Linux für Sinnlos-Abhängige

Michael Dienert

16. November 2008

Inhaltsverzeichnis

1 Muss das unbedingt sein?

Bei der Einführung in Linux musste ich mich oft fragen lassen, wozu denn das Erlernen dieses Betriebssystems gut sein soll. In der Praxis würden doch ohnehin alle mit Windows arbeiten und daher wäre ein WindowsXP-(Vista-)Kurs viel besser.

Warum Ihr aber doch GNU/Linux kennenlernen solltet, hat folgende Gründe:

- Viele grosse Soft- und Hardwarefirmen investieren bedeutende Summen in Linux-Projekte. An erster Stelle ist hier natürlich IBM zu nennen, aber auch HP, SUN, Oracle usw. fördern das GNU/Linux Projekt so dass man annehmen kann, dass die Bedeutung von GNU/Linux in Zukunft doch ziemlich zunehmen wird.
- Windows ist das Produkt *einer* Firma und reine Produktschulung sollte man in der Schule eigentlich nicht machen.
- Windows ist auch nur deshalb so weit verbreitet, weil die meisten Anwender illegale Kopien vom Betriebssytem und dem Officepaket verwenden. [1]
- GNU/Linux dagegen ist ein freies Betriebssystem von dessen Verbreitung weltweit alle profitieren und das legal kostenlos weitergegeben werden darf (soll!).
- GNU/Linux ist sehr gut dokumentiert. Es bringt seine eigene Dokumentation gleich mit. Dadurch eignet sich GNU/Linux sehr gut, um die Standardaufgaben und Komponenten eines Betriebssystems zu erlernen [2].
- GNU/Linux ist ein UNIX-Betriebssytem. Wer mit GNU/Linux umgehen kann, findet sich auch leicht in jedem anderen UNIX zurecht. UNIX ist inzwischen *das* Standardbetriebssytem für alle Workstations und Grossrechner. Beispiele: SOLARIS der Firma SUN, AIX von IBM, SINIX von Siemens.

  Und wer Mac-Fan ist, wird feststellen das Apples neues Betriebssytem OSX ein reines UNIX-System ist [3].

  Nun kommt man als Fachinformatiker oder Systemelektroniker eher selten an die Konsole eines Grossrechners. Viel wahrscheinlicher ist aber, dass man einen Router von Cisco oder HP oder dergleichen konfigurieren muss: diese Geräte besitzen eine Unix-ähnliche Firmware die man von einer Textkonsole aus konfiguriert. D.h., wer Unix bedienen kann, kommt auch schnell mit einem Router zurecht.

[1] Mit OpenOffice gibt es inzwischen aber ein Officepaket, das immerhin ein 2005 von der ISO spezifiziertes Datenformat besitzt (ODF, OpenDocumentFormat). Der Standard Office Open XML von Microsoft wurde dagegen am 5. September 2007 von der ISO abgelehnt!

[2] Linux entstand in Anlehnung an Minix, einem Mini-Unix, das der amerikanische Professor **Andrew Tanenbaum** zu Lehrzwecken für seine Studenten in Holland geschrieben hat. Er arbeitet an der Universität Amsterdam und hat eine sehr lesenswerte Homepage.

[3] Der Betriebssystemkern Darwin basiert auf Free-BSD und ist Open Source. Die meisten GNU/Linux Programme sind bereits nach Darwin portiert.

- Das gesamte Internet kommt aus der UNIX-Welt. TCP/IP, Telnet, FTP, HTTP und SMTP sind Bestandteile eines jeden UNIX-Systems.
- Weltweit ca. 50% aller Web-Server verwenden das Programm Apache, das bevorzugt auf GNU/Linux Rechnern installiert wird.

  Alles was man braucht, um einen leistungsfähigen Web-Server aufzubauen sind freie Programme (Apache, Perl, PHP, mySQL ...), die in jeder GNU/Linux Distribution enthalten sind.
- Da UNIX von Anfang an für Netzwerkbetrieb ausgelegt wurde, ist GNU/Linux ein ideales System, um alle Details der Konfiguration und des Betriebs eines TCP/IP Netzes zu erforschen.
- Die Konfiguration aller UNIX Programme wird mit Hilfe von editierbaren Textdateien erledigt. Das hat gegenüber der Konfiguration eines Windows-Programms mit unzähligen Mausklicks in zahllosen Menues und Untermenues Vorteile: Man kann die Konfigurationsdateien kopieren, an Kollegen weitergeben, ändern, die Vorversion wieder aktivieren und damit Änderungen rückgängig machen und man kann auch jeden Eintrag, den man in einer Datei vornimmt durch einen Kommentar beschreiben.
- Das Arbeiten mit Kommandozeilenprogrammen hat unter UNIX Tradition. Maus-Verwöhnte Windows Anwender fluchen dagegen regelmässig über die Arbeit mit der Tastatur. Wer sich aber einmal in die Bedienung eines Systems mit Kommandozeilenprogrammen eingearbeitet hat, kann dieses Wissen z.B. auch bei der Administration eines Windows-Netzwerks einsetzen. Aussserdem hat sogar Microsoft die Zeichen der Zeit erkannt und bietet inzwischen ein Unix an, das sich auf den Windows2000-Kernel aufsetzen lässt.

Diesen ganzen Vorteilen steht jedoch ein schwerwiegender Nachteil gegenüber:

Die vielen Programme des Betriebssytems und die Anwendungen wurden *unabhängig* voneinander von verschiedenen Programmierern und zu verschiedenen Zeitpunkten entwickelt. Das führte dazu, dass die Benutzerschnittstellen der Programme **nicht** alle einheitlich wurden.

Das macht vor allem für den Anfänger das Arbeiten mit GNU/Linux oft unnötig schwierig. Es kommt einfach kein Spass auf, wenn man vor dem Tippen jedes Befehls erstmal eine Beschreibung lesen muss.

Arbeitet man länger mit dem System, erkennt man aber nach und nach doch so eine einigermassen einheitliche Idee hinter den Programmen und dann fängt die Sache auch an Spass zu machen.

2 Was ist Linux und was ist es nicht

2.1 Der Linux-Kernel

Heutzutage weiss fast jeder etwas mit dem Begriff Linux anzufangen. Und auch jeder kennt seinen Urheber, eben Linus Thorvalds. Leider wissen aber die wenigsten, dass Linux nur ein winzig kleiner Teil eines Rechners ist, der “unter Linux” läuft. Dieser winzige Teil hat unter anderem folgende Eigenschaften und Aufgaben:

- Linux ist der sogenannte *Kernel* des Betriebssystems, der Betriebssystem*kern*.
- Der Kernel ist nur ca. 1 MByte gross (je nach hinzukompilierten Modulen). Auf dem System, auf dem die erste Version dieses Texts entstand, waren es genau 985 303 Bytes. [4] Eine komplette Installation mit allem drum und dran braucht einige hundert MByte Platz. Man kann auch leicht einige GByte Plattenplatz belegen, wenn man sehr viele Programmpakete installiert.
- Linus Thorvalds ist für die Kernelentwicklung verantwortlich und hat auch den größten Anteil an seiner Entwicklung.
- Der Kernel stellt die Schnittstellen zwischen der Hardware und den Programmen, die auf dem System laufen zur Verfügung.
- Den Kernel kann man sich daher wie eine Schale vorstellen, die die Hardware umhüllt. Die Aussenseite dieser Schale stellt eine *einheitliche* Schicht dar, die *unabhängig von der darin verborgenen Hardware* ist.
- Durch dieses Schalenprinzip wird es möglich, die meisten Unix-Standardprogramme auf den unterschiedlichsten Rechnerplattformen laufen zu lassen. Man muss bei der Programmierung eines Kernels für eine neue Plattform nur sicherstellen, dass die Schnittstellen des Kernels zu den Programmen hin einheitlich und passend sind.
- Der Kernel sorgt auch für eine Verteilung der Prozessorzeit an mehrere scheinbar parallel laufende Programme, sog. Prozesse.
- Der Kernel prüft bei Dateizugriffen, ob die Benutzerrechte eingehalten werden.

2.2 GNU is not Unix und GPL

Der ganze Rest einer "Linux"-Installation, also alle Programme, die das Arbeiten mit dem Rechner überhaupt erst möglich machen, gehören nicht zum eigentlichen Linux-Kernel. Viele von ihnen sind sogar älter als Linux selbst. Allen gemeinsam ist, dass sie nicht-kommerziell sind und unter der GNU Public License stehen. Das bedeutet:

- GNU = GNU Is Not Unix, eine rekursive Abkürzung. GNU ist ein Projekt des Programmierers Richard Stallman, gegründet 1983. Ziel von GNU ist es, freie Software zu entwicklen. Ursrprünglich nur freie Software für Unix-Systeme, aber inzwischen gibt es viele GNU-Programme auch für Macs und Sinnlos.
- Die GNU Public License ist ein Vertragswerk aus 12 Paragraphen. Kernpunkt dieser Lizenz ist die Auflage, den Quellcode eines unter der GPL stehenden Programms offenzulegen (Open Source).
- Verwendet ein Programm Teile eines unter der GPL stehenden Programms, ist man *vertraglich gezwungen*, dieses Programm ebenfalls unter die GPL zu stellen.
- Die Verpflichtung zur Offenlegung des Quellcodes ist ein Angebot an andere Personen, dieses Programm zu verbessern und zu erweitern und ebenfalls wieder unter die GPL zu stellen.

[4] Die aktuelle Version des Texts wird auf einem PowerBook G4 geschrieben, dessen Kernel 3 858 620 Bytes gross ist. Allerdings ist dies kein Linux- sondern ein Darwin-Mach-Kernel.

- Linus Thorvalds hat seinen Kernel selbstverständlich auch unter die GPL gestellt.
- Programme die unter der GPL stehen, dürfen nicht verkauft werden. Man darf nur einen kleinen Geldbetrag für das Anfertigen von Datenträgern (z.B. CDs) verlangen.

2.3 Das GNU/Linux Betriebssystem

Wenn man nun ausschliesslich vom Linux-Betriebssystem spricht, würde man die Arbeit vieler 1000 freier Programmierer nicht berücksichtigen. Aus diesem Grund ist (auch nach Meinung des Autors) die korrekte Bezeichnung des Betriebssystems **GNU/Linux**.

Linus Torvalds ist da anderer Meinung und bezeichnet mit Linux das ganze Betriebssystem, was ja auch allgemeiner Sprachgebrauch ist.

Hier das Glaubensbekenntnis der Kirche von Sankt IGNUcius (Richard Stallman). Durch 3-maliges Aufsagen wird man Mitglied der Emacs-Kirche:

> There is no system but GNU, and Linux is one of its kernels.

Das Betriebssystem GNU/Linux besteht aus dem *Kernel* Linux und einer Vielzahl von freien Anwendungsprogrammen.
Der Kernel selbst und alle Dienstprogramme des Betriebssystems stehen unter der GNU Public License. Das bedeutet u.a., dass ihre Quelltexte öffentlich sein müssen.

Auf der Installation im Raum R25 sind weit über 1500 Programme verfügbar.

Neben den Programmen, die unter der GPL stehen, enthalten die CDs der Distributionen oft noch Programme, deren Hersteller den Quellcode nicht offenlegen möchten und die evtl. auch nicht kostenlos sind. Beispiele:

- Der Quellcode von SUNs **JDK** (Java Development Kit) ist nicht frei zugänglich, die Benutzung des JDK ist aber gratis.
- Das Grafik-Konvertierprogramm `xv` ist Shareware, d.h. der Autor möchte einen kleinen Geldbetrag für die Lizenz.

2.4 GNU/Linux Distributionen

Wie oben erklärt, müssen die Quelltexte aller OpenSource Programme veröffentlicht werden. Die Standardprogrammiersprache ist dabei C/C++ mit dem Compiler `gcc`. Dabei bedeutete frueher `gcc` natürlich **GNU C-Compiler**. Jetzt heisst es **GNU Compiler Collection**, da gcc inzwischen c, c++, java, fortran und ada übersetzen kann.

Nun kann sich also jeder ein GNU/Linux Betriebsystem bauen, wenn er einen `gcc` hat und sich aus dem Internet alle gewünschten Quellpakete besorgt hat.

Das ist natürlich sehr mühsam und verlangt einiges an Hintergrundwissen. Aus diesem Grund gibt es Programmsammlungen mit fertig compilierten Programmen. Derartige Programmsammlungen nennt man *Distributionen*.

Verbreitete Distributionen in Deutschland sind z.B.:

1. Knoppix

2. (K)(X)Ubuntu / Debian
3. RedHat
4. Novell (ehem. SuSE)

Weltweit gibt es mehrere Hundert Distributionen, die zum Teil länderspezifisch sind. China hat z.B. seine eigene Red Flag Distribution. Dort wird GNU/Linux anstelle von Wintendo in öffentlichen Einrichtungen verwendet.

2.5 Die grafische Oberfläche

Das weltweit meistverwendete Betriebsystem hat seinen Namen von seiner grafischen Benutzeroberfläche, die mit Fenstern und der Maus bedient wird.

Fenster und Mäuse sind natürlich keine Erfingung von Microsoft. Die Idee einer grafischen Benutzeroberfläche hatte auch nicht Apple, sondern die Firma Rank Xerox aus Palo Alto, Kalifornien. Apple hat die Idee allerdings 1983 zur Marktreife weiterentwickelt.

Auch bei Windows gibt es einen Betriebssytemkern (s.u.). Bei Windows9x/ME ist das das Uralt-Dos und der Kern von NT/2000/XP ist eine Weiterentwicklung des VMS-Systems der Firma *Digital Equipment* (**DEC**). Microsoft hatte dazu einfach **Dave Cutler**, den Entwickler von VMS von DEC "weggekauft". [5]

Die Rechte am Quellcode des Original-Unix besitzt inzwischen die Firma Novell.

Mit *Windows* ist nun im Sprachgebrauch und auch offiziell das gesamte Betriebssytem und nicht nur dessen grafische Benutzerschnittstelle (GUI, Graphical User Interface) gemeint. Auch bei GNU/Linux gibt es grafische Benutzerschnittstellen. Und zwar nicht nur eine sondern einige Dutzend. Die bekannteste in Deutschland dürfte KDE (Kool Desktop Environment) sein, da KDE von SuSE als Standard Desktop gefördert wird und SuSE bei uns eben am weitesten verbreitet ist. Weitere bekannte GUIs sind Gnome, fvwm und WindowMaker.

> KDE, Gnome, fvwm und WindowMaker sind *grafische Benutzerschnittstellen*(**GUI**). Sie sind nicht Teil des Betriebssystems, sondern erleichtern nur dessen Bedienung.

Völlig falsch ist es also, die grafische Oberfläche mit Linux gleichzusetzen!

2.6 Alternativen zu Linux

Linux ist natürlich nicht der einzige Unix-Kernel. Neben Linux sind die Kernel der 3 BSD-Unixe - FreeBSD, NetBSD und OpenBSD - am weitesten verbreitet. Der von Mac-OSX verwendete Darwin-Kernel ist eine Fortentwicklung des FreeBSD-Kernels. BSD steht als Abkürzung für **Berkeley System Distribution**. BSD Unixe bilden einen Hauptzweig im Unix-Stammbaum.

[5] Zur Frage des geistigen Eigentums: Da der Arbeitgeber von Dave Cutler, **DEC** später von Compaq und Compaq inzwischen von HP übernommen wurde und Dave Cutler bei der Entwicklung des NT-Kernels ja garantiert auf Wissen aus seiner DEC-Zeit zurückgriff, könnte man ja zum Schluss kommen, dass die Rechte an Windows teilweise im Besitz von HP sein müssten...

Auch das GNU-Entwicklerteam versucht sich seit Jahr und Tag an einem eigenen Kernel: **Hurd** genannt. Hurd ist ein sehr modern konzipierter Kernel mit wirlich zukunftsweisenden Ansätzen. Leider wurde und wird seine Entwicklung durch den Erfolg von Linux stark gebremst. Es gibt immernoch eine Art Glaubenskrieg darüber, ob Hurd mit seiner Micro-Kernel-Architektur fortschrittlicher sei als der monolithische Linux-Kernel. [6]

Ein weiterer moderner Kernel ist der NT/Windows-Kernel! Jetzt wird sich sicher mancher fragen, wieso der Windows-Kernel hier bei den anderen Unix-Kerneln aufgeführt wird: Auf den Window-Kernel lässt sich problemlos ein Unix-Betriebssystem aufsetzen! Und dies sogar kostenlos: bei Microsoft kann man nach einer Registrierung das *Services For Unix* (SFU) herunterladen. Das ist ein waschechtes Unix mit bash, gcc, vi, emacs und X11 und allem was dazugehört.

Die Version SFU 3.5 gibts kostenlos hier:

`http://www.microsoft.com/windows/sfu/`

3 Überlebensregeln für Linux Einsteiger

Wer zum erstenmal mit Linux arbeitet, muss folgende Regeln beachten:

- GNU/Linux ist ein Mehrbenutzersystem. D.h. um damit arbeiten zu können, muss man sich zwingend beim System anmelden.
- GNU/Linux kennt mehrere sog. *Konsolen*. Eine Konsole bietet den Funktionsumfang eines Terminals: Tastatur und Bildschirm evtl. auch mit Mausunterstützung. GNU/Linux ist nach einer Installation so voreingestellt, dass es 6 verschiedene **Text**-Konsolen und eine **grafische** Konsole gibt. Mit den Tastenkombinationen CTRL-ALT-F1 bis CTRL-ALT-F6 kann man in eine der 6 Textkonsolen wechseln. Mit CTRL-ALT-F7 wechselt man in die grafische Konsole.

  Die grafische Konsole steht aber erst zur Verfügung, wenn ein spezielles Dienstprogramm, der **X-Server** gestartet wurde. Falls der X-Server nicht automatisch nach der Benutzeranmeldung mitgestartet wird, muss man ihn durch Eingabe des Kommandos `startx` nach dem Anmelden von Hand starten. Auf einem System läuft in der Regel nur ein X-Server und dieser ist auch für den Benutzer reserviert, der ihn gestartet hat.
- Hat man selbst ein GNU/Linux installiert, ist man der Administrator dieses Systems. Der Administrator hat immer den Benutzernamen root. Man sollte sich aber nur dann als root anmelden, wenn man wirklich Systemdateien verändern möchte. Für gewöhnliche Arbeiten, auch mit dem eigenen Rechner, sollte man unbedingt einen Benutzer anlegen, der mit Standard-Rechten arbeitet.
- Achtung:

GNU/Linux unterscheidet streng nach Gross- und Kleinschreibung!!

[6] Bei einem monolithischen Kernel sind alle Gerätetreiber Teil eines einzigen Kernel-Programms.

- GNU/Linux kennt keine Laufwerksbuchstaben. Alle Laufwerke werden in einen einzigen, grossen Verzeichnisbaum eingebunden. Dieses Einbinden nennt man mounten (Vergl. Kap. 5.1). Das gilt auch für Floppies und CDs! Eine gemountete Floppy darf man nicht einfach auswerfen, das könnte zu Datenverlusten führen.
- Die Festplattenlaufwerke des Rechners werden beim Systemstart automatisch in den Verzeichnisbaum eingebunden. Vor dem Ausschalten des Rechners müssen sie auch wieder sauber ausgetragen werden. Deshalb darf man einen Rechner mit GNU/Linux **keinesfalls** einfach ausschalten, sondern muss ihn vorher herunterfahren (shutdown). Dazu gibt es Befehle, die aber nur root ausführen darf (init 0, shutdown, halt). Damit auch gewöhnliche Benutzer den Rechner herunterfahren können, drückt man gleichzeitig die Tasten Strg-Alt-Entf (Ctrl-Alt-Del). Sobald die Meldung `Power Down` oder `System Halted` erscheint, darf der Rechner ausgeschaltet werden.
- Ein kleiner Ast des Verzeichnisbaums gehört ganz und gar dem jeweils angemeldeten Benutzer. Dieses Unterverzeichnis wird Heimatverzeichnis (home directory) des Benutzers genannt. Der Benutzer ist Eigentümer aller Dateien und Verzeichnisse in- und unterhalb seines Heimatverzeichnisses und hat deshalb dort sämtliche Schreib- und Leserechte.

3.1 Besonderheiten des Rechnernetzes an der Walther-Rathenau-Gewerbeschule

Die Benutzer des Rechnernetzwerks der wara werden zentral in einer LDAP-Datenbank gespeichert. LDAP steht für *Lightweight Directory Access Protocol*. Eine LDAP-Datenbank ist ein *Verzeichnisdienst* (Directory Service), der zentral die Anmeldeinformationen der Benutzer verwaltet und beim Anmelden an einem Hostrechner über das Netz zur Verfügung stellt.

Das Heimatverzeichnis jedes Benutzers liegt ebenfalls zentral auf einem Fileserver und wird mit Hilfe des smb-Protokolls auf der Arbeitsstation zur Verfügung gestellt, an der man sich anmeldet.

Dabei ist es egal, ob man Windows oder GNU/Linux startet, im einen Fall hat man sein Heimatverzeichnis unter *Eigene Dateien*, im anderen Fall ist das Heimatverzeichnis in den Verzeichnisbaum (siehe unten) eingebunden und erscheint hier (am Beispiel fuer den Benutzer *alfred* in der Klasse *e5it6t*):

```
/home/students/e5it6t/alfred
```

4 Das wichtigste Programm von GNU/Linux

Das wichtigste Programm von GNU/Linux heisst `bash` . `bash` ist eine Abkürzung für **B**ourne **A**gain **Sh**ell. Die `bash` ist eine Weiterentwicklung der Bourne-Shell.

Shell heisst auf deutsch Muschel oder Schale. Das soll verbildlichen, dass die Shell als weitere Schicht das Betriebssystem (also Kernel und Systemprogramme) umschliesst. Die `bash` wird damit zu einer Anpassungsschicht zwischen dem Benutzer und dem Betriebssystem. Abb. 1 zeigt dieses Modell.

Wer sich noch mit DOS auskennt, kennt bereits eine derartige Schicht: es ist das Programm command.com. Und unter Windows gibt es das Programm `cmd.exe`.

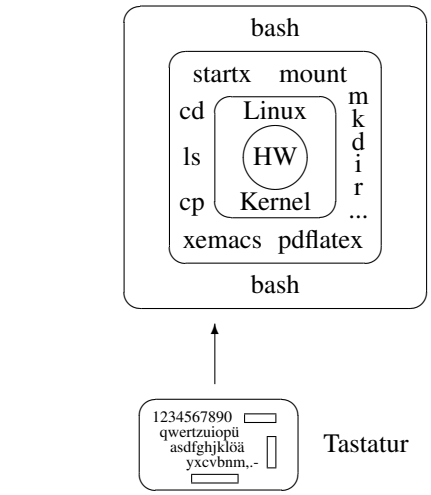

Abbildung 1: Kernel und Bourne Again Shell `bash`

Die `bash` ist aber ungleich vielseitiger und mächtiger als `command.com` bzw. `cmd.exe`. Und wie schon oben gesagt: `command` und `cmd` ignorieren Gross- und Kleinschreibung, **die `bash` unterscheidet hier aber sehr streng**: Das Kommando `cd` zum Wechseln eines Verzeichnisses kann man unter DOS als `CD, cD, Cd` oder `cd` eingeben, `bash` versteht aber nur die kleingeschriebene Version cd.

Hauptaufgabe der `bash` ist es nun, Befehle des Benutzers entgegenzunehmen und zu prüfen, ob die Eingabe mit einem Programm des Systems übereinstimmt und wenn ja, dieses Programm zu starten.

4.1 Arbeiten mit der `bash`

Mit der `bash` kann man alle Arbeiten erledigen, die man auch mit einem grafischen Dateiverwaltungsprogramm wie dem Windows Explorer erledigen kann. Allerdings lassen sich Dateien und Verzeichnisse nicht mit der Maus verschieben und kopieren, sondern das muss man mit den Programmen

- `mv` : move
- `cp` : copy
- `cd` : change directory
- `mkdir` : make directory
- `rm` : remove

erledigen.

4.1.1 Die 4 wichtigsten Tasten

Da die `bash` nun hauptsächlich mit der Tastatur bedient wird, muss man einigermassen gut tippen können, um flüssig arbeiten zu können.
Und damit die Tipperei noch schneller geht, gibt es einige wichtige Tasten, die einem viel Tipparbeit ersparen:

1. ENTER

   Die allerwichtigste Taste ist natürlich die ENTER-Taste.

2. CURSOR-HOCH

   Wenn man mit ENTER ein Kommando abgeschickt hat und nun feststellt, dass irgendwo ein Fehler ist, muss man die Zeile nicht nochmals tippen: mit der Taste CURSOR-HOCH kann man den letzten Befehl wiederholen. Bei mehrmaligem Betätigen von CURSOR-HOCH erreicht man alle Kommandos, die man in der Vergangenheit abgeschickt hatte. Wie viele Kommandozeilen von der `bash` gespeichert werden, lässt sich konfigurieren (Datei `~/.bashrc`).

3. CURSOR-RUNTER

   Mit der Taste CURSOR-RUNTER kann man in der Kommandozeilen-Geschichte wieder Richtung Gegenwart blättern.

4. TAB

   Die TAB -Taste sorgt dafür, dass die Arbeit mit der `bash` überhaupt erst Spass machen kann: Die TAB -Taste *expandiert* Programm-, Datei- und Verzeichnisnamen. Und mit expandieren ist *vervollständigen* gemeint: Drückt man TAB , versucht die `bash` den jeweils benötigten Namen automatisch zu vervollständigen. Welcher Name vervollständigt wird, hängt davon ab, an welcher Stelle einer Kommandozeile TAB betätigt wird. Im nächsten Unterkapitel (Kap. 4.1.2) steht dazu noch Näheres.

4.1.2 Kommandozeilen und Kommandoexpansion

Eine Kommandozeile ist ein Programmname gefolgt von einem oder mehreren Parametern. Allgemein sieht das so aus:

kommandoname ␣ *optionen* ␣ *pfad*

Dabei steht *pfad* für eine Datei oder ein Verzeichnis, ggfs. mit einer relativen oder absoluten Pfadangabe, wo sich die Datei oder das Verzeichnis befinden.

Die Optionen müssen nicht immer vorhanden sein und es gibt sogar Kommandos, die ohne Parameter aufgerufen werden (z.B. `ps`: processes).
Ganz wichtig sind die Leerzeichen: ␣ . Sie dienen der `bash` als Trennzeichen. Z.B. wechselt die `bash` nach der Eingabe von `cd` ␣ `..` ins darüberliegende Verzeichnis. Quittiert aber die Eingabe von `cd..` mit

```
bash: cd..: command not found
```

. Das heisst, die `bash` sucht nach einem Kommando, das `cd..` heisst (eben ”cd..” als Ganzes) und dieses Kommando gibt es normalerweise nicht.

Die Wirkung der TAB -Taste ist nun innerhalb einer Zeile so:

- Ist man gerade bei der Eingabe des Kommandonamens und drückt die TAB -Taste, vervollständigt die `bash` den Kommandonamen, falls dieser schon eindeutig ist.

  Beispiel: nach Eingabe von `konq` TAB drücken und die `bash` vervollständigt zu `konqueror`.

  Drückt man bereits nach der Buchstabenfolge `ko` auf TAB , piepst der Rechner zur Warnung, dass die Eingabe noch nicht eindeutig ist. Drückt man dann nochmals TAB , werden alle Kommandos gelistet, die mit `ko` beginnen. Dabei sucht die `bash` alle Verzeichnisse ab, die im Suchpfad (`$PATH`) stehen. Nun kann man durch Eingabe von weiteren Buchstaben die Eingabe genauer machen, bis man das gewünschte Kommando dastehen hat oder TAB es vollständig expandiert.

- Ist man bei der Eingabe eines Verzeichnis- oder Dateinamens innerhalb einer Kommandozeile, versucht die `bash` diese ebenfalls zu vervollständigen. Das funktioniert wie beim Kommandonamen, nur sucht die `bash` nun nach *Datei- oder Verzeichnisnamen*.

`.`	das aktuelle Verzeichnis
`..`	das übergeordnete Verzeichnis
∼	das Heimatverzeichnis des angemeldeten Benutzers
`/`	das Wurzelverzeichnis und Pfadtrennzeichen
`\`	hebt Sonderbedeutung des folgenden Zeichens auf; am Zeilenende: Befehl wird in der nächsten Zeile fortgesetzt
`"..."`	die meisten eingeschlossenen Sonderzeichen verlieren ihre Bedeutung
`'...'`	alle eingeschlossenen Sonderzeichen verlieren ihre Bedeutung
`` `...` ``	wird durch das Ergebnis des eingeschlossenen Befehls ersetzt
`?`	Jokerzeichen: genau ein beliebiges Zeichen eines Datei- oder Verzeichnisnahmens
`*`	Jokerzeichen: beliebig viele, beliebige Zeichen eines Datei- oder Verzeichnisnamens
`[...]`	steht für eines der Zeichen zwischen den Klammern
`!`	aktiviert die history expansion
`{}`	fasst zu einer Gruppe zusammen
`()`	fasst Kommandos zusammen und übergibt sie an eine Kinds-Shell
`<>`	Eingabe- und Ausgabeumleitung
`|`	verbindet Ein- und Ausgabe zweier Befehle
`;`	trennt Befehle innerhalb einer Zeile
`&`	startet ein Programm im Hintergrund
`$`	kennzeichnet einen Variablennamen
`&&`	verknüpft zwei Befehle: der zweite wird nur ausgeführt, wenn der erste erfolgreich war
`||`	verknüpft zwei Befehle: der zweite wird nur ausgeführt, wenn der erste nicht erfolgreich war

Abbildung 2: Zeichen mit besonderer Bedeutung innerhalb der `bash` .

5 Der GNU/Linux Verzeichnisbaum

Eine GNU/Linux Installation besteht aus vielen Tausend Dateien. Davon sind alleine 1000 bis 2000 ausführbare Dateien, also Kommandos. Dazu kommen dann noch jede Menge nicht ausführbare Konfigurations-, Text- und Binärdateien.

Damit man sich in diesem Chaos zurechtfindet, sind die Dateien in einer *Baumstruktur* organisiert. Dieses Ordnungsprinzip gibt es bei praktisch allen Betriebssystemen, ist aber keine Erfindung des Computerzeitalters, sondern viel älter: zum Beispiel ist jede Bibliothek (gemeint ist eine Bibliothek mit echten Büchern, keine Programmbibliothek) genauso in einer Baumstruktur organisiert.

Wie der Verzeichnisbaum unter GNU/Linux aussieht, zeigt Bild 3. Die Verzeichnisse unterhalb von `home` sind die Heimatverzeichnisse der Benutzer. Dieser Teil des Baums sieht daher auf jedem Rechner anders aus.

Der Baum beginnt mit dem sogenannten *Wurzelverzeichnis*. Das Zeichen "/" steht für dieses Verzeichnis. Manche sagen zum Wurzelverzeichnis auch root-Verzeichnis oder kurz root. Das kann aber verwirrend sein, da das Heimatverzeichnis des Systemadministrators wirklich `/root` heisst.

In Bild 3 sind nur Verzeichnisse, keine Dateien dargestellt. Viele der Verzeichnisse haben wiederum Unterverzeichnisse, die sind bis auf zwei Ausnahmen (`/home` und `/usr`) nicht herausgeklappt.

Hier nun eine knappe Beschreibung der einzelnen Pfade:

`/bin` : Hier stehen für alle Benutzer ausführbare Binärdateien, also Kommandos, wie z.B. `ls`, `mv` und `mkdir`. Die Kommandos, die nur der Systemverwalter ausführen darf, stehen in `/sbin`. `/bin` enthält nur elementare Kommandos zur Systemverwaltung. Weitere, speziellere Kommandos stehen in `/usr/bin`.

`/boot` : enthält Dateien, die für den Rechnerstart notwendig sind. Bei Debian GNU/Linux steht hier auch der Kernel, der beim Start in den Speicher geladen wird.

`/cdrom` : ist zunächst ein leeres Verzeichnis. An dieser Stelle wird mit dem Kommando `mount /cdrom` der Verzeichnisbaum, der sich auf einer eingelegten CD befindet "eingehängt".

`/dev` : hier stehen keine echten Dateien, sondern sog. Device-Dateien, das sind "Psuedo"-Dateien, über die man auf fast alle Hardware-Komponenten zugreifen kann.

`/etc` : hier stehen `alle` Konfigurationsdateien des Systems.

`/floppy` : hier gilt das Gleiche, wie für das Verzeichnis `/cdrom`. In Abb. 3 ist der Verzeichnisbaum einer "gemounteten" Diskette dargestellt.

`/home` : unterhalb dieses Verzeichnisses liegen die Heimatverzeichnisse der Anwender.

> Wenn man als Benutzer angemeldet ist, erreicht man sein Heimatverzeichnis immer mit dem Zeichen $\sim$.

Ist man z.B. als Benutzer `micha` angemeldet und liegt das zugehörige Heimatverzeichnis unterhalb von `/home/micha`, steht das Zeichen $\sim$ für den absoluten Pfad `/home/micha`.

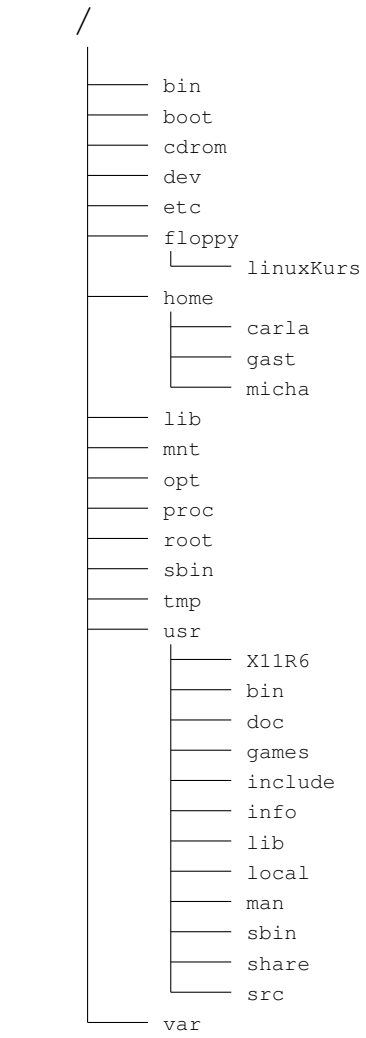

Abbildung 3: Die wichtigsten Teile des GNU/Linux -Verzeichnisbaums.

/lib : enthält von vielen Programmen gemeinsam benutzte Bibliotheken

/mnt : sollte normalerweise ein leeres Verzeichnis sein. Es dient dazu,auf die Schnelle ein Laufwerk an dieser Stelle einzuhängen, zu mounten.

/opt : hier können Softwarepakete installiert werden, die nicht zur Distribution gehören. Ein alternativer Ort für solche Pakete ist `/usr/local` .

/proc : enthält Unterverzeichnisse für alle laufenden Prozesse. Ähnlich wie die Dateien in `/dev` ist das ganze `/proc`-Verzeichnis nicht wirklich auf der Festplatte vorhanden. Die darin enthaltenen Dateien sind nur virtuell und werden bei Zugriff von Kernel dynamisch erzeugt. Ein Beispiel:

```
cat /proc/meminfo
```

Die Pseude-Datei `meminfo` enthält also Daten über die aktuelle Speichernutzung.

`/root` : ist das Heimatverzeichnisses des Benutzers `root`.

`/sbin` : hier stehen Systemkommandos, die nur vom Superuser (daher das s von sbin) ausgeführt werden dürfen.

`/tmp` : enthält temporäre Dateien.

`/usr` : usr ist nicht etwa eine Abkürzung von user, sondern heisst **U**nix **S**ystem **R**esources. Hier stehen die Anwendungsprogramme /usr/bin, das X-Window-System für die grafische Ausgabe, Interpreter, Compiler und Bibliotheken für verschiedene Programmiersprachen, jede Menge Dokumentationsdateien usw. In Abb. 3 sind die Unterverzeichnisse herausgeklappt um zu zeigen, dass hier nochmals ein eigener Unter-Verzeichnisbaum beginnt.

`/var` : var steht für *variabel*. D.h. `/var` enthält veränderliche Dateien, wie z.B. log-Dateien, Warteschlangen, `mail`-Dateien usw. Ein wichtiges Unterverzeichnis ist `/var/lib/mysql`. Hier stehen Datenbanktabellen von `mySQL`.

5.1 Das Arbeiten mit Wechselmedien

Wechselmedien sind z.B. CDRoms, Disketten, ZIP-Medien (alles veraltet), USB-Sticks (etwas moderner) oder Magnetbänder. Auf allen diesen Medien ausser den Magnetbändern kann sich ein Dateisystem mit einem Verzeichnisbaum befinden, das mit dem Befehl `mount` dem Verzeichnisbaum auf der Festplatte hinzugefügt wird.

Der Befehl `mount` ist recht kompliziert, deshalb betrachten wir hier nur zwei einfache Anwendungen:

```
mount /cdrom
mount /usb
```

Diese beiden Kommandos fügen den Verzeichnisbaum von einer CD oder eines (FAT-formatierten, ntfs geht nicht so leicht) USB-Sticks unterhalb der Verzeichnisse `/cdrom` bzw. `/usb` ein. Anschliessend kann man mit diesen Unterverzeichnissen arbeiten, als wären sie ein Teil der Festplatte.

Möchte man die CD oder USB-Stick wieder entnehmen, muss man dieses Einhängen wieder rückgängig machen. Dazu gibt es die Kommandos

```
umount /cdrom
umount /usb
```

Die CD lässt sich ohne vorhergehendes `umount`-Kommando nicht entnehmen, der Schlitten des Laufwerks fährt dann nicht aus.

Den USB-Stick könnte man natürlich jederzeit abziehen. Tut man dies aber ohne vorhergehendes `umount` , kann es zu Datenverlusten kommen.

Das liegt daran, dass der Kernel bei Dateioperationen nicht immer sofort auf das Laufwerk zugreift sondern mit einem Cache im RAM arbeitet. Schreibt man nun z.B. auf

den USB-Stick, landen die Daten erst im Cache und nicht sofort auf dem Stick. Bei einem `umount` wird der Cache auf jeden Fall geleert und alles auf den Stick geschrieben.

Mit dem Kommando `sync` kann man das Leeren des Caches auch zwischendurch erzwingen.

Achtung: die eben beschriebenen mount-Kommandos funktionieren nur, wenn das System entsprechend konfiguriert ist. Die dazu nötige Konfiguration muss in der Tabelle `/etc/fstab` stehen.

6 Der häufigste Anfängerfehler

Der häufigste und ärgerlichste Fehler, der Anfängern bei der Arbeit mit der `bash` unterläuft, ist das Verwechseln von absoluten und relativen Pfaden:

- Ein absoluter Pfad beginnt immer mit dem Zeichen "/". Bei einer solchen Pfadangabe sucht das System beim Wurzelverzeichnis beginnend den angegebenen Pfad ab. Diese Suche ist unabängig von aktuellen Verzeichnis.
- Ein relativer Pfad beginnt mit einem Datei- oder Verzeichnisnamen. Ein relativer Pfad wird vom *aktuellen* Verzeichnis ausgehend abgesucht!

Wenn man nun das führende Zeichen "/" eines absoluten Pfades vergisst - z.B. man gibt ein `home/gast` - sucht die `bash` im aktuellen Verzeichnis nach dem Unterverzeichnis `home`, das i.d.R. nicht vorhanden ist. Bei der Eingabe von `/home/gast` wird dagegen das gewünschte Verzeichnis gefunden.

7 Die wichtigsten Kommandos von GNU/Linux

Im Folgenden sind ein paar wenige, wichtige Kommandos beschrieben. Bei der Angabe der Kommando-Syntax werden eckige `[]` und spitze Klammern <> verwendet. Bei der späteren Eingabe eines Kommandos werden diese Klammern *nicht* mit eingegeben! Die Klammern sollen nur zeigen:

`[option]:` was in eckigen Klammern steht ist optional, muss also nicht zwingend eingegeben werden.

`<pfadname>:` was in spitzen Klammern steht *muss* eingegeben werden.

Die meisten Kommandos haben unzählige Optionen, die man unmöglich alle auflisten kann. Das ist auch gar nicht nötig, denn GNU/Linux bringt seine eigene Kommandodokumentation gleich mit:

1. **manual**

   ```
   man <kommandoname>
   ```

Manual bedeutet auf Deutsch *Handbuch*. Mit dem `man`-Kommando erhält man eine ausführliche Beschreibung zu allen Kommandos.

7.1 Navigation im Verzeichnisbaum

Die grundlegenden Aufgaben im Dateisystem lassen sich mit folgenden 6 Kommandos erledigen:

2. **list**

   ```
   ls [-optionen] [pfad]
   ```

   Damit wird der Inhalt des aktuellen Verzeichnisses gelistet, wenn keine Pfadangabe gemacht wurde, ansonsten wird eben das mit der Pfadangabe bezeichnete Verzeichnis aufgelistet.

   Beispiele:

   `ls`: erzeugt eine Liste der sichtbaren Dateien. Die Ausgabe erfolgt platzsparend mit mehreren Dateinamen pro Ausgabezeile.

   `ls -l`: die Option `-l` erzeugt eine ausführlichere Liste. Pro Zeile wird nur eine Datei aufgelistet, dafür werden aber wichtige Dateiattribute wie Ausführrechte, Besitzer, Gruppe, Dateigrösse, Datum und Uhrzeit ausgegeben.

   `ls -la`: die zusätzliche Option a steht für **all**. D.h. es werden jetzt auch versteckte Dateien aufgelistet. Versteckte Dateien sind alle Dateien, die mit *einem Punkt* beginnen.

   `ls -la > listing`: bei diesem Befehl wird die Ausgabe, die ja normalerweise auf dem Bildschirm landet, in eine Datei *umgeleitet*. Das wird durch das Zeichen '>' erreicht. Diese Umleitung funktioniert mit jedem Programm, das auf dem Bildschirm ausgibt.

3. **change directory**

   ```
   cd <pfad>
   ```

   Damit wechselt man ins angegebene Verzeichnis. Die Pfadangabe kann absolut, also vom Wurzelverzeichnis aus, oder relativ, vom aktuellen Verzeichnis ausgehend, angegeben werden.

4. **move**

   ```
   mv [-opt] <name_alt> <name_neu>
   mv [-opt] <dateien> <verzeichnis>
   ```

   mv dient zum Verschieben und Umbenennen von Dateien. Eine Datei wird umbenannt, wenn man sie innerhalb des selben Verzeichnisses verschiebt.

5. **copy**

   ```
   cp [-opt] <quelle> <ziel>
   cp [-opt] <dateien> <verzeichnis>
   ```

cp kopiert Dateien.

Optionen:

`-R`: kopiere rekursiv. Mit dieser Option lassen sich Verzeichnisse, ja ganze Pfade kopieren.

`-u`: ist diese Option gesetzt wird nur kopiert, wenn keine neuere Datei gleichen Namens überschrieben wird.

6. **make directory**

```
mkdir [-opt] <name>
```

Mit `mkdir` wird ein neues Verzeichnis angelegt.

Optionen:

`-p`: mit dieser Option lassen sich Verzeichnisse und Unterverzeichnisse auf einmal anlegen.

Beispiel:

`mkdir -p ~/linuxUebung/loesung` legt zuerst das Verzeichnis `linuxUebung` und innerhalb dieses Verzeichnisses das Unterverzeichnis `loesung` an.

7. **remove**

```
rm [-opt] <name>
```

Mit `rm` lassen sich Dateien und bei Verwendung von `-r` ganze Pfade löschen. Vorsicht: `rm` fragt nicht zurück!

7.2 Das Archivierkommando

8. **tape archive**

```
tar <aktion>f <archivdatei> [-C] <pfad>
```

`tar` dient dazu, Archivdateien zu erzeugen, oder Archivdateien wieder auszupacken. Ob nun gepackt oder entpackt werden soll, wird mit `<aktion>` gesteuert: **x** steht für *extrakt*, also auspacken und **c** steht für *create*, einpacken.

Mit der Option **z** wird bei diesen Vorgängen entsprechend noch komprimiert oder dekomprimiert (verwendet wird dazu gzip= GNU-Zip). Die Option **v** (verbose) schaltet die Ausgabe der internen Abläufe auf dem Bildschirm ein.

Die wichtigste Option ist aber das `f`: ursprünglich war `tar` dazu gedacht, Archive auf einem Bandlaufwerk zu schreiben oder lesen. Damit `tar` stattdessen auf eine Archiv**datei** auf der Platte zugreift, muss man immer die Option `f` verwenden, auf die der entsprechende Dateiname folgen musss.

Beispiele:

`tar -cvzf archiv.tgz .`: Archiviert und komprimiert das aktuelle Verzeichnis (.) in der Datei `archiv.tgz`

`tar -cvzf briefe.tgz briefe`: Archiviert und komprimiert das Verzeichnis `briefe` in der Datei `archiv.tgz`

`tar -xvzf archiv.tgz`: Extrahiert und entkomprimiert das Archiv ins aktuelle Verzeichnis.

`tar -xvzf archiv.tgz -C ~/dummy`: Extrahiert und entkomprimiert das Archiv ins Verzeichnis `/dummy`.

8 Dateirechte

Da GNU/Linux wie alle Unixe ein Mehrbenutzersystem ist, muss es einen Mechanismus geben, der die Dateien eines Benutzers vor unberechtigtem Zugriff durch die anderen Benutzer schützt.

Das wird durch das Abspeichern zusätzlicher Daten für jede Datei sichergestellt. Diese sog. *Meta*-Daten enthalten Informationen darüber **wem** die Datei gehört, wer sonst noch darauf zugreifen kann und für wen **Lese**-, **Schreib**- und **Ausführrechte** gegeben sind.

GNU/Linux unterscheidet dabei zwischen drei verschiedenen Benutzerarten:

u = user : der Eigentümer einer Datei.

g = group : eine Gruppe von Benutzern, der der Eigentümer besondere Rechte eingeräumt hat (z.B. seine Arbeitsgruppe). Wird eine Datei neu erzeugt, wird hier die Gruppe eingetragen, in der der Eigentümer selbst Mitglied ist.

o = others : das ist der Rest der Welt, also alle Anderen.

Bei den Zugriffsrechten gibt es ebenfalls drei Arten:

r = read : Leserecht

w = write : Schreibrecht

x = execute : x heisst, die Datei ist *ausführbar*. In GNU/Linux gibt es keine spezielle Dateiendung `.exe`. Bei einer ausführbaren Datei ist einfach nur das Ausführrecht gesetzt. Die Dateiendung spielt absolut `keine` Rolle.

Ist bei einem Verzeichnis das Ausführrecht gesetzt, heisst das, dass der Verzeichnis*inhalt* aufgelistet werden darf.

Nun haben wir also 3 x 3 verschiedene Kombinationen aus Rechten und Benutzern. Diese werden durch die 9 *Rechtebits*, die mit jeder Datei gespeichert werden, dargestellt. Mit dem Kommando `ls -l` werden sie angezeigt. Abb. 4 (am Ende des Texts) zeigt die Bedeutung der Ausgabe.

8.1 Ändern der Rechtebits

Geändert werden die Zugriffsrechte mit dem Befehl `chmod`. Die Syntax sieht so aus:

```
chmod <wer> +/- <welche Rechte> <datei>
```

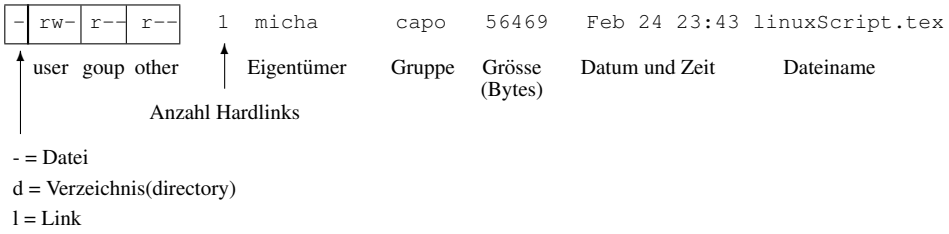

Abbildung 4: Die Bedeutung der Dateiattribute

Das Plus-Zeichen setzt ein Recht, das Minus-Zeichen entzeiht das Recht.
Beispiele:

1. Ausführecht für Eigentümer setzen:
   `chmod u+x skript.sh`
2. Leserecht für alle ausser dem Eigentümer entfernen:
   `chmod go-r linuxScript.tex`
3. Alle Rechte für Alle setzen:
   `chmod ugo+rwx datei`
4. `ugo` zusammengefasst zu `a`:
   `chmod a+rwx`

Alternativ zu dieser Syntax, kann man die drei Bits für jeden Benutzertyp auch direkt als Oktalzahl eingeben. Beispiel für das Setzen aller Rechte für Eigentümer und Gruppe und keine Rechte für den Rest:
`chmod 770 datei`.

8.2 Ändern von Gruppe und Eigentümer

Die Gruppe einer Datei kann der Eigentümer mit dem Befehl

```
chgrp <gruppe> <datei>
```

ändern.
Für den Administrator gibt es noch den Befehl `chown`. Mit diesem kann der Besitzer einer Datei geändert werden.

9 Details der Unix-Dateisysteme

Die internen Details der Unix-Dateisysteme werden nicht geheimgehalten, so dass ihre Funktionsweise gut untersucht werden kann.

Wir beschränken uns hier natürlich auf das Linux-Dateisystem. Im Moment wird das *third extended filesystem* (**ext3**) verwendet. Der Vorgänger von ext3 war ext2, das seit 1993 (!) im Einsatz ist.

Da ext3 abwärtskompatibel mit ext2 ist, genügt es an dieser Stelle, sich die Funktionsweise von ext2 anzuschauen. Hier eine **knappe** Beschreibung:

- Die Magnetspuren auf Festplatten werden von der Hardware in **Segmente** von 512 Bytes Länge unterteilt. Dies ist die kleinste Einheit, die auf den Platten adressiert werden kann. Manchmal werden diese Segmente auch als Blöcke gekennzeichnet, im Unix-Umfeld ist 'Segment' jedoch die bessere Bezeichnung (s.u.).
- Je 8 HD-Segmente werden im Dateisystem zu **Blöcken** von 4KiBytes ($4 \cdot 2^{10} Bytes$) zusammengefasst. Blockgrössen von 2KiB oder 1KiB sind auch möglich. In Windows-Dateisystemen, nennt man die Zusammenfassung von Segmenten Cluster.
- Die nächste und wichtigste Organisationseinheit von Unix-Dateisystemen sind die sog. *Inodes*. Die Inodes sind winzige Dateien von einheitlich 128 Bytes Länge, die **ausser dem Dateinamen** (!!) alle Attribute der eigentlichen Datendatei enthalten. Die Attribute werden *Meta-Daten* genannt. Das wären:
  - Ein Zähler, der die Anzahl der Verweise (=Dateinamen in Verzeichnissen) auf diesen Inode enthält. Fachausdruck Hard-Link-Zähler.
  - Besitzer und Gruppe
  - Zugriffsrechte
  - Typ der Datei: Datei, Verzeichnis, Verknüpfung (Link),
  - Grösse der Datei
  - Diverse Zeitstempel
  - Eine Liste mit den Nummern von bis zu 12 Blöcken (bei alten Unix-Dateisystemen: 10 Blöcke), die die eigentlichen Dateidaten enthalten. Anders ausgedrückt: der Inode enthält 12 Zeiger auf die Datenblöcke, die die eigentlichen Dateidaten enthalten.
  - Bei grossen Dateien enthält der 13. Listenplatz nicht die Blocknummer (Zeiger) direkt, sondern die Nummer eines Blocks, der wiederum eine Liste mit bis zu 256 Blocknummern enthält (**einfach indirekter Block**).
  - Wird noch mehr Platz benötigt, steht an 14. Stelle ein **zweifach indirekter Block** und evtl. an 15. Stelle **dreifach indirekter Block**.
- Jeder Inode ist mit einer eindeutigen Schlüsselnummer (**Inode-Nummer**) markiert. Diese wird nicht im Inode gespeichert, sondern ergibt sich aus dem physikalischen Speicherort des Inodes selbst: Alle Inodes haben die einheitliche Länge von 128 Bytes und werden hintereinander weg als Liste geschrieben. Diese Liste enthält im Normalfall 2048 Inodes.

Über die Inode-Nummer ist der eindeutige Zugriff auf eine bestimmte Datei möglich.

- Verzeichnisse werden unter Unix auch als Inodes und damit als Mini-Dateien (s.o.) verwaltet. Ein Verzeichnis ist demnach eine Datei, die einfach nur die Namen der enthaltenen Dateien und Unterverzeichnisse enthält und diese mit den zugehörigen Inode-Nummern verknüpft.

  Durch diese Verknüfpungen ist es möglich, einer physikalischen Datei mehrere Namen zu geben (Siehe Übungsaufgabe 34).

  Die Verknüpfung einer Inode-Nummer mit einem Dateinamen nennt man **Hardlink**. Zu einem Inode darf es beliebig viele Dateinamen (Hardlinks) geben.

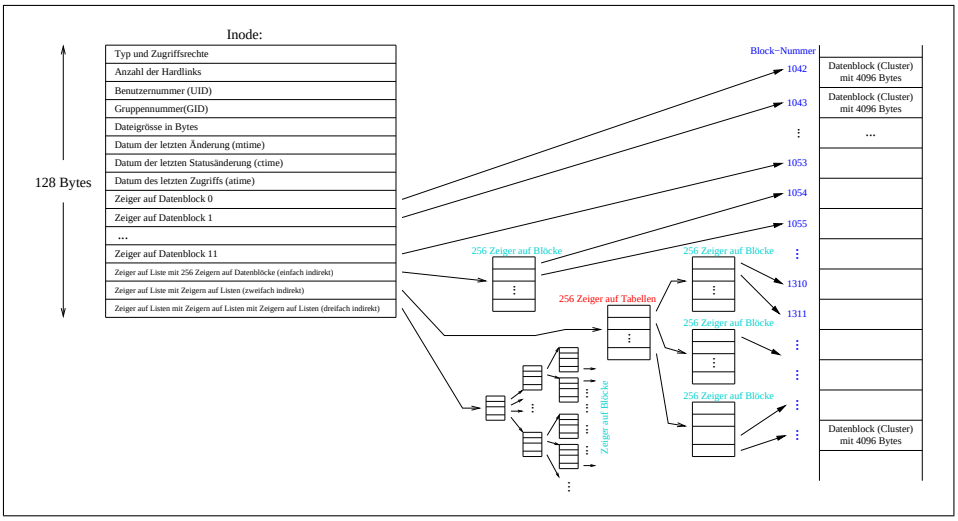

Abbildung 5: Modell eines Linux-Inodes

10 Multitasking

Unix und damit auch GNU/Linux kann mehrere Programme scheinbar gleichzeitig ausführen. Diese Eigenschaft wird als *Multitasking* (engl. Task = Aufgabe) bezeichnet. Da die meisten Rechner aber nur einen Prozessor haben, laufen die Programme nicht wirklich gleichzeitig, sondern bekommen vom Kernel *nacheinander* Rechenzeit zugeteilt. Dieser Vorgang des Umschaltens des gerade aktiven Programms läuft aber so schnell ab, dass der Anwender es nur selten bemerkt und den Eindruck hat, die aktiven *Tasks*, zu deutsch *Prozesse*, liefen gleichzeitig.

Manche dieser Prozesse laufen unbemerkt vom Anwender im Hintergrund. Einige Beispiele hierfür sind der Druckerdienst `lpd`, der Apache-Webserver `httpd` oder auch ein Datenbankserver wie `mysqld`. Das ”d” mit dem alle diese Dienste enden steht für `daemon`, Dämon. Man kann sie sich also als dienstbar Geister, eben als *Dienste* vorstellen.

Neben den Hintergrundprozessen gibt es auch welche, die im Vordergrund laufen. Einer davon ist das Programm `bash`, das ja ständig interaktiv mit dem Benutzer kommuniziert.

Nun kann ein Prozess Unterprozesse, sog. *Kinds*-Prozesse starten, die ihrerseits wieder Prozesse starten können. Auf diese Weise entsteht eine *Prozesshierarchie*.

Mit dem Befehl `pstree` kann man sich diese Hierarchie anzeigen lassen.
Nun kann man mit der `bash` von einer Konsole oder einem Terminalfenster (xterm) aus auch Prozesse starten. Wenn wir z.B. das Kommando

```
xdvi linuxKurs.dvi
```

eingeben, wird ein neuer Prozess gestartet.

10.1 Kindsprozesse einer Shell

Diese von `bash` aus gestarteten Prozesse nennt man *jobs*. Mit dem Kommando `jobs` werden alle Kindsprozesse einer `bash` die auf der Konsole oder in einem Terminalfenster läuft, aufgelistet. Wird der `emacs` so gestartet wie gerade angegeben, läuft er im *Vordergrund* und er blockiert seinen Vaterprozess, die `bash`.

Hängt man beim Starten eines Programms ans Ende der Kommandozeile ein & -Zeichen, wird das betreffende Programm sofort im Hintergrund gestartet und die `bash` bleibt aktiv:

```
xdvi linuxKurs.dvi &
```

Hat man das & -Zeichen vergessen, kann man in der blockierten Shell die Tastenkombination CTRL-Z drücken und damit den Kindsprozess *anhalten*. Die Shell (=bash) meldet dann, welche Jobnummer gestoppt wurde. Anschliessend kann man mit dem Kommando

```
bg <jobnummer>
```

den Kindsprozess im Hintergrund (bg = background) fortsetzen. Entsprechend kann man mit `fg <jobnummer>` einen Job auch wieder in den Vordergrund (foreground) holen.

11 Aufgaben

1. Wechsle in Dein Heimatverzeichnis
2. Erzeuge dort folgende Verzeichnisstruktur:

   ```
   ~
   |
   +-linuxUebung
      |
      +-<vorname>
      |
      +-<nachname>
      |
      +-loesungen
      |
      +-script
   ```

   Dabei soll das, was in spitzen Klammern steht, durch die tatsächlichen Namen ersetzt werden.
3. Kopiere die Datei

   `~/__vorlagen/vorlagen-<klasse>/linuxScript.pdf`

   in das Verzeichnis

   `~/linuxUebung/script`

   Der Ausdruck `<klasse>` muss natuerlich durch so etwas wie:

   `e1it2t`

   ersetzt werden.

   Hinweis: das Zeichen ~ ist eine Abkürzung für den absoluten Pfad in Euer Heimatverzeichnis.
4. Erzeuge in

   `~/linuxUebung/loesungen`

   die Datei `bash.txt`.

   Verwende dazu den Editor `nano`.

   Der heisst ausser hier bei Debian-basierten Systemen (z.B. ubuntu) überall sonst `pico`.

   Alle folgenden Antworten sollen direkt mit `nano` in diese Datei geschrieben werden.

   Wer vorhat eine LPI-Zertifizierung zu machen, sollte unbedingt mit dem Editor **vi** bzw. **vim** arbeiten, da Kenntnisse über diesen Editor in der LPI-Prüfung verlangt werden.

   Ein anderer, berühmter Editor, auf dem auch gerade dieser Text entsteht, heisst **emacs** bzw. **xemacs**.

5. Was ist ein absoluter Pfad, was ist ein relativer Pfad? Woran kann man sie bei der Eingabe unterscheiden?

6. Warum funktioniert das Kommando cd ␣ .. aber nicht cd.. ?

7. Wie lautet die Kommandozeile von Aufgabe 1 mit absoluten Pfaden?

8. Wie kann man durch Einsatz der TAB -Taste diesen Pfad leicht eingeben?

9. Wie lautet die Kommandozeile von Aufgabe 30 mit ausschliesslich relativen Pfaden, wenn man sie vom Verzeichnis $\sim$ aus ausführt?

10. Gib eine Liste aller sichtbarer Dateien im Heimatverzeichnis aus. Die Liste soll nach der Dateigrösse sortiert sein. Wie lautet die zugehörige Kommandozeile?

11. Nun erzeuge eine Dateiliste, die nach dem Alter der Dateien sortiert ist. Die älteste Datei soll zuerst in der Liste stehen! Wie lautet die zugehörige Kommandozeile?

12. Leite die Ausgabe der vorangehenden Aufgabe in die Datei `~/linuxUebung/loesungen/sortiert` um. Mit welcher Kommandozeile lässt sich das bewerkstelligen?

13. Erzeuge das Verzeichnis `~/linuxUebung/loesungen/dummy`. Kopiere die in der vorangehenden Aufgabe erzeugte Datei (`~/sortiert`) dort hinein.

14. Verschiebe die Datei `.../sortiert` nach `.../dummy`. Verwende dazu den Befehl `mv` mit der Option *-i* . Was bewirkt die Option *-i* ?

15. Kopiere die verschobene Datei wieder zurück und versuche sie anschliessend nochmals ins Verzeichnis `.../dummy` zu kopieren. Verwende dazu den Kopierbefehl `cp` mit der Option *-i* . Was bewirkt hier die Option `-i`?

16. Versuche das Verzeichnis `dummy` nach `dummy2` zu kopieren. Welche Option muss man verwenden, damit das funktioniert?

17. Führe folgendes Kommando aus:

    ```
    rm -ri dummy
    ```

    Beantworte die erste Rückfrage mit 'y' und alle weiteren mit 'n'! Beschreibe wie ein Pfad schrittweise gelöscht wird. Was bedeutet die Frage: `descend into directory 'dummy'`?

18. Wie kann man sich mit dem alias-Kommando den heissgeliebten Befehl `cd..` (ohne Leerzeichen) bauen?

19. Erzeuge ein Alias `dir`, das alle Dateien eines Verzeichnisses (auch die versteckten) nach Zeitstempel sortiert und mit ausführlicher Datums- und Zeitangabe ausgibt. Wie lautet die Kommandozeile für diese Alias-Erzeugung?

20. Auch der Standardbefehl `ls` ist ein Alias. Welches Kommando wird mit `ls` aufgerufen?

21. Teste das Kommando `whereis` und suche dabei nach dem Kommando `pdflatex`. Wie heisst die Kommandozeile und wo steht `pdflatex`?

22. Suche mit dem Kommando `find` nach `pdflatex`. Dabei soll der *gesamte* Verzeichnisbaum abgesucht werden, d.h. alles was unterhalb des Wurzelverzeichnisses liegt. Wie lautet die Kommandozeile?
    Hinweis: um nach einem Muster zu suchen, muss man die Option `-name ''muster''` verwenden.

23. Komprimiere die Datei `bash.txt` mit `gzip`. Die gezippte Datei soll ebenfalls im Verzeichnis `.../loesungen/` stehen und `bash.gz` heissen.

    Achtung: `gzip` wandelt eine Datei in eine komprimierte Datei um und löscht dann die Originaldatei. Für erste Experimente mit `gzip` sollte man daher eine Sicherheitskopie der Originaldatei machen. Die komprimierte Datei erhält die Endung `gz`.

24. Betrachte die Datei `bash.gz` mit `zless`.

25. Nun soll der Pfad `~/linuxUebung/` mit allen Dateien und Unterverzeichnissen in ein gzip-komprimiertes Archiv gepackt werden. Verwende dazu das tar-Kommando. Die Archivdatei soll `loesung.tgz` heissen und in `.../loesungen/` stehen.

26. Mit welchem Kommando lässt sich der Inhalt von `loesung.tgz` auflisten?

27. Führe folgendes Kommando aus: `cp /dev/stdin ~/textdatei` und gib anschliessend einige Zeilen Text ein. Jede Zeile muss mit RETURN abgeschlossen werden. Zum Beenden CTRL-Q drücken. Betrachte die neuentstandene Datei mit `less`. Wie kommen die Tastatureingaben mit dem `cp`-Befehl in eine Datei?

28. Nun soll folgender Befehl ausgeführt werden: `cp ~/textdatei /dev/stdout`. Wie lässt sich dieses Ergebnis erklären?

29. Ein USB-Stick oder eine CD soll mit dem Befehl `mount` in den Verzeichnisbaum eingebunden werden. Betrachte den Verzeichnisbaum mit den Dateimanagern `konqueror` oder `tree` vor und nach dem Einbinden der CD. (Aufruf von `tree: tree /`).

30. Kopiere eine grössere Datei (z.B. die Datei linuxScript.pdf) auf den USB-Stick.
    (Zur Erinnerung: der Stick wurde gerade ins Verzeichnis `/usb` eingebunden.)

    Lass Dir direkt nach dem Absenden des Kopierbefehls den Inhalt von `/usb` anzeigen. Wie lange braucht das System, um den Inhalt von `/usb` zu aktualisieren? Kann die PDF-Datei wirklich so schnell auf den Stick geschrieben worden sein?

31. Wiederhole den Kopiervorgang von Aufgabe 30. Nun soll direkt nach dem Absetzen des cp-Kommandos das Kommando `sync` gestartet werden. Wie lange bleibt die Shell nach dem Start von `sync` inaktiv?

32. Wie sieht der Verzeichnisbaum nun aus? Starte das Kommando `tree` um die Verzeichnisstruktur grafisch darzustellen. Nach Beenden von `tree` soll die Diskette wieder "ausgehängt" werden. Welches Kommando ist dazu notwendig?

33. Erzeuge mit dem Editor `nano` (auf SuSE, RedHat, OSX heisst dieser Editor `pico`) die Datei `skript.sh`. `skript.sh` soll folgenden Inhalt haben:

34. Wechsle ins Verzeichnis `/tmp` und Führe dort folgende Kommandosequenz aus:

```
touch eins.dummy
ln eins.dummy zwei.dummy
ln eins.dummy drei.dummy
ls -lai *.dummy
rm eins.dummy
ls -lai *.dummy
rm zwei.dummy
ls -lai *.dummy
```

Was bewirken die Kommandos `touch` und `ln`? Wie ändert sich der Hard-Link-Zähler? Welche zusätzlichen Informationen werden duch die Schalter `-lai` beim Befehl `ls` angezeigt?

35. Mit einem Editor soll die Datei `skript.sh` mit folgendem Inhalt erzeugt werden (Hinweis s.u.):

```
#! /bin/bash
cp skript.sh skript2.sh
ls -ltr .
echo 'hallo Du linux-fan'
echo 'dies hier ist ein '
echo 'sog. shell-script'
echo 'also ein programm,'
echo ' das von der bash'
echo 'ausgefuehrt wird'
echo 'die zweite zeile '
echo 'bewirkt, dass sich'
echo 'das skript selbst'
echo 'nach skript2.sh '
echo 'kopierte und das'
echo 'aktuelle verzeichnis
echo 'auflistet'
```

Startet den Editor mit der Kommandozeile `nano skript.sh`. Die Kommandos die `nano` versteht, werden unten angezeigt (vi Benutzer müssen die vi-manpage lesen: man vim). Das ^-Zeichen steht dabei für die CTRL-Taste. Beispiel: Speichern mit CTRL-O (O wie out, nicht 0!)

36. Nach dem Speichern der Datei `skript.sh` sollen die Dateirechte so gesetzt werden, dass man `skript.sh` ausführen kann.

37. Wie lautet die Kommandozeile, um die Datei `skript.sh` nur für den Eigentümer lesbar und schreibbar aber für alle ausführbar zu machen?

38. Wie kann man das gleiche Rechtebitmuster mit Oktalzahlen setzen?

39. Führe das Kommando `pstree` im Terminalfenster aus. Was stellt die gezeigte Baumstruktur da?

40. Wechsle durch gleichzeitiges Drücken von Strg-Alt-F2 in eine Textkonsole und melde Euch dort ebenfalls als `gast<nr>` an. Was ändert sich, wenn man hier den Befehl `pstree` ausführt? Durch Drücken von Strg-Alt-F7 kann man wieder in die grafische Konsole zurückwechseln.

41. Der Prozess `bash` läuft mehrfach. Was sind seine Elternprozesse?

42. Die Verzeichnisse, in denen `bash` ein eingegebenes Kommando sucht, stehen in der Systemvariablen `PATH`. Mit dem Kommando `echo $PATH` kann man sich die Liste dieser Verzeichnisse anzeigen lassen.

43. Wozu dient die Option -p beim Start von pstree? Starte pstree mit der Option -p.

44. Wie kann man mit `pstree` nur Teile der Prozesshierarchie betrachten? Zeige alle Kindsprozesse von `xinit` an.

45. Neben den eigenständigen Kommandos, gibt es noch Kommandos, die in `bash` fest eingebaut sind. Z.B. das Kommando `jobs`. Es zeigt die Kindsprozesse der Shell an, von der aus es gestartet wird. Führt das Kommando `jobs` in der Terminalemulation `xterm` aus. Welche Kindsprozesse hat diese Shell? Wie wurden die Kindsprozesse gestartet?

Kommandos zur Online-Hilfe

`apropos <schluesselwort>`	Sucht die **man**ual-Seiten (man-pages) für das gegebene Schlüsselwort.
`info <kommando>`	Startet das Info-System zum gegebenen Kommando. Beenden mit der Taste Q.
`man <kommando>`	Ruft die **man**ual-Seiten (man-pages) zum gegebenen Kommando auf.

Alias Namen für Befehle

`alias <neuer Name> = '<kommando>'`	Erzeugt einen Alias-Namen für das in Hochkommas stehende Kommando.
`type <alias>`	Zeigt an, welches Kommando sich hinter dem Alias versteckt.
`unalias <alias>`	Löscht das angegebene Alias.

Verwalten von Befehlen und Verzeichnissen

`cd <pfad>`	Wechselt ins angegebene Verzeichnis.
`chmod <rechteschluessel> <datei/verzeichnis>`	Ändert die Rechtebits einer Datei oder eines Verzeichnisses.
`chown <eigentuemer> <datei/verzeichnis>`	Setzt den Eigentümer einer Datei oder eines Verzeichnisses neu.
`chgrp <eigentuemer> <datei/verzeichnis>`	Setzt die primäre Gruppe einer Datei oder eines Verzeichnisses neu.
`cp <quelle> <ziel>`	Kopiert eine Datei oder ein Verzeichnis von Quelle nach Ziel.
`ln -s <pfad> <linkname>`	Erzeugt einen symbolischen Link. Nur mit der Option -s verwenden!
`mkdir <pfad>`	Erzeugt ein Verzeichnis.
`mv <quelle> <ziel>`	Verschiebt eine Datei oder ein Verzeichnis oder benennt eine Datei um.
`rm <datei/verzeichnis>`	Löscht eine Datei oder ein Verzeichnis.
`rmdir <leeres verzeichnis>`	Löscht ein *leeres* Verzeichnis.
`pwd`	Zeigt den aktuellen Pfad an.
`touch <datei/verzeichnis>`	Aktualisiert den Zeitstempel einer Datei oder eines Verzeichnisses.

Suchkommandos

`find <pfad> [optionen]`	`find` ist ein universelles Suchkommando. Die Suchkriterien werden mit den umfangreichen Optionen eingestellt. `find` durchsucht den Verzeichnisbaum unterhalb des angegebenen Pfads.
`locate <suchmuster>`	`locate` sucht in der Datenbank `locatedb` nach dem angegebenen Muster.
`updatedb`	Aktualisiert die Datenbank `locatedb`.
`whereis <datei>`	Ähnlich `find`. Es werden aber nur die sog. Standardpfade abgesucht. Daher ist `whereis` viel schneller als `find`.
`which <kommando>`	Sucht nach dem angegebenen Kommando. Dabei werden nur die Verzeichnisse abgesucht, die in der Umgebungsvariablen `PATH` aufgelistet sind.

Kommandos zum Bearbeiten und Anzeigen von Textdateien

`a2ps <datei>`	Druckt eine Textdatei auf einem Postscriptdrucker aus.
`grep <suchmuster> <datei>`	Sucht in der angegebenen Datei nach `<suchmuster>`.
`head <datei>`	Zeigt die ersten Zeilen einer Datei an.
`less <datei>`	Zeigt eine Datei seitenweise am Bildschirm an. Weiterblättern mit Leertaste, beenden mit Q.
`lpr <datei>`	Sendet eine Datei zur Druckerwarteschlange.
`sed <kommando> [ziel/quelle]`	Z.B. um bestimmte Zeichen einer Datei zu suchen und zu tauschen. Komplizierte Syntax, siehe man-pages.
`sort <datei>`	Sortiert eine Datei. Sortierkriterien müssen mit Optionen vorgegeben werden.
`tail <datei>`	Zeigt die letzten Zeilen einer Textdatei an. Das ist sehr hilfreich, um log-Dateien anzuschauen.
`zless <gezippte datei>`	`zless` funktioniert wie `less`, arbeitet aber direkt mit komprimierten Dateien.

Kommandos zum Komprimieren und Archivieren

`bzip2 <datei>`	Komprimiert oder expandiert (je nach Optionen) eine Datei. `bzip2` ist ca 20% leistungsfähiger als das bekannte `gzip`.
`gzip <datei>`	Komprimiert oder expandiert (je nach Optionen) eine Datei.
`tar <aktion> -f <datei/verzeichnis>`	Packt Dateien und/oder Verzeichnisse in eine Archivdatei. Je nach `<aktion>` kann tar auch den umgekehrten Vorgang ausführen.

Abbildung 6: Eine Übersicht der wichtigsten GNU/Linux Befehle.

Zugriff auf MS-DOS Disketten: die m-tools

`mcd a:<pfad>`	Wie das DOS-Kommando `CD`. Wechselt das aktive Verzeichnis auf der Diskette.
`mcopy <quelle> <ziel>`	Wie das DOS-Kommando `COPY`.
`mdir`	Wie das DOS-Kommando `DIR`.
`mformat a:` Wie das DOS-Kommando `FORMAT A:`.	
`mmd <pfad>`	Wie das DOS-Kommando `MD`.
`mren <alt> <neu>`	Wie das DOS-Kommando `REN`.

Kommandos zur Prozessverwaltung

`bg <job-/prozessnummer`	Setzt einen Prozess im Hintergrund fort.
`fg <job-/prozessnummer`	Holt einen angehaltenen oder im Hintergrund laufenden Prozess in den Vordergrund.
`free`	Zeigt die Hauptspeicherbelegung an.
`kill [signal] <pid>`	Sendet dem Prozess mit der Nummer ¡pid¿ ein Signal. Z.B. das Signal `-kill`, mit dem der Prozess hart beendet wird.
`halt`	Fährt das System herunter und hält den Rechner an.
`ps`	Zeigt eine Prozessliste mit Prozessnummern an. Je nach den Optionen werden nur eigene Prozesse oder z.B. alle (`ps -aux`) Prozesse angezeigt.
`pstree`	Zeigt eine Prozessliste mit Vererbungsstruktur.
`reboot`	Erklärt sich selbst.
`shutdown`	Wie `halt`.
`top`	Zeigt alle 5s eine Liste aller Prozesse an.

Benutzerverwaltung

`addgroup <name>`	Erzeut eine neue Gruppe.
`adduser <name>`	Legt einen neuen Benutzer an. Bei Debian interaktiv.
`groups [benutzer]`	Zeigt die Gruppen des angemeldeten oder angegebenen Benutzers an.
`passwd` bzw. `yppasswd`	Ändern des Benutzerpasswords. Die Variante `yppasswd` gilt nur für Benutzer, die sich via NIS anmelden.

Administration des Dateisystems

`dd [optionen]`	`dd` bedeutet disk dump. `dd` kopiert Sektorenweise von Laufwerk zu Laufwerk!
`du [datei/verzeichnis]`	Zeigt den Speicherbedarf von Dateien und Verzeichnissen an.
`mount <laufwerk> <zielverzeichnis>`	Einhängen eines Laufwerks in den Verzeichnisbaum. Das Zielverzeichnis ist der sog. mount-point. D.h. der Ort, an dem die Verzeichnisstruktur auf dem Laufwerk im gesamten Verzeichnisbaum erscheint.

Abbildung 7: Fortsetzung der Befehlsübersicht.