Von DOS nach Linux HOWTO

von Guido Gonzato (guido@ibogfs.cineca.it) und Tilo Wenzel

v1.3, 20. November 1997

Dieses HOWTO ist für all die (bald ehemaligen?) DOS-Nutzer, die sich gerade entschlossen haben, den Sprung in die Welt von Linux, der freien UNIX-Variante zu wagen. Ausgehend von Ähnlichkeiten zwischen DOS und UNIX ist es der Zweck dieses Dokumentes, das Wissen des Lesers über DOS in die UNIX-Welt hinüberzubringen, damit er so schnell wie möglich auf dem neuen System produktiv werden kann.

Inhalt

1 Einführung

1.1 Ist Linux das richtige für mich?

Sie haben also vor, von DOS auf Linux zu wechseln? Gute Idee, aber Vorsicht: es ist vielleicht nicht das richtige System für Sie. Meiner Meinung nach gibt es so etwas wie *den besten Computer* oder *das beste Betriebssystem* nicht: Es hängt davon ab, was man damit machen will. Darum glaube ich, daß Linux nicht notwendig das beste System für alle Leute ist, selbst wenn es technisch vielen kommerziellen Betriebssystemen überlegen ist. Typische Nutzer, die von Linux profitieren werden, sind Leute aus dem technischen Bereich, die Dinge wie TeX, Internet, Programmierung u.ä. betreiben.

Weiterhin ist Linux sehr gut geeignet für einen Einsatz als eine der vielen Arten von Server, da hier viel Software bereits dabei ist, die auf anderen Systemen erst für viel Geld gekauft werden muß. Darüber hinaus hat Linux oft deutlich geringere Hardwareanforderungen bei besserer Leistung als andere kommerzielle Serverbetriebssysteme.

Wenn man hauptsächlich kommerzielle Software benutzen will, sollte man sich vorher vergewissern, daß entsprechendes auch vorhanden ist (es gibt z.B. einige Office Programme wie *Star Office*, welches für den privaten Gebrauch kostenlos ist, *Applixware* oder *Word Perfect*). Wer nicht die Bereitschaft mitbringt, auch einmal in die Dokumentation zu schauen und das eine oder andere Konfigurationsfile von Hand mit dem Texteditor zu bearbeiten, der sollte sich besser nach etwas anderem umschauen. Außerdem wird unter Linux sehr viel mit der Kommandozeile gearbeitet. Ein paar Englischkenntnisse sind auch sehr hilfreich.

Linux ist (im Moment) nicht so einfach zu benutzen wie z.B. Windows oder Mac. Man sollte daher bereit sein, in sein System etwas *Bastelarbeit* zu stecken. Trotzdem bin ich 100%'ig überzeugt, daß, wenn man diese Art von Nutzer ist, man absolut begeistert von Linux sein wird. Es hängt alles nur von einem selbst ab. Außerdem kann man ja jederzeit noch andere Betriebssysteme wie DOS, Win 3.x, Win95 u.a. gleichzeitig auf der Platte haben.

Voraussetzungen, die ich für dieses Dokument mache:

- die grundlegenden DOS Kommandos und Begriffe sind bekannt;
- Linux ist - evtl. mit X-Windows - korrekt auf dem PC installiert;
- die Shell - das Äquivalent für `COMMAND.COM` - ist `bash`;
- es ist klar, daß dieses HOWTO nur einen allerersten Einstieg geben kann. Für weitere Informationen kann man sich Matt Welsh's *Linux Installation and Getting Started* und/oder Larry Greenfield's *Linux User Guide* zu Gemüte führen, die von

  `sunsite.unc.edu:/pub/Linux/docs/LDP`

bezogen werden können.

Dieses HOWTO ersetzt das alte *From DOS to Linux - Quick!* Mini-HOWTO. Man beachte, daß, wenn nicht anderweitig vermerkt, sich alle Informationen auf das *gute* alte DOS beziehen. Es gibt einen Abschnitt über Windows, aber man sollte immer im Gedächtnis behalten, daß Windows und Linux ganz verschieden sind, im Gegensatz zu DOS, das eine Art armer Verwandter von UNIX ist.

1.2 Es ist das Richtige. Was jetzt?

Sie haben gerade Linux und die Programme, die Sie brauchen, auf dem PC installiert. Sie haben sich einen Account eingerichtet (wenn nicht, sollte das *schleunigst* mit `adduser` nachgeholt werden) und Linux läuft. Name und Paßwort sind eingegeben, und Sie sitzen jetzt vor dem Bildschirm und fragen sich: "Was jetzt?".

Nicht verzweifeln. Sie sind fast soweit, die Dinge auch unter Linux zu machen, die Sie von DOS her kennen und viele mehr. Wenn Sie jetzt unter DOS wären, würden Sie wahrscheinlich einige der folgenden Dinge tun:

- Programme aufrufen, Dateien erstellen, kopieren, ansehen, löschen, drucken, umbenennen;
- mit Ihren Verzeichnissen arbeiten z.B. mit `cd`, `md`, `rd`, oder `dir`;
- Floppies formatieren und Dateien von/auf diese kopieren;
- anpassen Ihrer `AUTOEXEC.BAT` und `CONFIG.SYS`;
- schreiben von eigenen .BAT Dateien, QBasic, Pascal und C Programmen (oder was auch immer);
- und das verbleibende 1%.

Sie werden erfreut sein, daß diese Dinge unter Linux ganz ähnlich funktionieren wie unter DOS. Der Durchschnittsnutzer verwendet unter DOS nur einen kleinen Teil der über 100 verfügbaren Kommandos, was unter Linux mehr oder weniger ähnlich aussieht.

Noch ein paar Hinweise, bevor's weitergeht:

Wie kommt man wieder heraus: Auf der Textkonsole CTRL-ALT-DEL drücken und warten, bis Linux alles Notwendige für das Beenden des Systems getan hat und meldet, daß alles erledigt ist (wer ganz sicher gehen will, wartet, bis der Computer wieder anfängt zu booten) und schaltet dann den Rechner aus. Unter X beendet man zunächst den X-Server mit CTRL-ALT-BACKSPACE oder mit einem entsprechenden Menüpunkt des Windowmanagers. Danach kann dann Linux mit CTRL-ALT-DEL wie oben beendet werden. *Niemals* den Rechner einfach ausschalten, denn es könnte sein, daß Linux ein paar Informationen noch nicht auf die Platte geschrieben hat und das Filesystem so beschädigt wird. Unter Win95 ist ein solches Vorgehen jetzt auch als "Systemabschluß" vorgeschrieben.

Im Gegensatz zu DOS hat Linux eingebaute Sicherheitsmechanismen, da es als Multiuserbetriebssystem konzipiert ist. Dateien und Verzeichnisse besitzen Zugriffsrechte, so daß nicht alle Dateien von allen Nutzern gelesen oder beschrieben werden können (Siehe Abschnitt 2.3). Die einzige Ausnahme ist der Nutzer mit dem Loginnamen `root`, der sämtliche Rechte besitzt und der Systemverwalter ist. Wenn man an seinem eigenen PC arbeitet, ist man dann auch gleichzeitig sein eigener `root`. Man sollte sich jedoch *grundsätzlich* nur dann als `root` im System anmelden, wenn die Dinge, die man erledigen will, dieses *unbedingt* erfordern. Dieser Mechanismus zur Trennung von verschiedenen Nutzern ist nicht mit dem Loginmechanismus von Win95 zu vergleichen. Dort dient er im Wesentlichen nur dazu, verschiedenen Nutzern verschiedene Konfigurationen ihrer Umgebung bereitzustellen. Viren, wild gewordenen Programmen und unvorsichtigen oder böswilligen Nutzern ist es nach wie vor möglich, wie unter DOS, das gesamte System mit allen Daten auf einen Streich ins Nirwana zu befördern.

Das Beste ist, man experimentiert einfach ein wenig herum und probiert die Sachen selber aus. Auch hier gilt: Nur als `root` arbeiten, wenn unbedingt notwendig. Hilfe kann man auf eine der folgenden Arten bekommen:

- Um Hilfe zu den internen Kommandos der Shell zu bekommen gibt man `help` ein;

- Um Hilfe zu einem bestimmten Kommando/Programm zu erhalten, gibt man `man kommando` ein. Das ruft die zu `kommando` gehörige Manualseite auf (die *man page*).
- `info kommando` ruft, falls verfügbar, die zu `kommando` gehörige info-Seite auf. Info ist ein auf Hypertext basierendes Dokumentationssystem. Am Anfang ist es vielleicht etwas gewöhnungsbedürftig.
- Desweiteren kann man auch `apropos kommando` oder `whatis kommando` benutzen.
- Mit q kann die Betrachtung wieder beendet werden.

Die Dokumentation ist im allgemeinen englisch, Teile sind aber auch schon ins Deutsche übersetzt worden.

Ein Großteil der Mächtigkeit und der Flexibilität von UNIX rührt vom Konzept der Umleitung und des Pipings her, welches hier weit leistungsfähiger ist als unter DOS. Einfache Kommandos können auf diese Art und Weise kombiniert werden, um komplexe Aufgaben zu erledigen. Man sollte davon regen Gebrauch machen.

Bezeichnungen: `<...>` bedeutet, daß etwas angegeben werden muß, während `[...]` bedeutet, daß die Angabe optional ist. Beispiel:

```
$ tar -tf <file.tar> [> Zieldatei]
```

`file.tar` muß angegeben werden, während die Umleitung in die Datei `Zieldatei` nicht zwingend ist.

Von jetzt ab steht *MSL* hier als Abkürzung für *Bitte die Manualseiten für weitere Informationen zu Rate ziehen*

1.3 Für Ungeduldige

Wollen Sie sofort loslegen? Dann eine kurze Übersicht:

DOS	Linux	Bemerkung
BACKUP	tar -Mcvf device dir/	v\"{o}llig anders
CD dirname\	cd dirname/	fast die gleiche Syntax
COPY file1 file2	cp file1 file2	dito
DEL file	rm file	Vorsicht - kein undelete
DELTREE dirname	rm -R dirname/	dito
DIR	ls	nicht ganz die gleiche Syntax
EDIT file	vi file	sehr gew\"{o}hnungsbed\"{u}rftig
	emacs file	etwas besser
	jstar file	etwa wie DOS' edit
FORMAT	fdformat	unterschiedliche Syntax
	mount, umount	
HELP command	man command	gleiches Schema
MD dirname	mkdir dirname/	fast die gleiche Syntax
MOVE file1 file2	mv file1 file2	dito
NUL	/dev/null	dito
PRINT file	lpr file	dito
PRN	/dev/lp0,	
	/dev/lp1	dito
RD dirname	rmdir dirname/	fast die gleiche Syntax
REN file1 file2	mv file1 file2	nicht f\"{u}r mehrere Dateien
RESTORE	tar -Mxpvf device	andere Syntax
TYPE file	less file	viel besser
WIN	startx	Dazwischen liegen Welten!

Wenn Sie mehr als nur eine Tabelle haben wollen, können Sie weiteres in den folgenden Abschnitten erfahren.

2 Dateien und Programme

2.1 Dateien: Grundlegende Begriffe

Die Struktur des Filesystems von Linux ist für den Benutzer nach außen hin der von DOS recht ähnlich. Mit Struktur des Filesystems ist hier die Anordnung von Verzeichnissen und der darin enthaltenen Dateien gemeint. Die Namen für Verzeichnisse und Dateien gehorchen bestimmten Regeln, Dateien werden in Verzeichnissen abgelegt, es gibt ausführbare Dateien und diese haben oft auch wie unter DOS Kommandozeilenparameter. Darüber hinaus kann man auch Platzhalter, Umlenkung und Piping verwenden. Es gibt jedoch gegenüber DOS ein paar Unterschiede:

Unter DOS sind die Dateinamen in der 8.3-Form, d.h. wie etwa `NICHGENG.TXT`. Unter Linux sind die Regeln für Dateinamen bei Benutzung des UMSDOS- oder EXT2-Filesystems wesentlich liberaler, vergleichbar etwa mit Win95. Es können bis zu 255 Zeichen verwandt werden, und der Punkt kann beliebig oft auftreten. Ein Beispiel für einen Dateinamen unter Linux ist z.B. `Das_ist.ein.SEHR_langer.dateiname`. Man beachte, daß hier sowohl große als auch kleine Buchstaben verwandt werden, denn es wird auch hier zwischen großen und kleinen Buchstaben im Gegensatz zu DOS unterschieden. Das heißt, `FILENAME.tar.gz` und `filename.tar.gz` sind zwei unterschiedliche Dateien. So ist `ls` ein Kommando, `LS` dagegen wird höchst wahrscheinlich nur eine Fehlermeldung bringen.

Windows 95 Nutzer werden wahrscheinlich Leerzeichen innerhalb der Dateinamen verwenden wollen. Wenn ein Name solche enthält (was nicht sehr empfehlenswert ist), muß er immer wenn er verwendet wird in Anführungszeichen eingeschlossen werden. Beispielsweise:

```
$ # das folgende Kommando legt ein Verzeichnis namens "Meine alten Dateien"
$ mkdir "Meine alten Dateien"
$ ls
Meine alten Dateien     bin     tmp
```

Einige Zeichen können zwar verwendet werden, sollten es aber nicht, wie zum Beispiel `!*$&`. Ich will nicht erklären wie man das macht, denn es ist wirklich keine gute Idee. Diese Zeichen haben alle eine spezielle Bedeutung für die Shell, und wenn sie in Dateinamen auftauchen, kann das unerwartete Ergebnisse zeitigen wenn man nicht genau aufpaßt.

Für Programme gibt es keine zwangsweisen Erweiterungen wie .exe .com oder .bat für Batchdateien. Ausführbare Programme werden beim Auflisten mit dem Kommando `ls -F` mit einem Sternchen '`*`' am Ende des Namens versehen, zum Beispiel:

```
$ ls -F
Brief_an_Joe  cindy.jpg  cjpg*  Ein_Verzeichnis/  mein_1._script* alt~
```

Hier sind die Dateien `cjpg` und `mein_1._script` ausführbar, d.h. Programme. Man beachte, daß der Stern *nicht* Teil der Dateinamen ist, sondern nur zur Kennzeichnung als ausführbar beim Auflisten dient. Unter DOS enden Backup-Dateien üblicherweise auf .BAK, während sie unter Linux im allgemeinen mit einer Tilde '`~`' enden. Wenn Linux entsprechend konfiguriert ist, können Auflistungen von Verzeichnissen auch farbig gekennzeichnet werden. Namen von Programmen könnten z.B. rot ausgegeben werden, Verzeichnisse blau und Bilder lila. Dateien, deren Name mit einem Punkt beginnt, werden als versteckte Dateien behandelt. Sie werden bei einem normalen Auflisten mit `ls` nicht angezeigt. Die Datei `.Ich.bin.eine.versteckte.Datei` wird also normalerweise bei einer Auflistung ignoriert, erst ein `ls -a` bringt sie zum Vorschein.

Optionen und Schalter werden unter DOS als `/schalter` angegeben, unter Linux mit `-schalter` oder `--schalter`. Beispiel: `dir /s` wird zu `ls -R`. Man beachte, daß viele DOS-Programme Schalter nach UNIX-Art verwenden, so z.B. `PKZIP` oder `ARJ`.

Wer will, kann jetzt mit dem Abschnitt 2.4 weitermachen, aber es ist nützlich, sich vorher noch kurz ein paar Dinge anzuschauen, die es nicht in DOS oder Windows gibt.

2.2 Links

UNIX hat noch einen weiteren Dateityp, der bei DOS nicht existiert. Es ist der Link. Ein Link ist eigentlich keine richtige Datei, sondern nur eine Art Verweis auf eine andere, bereits existierende Datei oder Verzeichnis. Es gibt zwei Typen von Links, den Hardlink und den symbolischen Link. Es soll hier nicht weiter auf den Unterschied zwischen beiden eingegangen werden, da sie sich äußerlich für den Nutzer kaum unterscheiden. Heutzutage werden üblicherweise vorwiegend symbolische Links eingesetzt, da sie etwas flexibler sind. Am ehesten vergleichbar sind symbolische Links mit den Win95 Shortcuts. Beispiele für symbolische Links sind z.B. das Verzeichnis `/usr/X11` welches ein Link auf `/usr/X11R6` ist und `/dev/modem`, welches entweder auf `/dev/cua0` oder `/dev/cua1` zeigt.

Um einen symbolischen Link anzulegen gibt man ein:

```
$ ln -s <Datei_oder_Verzeichnis> <Linkname>
```

Beispiele:

```
$ ln -s /usr/doc/g77/DOC g77manual.txt
```

Jetzt kann man sich auf `g77manual.txt` beziehen anstelle von `/usr/doc/g77/DOC`. Links werden bei der Auflistung eines Verzeichnisses wie folgt angezeigt:

```
$ ls -F
g77manual.txt@
$ ls -l
(verschiedene Angaben zur Datei ...)   g77manual.txt -> /usr/doc/g77/DOC
```

2.3 Rechte und Eigentümer

Dos-Dateien haben folgende Attribute: A (archivieren), H (versteckt), R (nur lesbar) und S (System). Nur H und R sind unter Linux sinnvoll: H sind Dateien die mit einem Punkt anfangen, und R wird später besprochen.

Unter UNIX besitzt jede Datei *Rechte* und einen Eigentümer, der wiederum zu einer *Gruppe* gehört. Hier ein Beispiel:

```
$ ls -l /bin/ls
-rwxr-xr-x  1  root  bin  27281 Aug 15 1995 /bin/ls*
```

Das erste Feld enthält die Rechte der Datei `/bin/ls`, die `root` gehört, sowie der Gruppe `bin`. Die Zeichenfolge `-rwxr-xr-x` bedeutet von links nach rechts:

`-` ist der Dateityp (- = normale Datei, d = Verzeichnis, l = Link, usw.); `rwx` sind die Rechte für den Eigentümer der Datei (lesen,schreiben,ausführen); `r-x` sind die Rechte für die Gruppe des Eigentümers (lesen,ausführen) - auf das Prinzip von Gruppen soll hier nicht weiter eingegangen werden, man kann als Anfänger auch sehr gut ohne das auskommen ;-) - ; `r-x` sind die Rechte für den Rest der Nutzer (lesen,ausführen). Für die Leute, bei denen Englisch nicht die zweite Muttersprache ist, hier die englische Bedeutung der Kürzel: r - Read, w - Write, x - eXecute, sowie beim Dateityp d für Directory. Der Rest der Ausgabe soll hier nicht weiter behandelt werden, wer Näheres wissen will kann in den einschlägigen Büchern nachlesen (z.B. im Buch von Matt Welsh).

Im Falle unseres `/bin/ls` kann man also die Datei nicht verändern, es sei denn, man ist `root`: alle anderen haben nicht die notwendigen Schreibrechte. Das Kommando, um die Rechte einer Datei zu ändern, ist:

```
$ chmod <werXrecht> <datei>
```

wobei *wer* für den steht, dessen Rechte geändert werden, also entweder `u` (user, der Eigentümer), `g` (group, die Gruppe), `o` (other,der Rest) oder `a` (all, alle Nutzer), *X* ist entweder `+`, `-` oder `=`, je nachdem, ob das Recht hinzugefügt oder weggenommen wird, bzw. auf den angegebenen Wert gesetzt wird, und *recht* ist das Recht, was geändert wird, also entweder `r` (read), `w` (write), oder `x` (execute).

Beispiele:

```
$ chmod u+x file
```

setzt die Ausführungsrechte für den Eigentümer.

```
$ chmod go-wx file
```

nimmt das Schreibrecht und das Ausführungsrecht für alle außer den Eigentümer weg.

```
$ chmod ugo+rwx file
```

setzt für alle Schreib-, Lese- und Ausführungsrechte. Man kann hier auch die Folge *ugo* einfach durch *a* ersetzen.

```
$ chmod u+s file
```

dieses setzt das sogenannte (oben nicht erwähnte) *setuid* oder *suid* Recht (meistens setuid-Bit genannt). Damit wird eine Datei, wenn sie ausführbar ist, automatisch beim Aufruf mit den Rechten des *Eigentümers* ausgeführt und nicht wie sonst üblich mit den Rechten des Aufrufers. Gehört die Datei z.B. `root`, wird sie mit `root`-Rechten ausgeführt und hat somit vollen Zugriff auf das System (und kann bei einem Fehler auch entsprechenden Schaden anrichten). Also Vorsicht mit dem Setzen des suid-Bits.

Ein kürzerer Weg zur Angabe von Rechten ist die Angabe von Zahlen: `rwxr-xr-x` kann z.B. als 755 angegeben werden. Dabei entspricht jeder Buchstabe einem Bit: `---` ist 0, `--x` ist 1, `-w-` ist 2, `-wx` ist 3... . Es ist zu Beginn etwas gewöhnungsbedürftig, aber nach und nach bekommt man Routine mit diesen Werten.

Normalerweise darf nur der die Rechte einer Datei ändern, der auch ihr Eigentümer ist. `Root` jedoch als sogenannter Superuser kann die Rechte aller Dateien ändern. Zum Kommando `chmod` gibt es noch weiteres zu sagen, das aber nicht in diesen Rahmen paßt. Also — MSL.

2.4 Übertragen von Kommandos von DOS nach Linux

Auf der linken Seite ist das DOS Kommando aufgeführt, auf der Rechten das Linux-Pendent

```
COPY:           cp
DEL:            rm
MOVE:           mv
REN:            mv
TYPE:           more, less, cat
```

Umleitungs- und Pipingoperatoren: `< > >> |`

Platzhalter: `* ?`

`nul: /dev/null`

`prn, lpt1: /dev/lp0 or /dev/lp1; lpr`

Beispiele:

```
DOS                                     Linux
```

```
C:\GUIDO>copy joe.txt joe.doc          $ cp joe.txt joe.doc
C:\GUIDO>copy *.* total                $ cat * > total
C:\GUIDO>copy fractals.doc prn         $ lpr fractals.doc
C:\GUIDO>del temp                      $ rm temp
C:\GUIDO>del *.bak                     $ rm *~
C:\GUIDO>move paper.txt tmp\           $ mv paper.txt tmp/
C:\GUIDO>ren paper.txt paper.asc       $ mv paper.txt paper.asc
C:\GUIDO>print letter.txt              $ lpr letter.txt
C:\GUIDO>type letter.txt               $ more letter.txt
C:\GUIDO>type letter.txt               $ less letter.txt
C:\GUIDO>type letter.txt > nul         $ cat letter.txt > /dev/null
nicht vorhanden                        $ more *.txt *.asc
nicht vorhanden                        $ cat section*.txt | less
```

Bemerkungen:

Der * ist unter Linux intelligenter: * paßt auf alle Dateien, außer auf die versteckten, *.* paßt nur auf solche Dateien, die ein '.' in der Mitte oder am Ende haben, p*r paßt auf 'peter' und 'pfeiffer' (mit 3 f ;-), *c* paßt auf 'picken', 'pack.txt', 'mac' und 'c' selbst.

Wenn man `more` benutzt, kann man mit `LEERTASTE` in der Datei weiterblättern, mit 'q' oder CTRL-C beendet man es. `less` ist etwas intuitiver und läßt einen mit den Cursortasten durch die Datei wandern. Manchmal ist `more` einfach nur ein Link auf `less`, so daß sich beide gleich verhalten.

Es gibt kein `UNDELETE`, also *zweimal überlegen* bevor man etwas löscht.

Zusätzlich zu den DOS-üblichen < > >> hat Linux noch 2> um Fehlermeldungen umzulenken (`stderr`). Darüber hinaus lenkt `2>&1` `stderr` nach `stdout` um und `1>&2` `stdout` nach `stderr`.

Linux hat noch einen anderen Platzhalter: das `[]`. Verwendung: `[abc]*` paßt z.B. auf alle Dateien, die auf a, b oder c beginnen; `*[I-N,1,2,3]` paßt auf alle Dateien, die mit I, J, K, L, M, N, 1, 2 oder 3 enden.

Es gibt standardmäßig kein DOS-ähnliches `RENAME`; d.h. `mv` *.xxx *.yyy funktioniert nicht. Es gibt jedoch ein Programm namens `mmv` (Multiple MoVe), das Analoges leistet. Es ist in vielen Distributionen bereits enthalten. Um die Shell daran zu hindern, den Platzhalter selber zu interpretieren, muß der erste Dateiname in Anführungszeichen eingeschlossen werden. Beispiel:

```
# mmv '*.xxx' #1.yyy
```

#1 wird hierbei durch den zum ersten (und in diesem Falle einzigen) Platzhalter in `*.xxx` passenden String ersetzt. Es gibt noch ein paar andere kleine Unterschiede zu `RENAME`, also hier wieder mal — MSL.

Man benutze `cp -i` und `mv -i` , um gewarnt zu werden, wenn eine Datei dadurch überschrieben würde;

2.5 Programme starten: Multitasking und Sessions

Um ein Programm auszuführen, gibt man einfach den Namen wie unter DOS ein. Falls das Verzeichnis (Abschnitt 3), in dem sich das Programm befindet, im Pfad `PATH` (Abschnitt 6.1) ist, wird das Programm starten. Unterschied zu DOS: ein Programm, das sich im aktuellen Verzeichnis befindet, wird *nicht* gefunden - es sei denn, das aktuelle Verzeichnis ist als '.' explizit im Pfad enthalten. Wenn nicht, hilft ein `./programm`. Hinweis: das aktuelle Verzeichnis ist unter UNIX/Linux oft am Ende des Pfades eingetragen (aus Gründen der Systemsicherheit), d.h. es werden erst alle anderen Verzeichnisse nach `programm` durchsucht und zum Schluß erst das aktuelle Verzeichnis. Wenn man z.B. ein kleines Prögrämmchen geschrieben hat und es `test` nennt, wird, wenn man es mit `test` aufruft und nicht mit `./test`, zuerst das UNIX-Kommando `test` selbigen Namens gefunden (oder die Shell-interne Funktion, je nach Shell) und ausgeführt und nicht das eigene Programm im aktuellen Verzeichnis. Das führt oft zu langem Grübeln, bis man endlich merkt, daß das falsche Programm aufgerufen wurde, denn `test` ohne Parameter gibt keinerlei Meldungen o.ä. aus. Auf diesen ”Trick” sind schon Generationen von Einsteigern hereingefallen und werden wahrscheinlich auch noch weitere Generationen hereinfallen.

Hier das Aussehen eines typischen Kommandos:

```
$ kommando -s1 -s2 ... -sn par1 par2 ... parn < input > output
```

wobei `-s1`, ..., `-sn` die Programmschalter sind und `par1`, ..., `parn` die Parameter. Der Rest sind die Umlenkungen, d.h. das Programm erhält seine Eingaben aus der Datei `input` und schreibt die Ausgaben in die Datei `output`. Es müssen natürlich nicht immer alle Teile enthalten sein. Mehrere Kommandos hintereinander können so eingegeben werden:

```
$ kommando1 ; kommando2 ; ... ; kommandon
```

Das ist alles, was man braucht, um ein Kommando aufzurufen. Es gibt jedoch darüber hinaus Möglichkeiten, die Linux zusätzlich zu den von DOS bekannten bietet. Einer der Gründe die für Linux sprechen ist es, daß es ein Betriebsystem mit Multitasking ist, d.h. es kann mehrere Programme (ab jetzt Prozesse genannt) gleichzeitig ausführen. Man kann einen Prozeß im Hintergrund starten und mit einem anderen weiterarbeiten. Darüber hinaus bietet Linux auch mehrere Sitzungen (Sessions) gleichzeitig an. Es ist so, als ob man an mehreren Rechnern arbeiten würde.

Um zu den Sessions 1..6 zu wechseln:

```
$ ALT-F1 ... ALT-F6
```

Wenn man gerade unter X-Windows ist, benutzt man statt dessen CTRL-ALT-Fn.

Um eine neue Session zu starten ohne die aktuelle zu verlassen (z.B. um als anderer Nutzer mit anderen Rechten weiterzuarbeiten):

```
$ su - <loginname>
```

Beispiel:

```
$ su - root
```

Dieses ist nützlich wenn man etwas erledigen muß, was normalerweise nur `root` darf, z.B. eine Floppy mounten (siehe 4).

Um eine Session zu beenden:

```
$ exit
```

Wenn es noch angehaltene Jobs gibt (siehe unten), wird man gewarnt.

Um einen Prozeß im Vordergrund zu starten:

```
$ progname [-schalter] [parameter] [< input] [> output]
```

Um einen Prozeß im Hintergrund zu starten, fügt man ein Kaufmanns-Und am Ende der Zeile hinzu: ’`&`’

```
$ progname [-schalter] [parameter] [< input] [> output] &
[1] 123
```

Die Shell gibt dem Prozeß eine Jobnummer (z.B. `[1]`; siehe unten) und eine PID (ProzessID), 123 in unserem Beispiel.

Um alle Prozesse auflisten zu lassen:

```
$ ps -a
```

Dieses gibt eine Liste aller gerade laufenden Prozesse aus.

Um einen Prozeß zu beenden:

```
$ kill <PID>
```

Dies ist nützlich, um einen Prozeß zu beenden, wenn man entweder nicht weiß, wie man das Programm "vorschriftsmäßig" beendet, oder diese eigentlich vorgesehene Methode nicht mehr funktioniert. Manchmal kann ein Prozeß nur noch durch folgendes Kommando beendet werden:

```
$ kill -9 <PID>
```

kill -9 beendet jeden Prozeß, da dieses Signal nicht vom Prozeß abgefangen werden kann, d.h. der Prozeß wird vom Betriebssystem zwangsweise rausgeschmissen.

Darüber hinaus erlaubt es die Shell, einen Prozeß zu stoppen oder zeitweise anzuhalten, einen laufenden Prozeß in den Hintergrund zu schicken, oder aus dem Hintergrund in den Vordergrund zu holen. In diesem Zusammenhang werden die Prozesse *Jobs* genannt.

Um sich die laufenden Jobs der aktuellen Shell anzuschauen:

```
$ jobs
```

hierbei werden Jobs durch ihre Nummer und nicht durch ihre PID gekennzeichnet.

Um einen im Vordergrund laufenden Prozeß anzuhalten (funktioniert nicht immer, das Programm kann das explizit verhindern):

```
$ CTRL-C
```

Um einen im Vordergrund laufenden Prozeß zeitweise anzuhalten (dito):

```
$ CTRL-Z
```

Um einen vorübergehend angehaltenen Prozeß in den Hintergrund zu schicken (dadurch wird er zu einem Job):

```
$ bg <job>
```

Um einen Job in den Vordergrund zu bringen:

```
$ fg <job>
```

Um einen Job zu beenden:

```
$ kill <%job>
```

wobei <job> 1, 2, 3, ... sein kann. Mit diesen Kommandos kann man Disketten formatieren, Archive komprimieren, Programme kompilieren, Berechnungen ausführen usw. und trotzdem noch einen benutzbaren Prompt haben. Man versuche das einmal mit DOS! Windows kennt zwar auch Multitasking, jedoch ist das System meistens mit einem Programm bereits so ausgelastet, daß es für ein sinnvolles Arbeiten mit einem weiteren kaum noch reicht.

2.6 Programme via Netz auf anderen Rechnern ausführen

Um ein Programm auf einem anderen Rechner auszuführen, dessen Adresse z.B. `remote.bigone.edu` ist, gibt man ein:

```
$ telnet remote.bigone.edu
```

nachdem man eingeloggt ist, kann man sein gewünschtes Programm starten. Natürlich muß man einen Account (Zugangsberechtigung) auf dieser Maschine haben. Wenn man X-Windows benutzt, kann man sogar ein X-Programm auf dem anderen Rechner starten und sich die Ausgabe auf den eigenen Rechner umlenken. Wenn `remote.bigone.edu` der entfernte Rechner ist und `local.linux.box.de` der eigene Linuxrechner, muß man folgendes machen, um ein Programm auf `remote.bigone.edu` zu starten und es von `local.linux.box.de` aus zu bedienen:

- X11 auf `local.linux.box.de` starten, ebenso ein xterm oder Äquivalentes. Dort gibt man ein:

  ```
  $ xhost +remote.bigone.edu
  $ telnet remote.bigone.edu
  ```

- nach dem einloggen auf dem entfernten Rechner:

  ```
  remote:$ DISPLAY=local.linux.box.de:0.0
  remote:$ progname &
  ```

  (anstelle von `DISPLAY...`, muß man evtl. `setenv DISPLAY local.linux.box.de:0.0` eingeben. Das hängt von der Shell auf dem entfernten Rechner ab.)

Und siehe da! `progname` startet jetzt auf `remote.bigone.edu` und wird auf dem eigenen lokalen Rechner angezeigt. Man sollte jedoch vorsichtig sein beim Versuch, dies über eine langsame Modemleitung o.ä. zu machen. Bei so geringer Transferrate dürfte ein sinnvolles Arbeiten kaum möglich sein. Dafür gibt es eine spezielle Abart des X Windows, das Low-Bandwidth X, das hier jedoch nicht näher besprochen werden soll.

3 Verzeichnisse

3.1 Vorbemerkungen

Wir haben bereits die Unterschiede zwischen Dateien unter DOS/Windows und Linux besprochen. Was Verzeichnisse angeht, ist das Wurzelverzeichnis unter DOS während es unter Linux / ist. Ebenso werden Verzeichnisse in Pfadnamen unter DOS durch ein voneinander getrennt, unter Linux durch ein /. Darüber hinaus gibt es unter Linux keine Laufwerke wie z.B. C: unter DOS. Wenn man mehrere Festplatten oder Partitionen hat, werden diese unter Linux zu einem einzigen Verzeichnisbaum zusammengefaßt. Beispiele für Dateipfade:

```
DOS:    C:\PAPERS\GEOLOGY\MID_EOC.TEX
Linux:  /home/guido/papers/geology/mid_eocene.tex
```

Wie üblich ist `..` das übergeordnete Verzeichnis und `.` das aktuelle. Man beachte, daß man nicht überall in jedes Verzeichnis mit `cd` wechseln kann, oder mit `rmdir` oder `mkdir` Verzeichnisse löschen bzw. anlegen kann, in Abhängigkeit davon, ob man die entsprechenden Rechte hat oder nicht. Jeder Nutzer befindet sich nach dem Einloggen in seinem eigenen Verzeichnis, üblicherweise als HOME-Verzeichnis bezeichnet. Auf meinem PC ist mein Home-Verzeichnis z.B. `/home/guido`.

3.2 Zugriffsrechte von Verzeichnissen

Verzeichnisse haben natürlich auch Zugriffsrechte wie Dateien. Die Rechte, die im Abschnitt 2.3 für Dateien besprochen wurden, existieren auch analog bei Verzeichnissen (user,group,other). Für ein Verzeichnis bedeutet `rx`, daß man in das Verzeichnis wechseln kann und sich den Inhalt auflisten lassen kann. `w` dagegen bedeutet, daß man eine Datei im Verzeichnis löschen kann oder eine neue anlegen. Man kann eine Datei in einem Verzeichnis, für das man Schreibrechte besitzt, auch dann löschen, wenn die Datei für einen eigentlich schreibgeschützt ist. Das System fragt zwar an, ob man die Datei wirklich löschen will, aber wenn man ja sagt verschwindet die Datei. Der Schreibschutz für Dateien bezieht sich nur auf das Verändern der Datei selber. Das Löschen einer Datei ist aber eine Änderung am Verzeichnis, denn die Datei selbst wird nicht verändert, sondern nur ihr Eintrag aus dem Verzeichnis entfernt.

Um z.B. zu verhindern, daß andere Nutzer im Verzeichnis `/home/guido/text` herumschnüffeln, gibt man ein:

```
$ chmod o-rwx /home/guido/text
```

3.3 Übertragen von Kommandos von DOS nach Linux

```
DIR:            ls, find, du
CD:             cd, pwd
MD:             mkdir
RD:             rmdir
DELTREE:        rm -R
MOVE:           mv
```

Beispiele:

```
DOS                                     Linux
________________________________________________________________

C:\GUIDO>dir                            $ ls
C:\GUIDO>dir file.txt                   $ ls file.txt
C:\GUIDO>dir *.h *.c                    $ ls *.h *.c
C:\GUIDO>dir/p                          $ ls | more
C:\GUIDO>dir/a                          $ ls -l
C:\GUIDO>dir *.tmp /s                   $ find / -name "*.tmp"
C:\GUIDO>cd                             $ pwd
nicht vorhanden - siehe Bemerkung       $ cd
        dito                            $ cd ~
        dito                            $ cd ~/temp
C:\GUIDO>cd \other                      $ cd /other
C:\GUIDO>cd ..\temp\trash               $ cd ../temp/trash
C:\GUIDO>md newprogs                    $ mkdir newprogs
C:\GUIDO>move prog ..                   $ mv prog ..
C:\GUIDO>md \progs\turbo                $ mkdir /progs/turbo
C:\GUIDO>deltree temp\trash             $ rm -R temp/trash
C:\GUIDO>rd newprogs                    $ rmdir newprogs
C:\GUIDO>rd \progs\turbo                $ rmdir /progs/turbo
```

Bemerkungen:

1. Bei der Benutzung von `rmdir` muß das zu löschende Verzeichnis leer sein. Will man ein Verzeichnis mitsamt seinem Inhalt auf einmal loswerden hilft ein `rm -R`. *VORSICHT!* Mit diesem Befehl kann man ganze Verzeichnisbäume auf Nimmerwiedersehen verschwinden lasen. Also zweimal überlegen.

2. Das Zeichen '~' ist eine Abkürzung für den Namen des eigenen Home-Verzeichnisses, wenn es am Anfang des Pfad/Dateinamens steht. Das Kommando `cd` oder `cd ~` bringen einen sofort ins eigene Home-Verzeichnis, egal wo man gerade ist. `cd ~/tmp` bringt einen demnach ins Verzeichnis `/Pfad_zum_Homeverzeichnis/tmp`.

3. `cd -` ist ein "undo" für das letzte `cd`.

4 Floppies, Harddisks und Ähnliches

4.1 Devices verwalten

Wenn man unter DOS ein `FORMAT A:` macht, geschehen drei Dinge: 1. Die Diskette wird physisch formatiert, d.h. es werden Spuren und Sektoren darauf angelegt; 2. es wird das A: Verzeichnis angelegt und damit ein DOS-Filesystem; 3. die Diskette wird dem Nutzer zum Zugriff zur Verfügung gestellt. Diese drei Schritte werden unter Linux separat behandelt. Man kann Disketten im MSDOS-Format verwenden, es gibt aber auch andere Formate die z.B. lange Dateinamen und auch Zugriffsrechte unterstützen. Hier die Schrittfolge zum Vorbereiten einer Diskette (man muß dazu `root` sein):

Formatieren einer Standard 1,44 MB Floppydisk (A:):

```
# fdformat /dev/fd0H1440
```

Filesystem erstellen:

```
# mkfs -t ext2 -c /dev/fd0H1440
```

Anstelle von /dev/fd0H1440 muß auf manchen Systemen auch /dev/fd0h1440 verwendet werden. Hiermit wird das Standard-Linuxfilesystem, das ext2-Filesystem, auf der Diskette angelegt. Dieses Filesystem ist jedoch für kleine Datenträger wie Disketten eigentlich nicht so gut geeignet, da es für die Verwaltung großer Festplatten entwickelt wurde. Daher wird für Disketten, auf denen man ein UNIX-artiges Filesystem haben will, das Minix-Filesystem eingesetzt.

```
# mkfs -t minix -c /dev/fd0H1440
```

Um ein MS-DOS Filesystem zu erstellen, benutzt man folgendes Kommando:

```
# mformat a:
```

oder

```
# mkfs -t msdos -c /dev/fd0H1440
```

Vor der Benutzung der Diskette muß sie gemountet werden:

```
# mount -t <typ> /dev/fd0H1440 /mnt
```

wobei <typ> der Type des Filesystems ist, mit dem die Diskette formatiert wurde, also ext2, minix oder msdos. Jetzt kann man auf die Floppy zugreifen. Der Inhalt der Floppy steht jetzt als Inhalt des Verzeichnisses /mnt zur Verfügung, d.h. das Verzeichnis /mnt entspricht jetzt der Diskette. Wenn man fertig ist, *muß* man die Diskette wieder unmounten.

Zum unmounten der Disk:

```
# umount /mnt
```

Jetzt kann man die Diskette aus dem Laufwerk entnehmen. Natürlich muß man Disketten nur dann formatieren und ein Filesystem anlegen, wenn dieses auf den Disketten noch nicht geschehen ist. Für Laufwerk B: ersetzt man einfach `fd0H1440` durch `fd1H1440` in der obigen Anleitung. Alles was man unter DOS mit A: und B: gemacht hat, wird jetzt mit dem Verzeichnis /mnt gemacht. Beispiele:

```
DOS                                         Linux
---------------------------------------------------------------------------

C:\GUIDO>dir a:                             $ ls /mnt
C:\GUIDO>copy a:*.*                         $ cp /mnt/* /docs/temp
C:\GUIDO>copy *.zip a:                      $ cp *.zip /mnt/zip
C:\GUIDO>a:                                 $ cd /mnt
A:>_                                        /mnt/$ _
```

Wenn Sie dieses ganze mounten/unmounten von Disketten nicht mögen, benutzen Sie die `mtools` suite: dieses ist ein Satz von Kommandos, die wie ihr jeweiliges DOS-Gegenstück arbeiten, jedoch jeweils mit einem ‘m’ anfangen, also `mformat`, `mdir`, `mdel` und so weiter. Sie können damit sogar lange Dateinamen erhalten, jedoch keine Rechte. Man benutze diese Kommandos einfach wie die DOS-Kommandos.

Was für Disketten gilt, gilt natürlich auch für andere Arten von Laufwerken, wie z.B. Zip-Disks, CD-ROM’s, neue Festplatten usw. Für Zip-Laufwerke gibt es übrigens ein eigenes HOWTO, das *ZIP Drive mini-HOWTO* , bzw. eine deutsche Version als *Deutsches ZIP-HOWTO*. Hier das Verfahren zum mounten eines CD-ROM:

```
# mount -t iso9660 /dev/cdrom /mnt
```

Das ist der ”offizielle” Weg einen Datenträger ins Dateisystem des Rechners einzubinden (zu mounten). Da es jedoch etwas lästig ist, jedesmal zum mounten `root` sein zu müssen, gibt es eine ”Abkürzung”, die es jedem Nutzer erlaubt, die Datenträger einzubinden und der außerdem Schreibarbeit spart:

Als `root` legt man zuerst die Verzeichnisse `/mnt/a`, `/mnt/a:`, und `/mnt/cdrom` an.

Danach fügt man in `/etc/fstab` die folgenden Zeilen hinzu:

```
/dev/cdrom      /mnt/cdrom   iso9660  ro,user,noauto           0       0
/dev/fd0H1440   /mnt/a:      msdos    user,noauto,umask=000    0       0
/dev/fd0H1440   /mnt/a       minix    user,noauto              0       0
```

Um jetzt eine DOS-Diskette, eine Minix-Diskette und ein CDROM zu mounten, reicht nun folgendes:

```
$ mount /mnt/a:
$ mount /mnt/a
$ mount /mnt/cdrom
```

Auf `/mnt/a`, `/mnt/a:`, und `/mnt/cdrom` kann jetzt von jedem Nutzer zugegriffen werden. Man sollte beachten, daß man für ein frisch angelegtes Minix oder ext2 Filesystem auf der Diskette erst die entsprechenden Rechte vergeben muß, damit alle Nutzer darauf schreiben können:

```
# mount /mnt/a
# chmod 777 /mnt/a
# umount /mnt/a
```

Man sollte sich natürlich im Klaren darüber sein, daß dies ein dickes Sicherheitsloch ist. Wenn Sicherheit also von Bedeutung ist, z.B. in einer Firma, sollte man etwas vorsichtiger sein.

4.2 Das Backup

Jetzt, wo Sie wissen wie man mit Floppies umgeht, noch ein paar Zeilen zum Anlegen eines Backup. Es gibt verschiedene Pakete, die einem dabei helfen sollen, aber als einfachste Methode für ein Backup auf mehreren Medien gibt es:

```
# tar -M -cvf /dev/fd0H1440 /zu_sicherndes_Verzeichnis
```

Hierzu sind i.allg. `root`-Rechte nötig. Man braucht eine formatierte Diskette im Laufwerk sowie ausreichend weitere vorrätig. Um die Daten wiederherzustellen, legt man die erste Diskette ins Laufwerk und gibt ein:

```
# tar -M -xpvf /dev/fd0H1440
```

5 Und was ist mit Windows?

Das Äquivalent zu Windows im gewissen Sinne ist unter Linux das Grafiksystem X11. Im Gegensatz zu Windows oder dem Mac wurde X11 nie unter dem Gesichtspunkt der Benutzerfreundlichkeit oder des Aussehens entwickelt, sondern nur zur Bereitstellung grundlegender Grafikfähigkeiten für UNIX-Workstations. Hier ein paar der hauptsächlichen Unterschiede:

Während Windows überall auf der Welt gleich aussieht, ist das bei X11 nicht so: es ist in wesentlich weiterem Rahmen konfigurierbar. Das jeweilige Aussehen von X11 wird zum großen Teil vom sogenannten *Windowmanager* bestimmt. Seine Aufgabe ist es, die Fenster auf dem Bildschirm zu verwalten. Es gibt mehrere, aus denen man auswählen kann: Die am weitesten verbreiteten sind der `fvwm`, ein einfacher aber speichereffizienter Windowmanager, der `fvwm2`, der Nachfolger des `fvwm`, der `fvwm95`, der dem Desktop etwas Windows95-Flair verleiht, sowie viele andere. Einige davon sehen ausgesprochen schön aus.

Der Windowmanager kann auf vielerlei Weise konfiguriert werden: er kann wie unter Windows agieren, indem er ein Fenster in den Vordergrund bringt, das explizit angeklickt wurde. Alternativ kann er aber ein Fenster auch in den Vordergrund bringen, wenn sich nur die Maus einfach darüber befindet. Dieses und viele andere Dinge können durch das Editieren einer oder mehrerer Konfigurationsdateien verändert werden. Dazu lese man die entsprechenden Dateien der Dokumentation.

X11-Programme werden mit Hilfe spezieller Bibliotheken geschrieben, den Widgetbibliotheken. Diese Bibliotheken enthalten vorgefertigte Elemente wie Eingabefelder für Text, Listenfelder, Knöpfe usw. Da es davon im Gegensatz zu Windows mehrere gibt, sehen die Programme unter Linux teilweise unterschiedlich aus. Die Einfachsten sind die, die die Athenawidgets benutzen (2D-look, `xdvi`, `xman`, `xcalc`); andere nutzen Motif (`netscape`), wieder andere Tcl/Tk, XForms, Qt und was es noch so gibt. Einige dieser Bibliotheken besitzen ein Windows-ähnliches aussehen.

Soweit zum Aussehen von X11, aber was ist mit der Bedienung? Ünglücklicherweise verhalten sich auch hier viele Programme unterschiedlich. Wenn man zum Beispiel eine Textzeile mit der Maus markiert und dann `BACKSPACE` drückt, erwartet man, daß die Zeile verschwindet. Dieses funktioniert aber nicht bei den Applikationen, die auf den Athenawidgets basieren, dagegen funktioniert es mit Programmen die auf Motif, Qt, und Tcl/Tk basieren.

Scrollbars (deutsch ”Rollbalken”), Größenänderungen und das Minimieren zu einem Icon hängen ebenfalls vom Windowmanager und dem Widgetsatz ab. Es wäre zuviel hier alles aufzulisten, daher nur ein paar Punkte. Bei den Programmen mit den Athenawidgets ist es am besten, den Scrollbar mit der mittleren Maustaste zu bedienen. Wenn man keine solche hat, kann man die zwei Mausbuttons gleichzeitig drücken.

Programme haben per Default kein eigenes Icon, man kann ihnen jedoch mehrere zuweisen. Das Ob und Wie hängt vom jeweiligen Windowmanager ab. Der Bildschirm selber heißt *Rootwindow*, und sein Aussehen kann man mit Programmen wie `xsetroot` oder `xloadimage` verändern.

Die Zwischenablage kann nur Text enthalten und verhält sich anders als in Windows. Wenn ein Text markiert wurde wird er auch sofort automatisch in die Zwischenablage kopiert. Wenn man jetzt den Cursor woanders hinbewegt und die mittlere Maustaste klickt, wird der Text aus der Zwischenablage dort eingefügt. Es gibt ein Programm, `xclipboard`, das mehrere Zwischenablagen zur Verfügung stellt.

Drag und Drop ist nur vorhanden, wenn die X-Windowsprogramme dieses unterstützen.

Um Speicher zu sparen ist es besser, nur Programme zu verwenden, die die gleichen Bibliotheken verwenden. Dies ist in der Praxis jedoch nicht immer möglich. Es gibt ein Projekt namens K Desktop Environment, dessen Ziel es ist, X-Windows ähnlich wie Windows ein einheitliches Aussehen und Bedienung zu verpassen. Es ist derzeit im Betastadium, aber bereits recht eindrucksvoll. Wenn es fertig ist, könnte es Windows ziemlich alt aussehen lassen. Unter `http://www.kde.org` findet man im Internet mehr darüber.

6 Einrichten des Systems

6.1 Der Systemstart

Zwei wichtige Dateien unter DOS sind `AUTOEXEC.BAT` und `CONFIG.SYS`, welche beim Booten zur Initialisierung des Systems, zum Setzen von Umgebungsvariablen wie `PATH` oder `FILES` und zum Starten von Treibern und Programmen verwendet werden. Unter Linux gibt es mehrere Dateien für das Starten des Systems, wobei man eine Reihe von ihnen besser unangetastet läßt, bis man genau weiß, was man tut. Hier aber trotzdem eine Liste der wichtigsten Dateien:

```
Dateien                             Bemerkungen

/etc/inittab                        H\"{a}nde weg f\"{u}rs erste!
/etc/rc.d/* bzw. /sbin/int.d/*      dito
```

Falls man nur die PATH-Variable oder eine andere Umgebungsvariable setzen möchte, die Login-Meldung ändern oder automatisch nach dem Login ein Programm starten möchte, kann man dieses in den folgenden Dateien tun:

```
Dateien                         Bemerkungen

/etc/issue                      setzt pre-login Meldung
/etc/motd                       setzt post-login Meldung
/etc/profile                    setzt u.a. PATH und andere Variablen
/etc/bashrc                     setzt u.a. Aliase und Funktionen(s.u.)
/ihr_home_Verz/.bashrc          setzt Ihre Aliase und Funktionen
/ihr_home_Verz/.bash_profile    setzt Umgebung + startet Ihre Programme
/ihr_home_Verz/.profile         dito
```

Wenn letztere Datei existiert (man beachte, daß sie eine versteckte Datei ist), wird sie nach dem Login gelesen und die Anweisungen darin ausgeführt.

Beispiel eines `.profile`:

```
# Ich bin ein Kommentar
echo Umgebung:
printenv | less   # entspricht Kommando SET unter DOS
alias d='ls -l'   # d bedeutet ab jetzt ls -l
alias up='cd ..'
echo "Der Pfad ist jetzt "$PATH
echo "Heutiges Datum: `date`"  # Ausgabe des Kommandos 'date' verwenden
echo "Sch\"{o}nen Tag noch, "$LOGNAME
# Das Folgenden ist eine "Shell-Funktion"
```

```
ctgz() # Auflisten des Inhalts eines .tar.gz Archives.
{
  for file in $*
  do
    gzip -dc ${file} | tar tf -
  done
}
# Ende des .profile
```

`$PATH` und `$LOGNAME` sind Umgebungsvariablen; es gibt noch viele andere, mit denen man experimentieren kann. Hier gilt wieder MSL, z.B. für das Programm `less` und `bash`.

6.2 Dateien zur Programm-Initialisierung

Unter Linux kann fast alles den eigenen Bedürfnissen angepaßt werden. Die meisten Programme haben ein oder mehrere Initialisierungsdateien, an denen man herumbasteln kann, oft mit dem Namen `.Programmnamerc` in Ihrem Home-Verzeichnis. Die ersten, an denen man üblicherweise etwas verändert sind:

.inputrc

benutzt von `bash`, um Tastenzuordnungen festzulegen.

.xinitrc

benutzt von `startx`, um das X Windows System zu initialisieren.

.fvwmrc

benutzt vom Windowmanager `fvwm`. Ein Beispiel ist in: `/usr/lib/X11/fvwm/system.fvwmrc`

.Xdefault

benutzt z.B. von `rxvt`, einem Terminalemulator für X und anderen Programmen.

Mit den meisten bekommt man es früher oder später zu tun, also auch hier wieder MSL.

7 Ein wenig Programmierung

7.1 Shell-Skripte: viel mehr als .BAT Dateien

Falls Sie .BAT Dateien bisher benutzt haben, um lange Kommandoeingaben abzukürzen (wie ich zum Beispiel), kann dies jetzt besser durch das Einfügen von Alias-Anweisungen (siehe obiges Beispiel) in die Dateien `profile` oder `.profile` geschehen. Wenn Ihre .BAT Dateien jedoch komplizierter sind, werden Sie die Skriptsprache, die von der Shell bereitgestellt wird, mögen: Sie ist so mächtig wie QBasic - wenn nicht noch mächtiger. Sie hat Variablen, Strukturen wie while, for, case, if... then... else, und vieles anderes. Sie ist eine gute Alternative zu einer "echten" Programmiersprache.

Um ein Skript, das Gegenstück zu einer .BAT Datei unter DOS, zu schreiben, muß man lediglich eine ganz gewöhnliche ASCII-Datei erstellen, die die gewünschten Befehle enthält, diese speichern und mit dem Kommando `chmod +x <skripdatei>` ausführbar machen. Man kann es dann wie jedes andere Programm aufrufen.

Ein Hinweis: Der Systemeditor ist der `vi`, und nach meiner Erfahrung sind neue Nutzer selten davon erbaut, da sie ihn als schwer zu bedienen empfinden. Ich werde ihn hier nicht weiter erläutern, da ich ihn nicht mag und auch nicht benutze. Weitere Erläuterungen findet man dazu in Matt Welsh's *Linux installation...* (englisch) oder jedem anderen Buch über Linux/UNIX. Es ist vielleicht sinnvoll, sich einen anderen Editor zu besorgen bzw. zu installieren, z.B. `joe` oder `emacs` für X. Zum `vi` nur soviel:

- zum Text einfügen nach dem Start: 'i' eingeben, dann den Text;
- verlassen des vi ohne sichern, <ESC> eingeben, dann ':q!'
- verlassen und sichern, <ESC> eingeben, dann ':wq'
- zum weitereditieren nach einem <ESC> wieder 'i'
- zwischenspeichern, <ESC> eingeben, dann ':w'

Skripte für die bash zu schreiben ist ein weites Feld, und ich will hier nicht weiter darauf eingehen. Hier nur ein Beispiel eines Shell-Skriptes, aus dem man einige grundlegende Dinge erkennen kann:

```
#!/bin/sh
# beispiel.sh
# Ich bin ein Kommentar
# die erste Zeile nicht ändern, sie muß genau so dastehen
echo "Dieses System ist: `uname -a`" # Nutzung der Ausgabe des Kommandos
echo "Mein Name ist $0" # vordefinierte Variablen
echo "Es wurden die folgenden $# Parameter übergeben: "$*
echo "Erster Parameter ist: "$1
echo -n "Ihr Name? " ; read ihr_name
echo Unterschied beachten: "hi $ihr_name" # einklammern mit "
echo Unterschied beachten: 'hi $ihr_name' # einklammern mit '
DIRS=0 ; FILES=0
for file in `ls .` ; do
  if [ -d ${file} ] ; then # falls file ein Verzeichnis
    DIRS=`expr $DIRS + 1`  # DIRS = DIRS + 1
  elif [ -f ${file} ] ; then
    FILES=`expr $FILES + 1`
  fi
  case ${file} in
    *.gif|*jpg) echo "${file}: Bildateien" ;;
    *.txt|*.tex) echo "${file}: Textdateien" ;;
    *.c|*.f|*.for) echo "${file}: Quelldateien" ;;
    *) echo "${file}: allgemeine Dateien" ;;
  esac
done
echo "es gibt ${DIRS} Verzeichnisse und ${FILES} Dateien"
ls | grep "ZxY--!!!WKW"
if [ $? != 0 ] ; then # R\"{u}ckgabewert des letzten Kommandos
  echo "ZxY--!!!WKW nicht gefunden"
fi
echo "genug... 'man bash' eingeben f\"{u}r weitere Informationen."
```

7.2 Kurzer Blick auf C

Unter UNIX ist die Programmiersprache des Systems C, ob es einem gefällt oder nicht. Zur Programmierung von Programmen gibt es jedoch auch eine große Anzahl anderer Sprachen, z.B. FORTRAN, Pascal, Lisp, Basic, Perl, awk uva. .

Hier sind ein paar Hilfestellungen für die, die C bereits beherrschen und von Programmen wie Turbo C++ o.ä. verwöhnt wurden. Der C-Kompiler unter Linux heißt gcc und hat keinerlei der netten Features, wie sie unter DOS und Windows üblich sind: keine IDE, keine Online-Hilfe, kein integrierter Debugger usw. Es ist nur ein reiner Kommandozeilenkompiler der jedoch sehr leistungsfähig und effizient ist. Da er auch auf vielen anderen Unixsystemen verfügbar ist, arbeiten viele Leute lieber mit ihm, als mit dem zu ihrem System standardmäßig ausgelieferten Kompiler. Um sein Standardprogramm `hello.c` zu kompilieren, gibt man ein:

```
$ gcc hello.c
```

Das ergibt eine ausführbare Datei namens `a.out`. Um dieser Datei gleich einen anderen Namen geben zu lassen (z.B. hallo) gibt man ein

```
$ gcc -o hallo hello.c
```

Um eine Bibliothek an ein Programm zu linken, fügt man den Parameter -l<libname> hinzu. Zum Linken der Mathebibliothek zum Beispiel:

```
$ gcc -o mattteprog matteprog.c -lm
```

(Der `-l<libname>` Parameter veranlaßt `gcc` die Bibliothek `/usr/lib/lib<libname>.a` zu linken; also linkt `-lm` `/usr/lib/libm.a`).

So weit so gut. Wenn das Programm jedoch aus mehreren Quelldateien besteht, wird der Einsatz des Hilfsprogrammes `make` sinnvoll. Angenommen man hat einen Parser für Ausdrücke geschrieben: die Quelldatei ist `parser.c` und beinhaltet per #include zwei Headerdateien, `parser.h` und `xy.h`. Jetzt will man die Routinen aus `parser.c` in einem Programm, sagen wir `calc.c`, verwenden, welches seinerseits per #include `parser.h` enthält. Ein ziemliches Durcheinander! Was muß man machen, um `calc.c` zu kompilieren?

In diesem Falle schreibt man ein sogenanntes `makefile`, welches dem Kompiler Auskunft über die Abhängigkeiten zwischen den Quell- und Objektdateien gibt. In unserem Falle:

```
# Dies ist ein makefile zum kompilieren von calc.c
# <TAB> muss durch das Tabulator-Zeichen ersetzt werden!

calc: calc.o parser.o
<TAB>gcc -o calc calc.o parser.o -lm
# calc basiert auf zwei Objektdateien: calc.o und parser.o

calc.o: calc.c parser.h
<TAB>gcc -c calc.c
# calc.o basiert auf zwei Quelldateien

parser.o:  parser.c parser.h xy.h
<TAB>gcc -c parser.c
# parser.o basiert auf drei Quelldateien

# ende des makefile.
```

Abspeichern als Datei `makefile` und eingeben

```
$ make
```

um das Programm zu kompilieren, oder als `calc.mak` speichern und eingeben

```
$ make -f calc.mak
```

`make` sucht automatisch (wenn kein anderer Name per Option `-f` angegeben wird) nach einer Datei namens `Makefile` oder `makefile` im aktuellen Verzeichnis als Datei mit den Abhängigkeiten. Auch hier wieder MSL.

Man kann sich bezüglich der C-Funktionen Hilfe anzeigen lassen. Sie sind im Manual, Abschnitt 3 beschrieben. Beispiel:

```
$ man 3 printf
```

Darüber hinaus gibt es noch viele weitere Bibliotheken, die das Programmieren erleichtern. Darunter sind z.B. `ncurses`, mit der man Textmodus-Programme erstellen kann, `svgalib` um Grafik auch ohne X-Windows zu verwenden, sowie viele andere. Wer noch einen Schritt weitergehen möchte, kann sich an die Programmierung von X-Windows heranwagen. Auch dafür gibt es viele Bibliotheken, die die Programmierung unter X vereinfachen, wie z.B.:

XForms

`einstein.phys.uwm.edu/pub/xforms`

MGUI

`www.volftp.vol.it/IT/IT/ITALIANI/MORELLO/index.htm`

LessTif

`www.hungry.com`: Eine Bibliothek, die als freier Ersatz des kommerziellen Motif gedacht ist, welches die Standardbibliothek für grafische Programmierung unter UNIX ist. LessTif ist zum jetzigen Zeitpunkt (Oktober 1997) noch nicht ganz fertig, aber trotzdem schon weitgehend benutzbar.

Qt

`www.troll.no`: Auf dieser Bibliothek basiert der KDE, oben erwähnter Desktop, der alle Chancen hat, der Standard unter Linux zu werden.

Mehr Informationen über Bibliotheken und Programmiertools im allgemeinen gibt es unter:

`http://www.xnet.com/~blatura/linapp6.html`

Es gibt viele Editoren, die wie eine IDE arbeiten können: `emacs` und `jed` zum Beispiel haben auch Syntax Highlighting, automatische Einrückung und so weiter.

Wenn man unbedingt eine IDE wie unter den Borlandprodukten haben will, kann man sich z.B. die Pakete `rhide` von

`sunsite.unc.edu:/pub/Linux/devel/debuggers/`

oder `xwpe` von

`sunsite.unc.edu:/pub/Linux/apps/editors/`

anschauen. Es wird Ihnen wahrscheinlich gefallen.

8 Das verbleibende 1%

8.1 Virtuellen Speicher anlegen

Obwohl Linux bereits mit 2 MByte RAM laufen kann, gilt hier auch das, was für alle anderen Betriebsysteme gilt: RAM ist durch nichts zu ersetzen, außer durch mehr RAM. Hier ein paar Richtlinien für den RAM-Ausbau in der Praxis:

- ohne X-Windows minimal 4 MByte empfohlen, besser 8 MByte;
- mit X-Windows minimal 8 MByte empfohlen, besser 16 MByte.

Natürlich hängt die Größe des RAM's auch von den verwendeten Programmen ab. Zusätzlich kann man, analog zu Windows, sich noch virtuellen RAM auf der Festplatte anlegen, auch Swap-Space genannt. Um sich 8 MByte Swap-Space anzulegen gibt man als `root` ein:

```
# dd if=/dev/zero of=/swapfile bs=1024 count=8192
# mkswap /swapfile 8192
# sync
# swapon /swapfile
```

Entweder wird dann die letzte Zeile in `/etc/rc.d/rc.local` (oder `/sbin/rc.d/rc.local`, je nach dem wo `rc.local` ist) eingefügt, um das Swapfile beim Systemstart automatisch verfügbar zu machen, oder man fügt folgende Zeile in `/etc/fstab` hinzu:

```
/swapfile    swap    swap    defaults
```

Eine weitere und im allgemeinen effizientere Möglichkeit ist das Einrichten einer eigenen Swap-*Partition*. Hierzu legt man sich erst ganz normal eine neue leere Partition auf der Festplatte mit fdisk an (z.B. 30 Mb), gibt ihr den Partitionstyp 83, bereitet dann die Partition vor mit:

```
# mkswap -c /dev/mein_swap_device
```

und fügt zu guter letzt in die `/etc/fstab` ein:

```
/dev/mein_swap_device    swap    swap    defaults
```

Beim nächsten Sytemstart ist die neue Swap-Partition dann verfügbar.

8.2 Arbeiten mit tar & gzip

Unter UNIX gibt es einige weit verbreitete Programme zum Archivieren und Komprimieren von Dateien. Um Archive anzulegen wird `tar` benutzt, ähnlich dem `PKZIP`, aber ohne Kompression. Um ein neues Archiv anzulegen:

```
$ tar -cvf <archiv_name.tar> <Datei> [Datei...]
```

Um eine Datei aus einem Archiv zu extrahieren:

```
$ tar -xpvf <archiv_name.tar> [Datei...]
```

Zum Auflisten des Inhaltes eines Archives:

```
$ tar -tf <archiv_name.tar> | less
```

Man kann Dateien (und damit auch tar-Archive) mit `compress` komprimieren. Es ist jedoch eigentlich veraltet und sollte daher nur noch in Ausnahmefällen verwandt werden. Das gebräuchlichere Programm ist `gzip`. Aufruf:

```
$ compress <Datei>
$ gzip <Datei>
```

Das erzeugt eine Datei mit der Endung .Z (`compress`) oder .gz (`gzip`). Diese Programme können nur jeweils eine Datei gleichzeitig komprimieren. Zum entkomprimieren:

```
$ compress -d <Datei.Z>
$ gzip -d <Datei.gz>
```

MSL.

Die Programme `unarj`, `zip` und `unzip` (PK??ZIP kompatibel) existieren ebenfalls. Dateien mit der Endung `.tar.gz` oder `.tgz` (archiviert mit `tar`, dann komprimiert mit `gzip`) sind in der Unixwelt so verbreitet wie .ZIP Dateien unter DOS. So listet man den Inhalt eines `.tar.gz` Archivs auf:

```
$ gzip -dc <Datei.tar.gz> | tar tf - | less
```

8.3 Programme installieren

Zunächst erst einmal ist das Installieren von Programmen die Aufgabe von `root`. Einige Linuxprogramme sind als speziell vorbereitete `.tar.gz` oder `.tgz` Archive verfügbar. Diese kann man vom Verzeichnis / aus entpacken durch ein

```
# gzip -dc <Datei.tar.gz> | tar xvf -
```

oder auch (bewirkt das gleiche)

```
$ tar -zxf <file.tar.gz>
```

Die Dateien werden dann automatisch in den richtigen Verzeichnissen entpackt, die (wenn notwendig) gleich miterzeugt werden. Slackwarenutzer haben hier ein Hilfsprogramm namens `pkgtool`. Ein anderes mit ähnlichem Zweck ist `rpm`, welches dank Red Hat ebenfalls frei verfügbar ist. Archive für den `rpm` erkennt man an der Endung `.rpm`. Für ihn gibt es ein eigenes HOWTO, das *RPM HOWTO*, oder in der deutschen Version als *Deutsches RPM-HOWTO*.

Andere Archive sollten nicht von / aus installiert werden. Solch ein Archiv enthält üblicherweise ein Hauptverzeichnis `paketname/` und weitere Dateien und Unterverzeichnisse unter diesem. Nach dem Linux-Filesystemstandard sollten diese unter `/usr/local` installiert werden, wer eigenbrödlerisch veranlagt ist, kann sie aber auch gerne irgendwo anders installieren. Darüber hinaus werden viele Pakete auch als C- oder C++ - Quelldateien geliefert, die man erst übersetzen muß, um ein ausführbares Programm zu erhalten. In den meisten Fällen genügt (nach dem lesen der `README`- und/oder `INSTALL`-Datei des Paketes) ein `make` im Hauptverzeichnis. Natürlich sollte man dafür den `gcc` oder `g++` Kompiler installiert haben.

8.4 Nützliche Tips

Kommandovervollständigung: drücken der <TAB> Taste während der Eingabe eines Befehls auf der Kommandozeile vervollständigt den angefangenen Befehl. Beispiel: Man will die Zeile `gcc dies_ist_ein_sehr_langer_Name.c` eingeben. Eintippen von `gcc die`<TAB> veranlaßt die Shell, automatisch den langen Dateinamen zu ergänzen, falls die angegebenen Buchstaben ausreichen, um die Datei eindeutig zu identifizieren.

Zurückscrollen: drücken von SHIFT + Bild hoch (Page up) ermöglicht es, ein paar Seiten des Bildschirminhaltes zurückzuholen, je nachdem wieviel Videospeicher man hat.

Reset für den Bildschirm: Wenn man mit `more` oder `cat` eine Binärdatei auf den Bildschirm ausgegeben hat, kann es passieren, daß der Bildschirm danach völlig unlesbar wird. Um das wieder geradezubiegen, gibt man blind ein `reset` ein oder diese Folge von Zeichen: `echo CTRL-V ESC c RETURN`.

Text einfügen: auf der Konsole siehe unten; unter X klickt man in das (z.B. `xterm`) Fenster und zieht die Maus bei gedrückter linker Maustaste über den Text. Dann in dem Fenster, wo der Text hin soll, die mittlere Maustaste drücken, oder wenn man eine Zweitastenmaus hat, beide Tasten gleichzeitig. Es gibt auch das `xclipboard` (leider nur für Text) als Alternative.

Maus nutzen: Zunächst muß man den gpm installieren, einen Maustreiber für die Konsole. Text selektieren wie unter X und dann die rechte Maustaste zum Einfügen drücken (oder auch die mittlere). Dieser Mechanismus funktioniert über mehrere virtuelle Terminals hinweg.

Kernelmeldungen: man kann als root in /var/adm/messages oder /var/log/messages nachschauen, was der Kernel so an Meldungen produziert. Hier stehen auch die Meldungen vom Booten des Systems. Das Kommando dmesg ist hier auch hilfreich.

8.5 Nützliche Programme und Kommandos

Diese Liste ist natürlich nur eine persönliche und richtet sich nach meinen Bedürfnissen und Vorstellungen. Zunächst einmal, wo man sie finden kann: Da sich sicher alle auskennen mit dem Internet und Dinge wie archie und ftp beherrschen, hier nur die drei wichtigsten Adressen für Linux: sunsite.unc.edu, tsx-11.mit.edu, und nic.funet.fi. Man benutze soweit möglich den nächstgelegenen Mirror.

at

erlaubt das automatische Ausführen von Programmen zu bestimmten Zeiten;

awk

ist eine einfache und mächtige Sprache zum Bearbeiten von Datenfiles (und nicht nur). Wenn z.B. data.dat das Datenfile mit mehreren Feldern pro Zeile ist, dann gibt

```
$ awk '$2 ~ "abc" {print $1, "\t", $4}' data.dat
```

das 1. und 4. Feld jeder Zeile von data.dat aus, die *abc* enthält.

cron

ist ein nützliches Programm, welches zum periodischen Ausführen von Kommandos zu einem bestimmten Datum und einer Zeit dient;

delete-undelete

wie der Name schon sagt;

df

gibt Informationen über die gemounteten Disks (freier Platz, usw.);

dosemu

ermöglicht das Abarbeiten einer ganzen Reihe von DOS-Programmen; mit etwas Herumbasteln bekommt man sogar Windows 3.x zu laufen, allerdings sollte man dann davon nicht zuviel erwarten;

file <Dateiname>

gibt darüber Auskunft, was Dateiname für ein Typ ist (ASCII-Text, ausführbar, Archiv, etc.);

find

(siehe auch Kapitel 3.3) ist eines der mächtigsten und nützlichsten Kommandos. Es wird dazu benutzt, Dateien zu finden, die gewissen Bedingungen entsprechen und auf diese dann bestimmte Kommandos anzuwenden. Allgemeine Form des Kommandos find :

```
$ find <Verzeichnis> <Ausdruck>
```

wobei <Ausdruck> die Suchkriterien und Kommandos enthält. Beispiele:

```
$ find . -type l -exec ls -l {} \;
```

sucht alle Dateien, die symbolische Links sind und gibt aus, auf was sie ein Link sind.

```
$ find / -name "*.old" -ok rm {} \;
```

sucht alle Dateien, die auf `.old` enden und löscht diese nach vorheriger Rückfrage.

```
$ find . -perm +111
```

sucht alle Dateien, deren Rechte 111 sind (ausführbar).

```
$ find . -user root
```

sucht alle Dateien, die `root` gehören. Es gibt noch viele weitere Möglichkeiten, also wieder MSL.

gnuplot

hervorragendes Programm für wissenschaftliche Zeichnungen;

grep

sucht Textmuster in Dateien. z.B.:

```
$ grep -l "Geologie" *.tex
```

listet alle Dateien *.tex, die das Wort *Geologie* enthalten. Das Programm `zgrep` arbeitet mit gezippten Dateien. MSL.

tcx

komprimiert ausführbare Dateien und beläßt sie ausführbar;

joe

guter Editor. Wenn man ihn mit `jstar` aufruft kennt er die selben Tastaturkombinationen wie WordStar et al., einschließlich DOS und Borlands Turbo-Editoren;

less

wahrscheinlich der beste Textbrowser; wenn korrekt konfiguriert, kann man damit auch gzip-, tar- und zip-Dateien anzeigen;

lpr

<Datei> druckt eine Datei im Hintergrund aus. Zum Abfragen des Status des Druckauftrages gibt es `lpq`; um eine Datei aus der Warteschlange des Druckers zu entfernen, gibt es `lprm`;

mc

guter Dateimanager (ähnlich Norton Commander);

pine

gutes E-Mail Programm;

script <script_datei>

kopiert, was auf dem Bildschirm erscheint, nach `script_file`, bis man das Kommando `exit` gibt. Nützlich zur Fehlersuche;

sudo

erlaubt normalen Nutzern, bestimmte Aufgaben von `root` zu erledigen (z.B. Disketten formatieren und mounten; MSL);

uname -a

gibt Informationen über das eigene System aus;

zcat

und `zless` sind nützlich, um gezippte Textdateien anzuschauen, ohne sie erst zu entpacken. Mögliche Verwendungen:

```
$ zless textfile.gz
$ zcat textfile.gz | lpr
```

weitere:

Die folgenden Kommandos sind gelegentlich recht nützlich: `bc, cal, chsh, cmp, cut, fmt, head, hexdump, nl, passwd, printf, sort, split, strings, tac, tail, tee, touch, uniq, w, wall, wc, whereis, write, xargs, znew.` MSL.

8.6 Gebräuchliche Erweiterungen und zugehörige Programme

Man begegnet unter Linux vielen Arten von Dateierweiterungen. Unter Weglassung der ausgefalleneren Varianten (z.B. Fonts u.a.) hier eine Liste, was was ist:

.1 8

Manualseiten. Benutzt von `man`.

.arj

Archiv angelegt von `arj`. `unarj` zum entpacken.

.dvi

Ausgabedateien von TeX (siehe unten). `xdvi` zum Anschauen; `dvips` zum Umwandeln in eine Postscript .ps Datei.

.gif

Bilddatei. Prgramme: `seejpeg`, `xpaint`, `xv`, `xli`, `gimp`, ...

.gz

mit `gzip` komprimierte Datei.

.info

info Datei (eine Art Alternative zu Manualseiten). Programm: `info`.

.jpg, .jpeg

Bilddatei. Programme siehe `.gif`.

.lsm

Linux Software Map Datei. Einfache ASCII Datei, die die Beschreibung eines Paketes enthält.

.ps

Postscriptdatei. Zum Anschauen und Drucken `gs` und evtl. `ghostview`.

.rpm

Red Hat Paket. Kann mittels des Paketmanagers `rpm` installiert werden

.tgz, .tar.gz

Archive angelegt mit `tar` und komprimiert mit `gzip`.

.tex

Textdatei, die an TeX übergeben wird, ein sehr mächtiges Text-Satzprogramm. Dazu benötigt man das Paket `tex`, das in den meisten Distributionen enthalten ist. Man hüte sich jedoch vor `NTeX`, das Fonts zerstört hat und Teil einiger Slackwareversionen war.

.texi

Texinfodatei, kann als Quelle für TeX und info Dateien dienen (siehe `.info`). Programm: `texinfo`.

.xbm, .xpm, .xwd

Bilddatei. Programm: `xpaint`.

.Z

Datei komprimiert mit `compress`.

.zip

Archiv angelegt mit `zip`. Programme: `zip` und `unzip`.

9 Das Ende, fürs Erste

Glückwunsch! Sie haben jetzt UNIX ein bißchen kennengelernt und sind bereit zu beginnen, damit zu arbeiten. Vergessen Sie jedoch nicht, daß Sie immer noch nicht alles wissen und noch eine gute Portion Praxis brauchen werden, um Linux richtig ausschöpfen zu können. Wenn es aber darum ging, sich ein paar Programme zu besorgen und anzufangen, mit ihnen zu arbeiten, dann sollte das hier schon reichen.

Ich bin sicher, daß Sie Linux mögen werden und immer mehr darüber lernen werden - so wie alle anderen. Ich bin mir auch ziemlich sicher, daß Sie nicht mehr zu DOS zurückkehren werden. Hoffentlich war alles einigermaßen verständlich und hat meinen 3 oder 4 Lesern geholfen.

9.1 Copyright

9.2 Hinweis

From DOS to Linux HOWTO wurde geschrieben von Guido Gonzato, `guido@ibogfs.df.unibo.it`. In dieser Version wurden zusätzlich einige Korrekturen und Ergänzungen vom Übersetzer vorgenommen, die in das englische Original als Version 1.3 rückübertragen wurden. Vielen Dank an Matt Welsh, den Autor von *Linux Installation and*

Getting Started, an Ian Jackson, den Autor von *Linux frequently asked questions with answers*, an Giuseppe Zanetti, den Autor von *Linux*, an alle die mir Vorschläge gesandt haben und natürlich an Linus Torvalds und GNU, die Linux geschaffen haben.

Dieses Dokument wird so wie es ist zur Verfügung gestellt. Ich habe mir große Mühe gegeben, soweit wie möglich alles korrekt darzustellen. Dennoch kann ich keinerlei irgendwie geartete Garantie übernehmen. Der Gebrauch der Informationen ist also auf eigenes Risiko, und ich übernehme in keiner Weise Verantwortung für etwaige Schäden, die aus der Anwendung dieser Informationen entstehen.

Feedback ist willkommen. Für Anregungen, Flames und Wünsche wende man sich an mich.

Viel Spaß mit Linux und dem Rest,

Guido `=8-)`