

Betriebssystem



Vortrag

Abteilung: Informatik
Fach: Betriebssystem
Klasse: TS 5
Verfasser: M. Keller

Thema: **Linux**



Betriebssystem



<u>Inhaltsverzeichnis:</u>		Seite
1	Linux – eine Einführung:	4
1.1	Unix Einführung	4
1.2	Was ist Linux	4
1.3	Woher bekomme ich Linux	4
1.4	Wer benutzt Linux	5
1.5	Was ist das Besondere an Linux	5
1.6	Was kann Linux mehr	5
2	Erste Schritte:	6
2.1	Login	6
2.2	Logout	7
2.3	Virtuelle Konsolen	7
2.4	Verzeichnisse und Dateien	8
2.5	Hände weg von fremden Eigentum	12
2.6	Hilfe	14
2.7	Editoren	15
3	Das Dateisystem:	15
3.1	Dateien und Verzeichnisse	15
3.2	Jokerzeichen	16
3.3	Verzeichnisstruktur	17
3.4	Zugriffsrechte	19
3.5	Gruppenzwang	20
3.6	Besitzerwechsel	20
3.7	Andere Dateisysteme	21
3.8	Prozesse	21
3.9	Interprozesskommunikation	23
3.10	Das Prozessdateisystem	24
4	Wichtige Kommandos:	25
4.1	Kommandos zur Dateiverwaltung	25
4.2	Texte bearbeiten	26
4.3	Packen/Entpacken/Archivieren	27
4.4	Prozessverwaltung	27
4.5	Netzwerk-Administration	28



Betriebssystem

5	Der Kommandozeileninterpreter "bash":	28
5.1	Andere Shells	28
5.2	Kommandoeingabe	28
5.3	Wichtige Tastenkürzel	29
5.4	Alias	29
5.5	Pipes	30
5.6	Mehrere Kommandos	31
5.7	Programmierung der Bash	32
6	Der Kernel	33
6.1	Nachladbare Funktionalität - Module	33
6.2	Ein eigener Kernel	34
6.3	Linux Boot Loader (LILO)	35
6.4	Plug & Play	36



Betriebssystem

1 Linux Einführung:

1.1 Unix Einführung

UNIX wurde Ende der 60'er Jahre von Ken Thompson und Dennis Ritchie bei den Bell Laboratories entwickelt.

Es zeichnete sich durch folgende Neuerungen aus:

- in der Hochsprache (C) geschrieben (nur sehr wenig Assemblercode)
- dadurch sehr gut zu portieren
- von Anfang an für "gehobene Ansprüche" geplant (Großrechner)
- es ist ein Multi-User-Betriebssystem (jeder Benutzer hat seine zugewiesenen Rechte)
- war schon damals ein sog. "32-Bit-Betriebssystem"
- TCP/IP als Grundlage des Internets wurde darauf entwickelt
- daraus entwickelten sich Client-Server Grundlagen
- mit hierarchischem Filesystem (Directories)
- eines der ersten Betriebssysteme mit virtueller Speicherverwaltung
- Mitte der Achtziger Jahre Entwicklung von XWindow
- mit Firewall und anderen Schutzmechanismen

Vor allem bei Studenten war UNIX sehr beliebt und wurde an vielen Universitäten weiterentwickelt.

Trotz seiner Vorteile wurde es nicht als Betriebssystem für den IBM-PC verwendet, da dessen Hardware damals für UNIX zu schwach war (vor allem wegen der Segmentierung der Prozessoren 8088 bis 80286, die keine lineare Adressierung zuließ).

Da viele Hersteller von Computern eigene UNIX-Versionen herausbrachten, waren zudem die Programme sehr teuer (keine "Binärkompatibilität").

1.2 Was ist Linux

Linux ist ein Unix-ähnliches Betriebssystem, welches auf den Fähigkeiten von UNIX aufsetzt und keine Lizenzgebühren kostet. Es wurde von dem finnischen Studenten Linus Torvalds angefangen, der die Entwicklung des Kernels leitet. Mittlerweile sind tausende Programmierer weltweit an der Weiterentwicklung beteiligt. Es ist auch ein sogenanntes "32-Bit-Betriebssystem" und enthält alle bahnbrechenden Entwicklungen von Unix, wie Multi-User-Fähigkeit, Internet-Vernetzung, Server-Client-Konzept, Rechtevergabe, XWindow usw...

1.3 Woher bekomme ich Linux

Wer einen Internetanschluss hat, könnte sich Linux über das Netz herunterladen. Da es einige 100 MB sind, ist dies nicht besonders praktikabel. Deshalb gibt es so genannte



Betriebssystem

Distributoren, die diese Arbeit dem Anwender abnehmen und die einzelnen Dateien auf CD brennen. Das ist natürlich mit Kosten verbunden, die der Anwender bezahlen muß. Dennoch ist die eigentliche Software kostenlos.

1.4 Wer benutzt Linux

Es wird mittlerweile weltweit schon von schätzungsweise mehr als 7 Millionen Anwendern benutzt. Auch viele Firmen, wie z.B. Mercedes und Ikea verwenden es. Ferner entwickeln viele kommerziellen Anbieter ihre Produkte für Linux. Besonders im Serverbereich ist Linux stark vertreten, obwohl keine Werbung dafür gemacht wird. So benutzen ca. 80% der Bürgernetze Linux als Web-Server.

1.5 Was ist das Besondere an Linux

Das Herausragende an Linux ist, dass es weder einer Organisation noch einer Firma gehört. Alle Entwickler, die an Linux arbeiten, tun dies freiwillig und ohne Bezahlung. Auch die meisten Anwendungen unterliegen dem gleichen Prinzip. Diese Methode ist außerordentlich erfolgreich (sie wird auch als Basar-Methode bezeichnet). So kann Linux ohne Lizenzprobleme auf einen oder mehrere hundert Rechnern installiert werden. Es gibt nämlich niemanden, der eine Lizenz verlangt. Dies gilt auch für andere freie Betriebssysteme, wie z.B. [FreeBSD](#). Nur haben diese (noch) nicht die Bedeutung von Linux erlangt.

1.6 Was kann Linux mehr

- Es ist sehr stabil. So kann ein Rechner damit durchaus mehrere Monate laufen, ohne ein einziges Mal in der Zwischenzeit abzustürzen. Dadurch sehr gut zu portieren
- Es kann auf vielen Hardware-Plattformen eingesetzt werden: vom PC mit 386-Prozessor bis zu Supercomputern und 64-Bit Prozessoren.
- Es sind bisher kaum Viren für Linux bekannt, das heißt, es besteht kaum Virengefahr. Selbst dann, wenn ein Virus eingeschleppt wird, ist die Gefährdung minimal, wenn ein befallenes Programm nicht mit Root-Rechten gestartet wird.
- Es ist ein Multi-User-Betriebssystem; das heißt, auf einem Rechner können mehrere Benutzer gleichzeitig (z.B. über ein Netzwerk) Programme starten, die dann unter deren Berechtigung laufen. Bei einem typischen Fileserver, wie mit Novell oder Windows NT, ist das nicht möglich.
- Durch die Multi-User-Fähigkeiten ist z.B. Fernwartung leicht möglich. Ferner kann ein Anwender im Hintergrund ein lang dauerndes Programm ablaufen lassen, während ein anderer das System gerade benutzt. Es kann ohne Grafik gestartet werden. Alle Funktionen des Betriebssystems sind dann immer noch vorhanden. Daraus entwickelten sich Client-Server Grundlagen
- Die Grafik (XWindow) ist netzwerkfähig. Das bedeutet, dass ein Anwender einen Prozess auf einem weit entfernten (schnelleren) Rechner starten und die Grafik dieses Programms auf dem eigenen Rechner beobachten und bedienen kann. Eines der ersten Betriebssysteme mit virt. Speicherverwaltung

Betriebssystem



- Ein Anwender hat mehrere Desktops (Arbeitsflächen) gleichzeitig zur Verfügung, auf einem oder auch auf mehrere Monitore verteilt. Ein Mausklick genügt dann, um den Desktop zu wechseln. Bei vielen offenen Fenstern bleibt so die Übersicht erhalten.
- Es braucht so gut wie nie heruntergefahren werden. Selbst dann nicht, wenn die Systemkonfiguration komplett umgestellt wird. Eine Benachrichtigung der laufenden Prozesse (oder Dienste) genügt. Selbst System-Programme können im laufenden Betrieb aktualisiert werden.
- Es kann problemlos über ein Netzwerk installiert und eingerichtet werden (Fernadministration).
- Es kann als Fileserver für Windows-Rechner benutzt werden. Windows NT und Novell können damit fast komplett ersetzt werden. Ferner sind Vernetzungen mit nahezu allen bekannten Netzwerken möglich.
- Es bietet nicht nur eine sichere Rechtevergabe sondern auch die Möglichkeit, Benutzern einen bestimmten maximalen Plattenplatz (z.B. 2 MB) zur Verfügung zu stellen, der automatisch auf Überschreitung überwacht wird.
- etc..

2 Erste Schritte:

2.1 Login

Linux (und UNIX im Allgemeinen) ist ein Mehrbenutzersystem, d.h. gleichzeitig können theoretisch unbegrenzt viele Nutzer mit nur einem System arbeiten. Um Dateien eindeutig einem Nutzer zuordnen zu können, benötigt Linux Informationen über diesen, d.h. ein Nutzer muss sich beim System anmelden:

```
login: makeller
```

Ein Nutzer könnte sich leicht als ein anderer ausgeben, z.B. als root, einer Nutzerkennung, die auf jedem Unix-System existiert. Um vor Missbrauch zu schützen, muss dieser sich dem System gegenüber auch ausweisen, ein geheimes Passwort sorgt für den Schutz:

```
Password: *****
```

Stimmen Nutzerkennung und Passwort überein, gelingt das Anmelden am System und der Nutzer landet in seinem Heimatverzeichnis. Linux startet eine Shell, einen Kommandozeileninterpreter, der die Interaktion zwischen Nutzer und System steuert: die Shell nimmt Eingaben vom Nutzer entgegen, interpretiert diese und veranlasst das Betriebssystem zu entsprechenden Aktionen...

Betriebssystem



2.2 Logout

Unter DOS war es noch üblich, den Ausschalter zur Beendigung einer Sitzung am Rechner zu nutzen. Auch unter Linux dient der Ausschalter dem endgültigen Knock-out des Rechners; zuvor jedoch ist ein ordnungsgemäßes Herunterfahren des Systems dringend zu empfehlen. Der Grund hierfür ist in erster Linie die Verfahrensweise beim Zugriff auf dauerhafte Speichermedien, wie z.B. die Festplatte. Aus Performancegründen werden Schreibzugriffe auf diese nicht sofort durchgeführt, sondern erst bei Bedarf bzw. nach Ablauf einer voreingestellten Zeitspanne (meist 30 sec). Somit werden Systembus und DMA-Controller entlastet. Würde der Ausschalter Linux in die Jagdgründe befördern, wären die Daten für immer verloren, aber zum Glück existieren mehrere Möglichkeiten, dem System einen anständigen Abtritt zu ermöglichen:

```
root@maexen> reboot
root@maexen> shutdown -r now
```

booten den Rechner neu.

```
root@maexen> halt
root@maexen> poweroff
root@maexen> shutdown -h now
```

hält das System dagegen an (poweroff oder halt -p schaltet bei ATX-Boards den Rechner aus). Obige Befehle bedürfen aber der Berechtigung des Administrators (also root), dem gewöhnlichen Nutzer steht bestenfalls (falls nicht vom Administrator unterbunden) die Tastenkombination CTRL-ALT-DEL zur Verfügung, die ein reboot bewirkt.

2.3 Virtuelle Konsolen

Linux ist ein Multitasking- und Multiuser-System und bietet damit mehrere Möglichkeiten, Kommandos gleichzeitig auszuführen. Mit den Tastenkombinationen

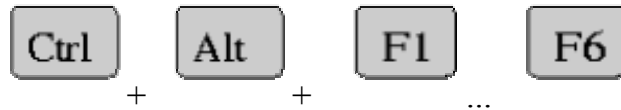


lässt sich zwischen den so genannten virtuellen Konsolen wechseln. Auf diesen Konsolen erscheint ein Login-Prompt und man kann auf jeder Konsole eine Sitzung eröffnen. So kann man auf der ersten Konsole ein Programm editieren und dieses auf der nächsten Konsole compilieren, ohne den Editor beenden zu müssen. Zu jeder Zeit kann natürlich nur eine physische Konsole (Bildschirm) aktiv sein; um die Ausgaben der einzelnen Programme regulieren zu können, werden alle "normalen" Ausgaben auf der virtuellen und nur Fehlermeldungen auf dem Bildschirm ausgegeben.

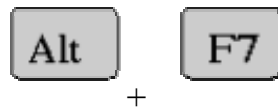


Betriebssystem

Arbeitet man unter dem X-Window-System, erreicht man eine der Textkonsolen über die Tastenkombinationen



Der X-Server selbst ist auf die 7. Konsole aktiv, zu der man über



gelangt.

2.4 Verzeichnisse und Dateien

Anmerkung:

In allen weiteren Beispielen sind Kommandofolgen, die die Rechte des Systemadministrators root erfordern, symbolisch durch das Prompt:

```
root@maexen>
```

dargestellt; Kommandofolgen, die dem "gewöhnlichen" Nutzer zur Verfügung stehen, werden durch:

```
user@maexen>
```

eingeleitet.

Dateien dienen zum Speichern von Daten. Verzeichnisse gruppieren die Dateien in einer baumartigen Struktur. Ein Baum besitzt (falls er nicht gefällt wurde :-)) eine Wurzel, also ist der Ursprung aller Verzeichnisse das root-Verzeichnis "/". Mit

```
user@maexen> cd /
```

wechselt man in das Wurzelverzeichnis `cd - change directory`, wo ein Listing (`ls - list`) auf den meisten Systemen die folgende Ausgabe liefern sollte:

```
bin      etc      lost+found  opt      sbin     var
boot    home    mnt         proc     tmp
dev      lib     msdos      root     usr
```

Wollen wir wieder nach Hause (`/home/user`), geben wir einfach

```
user@maexen> cd /home/user/
```

Betriebssystem



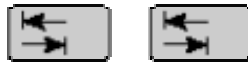
ein und landen in unserem Home-Verzeichnis. Das Kommando `cd` wertet im Falle fehlender Argumente (oft wird auch von Optionen gesprochen) die Shell-Variablen `HOME` aus, die den Pfad zum entsprechenden Verzeichnis enthält. Den Inhalt der Shell-Variablen kann man sich mittels des Aufrufes

```
user@maexen> echo $HOME
```

betrachten. Weitere wichtige Shell-Variablen seien an dieser Stelle kurz erwähnt:

Variable	Inhalt
<code>DISPLAY</code>	Name des Displays der Standardausgabe
<code>HOSTNAME</code>	Name des Rechners
<code>MANPATH</code>	Suchpfade für Manual pages
<code>PATH</code>	Suchpfade für Programme
<code>SHELL</code>	Enthält den Namen der Shell (z.B. <code>bash</code>)
<code>USER</code>	Nutzererkennung

Gibt man



```
user@maexen> echo $
```

ein, erhält man eine Auflistung aller Shell-Variablen. `echo` schreibt sein Argument auf die Standardausgabe, vergisst man das "\$"-Zeichen, würde `echo` den Namen der Shellvariablen wiedergeben.

```
user@maexen> mkdir new_directory
```

erstellt ein neues Verzeichnis `new_directory`. Betrachtet man dessen Inhalt:

```
user@maexen> ls -a new_directory
```

findet man `."` für das Verzeichnis selbst und `..`, was auf das übergeordnete Verzeichnis verweist.

Möchte man schließlich das Verzeichnis wieder löschen, trommelt man

```
user@maexen> rmdir new_directory
```

auf die Tastatur und vermisst beim nächsten Listing das Verzeichnis, sofern dieses leer war! Das zum Löschen von Dateien gedachte Kommando `rm` - remove kann auch auf Verzeichnisse angewendet werden:

```
user@maexen> rm -r new_directory
```



Betriebssystem

Mit der Option "-r" (für rekursiv) entfernt rm auch alle Unterverzeichnisse und letztlich das Verzeichnis selbst.

Erinnert man sich des Namens des aktuellen Verzeichnisses nicht, hilft

```
user@maexen> pwd
```

(*print working directory*), das den Namen (mit Pfad) wiedergibt.

Wenden wir uns nun den Dateien zu. Wie das Löschen funktioniert, haben wir schon kennengelernt: mit dem Kommando rm. Zum Erstellen neuer Dateien gibt es mehrere Möglichkeiten. Die Einfachste ist

```
user@maexen> touch Dateiname
```

das eine leere Datei anlegt, sofern der Dateiname noch nicht im aktuellen Verzeichnis existierte. Gab es sie bereits, ändert touch ihr letztes Änderungs- und Zugriffsdatum (und das ist die wahre Berufung des Kommandos;-); der Grund hierfür wird uns im Zusammenhang mit make klar werden.

Ausgaben von Kommandos lassen sich in eine Datei umlenken, zum Beispiel legt

```
user@maexen> ls / > root_inhalt
```

eine Datei root_inhalt mit folgendem Inhalt an:

bin	etc	lost+found	opt	sbin	var
boot	home	mnt	proc	tmp	
dev	lib	msdos	root	usr	

Den Inhalt einer solchen Datei schaut man sich am besten mit einem so genannten Pager an:

```
user@maexen> less root_inhalt
```

oder mit

```
user@maexen> more root_inhalt
```

Pager haben gegenüber Editoren (diese behandeln wir später) den Vorteil, dass man nicht aus Versehen den Inhalt der Datei ändern kann. Ob man more oder less bevorzugt, ist Geschmackssache; beide Pager unterscheiden sich nur geringfügig in der Bedienung (beide beendet man durch <Q>).



Betriebssystem

Für sehr lange Dateien interessiert man sich meist nur für die ersten/letzten Zeilen (z.B. werden in `/var/log/messages` alle Systemmeldungen protokolliert, wobei man den Grund für ein ungewöhnliches Systemverhalten sicher am Ende der Datei finden wird).

```
user@maexen> head Dateiname
```

zeigt die ersten (Voreinstellung 10) Zeilen der Datei `Dateiname` an,

```
user@maexen> tail Dateiname
```

dagegen die letzten (10) Zeilen.

Für das Betrachten kurzer Dateien (deren gesamter Inhalt auf einer Bildschirmseite darstellbar ist) eignet sich auch folgendes Kommando:

```
user@maexen> cat /etc/passwd
```

`cat` schreibt in diesem Beispiel den Inhalt der Passwortdatei auf die Standardausgabe. Es soll nützlich sein, Dateien unter Umständen kopieren zu können. Auch an ein solches Kommando haben die UNIX-Götter gedacht:

```
user@maexen> cp quelle ziel
```

`quelle` und `ziel` sind nun zwei Dateien mit identischem Inhalt.

Eine andere Möglichkeit zum Kopieren bietet die Umleitung von Ausgaben:

```
user@maexen> cat < quelle > ziel
```

Der Unterschied in der Verwendung der Kommandos `cp` und `cat` besteht im Erhalt der Zugriffsrechte der Datei bei ersterem Kommando, während im Falle von `cat` die Zieldatei die durch `umask` voreingestellten Rechte erhält.

Aber warum sollte man Dateien mit gleichem Inhalt mehrfach speichern? Einziger Grund hierfür kann doch nur der Zugriff auf dieselbe Datei aus verschiedenen Verzeichnissen heraus sein. Jedoch wird hiermit Speicherplatz verschwendet. Abhilfe schaffen hier so genannte Links, also Verweise auf Dateien/Verzeichnisse.

```
user@maexen> ln quelle ziel
user@maexen> ln -s quelle ziel2
```

In der ersten Zeile wird ein so genannter statischer (fester) Link angelegt. `ziel` ist jetzt nur ein anderer Name für `datei`; eine Änderung von `ziel` ändert auch den Inhalt von `datei`. Wird einer der Dateinamen gelöscht, kann über den anderen Namen weiter auf den Inhalt zugegriffen werden.



Betriebssystem

Anders verhält sich der Link der zweiten Zeile, ein so genannter symbolischer Link. Im Unterschied zum statischen Link, wo ein neuer Verzeichniseintrag auf den Verwaltungseintrag (Inode) der Ursprungsdatei erzeugt wird, speichert der symbolische Link in einem neuen Inode den Speicherplatz des Verwaltungseintrages der originalen Datei. Löscht man nun *quelle*, existiert *ziel2* weiter, zeigt aber ins "Nichts" (auf einen nicht mehr existierenden Inode), ist sozusagen unbrauchbar geworden. Der Vorteil symbolischer Links besteht in der Möglichkeit, sie auch auf über Partitionsgrenzen hinweg anwenden zu können. Ein Beispiel erläutert den Sachverhalt in Abschnitt Links.

Das Umbenennen bzw. Verschieben von Dateien erfolgt mit dem Kommando `mv -move`:

```
user@maexen> mv quelle ziel
```

Die Datei *quelle* erhält den neuen Namen *ziel*.

2.5 Hände weg von fremden Eigentum

Vielleicht hat man bei den obigen Beispielen schon Bekanntschaft mit Ausgaben wie `permission denied` geschlossen. Wenn nicht, wechselt man z.B. ins `root`-Verzeichnis und versucht einmal eine Datei in diesem anzulegen.

```
user@maexen> cd /
user@maexen> touch example
touch: example: permission denied
```

Der Grund hierfür ist, dass alles (Prozesse, Dateien, Verzeichnisse...) in Linux einen Besitzer hat und dieser bestimmt, wer was damit tun und wer gefälligst was zu unterlassen hat. Ausnahme ist hier wieder einmal der Oberboss - also `root` -, der über alles sein Zepter schwingt und die üblichen Gesetze zu seinen Gunsten außer Kraft setzen kann. Betrachten wir drei typische Einträge aus einem Verzeichnis, so wie sie vom Kommando `ls -l` dargestellt werden:

```
user@maexen> ls -l
lrwxrwxrwx   1   root   root           8   Aug 11 18:40   asm -> asm-i386
-rw-----   1   user   users        8139   Oct 13 07:53   av2.txt
drwxr-xr-x  16   user   users        1024   Oct 19 21:39   tmp
```

Interessant ist hier die erste Kolonne, die die jeweiligen Zugriffsrechte angibt. Das erste Zeichen gibt dabei den Typ der Datei an:

Betriebssystem



Symbol	Bedeutung
-	Alles, außer das Nachfolgende
b	Blockorientiertes Gerät (<i>Device</i>)
c	Zeichenorientiertes Gerät
d	Verzeichnis
l	Symbolischer Link (Verweis)
p	FIFO-Datei (named pipe)
s	Unix domain socket

Es folgen drei Gruppen zu je drei Spalten, die die Rechte

```
-rwxrwxrwx des Eigentümers,  
-rwxrwxrwx der Gruppe und  
-rwxrwxrwx der anderen
```

(in der angegebenen Reihenfolge) bezeichnen:

Symbol	Bedeutung
r	Datei darf gelesen werden
w	Datei darf geschrieben werden
x	Datei darf ausgeführt werden (Binary, Shell-Skript)

Theoretisch sind alle Kombinationen der Zugriffsrechte denkbar, aber der Eigentümer einer Datei darf diese immer lesen, selbst wenn das Lesebit nicht gesetzt ist. Genauso macht es wenig Sinn, eine gewöhnliche Textdatei als ausführbar zu setzen; die Shell wird damit nichts anfangen können. Ein x für ein Verzeichnis gibt an, dass in dieses gewechselt werden kann.

Interessant ist auch ein Konstrukt folgender Art:

```
-rwx---rwx
```

das angibt, dass mit der Datei alles angestellt werden kann, außer für Nutzer derselben Gruppe. Versucht irgendjemand, der nicht der Gruppe des Eigentümers angehört, die Datei zu modifizieren, wird ihm dies gelingen, einem Gruppenmitglied bleibt dies versagt, obwohl es ja gleichzeitig ein "anderer" ist. D.h. die Rechte der Gruppe sind verbindlicher, als die Rechte der anderen!



Betriebssystem

2.6 Hilfe

Unixe verfügen von Haus aus über ein ausgereiftes Hilfesystem, die Manual Pages. Zu nahezu jedem Kommando findet man dort eine ausführliche Beschreibung und jeder Programmierer ist dazu aufgefordert, zu seinen Programmen ebenfalls Manuals zu erstellen.

```
user@maexen> man ls
```

zeigt eine kompakte Beschreibung des Kommandos `ls` mit all seinen Optionen, Argumenten, Informationen zu bekannten Problemen, Verweise zu verwandten Kommandos...

Selbstverständlich gibt es zur Bedienung von `man` selbst ein Manual.

Das Manual eines jeden Kommandos ist einer so genannten Sektion zugeordnet, die Auskunft über den ungefähren Verwendungszweck des Programmes geben soll:

Nummer	Beschreibung der Sektion
1	Programme und Shellkommandos
2	Systemrufe
3	Bibliotheksfunktionen
4	Spezialdateien (Gerätedateien)
5	Dateiformate und -vereinbarungen
6	Spiele
7	Makropakete (z.B. Formatierer <code>groff</code>)
8	Kommandos für Root
9	Kernelroutinen

Die Manuals selbst folgen einem festgelegten Format:

Bezeichnung	Beschreibung
NAME	Kommandoname (bzw. Dateiname...) und Kurzbeschreibung
SYNOPSIS	Aufrufsyntax
DESCRIPTION	Detaillierte Beschreibung der Wirkungsweise aller möglichen Optionen
FILES	Vom Kommando benötigte/modifizierte Dateien
SEE ALSO	Hinweise auf verwandte Kommandos
DIAGNOSTICS	Erläuterungen zu Fehlercodes, die das Kommando zurück liefert
BUGS	Bekannte Fehlverhalten des Kommandos, aber auch Hinweise auf Wirkungen, die gewollt aber ungewöhnlich sind
EXAMPLE	Beispiele zur Verwendung des Kommandos (fehlt leider oft)



Betriebssystem

2.7 Editoren

Unix verfügt über eine Fülle von Editoren. Der Vater aller Editoren ist der [ed](#). Aber nur für masochistisch veranlagte Menschen wird der zeilenweise arbeitende Editor heute noch interessant sein.

Viele mögen über die Benutzer des vi ähnlich denken, jedoch ist der vi unbestritten einer der schnellsten Editoren überhaupt und zumindest in geklonter Form auf jedem Unix-System verfügbar. Das macht ihn zu dem Standardeditor für Systemadministratoren. Komfortabler (und leider auch langsamer) ist der emacs bzw. sein grafischer Verwandter der xemacs.

3 Das Dateisystem:

3.1 Dateien und Verzeichnisse

Wichtigste Merkmale des Linux-Filesystems ext2fs (kurz: ext2) sind:

- Dateinamen können bis zu 255 Zeichen lang sein.
- Groß- und Kleinschreibung werden unterschieden.
- Alle Zeichen sind erlaubt (aber auf Sonderzeichen sollte man besser verzichten). Es ist sogar möglich, einen aus Leerzeichen bestehenden Dateinamen zu bilden:

```
user@maexen> touch " "
user@maexen> ls -l
-rw-r--r--  1 user  users          0  Aug 6 17:33
user@maexen> rm " "
```

- Die Dateilänge ist auf 2 GB beschränkt.

Eine Dateinamenserweiterung wie unter DOS existiert in Linux nicht; der Punkt ist ein ganz normales Zeichen. Dennoch verwenden Archivierungs- und Komprimierungsprogramme wie tar und gzip derartige Suffixe [.tar](#), [.gz](#).



Betriebssystem

3.2 Jokerzeichen

Um sich nur Dateien mit einem bestimmten Namensmuster anzuschauen, verwendet man so genannte Jokerzeichen (Wildcards). Bevor wir uns Beispielen zuwenden, listen wir die Möglichkeiten zur Suchmusterangabe auf:

Muster	Steht für...
?	Genau ein beliebiges Zeichen
*	Beliebig viele (auch 0) beliebige Zeichen
[def]	Eines der Zeichen
[^czx]	Keines der angegebenen Zeichen
[!czx]	Wie oben
[a-f]	Alle Zeichen aus dem Bereich
~	Steht für das Heimatverzeichnis

```
user@maexen> ls /boot/*.map
/boot/System.map

user@maexen> ls -d /[abc]*
/sbin      /var

/var:
X11R6      cron      lock      named     spool     texfonts
adm        games     log       openwin   squid     tmp
...
```

Betrachten wir folgendes Beispiel:

```
user@maexen> rm -r *~
```

Die Option `-r` weist das Kommando `rm` an, rekursiv in allen Unterverzeichnissen Backup-Dateien (Endung `~`) zu löschen. Wir erwarten jetzt, dass tatsächlich auch in allen Unterverzeichnissen die Dateien entfernt wurden, stellen aber fest, dass dies nur im aktuellen Verzeichnis geschehen ist: Der Grund dafür ist die Shell, die für die Auflösung der Jokerzeichen verantwortlich ist. Die Shell betrachtet also nur `*~` und findet entsprechende Muster nur im aktuellen Verzeichnis. `rm` bekommt also nur eine Liste dieser Dateien!

Genau aus diesem Grund funktioniert in Linux (im Unterschied zu DOS) ein Kommando wie

```
user@maexen> cp *.x *.y
```

niemals. Die Shell expandiert zwar alle `*.x`, kann aber `*.y` nicht auflösen und übergibt dieses unverändert an das Kommando `cp`, welches damit aber nichts anzufangen vermag.

Betriebssystem



3.3 Verzeichnisstruktur

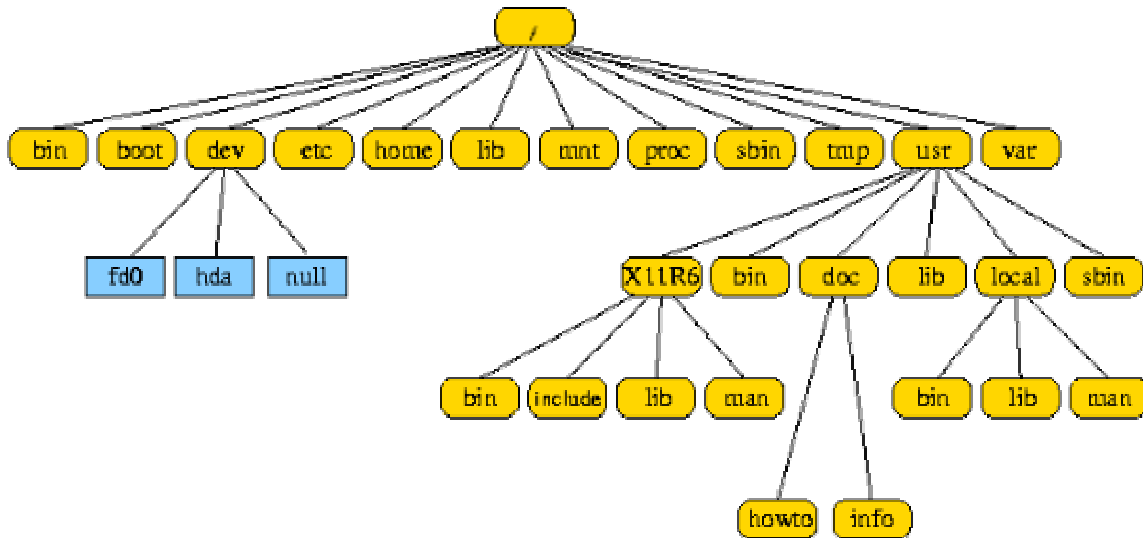


Abbildung: Die Linux-Verzeichnisstruktur (Ausschnitt)

Bei der Unmenge von Dateien in Unix-Systemen, ist eine hierarchische Struktur unabdingbar. Lange Zeit hat jedes Unix-Derivat seine eigenen Vorstellungen vom Aufbau seiner Dateiverwaltung mitgebracht, aber unterdessen ist man sich mehr oder weniger einig geworden und erarbeitete den Filesystem Hierarchie Standard, der wichtige Strukturen definiert. Die meisten Distributionen folgen diesen Richtlinien, wobei Abweichungen oft durch unterschiedliche Auslegung des Standards begründet werden.

Neben der Beschreibung der vorgesehenen Verwendung jedes Verzeichnisses, werden auch konkrete Kommandos genannt, die mindestens in diesen Verzeichnissen vorhanden sein müssen. Des Weiteren finden systemspezifische Vorgaben und optionale Komponenten Erwähnung.

Sobald der Kernel aktiv ist, lädt er als erstes das Root-Filesystem, in dem alle für die Aufgaben des Kernels notwendigen Programme und Konfigurationsdateien angesiedelt sein müssen.

Zu den Programmen gehören:

- Dienstprogramme zum Prüfen und Reparieren des Dateisystems
- Programme zum Sichern der Systemdaten und zur Installation neuer Systemteile
- Eventuell wichtige Netzwerkprogramme

Betriebssystem



Wenden wir uns also zunächst dem Inhalt des Wurzelverzeichnisses zu. Verzeichnisse und Dateien, die als optionale Komponenten im Standard enthalten sind, werden farblich dargestellt.

<code>/bin</code>	Die wichtigsten Kommandos, um mit dem System arbeiten zu können, findet man hier. Sie dürfen von allen Nutzern ausgeführt werden. Zu den Kommandos gehören u.a. <code>bash</code> , <code>cat</code> , <code>cp</code> , <code>dd</code> , <code>gzip</code> , <code>kill</code> , <code>login</code> , <code>ls</code> , <code>more</code> , <code>mount</code> , <code>mv</code> , ...
<code>/boot</code>	Hier findet man die statischen Dateien des Bootmanagers und die Kernel. (In früheren Linux-Versionen wurden der "Haupt"-Kernel im Root-Verzeichnis installiert und nur die optionalen Kernel in diesem Verzeichnis.)
<code>/dev</code>	In diesem Verzeichnis stehen die Gerätedateien (Devices), die die gesamte Hardware beschreiben (Festplatte, Floppy, RAM...), sowie einige Devices mit speziellen Aufgaben. Drei Informationen sind für jedes Device relevant:
<code>/etc</code>	Enthält alle lokalen Konfigurationsdateien (Tastatur, X, Netzwerk...)
<code>/home</code>	Alle Heimatverzeichnisse der Nutzer findet man hier. Nach dem Login landet jeder Benutzer (i.d.R.) in seinem Home. Heimatverzeichnisse können vom Systemverwalter auch an anderer Stelle angesiedelt werden.
<code>/lib</code>	Die beim Systemstart benötigten Bibliotheken stehen hier. Ebenso liegen die Kernelmodule in einem eigenen Unterverzeichnis unterhalb von <code>/lib</code> .
<code>/mnt</code>	Mountpunkt für temporäre Partitionen
<code>/opt</code>	Software, die nicht zum üblichen Installationsumfang von Unix-Systemen gehören, werden oft unter diesem Zweig installiert. So werden nahezu alle kommerziellen Softwarepakete hier eingerichtet
<code>/root</code>	Heimatverzeichnis des Administrators. In realen Unix-Installationen werden die Heimatverzeichnisse aller Nutzer oft auf einem Server gehalten. Bei einem Ausfall eines solchen Servers sollte aber zumindest Root in der Lage sein, vernünftig mit dem System zu arbeiten.
<code>/sbin</code>	Wichtige System-Programme (beim Booten benötigt; Ausführung erfordert Root-Rechte)
<code>/tmp</code>	Temporäre Dateien können hier abgelegt werden, jeder Nutzer ist dazu berechtigt.
<code>/usr</code>	Zweite Hierarchie
<code>/var</code>	Variable Daten



Betriebssystem

3.4 Zugriffsrechte

Zum Ändern der Zugriffsrechte steht das Kommando `chmod` zur Verfügung:

```
chmod [options] <mode> <dateiname>
```

Für die möglichen Optionen sei auf die Manuals verwiesen.

Symbolischer Modus

Symbolisch lassen sich die Rechte setzen durch eine Kombination aus der betreffenden Rechtegruppe:

Symbol	Rechtegruppe
u	Eigentümer (user)
g	Gruppe (group)
o	Andere (others)
a	Alle (all)

und den entsprechenden Rechten:

Symbol	Recht
r	Leserecht
w	Schreibrecht
x	Ausführungsrecht
s	s-Bit setzen
t	t-Bit setzen
+	Recht(e) hinzufügen
-	Recht(e) entfernen
=	Genau diese Recht(e) setzen

Beispiele

```
user@maexen> ls -l hello
-rw-r--r--  1 user  users   108 Apr  7 12:10 hello

user@maexen> chmod a+w hello
-rw-rw-rw-  1 user  users   108 Apr  7 12:10 hello

user@maexen> chmod g-rw,u+xs hello
-rws---rw-  1 user  users   108 Apr  7 12:10 hello
```

Betriebssystem



3.5 Gruppenszwang

Unter Unix muss jeder Benutzer mindestens einer Benutzergruppe angehören. Durch eine solche Gruppenzugehörigkeit lassen sich Zugriffsrechte auf Dateien für bestimmte Anwender ableiten. So kann ein einzelnes Verzeichnis nur für Mitglieder einer Projektgruppe zugänglich gemacht werden und auch für bestimmte Programme lässt sich der Zugriff auf bestimmte Ressourcen durch eine entsprechende Gruppeneinteilung kontrollieren.

Gruppen können durch Passwörter geschützt werden und nur bei Kenntnis dieses kann ein Benutzer sich dieser Gruppe anschließen. Mitglieder einer solchen Gruppe (in der Datei `/etc/group` festgelegt) können ohne Angabe des Passwortes die Zugehörigkeit zu einer Gruppe ändern. Mit dem Kommando:

```
user@maexen> newgrp <group>
```

wechselt der Nutzer die effektive Nutzergruppe. Die Default-Gruppe (der man automatisch nach dem Login angehört) ist in der Datei `/etc/passwd` festgelegt. Nach einem Gruppenwechsel gehören alle neu erstellten Dateien zur Gruppe `<group>`.

Eigene Dateien dürfen anderen Gruppen zugeordnet werden, falls der Besitzer auch Mitglied der neuen Gruppe ist:

```
user@maexen> chgrp <newgroup> <datei>
```

3.6 Besitzerwechsel

Zum Ändern des Eigentümers einer Datei dient das Kommando `chown` (change owner). Aus Sicherheitsgründen darf man allerdings nicht einfach seine Dateien einem anderen Nutzer unterschieben, deshalb erfordert dieses Kommando (meist) Root-Rechte. `chown` kann ebenso zum Wechsel der Gruppe verwendet werden:

```
root@maexen> chown user datei; ls -l datei
-rw-r--r-- 1 user users 0 Apr 7 12:10 datei
```

```
user@maexen> chown root datei
chown: datei: Operation not permitted
```

```
root@maexen> chown root.root datei; ls -l datei
-rw-r--r-- 1 root root 0 Apr 7 12:10 datei
```



Betriebssystem

3.7 Andere Dateisysteme

Linux ist ein Multitalent und versteht die Sprachen vieler Dateisysteme. Voraussetzung für den Zugriff auf diese ist natürlich ein entsprechend konfigurierter Kernel (wird später behandelt), sowie eine aktuelle Version desselben.

Von Linux unterstützte Dateisysteme sind (Auswahl):

ext	Vorgänger des ext2
ext2	Linux
iso9660	CD-ROM
hpfs	OS-2 (nur lesend)
minix	Minix (kaum noch verbreitet)
msdos	DOS
nfs	Network File System
ntfs	WindowsNT
SMB	(Server Message Block) NT, Windows für Workgroups
swap	Swap-Partitionen oder -Dateien
SystemV	Verschiedene Unixe
proc	Prozessverwaltung
umsdos	DOS-Dateisystem unter Linux verwenden
vfat	Windows95
...	...

Um alle unterstützten Dateisysteme unter einem Hut zu vereinigen, ist eine einheitliche Schnittstelle erforderlich. Diese wird vom virtuellen Dateisysteme bereitgestellt.

3.8 Prozesse

Unter Linux können mehrere Prozesse gleichzeitig ausgeführt werden. "Gleichzeitig" bedeutet hier quasi-parallel, d.h. dem Nutzer wird durch die sequentielle Abarbeitung mehrerer Prozesse in kleinen Zeitscheiben die Existenz mehrerer Prozessoren vorgekauelt.

Betriebssystem



Welche Prozesse im Augenblick laufen, verrät uns z.B. das Kommando `ps`:

```
user@maexen> ps ax
  PID TTY STAT TIME COMMAND
    1  ?  S   0:03  init [2]
    2  ?  SW   0:00  (kflushd)
    3  ?  SW<  0:00  (kswapd)
    8  ?  S   0:00  update (bdfush)
   57  ?  S   0:00  /sbin/kerneld
   68  ?  S   0:00  /usr/sbin/klogd -c 1
   70  ?  S   0:00  /usr/sbin/syslogd
   89  ?  S   0:00  /usr/sbin/rpc.mountd
   91  ?  S   0:00  /usr/sbin/rpc.nfsd
  105  ?  S   0:00  /usr/sbin/cron
  110  ?  S   0:00  /usr/sbin/inetd
  112  a0  S   0:00  /usr/bin/gpm -t ms -m /dev/mouse
  115  ?  S   0:00  /usr/sbin/lpd
  127  ?  S   0:01  /usr/sbin/sshd
  129  ?  S   0:00  sendmail: accepting connections on port 25
  144  3  S   0:00  /sbin/mingetty tty3
  145  4  S   0:00  /sbin/mingetty tty4
  146  5  S   0:00  /sbin/mingetty tty5
  160  ?  S   1:54  /usr/X11R6/bin/Xwrapper :0 -bpp 8
 1561  2  S   0:00  bash
 1568  2  R   0:00  ps x
```

Aus obigem Listing sind folgende Informationen abzulesen:

PID	Prozessnummer, jeder Prozess erhält bei seiner Erstellung eine eindeutige PID. Der erste Prozess beim Systemstart erhält die PID 1, der init-Prozess .																		
TTY	Welcher Prozess läuft auf welchem virtuellen Terminal (Konsole)? <code>tty3-6</code> warten auf ein Login (mingetty). Ein Fragezeichen kennzeichnet Prozesse, die kein Terminal kontrollieren.																		
STATE	Zustand des Prozesses: <table border="1"> <tr> <td>D</td> <td>Der Prozess wartet auf einen bestimmten Hardwarezustand (uninterruptible sleep).</td> </tr> <tr> <td>L</td> <td>Die Speicherseite(n) des Prozesses darf nicht ausgelagert werden.</td> </tr> <tr> <td>N</td> <td>Der Prozess läuft mit einer niedrigeren Priorität (ihm wird weniger Rechenzeit als gewöhnlichen Prozessen zugestanden).</td> </tr> <tr> <td>R</td> <td>Der Prozess wartet auf die Zuteilung der CPU (runable).</td> </tr> <tr> <td>S</td> <td>Der Prozess wartet auf das Eintreten eines Ereignisses, z.B. das Beenden eines IO-Transfers... (sleeping).</td> </tr> <tr> <td>T</td> <td>Der Prozess befindet sich im Einzelschrittlauf, z.B. für Debuggingzwecke (traced).</td> </tr> <tr> <td>Z</td> <td>Der Prozess existiert nicht mehr, aber der Vaterprozess hat den Rückgabestatus noch nicht geprüft (zombie).</td> </tr> <tr> <td>W</td> <td>Die dem Prozess zugehörige Speicherseite befindet sich nicht im Hauptspeicher (ist also ausgelagert).</td> </tr> <tr> <td><</td> <td>Der Prozess läuft mit einer höheren Priorität (ihm wird mehr Rechenzeit als gewöhnlichen Prozessen zugestanden).</td> </tr> </table>	D	Der Prozess wartet auf einen bestimmten Hardwarezustand (uninterruptible sleep).	L	Die Speicherseite(n) des Prozesses darf nicht ausgelagert werden.	N	Der Prozess läuft mit einer niedrigeren Priorität (ihm wird weniger Rechenzeit als gewöhnlichen Prozessen zugestanden).	R	Der Prozess wartet auf die Zuteilung der CPU (runable).	S	Der Prozess wartet auf das Eintreten eines Ereignisses, z.B. das Beenden eines IO-Transfers... (sleeping).	T	Der Prozess befindet sich im Einzelschrittlauf, z.B. für Debuggingzwecke (traced).	Z	Der Prozess existiert nicht mehr, aber der Vaterprozess hat den Rückgabestatus noch nicht geprüft (zombie).	W	Die dem Prozess zugehörige Speicherseite befindet sich nicht im Hauptspeicher (ist also ausgelagert).	<	Der Prozess läuft mit einer höheren Priorität (ihm wird mehr Rechenzeit als gewöhnlichen Prozessen zugestanden).
D	Der Prozess wartet auf einen bestimmten Hardwarezustand (uninterruptible sleep).																		
L	Die Speicherseite(n) des Prozesses darf nicht ausgelagert werden.																		
N	Der Prozess läuft mit einer niedrigeren Priorität (ihm wird weniger Rechenzeit als gewöhnlichen Prozessen zugestanden).																		
R	Der Prozess wartet auf die Zuteilung der CPU (runable).																		
S	Der Prozess wartet auf das Eintreten eines Ereignisses, z.B. das Beenden eines IO-Transfers... (sleeping).																		
T	Der Prozess befindet sich im Einzelschrittlauf, z.B. für Debuggingzwecke (traced).																		
Z	Der Prozess existiert nicht mehr, aber der Vaterprozess hat den Rückgabestatus noch nicht geprüft (zombie).																		
W	Die dem Prozess zugehörige Speicherseite befindet sich nicht im Hauptspeicher (ist also ausgelagert).																		
<	Der Prozess läuft mit einer höheren Priorität (ihm wird mehr Rechenzeit als gewöhnlichen Prozessen zugestanden).																		
TIME	Verbrauchte CPU-Zeit des Prozesses																		
COMMAND	Name des Programmes, welches vom Prozess ausgeführt wird																		

Betriebssystem



3.9 Interprozesskommunikation

Signale sind eine Möglichkeit zur Interprozesskommunikation. Vom Kernel verwendet, dienen sie dazu, Prozesse über bestimmte Ereignisse zu informieren; der Nutzer bedient sich ihrer, um hängen gebliebene Prozesse abubrechen oder diese anzuweisen, ihre Konfigurationsdateien neu einzulesen

Signal	Nummer	Aktion
SIGHUP	1	Terminal-Hangup, bei Dämonen verwendet, um ein erneutes Einlesen der Konfigurationsdateien zu erzwingen
SIGINT	2	Tastatur-Interrupt
SIGQUIT	3	Ende von der Tastatur
SIGILL	4	Illlegaler Befehl
SIGABRT	5	Abbruch-Signal von <i>abort(3)</i>
SIGFPE	8	Fließkommafehler (z.B. Division durch Null)
SIGKILL	9	Unbedingte Beendigung eines Prozesses
SIGSEGV	11	Speicherzugrifffehler
SIGPIPE	13	Schreiben in eine Pipe, ohne dass ein Prozess daraus liest
SIGTERM	15	Prozess soll sich beenden (default von kill)
SIGCHLD	17	Ende eines Kindprozesses
SIGCONT	18	Gestoppter Prozess wird fortgesetzt
SIGSTP	20	Ausgabe wurde angehalten
SIGUSR1	30	Nutzerdefiniertes Signal

Zum manuellen Versenden von Signalen dienen die beiden Kommandos kill und killall. kill erwartet im Argument die ID des Prozesses, während killall den Namen des Programmes verlangt. Beide Kommandos verstehen die Signalangabe in numerischer (1 für SIGHUP,..., 9 für SIGKILL,...) als auch in symbolischer (HUP für SIGHUP,..., KILL für SIGKILL,...) Form.

Als Beispiel nehmen wir an, der "Netscape" reagiere nicht mehr (was leider keine hypothetische Annahme darstellt). Also werden wir den entsprechenden Prozess per Hand beenden:

```
# mittels killall auf die höfliche Tour
user@maexen> killall -15 netscape

# reagiert Netscape immer noch nicht, dann auf die harte Tour
user@maexen> killall -KILL netscape

# mittels kill
user@maexen> ps ax | grep netscape
  887 tty1      S      2:12 /opt/netscape/netscape
user@maexen> kill -9 887
```

Betriebssystem



3.10 Das Prozessdateisystem

Das Prozessdateisystem stellt zur Laufzeit die Daten des Kernels in Form eines normalen Dateisystems dar. Als Mount-Point dient normalerweise /proc. Dieses Dateisystem existiert allein im Hauptspeicher und nicht auf der Festplatte!

Ein Blick in das Verzeichnis offenbart den Inhalt:

```
user@maexen > ls /proc
1      157  179  195  243  279  cmdline      kcore      misc      stat
105    160  180  196  244  281  cpuinfo      kcore_elf  modules   swaps
106    161  181  2    245  283  devices      kmsg       mounts    sys
112    162  182  205  246  3    dma          ksyms      net       tty
117    165  183  206  247  4    fb           loadavg    partitions uptime
121    166  184  209  249  5    filesystems  locks      pci       version
136    171  185  227  252  6    fs           lvm        rtc
147    176  186  233  258  74   ide          mdstat     scsi
151    177  187  236  276  78   interrupts  meminfo    self
155    178  188  239  277  bus   ioports     memstat    slabinfo
```

Sinn dieses Abbildes der Kerneldaten ist es, Programmen das Lesen dieser Daten zu ermöglichen, ohne auf den Kernelbereich zugreifen zu müssen (Sicherheit!!!).

Betriebssystem



4 Wichtige Kommandos:

4.1 Kommandos zur Dateiverwaltung

cat	Verkettet seine Argumente und schreibt das Ergebnis auf die Standardausgabe: <pre>user@maexen> cat file.1 file.2 file.3 > file.all</pre>
cd	Wechselt das aktuelle Verzeichnis: <pre># absolute Pfadangabe user@maexen> cd /usr/src # relative Pfadangabe user@maexen> cd ../../usr</pre>
cp	Kopiert Dateien: <pre>user@maexen> cp datei.txt file.txt</pre>
find	Sucht Dateien nach Namen -name, Datum -[a,c]time, Größe -size, Typ -type usw.: <pre>user@maexen> find /usr/include -name "*.h"</pre>
ln	Richtet einen Link ein: <pre># fester Link (Hardlink) user@maexen> ln file.1 link2.1 # symbolischer Link user@maexen> ln -s file1. symlink2.1</pre>
ls	Zeigt den Inhalt eines Verzeichnisses an: <pre>user@maexen> ls /boot System.map boot.b map vmlinuz boot.0803 chain.b os2_d.b</pre>
mkdir	Legt ein neues Verzeichnis an: <pre>user@maexen> mkdir test_dir</pre>
mv	Verschiebt Dateien und ändert ihren Namen:
rm	Löscht Dateien, bzw. entfernt Links auf diese:
rmdir	Löscht Verzeichnisse: <pre>user@maexen> rmdir dir</pre>

Betriebssystem



4.2 Texte Bearbeiten

cut	Extrahiert Spalten aus jeder Zeile eines Textes. <pre>user@maexen> ls -l /boot cut -b 1-11,56- total 718 -rw-r--r-- System.map -rw-r--r-- boot.b</pre>
expand	Ersetzt Tabulatoren durch Leerzeichen. <pre>user@maexen> expand file1 > file2</pre>
fromdos	Konvertiert DOS-Zeileneenden ins Linux-Format.
grep	Sucht Textmuster innerhalb der Eingabe. <pre>user@maexen> ps eax grep bash 167 2 S 0:00 -bash 166 1 S 0:00 -bash TERM=linux HZ=100 HOME=/...</pre>
head	Zeigt die ersten (10) Zeilen einer Datei an. <pre>root@maexen> head -20 /var/log/messages</pre>
less	Zeigt seitenweise Dateien an. Mittels / muster bzw. ? muster kann in der Datei nach muster gesucht werden. less beherrscht eine Unmenge an Optionen und Kommandos. Beendet wird das Programm durch <Q>.
more	Wie less, allerdings können die Cursortasten nicht zur Navigation verwendet werden. Im Text vorwärts scrollt man mit Hilfe der Leertaste; zurück geht's mit .
nl	Nummeriert die Zeilen der als Argumente übergebenen Dateien und schreibt das Ergebnis auf die Standartausgabe .
paste	Vereint mehrere Texte zeilenweise. <pre>user@maexen> paste test1.txt test2.txt Zeile1 aus test1.txt Zeile1 aus test2.txt Zeile1 aus test1.txt Zeile2 aus test2.txt</pre>
recode	Konvertiert zwischen verschiedenen Zeichensätzen.
sed	Stream-Editor (programmierbar, siehe Manual Page).
sort	Sortiert seine Eingabe. <pre>user@maexen> sort file</pre>
tac	"Verkehrtes cat". <pre>user@maexen> tac test1.txt Zeile2 aus test1.txt Zeile1 aus test1.txt</pre>
tail	Zeigt die letzten (10) Zeilen einer Datei an. <pre>user@maexen> tail /var/log/messages</pre>
uniq	Entfernt mehrfach auftretende Zeichen in einer sortierten Datei. <pre>user@maexen> sort testdat uniq eine erste Zeile eine zweite Zeile</pre>

Betriebssystem



4.3 Packen/Entpacken/Archivieren

```
tar | Archivierungsprogramm, das Verzeichnisstrukturen enthält.
user@maexen> tar cvfz archiv.tgz file* dir # Einpacken, Komprimieren

user@maexen> tar tzf archiv.tgz # Inhalt auflisten
/dir/contents
/dir/file
file_01

user@maexen> tar xvfz archiv.tgz # Entpacken
```

4.4 Prozessverwaltung

halt	Beendet Linux und hält den Rechner an. root@maexen> halt
kill	Sendet Prozessen (PID) Signale. user@maexen> kill -SIGTERM 255 user@maexen> kill -15 236
killall	Sendet Prozessen (Name) Signale. user@maexen> killall -SIGTERM xinit
nice	Startet einen Prozess mit veränderter Priorität. user@maexen> nice -n 19 gcc bigprogram.c
nohup	Führt einen Kindprozess unabhängig vom Vater aus. Stirbt der Vaterprozess, werden normalerweise alle seine Nachfahren beendet. Dieses Verhalten wird von nohup unterbunden.
ps	Listet laufende Prozesse auf. user@maexen> ps PID TTY STAT TIME COMMAND 166 2 S 0:00 -bash 169 4 S 0:00 (mingetty)
reboot	Beendet Linux und startet den Rechner neu.
shutdown	Beendet Linux. # reboot in 5 min root@maexen> shutdown -r +5 # sofortiges halt root@maexen> shutdown -h now
top	Listet alle Prozesse auf und aktualisiert per Voreinstellung alle 5 Sekunden diese Liste.

Betriebssystem



4.5 Netzwerk-Administration

ifconfig	Konfiguration eines Netzwerk-Interfaces; wird hauptsächlich beim Systemstart zur Initialisierung benötigt. <pre># aktivieren root@maexen> ifconfig eth0 192.168.10.101 broadcast 192.168.10.255 netmask 255.255.255.0 up # deaktivieren root@maexen> ifconfig eth0 down</pre>
netstat	Abfrage der Netzwerk-Schnittstellen; Anzeige von Statistiken und der Kernel-Routingtabelle.
ping	Sendet ICMP-Pakete mit einem ECHO_REQUEST an einen Netzwerkrechner.
route	Anzeige und Manipulation der Kernel-Routingtabelle.
traceroute	Zeichnet die Route auf, die ein Paket im Netzwerk durchläuft. <i>traceroute</i> sendet dazu je 3 IP-Pakete mit gesetztem Time to life (TTL). Durch Erhöhung des Wertes im TTL-Feld wird ein TIME_EXCEEDED beim jeweils nächsten Host erzwungen. Ein Stern (*) zeigt einen Fehlversuch an.

5 Der Kommandozeileninterpreter “bash“:

5.1 Andere Shells

Viele kluge Köpfe haben sich die Frage gestellt, was eine Shell leisten muss. Entsprechend vielfältig ist die Auswahl an unter Linux verfügbaren Kommandozeileninterpretern.

Sie reicht von Shells mit minimalem Umfang [ash](#), über historische Exemplare [csh](#), [ksh](#) bis hin zu den modernen und weit verbreiteten Shells [bash](#), [tcsh](#).

Sofern installiert, lässt sich auch unter Linux die entsprechende Shell nutzen. Die Loginshell ist durch einen Eintrag in der Passwortdatei festgelegt [/etc/passwd](#).

```
user1:x:500:100:example user:/home/user1:/bin/bash
user2:x:501:100:example user:/home/user2:/bin/tcsh
```

Durch Eingabe des entsprechenden Shellnamens (z.B. tcsh) kann man die aktive Shell wechseln und mittels exit kehrt man in die alte Shell zurück.

5.2 Kommandoeingabe

Der grundlegende Aufbau (nach weit verbreiteter Syntax) eines Kommandos sollte wie folgt aussehen:

```
kommandoname [-Optionen] [Argumente...]
```





Betriebssystem

Optionen werden hierbei mittels "-" eingeleitet und bestehen zumeist aus nur einem Buchstaben. Mehrere Optionen können gruppiert werden ls -la. Dem Kommando können beliebig viele Argumente übergeben werden. Meist können Optionen und Argumente in ihrer Reihenfolge vertauscht werden.

Das führende Minus zur Einleitung von Optionen ist gemäß dem Unix98-Standard. Einige Kommandos wie (ps, tar,...) verwenden darüber hinaus auch Optionen nach der BSD-Syntax, nach der kein führendes Minus vorgeschrieben ist. Ebenfalls Verwendung finden oft die GNU-Regeln für lange Optionen, die durch zwei führende Minus-Zeichen eingeleitet werden.

5.3 Wichtige Tastenkürzel

Die folgende Tabelle fasst die wichtigsten in der Shell verfügbaren Tastenkombinationen zusammen.

	Durch die zuletzt eingegebenen Kommandos scrollen
	Cursor bewegen
<POS 1> , <ENDE>	Cursor an Beginn/Ende der Zeile
<CTRL>+<A> , <CTRL>+<E>	Wie oben
<ALT>+ , <ALT>+<F>	Cursor um ein Wort vor/zurück
<ALT>+<D>	Wort löschen
<CTRL>+<K>	Alles bis Zeilenende löschen
<CTRL>+<T>	Die beiden vorangegangenen Zeichen vertauschen
<ALT>+<T>	Die beiden vorangegangenen Wörter vertauschen
<CTRL>+<L>	Bildschirm löschen
<CTRL>+<R>	Bereits eingegebenes Kommando suchen

5.4 Alias

Zur Abkürzung immer wiederkehrender Kommandofolgen lassen sich für diese so genannte Aliasse definieren:

```
user@maexen> alias rm='rm -i'  
user@maexen> unalias rm
```

Mit alias definiert man einen Alias und mit unalias lässt sich dieser wieder entfernen. Aliasse existieren bis zum Löschen dieser oder bis zum Beenden der aktiven Shell. Möchte man einen Alias permanent einrichten, trägt man die entsprechende Befehlszeile in die Datei .bashrc in seinem Home-Verzeichnis ein. Der Aufruf von alias ohne Argumente bewirkt eine Auflistung aller definierten Abkürzungen.



Betriebssystem

5.5 Pipes

Pipes (Röhren) dienen der Verknüpfung eines Ausgabe- mit einem Eingabestrom:

```
user@maexen> cd /dev
user@maexen> ls -l | less
```

speist die Ausgabe von `ls -l` in die Eingabe von `less` (ohne diese Pipe hätte man keine Chance, die Ausgaben - ca. 1500 Zeilen - am Bildschirm zu verfolgen).

Anstelle von Pipes können auch FIFO-Dateien verwendet werden (kaum gebraucht):

```
user@maexen> mkfifo fifo
user@maexen> ls -l > fifo&
user@maexen> less fifo
user@maexen> rm fifo
```

`ls -l` schreibt nun in die Datei `fifo`, aus dieser `less` seine Daten bezieht. Um die Shell für die nächste Eingabe frei zu bekommen, wurde `ls -l` im Hintergrund gestartet (& bewirkt diesen Effekt).

Alternativ kann man auch wie folgt vorgehen:

```
user@maexen> ls -l > fifo
^z
[1]+  Stopped                  ls --color=tty -l >fifo
user@maexen> bg
[1]+  ls --color=tty -l >fifo &
user@maexen> less < fifo
[1]+  Done                    ls --color=tty -l >fifo
```

<CTRL><Z> stoppt die Ausführung des aktiven Prozesses (es erscheint eine Meldung: ..Stopped...), `bg` lässt den Prozess nun im Hintergrund laufen. Mit Beenden von `less` terminiert auch `ls -l`. Mit `fg` kann man den zuletzt in den Hintergrund beförderten Prozess wieder in den Vordergrund holen (`fg jobnr` holt den Job mit der angegebenen Nummer in den Vordergrund).

Betriebssystem



5.6 Mehrere Kommandos

Die bash unterstützt mehrere Möglichkeiten, mehrere Kommandos nacheinander und in Abhängigkeit voneinander zu starten:

```
user@maexen> ls; date
```

Die Kommandofolge führt zunächst das Kommando ls aus und zeigt dann das aktuelle Datum an.

```
user@maexen> ls; date > datei
```

In datei steht das aktuelle Datum.

```
user@maexen> (ls; date) > datei
```

In datei stehen nun der Verzeichnisinhalt und das aktuelle Datum. Die Klammerung bewirkt die Ausführung der eingeschlossenen Kommandos in derselben Shell, so dass diese ein Ergebnis zurückliefern. Nachfolgende Tabelle fasst die weiteren Möglichkeiten zusammen

<code>komm1 ; komm2</code>	Führt die Kommandos nacheinander aus
<code>komm1 && komm2</code>	Führt komm2 nur aus, wenn komm1 erfolgreich war
<code>komm1 komm2</code>	Führt komm2 nur aus, wenn komm1 einen Fehler liefert
<code>komm1 &</code>	Führt Kommando als Hintergrundprozess aus
<code>komm1 & komm2</code>	Startet komm1 im Hintergrund, komm2 im Vordergrund
<code>(komm1 ; komm2)</code>	Startet beide Kommandos in einer Shell



Betriebssystem

5.7 Programmierung der Bash

Niemand wird auf die Idee kommen, mit den Mitteln der Bash ein Office-System zu programmieren. Für solche Zwecke fehlen der Programmiersprache einfach die Ausdrucksmittel und selbst wenn jemand sich die Arbeit machen würde, wäre ein solches Programm furchtbar langsam...

Sinn der Programmiersprache der Bash ist z.B. die Unterstützung der Automatisierung von wiederkehrenden Befehlsfolgen.

Kommen wir auf ein reales Problem zurück. Wir haben im System mit dem Programm tar 80 Archive erzeugt und diese anschließend mit dem Programm gzip komprimiert.

Die Archive bezeichneten wir mit archiv01 bis archiv80. tar ergänzte das Suffix .tar und gzip hing nochmals .gz hintenan, so dass die Dateinamen nun archivXX.tar.gz lauten.

Um die Archive auf ein anderes (nicht vernetztes) System zu kopieren, bedienen wir uns einer MSDOS-formatierten Diskette und speichern die Archive mittels mcopy archiv*.tar.gz a: auf diese.

Auf dem anderen System speichern wir die Dateien zurück. Zu unserer Überraschung nennen sich diese aber nun archi~XX.gz, da in MSDOS zwei Punkte in einem Dateinamen unzulässig sind und diese maximal 8 Zeichen enthalten dürfen. Unsere Namen wurden automatisch manipuliert...

Wir müssen nun also die ursprünglichen Namen restaurieren. Eine Lösung wäre, achtzig Mal eine Zeile der Art:

```
user@maexen> mv archiv~01.gz archiv01.tar.gz
```

einzugeben... eine mühevoll Arbeit :-(

Alternativ lässt sich die Arbeit von einigen wenigen Shellsript-Zeilen bewerkstelligen:

```
user@maexen> for i in $(ls archiv*.gz); do
> tmp=${i#*~}
> mv $i archiv${tmp%.*}.tar.gz
> done
```

Betriebssystem



6 Der Kernel:

6.1 Nachladbare Funktionalität - Module

Damit die gesamte im System installierte Hardware unterstützt wurde, musste der Kernel früher mit den entsprechenden Treibern konfiguriert und kompiliert werden. Seit Kernelversion 2.0 ist es möglich, fast alle Treiber auch als dynamisch ladbar bereitzustellen. In den meisten Fällen ist die Verwendung von Modulen sogar zwingend notwendig, um die Größe des Kernels soweit zu beschränken, dass er während des Hochfahrens in den konventionellen Speicher geladen werden kann. Diese dynamisch ladbaren Treiber nennt man Module. Die Module können von `root` mit Hilfe des Kommandos `insmod` geladen und mit dem Kommando `rmmmod` wieder aus dem Kernel entfernt werden. Das erheblich komfortablere Kommando, das neben anderem beide Funktionalitäten enthält, ist `modprobe`.

Dieses Kommando lädt Module und erlaubt diesen das automatische Erkennen spezifischer Hardwareeinstellungen wie Interrupt oder Portadresse. Schlägt dieses "Autoprobing" fehl, kann der Administrator die Werte dem Modul fest in der Datei `/etc/modules.conf` zuordnen, die ebenfalls von `modprobe` durchsucht wird. Außerdem beachtet `modprobe` so genannte `module-dependency`, wenn diese mit dem Kommando `depmod -a` erzeugt wurden. Dieses Kommando erstellt eine Datei, die die Zusammenhänge zwischen den Modulen enthält. Das folgende Beispiel zeigt das Laden des Modules für eine DECchip-Tulip-(dc21x4x)-PCI Netzwerkkarte, die hier auf den Interrupt 11 und dem Port 300 konfiguriert wurde. Das Modul heißt "`tulip.o`".

Beispiel

```
root@maexen> modprobe tulip
```

lädt das Module `tulip.o` und alle vom ihm benötigten weiteren Module mit den Parametern aus der Datei `/etc/modules.conf`, wenn diese konfiguriert sind, bzw. mit der "Autoprobing"-Funktion.

```
root@maexen> modprobe tulip io=0x300 irq=11
```

lädt das Module `tulip.o` und alle vom ihm benötigten weiteren Module mit den angegebenen Parametern: Portadresse 300 (`io=0x300`), Interrupt 11 (`irq=11`)

Die geladenen Module können mit dem Kommando `lsmod` angezeigt werden.

```
root@maexen> lsmod
Module          Size  Used by
tulip            14588  1 (autoclean)
serial          42932  0 (autoclean)
memstat         1604   0 (unused)
```



Betriebssystem

Um das Laden von Modulen vom Nutzer fernzuhalten und damit die Arbeit mit dem System zu vereinfachen, wurde der **kernel-Dämon** eingeführt. Dieser lädt die benötigten Module automatisch nach. Der **kernel-Dämon** arbeitet wie **modprobe** auf der Datei `/etc/modules.conf` und der "Autoprobing"-Funktion der Module. Das folgende Beispiel zeigt die Einstellung für die schon erwähnte Netzwerkkarte.

Beispiel

```
# /etc/conf.modules on master
#
# Alias, unter dem die Funktionalität des Modules verlangt wird.
alias eth0 tulip

# Optionen zum Modul
options tulip io=0x300 irq=11
```

6.2 Ein eigener Kernel

Bei jeder Distribution liegt ein Kernel bei, der allen Ansprüchen an Treiber usw. gerecht werden sollte. Durch den Modulsupport können fast alle Treiber dynamisch nachgeladen werden. Aus welchem Grund sollte man also einen eigenen Kernel bauen? Folgende Punkte sprechen für einen neuen Kernel:

- Nur die benötigten Treiber und Grundfunktionen werden eingebunden; das verkleinert (und beschleunigt) den Kernel
- Spezielle Eigenschaften können nur direkt im Kernel konfiguriert werden (z.B. Debug-Unterstützung, Automounter-Funktionalität usw.)
- Der Kernel wird für den entsprechenden Prozessor optimiert
- Neueste Kernel stehen immer als Sourcen, aber erst viel später fertig kompiliert zur Verfügung

Aus diesen Gründen soll hier kurz auf die Konfiguration und das Kompilieren eines eigenen Kernels eingegangen werden. Alle folgenden Befehle können nur als **root** ausgeführt werden.

Um verschiedene Kernelsourcen zu unterscheiden und deren Nebeneinander zu ermöglichen, wird "per Hand" ein Verzeichnis mit der Versionsnummer des neuen Kernels angelegt:

```
root@maexen> mkdir /usr/src/linux-2.2.14
```

und der Link linux auf dieses gerichtet:

```
root@maexen> ln -s /usr/src/linux-2.2.14 /usr/src/linux
```



Betriebssystem

Danach wird der Kernel mit

```
root@maexen> tar xzf linux-2.2.14.tar.gz -C /usr/src/linux
```

Entpackt. Im Verzeichnis [/usr/src/linux](#) (bzw. dort, wohin der Link [linux](#) zeigt) ist der folgende Befehl auszuführen, der eine graphische Oberfläche zur Konfiguration des Kernels zur Verfügung stellt:

```
root@maexen> make menuconfig
```

bzw. unter der grafischen Oberfläche [X](#):

```
root@maexen> make xconfig
```

Jetzt wird der Kernel entsprechend konfiguriert. Zu jedem Menüpunkt ist in dieser Oberfläche eine Hilfe verfügbar. Kompiliert werden der Kernel und alle Module dann mit folgendem Befehl:

```
root@maexen> make dep clean zImage modules modules_install
```

Der fertige Kernel liegt nach dem Kompilieren als Datei [zImage](#) im Verzeichnis [/usr/src/linux/arch/i386/boot](#). Dieser Kernel ist jetzt noch als Bootkernel einzurichten (z.B. im Bootloader [LILO](#)).

Nach der Installation muss das System gebootet werden.

6.3 Linux Boot Loader (LILO)

Der Linux Boot Loader ist ein mächtiges Tool zum Booten von Linux und anderen Systemen.

Der LILO wird entweder in den Master-Boot-Record (MBR) oder an den Anfang einer Partition innerhalb der ersten 1024 Zylinder der ersten oder zweiten Festplatte geschrieben. Diese Restriktionen werden nicht vom Linux, sondern vom BIOS gesetzt.

In diesem Beispiel liegt die root-Partition (hier [/dev/hda5](#)) des Linux-Systems in einer erweiterten Partition (hier [/dev/hda3](#)) der ersten Festplatte. Ein anderes System (z.B. Win95 oder DOS) liegt auf der ersten primären Partition der gleichen Platte.

Installiert wird der Bootloader durch Aufruf des Kommandos `lilo`.

Die obige Konfiguration setzt voraus, dass das Booten aller Betriebssysteme vom Lilo aus steuerbar ist. Leider streikt bei derartiger Einstellung häufig der NT-Bootmanager, so dass man diesen zum Booten von Linux erziehen muss.

Für einen solchen Fall installiert man den Lilo nicht in den MBR, sondern in den Bootsektor der Rootpartition und konfiguriert die `lilo.conf` nur für die Linux-Images.



Betriebssystem

Anschließend bootet man mit Hilfe einer Bootdiskette sein Linux und kopiert den Bootsektor in eine Datei (wir nehmen an, die Linux-Rootpartition ist /dev/hdb2):

```
root@maexen> dd if=/dev/hdb2 of=/tmp/bootsector bs=512 count=1
```

Die Datei `bootsector` speichert man auf eine von NT lesbare Diskette, z.B. eine DOS-formatierte Diskette:

```
root@maexen> mcopy /tmp/bootsector a:
```

Als Nächstes ist NT zu booten. Als Systemadministrator speichert man nun die Datei von der Diskette in ein NT-Verzeichnis und ergänzt die Datei `boot.ini` um einen entsprechenden Eintrag:

```
c:\bootsector="Linux"
```

6.4 Plug & Play

Plug and Play ist eine feine Sache... Dem Slogan Karte einbauen, Computer starten, fertig glauben ohnehin nur noch die größten Optimisten. Auch wenn für Otto Normalverbraucher mit seiner einen Plug-and-Play-Karte tatsächlich der Administrationsaufwand schwindet, erweist sich die vom BIOS vorgenommene Initialisierung schnell als Falle, wenn sich in einem Rechner gleich mehrere solcher Karten um die Zuweisung der wenigen freien Interrupts und Adressen bewerben. Während für Windows, DOS und Co. die Treiber für solche Karten noch mitgeliefert werden, müssen dieselben von Freaks in ihrer Freizeit erst noch für Linux entwickelt werden. Und während der Hersteller einer Karte noch weiß (wissen sollte), für welche Adressen und Interrupts er seine Treiber zurechtbastelt, beschränken sich die Linux-Treiber meist nur auf ausgewählte ("die normalen") Bereiche. Weist das BIOS einer Karte jetzt Ressourcen zu, die der Treiber nicht vorsieht, wird dieser die Hardware nicht entdecken...

Nun bleiben einem als Linux-Guru drei Offerten übrig:

- Plug and Play abschalten... funktioniert nur bei manchen Karten
- Andere Karte kaufen... funktioniert nur bei manchen Geldbeuteln
- Das Tool `isapnp` verwenden (das Paket liegt den meisten Distributionen bei)

Die obige Beschreibung gleicht eher einer komplizierten Bastelanleitung...

Speziell zur Administration von Plug&Play-Soundkarten hat RedHat ein Frontend geschaffen, das automatisch die korrekten Einstellungen in der Datei `/etc/isapnp.conf` vornimmt.



ENDE



<http://www.google.ch/linux>