -eq 25 -a $G -gt 11 -o $E -eq 24 ] ; then
    E=$(( $E + 1 ))
fi
TAG=$(( 44 - $E ))
if [ $TAG -lt 21 ] ; then
    TAG=$(( $TAG + 30 ))
fi
TAG=$(( ( $TAG + 7 - \ ( $D + $TAG \ ) % 7 ))
if [ $TAG -gt 31 ] ; then
    TAG=$(( $TAG - 31 ))
    MON=4
else
    MON=3
fi
echo "Ostern $JAHR ist am ${TAG}.${MON}.\n"
```

Wem die Stunde schlägt

Wer im Besitz einer Soundkarte ist, kann sich eine schöne Turmuhr, einen Regulator oder Big Ben basteln. Die folgende "Uhr" hat Stunden- und Viertelstundenschlag. Damit das auch klappt, ist ein Eintrag in der crontab nötig:

```
0,15,30,45 * * * * /home/sbin/turmuhr
```

So wird das Script turmuhr alle Viertelstunden aufgerufen. Es werden zwei Sounddateien verwendet, hour.au für den Stundenschlag und quater.au für den Viertelstundenschlag. Statt des Eigenbau-Programms audioplay kann auch der sox verwendet werden oder man kopiert die Dateien einfach nach /dev/audio. Die Variable VOL steuert die Lautstärke.

```
#!/bin/sh
BELL=/home/local/sounds/hour.au
BELL1=/home/local/sounds/quater.au
PLAY=/usr/bin/audioplay
VOL=60
DATE=`date +%H:%M`
MINUTE=`echo $DATE | sed -e 's/.*://'\`
HOUR=`echo $DATE | sed -e 's/:.*//'\`

if [ $MINUTE = 00 ]
then
    COUNT=`expr \ ( $HOUR % 12 + 11 \ ) % 12`
    BELLS=$BELL
    while [ $COUNT != 0 ];
    do
        BELLS="$BELLS $BELL"
        COUNT=`expr $COUNT - 1`
    done
    $PLAY -v $VOL -i $BELLS
elif [ $MINUTE = 15 ]
then    $PLAY -v $VOL -i $BELL1
elif [ $MINUTE = 30 ]
then    $PLAY -v $VOL -i $BELL1 $BELL1
elif [ $MINUTE = 45 ]
then    $PLAY -v $VOL -i $BELL1 $BELL1 $BELL1
else
    $PLAY -v $VOL -i $BELL1
fi
```

8.9 Shell-Funktionen

Ab System V können in der Bourne-Shell auch Funktionen definiert werden - eine weitere Strukturierungsmöglichkeit. Funktionen können in Shellscripits, aber auch interaktiv definiert werden. Sie lassen sich jedoch nicht wie Variablen exportieren. Sie werden nach folgender Syntax definiert:

```
Funktionsname ()
{
  Kommandofolge
}
```

Steht die schließende geschweifte Klammer nicht in einer eigenen Zeile, gehört ein Strichpunkt davor. Die runden Klammern hinter dem Funktionsnamen teilen dem Kommandozeileninterpreter der Shell mit, daß nun eine Funktion definiert werden soll (und nicht ein Kommando `Funktionsname` aufgerufen wird). Es kann keine Parameterliste in den Klammern definiert werden.

Der Aufruf der Shellfunktion erfolgt durch Angabe des Funktionsnamens, gefolgt von Parametern (genauso wie der Aufruf eines Scripts). Die Parameter werden innerhalb der Funktion genauso, wie beim Aufruf von Shellscripits über \$1 bis \$nn angesprochen. Ein Wert kann mit der Anweisung 'return <Wert>' zurückgegeben werden, er ist über den Parameter \$? abfragbar. Beispiel:

```
isdir () # testet, ob $1 ein Verzeichnis ist
{
  if [ -d $1 ] ; then
    echo "$1 ist ein Verzeichnis" # Kontrolle zum Test
    return 0
  else
    return 1
  fi
}
```

Im Gegensatz zum Aufruf von Shell-Scripts werden Funktionen **in der aktuellen Shell** ausgeführt und sie können bei der Bourne-Shell nicht exportiert werden; die Bash erlaubt dagegen das Exportieren mit `export -f`. Das folgende Beispiel illustriert die Eigenschaften von Shellfunktionen. Die folgende Funktion gibt den Eingangsparameter in römischen Zahlen aus. Dabei wird die Zahl Schritt für Schritt in der Variablen ZAHL zusammengesetzt. Würde man der Funktion ZIFF ein Script verwenden, ginge das nicht, da sich der Wert von ZAHL ja nicht aus dem aufgerufenen Script heraustransportieren ließe.

```
#
# Ausgabe des Eingangsparameters $1 in roemischen Ziffern
#
ZIFF ()
# Funktion zur Bearbeitung einer einstelligigen Ziffer $1
# Einer-, Zehner-, Hunderterstelle unterscheiden sich nur
# durch die verw. Zeichen $2: Einer, $3: Fuenfer, $4: Zehner
{ X=$1
  if test $X -eq 9; then
    ZAHL=${ZAHL}$2$4
  elif test $X -gt 4; then
    ZAHL=${ZAHL}$3
    while test $X -ge 6; do
      ZAHL=${ZAHL}$2 ; X=`expr $X - 1`
    done
  elif test $X -eq 4; then
    ZAHL=${ZAHL}$2$3
  else
    while test $X -gt 0; do
      ZAHL=${ZAHL}$2 ; X=`expr $X - 1`
    done
  }
```


Das Hauptprogramm prüft den Eingabeparameter, der die Turmhöhe angibt und ruft dann die Funktion auf:

```
#!/bin/bash

dohanoi()      # siehe oben
{ ... }

Moves=0        # Globale Variable fuer die Zahl der Zuege
if [ $# -eq 1 ]
then
  if [ "$1" -gt 0 ]
  then
    dohanoi $1 1 3 2
    echo "Total moves = $Moves"
    exit 0
  else
    echo "Parameter muss groesser 0 sein"
    exit 1
  fi
else
  echo "Parameter fehlt"
  exit 1
fi
```

Die Bash kann auch Zufallszahlen erzeugen. Da es sich aber nur um Pseudo-Zufallszahlen handelt, sollte man sie nicht für das Erzeugen eines kryptografischen Schlüssels verwenden. Die *Zufallsfunktion* verbirgt sich hinter einer *Variablen* namens \$RANDOM. Diese Pseudo-Variable liefert jedesmal einen neuen Zufallswert im Bereich zwischen 0 und 32767. Das folgende Beispiel liefert 10 Zufallswerte:

```
for I in 1 2 3 4 5 6 7 8 9 10
do
  number=$RANDOM
  echo $number
done
```

Will man den Bereich der Zufallszahlen beschränken, kann man folgendermaßen vorgehen. Das Beispiel liefert Werte zwischen 100 und 200:

```
MIN=100
MAX=200
number=0
while [ "$number" -le $MIN ]
do
  number=$RANDOM
  number=`expr "$number" % $MAX`
done
echo "Zufallszahl zwischen $MIN und $MAX: $number"
```

Das folgende Beispiel zeigt noch einen weiteren Shell-Trick: Verwendung von Arrays ohne echte Arrays. Es wird zufällig eine Spielkarte gezogen. Die Werte und Farben werden in Pseudo-Arrays gespeichert, die sich über mehrere Zeilen erstrecken (müssen). Deshalb geht folgendes auch wieder nur mit der Bash:

```
Farben="Karo
Herz
Kreuz
Pik"

Werte="2
3
```

```
4
5
6
7
8
9
10
Bube
Dame
Koenig
Ass"

Num_Farben=4
Num_Werte=13

Farbe=($Farben)          # Einlesen in Array-Variablen
Wert=($Werte)

echo -n "${Farbe[$((RANDOM%Num_Farben))]} "
echo    "${Wert[$((RANDOM%Num_Werte))]} "
```

Nun noch einige Berechnungen mit Zufallszahlen:

```
# Zufallszahlen zwischen 6 und 30.
number=$((RANDOM%25+6))

# Dito, aber die Zahl muss durch 3 teilbar sein
number=$((RANDOM%27/3*3+6))
```

Wird RANDOM ein Wert zugewiesen, setzt man damit einen Startwert für die Erzeugung der Zufallszahlen ("Seed"), nicht für den Zufallswert selbst.

```
RANDOM=1          # Setzen RANDOM-Seed
make_random_numbers

RANDOM=1          # Dieselbe Seed fuer RANDOM...
make_random_numbers # ...reproduziert genau dieselbe Zufallsfolge

RANDOM=2          # Eine andere Seed fuer RANDOM...
make_random_numbers # ...reproduziert eine andere Zufallsfolge
```

Es stellt sich die Frage, wie man RANDOM etwas "zufälliger" machen kann. Man kann nur versuchen, eine möglichst zufällige Seed zu wählen, indem man zum Beispiel die Prozessnummer (\$\$) oder `time` bzw. `date` verwendet. Eine bessere Quelle ist das Device `/dev/urandom`, das aber leider Binärzahlen liefert:

```
SEED=$(head -1 /dev/urandom | od -N 1 | awk '{ print $2 }')
# ein Wert von urandom | lesbar machen | Nummer extrahieren
RANDOM=$SEED
```

Statt `urandom` nur für die Seed zu verwenden, kann man natürlich gleich komplett auf das Device zurückgreifen - man erhält wesentlich "bessere" Zufallsfolgen. Nur muss man die Binärdaten immer noch in lesbare Zahlen umwandeln. Das folgende Kommando liefert beispielsweise eine Folge von XX gleichverteilten Zufallswerten als Datei (die dann mit `od` oder dergleichen gefiltert werden kann - wie oben gezeigt):

```
dd if=/dev/urandom of=zieldatei bs=1 count=XX
```

8.10 Fallen und Stolpersteine

Bei Linux sind 'bash' und 'sh' identisch, beidesmal wird die Bash aufgerufen. Bei anderen Unix-Varianten ist das keineswegs immer so. deshalb sollten Sie entweder das Skript per Shebang `#!/bin/bash` an die Bash binden oder (bei `#!/bin/sh`) nur die Befehle und Eigenschaften der Bourne-Shell nutzen.

Ein Script mit DOS/Windows_Zeilenenden (`\r\n`) wird sich nicht ausführen lassen, weil `#!/bin/bash\r\n` nicht erkannt wird, denn der Dateiname der Shell lautet hier ja "bash\r" und nicht "bash". Die Script-Datei muss also passend konvertiert werden.

Variablennamen

Es ist keine gute Idee, Befehle der Shell oder reservierte Worte als Variablennamen zu vergeben. In Namen sollten Nur Buchstaben, Ziffern und der Unterstrich auftauchen. Namen müssen mit einem Buchstaben beginnen (Namen, die mit einer Ziffer beginnen sind für die Shell reserviert). Folgende Beispiele funktionieren **nicht!**

```
case=Wert1           # Problem!
23foo=Wert2         # Problem!
_23foo=Wert3        # O. K.
_=Wert4; echo $_    # Problem: $_ liefert immer den letzten
                   # Parameter des letzten Kommandos
```

Jedoch ist '_' ein gültiger Funktionsname.

Es ist zulässig, für Funktionen und Variablen denselben Namen zu verwenden - das Script wird aber dadurch nahezu unlesbar:

```
do_something ()
{
  echo "Diese Funktion macht irgendwas mit \"$1\"."
}
do_something=do_something
do_something do_something
```

Wertzuweisung

Mit der beliebteste Anfängerfehler sind Leerzeichen bei der Wertzuweisung (weil ja sonst die Shell überall Leerzeichen braucht). Die Zeile

```
foo = 42
```

wird interpretiert als "Führe das Kommando 'foo' mit dem Parametern '=' und '42' aus. Richtig ist also:

```
foo=42
```

Das gilt natürlich auch für die Anwendung in Zusammenhang mit Backticks:

```
foo = `ls -l`      # Geht schief!
foo=`ls -l`       # so muss es sein!
```

Gerne wird auch das letzte Semikolon bei der Zusammenfassung mit geschweiften Klammern vergessen:

```
{ ls -l; df; echo "Done." }
# Die Bash gibt eine Fehlermeldung aus: "syntax error: unexpected end of file"
```

```
{ ls -l; df; echo "Done."; }  
# ^ das Semikolon muss sein!
```

Vergleichsoperatoren

Bei Vergleichen wird auch gerne '=' und '-eq' verwechselt. '=' vergleicht Zeichenketten ('==' gibt es übrigens auch nicht), '-eq' Zahlen. Bedauerlicherweise ist es bei Perl wieder genau andersrum.

```
a=" 4096"  
  
if [ "$a" = 4096 ]      # Stringvergleich liefert FALSCH  
if [ "$a" -eq 4096 ]   # Zahlenvergleich liefert WAHR
```

Quoting

Denken Sie auch daran, dass bei Vergleichen die Variablen innerhalb der eckigen Klammern sinnvollerweise in Gänsefüßchen stehen sollten, z. B.:

```
a=""                    # Leerstring  
  
if [ $a = "yes" ]      # Die Bash bricht mit Fehlermeldung ab, denn es  
                      # die Klammer ergibt [ = "yes" ]  
  
if [ "$a" = "yes" ]    # ergibt FALSCH; Ausführung läuft weiter
```

Vergessen Sie auch nicht, dass alles, was Leerzeichen oder irgendwelche Sonderzeichen enthält, am Besten in Gänsefüßchen eingeschlossen wird.

Subshells

Wie wir aus der Biologie wissen, können Eltern Eigenschaften an ein Kind vererben. Der umgekehrte Fall ist unmöglich. Bei Unix ist das genauso, wie folgendes Beispiel zeigt:

```
outer_variable="Ich bin draussen"  
export outer_variable  
echo "outer_variable = $outer_variable"    # Ausgabe wie erwartet  
  
(  
  # Begin subshell  
  echo "outer_variable inside subshell = $outer_variable"  
  inner_variable="Ich bin drin"           # Set  
  echo "inner_variable inside subshell = $inner_variable"  
  outer_variable="Ich bin drin"           # Wird der Wert global geaendert?  
  echo "outer_variable inside subshell = $outer_variable"  
  # End subshell  
)  
  
echo "inner_variable outside subshell = $inner_variable" # Unbekannt!  
echo "outer_variable outside subshell = $outer_variable" # Unveraendert!
```

Leitet man Daten per Pipe in ein read-Kommando, kann dies wie das Piping in eine Subshell behandelt werden - mit entsprechend unerwünschten Resultaten. Dazu ein Beispiel:

```
a=aaa ; b=bbb ; c=ccc  
  
# Versuch, den Variablen neue Werte zu geben:  
echo "one two three" | read a b c  
# Das ging leider schief:  
echo "a = $a"  # a = aaa  
echo "b = $b"  # b = bbb
```

Betriebssystem UNIX/Linux

```
echo "c = $c" # c = ccc
```

In diesem Fall könnte man alternativ das `set`_Kommando verwenden:

```
set -- one two three
a=$1; b=$2; c=$3
# diesmal klappt es
echo "a = $a" # a = one
echo "b = $b" # b = two
echo "c = $c" # c = three
```

Die einfachste Lösung ist es, ein Kommando die Daten in eine Datei schreiben zu lassen, die dann per Eingabeumleitung von `read` gelesen wird. Das folgende Beispiel von A. Richardson zeigt mögliche andere Lösungswege.

```
# Loop piping troubles.
# Example by Anthony Richardson, with addendum by Wilbert Berendsen.

foundone=false
find $HOME -type f -atime +30 -size 100k |
while true
do
    read f
    echo "$f is over 100KB and has not been accessed in over 30 days"
    echo "Consider moving the file to archives."
    foundone=true
    # -----
    echo "Subshell level = $BASH_SUBSHELL"
    # Subshell level = 1
    # Yes, we're inside a subshell.
    # -----
done

# foundone will always be false here since it is
# set to true inside a subshell
if [ $foundone = false ]
then
    echo "No files need archiving."
fi

# =====Now, here is the correct way:=====

foundone=false
for f in $(find $HOME -type f -atime +30 -size 100k) # No pipe here.
do
    echo "$f is over 100KB and has not been accessed in over 30 days"
    echo "Consider moving the file to archives."
    foundone=true
done

if [ $foundone = false ]
then
    echo "No files need archiving."
fi

# =====And here is another alternative=====

# Places the part of the script that reads the variables
# within a code block, so they share the same subshell.
# Thank you, W.B.

find $HOME -type f -atime +30 -size 100k | {
    foundone=false
    while read f
    do
```

```
    echo "$f is over 100KB and has not been accessed in over 30 days"
    echo "Consider moving the file to archives."
    foundone=true
done

if ! $foundone
then
    echo "No files need archiving."
fi
}
```

8.11 Weitere Beispiele

Die folgenden Beispiele stammen fast alle aus der täglichen Arbeit und helfen bei der Administration eines Systems. Bei einigen Beispielen wurden systemspezifische Teile des Originals weggelassen.

Eingabe ohne RETURN-Taste

Soll nur eine Taste zur Bestätigung gedrückt werden, z. B. 'j' oder 'n', läßt sich das mit dem `read`-Kommando nicht realisiert werden, da die Eingabe immer mit der Return-Taste abgeschlossen werden muß. Um eine Eingabe ohne Return zu realisieren sind zwei Dinge nötig:

- Schalten des Terminals auf ungepufferte Eingabe
- Einlesen der gewünschten Anzahl von Zeichen

Das Umschalten des Terminals wird mit `stty raw` erreicht. Für die Eingabe wird das `dd`-Kommando (Disk Dump) zweckentfremdet. `dd` liest von der Standardeingabe und schreibt auf die Standardausgabe. Für die geplante Aktion werden die Parameter `count` (Anzahl zu lesender Blöcke) und `bs` (Blocksize) verwendet. `count` enthält die Anzahl der einzulesenden Zeichen, `bs` wird auf 1 gesetzt. Der entstehende Programmteil sieht dann so aus:

```
echo "Alles Loeschen (j/n)\c"
stty raw -echo
INPUT=`dd count=1 bs=1 2> /dev/null`
stty -raw echo
echo $INPUT
case $INPUT in
  j|J) echo "Jawoll" ;;
  n|N) echo "Doch nicht" ;;
  *)   echo "Wat nu?" ;;
esac
```

Shell-Script zum Eintragen eines neuen Benutzers

Dieses Script soll nur zeigen, was beim Anlegen eines Benutzers alles notwendig ist. Normalerweise existieren bereits entsprechende Programme oder Scripten (z. B. `useradd` oder das Administrationstool `YaST`).

```
# Neuen Benutzer eintragen, Home-Directory erzeugen,
# .profile kopieren
PWDF=/etc/passwd
GRPF=/etc/group
STDPROF=.profile
if test $# -ne 3 ; then
    echo "Usage: newuser Login-Name Gruppe Voller Name" ; exit 1
fi
NAME=$1 ; GRUPPE=$2
HOME=/home/$1
case $NAME in
```

Betriebssystem UNIX/Linux

```
?????????) echo "Name ist zu lang" ; exit 1 ;;
*[A-Z]*)      echo "Name enthaelt Grossbuchstaben" ; exit 1 ;;
esac
if grep "^$NAME:" $PWDF ; then
    echo "Name schon vorhanden" ; exit 1
fi
UID=`tail -1 $PWDF | cut -f3 -d:`
UID=`expr $UID + 1`
if grep ".*:.*:$UID:" $PWDF ; then
    echo "Passwortdatei ist nicht sortiert" : exit 1
fi
if grep ".*:.*:$GRUPPE:" $GRPF ; then
    :
else
    echo "Gruppen-Nummer nicht vorhanden" ; exit 1
fi
mkdir $HOME
cp $STDPFROF $HOME
chown $UID $HOME $HOME/$STDPFROF
chgrp $GRUPPE $HOME $HOME/$STDPFROF
echo $NAME::$UID:$GRUPPE:$3:$HOME:/bin/sh >>$PWDF
echo $NAME::$UID:$GRUPPE:$3:$HOME:/bin/sh
```

Mit useradd geht das ganze viel einfacher, aber hier geht es ja um ein Beispiel. Das folgende Script vereinfacht die Anwendung von useradd, das relativ viele Parameter besitzt.

```
#!/bin/sh
# Shell-Script zum Anlegen eines Benutzers
# Aufruf: newuser username gruppe Bemerkung
#
if [ $# != 3 ] ; then
    echo "Usage: `basename $0` username gruppe Bemerkung"
    exit 2
fi
GRUP=$2
USR=$1
BEM=$3
GRP=/etc/group
PASS=/etc/passwd
SHAD=/etc/shadow
SKEL=/etc/skel
cp $PASS ${PASS}.bak
cp $SHAD ${SHAD}.bak
if [ "$USR" != "" ] ; then
    echo "--- Anlegen User: $USR, Bemerkung: $BEM ---"
    if `grep -s $GRUP $GRP >/dev/null` ; then
        :
    else
        echo "--- Gruppe $GRUP unbekannt ---"
        exit 2
    fi
    if `grep -s $USR $PASS` ; then
        echo "+++ User existiert bereits +++"
        exit 2
    fi
    /usr/sbin/useradd -d /home/${USR} -g $GRUP -s /bin/sh -c "$BEM" -m -k $SKEL $USR
    if [ $? -eq 0 ] ; then
        echo "+++ $USR angelegt +++"
    else
        echo "--- Fehler beim Anlegen von $USR ---"
    fi
    while [ -f /etc/ptmp ] ; do
        sleep 1
    done
fi
echo "--- Fertig ---"
```

Löschen von Usern en bloc

Zum Löschen der User einer Schulungsgruppe wird anstelle des "Realname" in der Datei /etc/passwd die Kursnummer eingetragen (z. B. "K1234"). Durch das folgende Script können durch Angabe eines Suchbegriffs die Teilnehmer eines Jahrgangs komplett als User gelöscht werden. Das eigentliche Löschen wird dabei von einem zweiten Script übernommen.

```
# rm-all - Löschen User nach Stichwort in der Passwort-Datei
if [ $# -ne 1 ]; then
    echo "Aufruf: $0 Suchbegriff"
    exit 1
fi
# Erst mal testweise die Kandidaten fürs Löschen ausgeben
KANDIDATEN=`grep "$1" /etc/passwd | sed -e '1,$s/:.*/'\`
echo "$KANDIDATEN loeschen (j/n)? \c"
read ANTWORT
if [ $ANTWORT != "j" ]; then
    echo "Abgebrochen!"
    exit 1
fi
# jetzt wird wirklich gelöscht
/bin/rmuser $KANDIDATEN
```

Löschen von Usern

Das Script "rmuser" ist etwas aufwendiger, da einige Sicherheitstest notwendig sind. Auf den meisten Systemen gibt es ein Programm "userdel", das den Benutzer aus /etc/passwd, /etc/group und /etc/shadow löscht. Bei anderen Anlagen könnte man das Gleiche mit einem Script erledigen. Das Löschprotokoll wird per Mail an root geschickt.

```
# rmuser - löschen user
if [ $# -lt 1 ]; then
    echo "Aufruf: $0 user [user ....]"
    exit 1
fi
{
# Sicherheitskopien anlegen
cp /etc/passwd /etc/passwd.bak
cp /etc/shadow /etc/shadow.bak

# Jetzt wird es ernst
while [ $# -gt 0 ] ; do
    USR=$1
    N=`grep -c "$USR" /etc/passwd`
    if [ $N -ne 1 ]; then
        echo "$USR nicht vorhanden oder doppelt"
    else
        if [ `grep $USR /etc/passwd | cut -d: -f3` -lt 100 ]; then
            echo "$USR hat eine ID kleiner 100, nicht geloescht"
        else
            echo "*** Loeschen User: $USR"
            # Homedir aus /etc/passwd extrahieren
            HOM=`grep $USR /etc/passwd | cut -f6 -d:`
            rm -rf $HOM 2>&1
            find / -user $USR -exec rm -f {} ";" 2>&1
            /usr/sbin/userdel $USR
            echo "--- $USR erledigt ..."
            echo ""
        fi
    fi
    shift
done
} | mailx -s "User-Loeschung" root
```

Man könnte das Script noch um eine Prüfung auf weitere nicht zu löschende User ergänzen. Außerdem sollte man das Ganze erst starten, wenn sonst kein User mehr im System ist. Da das Script bei vielen User recht lange dauert, ist es am günstigsten, es als Batch im Hintergrund zu starten.

Ständig kontrollieren, wer sich ein- und ausloggt

```
#!/bin/sh
# PATH=/bin:/usr/bin
NEW=/tmp/WW1.WHO
OLD=/tmp/WW2.WHO
>$OLD                # OLD neu anlegen
while :              # Endlosschleife
do
    who >$NEW
    diff $OLD $NEW
    mv $NEW $OLD
    sleep 60
done
```

Speicherbelegung

Das Script 'lsum' berechnet aus dem ls-Kommando den Gesamtspeicherplatz der ausgewählten Dateien. Einfacher geht es aber mit 'du'.

```
#!/bin/sh
# Calculate the amount of space used by the specified files
# Default is the actual directory
SUM=0
TMPF=$HOME/$0$$
ls -l $* >$TMPF
while read D1 D2 D3 D4 D5 REST ; do    # lesen aus TMPF
    # Feld 5 enthaelt Groesse
    SUM=`expr $SUM + 0$D5 / 1024`
done < $TMPF
echo "$SUM KBytes"
rm $TMPF
```

Preisfrage: Warum funktioniert folgende Variante nicht?

```
#!/bin/sh
SUM=0
ls -l | while read D1 D2 D3 D4 D5 REST ; do
    SUM=`expr $SUM + 0$D5 / 1024`
done
echo "$SUM KBytes"
```

Optionen ermitteln

Oft ist es wünschenswert, wenn man bei Shellscrippts auch Optionen angeben kann (auf dieselbe Weise, wie bei Programmen). Die Optionen bestehen aus einem Buchstaben mit dem '-' davor. Bei manchen Optionen folgt auch eine durch die Option spezifizierte Angabe (z. B. beim 'pr'-Kommando der Header). Das folgende Fragment zeigt, wie sich solche Optionen behandeln lassen. Die einzige Einschränkung besteht darin, daß sich mehrere Optionen nicht zusammenziehen lassen ('-abc' statt '-a -b -c' geht also nicht). Damit alle Optionen über eine Schleife abgehandelt werden können, wird mit 'shift' gearbeitet. Wie üblich, können nach den Optionen noch Dateinamen folgen. Ein Testaufruf könnte lauten:

```
otest -a -p "Parameter" -c *.txt

#!/bin/sh
```

Betriebssystem UNIX/Linux

```
# Bearbeiten von Optionen in Shellscripsts
# Beispiel: -a -b -c als einfache Optionen
#           -p <irgend ein Parameter> als "Spezial-Option"
READOPT=0
while [ $READOPT -eq 0 ] ; do      # solange Optionen vorhanden
  case $1 in
    -a) echo "Option a"
        shift ;;
    -b) echo "Option b"
        shift ;;
    -c) echo "Option c"
        shift ;;
    -p) PARAM=$2 ; shift # Parameter lesen
        echo "Option p: $PARAM"
        shift ;;
    *) if `echo $1 | grep -s '^-'` ; then # Parm. beginnt mit '-'
        echo "unknown option $1"
        shift
      else
        READOPT=1 # Ende Optionen, kein shift!
      fi ;;
  esac
done
echo "Restliche Parameter : $@"
```

'Rename'-Kommando

So mächtig das 'mv'-Kommando auch ist, es bietet keine Möglichkeit, eine ganze Reihe von Dateien nach gleichem Schema umzubenennen (z. B. 'kap???.txt' --> 'kapitel???.txt'). Das folgende Script leistet das gewünschte. Die ersten beiden Parameter enthalten den ursprünglichen Namensteil (z. B. 'kap') und den neuen Namensteil (z. B. 'kapitel'). Danach folgt die Angabe der zu bearbeitenden Dateien. Wenn die Zieldatei schon existiert, wird nicht umbenannt, sondern eine Fehlermeldung ausgegeben.

```
#!/bin/sh
# Alle Dateien umbenennen, die durch $3 - $n spezifiziert werden
# dabei wird der String $1 im Dateinamen durch $2 ersetzt,
# wobei auch reguläre Ausdrücke erlaubt sind
if [ $# -lt 3 ] ; then
  echo 'Usage: ren <old string> <new string> files'
  echo 'Example: ren foo bar *.foo renames all files'
  echo '              *.foo ---> *.bar'
  exit 1
fi
S1="$1" ; shift
S2="$1" ; shift
while [ $# -gt 0 ] ; do
  for OLDF in $1 ; do
    NEWF=`echo $OLDF | sed -e "s/${S1}/${S2}/"`
    if [ -f $NEWF ] ; then
      echo "$NEWF exists, $OLDF not renamed"
    else
      echo "renaming $OLDF to $NEWF"
      mv $OLDF $NEWF
    fi
  done
  shift
done
```

Löschen von Prozessen

Das Löschen eines Prozesses über die Prozeßnummer ist insofern lästig, als man die Nummer meist erst per 'ps'-Kommando ermitteln muß. Das folgende Script erlaubt die Angabe eines beliebigen Strings (z. B. User- oder Programmname) und löscht alle Prozesse, die diesen String in der Ausgabe

des 'ps'-Kommandos enthalten. Bei jedem Prozeß wird dann interaktiv gefragt, ob er gelöscht werden soll. Vor dem eigentlichen Löschen wird noch einmal nachgesehen, ob der Prozeß noch existiert, denn er könnte ja als Kindprozeß eines vorher gelöschten Prozesses schon gekillt worden sein. Kleiner Schönheitsfehler: Der zap-Prozess taucht auch in der Liste auf. Wie könnte man das beseitigen?

```
#!/bin/sh
# Loeschen Prozess durch Angabe eines Musters
TMPF=$HOME/zap..$$
if [ $# -lt 2 ]; then
    echo "Usage: zap -Signal Muster"; exit 1
fi
SIG=$1
# alle Prozesse nach Stichwort durchsuchen
ps aux | grep $2 | sed -e '1,$s/ */ /g' > $TMPF
while read X
do
    set $X
    echo "$X (j/n)? \c"
    read ANTWORT </dev/tty
    case $ANTWORT in
        j*|J*) X=`ps aux | grep -c $2`
                if [ $X -ne 0 ]; then
                    kill $$SIG $2
                fi ;;
        q*|Q*) break ;;
    esac
done <$TMPF
rm $TMPF
```

Backup

Das folgende Script zeigt, wie man den Backup eines Verzeichnisses automatisieren kann. Normalerweise wird ein incrementeller Backup erzeugt, d. h. es werden nur die Dateien gesichert, die nach dem letzten Backup geändert oder neu erstellt wurden. Will man einen vollen Backup, muß man den Parameter '-a' (für 'all') angeben. Um festzustellen, welche Dateien neu sind, wird im entsprechenden Verzeichnis eine leere Datei namens '.lastbackup' angelegt bzw. deren Zugriffsdatum aktualisiert. Nach deren Änderungsdatum richtet sich die Auswahl der zu sichernden Dateien. Beim allerersten Backup muß der Parameter '-a' angegeben werden. Die Angabe des Backup-Devices (/dev/tape) muß eventuell an die lokalen Gegebenheiten angepaßt werden. Die Angabe '-depth' beim find-Kommando sorgt dafür, daß die Dateien "von unten" her bearbeitet werden (nötig für das cpio-Kommando).

```
#!/bin/sh
# Taeglicher Backup, als Parameter wird ein
# Verzeichnis angegeben
if [ $# -eq 0 ] ; then
    echo "Aufruf: $0 [-a] <directory>"
    echo "-a Alles sichern (sonst incrementell)"
    exit
fi
echo "\nBand einlegen und [RETURN] druecken!"
read DUMMY
if [ "$1" = "-a" -o "$1" = "-A" ] ; then
    if [ -d "$2" ] ; then
        echo "Komplett-Backup von $2 ..."
        MARKER=$2/.lastbackup
        find $2 -depth -print | cpio -ovc >/dev/tape
        touch $MARKER
        echo "Fertig!"
    else
        echo "$2 ist kein Verzeichnis!"
        exit
    fi
fi
```

Betriebssystem UNIX/Linux

```
else
  if [ -d "$1" ] ; then
    echo "Inkrementeller Backup von $1 ..."
    MARKER=$1/.lastbackup
    find $1 -newer $MARKER -print | cpio -ovc >/dev/tape
    touch $MARKER
    echo "Fertig!"
  else
    echo "$2 ist kein Verzeichnis!"
    exit
  fi
fi
echo "\nBand herausnehmen\n"
```

Zwischen-Backup

Manchmal ist es neben dem "normalen" Backup auf Band auch günstig, einen weiteren Backup auf der Platte anzulegen - z. B. für den Fall, daß jemand irrtümlich eine Datei löscht. Die kann dann mit ein paar Tastenbetätigungen wieder hervorgezaubert werden. Das folgende Script sichert alle Dateien des WWW-Servers, die seit der letzten Sicherung hinzugekommen sind. Als Zeitmarkierung dient das Datei-Zugriffsdatum einer Datei namens ".lastcheck". Bei jedem Backup wird das Datum der Datei per touch-Befehl aktualisiert. Man sieht auch schön, wie sich das date-Kommando zum Erzeugen von einzigartigen Dateinamen verwenden läßt. Statt die Ausgabe zu unterdrücken, könnte man sie auch per Mail an den WWW-Admin schicken (dann sollte tar etwas auskunftsfreudiger gemacht werden).

```
#!/bin/sh
#
# Inkrementelles Sichern aller Dateien des WWW-Servers
#
{
  TMPFILE="/tmp/check.$$"
  TIMESTAMP="`date +%y%m%d`"
  DIRECTORY="/home/httpd/htdocs"
  WWWARCHIVE="/home/wwwarchive"
  cd $DIRECTORY
  find . -newer .lastcheck -print >$TMPFILE 2>/dev/null
  touch .lastcheck
  if [ `cat $TMPFILE | wc -l` -gt 0 ]
  then
    tar cf /$WWWARCHIVE/backup.$TIMESTAMP.tar $DIRECTORY
    gzip /$WWWARCHIVE/backup.$TIMESTAMP.tar
    chown wwwadm.staff /$WWWARCHIVE/backup.$TIMESTAMP.tar.gz
    chmod 660 /$WWWARCHIVE/backup.$TIMESTAMP.tar.gz
  fi
  rm $TMPFILE
} > /dev/null 2>&1
```

eval-Anwendung

Endlich eine Anwendung für das eval-Kommando, auf die jeder schon gewartet hat. Das folgende Fragment zeigt, wie man in der Bourne-Shell die Ausgabe des aktuellen Verzeichnisses im Prompt realisieren kann. Einziger Nachteil: Zum Logoff muß man Ctrl-C und Ctrl-D drücken.

```
while true ; do
  echo "`pwd`: $PS1\c"
  read KDO
  eval $KDO
done
```

Ausgaben aus cron- und at-Jobs auf ein Terminal

Wie schicke ich aus einem cron- oder at-Job etwas an den Benutzer, sofern er eingeloggt ist? Das Problem liegt darin, daß der Job nicht wissen kann, an welchem Terminal der User sitzt. Also muß zunächst per 'who'-Kommando festgestellt werden, ob der Adressat eingeloggt ist und an welchem Terminal er sitzt. Dann kann eine Nachricht nach folgendem Schema an den User geschickt werden.

```
#!/bin/sh
# Nachricht ($2-$nn) an User ($1) senden, sofern dieser eingeloggt ist
NAM="$1"
shift
MSG="$@"
if who | grep -q $NAM ; then      # User eingeloggt?
    write $NAM < $MSG
fi
```

Rundruf

Nach dem gleichen Schema kann man ein "wall für Arme" realisieren. Es werden jedoch im Gegensatz zum "echten" 'wall' nur die Benutzer erreicht, die Ihren Mitteilungsempfang offen haben.

```
who | while read USR REST ; do    # für alle aktiven User
    banner "Teatime!" | write $USR
done
```

Das "bang"-Script

Manche Kommandos möchten einen Dateinamen als Argument und nicht die Daten über die Standardeingabe haben. Für solche Kommandos muß man öfter eine temporäre Datei anlegen, die nach dem Kommandoaufruf wieder gelöscht wird. Ein typisches Beispiel dafür ist ein Vergleich zweier Dateien. Das Kommando "comm" benötigt die Namen der beiden Dateien. Wenn die Dateien vor dem Vergleich sortiert werden müssen, sieht der konventionelle Ansatz folgendermaßen aus:

```
sort file1 >/tmp/file1.sor
sort file2 >/tmp/file2.sor
comm /tmp/file1.sor /tmp/file2.sor
rm tmp/file[12].sor
```

Das folgende Script namens "!" führt ein Kommando aus, das als Parameter übergeben wird, und liefert den Namen einer temporären Datei zurück, in dem das Ergebnis der Kommandoausführung gespeichert wurde. Das wäre aber noch kein Fortschritt. Der Trick ist, daß die temporäre Datei nach fünf Minuten automatisch gelöscht wird. Unsere Aufgabe läßt sich dann als Einzeiler schreiben:

```
comm `! sort file1` `! sort file2`
```

Die Schwierigkeit beim Schreiben von "!" liegt darin, daß die aufrufende Shell ("comm"-Kommando) normalerweise wartet, bis das aufgerufene Script ("! sort ...") terminiert - aber dieses soll ja fünf Minuten warten und dann die Datei löschen. In diesem Fall wäre aber die temporäre Datei schon wieder leer. Die Lösung zeigt die Auflistung von "!":

```
#!/bin/sh
# Kommando ausführen,
# Ergebnis in temp. Datei,
# Dateiname zurückgeben
TEMPDIR=/tmp
TEMPF=$TEMPDIR/BANG..$$
# Trap zum Löschen von TEMPF
trap 'rm -f $TEMPF; exit' 1 2 15
# Falls kein Kommando, nur Dateinamen liefern
```

Betriebssystem UNIX/Linux

```
if [ $# -eq 0 ] ; then
    echo "Usage: `basename $0` command [args]" 1>&2
    echo $TEMPF
    exit 1
fi
# Kommando ausführen, Dateiname liefern
"$@" > $TEMPF
echo $TEMPF
# jetzt kommt der Trick:
exec >&- # Standardausgabe schließen, rufende Shell wartet nicht!
( sleep 300 ; rm -f $TEMPF ) & # Löschauftrag --> Hintergrund
exit 0
```

Rekursives Suchen in Dateien

```
#!/bin/sh
# Rekursives Script zum Suchen und Ersetzen von Text-Pattern
#
PROGNAME=`basename $0`
TEMPDAT=/tmp/`basename $0.$$`

if test $# -lt 4; then
    echo "$PROGNAME : Recursive search-and-replace-Script."
    echo "usage : $PROGNAME <start-dir> <file-expression> \
<search-pattern> <replace-pattern>"
    echo "example : $PROGNAME . \"*.html\" \"abc\" \"xxx\" "
    echo "Both patterns use ex/vi-syntax !"
else
    find $1 -type f -name "$2" -print > $TEMPDAT
    for NAME in `cat $TEMPDAT`
    do
        echo -n "Processing $NAME.."
        ex $NAME << EOT > /dev/null
1, \s /s3/s4/g
wq
EOT
        echo "done."
    done
    rm $TEMPDAT
fi
```

Datei mit Verzeichnisinfo anlegen

Bei FTP-Servern ist es üblich, eine Datei mit einem Verzeichnislisting in den einzelnen Download-Verzeichnissen anzulegen. Interessenten können sich erst einmal diese Datei holen und in Ruhe durchsehen. Das folgende Script erzeugt zwei Varianten: 'ls-lR' mit der Verzeichnisinfo und dieselbe Datei gepackt ('ls-lR.Z').

```
#!/bin/sh
# Generieren ls-lR und ls-RL.Z
# Eingabeparameter: Ausgangsverzeichnis
#
if [ $# -ne 1 ]; then
    echo "Aufruf: `basename $0` Verzeichnis"
    exit 1
fi
ROOT=$1
LSFILE=$ROOT/ls-lR
TMPFILE=/tmp/mkls.$$
cd $ROOT
echo "$ROOT" > $TMPFILE
# Verzeichnisinfo erstellen, Leerzeilen und Summenangabe raus
ls -lR 2>/dev/null | grep -v "^total" | grep -v "^$" >> $TMPFILE
cp $TMPFILE $LSFILE
```

```
cat $LSFILE | compress -f > $TMPFILE
cp $TMPFILE ${LSFILE}.Z
rm $TMPFILE
```

Ständig wachsende Dateien kürzen

Viele Logfiles wachsen notorisch. Wenn man nicht achtgibt, ist irgendwann die Platte voll. Über ein per crontab regelmäßig aufgerufenen Script kann man die Dateien klein halten. Nebenbei kann man auch gleich die spezielle Logdatei wtmp mit bearbeiten. Die Datei '/etc/prune_list' enthält die zu überwachenden Dateien in der Form 'dateiname länge'. Für jede Datei existiert eine Zeile, wobei der Dateiname mit vollem Pfad angegeben werden muß.

Schließlich löscht das Script noch alle Dateien aus dem /tmp-Verzeichnis, auf die seit 8 Tagen nicht mehr zugegriffen wurde und auf der gesamten Platte alle Dateien mit den Namen 'a.out', 'core' und '*.*' die 8 Tage alt oder älter sind.

```
#!/bin/sh
#
# prune: Shorten textfiles listetd in $FLIST.
# files are shortened to a certain number of lines at
# their end. In $FLIST are lines containing filename
# (full path) and number of remaining lines. E. g.:
#   /var/adm/messages      500
#   /var/adm/debug         100
#
# /var/adm/wtmp will also be shortened
# a.out, core, *.* and tmp-files will be deleted after 8 days
#
HOSTNAME=/bin/hostname      # Pfad hostname-Programm
FLIST=/etc/prune_list       # Liste zu loeschender Dateien
TMPF=/tmp/prune.$$          # Temporaerdatei
{
  while read FILE LEN
  do
    if [ -n $FILE -a -n $LEN ] ; then
      if [ `wc -l $FILE` -lt $LEN ] ; then
        echo "prune: ${FILE} nothing to do"
      else
        tail -$LEN $FILE >$TMPF
        cp $TMPF $FILE
        echo "prune: ${FILE} shortened to $LEN lines"
      fi
    else
      echo "prune: error in $FLIST, FILE or LEN missing"
    fi
  done &lt; $FLIST
  rm $TMPF
  cd /var/adm
  [ -f wtmp.3 ] && cp wtmp.3 wtmp.4
  [ -f wtmp.2 ] && cp wtmp.2 wtmp.3
  [ -f wtmp.1 ] && cp wtmp.1 wtmp.2
  cp wtmp wtmp.1
  cp /dev/null wtmp
  chmod 664 wtmp
  echo "wtmp shortened"
  [ -f wtmpx.3 ] && cp wtmpx.3 wtmpx.4
  [ -f wtmpx.2 ] && cp wtmpx.2 wtmpx.3
  [ -f wtmpx.1 ] && cp wtmpx.1 wtmpx.2
  cp wtmpx wtmpx.1
  cp /dev/null wtmpx
  chmod 664 wtmpx
  echo "wtmpx shortened"
# clean up /tmp and /usr/tmp
/usr/bin/find /tmp -type f -atime +7 -exec /bin/rm -f {} \;
/usr/bin/find /var/tmp -type f -atime +7 -exec /bin/rm -f {} \;
```

Betriebssystem UNIX/Linux

```
/usr/bin/find /usr/tmp -type f -atime +7 -exec /bin/rm -f {} \;  
/usr/bin/find / \(-name a.out -name core -name '*.o'\) -atime +7 \  
-exec /bin/rm -f {} \;  
  
} | mailx -s "Output from PRUNE ` $HOSTNAME`" root 2>&1
```

Kleiner Sicherheitscheck

Im Lauf der Zeit "verlottert" jedes System ein wenig. Es gibt noch Dateien von längst gelöschten Usern, manche User haben kein Passwort und dergleichen mehr. Ein paar Sicherheitschecks macht das folgende Script, das man nach den eigenen Bedürfnissen erweitern kann.

```
#!/bin/sh  
# Programm to run weekly to check some important items  
# must be run by root  
#  
# find accounts without password  
echo ""  
echo "Accounts without password"  
echo "-----"  
/usr/bin/grep '^[^:]*:.' /etc/passwd  
  
# find accounts with UID 0 and/or GID 0  
echo ""  
echo "Accounts with ID 0"  
echo "-----"  
/usr/bin/grep ':00*:' /etc/passwd  
  
# Check Permissions  
echo ""  
echo "Permissions of important files"  
echo "-----"  
ls -l /etc/passwd /etc/group /etc/hosts /etc/host.equiv /etc/inetd.conf  
echo ""  
  
echo "SUID-files"  
echo "-----"  
/usr/bin/find / -perm -4000 -type f -exec ls -l {} \;  
echo ""  
  
echo "SGID-files"  
echo "-----"  
/usr/bin/find / -perm -2000 -type f -exec ls -l {} \;  
echo ""  
  
echo "World-writable files"  
echo "-----"  
/usr/bin/find / -perm -2 \(-type f -o -type d\) -exec ls -l {} \;  
echo ""  
  
echo "Files without owner"  
echo "-----"  
/usr/bin/find / -nouser -exec ls -l {} \;  
echo ""  
  
echo "/var/adm/sulog:"  
echo "-----"  
cat /var/adm/sulog
```

Pack-Automat

Meine Seminarunterlagen auf dem Webserver ändern sich recht oft. Weil ich immer vergesse, auch eine gepackte Version des gesamten Verzeichnisses mit alle Bildern und HTML-Dateien zu erzeugen, läuft per cron-Auftrag jede Nacht der Automatik-Packer. Als Merker dienen zwei Dateien,

Betriebssystem UNIX/Linux

.lastpack, die sich im übergeordneten Verzeichnis befindet und angibt, wann zuletzt gepackt wurde. Gibt es im Verzeichnisbaum darunter keine Dateien, die neuer sind als .lastpack, wird nichts gemacht. In jedem darunterliegenden Verzeichnis gibt es eine Datei .lastupd, die es erlaubt, für das jeweilige Verzeichnis festzustellen, ob neue Dateien vorliegen. Ist das der Fall, wird das Packen gestartet. Das Ergebnis der Gesamtoperation wird per E-Mail an den Webmaster geschickt:

```
#!/bin/sh
#
# Erzeugt im WWW-Verzeichnis der Scripten
# jeweils gepackte Versionen.
# Die Dateien heissen <Verzeichnisname>.tar.gz
#
cd /home/httpd/lbs/Scripten
[ `find . -newer ".lastpack" -print | wc -l` -eq 0 ] && exit
{
ls > tmp.$$
while read DIR
do
if [ -d $DIR ] ; then
NAME=`basename $DIR`
cd $NAME
if [ -f .lastupd ] ; then
if [ `find . -newer ".lastupd" -print | wc -l` -gt 0 ] ; then
rm $NAME.tar.gz
/root/bin/upd-index-html
tar cf $NAME.tar *
gzip -f $NAME.tar
mv $NAME.tar.gz $NAME.tar.gz
touch .lastupd
echo "$NAME ... DONE!"
else
echo "$NAME ... nothing to do"
fi
else
touch .lastupd
fi
cd ..
fi
done < tmp.$$
rm tmp.$$
touch /home/httpd/lbs/Scripten/.lastpack
} | mailx -s "Scripten-Packer" webmaster
```

Dateinamen in Kleinbuchstaben wandeln

Auch folgendes kommt häufig vor. Es werden Webseiten unter Windows erstellt und nach dem Herunterladen werden diverse Dateien nicht gefunden, weil Ihr Name groß geschrieben wurde, die Hyperlinks aber nicht. Unter W. merkt man das nicht, alles funktioniert dort wunderbar. Das folgende Script wandelt alle Dateinamen in Kleinbuchstaben um, achtet aber darauf, dass keine vorhandenen Dateien überschrieben werden. Die Anzahl der bearbeiteten Dateien wird am Bildschirm ausgegeben. Die Anzahl der Misserfolge wird als Rückgabewert geliefert - für eine automatische Weiterbearbeitung.

```
#!/bin/sh
# tolower - rename files to all lower case

if [ $# -lt 1 ] # ein Parameter (Dateiname) ist notwendig
then
echo "rename files to all lower case"
echo "usage: tolower file [...]"
exit 1
fi
```

Betriebssystem UNIX/Linux

```
Errs=0      # Fehlerzaehler
n=0         # Dateizaehler
for i       # Alle Dateien der Parameterliste bearbeiten
do
  Lower=`echo "$i" | tr '[A-Z]' '[a-z]`
  if [ "$i" = "$Lower" ]
  then
    continue # name ist bereits in Kleinbuchstaben
  elif [ -r "$Lower" -o -w "$Lower" -o -x "$Lower" ]
  then
    # Zieldatei gibt es schon
    echo "could not rename $i: $Lower exists already"
    continue
  fi

  if mv "$i" "$Lower" # umbenennen erfolgreich?
  then
    n=`expr $n + 1`
  else
    Errs=`expr $Errs + 1` # Sonst Fehlerzaehler erhoehen
    echo "could not rename $i to $Lower"
  fi
done
echo "$n files renamed"
exit $Errs
```

Passwort ändern

Das Programm *passwd* ist so programmiert, daß nur interaktive Eingaben erlaubt sind, eine Umleitung ist nicht möglich. Mit einem weiteren, recht mächtigen Tool kann auch das per Script gesteuert werden. Das Tool heißt "expect". Man kann mit ihm interaktive Dialoge aller Art nachbilden. Es wird im Normalfall in einer Zeile des expect-Scripts ein über die Standardeingabe erwarteter String (expect ...) oder die Reaktion darauf (send ...) angegeben. So lassen sich beliebige Dialoge automatisieren. Eine Einführung in dies Tool kann an dieser Stelle nicht erfolgen, jedoch eine typische Anwendung. Das folgende Script setzt ein neues Passwort für einen User, es wird mit beiden Angaben als Argumente aufgerufen, z.B. "autopass meier Geheim". Es funktioniert nur für den User root:

```
#!/usr/bin/expect --
set passw [lindex $argv 1]
spawn /usr/bin/passwd [lindex $argv 0]
expect "password:"
send "$passw\r"
expect "password:"
send "$passw\r"
expect eof
```

Mit "spawn" startet das Script das passwd-Programm ("passwd meier"). Danach wird auf das "Enter ... password:" gewartet und mit der Eingabe des Passworts beantwortet. Das gleiche geschieht mit der Nachfrage ("Reenter ...").



[Zum Inhaltsverzeichnis](#)



[Zum nächsten Abschnitt](#)

Copyright © FH München, FB 04, Prof. Jürgen Plate

Letzte Aktualisierung: 14. Apr 2012



Vorlesung "UNIX"

von Prof. Jürgen Plate

9 Die Skriptsprache awk

9.1 Überblick

awk ist eine interpretierende Programmiersprache, die für das Manipulieren und Durchsuchen von Textdateien konzipiert wurde. Der Name awk steht dabei nicht etwa für eine bestimmte Aufgabe, Funktion oder Eigenschaft sondern setzt sich einfach aus den Anfangsbuchstaben der Nachnamen der Erfinder dieser Sprache, **A**ho, **K**ernighan und **W**einberger, zusammen.

Und damit es da gar kein Mißverständnis gibt: Sprechen Sie awk nicht wie das englische "awkward" oder "awful" aus. Sagen Sie einfach "**a w k**"!

Unter Linux wird die GNU Implementierung gawk benutzt, die außer allen im POSIX-Standard vorgesehenen Features auch die Erweiterungen aus SVR4 unterstützt.

Sie sollen zunächst einen Überblick der Sprache erhalten. In den folgenden Abschnitten werden dann die einzelnen Sprachkonstrukte etwas detaillierter behandelt.

Die Sprache awk ist recht einfach. Man kann Sie ohne weiteres aus der ziemlich umfangreichen Man-Page erlernen. Da es sich um einen Interpreter handelt, sind awk-Programme nicht besonders schnell. Man sollte also keine zu großen und komplexen Anwendungen in awk formulieren. Für Prototypen und kleine Toos ist awk aber bestens geeignet.

9.2 Aufruf und Optionen

Im einfachsten Fall ist das auszuführende Skript so kurz, daß man es gleich mit auf der Kommandozeile angeben kann:

```
awk [Optionen] [--] Programmtext Datei ...
```

Folgendes Beispiel gibt Namen der Benutzer und das jeweilige Home-Verzeichnis aus:

```
awk -F: '{print $5, $6}' /etc/passwd
```

Dabei gibt die Option

-F:

an, daß der Doppelpunkt als Trennzeichen zwischen Feldern aufgefaßt werden soll. Das eigentliche awk-Programm besteht aus

```
{print $1, $6}
```

Das eigentliche awk-Programm muss vor der Auswertung durch die Shell geschützt werden, deshalb die Einbettung in einfache Hochkommata. Die geschweiften Klammern gehören zum Programm und sind auch wichtig, wie wir später noch sehen werden.

Im awk werden Anweisungen, ähnlich der Shell, durch Semikolon getrennt. Damit kann man auch mehrere Anweisungen auf der Kommandozeile unterbringen. Trotzdem ist die Grenze der Übersichtlichkeit schnell erreicht und man schreibt das awk-Skript lieber in eine Datei. Der Programmaufruf sieht dann so aus:

```
awk [Optionen] -f Programmdatei [--] Datei ...
```

Üblicherweise wird das Ergebnis eines awk-Aufrufs auf die Standardausgabe (was meistens der Bildschirm ist) geschrieben. Ist das nicht gewünscht, so muß die Ausgabe auf eine Ergebnisdatei umgeleitet oder in ein weiteres Programm gepipt werden.

Zur Demonstration wird im folgenden Programm das erste Feld der Passwortdatei ausgegeben (der Feldseparator ist der Doppelpunkt), wobei die Zeilen zusätzlich nummeriert werden:

```
awk -F : '{print NR,$1}' /etc/passwd

1 root
2 bin
3 daemon
4 lp
5 news
...
```

9.3 Grundzüge der Sprache

9.3.1 Programmablauf

Im Prinzip sieht jedes awk-Programm folgendermaßen aus:

```
Bedingung1 { Aktion1 }
Bedingung2 { Aktion2 }
Bedingung3 { Aktion3 }
...
Bedingungn { Aktionn }
```

Als Bedingung wird meist ein Suchmuster (pattern) oder regulärer Ausdruck angegeben.

Aktionen sind immer in geschweifte Klammern eingeschlossen, wobei die öffnende Klammer immer in der gleichen Zeile stehen muß wie das Pattern. Mehrere Anweisungen werden durch Strichpunkt getrennt. Die Aktion wird in einer C-ähnlichen Sprache formuliert. Der Aktionsteil darf sich auch über mehrere Zeilen erstrecken. Im übrigen sind awk-Programme frei formatierbar (unter Berücksichtigung der obigen Einschränkungen).

In einer Programmzeile kann entweder die Bedingung oder der Aktionsteil weggelassen werden. Fehlt die Bedingung, wird die Aktion immer ausgeführt. Fehlt die Aktion, wird die Zeile komplett ausgegeben.

- awk liest zunächst das komplette Programm ein und überführt es in eine interne Darstellung, die sich schneller ausführen läßt.
Dann wird zuerst und einmalig ein eventuell vorhandener **BEGIN-Block** abgearbeitet. Dieser Block umfaßt die Anweisungen, die mit der speziellen Bedingung **BEGIN** angegeben wurden. Hier kann man z. B. Variable initialisieren.

Die Kommandozeilenoption im letzten Beispiel hätte man umgehen können, indem vor Ausführung der Kommandos der interne Feldseparator ("FS") auf den Doppelpunkt gesetzt worden wäre. Eine solche "Initialisierung" ist die typische Aufgabe des BEGIN-Musters:

```
awk 'BEGIN {FS = ":"} {print NR,$1}' /etc/passwd
1 root
2 bin
3 daemon
4 lp
```

Betriebssystem UNIX/Linux

```
5 news
...
```

- Wird ein awk-Programm ausgeführt, so läuft eine - im Programm nicht aufgeführte - äußere Schleife ab, die sequentiell alle Zeilen der zu verarbeitenden Eingabedatei liest und dabei jede Eingabezeile der Reihe nach mit allen aufgeführten Bedingungen vergleicht. Ist eine Bedingung erfüllt, so wird die zugehörige Aktion ausgeführt.

awk zerlegt eine Eingabezeile in Abhängigkeit vom Wert des Feldtrennzeichens FS (eine der awk-Standard-Variablen; siehe dazu 9.3.3) in einzelne Felder, die mit $\$1$, $\$2$, ..., $\$(NF-1)$ bezeichnet werden. NF ist eine andere awk-Standard-Variable, in der die Anzahl der Felder der Eingabezeile gespeichert wird. Auf die gesamte Eingabezeile kann mit $\$0$ zugegriffen werden.

- Wenn alle Eingabezeilen verarbeitet sind, werden einmalig die Anweisungen im **END-Block** abgearbeitet. Der END-Block ist durch das spezielle Suchmuster END gekennzeichnet. Hier werden häufig Summen ausgegeben.

Außerdem gibt es die Möglichkeit, Funktionen zu definieren:

```
function Name(Parameterliste) { Anweisungen }
```

$\$1$, $\$2$, $\$3$ usw. sind Zeichenketten und können dementsprechend im Aktionsteil manipuliert und verwendet werden.

Mit diesen Vorkenntnissen sollte die Funktionsweise des ersten Programms leicht nachvollziehbar sein; es zählt einfach nur die Zeilen der Eingabe und gibt das Resultat aus:

```
BEGIN {
  print "Zählen von Eingabezeilen";
  zaehler = 0
}

{ zaehler++ }

END { print "Ergebnis: " zaehler }
```

Ein weiteres einfaches Beispiel: Gegeben sei eine Datei, die Zeilen der Art

```
...
Emil Hofer 23.08.1960 1234 175 cm
Karl Müller 29.02.1957 1236 160 cm
...
```

mit den Angaben "Vorname", "Nachname", "Geburtsdatum", "Nebenstellen-Telefonnummer" und "Gewicht" enthält. Es soll eine Ausgabedatei generiert werden, die nur noch Geburtsdatum und Vorname (in dieser Reihenfolge) enthält. awk kennt folgenden AUSgabefunktionen:

- `print` gibt $\$0$ auf `stdout` aus (entspricht `print \$0`).
- `print` *ausdruck1*, *ausdruck2*, ... gibt die Ausdrücke in der angegebenen Reihenfolge durch OFS getrennt auf `stdout` aus. Abschließend wird ORS ausgegeben. `print` *ausdruck1* *ausdruck2* ... (ohne Komma!) bewirkt Ausgabe ohne Trennzeichen.
- `printf(format, ausdruck1, ausdruck2,...)` gibt die Ausdrücke gemäß dem Formatstring aus (siehe unten).

Damit kann das erste awk-Programm formuliert werden, was das Gewünschte leistet:

```
{print $3, $1}
```

Da ein Bedingungsteil nicht vorhanden ist, wird die aufgeführte Aktion unabhängig von Bedingungen für alle Eingabezeilen ausgeführt: Mittels der awk-Standard-Funktion `print` wird der Inhalt der Felder `$3` und `$1` in einem Standard-Format ausgegeben.

Das folgende Beispiel druckt eine etwas schöner formatierte Liste aus der `/etc/passwd`, wobei die C-ähnliche awk-Standard-Funktion `printf` verwendet wird:

```
BEGIN {
  FS=":";
  print "-----";
  printf "%-15s %-30s %s\n", "user-id", "Name", "Home";
  print "-----";
}
{printf "%-15s %-30s %s\n", $1, $5, $6}
```

In obigen Beispielen ist die Aktion von keiner Bedingung abhängig. Als Bedingung können jedoch ohne weiteres Vergleiche und logische Ausdrücke angegeben werden, wie im folgenden Beispiel gezeigt wird:

```
$4>170 && $6<=80 {print $0}
```

Angewendet auf die obige Eingabedatei bedeutet das, daß nur noch diejenigen Eingabezeilen (`$0`) aufgelistet werden, für die Größe `>170` (`$4>170`) **und** (`&&`) Gewicht `<=80` (`$6<=80`) gilt.

Bedingungen (Muster, Patterns)

Als häufigste Bedingung werden in awk wohl die von anderen UNIX-Programmen her bekannten Zeichenkettenmuster in Form regulärer Ausdrücke verwendet. Dazu zwei Beispiele:

```
/M[ea][iy]e?r/ {print $2, $3}
```

Als Bedingung ist hier ein regulärer Ausdruck aufgeführt, mit dem alle Eingabezeilen erkannt werden, die den Namen "Meier" (mit möglichen Varianten wie "Mayer", "Maier", "Meyr", "Meir", "Mayr" usw.) enthalten (Das Fragezeichen im regulären Ausdruck deutet an, daß das unmittelbar davor aufgeführte Element auch entfallen kann). Für die betreffenden Zeilen wird dann der Inhalt der Felder `$2` und `$3` ausgegeben.

```
$5 !~ /^[0-9][0-9][0-9]$/ {print $1, $2, $5}
```

Mit Hilfe der hier verwendeten Bedingung kann überprüft werden, ob das Eingabefeld `$5` (Körpergröße der Datei aus dem ersten Beispiel) einer Eingabezeile eine für ein bestimmtes Problem korrekt gebildete Zahl enthält. Genauer gesagt, es wird getestet, ob ein angegebenes Muster **nicht** (!~) auf das Eingabefeld `$5` "paßt".

Ist das der Fall, werden die Felder `$1`, `$2` und `$5` der betreffenden Zeile ausgegeben. Das Muster selbst beschreibt eine Zeichenkette, die aus drei Ziffern besteht.

In den bisherigen Programmbeispielen wurden die awk-Standard-Funktionen `print` und `printf` verwendet. Eine fast ebenso häufig eingesetzte Funktion ist `gsub`, mit der Zeichenketten in einer Zeile gegen andere ausgetauscht werden können. Dazu ein Beispiel:

```
/ä/ {gsub(/ä/, "&auml;")}
/Ä/ {gsub(/Ä/, "&Auml;")}
/ö/ {gsub(/ö/, "&ouml;")}
/Ö/ {gsub(/Ö/, "&Ouml;")}
/ü/ {gsub(/ü/, "&uuml;")}
/Ü/ {gsub(/Ü/, "&Uuml;")}
/ß/ {gsub(/ß/, "&szlig;")}
{print}
```

Mit Hilfe der in den Bedingungen angegebenen Muster für die deutschen Umlaute und das scharfe S und der awk-Funktion `gsub` werden in den betreffenden Zeilen diese Zeichen gegen die entsprechenden HTML-Umschreibungen ausgetauscht. Die letzte Programmzeile ist notwendig, damit das Ergebnis der vorgenommenen Umsetzungen auch die Standardausgabe ausgegeben wird.

Das folgende Beispiel zeigt ein Mini-Programm in der Kommandozeile:

```
awk ' {printf("%s-%4d %s\n", FILENAME, FNR, $0) }' eingabe*
```

Es werden die Zeilen aller Dateien, deren Name mit "eingabe" beginnt, numeriert und mit dem Dateinamen auf der Standardausgabe aufgelistet.

Umbenennen von Dateien (append ".new" to "files_list"):

```
ls files_list | awk '{print "mv \"$1\" \"$1\".new"}' | sh
```

9.3.2 Konstante, Variable, Sätze und Felder

awk kennt keine echten Datentypen, es wird nur mit Strings gearbeitet. Diese werden allerdings als Zahlen aufgefaßt, wenn das im Zusammenhang sinnvoll ist.

- Soll die Auswertung als Zahl erzwungen werden, so addiert man eine 0 (`string + 0`).
- Soll die Auswertung als String erzwungen werden, so hängt man einen Leerstring an (`zahl ""`).

Konstante

Zwei Arten von Konstanten können auftreten:

- String-Konstante
Eine Zeichenkette in Gänsefüßchen eingeschlossen. Im Gegensatz zur Shell findet keine Interpretation statt. So können auch die üblichen Escape-Sequenzen benutzt werden.
- numerische Konstante
Hierunter versteht man in diesem Zusammenhang alle ganzen Zahlen und Gleitkommazahlen wie 3.14 oder 1.6021892e-19. Intern werden alle Zahlen als Gleitkommazahlen gespeichert. Die Genauigkeit ist natürlich maschinenabhängig.

Variable

awk verwendet dynamische Variable. Diese brauchen nicht deklariert zu werden und können Strings oder Gleitkommawerte enthalten. Variablen können sowohl Strings, als auch numerische Werte beinhalten. Es muß ihnen kein expliziter Datentyp zugeordnet werden. Während eines Programmablaufs kann dieselbe Variable sowohl Strings, als auch numerische Werte speichern. Der Typ einer Variablen wird immer aus dem Kontext bestimmt. Hierbei werden ggf. Umwandlungen gemäß **OFMT** vorgenommen. Jede Variable wird mit einem Nullstring bzw. mit Null initialisiert. In Ausdrücken können 3 Arten von Variablen verwendet werden:

- vom Benutzer gewählte Variablen
Variablen können aus Buchstaben, Ziffern und Unterstrich bestehen. Das erste Zeichen muß ein Buchstabe oder der Unterstrich sein!
- Feld-Variablen
Feld-Variablen setzen sich aus \$ und der Feldnummer zusammen. In \$0 wird aktuelle Eingabezeile (Rekord) abgelegt. \$0 ist oft voreingestelltes Argument bei Funktionen, beispielsweise bei `print`. Der aktuelle Record wird gemäß FS (Feldtrennzeichen) in die

Felder $\$1 \dots \NF zerlegt (NF - Anzahl der Felder des Satzes).

Feld-Variablen können genau so wie andere Variablen behandelt werden, insbesondere können ihnen auch neue Werte zugewiesen werden. Diese Zuweisungen beeinflussen $\$0$ und NF !

Feld-Variablen können auch unter Verwendung von Ausdrücken angegeben werden: $\$(NF-1)$ ist beispielsweise das vorletzte Feld der Eingabezeile.

- **built-in Variablen**

Built-in Variablen sind Variablen die awk zur Verfügung stellt und mit Werten belegt. Sie geben Auskunft über den momentanen Programmzustand (siehe 9.3.3).

Sätze

Sätze entsprechen normalerweise Zeilen, sie werden also durch das LF voneinander getrennt. Man kann aber auch durch Setzen der speziellen Variable RS einen anderen Satztrenner festlegen. Wird RS mit einer leeren Zeichenkette belegt, so werden Sätze durch Leerzeilen getrennt.

Beim Lesen eines Satzes wird dieser von awk in Felder zerlegt. Die Trennung erfolgt normalerweise durch Whitespace (Leerzeichen oder Tabulatoren), kann aber durch Setzen der Variable FS anders geregelt werden. Wird FS mit einer leeren Zeichenkette belegt, so wird aus jedem einzelnen Zeichen ein Feld.

Enthalten RS und FS mehr als ein Zeichen, so wird ihr Inhalt als regulärer Ausdruck aufgefaßt. Oft findet man auch awk-Aufrufe folgender Gestalt:

```
awk -v 'FS=^\"?|\"?|\"?|\"?|$' '{print $3, $2}' datei
```

Hier wird die Zuweisung der Variable FS auf der Kommandozeile vorgenommen. Der oben angegebene Ausdruck zerlegt CSV-Dateien, wie sie von Excel oder dBase exportiert werden, in einzelne Felder.

Arrays (Vektoren)

awk verfügt über ein- und mehrdimensionale Vektoren. Diese weisen zwei Besonderheiten auf:

- Die Größe eines Vektors muß nicht explizit angegeben werden. Neue Elemente werden einfach eingefügt. Elemente werden durch ihren Index bestimmt. Existiert ein Index noch nicht, so wird dieser mit entsprechendem Wert in den Vektor eingetragen.
- Neben den einfachen Strings kennt awk noch assoziative Arrays. Die Indizes eines solchen Arrays sind Strings, z. B. `a["Fred"] = "Feuerstein"`.

Ein Vektorelement wird durch *vektor* [*index*] angesprochen oder initialisiert.

Beispiel: Zählen von Worten

```
{
  gsub(/[,.;!()?()-+]/, "") # Interpunktionszeichen weg
  for (i=1 ; i<=NF ; i++)
    feld[i]=feld[i]+1
}
END {
  for (i in feld)
    print i, feld[i]
}
```

Mehrdimensionale Vektoren verhalten sich wie eindimensionale Vektoren. Intern werden sie zu solchen mit Hilfe von $SUBSEP$ umgeformt. Die Indizes werden durch Kommata voneinander getrennt. Im Zusammenhang mit Operatoren sollten die Tupel mit runden Klammern eingeschlossen werden, z. B.:

```
if ((i,j,k) in vektor)
```

9.3.3 Eingebaute Variable

awk verfügt über einige eingebaute Variable. Die wichtigsten davon sind:

ARGC

Anzahl der Befehlszeilenparameter

ARGV

Array der Befehlszeilenparameter. Die Indizes laufen von 0 bis ARGC-1. Durch ändern von ARGV kann man vom Skript aus weitere Dateien öffnen.

CONVFMT

Das voreingestellte Format für Zahlen. Standardwert ist "%.6g".

ENVIRON

Stellt die Umgebungsvariablen als assoziatives Array zur Verfügung. Z.B. liefert ENVIRON["HOME"] unser Homerverzeichnis.

ERRNO

Text zum letzten aufgetretenen Fehler bei einer Dateioperation

FIELDWIDTHS

Wenn man diese Variable mit einer durch Leerzeichen getrennten Liste von Zahlen füllt, so werden die Felder nicht durch die in FS angegebenen Trennzeichen, sondern an den entsprechenden festen Positionen getrennt.

FILENAME

Name der aktuellen Eingabedatei.

FNR

Die Nummer des aktuellen Eingabesatzes. Ein awk '{print FNR, \$0}' liefert ein Listing mit Zeilennummern.

FS

legt die Trennzeichen für die Felder in einer Eingabezeile (= Record, siehe RS) fest; **Voreinstellung:** " ". FS kann ein Einzelzeichen, ein regulärer Ausdruck oder ein leerer String sein. Im letzten Fall ist jedes Einzelzeichen ein eigenes Feld.

IGNORECASE

Hat diese Variable einen von Null verschiedenen Wert, so werden alle Stringvergleiche, das Trennen der Eingabe mit FS bzw. RS und die Auswertung regulärer Ausdrücke unabhängig von Groß- bzw. Kleinschreibung vorgenommen.

NF

Liefert die Anzahl Felder im aktuellen Eingabesatz.

NR

Anzahl der bisher gelesenen Eingabezeilen.

OFMT

Das Standard-Ausgabeformat für Zahlen. Voreingestellt ist "%.6g"

OFS

Das Feldtrennzeichen für die Ausgabe. Voreingestellt ist ein Leerzeichen.

ORS

Das Satztrennzeichen für die Ausgabe. Voreingestellt ist LF. Braucht man Zeilenenden im DOS-Format (CR+LF), kann man das (unter anderem) mit dem awk erledigen: awk -v 'ORS=\r\n' '{print \$0}'

RLENGTH

Länge der Zeichenkette, die durch den letzten Aufruf der Funktion *match* gefunden wurde.

RS

Trennzeichen zwischen einzelnen Eingabezeilen; **Voreinstellung:** "\n" RS kann ein Einzelzeichen, ein regulärer Ausdruck oder ein leerer String sein. Im letzten Fall werden die einzelnen Sätze durch Leerzeilen getrennt.

RSTART

Anfangsposition der Zeichenkette, die durch den letzten Aufruf der Funktion *match* gefunden wurde

SUBSEP

Trennzeichen für die Indizes "mehrdimensionaler" Felder; **Voreinstellung:** "\034"

9.3.4 Sonderzeichen in regulären Ausdrücken

Zur Wiederholung und damit man nicht blättern muß, hier nochmals die Zeichen mit besonderer Bedeutung in regulären Ausdrücken.

^	Beginn einer Zeichenfolge.
\$	Ende einer Zeichenfolge.
.	Jedes beliebige Zeichen.
*	Das vorangehende Muster beliebig oft (auch keinmal).
+	Das vorangehende Muster einmal oder mehrmals.
?	Das vorangehende Muster keinmal oder einmal.
\	Sonderbedeutung des nachfolgenden Metazeichens aufheben. Dazu gehören auch folgende Escape-Sequenzen: \b \f \n \r \t \ooo Zeichen, das dem Oktalwert ooo entspricht
	Logisches oder.
[ab]	Zeichenmenge (a und b).
[a-z]	Zeichenmenge (alle Zeichen von a bis z).
[^ab]	Verneinte Zeichenmenge (nicht a oder b).
()	Klammerung für Teile komplexerer Muster

9.3.5 Zusammengesetzte reguläre Ausdrücke

Seien A und B reguläre Ausdrücke:

A B	Alternation (A oder B)
A B	Konkatenation (A gefolgt von B)

9.4 Suchmuster und Aktionen

Die Anweisungen des awk bestehen aus dem Suchmuster und den in geschweifte Klammern eingeschlossenen Aktionen. Fehlt das Suchmuster, so werden die Aktionen für alle Eingabezeilen durchgeführt. Fehlt die Aktion, so wird die Eingabezeile unverändert ausgegeben. Der folgende awk-Aufruf entspricht einem einfachen grep:

```
awk '/reg. Ausdruck/' Datei
```

Das Skript kann Kommentare enthalten. Diese beginnen mit dem Zeichen # und erstrecken sich bis zum Zeilenende. Leerzeilen sind zulässig. Ein Statement endet normalerweise am Zeilenende. Um Statements auf der nächsten Zeile fortzusetzen, maskiert man das Zeilenende mit einem Backslash.

9.4.1 Suchmuster

Man kann folgende Suchmuster verwenden:

- **BEGIN**
Die BEGIN-Anweisung im Rumpf wird ausgeführt, **bevor** die erste Eingabezeile verarbeitet wird.
- **END**
Die END-Anweisung im Rumpf wird ausgeführt, **nachdem** die letzte Eingabezeile verarbeitet wurde.
- **Ausdruck**
In Abhängigkeit des ausgewerteten Ausdrucks, der einen booleschen Wert liefert, wird die *Aktion* ausgeführt oder nicht.
- **/regulärer Ausdruck/**
Die regulären Ausdrücke entsprechen denen von `egrep`. Wird die aktuelle Eingabezeile durch den regulären Ausdruck abgedeckt, so wird die *Aktion* ausgeführt.
- **vergleichender Ausdruck**
Vergleichende Ausdrücke können umfangreicher sein. Man kann z. B. testen, ob bestimmte Felder einem regulären Ausdruck entsprechen.
- **Suchmuster && Suchmuster**
Suchmuster || Suchmuster
! Suchmuster
Muster können sich aus mehreren Mustern zusammensetzen, die durch die Operatoren **&&** (logisches UND), **||** (logisches ODER) und **!** (Negation). Die Aktion wird genau dann ausgeführt, wenn der Gesamt-Ausdruck einen wahren Wert liefert.
- **Suchmuster1 ? Suchmuster2 : Suchmuster3**
Der Ausdruck wird analog der Sprache C ausgewertet. Liefert Suchmuster1 einen wahren Wert, gibt der Ausdruck Suchmuster2 zurück, sonst Suchmuster3.
- **(Suchmuster)**
() (runde Klammern) dienen der Strukturierung und Vorrangregelung bei Ausdrücken.
- **Suchmuster, Suchmuster**
Zwei durch Komma getrennte Suchmuster ermitteln einen Bereich von Zeilen. Die Aktion wird für den Bereich ausgeführt, der mit der Zeile beginnt, die durch das erste Muster abgedeckt wird, und mit der Zeile endet, die durch das zweite Muster abgedeckt wird.

Beispiel 1: Der Selektor

```
((NF > 2) || (NF < 8)) && ($0 !~ /^[^#]/)
```

wählt alle Zeilen aus, die zwischen 3 und 7 Feldern besitzen und keine Kommentarzeilen sind (also mit "#" beginnen).

Mit "*Zeichenkette* ~ *Muster*" lassen sich beliebige Zeichenketten vergleichen. So extrahiert nachfolgendes Awk-Programm alle Benutzernamen aus der Datei `/etc/passwd`, deren Heimatverzeichnis unterhalb von `/home` liegt:

```
awk -F ':' '$6 ~ /^\/home/ {print $1}' /etc/passwd
```

Die zu vergleichende Zeichenkette ist hier das 6. Feld (\$6) der Passwortdatei (Heimatverzeichnis).

Zeichenklassen

Guter Programmierstil ist die Verwendung sogenannter "Zeichenklassen". Ihre Syntax entspricht nicht nur dem POSIX-Standard, sondern berücksichtigt auch länderspezifischen Unterschiede. Ihr Programm wird somit portabel und ist nicht mehr von einem bestimmten Zeichensatz abhängig. Dazu gleich ein Beispiel:

```
#!/usr/bin/awk -f
BEGIN {
    String = ÄÖÜäöü
    if (String ~ /[A-Za-z]/)    print "A-Z funktioniert.";
    if (String ~ /[[:alpha:]]/) print "Alpha funktioniert.";
}
```

[[:alnum:]]	Alphanumerische Zeichen
[[:alpha:]]	Alphabetische Zeichen
[[:blank:]]	Leerzeichen und Tabulatoren
[[:cntrl:]]	Steuerzeichen
[[:digit:]]	Numerische Zeichen
[[:graph:]]	Druck- und sichtbare Zeichen (Ein Leerzeichen ist druckbar aber nicht sichtbar, wogegen ein "a" beides ist)
[[:print:]]	Druckbare Zeichen (also keine Steuerzeichen)
[[:punct:]]	Punktierungszeichen <i>Punctuation characters</i> (Zeichen die keine Buchstaben, Zahlen, Steuerzeichen oder Leerzeichen sind, z. B. ".", ",", ":",")
[[:space:]]	Druckbare aber nicht sichtbaren Zeichen (Leerzeichen, Tabulatoren, Zeichenende, etc.)
[[:lower:]]	Kleinbuchstaben
[[:upper:]]	Großbuchstaben
[[:xdigit:]]	Hexadezimale Zeichen (0-9,A-F,a-f)

Um zu testen, ob in einer Variablen eine gültige Zahl (ganzzahlig) gespeichert ist, bietet sich folgendes Konstrukt an:

```
echo "4711" | awk '/^[[:digit:]]+$/ {print "eine Zahl"}'
```

9.4.2 Operatoren

Operator	Bedeutung
(...)	Gruppierung
\$	Feldreferenz
++ --	Inkrement und Dekrement, in postfix- oder präfix-Notation
^	Potenzieren
+ - !	unäres plus, minus und logische Negation

* / %	Multiplikation, Division, Modulo-Funktion
+ -	Addition, Subtraktion
space	Stringverkettung
< > <= >= == !=	Vergleiche
~ !~	Vergleich mit regulären Ausdrücken; rechter String im linken enthalten/nicht enthalten
in	Test auf Enthaltensein in einem Array
&&	Logisches Und
	Logisches Oder
? :	Bedingter Ausdruck, wie in C (A1 ? A2 : A3)
= += -= *= /= %= ^=	Zuweisung, wie in C

9.4.3 Aktionen (Steueranweisungen)

Aktionen werden in geschweifte Klammern { } eingeschlossen. Aktionen können Zuweisungen, Verzweigungen und Schleifen enthalten. Die Syntax ist dabei ähnlich C.

Während Sie mit Mustern die Zeilen adressieren, legen Sie mit Aktion fest, was für diese Zeile zu tun ist. Eine Aktion kann sich aus vielen Anweisungen zusammensetzen. Dieses sind dann durch Semikolon oder neue Zeile voneinander zu trennen. Die meisten Steueranweisungen haben eine direkte Entsprechung in C. Anweisungen in einer Aktion können sein:

- Ausdrücke mit Konstanten, Variablen, Zuweisungen, Funktionsaufrufe usw.
- **if (Bedingung) Anweisung1 [else Anweisung2]**
Wenn *Bedingung* einen wahren Wert liefert, wird *Anweisung1* ausgeführt, andernfalls nicht. Existiert der *else*-Zweig, wird im Falsch-Fall *Anweisung2* ausgeführt. Bei geschachtelten *if-else*-Anweisungen gehört das *else* zum letzten freien *if*.
- **while (Bedingung) Anweisung**
Solange *Ausdruck* einen wahren Wert liefert, wird *Anweisung* ausgeführt.
- **do Anweisung while (Bedingung)**
Führt *Anweisung* aus, wertet die *Bedingung* aus und iteriert die *Anweisung* falls die *Bedingung* wahr ist.
- **for (Ausdruck1; Ausdruck2; Ausdruck3) Anweisung**
ist eine andere Schreibweise für:

```
Ausdruck1;
while (Ausdruck2)
{ Anweisung;
  Ausdruck3 }
```

- **for (Variable in Array) Anweisung**
Das Array wird über alle seine Elemente durchlaufen, d. h. *Variable* nimmt nacheinander den Wert der jeweiligen Array-Komponente an.
- **break**
Innerhalb einer *do*-, *for*- oder *while*-Schleife bewirkt *break* den sofortigen Ausstieg aus dieser Schleife.
- **continue**
Innerhalb einer *do*-, *for*- oder *while*-Schleife wird der nächste Schleifendurchlauf

- gestartet.
- **next**
veranlaßt das Lesen einer neuen Eingabezeile und startet das Programm erneut. Die BEGIN-Aktion wird hierbei aber nicht ausgeführt.
 - **delete Array[Index]**
Löscht eine Komponente aus einem assoziativen Array.
 - **delete Array**
Löscht das gesamte assoziative Array.
 - **exit [Ausdruck]**
Bewirkt den Sprung zur END-Aktion. Ist diese nicht vorhanden, ist der Wert von *Ausdruck* der Exit-Status des *awk*-Kommandos.
 - **{ Anweisungen }**
Überall dort, wo eine Anweisung stehen darf, dürfen die geschweiften Klammern einen Block einleiten. Innerhalb von geschweiften Klammern können beliebige Anweisungen, durch Semikolon oder neue Zeile getrennt, stehen.
 - **return [Ausdruck]**
Innerhalb einer Funktion wird diese mit dem Wert von *Ausdruck* verlassen.

9.4.4 Ein- und Ausgabe-Anweisungen

Die beiden Funktionen `print` und `printf` wurden schon am Anfang behandelt. Für die Eingabe gibt es noch die Funktion `getline`. Die folgende Tabelle faßt die Ein- und Ausgabemöglichkeiten zusammen:

Anweisung	Bedeutung
<code>close(Datei)</code>	Datei (oder Pipe) schließen.
<code>getline</code>	nächste Zeile in \$0 laden.
<code>getline <Datei</code>	Nächste Zeile aus bestimmter Datei lesen.
<code>getline Variable</code>	Nächste Zeile in Variable, statt \$0, laden.
<code>getline Variable <Datei</code>	Zeile aus Datei in Variable laden.
<code>next</code>	Nächste Zeile lesen und ab Anfang des Skriptes bearbeiten.
<code>nextfile</code>	Aktuelle Datei schließen und mit nächster fortfahren.
<code>print</code>	Gibt den aktuellen Satz aus.
<code>print Ausdrucksliste</code>	Gibt Ergebnis der Ausdrücke aus. Auch: <code>print (...)</code> .
<code>print Ausdrucksliste > Datei</code>	Schreibt Ergebnis der Ausdrücke in Datei.
<code>print Ausdrucksliste Programm</code>	Schreibt Ergebnis der Ausdrücke in eine Pipe.
<code>printf Format, Ausdrucksliste</code>	Gibt Ergebnisse formatiert aus. Auch: <code>printf (...)</code> .
<code>printf Format, Ausdrucksliste > Datei</code>	formatierte Ausgabe in Datei.
<code>printf Ausdrucksliste Programm</code>	Schreibt Ergebnis der Ausdrücke in eine Pipe.
<code>fflush([Datei])</code>	Erzwingt das Schreiben der Puffer

Bei der Ausgabe werden die Variablen \$0, OFS und ORS berücksichtigt.

Zur Ausgabe in Dateien werden die Zeichen der Ein- und Ausgabeumleitung von UNIX verwendet. Dateinamen müssen in Anführungszeichen (" ") eingeschlossen werden!

Beispiel: (Entsprechend des Wertes der Zahl in der ersten Spalte der Datei soll die Datei in zwei

Dateien zerlegt werden.

```
awk '$1 > 100 { print > "klein"};
    $1 <= 100 {print > "gross"}'
```

Die Ausgabe einer `print`- bzw. `printf`-Anweisung kann sogar in eine Pipe geschrieben werden. Das folgende Programm sortiert alle Zeilen einer Datei die den String "test" enthalten:

```
awk '/test/ { print | "sort" }'
```

Dateien, die Sie öffnen, sollten Sie auch wieder schließen. Zum einen dürfen Sie meistens nur eine endliche Zahl von Dateien gleichzeitig geöffnet haben, zum anderen kann es Probleme geben, wenn die Dateien auch von anderen Prozessen verwendet werden sollen (Logfiles, Infodateien etc.). Auch Pipes sollten wieder geschlossen werden! Sie erreichen dies mit:

```
close(dateiname)
close(kommando)
```

Hierbei muß der komplette Name mit Pfad - wie beim Öffnen - wieder angegeben werden.

Das Format für `printf` stimmt im wesentlichen mit dem für die entsprechende C-Funktion überein. Die `printf`-Kontrollzeichen (werden immer mit einem Prozentzeichen eingeleitet):

Zeichen	drucke den entsprechenden Ausdruck als
c	ASCII-Zeichen
d	Integerwert (dezimal)
e	<code>[-]d.dddE[+-]dd</code>
f	<code>[-]ddd.ddd</code>
g	e oder f Format (wählt das kürzere)
o	Oktalwert (ohne Vorzeichen)
s	Zeichenkette
x	Hexadezimalwert (ohne Vorzeichen)
%	Verwendet kein Argument! gibt ein % aus!

Eingabe mit `getline`

Mit dieser Funktion kann explizit der nächste Eingabe-Rekord gelesen werden. Das Programm fährt dabei mit der Abarbeitung fort und wird nicht neu gestartet. `getline` kann drei Rückgabewerte annehmen: 1, wenn der Rekord gelesen werden konnte; 0, wenn das Dateiende erreicht ist und -1, wenn beim Lesen ein Fehler auftrat. `getline` kann auf folgende Weisen eingesetzt werden:

Ausdruck	setzt folgende Variablen
<code>getline</code>	<code>\$0, NF, NR, FNR</code>
<code>getline variable</code>	<code>variable, NR, FNR</code>
<code>getline <dateiname</code>	<code>\$0, NF</code>
<code>getline variable <dateiname</code>	<code>variable</code>

<i>kommando</i> <i>getline</i>	<i>\$0, NF</i>
<i>kommando</i> <i>getline variable</i>	<i>variable</i>

9.4.5 Stringfunktionen

Hier erläutere ich nur einige der Stringfunktionen, genaueres erfährt man in der Manual-Page.

`gensub(r, s, h [, t])`

Ersetzt einige oder alle Vorkommen eines einem reg. Ausdruck entsprechenden Teilstrings. Das entspricht dem `substitute` im `sed` bzw. `vi`.

`gsub(r, s [, t])`

Macht dasselbe wie `gensub`, nimmt die Ersetzung aber direkt im entsprechenden Feld vor und liefert die Anzahl vorgenommener Ersetzungen.

`index(s, t)`

Liefert die Position der Zeichenkette `s` in `t`, bzw. 0, wenn `s` nicht in `t` enthalten ist.

`length([s])`

Liefert die Länge von `s`. Ohne Angabe von `s` erhält man die Länge des Eingabesatzes `$0`

`match(s, r)`

Liefert die Position in `s`, ab der der reguläre Ausdruck `r` paßt, anderenfalls 0. Diese Funktion wird häufig dazu genommen, in einem Suchmuster zu bestimmen, ob ein Feld einem reg. Ausdruck entspricht. Beispiel:

```
awk -F: 'match($7, "/bin/bash")' /etc/passwd
```

`split(s, a, [, r])`

Zerlegt den String `s` in ein Array `a` und liefert die Anzahl Teilstrings. Die Zerlegung erfolgt an Hand des reg. Ausdrucks `r`. Falls dieser weggelassen wird, verwendet `awk` den Inhalt der Variable `FS`

`sprintf(Format, Ausdrucksliste)`

Liefert formatierte Ergebnisse der Ausdrücke als String.

`sub(r, s [, t])`

wie `gsub`, nimmt aber nur die erste Ersetzung vor.

`substr(s, i [, n])`

Liefert den `n` Zeichen langen Teilstring von `s` ab Position `i`. Wird `n` weggelassen, so erhält man den Rest von `s` ab Position `i`.

`toupper(s)`

Konvertierung in Großbuchstaben

`tolower(s)`

Konvertierung in Kleinbuchstaben

9.4.6 System-Funktionen

Die Funktion `system(ausdruck)` führt das durch `ausdruck` spezifizierte Kommando in der Shell aus. Der Rückgabewert der Funktion ist der Exit-Status des Kommandos.

Die Funktion `systeme()` liefert die Systemzeit in Sekunden seit dem 1.1.1970 0.00 Uhr.

Die Funktion `strftime(Format [, Zeit])` liefert eine formatierte Zeitausgabe. Ausgabe der Systemzeit, wenn nur der Formatstring angegeben wird.

Beispiel:

```
BEGIN {
    start = systeme }
```

```
{ for(i = 1; i > 100000; i++) }
END {
  end = systime;
  print "Gesamtzeit betrug " end - start " Sekunden." }
```

9.4.7 Arithmetische Funktionen

Funktion	Beschreibung
atan2(x,y)	Arcustangens von x/y (Im Bereich von -Pi bis +Pi)
cos(x)	Cosinus von x
exp(x)	Exponentialfunktion von x (e hoch x)
int(x)	ganzzahliger Anteil von x
log(x)	natürlicher Logarithmus von x
rand()	Zufahlszahl im Bereich 0 <= r < 1
sin(x)	Sinus von x
sqrt(x)	Quadratwurzel von x
srand([x])	Ausgangswert für rand ändern

- Pi entspricht atan2(0,-1)
- e entspricht exp(1)
- Der Zehnerlogarithmus von x entspricht log(x) / log(10)

9.5 Benutzerdefinierte Funktionen

In awk (genauer nawk, gawk) können auch Funktionen definiert werden. Die Definitionen werden üblicherweise am Ende des Programms notiert. Deklarationen sind nicht nötig. Eine Funktionsdefinition genügt der Form

```
function name(par1,par2, ...)
{
  Anweisungen
}
```

Die öffnende Klammer **muß** direkt auf den Funktionsnamen folgen. Der Funktionsname muß eindeutig gewählt werden. Die Parameterliste ist eine Folge von Variablennamen, die durch Komma getrennt sind. Parameterübergabe erfolgt bei skalaren Variablen als call-by-value, bei Arrays hingegen als call-by-reference. Daher lassen sich die Array-Inhalte innerhalb der Funktion ändern. Parameter sind lokale Variablen, globale Variable gleichen Namens sind daher in der Funktion nicht zugreifbar.

Benutzerdefinierte Funktionen sind typlos, die gleiche Funktion kann ganze Zahlen, Gleitpunktzahlen oder Strings zurückgeben. Die Funktion wird beendet, wenn die letzte Anweisung abgearbeitet wurde. Üblicherweise wird eine Funktion mit der Anweisung

```
return [Ausdruck]
```

beendet. Ist ein Ausdruck angegeben, wird der Wert dieses Ausdrucks an das aufrufende Programm zurückgegeben.

Innerhalb der Funktion neu eingeführte Variablen sind global. Die Funktionsargumente sind nur im Funktionsrumpf gültig. Neue Variablen, die im Rumpf definiert werden, sind global gültig. Werden beim Aufruf weniger Parameter als vorgeschrieben angegeben, so werden die fehlenden Parameter mit "" initialisiert.

9.6 awk-Aufruf

Der Aufruf von awk wurde schon am Anfang (9.2) kurz behandelt. Hier noch einige Ergänzungen.

9.6.1 Variablen in der Kommandozeile setzen

```
awk '...' flag=1 datei flag=0 datei
```

Wertzuweisung erfolgt zu dem Zeitpunkt, zu dem auf eine durch den Parameter angegebene Datei zugegriffen würde

9.6.2 Zugriff auf Shell-Parameter

1. Möglichkeit:
Sei *var* ein Shell-Parameter.
 - ◆ '*var*' awk betrachtet den Wert des Parameters als numerischen Wert
 - ◆ "*var*" awk betrachtet den Wert des Parameters als String-Wert
2. Möglichkeit:
Wird das awk-Programm mit "..." geklammert, so bezeichnen \$0, \$1 etc. die Shell-Parameter. Die awk-Feldvariablen müssen dann über \\$\$1 oder \\${1} (je nach Shell) referenziert.
3. Möglichkeit
Shell-Parameter in der awk-Kommandozeile an Variablen zuweisen (siehe 9.6.1).

9.6.3 Die Kommandozeilenoptionen

Einige Optionen wurden schon verwendet, ohne Bedeutung näher zu erläutern. Dies soll nun nachgeholt werden:

- F *Feldtrenner*** awk arbeitet auf Feldern einer Eingabezeile. Normalerweise dient das Leerzeichen/Tabulator zur Trennung einzelner Felder. Mit der Option -F wird der Wert der internen Variable **FS** (field separator) verändert.
- v** Eine im Programm verwendete Variable kann somit "von außen" initialisiert werden (eine interne Initialisierung wird damit nicht überschrieben).
- f** awk liest den Quellcode aus der angegebenen Datei.
Programmdatei
- W *compat*** GNU awk verhält sich wie UNIX awk, d.h. die GNU-Erweiterungen werden nicht akzeptiert.
- W *help*** Eine Kurzanleitung wird ausgegeben.
- W *posix*** GNU awk hält sich exakt an den POSIX-Standard.

9.7 Beispiele

9.7.1 Notendurchschnitt

Gegeben ist eine Datei mit Studentennamen und Noten, wobei jeder Student beliebig oft auftauchen darf. Zum Beispiel:

```
Maier 3.3
Mueller 1.3
Huber 2.3
Maier 4.3
Huber 2.6
Maier 4.6
```

Das folgende Programm berechnet die Durchschnittsnote für jeden Studenten:

```
{ sum[$1] += $2 ; oft[$1] += 1 }
END { for (Student in sum)
      print Student, sum[Student]/oft[Student]
    }
```

Die Programmausgabe für das Beispiel lautet:

```
Huber 2.45
Mueller 1.3
Maier 4.06667
```

9.7.2 Wortliste

Wie schon bei der Erläuterung der Arrays, wird hier die Anzahl jedes Wortes einer Datei ermittelt. Nur wird hier mit Hilfe eines regulären Ausdrucks festgelegt, daß die awk-Variable FS (in der die Feldtrennzeichen für die Eingabezeilen festgelegt werden) bestimmte Zeichen nicht annimmt. Alle anderen nicht aufgeführten Zeichen sind dann Trennzeichen.

```
BEGIN {FS="^[a-zA-Z0-9ÄÖÜáâãäåæéêëìíîïóôõùú]"}
      {for (j=1; j<=NF; j++)
        {anzahl [$j] ++}
      }
END   {for (j in anzahl)
      {printf ("%40s %4d\n", j, anzahl [j] ) }
      }
```

9.7.3 Spielen mit dem Mailspool

Beispielsweise werden von Zeit zu Zeit werden einige spezielle Nachrichten (z. B. monatliche Berichte) in einem speziellen Format (Subject: '[Monatsbericht] Monat , Abteilung') geschickt. Plötzlich, am Ende des Jahres entscheiden wir uns all diese Nachrichten zusammen, getrennt von den anderen, zu speichern.

```
BEGIN { # Variablen setzen
BEGIN_MSG = "From"
BEGIN_BDY = "Precedence:"
MAIN_KEY = "Subject:"
VALIDATION = "[MONATSBERICHT]"

HEAD = "NO"; BODY = "NO"; PRINT="NO"
OUT_FILE = "Month_Reports"
}

{ # keine Bedingung, also alle Zeilen bearbeiten
if ( $1 == BEGIN_MSG ) {
  HEAD = "YES"; BODY = "NO"; PRINT="NO"
}

if ( $1 == MAIN_KEY ) {
  if ( $2 == VALIDATION ) {
    PRINT = "YES"
    $1 = ""; $2 = ""
  }
}
```

Betriebssystem UNIX/Linux

```
    print "\n\n"$0"\n" >> OUT_FILE
  }
}

if ( $1 == BEGIN_BDY ) {
  getline
  if ( $0 == "" ) {
    HEAD = "NO"; BODY = "YES"
  } else {
    HEAD = "NO"; BODY = "NO"; PRINT="NO"
  }
}

if ( BODY == "YES" && PRINT == "YES" ) {
  print $0 >> OUT_FILE
}
}
```

9.7.4 Rechner

Das folgende Beispiel beschreibt einen Taschenrechner, der Eingaben in der Infixnotation akzeptiert. Das Programm muß folgende Grammatik nachbilden, die einen Infixausdruck beschreibt:

```
expr    ->    term
           expr + term
           expr - term

term    ->    factor
           term * factor
           term / factor

factor  ->    number
           ( expr )
```

Die obige Grammatik beinhaltet auch die unterschiedliche Wertigkeit und die Assoziativität der Operatoren. Das zugehörige Programm sieht folgendermaßen aus:

```
# Infix Rechner
# Das Hauptprogramm wird nur auf die aktuelle Eingabezeile
# angewendet, wenn sie mehr als ein Feld beinhaltet. D. h.
# in jeder Zeile darf nur ein Ausdruck stehen und seine
# Operanden muessen von Leerzeichen eingeschlossen werden.
# z. B. ( 5 + 2 ) * 3

NF > 0 {
  f = 1
  e = expr()
  if ( f <= NF) printf("error at %s\n", $f)
  else printf("\t%.8g\n", e)
}

# Unterprogramme, die zur Erkennung der Struktur des
# Ausdrucks verwendet werden.

function expr( e ) {
  # term | term [+ -] term
  e = term()
  while ( $f == "+" || $f == "-" )
    e = $(f++) == "+" ? e + term() : e - term()
  return e
}

function term( e ) {
  # factor | factor [*/] factor
  e = factor()
  while ( $f == "*" || $f == "/" )
    e = $(f++) == "*" ? e * factor() : e / factor()
  return e
}
```

```
}  
  
function factor( e) { # number | (expr)  
    if ($f ~ /^[+-]?([0-9]+[.]?[0-9]*|.[0-9]+)$/) {  
        return $(f++)  
    } else if ($f == "(") {  
        f++  
        e = expr()  
        if ($(f++) != ")")  
            printf("error: missing ) at %s\n", $f)  
        return e  
    } else {  
        printf("error: expected number or ( at %s\n", $f)  
        return 0  
    }  
}
```



[Zum Inhaltsverzeichnis](#)



[Zum nächsten Abschnitt](#)

Copyright © FH München, FB 04, Prof. Jürgen Plate

Letzte Aktualisierung: 21. Sep 2011



Vorlesung "UNIX"

von Prof. Jürgen Plate

10 Kommandoreferenz

10.1 vi Short Quick Reference Guide

vi-Kurzreferenz

Der Editor vi ('vi' steht für 'visual') ist ein bildschirmorientierter Editor, d. h. der Text ist in seiner aktuellen Version auf dem Bildschirm zu sehen. Es werden hier nur die wichtigsten Kommandos für Einsteiger aufgeführt. Aufruf:

vi Dateiname

Ist die Datei vorhanden, wird sie in den Editorpuffer geladen, andernfalls wird sie neu angelegt. Nach dem Aufruf des vi befindet sich der Benutzer im Kommandomodus. Die Arbeit spielt sich in zwei Ebenen ab.

Kommandomodus:

Freies Positionieren des Cursors innerhalb des Textes, Umsetzen von Textblöcken, Schreiben und Lesen von Dateien, Löschen von Textblöcken und Aufruf von UNIX-Kommandos.

Eingabemodus:

Einfügen von Text oder Überschreiben vorhandener Textpassagen.

Das Umschalten zwischen den Modi erfolgt mit der ESC-Taste.

Cursor-Bewegung (falls die Cursortasten nicht das Gewünschte tun)

```
      k
      ^
h < > l
      v
      j
```

Die Handhabung des Kommandomodus des vi macht die Bedienung für den Anfänger etwas ungewohnt. Die Cursorpositionierung erfolgt mit verschiedenen Tasten, für die Eingabe von Dateibefehlen oder anderen Funktionen (z. B. Suchen/Ersetzen) ist die Eingabe eines Doppelpunktes nötig. Man befindet sich dann in ex-Modus, der ähnliche Eigenschaften hat, wie der (--> ed). Bleiben wir gleich bei den ex-Befehlen. Nach der Eingabe des ':' springt der Cursor in die letzte Zeile und erlaubt die Eingabe eines ex-Befehls, der mit der Return-Taste abgeschlossen wird. Es sind nach dem Doppelpunkt alle ed-Befehle möglich. Für den Einstieg genügen:

```
:wq      Text auf die Platte schreiben und vi beenden.
ZZ       Text auf die Platte schreiben und vi beenden.
:w       Text auf Platte schreiben.
:w Datei Text auf die angegebenen Datei schreiben.
:w>>Datei Text an die angegebene Datei anhängen.
:x       wie :wq.
:q       Abbrechen, ohne den Text auf Platte zu schreiben.
:q!      Abbrechen, ohne den Text auf Platte zu schreiben.
:r Datei Text aus der angegebenen Datei nach der momentanen
        Zeile einfügen.
:e Datei Neue Datei bearbeiten
```

Umschalten in den Eingabemodus:

```
a       (append) Eingabe rechts vom Cursor
```

Betriebssystem UNIX/Linux

A	Eingabe am Zeilenende
i	(insert) Eingabe links vom Cursor
I	Eingaben am Zeilenanfang
o	Eingabe in der folgenden Zeile, Spalte 1.
O	Eingabe in der vorhergehenden Zeile, Spalte 1.

Der Eingabemodus wird durch Drücken der ESC-Taste verlassen. Wenn es die Hardware ermöglicht, ändert sich die Form des Cursors beim Wechsel zwischen Eingabe- und Kommandomodus.

Blättern auf dem Bildschirm können Sie mit folgenden Tastenkombinationen:

CTRL-F	Eine Bildschirmseite vorwärts
CTRL-B	Eine Bildschirmseite rückwärts
CTRL-D	Eine halbe Bildschirmseite vorwärts
CTRL-L	Bildschirm neu aufbauen (wenn z.B. eine Nachricht kam)

Cursorpositionierung (im Kommandomodus):

Normalerweise ist der vi an die Pfeiltasten der Tastatur richtig angepaßt. Daneben gibt es noch folgende Möglichkeiten (Auswahl):

h	Zeichen links (auch Backspace)
l	Zeichen rechts (auch blank)
k	Spalte höher
j	Spalte tiefer
b	Wortanfang
e	Wortende
w	Anfang nächstes Wort
H	Anfang erste Zeile des Bildschirms
L	Anfang letzten Zeile des Bildschirms
0	(Null) Zeilenanfang
\$	Zeilenende
RETURN	nächste Zeile
nG	(Go) Gehe zu Zeile n. n ist eine Zeilennummer. Fehlt die Zahl, wird zur letzten Zeile der Datei gesprungen.

Löschen von Text (im Kommandomodus) und Text ändern. Die Änderungskommandos wechseln automatisch in den Eingabemodus, der mit der ESC-Taste verlassen werden muß.

x	Zeichen unter dem Cursor löschen
X	Zeichen vor dem Cursor löschen
dw	ab Cursorpos. bis Wortende löschen
db	ab Cursorpos. bis Wortanfang löschen
dd	ganze Zeile löschen
r	Zeichen ersetzen (kein Abschluß mit ESC)
R	Mehrere Zeichen ersetzen
s	Ein Zeichen ersetzen und zusätzliche Zeichen anschließend einfügen
cc	Gesamte Zeile ändern
cw	Wort ändern

Suchen nach Zeichen oder Strings. Außer 'fx' und 'Fx' müssen die Suchkommandos mit der Return-Taste abgeschlossen werden.

fx	Zeichen 'x' in der momentanen Zeile suchen (vorwärts)
Fx	Zeichen 'x' in der momentanen Zeile suchen (rückwärts)
/str	String 'str' vorwärts in der Datei suchen
?str	String 'str' rückwärts in der Datei suchen
//	Letzten Suchbefehl wiederholen (vorwärts)
??	Letzten Suchbefehl wiederholen (rückwärts)

Sonstige Kommandos:

J Zeilen verbinden (nächste Zeile anhängen)
 u Letzten Befehl rückgängig machen
 U Aktuelle Zeile wiederherstellen
 . Letztes Kommando wiederholen

10.2 Liste der Meta-Zeichen in regulären Ausdrücken

Wirkung	Dateinamen-Expandierung	awk, egrep	sed, ed, grep, csplit, pg, more	ex, vi
ein beliebiges Zeichen	?	.	.	.
beliebige Zeichenkette	*	.*	.*	.*
vorhergehendes Zeichen 0 mal, 1 mal oder mehrmals		*	*	*
1 mal oder mehrmals		+	\{ 1, } \	
0 mal oder 1 mal		?	\{ 0,1 } \	
genau n mal			\{ n \}	
n bis m mal			\{ n,m \}	
mindestens n mal			\{ n, \}	
aus der Menge ¹⁾	[...]	[...]	[...]	[...]
... aus der Komplementärmenge	[! ...]	[^ ...]	[^ ...]	[^ ...]
Ausdruck nur am Zeilenanfang suchen		^Ausdr.	^Ausdr.	^Ausdr.
Ausdruck nur am Zeilenende suchen		Ausdr.\$	Ausdr.\$	Ausdr.\$
Zeichenfolge am Wortanfang suchen	abc*			\<Ausdr.
Zeichenfolge am Wortende suchen	*abc			Ausdr.\>
logisches Oder (Disjunktion)		A1 A2		

¹⁾ Klammern können enthalten:

- Zeichen dicht geschrieben: [abcx123]
- Bereich von ... bis ...: [a-fu-x]
(alle Zeichen, deren ASCII-Code dazwischen liegt)
- das Zeichen "]" (nur als erstes): []abx17]
- das Zeichen "-" (nur als erstes oder letztes): [-+.0-9]

Wiederholungs-Operatoren

(Quantifier) beschreiben, wie oft eine bestimmte Zeichenfolge erkannt werden soll. Sie werden zusammen mit Zeichen-Erkennungsmustern eingesetzt, um nach mehreren Zeichen zu suchen. Achtung: Je nach Programm werden die Operationen verwendet bzw. ignoriert; man sollte also einen Blick in die Dokumentation werfen.

Operator	Beschreibung	Beispiel	Ergebnis
?	Passt auf ein beliebiges Zeichen einmal, sofern es vorhanden ist	egrep "?erd" sample.txt	Erkennt "berd", "herd" etc. und "erd"
*	Erkennt das vorstehende Element mehrfach, sofern es vorhanden ist	egrep "n.*rd" sample.txt	Erkennt "nerd", "nrd", "neard" etc.
+	Erkennt das vorstehende Element ein- oder mehrmals	egrep "[n]+erd" sample.txt	Erkennt "nerd", "nnerd" etc., aber nicht "erd"
{n}	Erkennt das vorstehende Element genau n-mal	egrep "[a-z]{2}erd" sample.txt	Erkennt "cherd", "blerd" etc., aber nicht "nerd", "erd", "buzzerd" etc.
{n,}	Erkennt das vorstehende Element mindestens n-mal	egrep ".{2,}erd" sample.txt	Erkennt "cherd" und "buzzerd", aber nicht "nerd"
{n, N}	Erkennt das vorstehende Element mindestens n-mal, aber höchstens N-mal	egrep "n[e]{1,2}rd" sample.txt	Erkennt "nerd" und "neerd"

Anker

Diese beschreiben, wo das Muster erkannt werden soll. Sie sind nützlich, wenn man nach gemeinsamen Zeichenkombinationen sucht.

Operator	Beschreibung	Beispiel	Ergebnis
^	Passt auf den Anfang einer Zeile	S/^/rhabarber /	Fügt "rhabarber " am Anfang der Zeile ein
\$	Passt auf das Ende einer Zeile	S\$/ rhabarber/	Fügt " rhabarber" am Ende der Zeile ein
\<	Erkennt den Anfang eines Wortes	S\</rhabarber/	Fügt "rhabarber" am Anfang des Wortes ein
		egrep "\<rhabarber" sample.txt	Erkennt "rhabarberkuchen" etc.
\>	Erkennt das Ende eines Wortes	S/>/rhabarber/	Fügt "rhabarber" am Ende des Wortes ein
		egrep "\>rhabarber" sample.txt	Erkennt "laberrhabarber" etc.
\b	Passt am Anfang oder Ende eines Wortes	egrep "\brhabarber" sample.txt	Erkennt "rhabarberkuche" und "laberrhabarber"
\B	Passt in der Mitte eines Wortes	egrep "\Brhabarber" sample.txt	Erkennt "laberhabarbercake" etc.

Posix-Klassen

In vielen neueren Implementationen können innerhalb der eckigen Klammern nach POSIX auch Klassen angegeben werden, die selbst wieder eckige Klammern enthalten. Sie lauten beispielsweise:

- [:alnum:]** Alphanumerische Zeichen: [:alpha:] und [:digit:].
- [:alpha:]** Buchstaben: [:lower:] und [:upper:].
- [:blank:]** Leerzeichen und Tabulator.
- [:cntrl:]** Steuerzeichen. Bei ASCII sind das die Zeichen 00 bis 1F, und 7F (DEL).

- [:digit:]** Ziffern: 0, 1, 2,... bis 9.
 - [:graph:]** Graphische Zeichen: [:alnum:] und [:punct:].
 - [:lower:]** Kleinbuchstaben¹: nicht notwendigerweise nur von a bis z.
 - [:print:]** Druckbare Zeichen: [:alnum:], [:punct:] und Leerzeichen.
 - [:punct:]** Zeichen wie: ! " # \$ % & ' () * + , - . / : ; < = > ? @ [\] ^ _ ` { | } ~ .
 - [:space:]** whitespace: Horizontaler und vertikaler Tabulator, Zeilen- und Seitenvorschub, Wagenrücklauf und Leerzeichen.
 - [:upper:]** Großbuchstaben¹: nicht notwendigerweise nur von A bis Z.
 - [:xdigit:]** Hexadezimale Ziffern: 0 bis 9, A bis F, a bis f.
- ¹Was Buchstaben sind, ist im Allgemeinen *locale*-abhängig.

Vordefinierte Zeichenklassen

Es gibt vordefinierte Zeichenklassen, die allerdings nicht von allen Implementierungen unterstützt werden, da sie lediglich Kurzformen sind und auch durch eine *Zeichenauswahl* beschrieben werden können. Wichtige Zeichenklassen sind:

- **\d** : eine Ziffer [0-9]
- **\D** : ein Zeichen, das keine Ziffer ist, also [^\d]
- **\w** : ein Buchstabe, eine Ziffer oder der Unterstrich, also [a-zA-Z_0-9] (und evtl. weitere Buchstaben, z. B. Umlaute)
- **\W** : ein Zeichen, das weder Buchstabe noch Zahl noch Unterstrich ist, also [^\w]
- **\s** : whitespace; meistens die Klasse der Steuerzeichen \f, \n, \r, \t und \v
- **\S** : ein Zeichen, das kein whitespace ist [^\s]

10.3 Tabellen zur Shell-Programmierung

Befehlsmöglichkeiten

;	es können mehr als ein Befehl in einer Zeile untergebracht werden, vorausgesetzt die Befehle werden durch den Strichpunkt getrennt
&	gibt man einen Befehl gefolgt von dem Et-Zeichen (&) ein, so wird dieser Befehl im Hintergrund ausgeführt, der Befehl nimmt also das Terminal nicht in Anspruch und man kann in dieser Zeit weiterarbeiten; die Standardausgabe dieses Befehls erscheint trotzdem am Bildschirm; wird ein Befehl in den Hintergrund geschickt, gibt das UNIX-System automatisch eine Nummer, die Prozeßidentifikationsnummer (siehe ps und kill), dieses Befehls aus

Wildcards

*	das Sternchen steht für die Namen (Zeichen) aller Dateien des Verzeichnisses, es kann am Anfang, in der Mitte oder am Ende eines Dateinamens plziert werden
?	das Fragezeichen ist Stellvertreter für <u>genau</u> ein Zeichen
[...]	eine Liste von Zeichen, die erlaubt sein können, werden in eckigen Klammern angegeben, das erste Zeichen muß allerdings kleiner sein bzw. weiter vorn im

Betriebssystem UNIX/Linux

	Alphabet liegen als das letzte Zeichen; Beispiele: [abc], [123], [0-9], [a-z], aber <u>nicht</u> gültige Substitution: [z-f], [5-1]
[!...]	ist das erste Zeichen nach der öffnenden Klammer ein Ausrufezeichen, so wird dadurch der Sinn der Übereinstimmungsprüfung umgekehrt; Beispiel: [!a-z] = Übereinstimmung mit allen Zeichen ausgenommen Kleinbuchstaben
[...]*[...]	mischt der Wertebereich mit anderen Zeichen; Beispiele: [a-np-z]* alle Kleinbuchstaben außer o [a-z]*[!0-9] alle Dateien, die mit einem Kleinbuchstaben beginnen und nicht mit einer Zahl enden

Quotings

" "	Keine Ersetzung der Metazeichen * ? [], jedoch Ersetzung von Shellvariablen (siehe unten) und Ersetzung durch die Ergebnisse von Kommandos (Backquote). Auch \ funktioniert weiterhin. Dazu ein Beispiel: echo Der * wird hier durch alle Dateinamen ersetzt echo "Der * wird hier nicht ersetzt"
' '	Das einfache Anführungszeichen unterdrückt jede Substitution. Zum Beispiel: echo 'Weder * noch `pwd` werden ersetzt'
` `	Zwischen Backquote (Accent Grave) gesetzte Kommandos werden ausgeführt und das Ergebnis wird dann als Parameter übergeben (d. h. die Ausgabe des Kommandos landet als Parameter in der Kommandozeile). Dabei werden Zeilenwechsel zu Leerzeichen. Braucht dieses Kommando Parameter, tritt die normale Parameterersetzung in Kraft. Zum Beispiel: echo "Aktuelles Verzeichnis: `pwd`" Weil die verschiedenen Quotes manchmal schwer zu unterscheiden sind, wurde bei der <i>bash</i> eine weitere Möglichkeit eingeführt. Statt in Backquotes wird die Kommandofolge in \$(...) eingeschlossen., z. B.: echo "Aktuelles Verzeichnis: \$(pwd)"

Eingabeumleitung

<	Umlenken des Standardeingabe
>	Umlenken der Standardausgabe zum Erzeugen / Überschreiben einer Datei
>>	Umlenken der Standardausgabe zum Erzeugen / Anhängen einer Datei
<<	Umlenken der Standardeingabe vom folgenden Text
n<	öffne für Eingabe von Datei-Deskriptor n
n>	öffne für Ausgabe von Datei-Deskriptor n (Überschreiben)
n>>	öffne für Ausgabe von Datei-Deskriptor n (Anhängen)
n<<	öffne für Eingabe von Datei-Deskriptor n des folgenden Textes
<&n	verbinde Standardeingabe mit offener Datei in Datei-Deskriptor n

<&-	schlieÙe Standardeingabe
>&n	verbinde Standardausgabe mit offener Datei in Datei-Deskriptor n
>&-	schlieÙe Standardausgabe
n<&m	verbinde für die Eingabe Datei-Deskriptor n mit Datei-Deskriptor m
n<&-	schlieÙe Datei in Datei-Deskriptor n
n>&m	verbinde für die Ausgabe Datei-Deskriptor n mit Datei-Deskriptor m
n>&-	schlieÙe Datei in Datei-Deskriptor n
<i>Befehl</i> 2> <i>file</i>	alle Fehlermeldungen, die normalerweise zur Standardfehlerausgabe gelangen würden, werden jetzt in die Datei <i>file</i> umgeleitet

pipes

	die Pipe () verbindet zwei Unix-Befehle miteinander, die Ausgabe des einen Befehls ist die Eingabe für den anderen Befehl; dazu muß es zwischen den betreffenden Befehlen plaziert werden; Beispiel: who more, seitenweises Auflisten der eingeloggten user; who wc -l, erstellt eine detaillierte Liste der eingeloggten user
--	--

Ausdrücke beim test-Kommando

Eigenschaften von Dateien

Ausdruck	Bedeutung
-e <datei >	datei existiert
-r <datei >	datei existiert und Leserecht
-w <datei>	datei existiert und Schreibrecht
-x <datei>	datei existiert und Ausführungsrecht
-f <datei>	datei existiert und ist einfache Datei
-d <datei>	datei existiert und ist Verzeichnis
-h <datei>	datei existiert und ist symbolisches Link
-c <datei>	datei existiert und ist zeichenor. Gerät
-b <datei>	datei existiert und ist blockor. Gerät
-p <datei>	datei existiert und ist benannte Pipe
-u <datei>	datei existiert und für Eigentümer s-Bit gesetzt
-g <datei>	datei existiert und für Gruppe s-Bit gesetzt
-k <datei>	datei existiert und t- oder sticky-Bit gesetzt
-s <datei>	datei existiert und ist nicht leer
-L <datei>	datei ist symbolisches Link
-t <dateikennzahl>	dateikennzahl ist einem Terminal zugeordnet

Vergleiche und logische Verknüpfungen

Vergleich von Zeichenketten	
Ausdruck	Bedeutung
-n <String>	wahr, wenn String nicht leer
-z <String>	wahr, wenn String leer ist
<String1> = <String2>	wahr, wenn die Zeichenketten gleich sind
<String1> != <String2>	wahr, wenn Zeichenketten verschieden sind
Algebraische Vergleiche ganzer Zahlen	
Operator	Bedeutung
-eq	equal - gleich
-ne	not equal - ungleich
-ge	greater than or equal - größer gleich
-gt	greater than - größer
-le	less than or equal - kleiner gleich
-lt	less than - kleiner
Logische Verknüpfung zweier Argumente	
UND	<bedingung1> -a <bedingung2>
ODER	<bedingung1> -o <bedingung2>
Klammern	\(<ausdruck> \)
Negation	! <ausdruck>

sed (stream editor)

sed '5d'	lösche Zeile 5
sed '/[Tt]est/d'	löscht alle Zeilen, die Test oder test beinhalten
sed -n '20,25p' text	gibt nur Zeile 20 bis 25 aus
sed '1,10s/unix/UNIX/g' intro	macht aus unix UNIX, egal an welcher Stelle der ersten 10 Zeilen von intro es auftaucht
sed '/jan/s/-1/-5'	wechselt das erste -1 in -5 in allen Zeilen, die jan beinhalten
sed 's/.../' data	löscht die ersten drei Zeichen jeder Zeile von data
sed 's/...\$/' data	löscht die letzten drei Zeichen jeder Zeile von data
sed -n '1' text	gibt alle Zeilen von text aus, wobei alle nicht druckbaren Zeichen als \nn (nn ist der oktale Wert des Zeichens) und alle Tabzeichen als > ausgegeben werden

tr

tr char1 char2	wird benutzt, um Zeichen <i>char1</i> aus der Standardeingabe zu <i>char2</i> übersetzen
----------------	--

tr : '\nnn'	ersetzt den Doppelpunkt durch das Zeichen des entsprechenden oktalen Wertes nn
tr 'X' 'x'	ersetze alle großen X durch kleine x
tr '()' '{}'	ersetze alle runden Klammern auf durch geschweifte Klammern auf und alle runden Klammern zu durch geschweifte Klammern zu
tr '[a-z]' '[A-Z]'	ersetze alle Kleinbuchstaben durch Großbuchstaben
tr '[A-Z]' '[N-Z]'	ersetze alle großen A-M durch N-Z und alle N-Z durch A-M
tr '\11' ''	ersetze alle Tabzeichen durch ein Leerzeichen
tr -s '' ''	ersetze mehrere aufeinanderfolgende Leerzeichen durch ein einzelnes Leerzeichen
tr -d '\14'	lösche alle Seitenvorschubzeichen (oktal 14)
tr -d '[0-9]'	löscht alle Ziffern

10.4



[Zum Inhaltsverzeichnis](#)



[Zum nächsten Abschnitt](#)

Copyright © FH München, FB 04, Prof. Jürgen Plate



Vorlesung "UNIX"

von Prof. Jürgen Plate

11 Literatur und Anhang

11.1 Literatur

Zu UNIX und Linux gibt es im Web unzählige Links. Deshalb hier nur ein paar, die gerade für Anfänger wichtig sind:

Allgemeines

www.linux-ag.de/linux/LHB/
Das Linux-Handbuch online.

<http://www.selflinux.org>
Deutsche Linux-Dokumentation.

A HREF="http://www.elearnit.de/live_cds/elpicx/index.html">Lern-DVD für die LPI-Prüfung

http://www.oreilly.de/german/freebooks/rlinux3ger/linux_wegSIX.html
Der Linux Wegweiser zur Installation & Konfiguration (OnLine Buch)

<http://ldp.uni-frankfurt.de/>
Linux Documentation Project (deutscher Mirror)

<http://www.rz.uni-hohenheim.de/~feiler/uni/linux.html>
Mathias Feilers Linux Page

<http://www.fokus.gmd.de:80/Linux/Linux-doc.html>
Deutsche OnLine Linux Dokumentation (diverse Themen)

<http://www.tldp.org/>
The Linux Documentation Project

<http://www.linuxhaven.de/dlhp/>
Deutsches Linux-HOWTO-Projekt

<http://www.linuxfaq.de/>
Deutschsprachige Linux FAQ mit der Möglichkeit, sie durch eigene Artikel zu erweitern.

<http://www.linuxhilfen.org>
Katalogisierte Tips, Infos, Links und Workshops mit Volltextsuche zum Thema Linux

<http://www.adsl4linux.de>
ADSL unter Linux: Technik, Basiswissen, Hardware, Software, Installation und Konfiguration

<http://www.linux-tip.net>
Linux Tipps und Tricks, Workshops; Tips and Tricks

<http://3d-crew.com/unixtexte/utexte.html>
Links zu diversen Texten

<http://www.linux-ag.de/>, die Linux AG.

<http://www.fibel.org/>, die Linux Fibel.

Linux Distributoren

http://de.wikipedia.org/wiki/Liste_von_Linux-Distributionen Wikipedia: Liste von Linux-Distributionen

<http://www.knoppix.de/>

Knoppix ist die ideale Lösung für Leute, die sich ohne Linux in der Tasche nicht wohl fühlen oder das PC-Unix ohne Installationsstress testen möchten. Im Rahmen eines LinuxTag-Projekts hat Klaus Knopper eine vollständige Linux-Distribution auf Basis von Debian Woody mit Kemei 2.4 auf einer bootfähigen CD-ROM untergebracht. Knoppix läuft komplett von CD und fasst die Festplatten im System nicht mal an - ideal, um sich ohne Installation einen schnellen Linux-Eindruck zu verschaffen oder auf fremden Rechnern eine vertraute Arbeitsumgebung zu schaffen. Beim Booten erkennt Knoppix die eingebaute Hardware erstaunlich gut. Im LAN entdeckt Knoppix einen vorhandenen DHCP-Server und nimmt das Netz selbstständig in Betrieb. Eine DSL-Verbindung ließ sich in einer Minute konfigurieren. Um die Einstellungen nicht nach jedem Booten vornehmen zu müssen, kann man Konfigurationsdateien auf Diskette sichern. Auf vorhandene Festplatten hat der Anwender nur Lesezugriff. Schaltet man den Schreibschutz aus, dient Knoppix dank Systemrettungstools als Recovery-System. Die CD enthält über 2000 Programme, darunter aktuelle Versionen von OpenOffice und Gimp. Beim Start lassen sich die Programme zwar mehr Zeit als bei einem auf Festplatte installierten System, doch dann lässt sich zügig arbeiten. Knoppix unterliegt der GPL; es ist daher möglich, sich eine individuelle CD zusammenzustellen, indem man Dateien hinzufügt oder modifiziert. Die jeweils aktuelle Version lässt sich als ISO-Image downloaden oder auf CD für rund fünf Euro beziehen.

Wichtiger Hinweis: Wenn Sie die Grundeinstellungen von Knoppix für Ihren Computer ändern, wählen Sie anschliessend im Menü "Knoppix" den Punkt "Einstellungen speichern", um die Info auf Diskette zu speichern. Beim späteren Start von Knoppix stecken Sie die Diskette ins Floppy-Laufwerk und geben beim Boot-Prompt

```
boot: knoppix floppyconfig
```

ein.

Linux-Feeling unter Windows

<http://sources.redhat.com/cygwin/>

Cygwin ist ein UNIX-Environment für Windows. Es besteht aus zwei Teilen:

- Eine DLL (cygwin1.dll) die als UNIX-Emulations-Schicht arbeitet und die wichtigsten Eigenschaften einer UNIX-API bereitstellt.
- Eine riesige Sammlung von Programmen, die das Look als Feel von UNIX/Linux in einer DOS-Box vermittelt. Neben Shell, VI, AWK, Perl und dem C-Compiler sind alle wichtigen Kommandos vorhanden - bis hin zu X.

Wer auf Unix unter Windows nicht verzichten möchte, kann mit Cygwin eine solche Umgebung unter Windows installieren. Neben den bekannten Unix-Kommandos stehen Ihnen auch einige Portierungen bekannter Unix-/Linux-Programme zur Verfügung. Ebenfalls enthalten ist die GNU-Compiler-Collection (gcc), mit der Sie auch Ihre eigenen Programme erstellen oder als Quelltext vorliegende Programme unter Cygwin installieren können.

Eine Auswahl von Programmen finden Sie auch hier. Entpacken Sie zunächst das Archiv [CYGWIN.ZIP \(175 MByte\)](#) in ein beliebiges Verzeichnis auf Ihrem Rechner. Starten Sie dort die Anwendung `SETUP.EXE`, und wählen Sie im folgenden Fenster die Option "Install from Local Directory". Danach wählen Sie das Verzeichnis, in das Sie Cygwin installieren wollen, und

Betriebssystem UNIX/Linux

anschließend das Verzeichnis, in dem Sie die Dateien entpackt haben. Dann erscheint ein Fenster mit einer Paketauswahl, in dem Sie einzelne Pakete an- oder abwählen können. Windows-Partitionen werden im versteckten Verzeichnis "cygdrive" gemountet, die Sie mit dem Befehl "mount" auslesen können.

Aus der Cygwin-Shell können Sie ebenfalls sämtliche Windows-Programme per Befehlszeile starten, etwa Notepad mit dem Befehl `/cygdrive/C/windows/notepad.exe`

In Cygwin ist auch eine komplette Portierung des unter Linux und Unix bekannten Xfree enthalten.

Damit starten Sie eine grafische Bedienungsführung, in der Sie die mitgelieferten

X-Window-Programme verwenden können. Mit Xfree können Sie Ihren Windows-Rechner außerdem als Client nutzen, um sich damit auf einem Linux-Server einzuloggen. Anschließend stehen Ihnen sämtliche grafischen Anwendungen dieses Linux-Rechners zur Verfügung. Auf dem Linux-Rechner kommentieren Sie die Zeile `DisplayManager.requestPort:0` in der Datei

`/etc/X11/xdm-config` aus. Ferner fügen Sie in der Datei

`/etc/opt/kde2/share/config/kdm/kdmrc` folgenden Abschnitt hinzu, falls dieser nicht vorhanden sein sollte:

```
Enable=true
xaccess=/var/X11R6/lib/xdm/Xaccess
Willing=/var/X11R6/lib/xdm/Zwilling
Port=177
```

Falls diese Einträge schon vorhanden sind, müssen Sie lediglich prüfen, ob der Eintrag "Enable" auf "true" gesetzt ist. Mit der Befehlszeile

```
./XWin.exe -broadcast
```

starten Sie den X-Client von Ihrem Windows-Rechner aus.

Linux-Ressourcen

[Linux Documentation Project](#) [Das Linux-Handbuch](#) [How To Ask Questions The Smart Way](#)

[FAQ-Archiv](#) <http://www.linux.org> (Linux.Org)

<http://www.sourceforge.org> (Sourceforge)

<http://www.slashdot.org> (Slashdot)

<http://www.freshmeat.org> (Freshmeat)

<http://www.linuxmall.com> (Linux Mall)

<http://www.li.org> (Linux International)

<http://www.opensource.org> (Opensource)

[Request for Comments \(RFC\)](#) [Mailinglisten-Archive](#) [The Linux-Kernel Mailing List](#) [FAQ Linux on](#)

[Laptops](#) [Open Source Embedded Linux Implementations](#) [LWN.net](#) [Linux Distributions List](#)

[Sicherheitsinfos zu Linux](#) [Tipps und Tricks zu Dotfiles](#) [Sound & MIDI Software For Linux](#) [Das](#)

[Streaming-Projekt "VideoLAN"](#) [UTF-8 and Unicode FAQ for Unix/Linux](#) [SuSE-Support-Datenbank](#)

[Support für Debian](#) [Support für Slackware](#) [Red-Hat-Support-Datenbank](#) [Support für Mandrake](#)

[Gentoo.org-Diskussionsforum](#)

Bücher

Die folgende Literaturliste bietet eine kleine Auswahl aus einer großen Zahl von Linux- und UNIX-Büchern:

Open Books

- [Learning Debian/GNU Linux](#)
- [Linux Device Drivers, 2nd Edition](#)
- [Linux Network Administrator's Guide, 2nd Edition](#)

- [Using Samba](#)
- [Linux Network Administrator's Guide](#)
- [Unix Text Processing \(Hayden Books\)](#)
- [Web Client Programming with Perl](#)
- [World Wide Web Journal](#)

Deutschsprachige Online-Bücher

- [Debian GNU/Linux Anwenderhandbuch](#)
- [Linux Gerätetreiber, 2. Auflage](#)
- [Linux Wegweiser zur Installation & Konfiguration, 3. Auflage](#)
- [Linux Wegweiser für Netzwerker, 2. Auflage](#)

Print

Linus Torvalds:

Just for Fun

Hanser-Verlag

Stefanie Teufel:

Jetzt lerne ich SuSE Linux

Verlag Markt & Technik

David Pitts, Bill Ball:

Linux Kompendium

Verlag Markt & Technik

Fuhs, Hasenbein:

Linux für Windows-Anwender

dpunkt Verlag

Helmut Herold:

Linux-UNIX Kurzreferenz

Verlag Addison Wesley

Michael Kofler:

Linux

Verlag Addison Wesley

Jochen Hein:

Linux Systemadministration

Verlag Addison Wesley

Henze, Hondel, Müller, Kirch:

Linux Anwenderhandbuch

Lunetix

Jessica Heckman:

Linux in a Nutshell

Verlag O'Reilly

Michael Renner:

Linux für Onliner

Verlag O'Reilly

Olaf Kirch:
Linux Netzwerkadministration
Verlag O'Reilly

Jessica Perry Hekman:
Linux in a Nutshell
Verlag O'Reilly

Rainer Krienke:
UNIX für Einsteiger
Hanser-Verlag

Peter Kuo:
UNIX Kompendium
Verlag Markt & Technik

Arne Burmeister:
Der Einstieg un UNIX
Hanser-Verlag

Levine/Young:
UNIX für Anfänger
iwt-Verlag

Abrahams/Larson:
UNIX for the Impatient
Verlag Addison-Wesley

Nemeth/Snyder/Seebass:
Systemadministration unter UNIX
Verlag Prentice-Hall

Aeleen Frisch:
Essential System Administration
Verlag O'Reilly

Helmut Herold:
UNIX-Grundlagen
Verlag Addison-Wesley

Helmut Herold:
UNIX-Shells
Verlag Addison-Wesley

Rainer Krienke:
UNIX Shell-Programmierung
Hanser-Verlag

Rosen/Rosinski/Faber:
UNIX System V Rel. 4
tewi-Verlag

Boss/Reimann:
UNIX System V

bhv Verlag

Garfinkel/Spafford:

Practical UNIX Security

Verlag O'Reilly

Zu einzelnen Spezialthemen (TCP/IP, DNS, sendmail, termcap, Internet-Server, etc.) gibt es eine breite Buchpalette vom Verlag O'Reilly. Die Bücher sind teilweise in deutscher Sprache, teils im englischen Original lieferbar.

11.2 Erfinder von UNIX und C geben zu: Alles Quatsch!

In einer Ankündigung, die die Computerindustrie verblüffte, haben Ken Thompson, Dennis Ritchie und Brian Kernighan zugegeben, daß das von ihnen geschaffene Betriebssystem Unix und die Programmiersprache C ein raffinierter Aprilscherz sind, der sich über 20 Jahre am Leben erhalten hat. Bei einem Vortrag vor dem letzten UnixWorld-Software-Entwicklungsforum enthüllte Thompson:

"1969 hatte AT&T gerade die Arbeit am GE/Honeywell/AT&T-Multics-Projekt beendet. Brian und ich experimentierten zu diesem Zeitpunkt mit einer frühen Pascal-Version von Professor Niklaus Wirth vom ETH-Laboratorium in der Schweiz und waren beeindruckt von seiner Einfachheit und Mächtigkeit. Dennis hatte gerade "Der Herr der Klinge" gelesen, eine spöttische Parodie auf Tolkiens Trilogie "Der Herr der Ringe". Im Übermut beschlossen wir, Parodien zur Multics-Umgebung und zu Pascal zu verfassen. Dennis und ich waren für die Betriebssystemumgebung verantwortlich. Wir sahen uns Multics an und entwarfen ein neues System, das so komplex und kryptisch wie möglich sein sollte, um die Frustration der gelegentlichen User zu maximieren. Wir nannten es "Unix" in Anspielung auf "Multics" und fanden es auch nicht gewagter als andere Verballhornungen. Danach entwickelten Dennis und Brian eine wirklich perverse Pascal-Version namens "A". Als wir bemerkten, daß einige Leute tatsächlich versuchten, in "A" zu programmieren, fügten wir schnell einige zusätzliche Fallstricke hinzu und nannten es "B", "BCPL" und schließlich "C". Wir hörten damit auf, als wir eine saubere Uebersetzung der folgenden Konstruktion erhielten:

```
for (;P ("\n"), R--;P ("|"))
for (e=C;e--;P ("_" + (*u++/8) %2))
P ("|" + (*u/4) %2)
```

Der Gedanke, daß moderne Programmierer eine Sprache benutzen würden, die solch eine Anweisung zuließ, lag jenseits unseres Vorstellungsvermögens. Wir dachten allerdings daran, alles den Sowjets zu verkaufen, um ihren Computerfortschritt 20 Jahre und mehr zu behindern. Unsere Ueberraschung war groß, als dann AT&T und andere US-Unternehmen tatsächlich begannen, Unix und C zu verwenden! Sie haben 20 weitere Jahre gebraucht, genügend Erfahrungen zu sammeln, um einige bedeutungslose Programme in C zu entwickeln, und das mit einer Parodie auf die Technik der 60er Jahre! Dennoch sind wir beeindruckt von der Hartnäckigkeit (falls nicht doch Gemeinsinn) des gewöhnlichen Unix- und C-Anwenders. Jedenfalls haben Brian, Dennis und ich in den letzten Jahren nur in Pascal und einem Apple Macintosh programmiert und wir fühlten uns echt schuldig an dem Chaos, der Verwirrung und dem wirklich schlechten Programmierstil, der von unserem verückten Einfall vor so langer Zeit ausging."

Namhafte Unix- und C-Anbieter und Benutzer, einschließlich AT&T, Microsoft, Hewlett-Packard, GTE, NCR und DEC haben vorläufig jede Stellungnahme abgelehnt. Borland International, ein führender Anbieter von Pascal- und C-Werkzeugen, einschließlich der populären Turbo Pascal, Turbo C und Turbo C++, meinte, sie hätten diesen Verdacht schon seit Jahren gehegt und würden nun dazu übergehen, ihre Pascal-Produkte zu verbessern und weitere Bemühungen um die C-Entwicklung stoppen. Ein IBM-Sprecher brach in unkontrolliertes Gelächter aus.

(Quelle: Bernhard L. Hayes, NetNews-Gruppe)

11.3 The UNIX Acrony List

Compiled by Wolfram Roesler <wr@bara.oche.de>

Additional contributions by:

Marcel Waldvogel <Marcel.Waldvogel@nice.usergroup.ethz.ch>
 Martin P. Ibert <martini@heaven7.in-berlin.de>
 Oliver Laumann <net@cs.tu-berlin.de>
 Peter Funk <pf@artcom0.north.de>
 Volker Lausch <volli@cs.tu-berlin.de>
 Ulf Moeller <um@ulf.mali.sub.org>
 Stefan Stapelberg <stefan@rent-a-guru.de>
 Christopher J. Calabrese <cjc@ulysses.att.com>
 Chris Siebenmann <cks@hawkwind.uts.toronto.edu>
 Casper H.S. Dik <casper@fwi.uva.nl>
 Ken Keys <kkeys@ucsd.edu>
 Peter da Silva <peter@nmti.com>
 Michael P Urban <urban@sideshow.jpl.nasa.gov>
 Drew Sullivan <drew@lethe.hades.gts.org>

Last change: 01-Jun-93

All comments are welcome.

All mail comments are subject to be included in the list without further asking for permission. Please include a note if you object. Does not include names that are no abbreviations, like `basename' or `sort'. "?" indicates guesses. Types:

C command

B shell built-in

D directory

E shell/environment variable

S special file

O other

NAME	TYPE	MEANING
adb	C	algol _or_ absolute _or_ assembly _or_ advanced debugger
ar	C	archiver
as	C	assembler
awk	C	Aho, Weinberger, Kernighan (program authors)
bash	C	Bourne again shell ("born again shell")
bin	D	binaries
bsh	C	Bourne shell (program author)
bc	C	better calculator
c89	C	1989 Ansi-C compiler
cal	C	calender
cat	C	concatenate
cc	C	C compiler
cd	B	change directory
chgrp	C	change group
chmod	C	change mode
chown	C	change owner
ci	C	check in
cmp	C	compare
co	C	check out
cp	C	copy

Betriebssystem UNIX/Linux

cpio	C	copy (archive files) in and out
cpp	C	C pre-processor
cron	C	Chronos (greek god)
csch	C	C shell
...d	C	daemon (e.g. inetd = internet daemon)
dbx	C	extended debugger (?) <code>_or_</code> dbx (=extended decibel, noise reduction system)
dc	C	desk calculator
dd	C	(opinion 1) Dataset Definition (named after the OS/3x0 JCL DD command who's syntax it also ripped off as a joke)
dd	C	(opinion 2) copy and convert (called `dd' because `cc' is the C compiler)
dd	C	(opinions 3-n) data, device, disk, dump in various combinations
dev	D	devices
df	C	disk free
diff	C	difference
dirname	C	directory name
du	C	disk usage
ed	C	editor
egrep	C	extended grep
elm	C	electronic mail
emacs	C	editing macros Sometimes: Eight megabytes and constantly swapping <code>_or_</code> : Escape Meta Alternate Control Shift <code>_or_</code> : Emacs makes any computer slow
env	B	environment
eqn	C	equation
esac	B	case (reversed)
etc	D	et cetera
ex	C	extended editor (?)
expr	C	expression
fd	S	floppy disk
fd	D	file descriptors (as in /dev/fd)
fgrep	C	fixed-string grep
fi	B	if (reversed)
fmt	C	format
fsck	C	file system check
ftp	C	file transfer protocol
g...	C	GNU (e.g. gawk = GNU awk) (GNU = "GNU is not Unix")
getty	C	get tty
grep	C	global regular expression print (from the ed subcommand "g/RE/p" where RE is a regular expression)
hd	S	hard disk
id	C	identity
IFS	E	internal field seperators
inode	O	index (or indirection) node
irc	C	internet relay chat
jsh	C	job-control shell
kmem	S	kernel memory
ksh	C	Korn shell (program author)
LANG	E	language
ld	C	link editor <code>_or_</code> loader
lex	C	lexical analyser
lib	D	library
ln	C	link
lp	S	line printer
lpp	D	licensed program products
lpq	C	(display) line printer queue
lpr	C	line print
ls	C	list
mem	S	memory
mkfs	C	make file system
mv	C	move
mkdir	C	move directory
nawk	C	new awk
nfs	O	network file system <code>_or_</code> : nightmare file system
nm	C	names
nn	C	no news

Betriebssystem UNIX/Linux

nohup	C	no hang-up
nroff	C	new roff (roff = program's ancestor)
od	C	octal dump
passwd	C	password
pcc	C	portable C compiler
pg	C	pager
pr	C	prepare (for printing)
ps	C	process status
PS1	E	prompt string 1 (same for PS2 etc.)
pty	S	pseudo teletype
pwd	C	print work directory
qdaemon	C	queue daemon
r...	S	raw (e.g. rfd0 = raw floppy disk 0)
r...	C	remote (e.g. rsh = remote shell)
rm	C	remove
rmdir	C	remove directory
rmt	S	raw magnetic tape
rmt	C	remote magnetic tape
roff	C	run-off (similar program)
sdb	C	symbolic debugger
sed	C	stream editor
sh	C	(Bourne) shell
stty	C	set tty
su	C	superuser
tar	C	tape archive(r)
tbl	C	table
tcsh	C	Tenex C shell
tee	C	T pipe fitting (plumbing device)
telnet	C	telephone network
tex	C	tau epsilon chi
tmp	D	temporary
tr	C	translate
troff	C	typesetter roff
tsh	C	trusted shell
tty	S	teletype
TZ	E	time zone
rcp	C	remote copy
rcs	C	revision control system
rlogin	C	remote login
rm	C	remove
rmdir	C	remove directory
rsh	C	remote shell _or_ restricted shell (sometimes Rsh)
sccs	O	source code control system
termcap	D	terminal capability
terminfo	D	terminal information
u	D	user
ucb	D	University of California at Berkeley
uniq	C	unique
usr	D	user
uucp	C	unix-to-unix copy
vi	C	visual (from the ex subcommand "vi" that switches into visual mode)
wall	C	write all
wc	C	word count
xargs	C	extended arguments
yacc	C	yet another compiler compiler
yp	O	yellow pages

11.4 The ABCs of Unix

A is for Awk, which runs like a snail, and
B is for Biff, which reads all your mail.

C is for CC, as hackers recall, while
D is for DD, the command that does all.

E is for Emacs, which rebinds your keys, and
F is for Fsck, which rebuilds your trees.

G is for Grep, a clever detective, while
H is for Halt, which may seem defective.

I is for Indent, which rarely amuses, and
J is for Join, which nobody uses.

K is for Kill, which makes you the boss, while
L is for Lex, which is missing from DOS.

M is for More, from which Less was begot, and
N is for Nice, which it really is not.

O is for Od, which prints out things nice, while
P is for Passwd, which reads in strings twice.

Q is for Quota, a Berkeley-type fable, and
R is for Ranlib, for sorting ar [sic] table.

S is for Spell, which attempts to belittle, while
T is for True, which does very little.

U is for Uniq, which is used after Sort, and
V is for Vi, which is hard to abort.

W is for Whoami, which tells you your name, while
X is, well, X, of dubious fame.

Y is for Yes, which makes an impression, and
Z is for Zcat, which handles compression.

11.5 An amusing photo

Here's a publicity photo from about 1972, showing Ken Thompson and me (Dennis Ritchie) in front of a PDP-11 with two Teletype 33 terminals. In front, we have Ken (sitting) and me (standing), both with more luxuriant and darker hair than we have now.



From the right, the major items of equipment are

- At the far right, on the table, are what someone discerned was a VT01A storage-tube display (based on Tek 611) and a small keyboard for it. Slightly hard to make out.
- A main CPU cabinet, partly behind the table. The processor is a PDP-11/20; it must have been our second one, with the Digital Special Systems KS-11 memory management unit. The very first just said "PDP11," not "11/20." The arrays of distorted rectangles above it and in other cabinets are the labels on DECTape canisters.
- Another cabinet with DECTape drive at the top. Careful examination of the image by Steve Westin detects the top of the bezel of an 11/45 CPU just peeking above the TTY that Ken is typing at. The paper tape reader is above.
- A cabinet with another DECTape drive, probably also containing BA-11 extension boxes within.
- A cabinet with RK03 disk drives. These were made by Diablo (subsumed by Xerox) and OEMed to Digital. Digital later began manufacturing their own version (RK05).
- A cabinet probably containing RF11/RS11 controller and fixed-head disks.
- On top of the machine are what look like magtapes. A probable TU10 transport is barely visible just below Ken's chin, at least if you have the monitor brightness and contrast adjusted favorably.

11.6 Manpage BABY

BABY (1)

USER COMMANDS

BABY (1)

NAME

BABY - create new process from two parent processes

