Kommandozeilen-Handbuch

Mandrakelinux 10.1

http://www.mandrakesoft.com

Kommandozeilen-Handbuch: Mandrakelinux 10.1
Veröffentlicht 2004-09-30

von Camille Bégnis, Christian Roy, Fabian Mandelbaum, Roberto Rosselli del Turco, Marco De Vitis, Alice Lafox, John Rye, Wolfgang Bornath, Funda Wang, Patricia Pichardo Bégnis, Debora Rejnharc Mandelbaum, Mickael Scherer, Jean-Michel Dault, Lunas Moon, Céline Harrand, Fred Lepied, Pascal Rigaux, Thierry Vignaud, Giuseppe Ghibò, Stew Benedict, Francine Suzon, Indrek Madedog Triipus, Nicolas Berdugo, Thorsten Kamp, Fabrice Facorat, Xiao Ming und Snature

Rechtliche Hinweise

Frontseiten-Text

```
    Mandrakesoft September 2004
http://www.mandrakesoft.com/
Copyright (c) 1999-2004 by Mandrakesoft S.A. and Mandrakesoft Inc.
```

Die in der folgenden Tabelle gelisteten Kapitel unterliegen einer anderen Lizenz. Weitere Informationen dazu entnehmen Sie bitte der Tabelle und den darin angegebenen Links.

	Original Copyright	**License**
Kompilieren und Installieren Freier Software, Seite 83	von Benjamin Drieu , **APRIL** (`http://www.april.org/`)	GNU General Public License GPL (`http://www.gnu.org/copyleft/gpl.html`)

Für dieses Handbuch benutzte Programme

Dieses Handbuch wurde in XML DocBook geschrieben. Borges (`http://www.mandrakelinux.com/en/doc/project/Borges/`) wurde als Verwaltungssystem eingesetzt. Die XML-Quell-Dateien wurden mittels xsltproc, `openjade` und `jadetex` unter Zuhilfenahme angepasster Stylesheets von Norman Walsh verarbeitet. Bilder wurden mittels `xwd` oder GIMP aufgenommen und mit `convert` aus ImageMagick konvertiert. Die komplette verwendete Software ist frei verfügbar und Bestandteil Ihrer Mandrakelinux Distribution.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Vorwort

1 Über Mandrakelinux

Mandrakelinux ist eine durch **Mandrakesoft** S.A. herausgebrachte GNU/Linux-Distribution. Zur „Internetgeburt“ **Mandrakesoft**s kam es 1998. **Mandrakesoft** hatte und hat das Ziel, ein leicht erlernbares und benutzerfreundliches GNU/Linux-System zur Verfügung zu stellen. Die zwei großen Pfeiler, auf denen **Mandrakesoft** ruht, sind Open-Source und kollaborative Arbeit am Produkt.

1.1 Fragen Sie die Mandrake-Gemeinde

Im Folgenden erhalten Sie zahlreiche Internet-Links auf verschiedene Seiten mit Bezug zu Mandrakelinux. Wollen Sie mehr über die Firma **Mandrakesoft** erfahren so besuchen Sie unsere Website (`http://www.mandrakesoft.com/`). Sehen Sie sich bitte auch die Website der Mandrakelinux-Distribution (`http://www.mandrakelinux.com/de/`) an sowie ihre zahlreichen Ableger.

MandrakeExpert (`http://www.mandrakeexpert.com/`) ist **Mandrakesoft**s offene Hilfe-Plattform. Erleben Sie eine neue Erfahrung, basierend auf Vertrauen und der Freude, andere Benutzer für ihren Beitrag und ihre Hilfe zu belohnen.

Wir möchten Sie gleichfalls einladen, an den zahlreichen Mailinglisten (`http://www.mandrakelinux.com/de/flists.php3`) teilzunehmen, in denen man die Mandrakelinux-Gemeinde sehr lebhaft bei der Arbeit an und mit der Distribution erleben kann.

Schlussendlich wollen wir auch Mandrakesecure (`http://www.mandrakesoft.com/security`) nicht vergessen. Auf dieser Seite wird das gesamte sicherheitsrelevante Material über alle Mandrakelinux-Distributionen gesammelt und bereitgestellt. Sie finden dort Sicherheitshinweise und Möglichkeiten zur Behebung bekannter Fehler. Es gibt auch allgemeine Artikel über Datenschutz und Sicherheit. Ein Muss für alle Administratoren und Personen, die sich über Sicherheit Gedanken machen.

1.2 Kommen Sie in den Klub

Mandrakesoft bietet Ihnen eine breite Palette von Vorteilen und Diensten durch den Mandrakeclub (`http://www.mandrakeclub.com`). Sie können dort

- kommerzielle Programme herunterladen, die normalerweise nur in Boxen vertrieben werden (etwa spezielle Treiber, kommerzielle Vollversionen, Freeware und Demoversionen);
- in einem RPM-Auswahlverfahren neue Softwarepakete vorschlagen bzw. über deren Aufnahme in die Distribution mit abstimmen;
- auf ein Software-Depot mit über 50.000 RPM-Paketen für alle Mandrakelinux Distributionen zugreifen;
- Rabatte für Produkte und Dienstleistungen im Mandrakestore (`http://store.mandrakesoft.com`) erhalten;
- über eine exklusiv für Klubmitglieder zusammengestellte Liste von Spiegelservern schnellere Downloads durchführen;
- multilinguale Diskussionsforen und Artikel besuchen.

Mit der Finanzierung von **Mandrakesoft** durch den Mandrakeclub helfen Sie aktiv, die Mandrakelinux Distribution zu verbessern, sodass wir unseren Anwendern auch in Zukunft den bestmöglichen GNU/Linux-Desktop bieten können.

1.3 Abonnieren Sie Mandrakeonline

Mandrakesoft bietet Ihnen eine sehr bequeme Möglichkeit, Ihr System automatisch auf dem aktuellsten Stand zu halten und dabei Bugs zu beseitigen sowie Sicherheitslücken zu schließen. Auf der Mandrakeonline Website (`https://www.mandrakeonline.net/`) erfahren Sie mehr über diesen Service.

1.4 Erwerb von Mandrakesoft-Produkten

Die Anwender von Mandrakelinux können alle Produkte online im Mandrakestore (`http://store.mandrakesoft.com`) erwerben. Dort bekommen Sie verschiedene Versionen von Mandrakelinux – Betriebssysteme und „Live-CDs“ (z.B. Move) – aber auch ausgewählte Abonnement-Angebote, Support, Software von Drittanbietern, Lizenzen, Handbücher sowie Bücher rund um Linux und natürlich auch die beliebten Fanartikel.

1.5 Zu Mandrakelinux beitragen

Alle Talentierten unter Ihnen sollten sich nun angesprochen fühlen: Ihre Fähigkeiten sind sicher hilfreich, um eine der zahlreichen Aufgaben bei der Erstellung einer neuen Version von Mandrakelinux zu übernehmen:

- **Paketerstellung.** Ein GNU/Linux-System besteht vornehmlich aus Programmen, die aus dem Internet stammen. Diese müssen in Pakete geschnürt werden, um ihre reibungslose Zusammenarbeit sicherzustellen.
- **Programmieren.** Es gibt unzählige Projekte, die direkt von **Mandrakesoft** unterstützt werden: Suchen Sie sich das heraus, das Ihnen am meisten zusagt und bieten Sie dem Autor Ihre Hilfe an.
- **Internationalisierung.** Wir benötigen ständig Hilfe bei der Übersetzung der Web-Seiten, Programme und der Dokumentation.
- **Dokumentation.** Zu guter Letzt braucht natürlich auch das Handbuch, das Sie gerade lesen, viel Beistand, um mit der schnellen Evolution der Distribution schritthalten zu können.

Besuchen Sie die Seite der Mitwirkenden (`http://www.mandrakesoft.com/labs/`), um herauszufinden, wo und wie Sie bei der Verbesserung von Mandrakelinux mithelfen können.

2 Einführung

Dieses *Kommandozeilenhandbuch* richtet sich an Benutzer, die in die Tiefen ihres Mandrakelinux-Systems eintauchen und seine ganze Leistung ausnutzen wollen. Wir möchten erreichen, dass Ihnen nach der Lektüre dieses Handbuches die tägliche Administration Ihres GNU/Linux-Computers leichter fällt als vorher. Im Folgenden finden Sie einen Ausblick auf die drei Bereiche des Handbuches und eine kurze Beschreibung der enthaltenen Kapitel:

- In *Das Linux System* stellen wir Ihnen die Kommandozeile mit ihren verschiedenen Einsatzmöglichkeiten vor. Außerdem beschäftigen wir uns mit den Grundlagen der Textbearbeitung, einem sehr wichtigen Kapitel unter GNU/Linux.

  Im ersten Kapitel (*Grundlegende UNIX Konzepte*, Seite 7) reden wir etwas genauer über die GNU/Linux-Welt und versuchen, Ihnen die UNIX®-Paradigmen näher zu bringen. Sie erfahren etwas über Standard-Dateioperationen sowie über nützliche Eigenschaft der shell. Dem schließt sich ein ergänzendes Kapitel an (*Festplatten und Partitionen*, Seite 17), das die Festplatten-Verwaltung und Partitionierung unter GNU/Linux behandelt. Die Konzepte dieser Kapitel sollten Ihnen geläufig sein, bevor Sie mit *Einführung in die Kommandozeile*, Seite 23 weitermachen.

  Das folgende Kapitel (*Textbearbeitung: Emacs und VI*, Seite 33) behandelt die Textbearbeitung. Unter UNIX® werden die meisten Konfigurationseinstellungen aus historischen Gründen in reinen Textdateien gespeichert. Sie werden daher über kurz oder lang diese Dateien mit einem ***Texteditor*** bearbeiten wollen. Aus diesem Grund werden Sie zwei der bekanntesten Texteditoren der UNIX®- und GNU/Linux-Welt kennen lernen: den mächtigen Emacs und den guten alten Vi, der bereits 1976 von Bill Joy entwickelt wurde!

  Danach sollten Sie die grundlegende Verwaltung Ihres Systems beherrschen. Die folgenden beiden Kapitel beschäftigen sich mit der praktischen Nutzung der Kommandozeile (*Kommandozeilenwerkzeuge*, Seite 41) und der Prozess-Kontrolle (*Prozesskontrolle*, Seite 51) im Allgemeinen.

- In *Linux in aller Ausführlichkeit* wenden wir uns dem Linux und der Struktur des Dateisystems zu.

  Wir erforschen in *Datei-Organisation*, Seite 55 die Organisation des Dateibaumes. Unter UNIX® pflegen Dateisysteme gerne recht umfangreich zu werden, jede Datei hat jedoch ihren Platz im gesamten Gefüge. Nach Lektüre dieses Kapitels werden Sie wissen, wo auf Ihrem Rechner Sie die Suche nach bestimmten Dateien beginnen können.

Anschließend werden wir die Themen ***Dateisysteme*** und ***Einhängepunkte*** (*Dateisysteme und Einhängpunkte*, Seite 59) behandeln. Wir werden beide Begriffe definieren und anhand praktischer Beispiele erklären.

Das folgende Kapitel beschreibt einen bestimten Dateisystemtyp (*GNU/Linux Dateisysteme*, Seite 65). Nach einem Blick auf die verfügbaren Dateisysteme beschäftigen wir uns mit anderen Dateisystemen, zusätzlichen Konzepten und Werkzeugen wie Inodes und Pipes. Das daran anschließende Kapitel (*Das Dateisystem /proc*, Seite 75) stellt ein spezielles GNU/Linux-Dateisystem vor: `/proc`.

Die Startdateien: sysv initialisieren, Seite 81 beschreibt den Betriebssystemstart bei Mandrakelinux und wie man ihn effizient einsetzt.

- In *Fortgeschrittene Anwender* werden wir schließlich in Bereiche vordringen, mit denen sich nur die mutigen oder sehr erfahrenen Benutzer beschäftigen werden. Wir werden Sie mit den nötigen Schritten zum Durchbauen und Installieren freier Software aus den Quellen vertraut machen (*Kompilieren und Installieren Freier Software*, Seite 83). Beim Lesen des Kapitels werden Sie sicher die Lust am Ausprobieren spüren, auch wenn Ihnen das Ganze zuerst zu schwer vorkommt. Schlussendlich, werden Sie durch die Informationen im letzten Kapitel (*Konfigurieren und Installieren neuer Kerne*, Seite 99) in der Lage versetzt, ihr GNU/Linux vollkommen zu beherrschen. Nach der Lektüre und der Anwendung der in diesem Kapitel vorhandenen Informationen können Sie getrost damit beginnen, Windows®-Benutzer zu GNU/Linux zu bekehren (falls Sie das nicht schon getan haben!).

3 Anmerkungen des Herausgebers

Einer der Grundpfeiler der „Open Source“-Bewegung ist die Mitarbeit von freiwilligen Helfern. Die Aktualisierung der Mandrakelinux-Dokumentation ist sehr aufwendig und das Dokumentations-Team sucht ständig Helfer in den folgenden Bereichen:

- Verfassen oder Aktualisieren;
- Übersetzen;
- Editieren von Kopien;
- XML/XSLT-Programmierung.

Falls Ihnen genug Zeit zur Verfügung steht, können Sie gerne ein ganzes Kapitel neu schreiben oder aktualisieren; falls Sie eine fremde Sprache sprechen, helfen Sie uns bei der Übersetzung der Handbücher. Neue Ideen zur Verbesserung des Inhaltes sind uns ebenso willkommen wie die Hilfe bei der Verbesserung des Borges Documentation Management System (`http://www.mandrakelinux.com/en/doc/project/Borges`). Bitte benachrichtigen Sie uns auch, wenn Sie „nur“ einen Tippfehler finden, so dass wir ihn beseitigen können!

Für weitere Informationen zur Dokumentation von Mandrakelinux sprechen Sie bitte den Dokumentations-Koordinator (`mailto:documentation@mandrakesoft.com`) an oder schauen Sie sich die Website des Mandrakelinux Documentation Projekts (`http://www.mandrakelinux.com/en/doc/project/`) an.

4 Konventionen, die in diesem Buch benutzt werden

4.1 Schriftsatz-Konventionen

Um einige spezielle Wörter vom normalen Fließtext hervorzuheben, werden verschiedene Schriftsätze verwendet. In der folgenden Tabelle finden Sie Beispiele von speziellen Wörtern und der Gruppen, für die diese stehen sowie Beispiele für den jeweils verwendeten Schriftsatz.

Beispiel	**Bedeutung**
Inode	Diese Formatierung wird dazu verwendet, einen technischen Begriff hervorzuheben, der im *Glossar*, Seite 109 erklärt wird.
`ls -lta`	Dieser Schriftsatz kennzeichnet Kommandos und Parameter von Kommandos. Er gilt auch für Optionen und Dateinamen (siehe Abschnitt *Kommando-Übersicht*, Seite 4).

Beispiel	Bedeutung
ls(1)	Hinweis auf eine man-Seite (Manual = Handbuch). Zum Ansehen in der Textkonsole tippen Sie einfach `man 1 ls`.
`$ ls *.pid`	Dies wird für Bildschirmauszüge benutzt. Darin sind sowohl Interaktionen mit Ihrem Rechner, als auch Programm-Listings enthalten.
`localhost`	Dieser Schriftsatz wird für alle Daten verwendet, die in keine der oben genannten Kategorien fallen. Beispielsweise könnte dies ein Schlüsselwort aus einer Konfigurationsdatei sein.
Konqueror	So werden Applikationen gesetzt. Je nach Kontext können zwar Name und zugehöriger Befehl einer Anwendung gleich lauten, werden aber unterschiedlich formatiert. So werden die meisten Befehle in kleinen Buchstaben geschrieben während die Namen der Anwendungen meist mit einem Großbuchstaben beginnen.
Datei	Dies wird für Menü-Einträge und Schaltflächen verwendet. Der unterstrichene Buchstabe kennzeichnet einen vorhandenen Tastaturkürzel, mit dem man bei gleichzeitigem Drücken der Taste **Alt** den Menüeintrag / die Schaltfläche aktivieren kann.
SCSI-Bus	Dies bezeichnet den Rechner oder einen Teil davon.
Le petit chaperon rouge	So werden Ausdrücke gekennzeichnet, die in einer anderen Sprache sind, als die, in der das Buch geschrieben wurde.
Achtung!	Dies ist natürlich nur für einige spezielle Warnungen reserviert um die Bedeutung der Wörter deutlich zu machen. Bitte laut lesen **:-)**

Dieses Symbol ziert eine Notiz. Es handelt sich dabei meistens um eine Bemerkung, die im aktuellen Kontext Zusatzinformationen bietet.

Dieses Symbol weist auf einen Tipp hin. Es kann ein allgemeiner Hinweis zur Ausführung einer bestimmten Aktion sein oder auch ein Hinweis auf eine Möglichkeit, wie Sie sich die Arbeit an dieser Stelle erleichtern können, z.B. mit Tastaturkürzeln.

Sollten Sie dieses Symbol entdecken, müssen Sie vorsichtig sein, es weist auf einen wichtigen Zusammenhang hin, auf den Sie unbedingt achten sollten.

4.2 Allgemeine Konventionen

4.2.1 Kommando-Übersicht

Das folgende Beispiel zeigt Ihnen die Zeichen, die wir normalerweise benutzen werden, um ein Kommando zu erklären:

```
Kommando <Nicht-Literal> [--option={arg1,arg2,arg3}]
[optionale Argumente ...]
```

Diese Konventionen stellen einen Standard dar und Sie werden sie wahrscheinlich auch an anderen Stellen finden, wie beispielsweise den Handbuchauszügen des Systems (auch man-Seiten genannt).

Die spitzen Klammern „<“ (kleiner als) und „>“ (größer als) bezeichnen ein **Pflicht-Argument**, das Sie nicht wörtlich abschreiben dürfen, sondern an Ihre Bedürfnissen anpassen müssen. So steht etwa `<Dateiname>` für den Namen einer tatsächlich existierenden Datei. Wenn dies `bla.txt` ist, so sollten sie auch genau dies einsetzen und nicht `<bla.txt>` oder gar `<Dateiname>`.

Die eckigen Klammern „[“ und „]“ bezeichnen optionale Parameter, die Sie beim Kommando-Aufruf verwenden können, aber nicht müssen.

Die Punktreihe („...“) bedeutet, dass beliebig viele Optionen benutzt werden können.

Die geschweiften Klammern „{“ und „}“ beinhalten Argumente, die Sie an dieser Stelle benutzen können. Eines davon muss hier stehen.

4.2.2 Tasten und Menüs

Sie werden manchmal aufgefordert Tastenkombinationen zu drücken. Sie sehen im Text etwa **Strg**-**R**. Das bedeutet, dass Sie die Taste **Strg** drücken sollen und so lange gedrückt halten müssen, bis Sie die Taste **R** gleichzeitig gedrückt haben. Entsprechendes gilt für die **Alt**- und die **Umschalt**-Taste.

Es kann Ihnen auch folgendes begegnen: Bearbeiten→Ersetzen … (**Strg-R**) Das bedeutet: Wählen Sie im Menü Bearbeiten des jeweiligen Programms den Menüpunkt Ersetzen …. Weiterhin werden Sie informiert, dass Sie alternativ auch die Tastenkombination **Strg**-**R** drücken können, um die gleiche Aktion auszulösen.

4.2.3 Generische System-Benutzer

Sofern möglich werden wir zwei generische Benutzer in unseren Beispielen benutzen:

Franz Mustermann	Dies ist der Beispiel-Benutzer, der in den meisten Beispielen dieses Buches benutzt wird.
Birgit Mustermann	Dieser Benutzer kann später vom System-Administrator angelegt werden und wird hier manchmal zur Unterscheidung in den Beispielen benutzt.

Kapitel 1. Grundlegende UNIX Konzepte

Der Name „UNIX®“ mag Einigen von Ihnen bekannt vorkommen, vielleicht arbeiten Sie sogar im Büro mit einem UNIX® System. In diesem Fall wird dieser Abschnitt nicht besonders interessant für Sie sein.

Alle Anderen, die noch nie ein UNIX® System benutzt haben, sollten diesen Abschnitt unbedingt sorgfältig lesen. Das Verständnis der hier vorgestellten Konzepte wird Ihnen eine überraschend große Anzahl der typischen Fragen beantworten, die normalerweise von Einsteigern in die Welt von **GNU/Linux** gestellt werden. Auch viele der Probleme, die Sie möglicherweise in der Zukunft haben werden, lassen sich mit der richtigen Anwendung dieser Konzepte lösen.

1.1 Benutzer und Gruppen

Da es alle anderen Konzepte direkt beeinflusst, beschäftigen wir uns in diesem Kapitel mit dem überaus wichtigen Konzept der Benutzer und Gruppen.

Linux ist ein echtes Mehrbenutzersystem (engl. ***Multiuser***-System), d.h., zum Arbeiten an Ihrem GNU/Linux-Computer brauchen Sie ein Benutzerkonto (engl. ***Account*** genannt). Während der Installation haben Sie im Schritt „Einrichtung eines Benutzers“ also genauer gesagt ein Benutzerkonto angelegt. Falls Sie sich nicht erinnern, Sie wurden zur Eingabe folgender Daten aufgefordert:

- den „vollen Namen“ (der völlig beliebig sein kann)
- einen Namen zur ***Anmeldung***
- und ein ***Passwort***.

Die beiden wichtigen Parameter für das System sind der Anmeldename, das sogenannte „Benutzerkennzeichen“ (oft auch als „Login“ bezeichnet) und das Passwort. Beide werden unbedingt für den Zugang zu Ihrem System benötigt.

Beim Erstellen eines Benutzerkontos wird gleichzeitig eine Standardgruppe angelegt . Später werden wir zeigen, wie Gruppen bei der gemeinsamen Nutzung von Dateien mit anderen Benutzerkonten eine wichtige Rolle spielen. Eine Gruppe kann beliebig viele Benutzerkonten umfassen. Diese Einteilung in Gruppen ist eine oft benutzte Praxis in großen Netzwerken, z.B. in Universitäten, wo man eine Gruppe pro Fakultät und eine Gruppe für die Dozenten einrichten könnte. Die Einteilung funktioniert aber auch so, dass ein Benutzer in mehreren Gruppen gleichzeitig sein kann. So kann z.B. ein Physik-Professor gleichzeitig Mitglied der Dozentengruppe und der Gruppe der physikalischen Fakultät sein.

Da Sie jetzt die grundlegenden Hintergrund-Informationen haben, lassen Sie uns nun auf die Anmeldung im System eingehen.

Falls beim Booten des Systems automatisch das grafische Interface (X) gestartet wird sollte Ihr Anmeldeschirm etwa so wie Abbildung 1-1 aussehen.

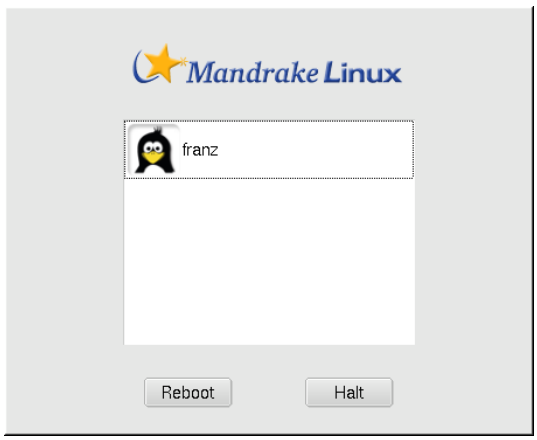


Abbildung 1-1: Anmeldung im grafischen Modus

Zum Anmelden müssen Sie zuerst Ihr Benutzerkennzeichen aus der Liste auswählen. Danach erscheint ein neues Fenster und fragt nach Ihrem Passwort . Das Passwort geben Sie „blind“ ein, da anstelle der eingege-

benen Zeichen nur * (Sterne) angezeigt werden. Nachdem Sie dann noch Ihren bevorzugten Fenstermanager ausgewählt haben, klicken Sie auf die Schaltfläche Anmelden.

Auf der Konsole (oder auch im „Text"-Modus) sehen Sie ein ähnliches Bild vor sich:

```
Mandrakelinux Release 10.1 (CodeName) for i586 Kernel 2.6.8-3mdk on an i686 / tty1
[machine_name] login:
```

Zur Anmeldung geben Sie Ihren Anmeldenamen hinter dem Login: Prompt ein und drücken auf **Enter**. Als Nächstes zeigt Ihnen das Anmeldeprogramm (login) einen Password: Prompt und wartet auf die Eingabe Ihres Passwortes. Beachten Sie bitte, dass Sie im Konsolenmodus bei der Eingabe des Passwortes keinerlei Zeichen auf dem Bildschirm sehen werden.

Sie können sich mit dem gleichen Benutzerkennzeichen mehrmals auf verschiedenen ***Konsolen*** und unter X anmelden. Jede Sitzung ist vollkommen autark und Sie können sogar verschiedene X-Sitzungen gleichzeitig starten (obwohl das wegen des großen Verbrauchs an Ressourcen nicht empfehlenswert ist). Standardmäßig sind in Mandrakelinux sechs ***virtuelle Konsolen*** eingerichtet, dazu die reservierte Konsole für die grafische Oberfläche. Das Umschalten zwischen den Konsolen geschieht mit der Tastenkombination **Strg-Alt-F<n>**, wobei <n> die Nummer der gewünschten Konsole darstellt. Die grafische Oberfläche liegt standardmäßig auf Konsole 7. Um nun zu Konsole 2 umzuschalten, drücken Sie gleichzeitig die Tasten `Strg`, `Alt` und `F2`.

Während der Installation wurden Sie von DrakX dazu aufgefordert, ein Passwort für ein ganz spezielles Benutzerkennzeichen einzugeben: `root` . `root` ist der System-Administrator (im Allgemeinen also Sie selbst). Zum Schutz Ihres Systems sollten Sie dem Benutzer `root` ein wirklich gutes und nicht einfach zu erratendes Passwort geben!

Falls Sie sich regelmäßig als `root` anmelden, steigt natürlich die Gefahr, dass Sie durch einen kleine Unachtsamkeit Ihr gesamtes System unbenutzbar machen — ein kleiner Fehler reicht aus. Besonders gefährlich ist es, wenn Sie dem `root` kein Passwort eingerichtet haben. Dann kann **jeder** Benutzer **jeden** Teil Ihres Systems (und auch der anderen Betriebssysteme – soweit vorhanden – auf Ihrem Rechner!) verändern. Keine gute Idee!

Zu bemerken wäre noch, dass das System Sie nicht unter Ihrem Benutzerkennzeichen sondern durch eine dem Benutzerkennzeichen zugeordnete Nummer identifiziert: die *User ID* (kurz UID) . Dem entsprechend werden die Gruppen auch per *Group ID* (GID) identifiziert.

1.2 Grundlagen des Dateisystems

In der Arbeit mit Dateien gibt es zwischen GNU/Linux und Windows® sowie den meisten anderen ***Betriebssystemen*** wesentliche Unterschiede. Dieses Kapitel wird die hauptsächlichen Merkmale aufzeigen. Weitere Informationen erhalten Sie im Kapitel *GNU/Linux Dateisysteme*, Seite 65.

Die wesentlichen Unterschiede resultieren aus der Tatsache, dass Linux ein Mehrbenutzer-System ist: jede Datei ist alleiniger Besitz eines Benutzerkennzeichens und einer Gruppe. Jedes Benutzerhonto besitzt ein persönliches Verzeichnis (genannt ***Home-Verzeichnis***). Das Benutzerkennzeichen ist der Eigentümer des Verzeichnisses und aller darin befindlichen Dateien.

Das wäre aber ungenügend als alleiniges Zeichen des Besitzes an einer Datei. Der Eigentümer einer Datei kann ihr verschiedene **Rechte** zuweisen. Diese Rechte unterscheiden zwischen drei Benutzerkategorien: dem **Besitzer** der Datei, der **Gruppe**, der diese Datei gehört (und dazu zählen alle Mitglieder der Gruppe, daher auch der Name ***Besitzergruppe***) und schließlich der Rest der Welt, die **Anderen**. Dazu gehören alle Benutzer, die weder der Besitzer noch Mitglieder der Besitzergruppe sind.

Es gibt drei verschiedene Rechte:

1. *Read* (`r`, lesen) erlaubt dem Benutzer, den Inhalt der Datei zu lesen. Für ein Verzeichnis bedeutet das, der Benutzer kann den Inhalt auflisten (also die einzelnen Dateien).
2. *Write* (`w`, schreiben) erlaubt das Verändern des Inhaltes der Datei. Bei einem Verzeichnis bedeutet das, der Benutzer kann Dateien in das Verzeichnis einfügen und daraus entfernen, auch wenn er nicht der Besitzer dieser Dateien ist.
3. *eXecute* (`x`, ausführen) erlaubt die Ausführung einer Datei (normalerweise wird dieses Recht nur bei ausführbaren Dateien gesetzt). Bei einem Verzeichnis bedeutet dieses Recht, dass ein Benutzer es betreten

bzw. durchqueren (***traverse***) darf. Das ist nicht zu verwechseln mit dem Lese-Recht: Sie können z.B. das Recht zum Betreten/Durchqueren eines Verzeichnisses besitzen, nicht aber das Leserecht für die darin enthaltenen Dateien!

Jede beliebige Kombination von Rechten ist möglich. Sie könnten sich z.B. das alleinige Leserecht einer Datei erteilen, den Zugriff aller anderen Benutzer aber sperren. Als Besitzer der Datei können Sie natürlich auch die Besitzergruppe ändern (allerdings nur, wenn Sie auch Mitglied der neuen Gruppe sind).

Lassen Sie uns das an einem Beispiel betrachten. Im Folgenden sehen Sie die Ausgabe auf den Befehl `ls -l` auf der ***Befehlszeile***:

```
$ ls -l
total 1
-rw-r-----   1 franz    users           0 Jul  8 14:11 eine_datei
drwxr-xr--   2 birgit    users        1024 Jul  8 14:11 ein_verzeichnis/
$
```

Die Bedeutung des Befehls `ls -l` (von links nach rechts):

- Die ersten zehn Zeichen zeigen den Dateityp und die mit ihr verbundenen Rechte. Das erste Zeichen ist der Dateityp: bei einer normalen Datei ist es ein Minuszeichen (`-`). Ein Verzeichnis wird durch ein `d` angezeigt. Es gibt noch weitere Dateitypen, die wir später behandeln. Die nun folgenden neun Zeichen zeigen die für diese Datei vergebenen Rechte an. Sie gliedern sich in 3 Gruppen zu je drei Rechten. Die erste Dreiergruppe bezeichnet die Rechte des Dateibesitzers, die zweite Dreiergruppe die Rechte der Besitzergruppe und die letzte Dreiergruppe zeigt die Rechte aller anderen Benutzer. Ein Minuszeichen (`-`) bedeutet, dass das entsprechende Recht nicht vergeben wurde.
- Es folgt die Anzahl der Verweise auf eine Datei. Später werden wir sehen, dass die eindeutige Identifikation einer Datei nicht durch ihren Namen, sondern durch eine Nummer (die ***Inode Nummer***) erfolgt, sodass es möglich ist, dass eine Datei unter mehreren Namen angesprochen werden kann. Für ein Verzeichnis hat die Anzahl der Verweise eine spezielle Bedeutung, auf welche ebenfalls später eingegangen wird.
- Die nächsten Informationen sind das Benutzerkennzeichen, das Besizer der Datei ist und der Besitzergruppe.
- Schlussendlich wird die Grösse der Datei(in ***Bytes***) und ihre letzte Veränderungszeit angezeigt. Der Name der Datei oder des Verzeichnisses wird am Ende der Zeile dargestellt.

Schauen wir uns mal die Zugriffsrechte der Dateien an, die jede besitzt. Zuerst streichen wir den ersten Buchstaben, der für den Typ steht, und bekommen so für `eine_Datei`die folgenden Berechtigungen: `rw-r-----`. Hier nun eine Auflistung von Zugriffsrechten.

- Die ersten drei Buchstaben (`rw-`) sind die Rechte des Benutzerkennzeichen des Besitzers, welcher in diesem Zusammenhang franz ist. Folglich hat franz das Recht, die Datei zu lesen (`r`), ihren Inhalt zu editieren (`w`) aber keine Rechte um sie auszuführen (`-`).
- Die nächsten drei Zeichen (`r--`) sind für alle Benutzer außer franz gedacht, die aber in die gleiche Benutzergruppe (`users`) wie franz gehören. Sie werden die Datei lesen (`r`), aber nicht ändern oder ausführen können (`--`).
- Die letzten drei Zeichen (`---`) sind für alle Benutzer bestimmt, die weder franz noch Mitglied der Benutzergruppe `users` sind. Diese Benutzer haben überhaupt keinen Zugriff auf die Datei.

Für das Verzeichnis `ein_Verzeichnis`, sind die Rechte `rwxr-xr--`, also:

- birgit, Der Eigentümer kann den Inhalt auflisten (`r`), Dateien hinzufügen oder aus dem Verzeichnis entfernen (`w`) und es betreten/durchqueren (`x`).
- Jedes Mitglied der Gruppe `users`, außer birgit kann Dateien des Verzeichnisses auflisten (`r`), aber weder Dateien entfernen oder hinzufügen (`-`) noch das Verzeichnis betreten/durchqueren (`x`).
- Alle anderen Benutzer werden nur den Inhalt des Verzeichnisses auflisten können (`r`). Da sie keine `wx` Rechte haben, ist ihnen das Beschreiben und Betreten/Durchqueren des Verzeichnisses nicht möglich.

Es gibt allerdings **eine** einzige Ausnahme von diesen Regeln: `root`. `root` kann die Attribute (Rechte, Besitzer und Besitzergruppe) aller Dateien und Verzeichnisse ändern - selbst wenn er nicht der Eigentümer ist - und

sich so die Besitzerschaft einer Datei sichern. Er kann Dateien lesen, auf die er keinen Zugriff hat, in Verzeichnisse wechseln, auf die er normalerweise keinen Zugriff hätte, und so weiter. Und wenn er einige Rechte ändern möchte, ist dies ebenfalls möglich. `root` hat die komplette Kontrolle über das System, was sehr viel Vertrauen in die Person mit dem `root` Passwort voraussetzt.

Schließlich sollte man sich noch den Unterschied in den Dateinamen zwischen UNIX® und Windows® ansehen. Da ist man unter UNIX® viel flexibler und nicht so eingeengt:

- Ein Dateiname kann alle möglichen Zeichen enthalten, einschließlich der nicht druckbaren, ausgenommen das ASCII-Zeichens 0 (Ende eines Strings) und /, das ein Verzeichnis-Seperator ist. Überdies unterscheidet UNIX® Groß- und Kleinschreibung: Die Dateien `readme` und `Readme` sind verschieden, da `r` und `R` zwei unterschiedliche Zeichen in UNIX®-basierenden Betriebssystemen sind.
- Wie Sie bemerkt haben, muss ein Dateiname keine Erweiterung erhalten, es sei denn, Sie wünschen dies so. Dateierweiterungen bieten unter GNU/Linux, wie in der Mehrheit existierender Betriebssysteme, keinerlei Rückschluss auf deren Inhalt. Sogenannte „Dateierweiterungen“ sind allerdings eine nützliche Einrichtung. Der Punkt (`.`) ist unter UNIX® ein Zeichen wie jedes Andere. Unter UNIX® hat er nur am Beginn von Dateinamen eine spezielle Bedeutung: Dateien, deren Name mit einem Punkt beginnt, sind „versteckte Dateien“[1]. Das gleiche gilt für Verzeichnisse, deren Name mit `.` beginnt.

Etliche grafische Anwendungen (Datei-Manager, Office Anwendungen, etc.) benutzen jedoch Dateiendungen zur Erkennung von Dateien, die sie selbst erstellt haben bzw. verarbeiten können. Darum ist es eine gute Idee, Dateierweiterungen für die Programme zu verwenden, die dies vorschlagen.

1.3 Prozesse

Ein ***Prozess*** definiert eine Instanz eines ausgeführten Programms und seiner ***Umgebung***. Wir wollen hier nur die herausragendsten Unterschiede zwischen GNU/Linux und Windows® erwähnen (weitere Informationen erhalten Sie in *Prozesskontrolle*, Seite 51).

Der bedeutendste Unterschied hängt mit dem **Benutzer**-Konzept zusammen: jeder Prozess ist mit den Rechten des Benutzerkennzeichens verbunden, der ihn gestartet hat. Intern identifiziert das System Prozesse anhand einer eindeutigen Nummer, genannt *ProzessID*, oder PID. Anhand dieser PID, weiß das System, wer (welches Benutzerkennzeichen) den Vorgang gestartet hat, sowie eine Reihe andere Informationen und muss nur noch die „Validität“ des Prozesses überprüfen. Wenn wir also unser Beispiel mit `eine_datei` ansehen, kann birgit die Datei zwar im Nur-Lese-Modus (engl. ***read-only Mode***), nicht aber im Schreib-Lese-Modus (engl. ***read-write Mode***) öffnen, weil die Rechte der Datei dieses verbieten. Auch in diesem Fall bildet `root` eine Ausnahme.

Durch dieses System ist GNU/Linux weitgehend immun gegen Viren. Um funktionieren zu können, brauchen Viren ausführbare Dateien. Als einfacher Benutzer hat man keinen Schreibzugriff auf gefährdete Systemdateien, wodurch das Risiko sehr eingeschränkt wird. Generell kann man sagen, dass Viren in UNIX® sehr selten vorkommen. Es sind nur wenige Viren für Linux bekannt und selbst die sind harmlos, wenn sie durch einen normalen Benutzer ausgeführt werden. Nur ein Benutzer, der sich als `root` einloggt, kann mit ihnen ein System zerstören.

Interessanterweise gibt es trotzdem Anti-Viren Software für GNU/Linux aber vorwiegend für DOS/Windows® Dateien. Warum laufen diese Anti-Viren-Programme für DOS/Windows® unter Linux? Immer öfter werden GNU/Linux-Dateiserver mit Hilfe des Samba Software Paketes als Dateiserver für Windows® eingesetzt (siehe das Kapitel Gemeinsame Nutzung von Daten und Druckern des *Server Schnellkonfigurationshandbuch*).

Linux macht es leicht, Prozesse zu kontrollieren. Ein Weg führt über „Signale“. Diese erlauben es dem Benutzer, Prozesse abzubrechen, indem das zugehörige Signal zu dem entsprechenden Prozess gesandt wird. Es ist jedoch nur dem jeweiligem Eigentümer erlaubt, Signale zu seinen Prozessen zu senden. Mit der Ausnahme von `root` verbietet es UNIX® Signale an Prozesse zu senden, die ein anderer Anwender gestartet hat.

1. Standardmäßig werden versteckte Dateien nicht im Dateimanager angezeigt, es sei denn, Sie ändern diese Einstellung. Zur Auflistung von versteckten Dateien in einem Terminal müssen Sie beii dem Kommando `ls` die option `-a`eingeben. Generell beinhalten diese Dateien Konfigurationseinstellungen. Schauen Sie sich als Beispiele in Ihrem `home/`-Verzeichnis einmal die Verzeichnisse `.mozilla` oder `.openoffice` an.

In *Prozesskontrolle*, Seite 51 können Sie lernen, wie Sie die PID eines Prozesses erfahren und Signale zu ihm senden.

1.4 Eine kurze Einführung in die Kommandozeile

Die Kommandozeile ist der direkteste Weg um Ihrem Rechner Befehle zu erteilen. Wenn Sie die GNU/Linux-Kommandozeile benutzen, werden Sie bemerken, dass diese sehr viel leistungsfähiger ist als andere Kommandozeilen, die Sie vorher verwendet haben. Diese Stärke wurde erreicht, weil Sie nicht nur auf alle X Anwendungen, sondern auch auf Tausende von Textwerkzeugen Zugriff haben, zu denen es keine gleichwertigen Programme mit grafischer Benutzeroberfläche gibt. Die mächtigkeit die durch Verwendung von Kommandozeilenoptionen entsteht lässt sich nur schwer in Form von Schaltflächen und Menüs nachbilden.

Zugegebenermaßen brauchen die meisten Anwender am Anfang ein wenig Hilfe. Das Erste, was Sie tun müssen, wenn Sie noch nicht im Konsolenmodus gearbeitet haben, ist einen Terminal-Emulator zu starten. Öffnen Sie das **Mandrakelinux**-Menü. Dort finden Sie eine Reihe von Emulatoren unter System+Terminals. Wählen Sie einen aus, etwa Konsole oder XTerm. Abhängig von Ihrer Benutzeroberfläche existiert auch ein Symbol in der Startleiste, über das eine der Anwendungen gestartet werden kann. (Abbildung 1-2).

Abbildung 1-2: Das Terminal-Symbol in dem KDE Startleiste

Wenn Sie den Terminal-Emulator gestartet haben, benutzen Sie eigentlich eine shell. Das ist der Name des Programms, dass Ihnen die Interaktion mit dem Rechner erlaubt. Sie sehen jetzt eine Eingabeaufforderung (engl. ***Prompt***) vor sich:

```
[franz@localhost franz]$
```

Dies zeigt an, dass Ihr Benutzerkennzeichen franz und der Name Ihres Rechners `localhost` ist (das ist der Fall, wenn Ihr PC kein Teil eines Netzwerks ist). Hinter der Eingabeaufforderung ist Platz zum tippen Ihrer Befehle. Beachten Sie, dass das $-Zeichen am Ende der Eingabeaufforderung durch ein # ersetzt wird, wenn Sie als `root` angemeldet sind (das trifft nur auf den Auslieferungszustand Ihrer Distribution zu, da Sie ja alle diese Details in GNU/Linux nach Ihren Vorlieben anpassen können). Um in den Kontext des privilegierten Benutzerkennzeichens `root` zu wechseln, tippen Sie nach dem Aufrufen der shell den Befehl `su`.

```
# Geben Sie das Root-Passwort ein (es erscheint nicht auf dem Bildschirm)
[franz@localhost franz]$ su
Password:
# mit exit oder Strg-D kommt man wieder zum unprivilegierten Benutzerkontext zurück
[root@localhost franz]# exit
[franz@localhost franz]$
```

An allen anderen Stellen des Buches wird die Eingabeaufforderung durch $ symbolisiert, egal, ob Sie ein normaler Benutzer oder `root` sind. Sie werden darauf hingewiesen wenn Sie sich als `root` anmelden müssen, um einen Befehl ausführen zu können. Also erinnern Sie sich dann an das Kommando `su`.

Wenn Sie eine shell zum ersten Mal ***starten***, befinden Sie sich normalerweise im persönlichem Verzeichnis (`home/`). Um das Verzeichnis, in dem Sie sich gerade befinden. anzuzeigen, geben Sie das Kommando `pwd` (Abkürzung für *Print Working Directory*) ein:

```
$ pwd
/home/franz
```

Als nächstes werden wir einige grundlegende Befehle erläutern, die, wie Sie sehen werden, sehr nützlich sein können.

1.4.1 cd: Wechsle Verzeichnis

Der Befehl `cd` ist ähnlich wie unter DOS, allerdings mit einigen Extras. Dieser Befehl wechselt das aktuelle Verzeichnis. Sie können `.` und `..` verwenden, was für das aktuelle und das übergeordnete Verzeichnis steht. Geben Sie nur `cd` ein, gelangen Sie in ihr persönliches Verzeichnis. `cd -` bringt Sie zum zuletzt verwendeten Verzeichnis zurück. Und letztlich können Sie zum persönlichen Verzeichnis von birgit gelangen, indem Sie `cd ~birgit` eingeben. (`~` für sich allein steht für Ihr persönliches Verzeichnis). Beachten Sie, dass unprivilegierte Benutzerkennzeichen standardmäßig keinen Zugriff auf persönliche Verzeichnisse anderer Kennzeichen haben (es sei denn, deren Benutzer haben das ausdrücklich so eingerichtet). Eine Ausnahme bider wie immer daas privilegierte Kennzeichen `root`. Zum Üben werden Sie `root` und probieren diese Kommandos:

```
$ pwd
/root
$ cd /usr/share/doc/HOWTO
$ pwd
/usr/share/doc/HOWTO
$ cd ../FAQ-Linux
$ pwd
/usr/share/doc/FAQ-Linux
$ cd ../../../lib
$ pwd
/usr/lib
$ cd ~birgit
$ pwd
/home/birgit
$ cd
$ pwd
/root
```

Geben Sie nun Ihre Privilegien wieder ab und kehren in den Kontext eines einfachen Anwenders zurück: Geben Sie dafür den Befehl `exit` ein (oder verwenden Sie die Tastenkombination **Strg**-**D**).

1.4.2 Einige Umgebungsvariablen und das Echo-Kommando

Alle Prozesse haben ihre ***Umgebungsvariablen*** und die shell erlaubt es Ihnen, sich diese direkt mit `echo` anzeigen zu lassen. Einige interessante Variablen sind:

1. `HOME`: Diese Umgebungsvariable enthält eine Zeichenkette, die dem Pfad zu Ihrem perspersönlichen Verzeichnis entspricht.
2. `PATH`: Diese Variable beinhaltet eine Liste aller Verzeichnisse in denen die shell nach ausführbaren Dateien suchen soll, wenn ein Kommando eingeben wird. Beachten Sie, dass eine UNIX® shell standardmäßig **nicht** im aktuellen Verzeichnis nach ausführbaren Dateien sucht (im Gegensatz etwa zu DOS)!
3. `USERNAME`: Diese Variable beinhaltet Ihren Anmelde-Namen, also Ihr Benutzerkennzeichen.
4. `UID`: das ist Ihre User-ID.
5. `PS1`: Diese Variable entscheidet, wie Ihre Eingabeaufforderung aussieht. Es handelt sich oft um eine Kombination von speziellen Sequenzen. Sie können nähere Informationen dazu im bash(1) ***Handbuchauszug*** nachlesen. Geben Sie dazu das Kommando `man bash` im Terminal ein.

Um eine shell dazu zu bringen eine Variable auszugeben, müssen Sie ein `$` vor den Variablennamen setzen. Hier ein Beispiel mit `echo`

```
$ echo Hello
Hello
$ echo $HOME
/home/franz
$ echo $USERNAME
franz
$ echo Hello $USERNAME
Hello franz
$ cd /usr
$ pwd
/usr
$ cd $HOME
```

```
$ pwd
/home/franz
```

Wie Sie bemerken, ersetzt die shell die Variable, bevor sie den Befehl ausführt. Ansonsten würde der Befehl `cd $HOME` nicht funktionieren. Tatsächlich, ersetzt die shell erst `$HOME` mit dem entsprechenden Wert (`/home/franz`), sodass die Eingabezeile `cd /home/franz` zeigt, was beabsichtigt war. Das gleiche passiert in dem Beispiel `echo $USERNAME`.

Falls eine der Umgebungsvariablen nicht existiert, können Sie diese mit dem Befehl `export ENV_VAR_NAME=value` temporär anlegen. Danach sollten Sie kontrollieren, ob sie angelegt wurde:

```
$ export USERNAME=franz $ echo USERNAME franz
```

1.4.3 cat: Gibt den Inhalt einer oder mehrerer Dateien aus

Wie schon gesagt, macht dieses Kommando folgendes: der Inhalt von einer oder mehreren Dateien wird auf die Standard-Ausgabe, normalerweise ist das der Bildschirm, ausgegeben:

```
$ cat /etc/fstab
/dev/hda5 / ext2 defaults 1 1
/dev/hda6 /home ext2 defaults 1 2
/dev/hda7 swap swap defaults 0 0
/dev/hda8 /usr ext2 defaults 1 2
/dev/fd0 /mnt/floppy auto sync,user,noauto,nosuid,nodev 0 0
none /proc proc defaults 0 0
none /dev/pts devpts mode=0620 0 0
/dev/cdrom /mnt/cdrom auto user,noauto,nosuid,exec,nodev,ro 0 0
$ cd /etc
$ cat modules.conf shells
alias parport_lowlevel parport_pc
pre-install plip modprobe parport_pc ; echo 7 > /proc/parport/0/irq
#pre-install pcmcia_core /etc/rc.d/init.d/pcmcia start
#alias char-major-14 sound
alias sound esssolo1
keep
/bin/zsh
/bin/bash
/bin/sh
/bin/tcsh
/bin/csh
/bin/ash
/bin/bsh
/usr/bin/zsh
```

1.4.4 less: Ein Pager

Der Name ist ein Wortspiel. Er bezieht sich auf den ersten Pager, den es unter UNIX® gab: `more`. Ein ***Pager*** ist ein Programm, das dem Benutzer erlaubt, große Textdateien seitenweise anzuzeigen (genauer: Bildschirmweise) Der Grund, warum wir `less` erklären, anstatt `more` ist, dass `less` sich viel intuitiver benutzen läßt. Sie sollten `less` benutzen, um große Dateien anzusehen, die sich nicht auf einem einzigen Bildschirm anzeigen lassen. Beispielsweise:

```
less /etc/termcap
```

Um durch diese Datei zu „blättern“, benutzen Sie die Aufwärts- und Abwärtstaste. Benutzen Sie **q** um `less` zu beenden. `less` kann weit mehr, als das: durch drücken der Taste `h` erhalten Sie einen Hilfeschirm mit den verschiedenen Optionen angezeigt.

1.4.5 ls: Dateiauflistung

Der Befehl `ls` (*LiSt*) entspricht etwa dem Kommando `dir` unter DOS, kann jedoch viel mehr. Tatsächlich ist er leistungsfähiger, da Dateien unter UNIX® mehr Eigenschaften besitzen. Die Syntax für `ls` lautet:

```
ls [optionen] [datei|verzeichnis] [datei|verzeichnis...]
```

Wenn kein Verzeichnis in der Kommandozeile angegeben wurde, listet `ls` alle Dateien im aktuellem Verzeichnis auf. Seine Optionen sind sehr zahlreich, sodass wir nur eine kleine Anzahl davon behandeln:

- `-a`: Zeigt alle Dateien, einschließlich der ***versteckten Dateien*** Wie Sie vielleicht schon wissen, beginnen in UNIX® alle versteckten Dateien mit einem Punkt „`.`“. Bei der Option `-A` werden im Unterschied dazu die Dateien „.“ und „..“ (also die Verweise auf das aktuelle und das übergeordnete Verzeichnis) nicht angezeigt.
- `-R`: listet rekursiv alle Dateien und Unterverzeichnisse von Verzeichnissen auf, die in der Kommandozeile angegeben wurden.
- `-s`: gibt jeweils die Dateigröße in Kilobyte aus
- `-l`: gibt nähere Informationen für jede Datei an, wie etwa die Rechte, den Eigentümer und die Besitzergruppe, die Größe und die Zeit des letzten Zugriffs.
- `-i`: zeigt die Nummer des Inode (die Nummer zur eindeutigen Identifikation einer Datei, siehe auch Kapitel *GNU/Linux Dateisysteme*, Seite 65) neben der Datei an.
- `-d`: behandelt Verzeichnisse so, als wären sie normale Dateien, d.h. zeigt nur die Namen der Verzeichnisse, nicht aber deren Inhalt an.

Hier einige Beispiele:

- `ls -R`: listet rekursiv den Inhalt des aktuellen Verzeichnisses auf.
- `ls -is images/ ..`: listet die Inode-Nummer und die Größe in Kilobytes für jede Datei im Verzeichnis `images/` und im Elternverzeichnis des aktuellen Verzeichnises auf.
- `ls -l images/*.png`: listet alle Dateien im Verzeichnis `images/` auf, deren Name mit `.png` endet, inklusive der Datei `.png` falls diese existiert.

1.4.6 Hilfreiche Tastenkombinationen

Es gibt eine Menge von Tastenkombinationen (engl. *Shortcuts*), mit denen man viel Zeit spart. Wir setzen hier voraus, dass Sie die Standard- shell unter Mandrakelinux verwenden: bash. Die Mehrzahl der Tastenkombinationen sollten aber auch mit anderen Shells funbktionieren.

Zuerst: die Pfeil-Tasten. bash speichert eine Reihe von vorher getätigten Kommandos, die mit den Pfeil-Tasten erneut angezeigt werden können. Sie können maximal die Anzahl Befehle zurückblättern, die durch die Umgebungsvariable `HISTSIZE` festgelegt ist. Außerdem werden die Kommandos in Auslieferzustand Ihrer Distribution dauerhaft gespeichert, sodass Sie die in vorhergehenden Sitzungen eingegebenen Befehle nicht verlieren.

Die linke und rechte Pfeil-Taste bewegt den Blinker nach links bzw. rechts und erlauben Ihnen, Ihre Eingaben zu bearbeiten. Aber es gibt mehr Funktionen beim Editieren, als zu bestimmten Punkten zu springen. **Strg-A** und **Strg-E** brint Sie zum Anfang oder Ende der Zeile. Die Tasten **Backspace** und **Entf** bewirken das Erwartete. **Backspace** und **Strg-H** haben die gleiche Wirkung. **Entf** und **Strg-D** desgleichen. **Strg-K** löscht die Zeile von der Blikerposition bis zum Ende und **Strg-W** löscht das Wort vor dem Blinker (das Gleiche wie **Alt-Backspace**).

Strg-D in einer leeren Zeile schließt die aktuelle Sitzung, ein schnellerer Weg als die Eingabe von `exit`. **Strg-C** unterbricht den gerade laufenden Befehl, es sei denn, Sie editieren gerade Ihre Befehlszeile. In diesem Fall wird das Editieren abgebrochen und Sie finden sich an einer Eingabeaufforderung wieder. **Strg-L** löscht den Bildschirm. **Strg-Z** unterbricht einen Befehl temporär. Dieser Kürzel ist sehr hilfreich, wenn Sie das Zeichen „&“ hinter einem Befehl vergessen haben. Beispiel:

```
$ xpdf MeinDokument.pdf
```

Hier können Sie Ihre shell nicht mehr benutzen, da sie durch den Prozess xpdf im Vordergrund blockiert ist. Geben Sie hier zuerst die Kombination **Ctrl**-**Z** und danach den Befehl bg ein. Damit schieben Sie den Prozess in den Hintergrund und haben wieder Zugriff auf die Shell.

Zuletzt zu erwähnen seien noch **Strg**-**S** und **Strg**-**Q**, mit denen die Ausgabe auf dem Bildschirm gesperrt und wieder hergestellt wird. Diese Kombinationen sind nicht sehr gebräuchlich. Allerdings könnten Sie aus Versehen **Strg**-**S** eingeben (**S** und **D** liegen immerhin dicht nebeneinander). Wenn Sie also schreiben und es erscheint nichts mehr auf dem Bildschirm, drücken Sie einfach die Kombination **Strg**-**Q**. Danach werden alle in der Zwischenzeit eingegebenen Zeichen auf einmal auf dem Bildschirm sichtbar.

Kapitel 2. Festplatten und Partitionen

Dieser Bereich bietet Informationen für Benutzer, die ein wenig mehr über die technischen Details ihres Systems wissen wollen. Er enthält eine komplette Beschreibung des Partitionierungsschemas eines PC und wird Ihnen eine wertvolle Hilfe bei der manuellen Partitionierung Ihrer Festplatte sein. Da dieser Vorgang auch automatisch durch das Installationsprogramm erledigt werden kann ist ein Verständnis dieses Abschnitts nicht unbedingt für eine Standardinstallation erforderlich.

2.1 Struktur einer Festplatte

Eine Festplatte ist physikalisch unterteilt in Sektoren, von denen eine definierte Menge wiederum eine Partition bilden. Vereinfacht gesagt, können Sie beliebig viele Partitionen einrichten, bis zu 67 (3 primäre Partitionen und eine sekundäre Partition, die bis zu 64 logische Partitionen enthalten kann). Jede davon wird als separate „Festplatte“ behandelt.

2.1.1 Sektoren

Wiederum vereinfacht kann man eine Festplatte als eine Sammlung von Sektoren verstehen, die die kleinste Einheit einer Festplatte bilden. Die typische Größe eines Sektors ist 512 Bytes. Die Sektoren einer Festplatte mit „n“ Sektoren werden von „0“ bis „n-1“ durchnummeriert.

2.1.2 Partitionen

Die Verwendung mehrerer Partitionen entspricht der Einteilung Ihrer physikalischen Festplatte in viele virtuelle Festplatten und bietet mehrere Vorteile:

- Verschiedene Betriebssysteme benutzen auch verschiedene Festplattenstrukturen (genannt ***Dateisystem***e): so ist es auch bei Windows® und GNU/Linux. Mehrere verschiedene Partitionen auf einer Festplatte erlauben also den Betrieb von mehreren verschiedenen Betriebssystemen auf der gleichen Festplatte.
- Aus Performancegründen kann ein einzelnes Betriebssystem den Betrieb von verschiedenen Dateisystemen für jeweils verschiedene Aufgaben bevorzugen. Ein treffendes Beispiel dafür ist GNU/Linux, das zumindest eine zweite Partition mit der Bezeichnung swap benötigt. Diese wird von der Speicherverwaltung als virtueller Speicher genutzt.
- Auch wenn all Ihre Partitionen das gleiche Dateisystem verwenden, kann es doch empfehlenswert sein, die verschiedenen Teile des Betriebssystems auf verschiedene Partitionen zu verteilen. Eine simple Einteilung wäre beispielsweise die Trennung von persönlichen Daten und Programmen in zwei Partitionen. Das wäre bei einer Aktualisierung des Systems von Vorteil, da man dabei nur die Partition mit den Programmen neu installieren müsste und die persönlichen Daten erhalten blieben.
- Da physikalische Fehler auf Festplatten typischerweise in benachbarten Sektoren auftreten und nicht über die ganze Platte verteilt sind, ist eine Verteilung der Daten auf mehrere Partitionen von Vorteil. Es begrenzt den Datenverlust falls Ihre Festplatte einen physikalischen Schaden erleidet.

Normalerweise gibt der Partitionstyp schon das Dateisystem vor, das benutzt wird. Jedes Betriebssystem ist in der Lage mit einer Menge an Dateisystemen umzugehen, jedoch nicht mit allen existierenden Varianten. Weitere Informationen dazu, welche Typen GNU/Linux kennt, finden Sie in den Kapiteln *Dateisysteme und Einhängpunkte*, Seite 59 und *GNU/Linux Dateisysteme*, Seite 65.

2.1.3 Festlegen der Struktur Ihrer Festplatte

2.1.3.1 Die einfachste Aufteilung

Diese Aufteilung gibt nur zwei verschiedene Partitionen vor: eine für die Auslagerung (swap), die andere für die Daten[1].

Eine sehr gebräuchliche Regel zur Bestimmung der Größe einer Swap-Partition ist die zweifache Größe des physikalisch vorhandenen RAM (Arbeitsspeicher), d.h., bei vorhandenen 128 MB RAM sollte die Swap-Partition 256 MB groß sein. Bei einer Ausstattung mit größerem Arbeitsspeicher (>512 MB) ist diese Regel natürlich nicht mehr so strikt anzuwenden – auch kleinere Swap-Partitionen reichen hier normalerweise aus. Denken Sie auch daran, dass je nach verwendeter Plattform die Größe der Swap-Partition limitiert sein kann. So beträgt die maximale Größe 2GB bei x86, PowerPC and MC680x0; 512MB auf MIPS; 128GB bei Alpha und bis zu 3TB bei Ultrasparc.

2.1.3.2 Ein weiteres oft benutztes Schema

Trennen Sie Daten von Programmen. Normalerweise definiert man eine dritte Partition, als Verzeichnisbaumwurzel (manchmal auch als „Root"-Partition bezeichnet: /). Darin sind die Programme enthalten, die zum Start und zur Verwaltung des Systems benötigt werden.

Also definieren wir nun vier Partitionen:

Swap

Eine swap Partition, deren Größe ungefähr dem doppelten physikalisch vorhandenen RAM entspricht.

Root: /

Die wichtigste Partition. Sie enthält nicht nur die für das System notwendigen Daten und Programme sondern dient auch als Einhängpunkt für andere Partitionen (siehe *Dateisysteme und Einhängpunkte*, Seite 59).

Der Platzbedarf der Verzeichnisbaumwurzel ist sehr gering, 400MB sollten im Allgemeinen genügen. Wenn Sie allerdings kommerzielle Programme installieren wollen, die meist unterhalb `/opt` angesiedelt werden, sollte die Rootpartition entsprechend größer ausfallen. Alternativ können Sie natürlich auch eine separate Partition für `/opt` einrichten.

Statische Daten: `/usr`

Die meisten Programme installieren ihre Programm- und Datendateien hauptsächlich innerhalb `/usr`. Wenn Sie für diese Hierarchie eine separate Partition anlegen, haben Sie es wesentlich leichter, all diese Programme und Daten für andere Rechner des Netzwerkes freizugeben.

Die empfohlene Größe hängt von der Anzahl und dem Platzbedarf der zu installierenden Programme ab und kann von 100MB bei einer Minimal-Installation bis zu mehreren GB für eine Vollinstallation reichen. Ein guter Kompromiss ist (je nach Plattengröße) zwei bis drei GB.

1. das zur Zeit unter GNU/Linux gebräuchliche Dateisystem ist `ext3`

Persönliche Verzeichnisse: `/home`

Dieser Ordner enthält die persönlichen Verzeichnisse aller auf diesem Rechner angelegten Benutzerkonten. Die Größe variiert natürlich, abhängig von der Anzahl der Benutzer (und Dienste) und deren Bedürfnissen.

Es ist natürlich auch möglich, **keine** separate Partition für `/usr` einzurichten: dann ist `/usr` nur ein weiteres Unterverzeichnis innerhalb der Wurzel (`/`). In dem Fall muss deren Größe natürlich entsprechend angepasst werden.

Schließlich können Sie sich auch auf die Einrichtung von swap und der Verzeichnisbaumwurzel (`/`) beschränken wenn Sie sich noch nicht über die weitere Nutzung Ihres Rechners klar sind. In diesem Fall liegen dann sowohl Ihr `/home`-Verzeichnis als auch die Verzeichnisse `/usr` und `/var` in ihrer Wurzel-Partition.

2.1.3.3 „Exotische“ Konfigurationen

Bei der Einrichtung Ihres Rechners für einen bestimmten Zweck – etwa als Webserver oder Firewall – sind die Voraussetzungen natürlich andere als für einen Standard-Arbeitsplatzrechner. So wird etwa ein FTP-Server normalerweise eine große separate Partition für `/var/ftp` benötigen während das Verzeichnis `/usr` verhältnisweise klein sein wird. Sie sollten also in jedem Fall sorgfältig über die voraussichtliche Nutzung und den entsprechenden Platzbedarf nachdenken bevor Sie den Installationsprozess starten.

Falls Sie Ihre Partitionen später verändern müssen oder insgesamt ein anderes Partitionsschema benötigen, können Sie die Größe der meisten Partitionen ohne Neuinstallation und ohne Datenverlust verändern. Bitte lesen Sie dazu das Kapitel *Ihre Partitionen verwalten* des *Starter Handbuch* .

Mit etwas Übung werden Sie auch in der Lage sein, eine belegte Partition auf eine neue Festplatte zu übertragen.

2.2 Namenskonventionen für Festplatten und Partitionen

GNU/Linux benutzt zur Benennung von Partitionen ein einfaches logisches Schema. Erstens ignoriert es bei der Benennung die Dateisysteme der Partitionen und zweitens benennt es die Partitionen nach der jeweiligen Festplatte, auf der sie sich befinden. Die Festplatten werden wie folgt bezeichnet:

- Die Master- und Slave-Geräte des ersten IDE-Ports (egal, ob Festplatten, CD-ROMs oder irgendetwas Anderes) werden `/dev/hda` bzw. `/dev/hdb` genannt;
- die Geräte des zweiten IDE-Ports nennen sich entsprechend `/dev/hdc` (Master) und `/dev/hdd` (Slave);
- Falls Ihr Rechner weitere IDE-Anschlüsse besitzt (beispielsweise einen IDE-Anschluss auf einer Soundblaster Karte), werden die daran angeschlossenen Geräte mit `/dev/hde`, `/dev/hdf`, usw. benannt. Das gilt natürlich auch für zusätzliche IDE RAID-Controller.
- SCSI-Platten werden mit `/dev/sda`, `/dev/sdb`, usw. in der Reihenfolge ihrer Anordnung auf dem SCSI-Bus benannt (entsprechend der ID). Die SCSI CD-ROM-Laufwerke nennen sich `/dev/scd0`, `/dev/scd1`, ebenfalls in der Reihenfolge auf dem SCSI-Bus.

Bei SATA IDE Platten wird das SCSI-Benennungsschema angewendet.

Die einzelnen Partitionen werden nach der Festplatte benannt, auf der sie sich befinden (in unserem Beispiel gehen wir von Partitionen aus, die sich auf einem primären Master und Slave befinden):

- Primäre oder erweiterte Partitionen tragen die Bezeichnung `/dev/hda1` bis `/dev/hda4` (falls vorhanden);

- Logische Partitionen, falls vorhanden, werden mit `/dev/hda5`, `/dev/hda6`, usw. benannt, entsprechend ihrer Reihenfolge in der logischen Partitionstabelle.

Also benennt GNU/Linux die Partitionen in folgender Weise:

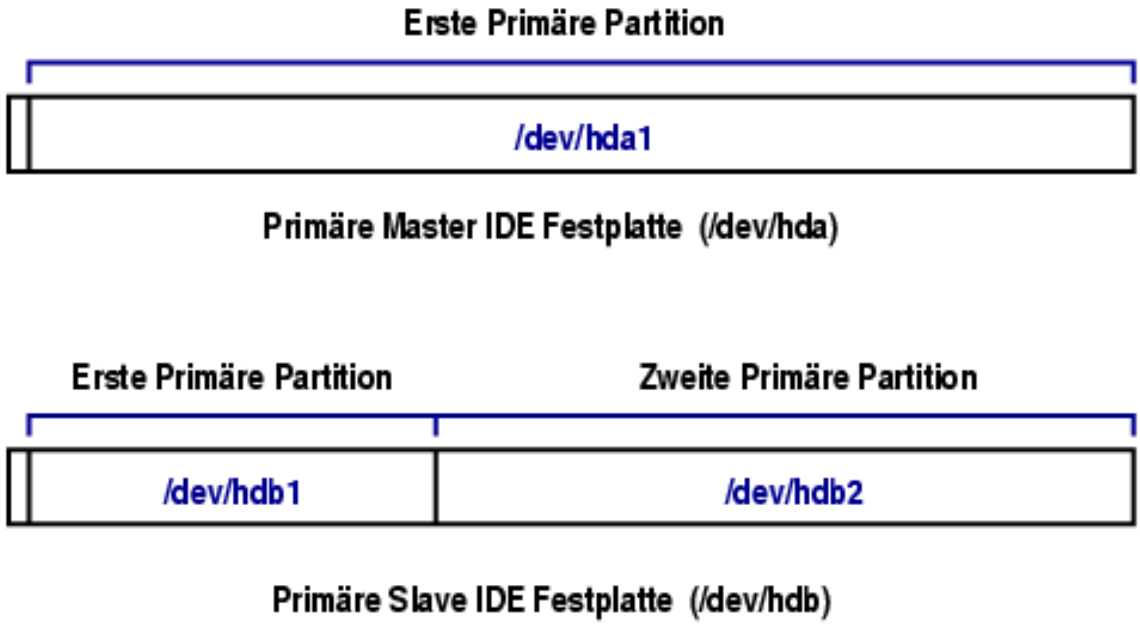


Abbildung 2-1: Erstes Beispiel der Partitionsbezeichnung unter GNU/Linux

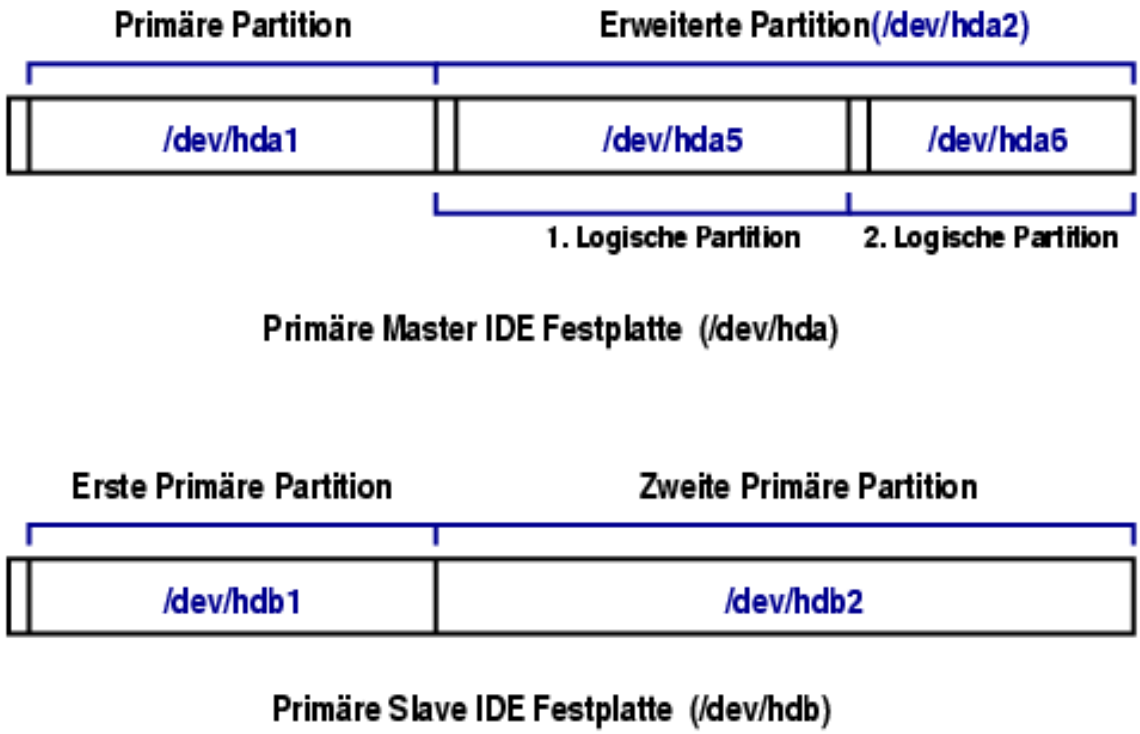


Abbildung 2-2: Zweites Beispiel der Partitionsbezeichnung unter GNU/Linux

Mit diesem Wissen ausgerüstet können Sie jetzt bei Bedarf Ihre verschiedenen Partitionen und Laufwerke ansprechen. Sie werden auch feststellen, dass GNU/Linux sogar Partitionen in dieser Weise benennt, die es nicht bearbeiten kann (es ignoriert bei der Benennung die Tatsache, dass es sich nicht um native GNU/Linux-Partitionen handelt).

Mandrakelinux benutzt seit der neuesten Version jetzt udev (Informationen darüber bekommen Sie in der udev FAQ (`http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev-FAQ`)).
Es ist absolut kompatibel mit dem oben beschriebenen Schema sowie den Standards des Linux Standards Base Project (`http://www.linuxbase.org/`). Bei diesem System wird jedes Gerät dynamisch eingebunden sobald es zur Verfügung steht oder benötigt wird.

Kapitel 3. Einführung in die Kommandozeile

Im Kapitel *Grundlegende UNIX Konzepte*, Seite 7 lernten Sie, wie man eine shell öffnet. In diesem Kapitel werden Sie lernen, wie man damit arbeitet.

Der wichtigste Vorteil der shell ist die Anzahl der verfügbaren Hilfsprogramme: es gibt tausende davon und jedes einzelne erledigt eine bestimmte Aufgabe. Wir sehen uns hier nur eine kleine Auswahl davon an. Weiterhin werden Sie einen der großen Vorteile von UNIX® kennenlernen: man kann diese kleinen Helfer kombinieren.

3.1 Datei-Werkzeuge

In diesem Kontext bezeichnet der Ausdruck „Dateiverwaltung“ das Kopieren, Verschieben und Löschen von Dateien. Später werden wir auch das Ändern von Dateiattributen (Besitzer, Rechte) behandeln.

3.1.1 mkdir, touch: Erzeugen von leeren Verzeichnissen und Dateien

`mkdir` (engl. *MaKe DIRectory*) Wird zum Erzeugen von Verzeichnissen benutzt. Die Syntax ist relativ einfach:

```
mkdir [optionen] <verzeichnis> [verzeichnis ...]
```

Als einzige Option sollte man sich die Option `-p` anschauen, die zwei Dinge bewirkt:

1. erstens erzeugt sie Eltern-Verzeichnisse, falls diese nicht schon existieren. Ohne die Option würde `mkdir` mit der Fehlermeldung abbrechen, dass die besagten Eltern-Verzeichnisse nicht existieren;
2. zweitens bewirkt die Option, dass `mkdir` keine Fehlermeldung ausgibt, falls das anzulegende Verzeichnis bereits existiert. Ohne Option würde `mkdir` eine entsprechende Fehlermeldung ausgeben.

Einige Beispiele:

- `mkdir foo`: erzeugt ein Verzeichnis `foo` im aktuellen Verzeichnis;
- `mkdir -p images/misc docs`: erzeugt das Verzeichnis `misc` im Verzeichnis `images`. Zuerst wird das Elternverzeichnis erzeugt, falls es nicht existiert (`-p`); anschließend wird das Verzeichnis `docs` im aktuellen Verzeichnis angelegt.

Ursprünglich war der Befehl `touch` nicht zur Erzeugung von Dateien sondern zur Aktualisierung der Zugriffs- und Änderungszeiten[1] gedacht. Als Nebeneffekt erzeugt `touch` auch leere Dateien, falls eine Datei dieses Namens nicht bereits existiert. Die Syntax ist:

```
touch [optionen] datei [datei...]
```

Also erzeugt der Befehl:

```
touch datei1 images/datei2
```

eine leere Datei mit Namen `datei1` im aktuellen Verzeichnis und eine leere Datei `datei2` im Verzeichnis `images`, falls die Dateien noch nicht existieren.

3.1.2 rm: Dateien oder Verzeichnisse löschen

Der Befehl `rm` (engl. *ReMove*) ersetzt die DOS-Befehle `del` und `deltree`, enthält allerdings weitere Optionen. Es gilt die folgende Syntax:

```
rm [optionen] <datei|verzeichnis> [datei|verzeichnis...]
```

Die Optionen sind:

1. Unter UNIX® gibt es drei verschiedene Zeitstempel für jede Datei: für den letzten Zugriff (`atime`), d.h., als die Datei zuletzt zum Lesen oder Schreiben geöffnet wurde; die letzte Veränderung der Inode-Attribute (`mtime`); und schließlich die letzte Änderung des **Inhalts** der Datei (`ctime`).

- `-r`, oder `-R`: löscht rekursiv. Diese Option **muss** bei Verzeichnissen (egal ob leer oder nicht) angewendet werden. Allerdings können Sie zum Löschen von leeren Verzeichnissen auch den Befehl `rmdir` verwenden.
- `-i`: verlangt vor jedem Löschvorgang eine Bestätigung. Beachten Sie bitte, dass unter Mandrakelinux `rm` aus Sicherheitsgründen eigentlich ein ***Kürzel*** für `rm -i` ist (Gleiches gilt für `cp` und `mv`). Falls Ihnen das nicht zusagt können Sie eine leere Datei `~/.alias` anlegen und so alle systemweiten Kürzel ausschalten. Alternativ können Sie in Ihrer `~/.bashrc` durch die Eingabe von: `unalias rm cp mv` gezielt diese Kürzel deaktivieren.
- `-f`, das Gegenteil der Option `-i`, erzwingt die Löschung der Dateien/Verzeichnisse, selbst wenn der Benutzer keine Schreibberechtigung für die Dateien hat[2].

Einige Beispiele:

- `rm -i images/*.jpg datei1`: löscht alle Dateien mit der Endung `.jpg` im Verzeichnis `images`, löscht die Datei `datei1` im aktuellen Verzeichnis und fragt bei jeder Datei nach, ob Sie sie wirklich löschen wollen. Geben Sie zur Bestätigung **y**, zur Verneinung **n** ein.
- `rm -Rf images/misc/ file*`: löscht ohne Bestätigungsabfrage das gesamte Verzeichnis `misc/` im Verzeichnis `images/`, dazu alle Dateien des aktuellen Verzeichnisses, die mit `file` beginnen.

`rm` löscht Dateien **unwiederbringlich**. Es gibt keine Möglichkeit, die Dateien wiederherzustellen (es gibt dazu zwar ein paar Möglichkeiten, die aber alles andere als trivial sind). Verwenden Sie ruhig die Option `-i` als zusätzliche Sicherheit, so dass Sie nichts irrtümlich löschen.

3.1.3 mv: Verschieben oder Umbenennen von Dateien

Die Syntax des Befehls `mv` (engl. *MoVe*) ist wie folgt:

```
mv [options] <datei|verzeichnis> [datei|verzeichnis ...] <ziel>
```

Einige Optionen:

- `-f`: erzwingt die Operation – ohne Warnung, falls eine existierende Datei dabei überschrieben wird.
- `-i`: das Gegenteil: fragt vor Überschreiben einer existierenden Datei.
- `-v`: ***verbose*** Modus, gibt alle Änderungen und Operationen aus.

Einige Beispiele:

- `mv -i /tmp/pics/*.png .`: verschiebt alle Dateien im Verzeichnis `/tmp/pics/`, deren Name auf `.png` endet, in das aktuelle Verzeichnis (`.`) und fragt nach Bestätigung, falls dabei eine existierende Datei überschrieben werden sollte.
- `mv foo bar`: benennt die Datei `foo` um in `bar`. Falls ein Verzeichnis `bar` existiert, wird die Datei `foo` oder das gesamte Verzeichnis des gleichen Namens inklusive aller darin enthaltenen Dateien und Unterverzeichnissen in das Verzeichnis `bar` verschoben.
- `mv -vf file* images/ trash/`: verschiebt ohne Nachfrage alle Dateien des aktuellen Verzeichnisses mit dem Anfang `file` zusammen mit dem gesamten Verzeichnis `images/` in das Verzeichnis `trash/` und listet alle Einzeloperationen auf.

2. Es reicht die Schreibberechtigung für das **Verzeichnis** um darin enthaltene Dateien zu löschen.

3.1.4 cp: Kopieren von Dateien und Verzeichnissen

cp (engl. *CoPy*) ersetzt die DOS-Befehle copy und xcopy, allerdings mit ein paar zusätzlichen Optionen. Die Syntax ist wie folgt:

```
cp [options] <datei|verzeichnis> [datei|verzeichnis ...] <ziel>
```

cp hat eine Vielzahl an Optionen. Es folgen die meist benutzten:

- -R: rekursives kopieren; **muss** bei Verzeichnissen angewendet werden (auch bei leeren Verzeichnissen).
- -i: fragt nach Bestätigung bevor eine Datei überschrieben wird.
- -f: das Gegenteil von -i, überschreibt existierende Dateien ohne Nachfrage.
- -v: verboser Modus, listet alle Einzeloperationen von cp.

Einige Beispiele:

- cp -i /timages/* images/: kopiert alle Dateien des Verzeichnisses /timages/ in das Unterverzeichnis images/ des aktuellen Verzeichnisses. Vor Überschreiben einer Datei wird nachgefragt.
- cp -vR docs/ /shared/mp3s/* meinkram/: kopiert das gesamte Verzeichnis docs plus alle Dateien des Verzeichnisses /shared/mp3s in das Verzeichnis meinkram.
- cp foo bar: fertigt eine Kopie der Datei foo unter dem Namen bar im aktuellen Verzeichnis an.

3.2 Dateiattribute

Die folgenden Befehle dienen dazu, den Eigentümer oder die Benutzergruppe einer Datei oder deren Rechte zu ändern. Die verschiedenen Rechte werden im Kapitel Grundlegende UNIX Konzepte behandelt.

3.2.1 chown, chgrp: Ändern der Besitzrechte einer oder mehrerer Dateien

Die Syntax des Befehls chown (engl. *CHange OWNer*) ist wie folgt:

```
chown [options] <benutzer[:gruppe]> <datei|verzeichnis> [datei|verzeichnis...]
```

Die Optionen:

- -R: rekursiv. Zum Ändern der Besitzer aller Dateien und Unterverzeichnisse in einem Verzeichnis.
- -v: verboser Modus. Listet alle Aktionen von chown; listet, welche Dateien infolge des Befehls den Besitzer gewechselt haben und welche unverändert geblieben sind.
- -c: wie -v, listet aber nur die veränderten Dateien auf.

Einige Beispiele:

- chown nobody /shared/book.tex: ändert den Besitzer der Datei /shared/book.tex in nobody.
- chown -Rc franz:music *.mid concerts/: setzt das Besitzrecht aller Dateien im aktuellen Verzeichnis mit der Endung .mid und aller Dateien und Unterverzeichnisse des Verzeichnisses concerts/ auf den Benutzer franz und die Gruppe music. Dabei werden nur die veränderten Dateien aufgelistet.

Der Befehl chgrp (engl. *CHange GRouP*) bewirkt die Änderung des Gruppen-besitzrechts von Dateien; die Syntax ist ähnlich der von chown:

```
chgrp [options] <gruppe> <datei|verzeichnis> [datei|verzeichnis...]
```

Die Optionen für diesen Befehl sind die gleichen wie bei chown und der Befehl wird in gleicher Weise angewendet. So bewirkt z.B. der Befehl

```
chgrp disk /dev/hd*
```

die Änderung des Besitzrechts aller Dateien im Verzeichnis `/dev/`, deren Namen mit `hd` beginnt, auf die Gruppe `disk`.

3.2.2 chmod: Änderung von Zugriffsrechten bei Dateien und Verzeichnissen

Der Befehl `chmod` (engl. *CHange MODe*) hat eine recht unterschiedliche Syntax. Generell lautet sie:

```
chmod [options] <Änderungsmodus> <datei|verzeichnis> [datei|verzeichnis...]
```

Die Unterschiede liegen dabei in den verschiedenen Änderungsmodi begründet. Diese können auf zwei verschiedene Arten angegeben werden:

1. in Octalzahlen. Dabei haben die Rechte des Besitzers die Form `<x>00`, wobei `<x>` für das jeweilige Recht steht: 4 für Leseberechtigung, 2 für Schreibberechtigung und 1 für das Ausführen der Datei oder des Verzeichnisses. Gleiches gilt für die Rechte der Besitzergruppe mit der Form `<x>0` und die Rechte für den Rest der Welt in der Form `<x>`. Nun müssen Sie nur die zu vergebenden Rechte zusammen addieren. So ergibt z.B. die Rechtefolge `rwxr-xr--` auseinander geschrieben 400+200+100 (für den Besitzer, `rwx`) +40+10 (für dei Gruppe, `r-x`) +4 (für alle Anderen, `r--`) = 754. In dieser Art werden die Rechte absolut ausgedrückt, d.h., vorherige Berechtigungen werden bedingungslos ersetzt;
2. in Ausdrücken. Dabei werden die Ausdrücke in Folge geschrieben und durch Kommata getrennt. Das sieht dann so aus: `[kategorie]<+|-|=><berechtigungen>`.

   Die Kategorie ist eine oder mehrere der Folgenden:

   - `u` (*User*, Berechtigungen des Besitzers);
   - `g` (*Group*, Berechtigungen der besitzenden Gruppe);
   - `o` (*Others*, Berechtigungen für „Andere“).

   Wenn keine Kategorie angegeben wird gelten die Änderungen für alle Kategorien. Das Zeichen `+` setzt die Berechtigung, das `-` hebt sie auf. Die Berechtigung ist eine (oder mehrere) der Folgenden:

   - `r` (*Read*): Lesen;
   - `w` (*Write*): Schreiben;
   - `x` (*eXecute*): Ausführen.

Die wichtigsten Optionen sind denen von `chown` and `chgrp` recht ähnlich:

- `-R`: ändert die Rechte rekursiv.
- `-v`: verboser Modus. Listet die Ergebnisse des Befehls für jede Datei.
- `-c`: wie `-v`, listet aber nur alle vom Befehl betroffenen Dateien.

Beispiele:

- `chmod -R o-w /shared/docs`: deaktiviert rekursiv die Schreibberechtigung für `others` bei allen Dateien im Verzeichnis `/shared/docs/`.
- `chmod -R og-w,o-x private/`: deaktiviert rekursiv die Schreibberechtigung für `group` und `others` im gesamten Verzeichnis `private/` directory und deaktiviert die Ausführrechte für `others`.
- `chmod -c 644 misc/file*`: ändert die Berechtigungen aller Dateien im Verzeichnis `misc/` die mit `file` beginnen auf `rw-r--r--` (d.h., Leseberechtigung für alle uns Schreibberechtigung nur für den Besitzer). Die durch den Befehl betroffenen Dateien werden aufgelistet.

3.3 Platzhalter in der Shell

Sie benutzen vermutlich längst das ***Globbing*** (Ersetzen durch Platzhalter) von Zeichen ohne es zu wissen. Wenn Sie eine Datei unter Windows® angeben oder eine Datei suchen, benutzen Sie meist * als Platzhalter für eine beliebige Zeichenfolge. Beispielsweise steht `*.txt` für alle Dateien, deren Name mit `.txt` endet. Auch wir haben es in diesem Kapitel bereits oft eingesetzt. Aber ausser * gibt es noch Einiges mehr im Bereich Globbing.

Wenn Sie einen Befehl wie etwa `ls *.txt` eingeben und dann auf `Enter` drücken, so wird die Suche nach Dateien, die dem Muster `*.txt` entsprechen, nicht vom Befehl `ls` sondern von der shell selbst durchgeführt. Zum Verständnis folgt eine kurze Erklärung des Vorganges, wie eine Befehlszeile von der shell ausgewertet und abgearbeitet wird. Bei der Eingabe von:

```
$ ls *.txt
      readme.txt  recipes.txt
```

wird die Befehlszeile erstmal in Wörter aufgeteilt (in diesem Beispiel in `ls` und `*.txt`). Wenn die Shell ein * in einem Wort findet wird das ganze Wort als Globbing-Muster angesehen und durch die Namen aller passenden Dateien ersetzt. Also wurde der Befehl, kurz bevor er von der Shell ausgeführt wurde, zum ausführlichen Befehl `ls readme.txt recipe.txt`, was der Ausgabe entspricht. Auch weitere Zeichen veranlassen die Shell dazu, in gleicher Weise zu reagieren:

- `?`: ersetzt ein einzelnes Zeichen, egal welcher Art;
- `[...]`: passt auf jedes Zeichen innerhalb der Klammern. Man kann einen Bereich von zeichen angeben (etwa `1-9`) oder ***bestimmte Werte*** oder sogar beides. Beispiel: `[a-zBE5-7]` passt auf alle zeichen zwischen `a` und `z`, auf `B`, auf `E`, auf `5`, auf `6` oder auf `7`;
- `[!...]`: passt auf alle Zeichen, die **nicht** in den Klammern angegeben sind. `[!a-z]`, passt zum Beispiel auf alle Zeichen, die keine Kleinbuchstaben sind[3];
- `{c1,c2}`: passt auf `c1` oder `c2`, wobei `c1` und `c2` ebenfalls Platzhalter sind. Sie könnten zum Beispiel auch schreiben: `{[0-9]*,[acr]}`.

Einige Beispiele:

- `/etc/*conf`: alle Dateien im Verzeichnis `/etc` deren Name auf `conf` endet. Das passt auf `/etc/inetd.conf`, `/etc/conf.linuxconf`, **aber auch auf** `/etc/conf` falls eine solche Datei existiert. Beachten Sie bitte, dass * auch für eine leere Zeichenfolge stehen kann.
- `image/{cars,space[0-9]}/*.jpg`: alle Dateien mit der Endung `.jpg` in den Verzeichnissen `image/cars`, `image/space0`, (...), `image/space9`, falls diese Verzeichnisse existieren.
- `/usr/share/doc/*/README`: alle Dateien mit dem Namen `README` in allen direkten Unterverzeichnissen von `/usr/share/doc`. Das passt beispielsweise auf `/usr/share/doc/mandrake/README` aber nicht auf `/usr/share/doc/myprog/doc/README`.
- `*[!a-z]`: alle Dateien im aktuellen Verzeichnis, deren Namen **nicht** mit einem Kleinbuchstaben endet.

3.4 Umleitungen und Pipes

3.4.1 Noch etwas über Prozesse

Um das Prinzip der Umleitungen und Pipes richtig zu verstehen müssen wir vorher einen Aspekt der Prozesse behandeln, um den wir uns bisher nicht gekümmert haben. Die meisten UNIX®-Prozesse (inklusive grafische Anwendungen aber exklusive der meisten Demons) benutzen mindestens drei Datei-Werte: Standard-Eingabe, Standard-Ausgabe und Standard-Fehler. Die entsprechenden Nummern lauten 0, 1 und 2. Im Allgemeinen gehören diese drei Werte zu dem jeweiligen Terminal, in dem der Prozess gestartet wurde, wobei die Eingabe durch die Tastatur vorgenommen wird. Das Ziel der Umleitungen und Pipes ist die Umleitung dieser Rückgabe-Werte. Die Beispiele in diesem Abschnitt sollen dieses Prinzip veranschaulichen.

3. Vorsicht! Das trifft auf die meisten Sprachen zu, muss aber nicht bei Ihrer Sprache zutreffen. Das hängt von der **Zeichenordnung** ab. In manchen Spracheinstellungen trifft `[a-z]` auf a, A, b, B, (...), und z zu. Und dabei reden wir noch nicht einmal von Umlauten ...

3.4.2 Umleitungen

Nehmen wir an, Sie wollen eine Liste der Dateien mit der Endung `.png` [4] im Verzeichnis `images` aufstellen. Da diese Liste sehr umfangreich ist wollen Sie sie in einer Datei speichern um sich die Aufstellung später in Ruhe anzusehen. Dazu geben Sie den folgenden Befehl ein:

```
$ ls images/*.png 1>file_list
```

Die Standard-Ausgabe dieses Befehls (`1`) wird also umgeleitet (`>`) in eine Datei `file_list`. Der Operator `>` ist dabei der Ausgabe-Umleiter. Falls die Umleitungsdatei nicht existiert wird sie angelegt, fals sie bereits existiert, wird ihr Inhalt durch die Ausgabe des Befehls überschrieben. Der Standard-Wert (`1`) ist jedoch vorgegeben und muss nicht gesondert spezifiziert werden. Also können Sie in diesem Fall vereifacht schreiben:

```
$ ls images/*.png >file_list
```

und das Resultat wird das Gleiche sein. Anschließend können Sie sich die Datei mit einem Datei-Anzeiger wie z.B. `less` anschauen.

Wenn Sie nun wissen wollen, wieviele solcher Dateien (in diesem Beispiel) existieren, müssen Sie nicht mühsam anfangen zu zahlen, dazu gibt es den Befehl `wc` (engl. *Word Count*) mit der Option `-l`, die Ihnen die Anzahl der Zeilen einer Datei in der Standard-Ausgabe angibt. Eine mögliche Verwendung wäre:

```
wc -l 0<file_list
```

und Sie erhalten die gewünschte Anzeige. Der Operand `<` ist der Umleiter der Eingabe und der standardmäßige Descriptor dazu ist `0` (Standard-Eingabe). Also lautet der vereinfachte Befehl:

```
wc -l <file_list
```

Jetzt wollen Sie alle Dateinamen-„Erweiterungen“ entfernen und das Ergebnis in einer anderen Datei speichern. Das hierfür ideale Werkzeug ist `sed` (*Stream EDitor*). Sie leiten also die Standard-Eingabe des Befehls `sed` auf die Datei `file_list` und die Ausgabe in die Zieldatei, z.B. `the_list`:

```
sed -e 's/\.png$//g' <file_list >the_list
```

Nun wurde die Liste erstellt und Sie können Sie in Ruhe mit dem Betrachter Ihrer Wahl ansehen.

Sehr hilfreich kann die Umleitung von Standard-Fehlermeldungen sein. Sie wollen z.B. wissen, auf welche Unterverzeichnisse Sie im Verzeichnis `/shared` nicht zugreifen können? Eine der möglichen Lösungen ist das rekursive Auflisten dieses Verzeichnisses wobeiSie die Fehlermeldungen in eine Datei umleiten und die Standard-Ausgabe nicht anzeigen:

```
ls -R /shared >/dev/null 2>errors
```

Dies bedeutet im Einzelnen, dass die Standard-Ausgabe umgeleitet wird (`>`) in die Datei `/dev/null`, eine spezielle Datei, deren Inhalt sofort entsorgt wird (d.h., die Standard-Ausgabe wird nicht angezeigt), und die Standard-Fehler (`2`) werden umgeleitet (`>`) nach `errors`.

3.4.3 Pipes

Pipes sind eine Art Kombination von Eingabe- und Ausgabe-Umleitung. Das Prinzip gleicht dem einer physikalischen Röhre, daher der Name: ein Prozess sendet Daten in das eine Ende der Röhre und ein anderer Prozess liest diese Daten am anderen Ende aus. Das Zeichen für eine Pipe ist `|`. Gehen wir zu unserem Beispiel von eben zurück und nehmen wir an, Sie wollen sofort wissen, wieviele Dateien mit der Endung `.png` vorkommen, ohne den Umweg über eine temporäre Datei zu nehmen. Dazu benutzen Sie eine Pipe mit folgendem Befehl:

```
ls images/*.png | wc -l
```

Dabei wird die Standard-Ausgabe des Befehls `ls` (also die Dateiliste) auf die Standard-Eingabe des Befehls `wc` umgeleitet und ergibt sofort das gewünschte Resultat.

4. Sie nehmen jetzt vermutlich an, dass man anstatt „Dateien mit der Endung `.png`“ doch einfacher „PNG-Bilder“ sagen könnte. Jedoch sollten Sie sich nochmals merken, dass Dateien unter UNIX® die üblichen Endungen nur aus Bequemlichkeit haben, die Endungen definieren **nicht den Dateityp**. Eine Datei mit der Endung `.png` kann also durchaus eine JPEG-Grafik, eine Programmdatei, eine Textdatei oder jede beliebige andere Dateiart sein. Das gilt auch für Windows®!

Auch die Zusammenstellung der Dateien „ohne Namenserweiterungen“ funktioniert auf diese Weise:

```
ls images/*.png | sed -e 's/\.png$//g' >the_list
```

oder, falls Sie die Liste ansehen wollen ohne sie in einer Datei zu speichern:

```
ls images/*.png | sed -e 's/\.png$//g' | less
```

Pipes und Umleitungen sind aber in ihrer Funktion nicht auf lesbaren Text beschränkt. Der folgende Befehl, eingegeben in einem Terminal:

```
xwd -root | convert - ~/my_desktop.png
```

wird beispielsweise einen Screenshot Ihres Desktops in die Datei `my_desktop.png` [5] in Ihrem persönlichen Verzeichnis abspeichern.

3.5 Automatische Komplettierung in der Kommandozeile

Die automatische ***Komplettierung*** ist eine sehr nützliche Funktion, die alle modernen shells (auch die bash) bieten. Ihre einzige Aufgabe ist, die Arbeit für den Anwender zu vereinfachen. Am besten schauen Sie sich das in einem Beispiel an.

3.5.1 Beispiel

Stellen Sie sich vor, in Ihrem persönlichen Verzeichnis gibt es diese Datei `datei-mit-einem-langen-namen` und Sie wollen sich diese Datei ansehen. Ausserdem gibt es noch die Datei `datei_text` im gleichen Verzeichnis. Sie wechseln also in Ihr persönliches Verzeichnis und tippen folgenden Befehl:

```
$ less da<TAB>
```

(d.h., Sie tippen `less da` und drücken dann die Taste `TAB`). Die shell wird nun die Befehlszeile für Sie erweitern:

```
$ less datei_
```

und Ihnen eine Liste von möglichen Erweiterungen anzeigen (das ist der Standard, der aber angepasst werden kann). Nun geben Sie die folgende Tastenfolge ein:

```
less datei_m<TAB>
```

und die shell wird die Befehlszeile komplettieren:

```
less datei-mit-einem-langen-namen
```

Nun drücken Sie nur noch auf `Enter` und Sie bekommen die Datei angezeigt.

3.5.2 Weitere Komplettierungs-Methoden

Die Taste TAB ist nicht die einzige Möglichkeit, die Komplettierung zu aktivieren, allerdings die meist genutzte. Im Allgemeinen wird das erste zu komplettierende Wort in einer Kommandozeile ein Befehl sein (`nsl<TAB>` wird zu `nslookup` erweitert) und alle weiteren Wörter normalerweise Dateinamen, falls das Wort nicht mit einem „magischen“ Zeichen beginnt (`~`, `@` oder `$`). In diesen Fällen wird die shell versuchen, einen Usernamen, einen Maschinennamen oder eine Systemvariable mit entsprechendem Anfangszeichen zu finden[6]. Des weiteren gibt es das Anfangszeichen (`/`) für eine Datei und einen Befehl aus dem Befehlsspeicher (`!`).

Weitere zwei Möglichkeiten zur Aktivierung der Komplettierung sind die Sequenzen `Esc-<x>` und `Strg+x<x>`, wobei `<x>` eines der vorher erwähnten magischen Zeichen ist. `Esc-<x>` wird versuchen, eine eindeutige Komplettierung vorzunehmen. Falls das nicht möglich ist, wird die längste Zeichenfolge aus den möglichen Resultaten angezeigt. Ein ***Beep*** bedeutet entweder, dass die Auswahl nicht eindeutig ist oder dass es keine passende

5. Ja, ein richtiges PNG-Bild (allerdings muss dazu das Programm ImageMagick installiert sein...).
6. Denken Sie daran, dass UNIX® zwischen Groß- und Kleinschreibung unterscheidet. Die Variablen `HOME` und `home` sind nicht identisch.

Auswahl gibt. Die Sequenz Strg+x <x> zeigt nur die möglichen Resultate an ohne eine Komplettierung vorzunehmen. Ein Druck auf TAB bedeutet das Gleiche wie die Eingabe von Esc-<x> und Strg+x<x> hintereinander, wobei das magische Zeichen aus dem Kontext hervorgeht.

Also ist eine Möglichkeit, alle Umgebungsvariablen aufzulisten, die Sequenz Strg+x $ in einer leeren Zeile einzugeben. Ein weiteres Beispiel: wenn Sie sich die „man page“ für den Befehl nslookup ansehen wollen, geben Sie einfach die Folge man nsl ein und anschließend Esc-!. Die Shell wird diese Sequenz automatisch auf man nslookup erweitern.

3.6 Starten und Verwalten von Hintergrund-Prozessen: Job Control

Sie haben mittlerweile sicher bemerkt, dass Sie nach der Eingabe eines Befehls im Terminal normalerweise warten müssen, bis der Befehl seine Arbeit vollendet hat, ehe Sie wieder Kontrolle über die shell bekommen. Das ist der Fall, wenn Sie, wie bisher, Ihre Befehle im ***Vordergrund*** laufen lassen. Es gibt jedoch Situationen, in denen das nicht wünschenswert ist.

Nehmen wir an, Sie wollen ein umfangreiches Verzeichnis rekursiv in ein anderes Verzeichnis kopieren. Sie wollen dabei keine Fehlermeldungen sehen, also leiten Sie die Fehler um nach /dev/null:

```
cp -R images/ /shared/ 2>/dev/null
```

Solch ein Befehl kann schon einmal einige Minuten für die Ausführung benötigen, Zeit, die Sie gerade nicht zur Verfügung haben. Nun haben Sie zwei Möglichkeiten: die erste ist ziemlich rigoros. Sie stoppen den Prozess und starten ihn aufs Neue, wenn Sie mehr Zeit dafür haben. Dafür genügt ein einfaches Strg+c: damit wird der Prozess beendet und Sie haben Ihre Eingabeaufforderung wieder zur Verfügung. Aber halt! Tun Sie das nicht sondern lesen Sie erstmal weiter.

Warum lassen Sie nicht einfach den Prozess weiterlaufen während Sie etwas Anderes machen?. Die Lösung: Sie stellen den Prozess in den ***Hintergrund***. Geben Sie dazu Strg+z ein, um den Prozess auszusetzen:

```
$ cp -R images/ /shared/ 2>/dev/null
# Jetzt geben Sie Strg-z ein
[1]+  Stopped        cp -R images/ /shared/ 2>/dev/null
$
```

und Sie befinden sich wieder am Prompt. Der Prozess pausiert und wartet darauf, wieder gestartet zu werden (daher das Wort Stopped in der Ausgabe). Das wollen Sie ja auch, allerdings im Hintergrund. Geben Sie bg ein (für *BackGround*) und Sie erhalten das gewünschte Resultat:

```
$ bg
      [1]+ cp -R images/ /shared/ 2>/dev/null &
      $
```

Der Prozess läuft als Hintergrund-Job weiter, wie das & (Et-Zeichen, engl.*Ampersand*) am Ende der Zeile anzeigt. Sie können nun mit anderen Befehlen an der Eingabeaufforderung weiterarbeiten. Ein Prozess, der im Hintergrund läuft, wird Hintergrundprozess (engl. *Background* ***Job***) genannt.

Natürlich können Sie Prozesse direkt als Hintergrundprozesse starten, indem Sie am Ende der Befehlssequenz das Et-Zeichen (**&**) anfügen. Für unser Beispiel sieht das so aus:

```
cp -R images/ /shared/ 2>/dev/null &
```

Durch die Eingabe von fg (engl. *ForeGround*) holen Sie den Prozess wieder in den Vordergrund. Um ihn dann wieder in den Hintergeund zu stellen, geben Sie wieder zuerst Strg+z und dann bg ein.

Sie können auf diese Weise mehrere Jobs hintereinander starten, jeder Befehl bekommt dann eine Jobnummer. Der shell-Befehl jobs listet alle gerade in dieser aktuellen shell laufenden Jobs auf. Dabei steht vor dem zuletzt in den Hintergrund gestellten Job ein +. Um einen bestimmten Job in den Vordergrund zu holen, geben Sie fg <n> ein, wobei <n> die Jobnummer ist, also beispielsweise fg 5.

Bedenken Sie, dass Sie natürlich auf diese Art auch ***Vollbild***-Programme wie z.B. less oder einen Texteditor wie Vi im Hintergrund starten und sie bei Bedarf in den Vordergrund holen können.

3.7 Ein Wort zum Schluß

Wie Sie sehen, ist die shell ein sehr weitläufiges Gebiet und der Umgang damit eine reine Übungssache. In diesem relativ langen Kapitel haben wir nur einen kleinen Teil der verfügbaren Befehle behandelt: Mandrakelinux umfasst tausende von Befehlen und Werkzeugen und selbst die erfahrendsten Anwender benutzen nur einen Teil davon.

Es gibt Programme für jeden Geschmack und für jede Aufgabe: Sie finden Werkzeuge zur Bearbeitung von Grafiken (wie etwa `convert`, den GIMP ***Batch*** Modus und all die ***Pixmap*** Werkzeuge), von Sound (Ogg Vorbis-Kodierer, Audio CD-Spieler), zum Brennen von CDs, e-mail-Clients, FTP-Clients und sogar Webbrowser (wie `lynx` oder `links`); nicht zu vergessen: all die Verwaltungs-Werkzeuge.

Selbst wenn grafische Programme mit den gleichen Funktionen existieren, meist sind es nur grafische Frontends für die genannten Kommandozeilen-Befehle. Ein weiterer Vorteil der Kommandozeilen-Befehle besteht in ihrer Option, im nicht-interakiven Modus zu laufen. Das bedeutet, dass Sie beispielsweise das Brennen einer CD starten und sich dann abmelden können. Der Brennvorgang wird ohne Sie weiterlaufen (lesen Sie dazu den Handbuchauszug von nohup(1) oder den von screen(1)).

Kapitel 4. Textbearbeitung: Emacs und VI

Wie bereits in der Einleitung dargestellt zählt die Textbearbeitung [1] zu den grundlegenden Tätigkeiten in einem UNIX®-System. Die beiden Editoren, die wir hier behandeln, mögen zu Anfang etwas schwierig zu erlernen sein, erweisen sich jedoch bei näherer Betrachtung als wirksame Werkzeuge. Wirksam besonders wegen der Vielzahl an vorhandenen Editiermodi für verschiedene Dateitypen (Perl, C++, XML, etc.).

4.1 Emacs

Emacs ist vermutlich der leistungsfähigste Texteditor überhaupt. Er kann fast alles und ist dazu noch unbegrenzt erweiterbar mit Hilfe der eingebauten auf lisp basierenden Programmiersprache. Mit Emacs können Sie im Web surfen, Ihre E-Mails und Newsbeiträge lesen, Kaffee kochen, usw. Sie werden in diesem Kapitel nicht all diese Dinge lernen, aber Sie bekommen einen guten Einstieg. Sie werden lernen, wie Sie Emacs starten, eine oder mehrere Dateien bearbeiten, diese wieder speichern und Emacs verlassen.

Wenn Sie nach der Lektüre dieses Kapitels mehr über Emacs wissen wollen, sollten Sie sich die Website Tutorial Introduction to GNU Emacs (`http://www.lib.uchicago.edu/keith/tcl-course/emacs-tutorial.html`) ansehen.

4.1.1 Kurzer Überblick

Emacs zu starten ist relativ simpel:

```
emacs [Datei(en)]
```

Emacs wird jede als Argument angegebene Datei in einem eigenen Puffer öffnen, wobei maximal 2 Puffer gleichzeitig sichtbar sind. Wenn Sie Emacs ohne die Angabe von Dateinamen starten finden Sie sich in einem Puffer mit Namen `*scratch*` wieder. Innerhalb einer grafischen Umgebung werden Sie auch ein Menü sehen. In diesem Kapitel widmen wir uns jedoch dem Betrieb von Emacs mit der Tastatur.

4.1.2 Der Anfang

Jetzt ist es an der Zeit, in die Praxis einzusteigen. In unserem Beispiel werden wir mit dem Öffnen von 2 Dateien anfangen: `datei1` und `datei2`. Falls diese Dateien nicht schon existieren werden sie erstellt, sobald Sie etwas in sie einfügen:

```
$ emacs datei1 datei2
```

Sie werden ein Bild wie das Folgende sehen:

1. „Text bearbeiten“ bedeutet das Ändern des Inhaltes von Dateien, die ausschließlich Buchstaben, Ziffern und Satzzeichen enthalten. Sie beinhalten keinerlei Layout-Informationen wie etwa Schriftarten. Dazu zählen unter Anderen Emails, Quellcode, Dokumente und sogar Konfigurationsdateien.

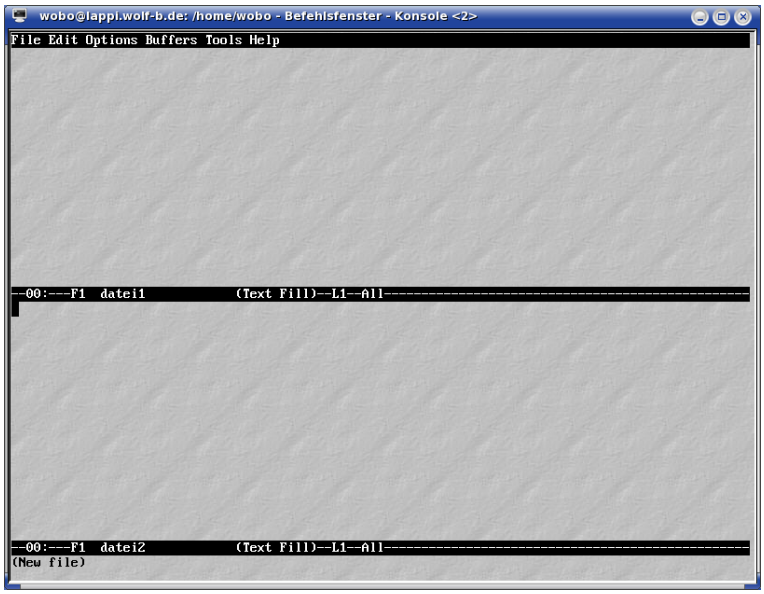


Abbildung 4-1: Emacs, Editieren von 2 Dateien

Wie Sie sehen können, wurden 2 Puffer geöffnet, je einer pro Datei. Einen dritten Puffer sehen Sie am unteren Rand des Bildschirms (da steht `(Neue Datei)`); das ist der sogenannte Mini-Puffer. Sie können auf diesen Puffer nicht ohne Weiteres zugreifen, sondern nur wenn Emacs Sie dazu einlädt. Um den Inhalt des aktuellen Puffers zu ändern geben Sie jetzt `Strg+x o` ein. Schreiben Sie wie in einem „normalen" Texteditor und löschen Sie Zeichen mit der Taste `Entf` oder `Backspace`.

Zum Bewegen innerhalb des Puffers benutzen Sie die Pfeiltasten oder eine der folgenden Tasten-Kombinationen: `Strg+a` führt sie zum Anfang und `Strg+e` zum Ende einer Zeile, `Alt+<` zum Anfang und `Alt+>` zum Ende des Puffers. Es gibt viele solcher Kombinationen, sogar welche für die Pfeiltasten [2].

Wenn Sie das Editieren beenden und Ihre geänderte Datei auf der Festplatte speichern wollen, geben Sie `Strg+x Strg+s` ein. Soll die Datei unter einem anderen Namen gesichert werden, so lautet die Kombination `Strg+x Strg+w`. Emacs fragt Sie in diesem Fall nach dem neuen Namen der Datei. Sie können dazu natürlich die automatische ***Komplettierung*** benutzen.

4.1.3 Arbeiten mit Puffern

Sie können bei Bedarf auch auf die Darstellung eines einzelnen Puffers umschalten. Dazu gibt es zwei Möglichkeiten:

- Wenn der Puffer, in dem Sie sich befinden, versteckt werden soll, tippen Sie `Strg+x 0`
- Wenn der Puffer, in dem Sie sich befinden, sichtbar bleiben soll, lautet die Kombination `Strg+x 1`.

Ebenso gibt es zwei Möglichkeiten, einen versteckten Puffer wieder sichtbar zu machen:

- Geben Sie `Strg+x b` ein und geben Sie den Namen des Puffers an oder
- schreiben Sie `Strg+x Strg+b`. Hiermit öffnen Sie einen neuen Puffer mit der sogenannten `*Buffer Liste*`. Sie können sich in diesem Puffer mit der Kombination `Strg+x o` umherbewegen und den gewünschten Puffer auswählen und die Taste `Enter` drücken oder den Namen des Puffers aus der Liste in den Mini-Puffer eingeben. Nachdem Sie Ihre Wahl getroffen haben, verschwindet der Puffer mit der `*Buffer Liste*` wieder im Hintergrund.

Nach dem Editieren einer Datei können Sie damit verbundene Puffer mit der Kombination `Strg+x k` wieder schließen. Emacs wird Sie fragen, welcher Puffer geschlossen werden soll. Die Vorgabe ist der Puffer, in dem Sie sich befinden. Falls Sie einen anderen Puffer schließen wollen, geben Sie dessen Namen direkt ein oder drücken die `TAB`-Taste: nun öffnet Emacs einen weiteren Puffer mit einer Liste von möglichen Namen. Wählen Sie einen aus und drücken Sie zur Bestätigung auf `Enter`.

2. Emacs wurde für den Betrieb auf vielen verschiedenen Computer-Modellen geschrieben, darunter auch Modelle, deren Tastaturen keine Pfeiltasten besitzen. Dieser Grundsatz gilt noch mehr für den Editor Vi.

Sie können zu jeder Zeit auch zwei Puffer gleichzeitig wieder zur Anzeige bringen, indem Sie `Strg+x 2` eingeben. Standardgemäß enthält der zweite Puffer eine Kopie der Datei, die der aktuelle Puffer enthält (was bei der Bearbeitung von großen Dateien an mehreren Stellen „zur gleichen Zeit“ sehr hilfreich ist). Zum Bewegen zwischen den Puffern benutzen Sie die vorher beschriebenen Befehle.

Sie können jederzeit eine weitere Datei mit dem Befehl `Strg+x Strg+f` öffnen. Emacs wird nach dem Namen fragen und Sie geben ihn ein, wobei Sie natürlich wieder die automatische Wortergänzung einsetzen können.

4.1.4 Kopieren, Ausschneiden, Einfügen, Suchen

Stellen Sie sich folgende Situation vor: Abbildung 4-2.

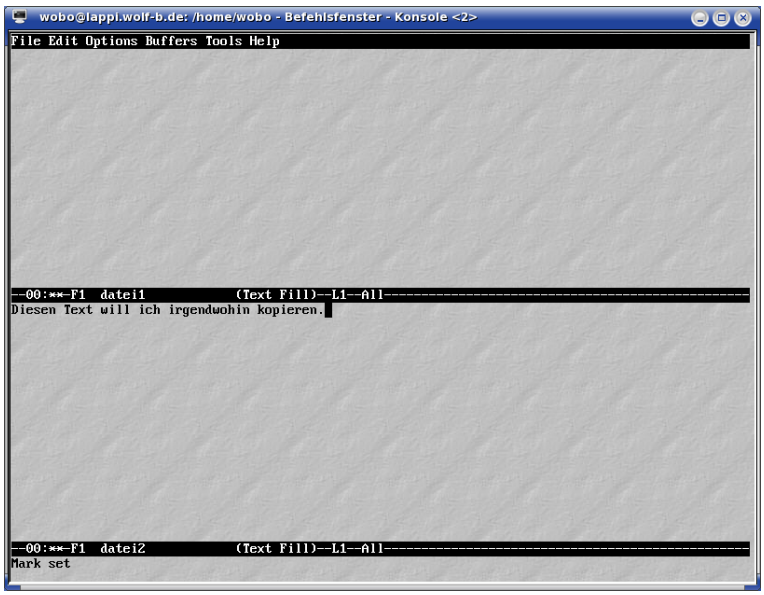


Abbildung 4-2: Emacs, vor dem Kopieren eines Textblocks

Zuerst müssen Sie den zu kopierenden Text markieren. In diesem Beispiel wollen wir den gesamten Satz kopieren. Zuerst setzen wir eine Marke an den Anfang des Bereiches. Nehmen wir an, der Cursor ist da, wo ihn die Abbildung weiter oben zeigt, dann wäre die Befehlsfolge: `Strg+Leer` (`Strg-Taste` + Leertaste). Emacs zeigt daraufhin die Meldung `Mark set` im Mini-Puffer an. Nun gehen wir mit dem Befehl `Strg+a` an den Anfang der Zeile. Der jetzt ausgewählte Bereich zum Kopieren oder Ausschneiden umfasst den gesamten Bereich zwischen der gesetzten Marke und der aktuellen Position des Cursors, also die ganze Zeile in unserem Beispiel. Jetzt stehen zwei Befehls-Sequenzen zur Verfügung–`Alt+w` (zum Kopieren) und `Strg+w` (zum Ausschneiden). Beim Kopieren geht Emacs kurz zum markierten Punkt zurück um Ihnen nochmal den in Frage kommenden Bereich anzuzeigen.

Schließlich müssen Sie zu dem Puffer gehen, in den Sie den Text einfügen wollen und geben Sie da das Kommando `Strg+y`. Damit sollte das Resultat so aussehen:

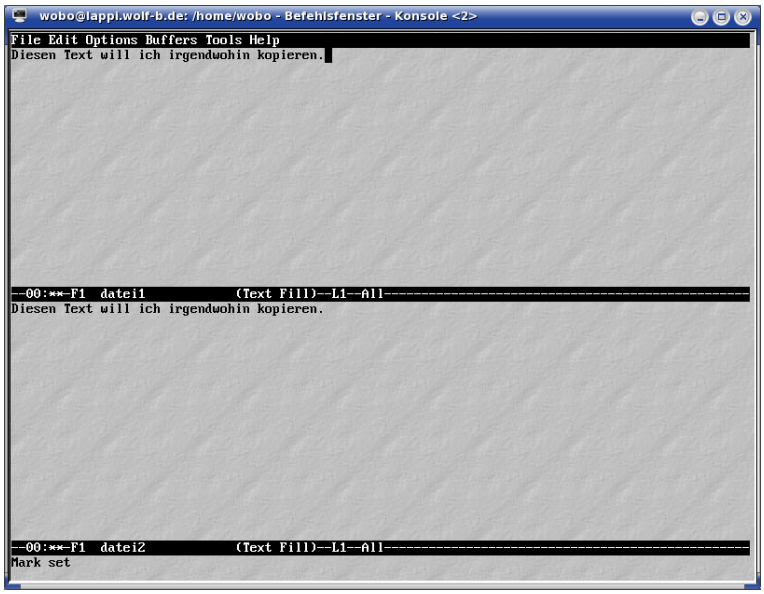


Abbildung 4-3: Text kopieren mit Emacs

Was Sie hier tatsächlich gemacht haben: Sie haben den Text in den sogenannten *„Zwischenspeicher“* des Emacs kopiert. Dieser Zwischenspeicher enthält alle kopierten oder ausgeschnittenen Textteile seit dem Start des Programms. **Jeglicher** kopierter oder ausgeschnittener Bereich landet an erster Position des Zwischenspeichers. Die Tastenfolge `Strg+y` fügt lediglich den Bereich ein, der an Position 1 steht. Um an die weiteren Bereiche zu gelangen, drücken Sie erst die Tastenfolge `Strg+y` und dann solange `Alt+y` bis Sie den gewünschten Textteil gefunden haben.

Die Suche nach bestimmten Textteilen starten Sie durch die Eingabe von `Strg+s`. Emacs fordert Sie dann auf, das Suchmuster einzugeben. Eine Fortsetzung einer Suche im gleichen Puffer nach dem gleichen Muster wie vorher starten Sie mit `Strg+s`. Wenn Emacs das Ende des Puffers erreicht hat können Sie mit `Strg+s` die Suche erneut am Anfang des Puffers beginnen lassen. Mit der Taste `Enter` beenden Sie die Suche.

Die Aktion „Suchen und Ersetzen“ starten Sie durch `Alt+%`. Emacs fragt Sie zuerst nach dem zu suchen den Text, dann nach dem Ersatztext und bittet Sie bei jeder Fundstelle um Bestätigung, bevor eine Änderung vorgenommen wird.

Sie machen eine Aktion rückgängig, indem Sie die Tastenfolge `Strg+x u` eingeben. Diese Eingabe können Sie beliebig oft vornehmen.

4.1.5 Emacs beenden

Das Tastenkürzel dafür lautet `Strg+x Strg+c`. Falls Sie die letzten Änderungen nicht gespeichert haben, werden Sie von Emacs darauf aufmerksam gefragt.

4.2 Vi: der Urvater

Vi war der erste Vollbild-Editor, den es gab. Er ist einerseits eines der meist genannten Argumente der UNIX®-Gegner, andererseits aber auch eines der Hauptargumente der Befürworter: Er ist recht kompliziert und erfordert einen hohen Lernaufwand aber er ist eines der mächtigsten Werkzeuge wenn man ihn einmal beherrscht. Ein erfahrener Benutzer von Vi bewegt Berge mit ein paar Tastenanschlägen und es gibt wohl ausser Emacs keinen anderen Texteditor mit vergleichbarer Leistung.

Eigentlich ist die in Mandrakelinux enthaltene Version der Vim, für *VI iMproved*,aber wir werden ihn hier weiterhin Vi nennen.

Falls Sie mehr über den Vi lernen wollen, können Sie sich die Seite Hands-On Introduction to the Vi Editor (`http://www.library.yale.edu/wsg/docs/vi_hands_on/`) oder die Vim Website (`http://www.vim.org/`) ansehen.

4.2.1 Einfügemodus, Befehlsmodus, `ex`-Modus...

Wir werden für unsere Übung mit Vi die gleiche Kommandozeile benutzen wie vorher mit dem Emacs. Also zurück zu unseren beiden Dateien, geben Sie ein:

```
$ vi datei1 datei2
```

Danach sollten Sie ein ähnliches Bild vor sich sehen wie das Folgende:

Abbildung 4-4: Ausgangsposition in VI

Sie befinden sich jetzt im ***Befehlsmodus*** und sehen die erste geöffnete Datei. Da Sie im Befehlsmodus keinen Text eingeben können, wechseln Sie in den ***Einfügemodus***.

Es folgen einige Tastenkürzel für den Einfügemodus:

- **a** und **i**: Eingabe von Text vor und hinter dem Cursor (**A** und **I** fügen Text am Ende und am Anfang der aktuellen Zeile ein)
- **o** und **O**: Einfügen von Text über und unter der aktuellen Zeile.

Im Einfügemodus erscheinen die Zeichen `--INSERT--` („--EINFÜGEN--“) am unteren Rand des Schirms damit Sie sehen, in welchem Modus Sie sich befinden. Sie können nur in diesem Modus Text eingeben. Durch drücken der Taste **Esc** kommen sie wieder den Befehlsmodus zurück.

Während der Texteingabe können Sie die Tasten **Backspace** und **Entf** zum Löschen von Zeichen benutzen. Die Pfeiltasten funktionieren in beiden Modi wie gewohnt. Im Befehlsmodus stehen Ihnen noch andere Tastenkombinationen zur Verfügung, die wir später behandeln werden.

Den `ex`-Modus erreichen Sie im Befehlsmodus durch Eingabe von **:**. Das Zeichen **:** erscheint im Mini-Puffer am unteren Rand des Schirms mit einem blinkenden Cursor daneben. Alles, was Sie nun von dort bis zum Drücken der Eingabe-Taste tippen, wird von Vi als `ex`-Befehl angesehen. Wenn Sie vor dem Drücken der Taste `Enter` alles Eingegebene inklusive des **:** löschen, verlassen Sie den `ex`-Modus und der Cursor springt wieder zurück auf seine vorherige Position.

Zum Sichern der Änderungen in eine Datei geben Sie im Befehlsmodus `:w` ein, zum Sichern des Pufferinhaltes in eine andere Datei dient `:w <datei_name>`.

4.2.2 Arbeiten mit Puffern

Da wir ja mehrere Dateien im Vi geöffnet haben, können wir mit `:next` (bzw. `:n`) zur nächsten Datei und mit `:prev` (bzw. `:N`) zur vorhergehenden Datei umschalten. Sie können auch die Sequenz `:e dateiname` benutzen, die Ihnen erlaubt, einen Dateinamen einzugeben und in diese Datei zu wechseln, falls sie bereits geöffnet ist. Auch hier kann die automatische Textvervollständigung benutzt werden.

Genauso wie im Emacs haben Sie auch hier die Option, mehrere Puffer auf dem Schirm darzustellen. Sie erreichen das mit dem Befehl `:split`.

Mit der Tastenkombination `Strg+w j` wechseln Sie in den unteren Puffer und mit `Strg+w k` wieder in den oberen zurück. Anstelle der Buchstaben **j** und **k** können Sie auch die jeweiligen Pfeiltasten einsetzen. Mit `:close` verschieben Sie den Puffer in den Hintergrund („verstecken“) und mit `:q` schließen Sie ihn.

Falls Sie versuchen, einen Puffer zu verstecken oder zu schließen, dessen Inhalt noch nicht gespeichert wurde, so wird der Befehl nicht ausgeführt sondern der folgende Text angezeigt:

`No write since last change (use ! to override)` („Seit der letzten Änderung nicht gespeichert (Benutzen Sie ! um das Schließen zu erzwingen)“).

Folgen Sie in diesem Fall der Anweisung und schreiben Sie `:q!` oder `:close!`.

4.2.3 Textbearbeitung und Verschieben von Text

Abgesehen von den Tasten **Backspace** und **Entf** im Editiermodus, kennt Vi im Befehlsmodus noch weitere Befehlsfolgen für das Entfernen, Kopieren, Einfügen und Ersetzen von Text. Alle hier aufgelisteten Befehle haben zwei Teile: die auszuführende Aktion und worauf diese Aktion einwirkt. Die Aktionen:

- **c**: Ersetzen (*Change*); der Editor löscht den gewählten Text und begibt sich zurück in den Editiermodus
- **d**: Löschen (*Delete*)
- **y**: Kopieren (*Yank*). Das sehen wir uns im nächsten Kapitel an.
- **.**: Wiederholen der letzten Aktion.

Hier wird erklärt, worauf diese Aktionen einwirken.

- **h**, **j**, **k**, **l**: ein Zeichen nach links, unten, oben, rechts[3]
- **e**, **b**, **w**: bewegt den Cursor zum Ende, zum Anfang des aktuellen Wortes und zum Anfang des nächsten Wortes.
- **^**, **0**, **$**: zum ersten Zeichen der aktuellen Zeile, zum Anfang der aktuellen Zeile und zu deren Ende.
- `f<x>`: zum nächsten Vorkommen des Zeichens `<x>`. So springt der Cursor bei Eingabe von `fe` zum nächsten Vorkommen von **e**.
- `/<string>`, `?<string>`: zum nächsten bzw. vorherigen Vorkommen der Zeichenkette `<string>`. `/foobar` bewegt den Cursor zum nächsten Fundort des Wortes `foobar`.
- {, }: zum Anfang bzw. Ende des aktuellen Absatzes;
- **G**, **H**: zum Ende der Datei, zum Anfang des Bildschirms.

Jede dieser „Zielmarkierungen“ oder Sprungbefehle kann eine vorangestellte Wiederholungsangabe bekommen. Für den Befehl **G** („Go“) bedeutet eine vorangestellte Zahl die Zeilennummer, wohin der Cursor bewegt werden soll. Mit diesen Vorgaben können Sie alle möglichen Kombinationen bilden.

Ein paar Beispiele:

- `6b`: geht 6 Wörter zurück
- `c8fk`: löscht den Text bis zum 8. Vorkommen des Buchstabens **k** und wechselt dann in den Editiermodus.
- `91G`: geht zur 91. Zeile der Datei.
- `d3$`: löscht den Rest der Zeile bis zu ihrem Ende und die beiden folgenden Zeilen.

3. Eine Abkürzung für `dl` (löscht ein Zeichen rechts) ist **x**; für `dh` kann man **X** benutzen und `dd` löscht die aktuelle Zeile.

Nun sind viele dieser Befehle alles Andere als intuitiv. Daher ist die beste Option, sie zu lernen, die Übung. Sie können jedoch erkennen, dass der Ausdruck „Berge mit ein paar Tastendrücken bewegen“ keine allzu große Übertreibung ist.

4.2.4 Ausschneiden, Kopieren, Einfügen

Eines der Kommandos von Vi, das wir bereits beim Kopieren von Text kennen gelernt haben, ist **y**. Zum Ausschneiden von Text benutzen wir **d**. Es gibt 27 Speicherseiten oder Puffer zum Speichern der Textteile: ein anonymer Puffer und 26 Puffer, die nach den Buchstaben des kleinen Alphabets benannt werden.

Zur Benutzung des anonymen Puffers geben Sie das entsprechende Kommando ohne Zusätze ein. Also kopiert der Befehl `y12w` die nächsten 12 Wörter nach dem Cursor in den anonymen Puffer [4]. Mit `d12w` schneiden Sie den betreffenden Bereich aus.

Um jetzt einen der 26 benannten Puffer zu benutzen geben Sie vor dem Befehl die Sequenz `"<x>` ein, wobei `<x>` den Namen des Puffers angibt. Um also die gleichen 12 Wörter in den Puffer **k** zu kopieren lautet der Befehl `"ky12w`, zum Ausschneiden entsprechend `"kd12w`.

Mit dem Befehl **p** oder **P** (für *Paste*) fügen Sie den Inhalt des anonymen Puffers hinter bzw. vor dem Cursor ein. Bein einem benannten Puffer lautet der Befehl `"<x>p` oder `"<x>P` entsprechend (zum Beispiel `"dp` zum Einfügen des Inhaltes des Puffers `d` hinter dem Cursor).

Schauen wir uns das Beispiel an:

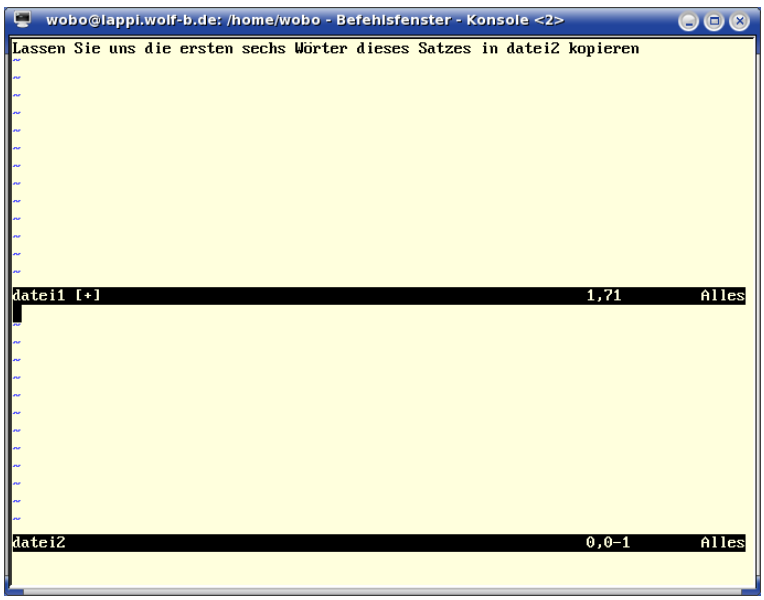


Abbildung 4-5: VI, vor dem Kopieren des Textblocks

Wir führen diese Aktion folgendermaßen aus:

- Wir kopieren die ersten 6 Wörter des Satzes in den Puffer **r** (zum Beispiel): `"ry6w`[5];
- wir gehen zum Puffer `datei2`, der im unteren Fenster dargestellt wird: `Strg+w j`;
- Hier fügen wir den Inhalt des Puffers **r** vor dem Cursor ein: `"rp`.

Damit bekommen wir das Resultat wie in Abbildung Abbildung 4-6.

4. Aber nur, wenn der Cursor auf dem Anfang des ersten Wortes steht!
5. `y6w` bedeutet tatsächlich: „*Yank 6 words*“.

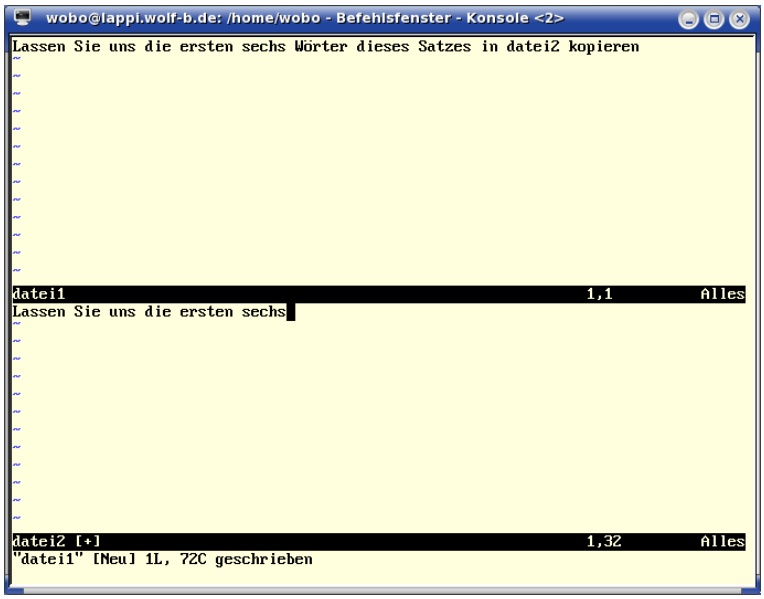


Abbildung 4-6: VI, nach dem Kopieren des Textblocks

Die Suche nach einer Textstelle ist recht einfach: Geben Sie im Befehlsmodus das Zeichen **/** ein, gefolgt von dem zu suchen den Text, und drücken Sie `Enter`. So sucht zum Beispiel `/Party` von der aktuellen Cursorposition ausgehend nach dem String `Party`. Mit **n** finden Sie nach dem ersten Vorkommen das zweite Vorkommen wobei die Suche bei Erreichen des Dateiendes wieder am Anfang der Datei beginnt. Die Rückwärts-Suche leiten Sie ein mit **?** anstelle von **/** ein.

4.2.5 Vi beenden

Sie beenden Vi mit `:q` (eigentlich schließt dieser Befehl den aktuellen Puffer, wie wir bereits gelernt haben. Wenn das allerdings der einzige Puffer ist, wird auch Vi beendet). Es gibt euch eine Abkürzung, da Sie meist nur eine einzelne Datei bearbeiten werden:

- `:wq` zum Speichern aller Änderungen und Verlassen von Vi (schneller ist `ZZ`), oder
- `:q!` zum Verlassen ohne Speichern.

Sie sollten beim Betrieb von mehreren Puffern daran denken, dass `:wq` nur den aktiven Puffer speichert und schließt.

4.3 Ein letztes Wort...

Natürlich haben wir hier viel mehr erklärt als ursprünglich benötigt (eigentlich wollten wir nur eine Textdatei bearbeiten), aber wir hoffen, Ihnen diese beiden Editoren mit ihren jeweiligen Möglichkeiten etwas näher gebracht zu haben. Es gibt zu den beiden noch eine Menge zu sagen, was man auch an der Anzahl der existierenden Bücher zu den beiden Editoren ermessen kann.

Nehmen Sie sich Zeit, diese Informationen zu verarbeiten, entscheiden Sie sich für einen der beiden Editoren oder lernen Sie nur, was Sie für notwendig halten. Auf jeden Fall wissen Sie jetzt, wo Sie weiter lernen können, falls gewünscht.

Kapitel 5. Kommandozeilenwerkzeuge

Wir möchten Ihnen in diesem Kapitel einige Werkzeuge für die Kommandozeile vorstellen, die sich im täglichen Einsatz als hilfreich erweisen können. Wenn Sie ausschließlich die grafische Oberfläche bevorzugen, können Sie dieses Kapitel sicher überspringen. Allerdings könnte ein kurzer Einblick Ihre Meinung auch ändern.

Jeder Befehl wird durch Beispiele erläutert. Dieses Kapitel ist aber auch als Übung für Sie gedacht um Funktionen und die Benutzung der Befehle zu verstehen.

5.1 Dateioperationen und Filter

Die meisten Operationen auf der Kommandozeile beziehen sich auf Dateien. In diesem Abschnitt behandeln wir das Betrachten und Filtern von Dateiinhalten, holen uns durch einfache Befehle benötigte Informationen aus Dateien und sortieren den Inhalt einer Datei.

5.1.1 cat, tail, head, tee: Befehle zur Dateiausgabe

Alle diese Befehle haben fast die gleiche Syntax: `befehls_name [Option(en)] [Datei(en)]`. Sie können auch mittels einer *Pipe* miteinander verbunden werden. Diese Befehle haben das gleiche Ziel: das Darstellen einer Datei nach bestimmten Kriterien.

Das Werkzeug `cat` verbindet Dateien und gibt das Resultat auf der Standardausgabe aus. Es ist eines der meistgenutzten Werkzeuge. Schreiben Sie beispielsweise

```
# cat /var/log/mail/info
```

um den Inhalt des Mailtransport-Logs auf der Standardausgabe darzustellen[1]. `cat` besitzt eine sehr hilfreiche Option (`-n`), mit der bei der Ausgabe die Zeilen nummeriert werden.

Da einige Dateien, speziell Log-Dateien von Diensten (wenn diese aktiv sind), gewöhnlich recht umfangreich sind[2], ist die Ausgabe der gesamten Datei auf dem Bildschirm nicht sehr hilfreich. Normalerweise benötigen Sie nur die Ansicht einiger Zeilen aus solch einer Datei. Dazu können Sie den Befehl `tail` benutzen. Das folgende Kommando gibt standardgemäß die letzten 10 Zeilen der Datei (also etwa `/var/log/mail/info`) aus:

```
# tail /var/log/mail/info
```

Mit der Option `-n` zeigen Sie die letzten `n` Zeilen einer Datei an, etwa die letzten beiden Zeilen:

```
# tail -n2 /var/log/mail/info
```

Der Befehl `head` ist dem Befehl `tail` sehr ähnlich, zeigt aber die ersten Zeilen einer Datei. Mit dem folgenden Befehl zeigen Sie die ersten 10 Zeilen der Datei `/var/log/mail/info` an:

```
# head /var/log/mail/info
```

Wie bei `tail` können Sie auch hier die Option `-n` zur Angabe der anzuzeigenden Zeilenanzahl einsetzen. Die ersten beiden Zeilen werden mit diesem Befehl angezeigt:

```
# head -n2 /var/log/mail/info
```

Sie können diese beiden Befehle gemeinsam nutzen. Um etwa die Zeilen 9 und 10 anzuzeigen, lassen Sie `head` die ersten 10 Zeilen der Datei auswählen und schicken das Ergebnis dann durch eine Pipe zum Befehl `tail`.

```
# head /var/log/mail/info | tail -n2
```

1. Einige Beispiele dieses Abschnitts basieren auf realistischen Arbeitsgängen mit Log-Dateien und Servern (Systemdiensten, Daemons). Dabei muss der Protokolldienst syslogd (er steuert die Logfunktion der Dienste) laufen. Für obiges Beispiel muss auch der E-Mail Dienst (etwa Postfix) laufen und Sie müssen als `root` angemeldet sein. Natürlich können Sie unsere Beispiele auch auf andere Dateien anwenden.
2. So enthält beispielsweise die Datei `/var/log/mail/info` Informationen über alle gesendeten E-Mails, über den Empfang der Mails von Benutzern des POP-Protokolls, usw.

Der zweite Teil des Kommandos wählt dann die letzten beiden Zeilen aus und stellt sie auf dem Bildschirm dar. In gleicher Weise wählen Sie die Zeile 20, gezählt vom Ende der Datei:

```
# tail -n20 /var/log/mail/info |head -n1
```

In diesem Beispiel weisen wir `tail` an, die letzten 20 Zeilen der Datei auszuwählen und sie durch eine Pipe an `head` zu übergeben. Der Befehl `head` zeigt dann die erste Zeile der ihm übergebenen Auswahl an.

Nehmen wir an, wir wollen das Resultat des letzten Beispiels auf dem Bildschirm darstellen und es gleichzeitig in der Datei `results.txt` speichern. Dabei hilft uns das Werkzeug `tee`. Dessen Syntax wie folgt lautet:

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

Jetzt können wir den vorherigen Befehl ändern:

```
# tail -n20 /var/log/mail/info |head -n1|tee results.txt
```

Lassen Sie uns ein weiteres Beispiel ansehen. Wir wollen die letzten 20 Zeilen auswählen, sie in der Datei `results.txt` speichern, aber nur die erste der 20 Zeilen am Bildschirm ansehen. Dazu geben wir ein:

```
# tail -n20 /var/log/mail/info |tee results.txt |head -n1
```

Der Befehl `tee` besitzt eine hilfreiche Option (`-a`), die das Anhängen von Daten an eine existierende Datei ermöglicht.

Gehen wir zurück zum Befehl `tail`. Da laufende Dienste kontinuierlich neue Aktionen und Ereignisse in ihren Log-Dateien speichern, ändern sich diese Dateien dauernd. Wenn Sie diese Veränderungen der Log-Datei in Echtzeit verfolgen wollen nutzen Sie die Option `-f` von `tail`:

```
# tail -f /var/log/mail/info
```

Hier werden alle Änderungen der Datei `/var/log/mail/info` sofort auf dem Bildschirm angezeigt. Diese Vorgehensweise ist besonders hilfreich, wenn man die Arbeitsweise des Systems beobachten will. Man kann beispielsweise bei der Verfolgung der Datei `/var/log/messages` die Meldungen des Systems und verschiedener Systemdienste beobachten.

Im nächsten Abschnitt werden wir zeigen, wie man mit Hilfe des Befehls grep die Meldungen des Postfix Dienstes aus allen anderen Meldungen herausfiltert.

5.1.2 grep: Auffinden von Textfolgen in Dateien

Weder der Name noch die Abkürzung (*„General Regular Expression Parser“*) sind sehr intuitiv, um so einfacher ist das, was dieser Befehl macht und wie er benutzt wird: grep durchsucht eine oder mehr Dateien nach einem angegebenen Muster. Die Syntax lautet:

```
grep [Option(en)] <muster> [Datei(en)]
```

Werden mehrere Dateien angegeben, so wird im Suchergebnis der Name der jeweiligen Datei vor der jeweiligen Fundstelle angezeigt. Mit der Option `-h` schalten Sie diese Anzeige aus. Die Option `-l` bewirkt, dass nur die auf das Suchmuster passende Dateinamen ausgegeben werden. Das Muster ist ein regulärer Ausdruck, wenn es auch meist nur aus einem einfachen Wort besteht. Es folgen die meistgenutzten Optionen von grep:

- `-i`: führt eine *case insensitive* Suche durch, d.h. Unterschiede in der Groß-/Kleinschreibung werden ignoriert;
- `-v`: umgekehrte Suche. Zeigt die Zeilen an, die das Muster **nicht** enthalten;
- `-n`: zeigt die Zeilennummer für jede Fundstelle an;
- `-w`: weist grep an, nur nach dem Muster als ganzem Wort zu suchen.

Lassen Sie uns zur Untersuchung der Log-Datei des E-Mail-Dienstes zurückkehren. Wir wollen alle Zeilen in der Datei `/var/log/mail/info` finden, die das Muster „postfix“ enthalten. Dazu geben wir folgendes Kommando ein:

```
# grep postfix /var/log/mail/info
```

Der grep-Befehl kann auch in einer Pipe benutzt werden. Daher bekommen wir das gleiche Resultat wie eben auch mit diesem Kommando:

```
# cat /var/log/mail/info | grep postfix
```

Wenn wir alle Zeilen auflisten wollen, die das Muster „postfix“ nicht enthalten, benutzen wir die Option `-v`:

```
# grep -v postfix /var/log/mail/info
```

Nehmen wir an, wir wollen alle Meldungen über erfolgreich versandte Mails finden. In diesem Fall müssen wir alle Zeilen der Log-Datei ausfiltern, die vom E-Mail Dienst stammen (also das Muster „postfix“ enthalten) und diese Zeilen nach der Meldung über einen erfolgreichen Versand („status=sent“) durchsuchen:

```
# grep postfix /var/log/mail/info |grep status=sent
```

Hier wird grep zweimal nacheinander benutzt. Das ist zwar erlaubt aber nicht besonders elegant. Wir erreichen das gleiche Ergebnis mit dem Befehl `fgrep`. Zuerst müssen wir eine Datei mit den Suchmustern erstellen. Die Muster müssen darin in einer Spalte untereinander angeordnet sein. Diese Datei (wir benutzen den Namen `patterns.txt`) wird mit dem folgenden Kommando erstellt:

```
# echo -e 'status=sent\npostfix' >./patterns.txt
```

Nun rufen wir das nächste Kommando auf, bei dem wir die Datei `patterns.txt` mit ihrer Liste der Suchmuster an das Werkzeug `fgrep` übergeben anstatt die „Doppelnutzung“ von grep anzuwenden:

```
# fgrep -f ./patterns.txt /var/log/mail/info
```

Die Datei `./patterns.txt` darf beliebig viele Suchmuster enthalten, wobei jedes Muster in einer Zeile für sich stehen muss. Um nun beispielsweise alle erfolgreich abgesandten Mails an `peter@mandrakesoft.com` zu finden, genügt es, mit folgendem Befehl die Mailadresse in die Datei `./patterns.txt` einzufügen:

```
# echo 'peter@mandrakesoft.com' >>./patterns.txt
```

Natürlich können Sie grep mit `tail` und `head` kombinieren. Die vorletzte Mail an `peter@mandrakesoft.com` finden Sie mit der folgenden Kommandozeile:

```
# fgrep -f ./patterns.txt /var/log/mail/info | tail -n2 | head -n1
```

Hier wenden wir den oben beschriebenen Filter an und schicken das Resultat über eine Pipe an die Befehle `tail` und `head`. Diese suchen dann den vorletzten Eintrag aus.

5.1.3 wc: Zählen von Elementen in Dateien

Der Befehl `wc` (*Word Count*) wird zum Erfassen der Anzahl von Zeichenfolgen und Wörtern in Dateien benutzt. Er hilft auch beim Zählen von Bytes, einzelnen Zeichen und beim Auffinden der längsten Zeile. Die Syntax ist wie folgt:

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

Einige hilfreiche Optionen:

- `-l`: gibt die Anzahl der Zeilen aus;
- `-w`: gibt die Anzahl der Wörter aus;
- `-m`: gibt die Gesamtzahl der Zeichen aus;
- `-c`: gibt die Anzahl der Bytes aus;
- `-L`: gibt die Länge der längsten Zeile des jeweiligen Textes aus.

Standardmäßig gibt der Befehl `wc` die Anzahl der Zeilen, Wörter und Zeichen aus. Es folgen einige Anwendungsbeispiele:

Gesucht wird die Anzahl der Benutzer in unserem System:

```
# wc -l /etc/passwd
```

Gesucht wird die Anzahl der Prozessoren in unserem System:

```
# grep "model name" /proc/cpuinfo |wc -l
```

Im vorigen Abschnitt ließen wir uns eine Liste der erfolgreich versandten Mails an Adressen in der Datei `./patterns.txt` anzeigen. Wenn wir nun die Anzahl dieser Mails wissen wollen, können wir das Resultat des Filters durch eine Pipe an den Befehl `wc` schicken:

```
# fgrep -f ./patterns.txt /var/log/mail/info | wc -l
```

5.1.4 sort: Sortieren des Datei-Inhaltes

Hier die Syntax dieses mächtigen Sotierwerkzeugs[3]:

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

Lassen Sie uns einen Teil der Datei `/etc/passwd` sortieren. Wie Sie sehen können, ist der Inhalt nicht sortiert:

```
$ cat /etc/passwd
```

Zur Sortierung nach dem Feld `login` verwenden wir den folgenden Befehl:

```
$ sort /etc/passwd
```

Der Befehl `sort` sortiert, ohne Optionen gestartet, Daten in aufsteigender Folge, beginnend beim ersten Feld (in unserem Fall das Feld `login`). Wenn wir die Daten in absteigender Folge sehen wollen müssen wir die Option `-r` verwenden:

```
$ sort -r /etc/passwd
```

Jeder Benutzer hat seine eigene `UID` in der Datei `/etc/passwd`. Lassen Sie uns die Datei in absteigender Folge nach dem Feld `UID` sortieren:

```
$ sort /etc/passwd -t":" -k3 -n
```

In diesem Beispiel werden die folgenden `sort` Optionen verwendet:

- `-t":"`: legt fest, dass das Trennungszeichen zwischen den Feldern das Zeichen `":"` ist;
- `-k3`: die Sortierung soll nach dem dritten Feld erfolgen;
- `-n`: die Sortierung soll nur numerische Daten umfassen, keine alphabetischen Daten.

Das Gleiche kann man natürlich auch in umgekehrter Folge machen:

```
# sort /etc/passwd -t":" -k3 -n -r
```

`sort` hat zwei weitere wichtige Optionen:

- `-u`: führt eine strikte Sortierung durch, doppelte Felder werden missachtet;
- `-f`: ignoriert Groß-/Kleinschreibung.

Wenn wir schließlich den Benutzer mit der höchsten `UID` finden wollen, benutzen wir diese Befehlskette:

```
$ sort /etc/passwd -t":" -k3 -n |tail -n1
```

wobei wir die Datei `/etc/passwd` in aufsteigender Folge nach dem Feld `UID` sortieren und das Resultat durch eine Pipe an das Programm `tail` übergeben. Dieses gibt den ersten Wert der Liste aus.

3. Wir behandeln `sort` hier nur kurz, man könnte aber über die gesamten Eigenschaften ganze Bücher schreiben.

5.2 find: Sucht nach Dateien, die vorgegebenen Kriterien entsprechen

`find` ist eines der althergebrachten UNIX®-Werkzeuge. Seine Aufgabe ist es, rekursiv ein oder mehrere Verzeichnisse nach Dateien zu durchsuchen, die ein gegebenes Muster aufweisen. Einerseits ist das Werkzeug sehr hilfreich, andererseits ist seine Syntax nicht einfach und es braucht einige Übung bevor man es richtig anwenden kann. Die allgemeine Syntax lautet:

```
find [Verzeichnis] [Option(en)] [Test] [Aktion]
```

Ohne Angabe eines Verzeichnisses wird das aktuelle Verzeichnis durchsucht. Ohne spezielle Suchkriterien wird das Ergebnis wie „vorhanden“ gewertet und daher alle vorhandenen Dateien aufgelistet. Die möglichen Optionen, Tests und Aktionen sind derart zahlreich, dass wir hier nur einige wenige behandeln. Lassen Sie uns mit den Optionen beginnen:

- `-xdev`: keine Suche in Verzeichnissen, die auf anderen Dateisystemen liegen.
- `-mindepth <n>`: Beginn der Suche erst `n` Stufen unterhalb des angegebenen Verzeichnisses.
- `-maxdepth <n>`: Suche nach Dateien in maximal `n` Stufen unterhalb des angegebenen Verzeichnisses.
- `-follow`: Verfolgen von symbolischen Links falls diese auf Verzeichnisse zeigen. Standardmäßig folgt `find` den Links nicht.
- `-daystart`: bei zeitbezogenen Tests (siehe unten) wird der aktuelle Tag als Zeitmarke benutzt anstelle des standardmäßigen Zeitstempels, der 24 Stunden zur aktuellen Zeit zurück liegt.

Ein Test kann einer oder mehrere ***elementare*** Eigenschaften umfassen. Einige dieser Eigenschaften sind:

- `-type <file_type>`: Suche nach einem angegebenen Dateityp. Der Dateityp kann einer der folgenden sein: `f` (normale Datei), `d` (Verzeichnis), `l` (symbolischer Link), `s` (Socket), `b` (blockorientierte Datei), `c` (zeichenorientierte Datei) oder `p` (eine „named Pipe“).
- `-name <pattern>`: findet Dateien, die dem gegebenen Muster entsprechen. Mit dieser Option wird das Muster wie ein Muster für ***Shell Namenserweiterung*** angesehen (siehe Kapitel *Platzhalter in der Shell*, Seite 26).
- `-iname <pattern>`: gleiche Wirkung wie `-name`, allerdings ohne auf Groß-/Kleinschreibung zu achten.
- `-atime <n>`, `-amin <n>`: findet Dateien, die zuletzt vor `n` Tagen (`-atime`) oder vor `n` Minuten (`-amin`) geöffnet wurden. Sie können auch die Optionen `<+n>` oder `<-n>` angeben, wobei die Suche nach Dateien ausgeführt wird, die vor höchstens oder mindestens `n` Tagen/Minuten geöffnet wurden.
- `-anewer <a_file>`: sucht nach Dateien, die später als die Datei `a_file` geöffnet wurden.
- `-ctime <n>`, `-cmin <n>`, `-cnewer <file>`: gleiche Wirkung wie bei den Optionen `-atime`, `-amin` und `-anewer`, bezieht sich aber auf den Zeitpunkt der letzten Änderung des Inhaltes der Datei.
- `-regex <pattern>`: gleiche Wirkung wie die Option `-name`. Allerdings wird `pattern` als ***regulärer Ausdruck*** behandelt.
- `-iregex <pattern>`: wie `-regex`, allerdings ohne Beachtung von Groß-/Kleinschreibung.

Es gibt zahlreiche andere Tests, die Sie in `find`(1) finden können. Zusammengesetzte Tests können Sie in dieser Art anwenden:

- `<c1> -a <c2>`: ist wahr, wenn sowohl `c1` als auch `c2` zutreffen; `-a` ist sowieso vorgegeben, also können Sie `<c1> <c2> <c3>` schreiben, wenn `c1`, `c2` und `c3` zutreffen sollen.
- `<c1> -o <c2>`: ist wahr, wenn entweder `c1` oder `c2` zutreffen oder beide. Beachten Sie, dass `-o` eine niedrigere ***Priorität*** als `-a` besitzt. Daher müssen Sie bei der Suche nach Dateien, die entweder auf `c1` oder `c2` und auf `c3` passen, eine Klammerkonstruktion verwenden: `( <c1> -o <c2> ) -a <c3>`. Klammern als Teil eines Ausdrucks müssen ***maskiert*** werden, da sie sonst von der shell interpretiert werden!
- `-not <c1>`: kehrt Bedingungen um, also ist `-not <c1>` wahr, wenn `c1` nicht zutrifft.

Schließlich können Sie für jede gefundene Datei auch eine Aktion festlegen. Die meist genutzten Aktionen sind:

- `-print`: gibt einfach den Namen der Datei auf der Standardausgabe an. Das ist die Standard-Aktion.
- `-ls`: gibt auf der Standardausgabe das Äquivalent des Befehls `ls -ilds` für jede gefundene Datei aus.

- `-exec <command_line>`: wendet die Befehlszeile `command_line` auf jede gefundene Datei an. Die Befehlszeile `command_line` muss mit dem Zeichen `;` enden, das allerdings wieder maskiert werden muss um nicht von der shell interpretiert zu werden. Der Dateiname wird dabei von `{}` eingefasst. Siehe Beispiele.
- `-ok <command>`: identisch mit `-exec`. Allerdings wird bei jeder Datei eine Bestätigung verlangt.

Am besten kann man all diese Optionen und Parameter mit einigen Beispielen darstellen. Nehmen wir an, Sie wollen alle Unterverzeichnisse im Verzeichnis `/usr/share` auflisten. Die Eingabe dazu lautet:

```
# find /usr/share -type d
```

Angenommen, Sie betreiben einen HTTP Server und alle Ihre HTML-Dateien liegen im Verzeichnis `/var/www/html`, das auch das aktuelle Verzeichnis ist. Jetzt wollen Sie alle Dateien finden, die seit einem Monat nicht verändert wurden. Da die Seiten von verschiedenen Autoren stammen gibt es sowohl Dateien mit der Endung `html` als auch solche mit der Endung `htm`. Sie wollen all diese Dateien in das Verzeichnis `/var/www/obsolete` verlinken. Dazu geben Sie ein[4]:

```
# find \( -name "*.htm" -o -name "*.html" \) -a -ctime -30 -exec ln {} /var/www/obsolete \;
```

Dies ist ein recht komplexes Beispiel, das wir jetzt genauer erklären werden. Die Tests lauten wie Folgt:

```
\( -name "*.htm" -o -name "*.html" \) -a -ctime -30
```

wodurch das erreicht wird, was wir wollen: es findet alle Dateien, die auf `.htm` oder `.html` enden („`\( -name "*.htm" -o -name "*.html" \)`“) **und** (`-a`) die innerhalb der letzten 30 Tage nicht verändert wurden (`-ctime -30`). Beachten Sie die Klammern: sie müssen hier angewendet werden, da `-a` eine höhere Priorität hat. Ohne Klammern würden alle Dateien aufgelistet, die mit `.htm` enden und dazu alle Dateien, die mit `.html` enden und in den letzten 30 Tagen nicht geändert wurden. Das ist aber nicht genau das, was wir wollten. Beachten Sie auch, dass die Klammern maskiert wurden: Wenn wir `( .. )` anstelle von `\( .. \)` schreiben würden, hätte die shell die Klammern interpretiert und versucht, den Befehl `-name "*.htm" -o -name "*.html"` in einer Sub-Shell auszuführen... Weitere mögliche Arten der Maskierung bestehen in der Einfassung mit Anführungszeichen, jedoch ist hier ein Backslash vorzuziehen, da nur ein einzelnes Zeichen maskiert werden muss.

Schließlich fehlt noch der für jede Datei auszuführende Befehl:

```
-exec ln {} /var/www/obsolete \;
```

Hier muss das Zeichen `;` vor der shell maskiert werden, da die Shell das Zeichen anderenfalls als Befehlstrenner interpretieren und `find` das Fehlen eines Argumentes für `-exec` melden würde.

Ein letztes Beispiel: Sie haben ein ausgedehntes Verzeichnis (`/shared/images`) mit allen möglichen Arten von Bilddateien. Sie benutzen regelmäßig den Befehl touch um den Zeitstempel der Datei `stamp` in diesem Verzeichnis als Zeitreferenz zu aktualisieren. Sie wollen nun alle ***JPEG***-Bilder finden, die neuer als die Datei `stamp` sind. Da Ihre Bilder aus verschiedenen Quellen stammen, besitzen diese Bilder die Endungen `jpg`, `jpeg`, `JPG` oder `JPEG`. Natürlich wollen Sie auch keine Suche im `alten` Verzeichnis durchführen. Die Dateiliste soll per Mail an Sie geschickt werden (Ihr Kennzeichen ist `birgit`):

```
# find /shared/images -cnewer     \
     /shared/images/stamp       \
     -a -iregex ".*\.jpe?g"     \
     -a -not -regex ".*/old/.*" \
       | mail birgit -s "New images"
```

Natürlich ist es nicht gerade effizient, diese Befehlszeile jedes Mal neu eingeben zu müssen. Außerdem soll der Befehl regelmäßig ausgeführt werden. Hier ist eine einfache Lösung zur regelmäßigen Ausführung des Befehls:

4. Beachten Sie, dass sowohl `/var/www` als auch `/var/www/obsolete` im gleichen Dateisystem liegen müssen!

5.3 Planung der Ausführung von Befehlen

5.3.1 crontab: Auflisten und Editieren der Datei crontab

`crontab` ist ein Hilfsprogramm mit dem Sie Befehle in regelmäßigen Zeitintervallen starten können ohne dabei selbst am System angemeldet sein zu müssen. `crontab` wird die Ausgabe Ihres Befehls per Mail an Sie schicken. Sie können die Zeitintervalle in Minuten, Stunden, Tagen und sogar Monaten festlegen. Je nach angegebener Option verhält sich `crontab` unterschiedlich:

- `-l`: gibt den Inhalt Ihrer aktuellen Datei `crontab` aus.
- `-e`: damit starten Sie das Editieren der Datei `crontab`.
- `-r`: entfernt Ihre aktuelle `crontab`-Datei.
- `-u <user>`: führt eine der oben genannten Optionen für den Benutzer `<user>` aus. Das darf natürlich nur `root`.

Fangen wir mit dem Editieren einer `crontab`-Datei an. Wenn Sie die Umgebungsvariable `EDITOR` oder `VISUAL` gesetzt haben befinden Sie sich nach der Eingabe von `crontab -e` in Ihrem bevorzugten Editor, anderenfalls wird Vi geöffnet. Eine Zeile in der `crontab`-Datei besteht aus sechs Feldern. Die ersten fünf Felder bestehen aus den Angaben für die Minuten, Stunden, Tage und Wochentage des Intervalls. Das sechste Feld beinhaltet den auszuführenden Befehl. Zeilen, die am Anfang das Zeichen # aufweisen, werden als Kommentar betrachtet und von `crond` (dem Programm, das für die Ausführung der `crontab`-Dateien verantwortlich ist) ignoriert. Hier ist ein Beispiel einer `crontab`:

Um das Folgende in lesbarer Form darzustellen, mussten einige lange Zeilen umbrochen und in einer neuen Zeile fortgesetzt werden. Wenn also am Ende einer Zeile das Zeichen \ steht, bedeutet das, dass die Zeile nach einem Umbruch fortgesetzt wird. Diese Konvention wird in `Makefiles`, in der shell und auch in anderen Fällen eingesetzt.

```
# Falls Sie nicht per Mail benachrichtigt werden wollen
# kommentieren Sie die folgende Zeile aus:
#MAILTO="your_email_address"
#
# Bericht über neue Bilder alle 2 Tage um 2 Uhr nachmittags,
# ausgehend vom oben genannten Beispiel - danach soll die
# "stamp"-Datei wieder aktualisiert werden.
# Das Zeichen "%" wird als neue Zeile interpretiert.
# Dadurch können Sie mehrere Befehle in eine Zeile schreiben.
0 14 */2 * *  find /shared/images              \
  -cnewer /shared/images/stamp                 \
  -a -iregex ".*\.jpe?g"                       \
  -a -not -regex                               \
    ".*/old/.*"%touch /shared/images/stamp
#
# Am ersten Weihnachtsfeiertag eine Melodie spielen :)
0 0 25 12 * mpg123 $HOME/sounds/merryxmas.mp3
#
# Drucke jeden Dienstag um 17 Uhr die Einkaufsliste aus...
0 17 * * 2 lpr $HOME/shopping-list.txt
```

Es gibt ausser den hier gezeigten noch mehrere andere Arten, die Zeitintervalle anzugeben. Sie können etwa eine durch Kommata getrennte Folge von ***bestimmten Zahlen*** (`1,14,23`) oder einen Zahlenbereich (`1-15`), ja, sogar eine Kombination von beiden (`1-10,12-20`), optional mit einer Schrittangabe (`1-12,20-27/2`), angeben. Jetzt ist es an Ihnen, hier hilfreiche und sinnvolle Befehle einzugeben!

5.3.2 at: Planen eines nur einmal auszuführenden Befehls

Sie möchten vielleicht auch einen Befehl eingeben, der einmal zu einer vorbestimmten Zeit, aber nicht regelmäßig ausgeführt wird. Lassen Sie sich beispielsweise an eine Verabredung um 18:00 Uhr erinnern. Sie arbeiten mit X, das Paket `X11R6-contrib` ist installiert und Sie wollen um 17:30 Uhr an den Aufbruch erinnert werden. Dann ist `at` das richtige Programm:

```
$ at 5:30pm
  # Sie befinden sich jetzt am "at"-Prompt
at> xmessage "Zeit, aufzubrechen! Verabredung um 18:00"
  # Drücken Sie Strg-d zum Verlassen
at> <EOT>
$
```

Die Zeit kann in mehreren Varianten angegeben werden:

- `now + <interval>`: jetzt plus dem angegebenen Intervall (Optional. Ohne angegebenes Intervall bedeutet es: Jetzt.). Die Syntax für das Interval ist `<n> (minuten|stunden|tage|wochen|monate)`. Sie könnten beispielsweise angeben: `now + 1 hour` (in einer Stunde ab jetzt), `now + 3 days` (in drei Tagen ab heute), usw.
- `<zeit> <tag>`: damit wird der Zeitpunkt genau festgelegt. Die Angabe des Parameters `<zeit>` ist vorgeschrieben. `at` ist bei der Formatierung sehr liberal: Sie können beispielsweise `0100`, `04:20`, `2am`, `0530pm`, `1800` oder einen von drei festgelegten Zeitpunkten angeben: `noon`, `teatime` (16:00) oder `midnight`. Die Angabe des Parameters `<tag>` hingegen ist optional. Auch diese Angabe kann in verschiedenen Arten erfolgen: `12/20/2001` (also der 20. Dezember 2001) oder in europäischer Art `20.12.2001`. Wenn Sie die Jahresangabe weglassen wollen ist nur die europäische Angabe erlaubt: `20.12`. Den Monat können Sie auch in Form von Buchstaben eingeben: `Dec 20` oder `20 Dec` sind beide gültig.

`at` akzeptiert verschiedene Optionen:

- `-l`: gibt eine Liste der aktuell anliegenden Jobs aus, in der das erste Feld die Nummer des Jobs angibt. Das ist das Äquivalent zum Befehl `atq`.
- `-d <n>`: entfernt den Job Nummer `<n>` aus der Warteschlange. Die Jobnummern bekommen Sie mit dem Befehl `atq`. Das ist das Äquivalent zu dem Befehl `atrm <n>`.

Wie gewohnt finden Sie weitere Optionen in der `manpage` von at(1).

5.4 Archivieren und Datenkomprimierung

5.4.1 tar: Tape ARchiver

Wie `find` ist auch `tar` eines der älteren UNIX®-Werkzeuge. Dementsprechend „speziell“ ist die Syntax:

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

Hier finden Sie eine Liste mit einigen Optionen. Alle Optionen haben auch eine längere Schreibweise, die Sie aber bitte in der *man page* von `tar` nachlesen wollen.

der den kurzen Optionsangaben vorangehende Strich (`-`) kann bei der aktuellen Version von `tar` weggelassen werden, es sei denn, es folgt eine kurze Schreibweise nach einer Langversion.

- `c`: erstellt ein neues Archiv.
- `x`: extrahiert Dateien aus einem bestehenden Archiv.
- `t`: listet die Dateien eines bestehenden Archives auf.
- `v`: listet die einem Archiv hinzugefügten oder daraus extrahierten Dateien auf. In Verbindung mit der Option `t` (siehe oben) wird eine lange Liste von Dateien anstelle einer Kurzform ausgegeben.

- `f <datei_name>`: erstellt ein Archiv mit dem Namen `datei_name`, extrahiert Dateien aus einem Archiv mit diesem Namen oder listet die Dateien eines Archivs mit dem Namen `file_name` auf. Wird dieser Parameter weggelassen so wird als Standard `/dev/rmt0` angenommen, da dies im Allgemeinen der spezielle Dateiname für einen ***Streamer*** ist. Ist der Parameter für die Datei ein - (ein Strich) so wird die Ein- oder Ausgabe (je nachdem, ob es sich um eine Erstellung eines Archivs oder eine Extrahierung aus einem Archiv handelt) auf die Standard-Ein-/Ausgabe geleitet.
- `z`: weist `tar` an, das zu erstellende Archiv mit `gzip` zu komprimieren. Bei einer Extraktion wird damit angegeben, dass das Archiv mit `gzip` komprimiert wurde.
- `j`: bedeutet das Gleiche wie `z`, nur, dass es sich bei dem Komprimierungsprogramm um `bzip2` handelt.
- p: bei der Extraktion von Dateien aus einem Archiv werden alle Dateiattribute beibehalten (Besitzrechte, letzter Zugriff, usw.). Sehr hilfreich bei Sicherungen eines Systems.
- `r`: fügt eine auf der Kommandozeile angegebene Liste von Dateien zu einem existierenden Archiv hinzu. Beachten Sie hierbei, dass das Archiv, dem Sie die Dateien hinzufügen wollen, **nicht** komprimiert sein darf!
- `A`: fügt auf der Kommandozeile angegebene Archive zu einem mit der Option `f` erstellten Archiv hinzu. Wie bei der Option `r` dürfen auch hier die Archive **nicht** komprimiert sein.

Es gibt eine Unmenge weiterer Optionen, die Sie ihrer Gesamtheit in tar(1) finden können. Schauen Sie sich dort beispielsweise die Option `d` an. Lassen Sie uns mit einem kleinen Beispiel fortfahren. Sie wollen ein Archiv erstellen, dass alle Bilder im Verzeichnis `/shared/images` enthält, mit `bzip2` komprimiert ist, den Namen `images.tar.bz2` trägt und in Ihrem persönlichen Verzeichnis gespeichert wird. Dazu verwenden Sie die folgende Befehlszeile:

```
 #
 # Achtung: Sie müssen sich dazu in dem Verzeichnis befinden, aus dem
 # die zu archivierenden Dateien stammen!
 #
$ cd /shared
$ tar cjf ~/images.tar.bz2 images/
```

Wie Sie sehen, haben wir hier drei Optionen benutzt: `c` zur Erstellung des Archivs, `j` zur Komprimierung mit `bzip2` und `f ~/images.tar.bz2` zur Angabe, dass das Archiv den Namen `images.tar.bz2` bekommt und in Ihrem persönlichen Verzeichnis abgelegt wird. Nun prüfen wir, ob das Archiv valide ist und lassen uns dazu die archivierten Dateien auflisten:

```
 #
 # Gehen Sie dazu in Ihr persönliches Verzeichnis
 #
$ cd
$ tar tjvf images.tar.bz2
```

Wir weisen also `tar` an, die Dateien des Archivs `images.tar.bz2` (`f images.tar.bz2`) aufzulisten (`t`), teilen ihm mit, dass das Archiv mit `bzip2` komprimiert wurde (`j`) und dass wir eine ausführliche Liste sehen wollen (`v`). Angenommen, Sie haben das Verzeichnis mit den Bildern mittlerweile gelöscht. Glücklicherweise ist das Archiv in Ordnung und Sie können es wieder an den ursprünglichen Platz (`/shared`) entpacken. Dabei wollen Sie aber die Funktionalität Ihres `find`-Befehls für neue Bilder nicht durcheinander bringen. Daher müssen Sie nun die Option zur Erhaltung aller Dateiattribute benutzen:

```
 #
 # Wechseln Sie zum Zielverzeichnis
 #
$ cd /shared
$ tar jxpf ~/images.tar.bz2
```

Das wars!

Als Nächstes wollen Sie vielleicht das Verzeichnis `images/cars` aus dem Archiv extrahieren, aber nur dieses Verzeichnis, sonst nichts. Dazu geben Sie die folgende Befehlszeile ein:

```
$ tar jxf ~/images.tar.bz2 images/cars
```

Falls Sie versuchen, spezielle Dateien zu archivieren wird `tar` die Dateien behandeln, als das, was sie sind (spezielle Dateien) und ihren Inhalt nicht sichern. Daher können Sie beispielsweise die Datei `/dev/mem` in ein Archiv einfügen. `tar` geht auch korrekt mit Links um, also brauchen Sie sich darum ebenfalls keine Sorgen machen. Zum Thema „symbolische Links“ sollten Sie sich die Option `h` im Handbuchauszug ansehen.

5.4.2 bzip2 und gzip: Programme zur Datenkompression

Wir haben diese beiden Programm bereits bei der Behandlung von `tar` erwähnt. Entgegen der Funktionsweise von WinZip® unter Windows® wird das Archivieren und Komprimieren von Daten von zwei verschiedenen Programmen erledigt – `tar` zur Archivierung und die die zwei Programme, über die wir jetzt schreiben, zur Datenkompession: `bzip2` und `gzip`. Sie können auch andere Komprimierungs-Programme benutzen: `zip`, `arj` oder `rar` existieren auch unter GNU/Linux, werden aber seltener benutzt.

Am Anfang wurde `bzip2` eigentlich als Nachfolger für `gzip` entwickelt. Seine Kompressionsrate war im Allgemeinen besser als bei `gzip`, dafür braucht es mehr Arbeitsspeicher zur Bearbeitung der Daten. `gzip` wird allerdings aus Kompatibilitätsgründen immer noch oft benutzt.

Beide Programme haben eine ähnliche Syntax:

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

Wenn kein Dateiname angegeben wird warten beide Befehle auf Daten von der Standardeingabe und geben das Resultat ihrer Verarbeitung an die Standardausgabe aus. Daher können beide Programme auch in Pipes benutzt werden. Beide Programme besitzen auch eine Reihe von gemeinsamen Optionen:

- `-1`, ..., `-9`: setzt die Kompressionsrate fest. Je höher die Ziffer, desto stärker die Komprimierung, aber auch desto langsamer die Verarbeitung.
- `-d`: de-komprimieren von Dateien. Ein Äquivalent zu den Befehlen `gunzip` oder `bunzip2`.
- `-c`: gibt das Ergebnis der Kompression/Dekompression von als Parameter angegebenen Dateien an die Standardausgabe.

Standardmäßig löschen beide Programme ohne die Angabe der Option `-c` die Dateien, die sie komprimiert oder dekomprimiert haben. Bei `bzip2` können Sie das durch die Benutzung der Option `-k` vermeiden. `gzip` besitzt keine ähnliche Option.

Es folgen ein paar Beispiele. Nehmen wir an, Sie möchten alle Dateien im aktuellen Verzeichnis mit der Endung `.txt` komprimieren und benutzen dazu `bzip2` mit maximaler Kompressionsrate. Dazu geben Sie diesen Befehl ein:

```
# bzip2 -9 *.txt
```

Nun wollen Sie Ihr Bildarchiv jemandem zeigen, der aber anstelle von `bzip2` nur `gzip` benutzen kann. Dazu müssen Sie das ganze Archiv nicht entpacken und wieder komprimieren. Entpacken Sie es auf die Standardausgabe, nutzen eine Pipe, komprimieren die Daten der Standardeingabe und leiten die Ausgabe in das neue Archiv. Und das geht so:

```
bzip2 -dc images.tar.bz2 | gzip -9 >images.tar.gz
```

Sie könnten auch `bzcat` anstelle von `bzip2 -dc` nehmen. Es gibt ein Äquivalent für `gzip`, der Name ist aber `zcat`, nicht **gzcat**. Sie können auch `bzless` statt `bzip2` Datei und `zless` statt `gzip` verwenden, falls Sie die komprimierten Dateien direkt ansehen wollen anstatt sie erst zu dekomprimieren. Als Übung sollten Sie jetzt den Befehl finden, mit dem Sie – ohne `bzless` oder `zless` zu benutzen – komprimierte Dateien direkt ansehen können, ohne sie vorher zu entpacken.

5.5 Viel, viel mehr...

Es gibt derartig viele Befehle, dass ein umfassendes Buch darüber die Größe einer Enzyklopädie erreichen würde. Dieser Abschnitt hat nicht einmal ein Zehntel des gewählten Themas behandelt, trotzdem können Sie mit dem, was Sie hier gelernt haben, schon Einiges anfangen. Wenn Sie daran interessiert sind, sollten Sie sich einige Handbuchauszüge ansehen: sort(1), sed(1) und zip(1) (ja, genau das, was Sie annehmen: Sie können unter GNU/Linux auch `.zip`-Archive anlegen), convert(1), usw. Die beste Art, sich mit diesen Werkzeugen vertraut zu machen ist ständiges Üben und Experimentieren. Sie werden sicher eine Menge Möglichkeiten dazu finden, sogar einige unerwartete. Viel Spaß!

Kapitel 6. Prozesskontrolle

Wir haben im Kapitel *Prozesse*, Seite 10 bereits gezeigt, was ein Prozess ist. Jetzt lernen Sie, wie man Prozesse und ihre Eigenschaften auflistet und wie man sie „manipulieren“ kann.

6.1 Einiges über Prozesse

Man kann Prozesse überwachen, sie beenden, anhalten und wieder starten, usw. Zum Verständnis der folgenden Beispiele sollten Sie etwas mehr über Prozesse wissen.

6.1.1 Der Prozessbaum

Wie alle Dateien, so sind auch alle auf einem GNU/Linux-System laufenden Prozesse in Form eines Baums organisiert. Die Wurzel des Baumes ist `init`, ein Prozess der Systemebene, der beim Betriebssystemstart gestartet wird. Das System ordnet jedem Prozess eine Nummer (PID, *Process ID*) zu, um ihn einwandfrei identifizieren zu können. Jeder Prozess erbt ebenso die PID seines Elternprozesses (PPID, *Parent Process ID*). `init` ist sein eigener Vater, die PID und PPID von `init` ist 1.

6.1.2 Signale

Jeder Prozess unter UNIX® kann auf an ihn gesendete Signale reagieren. Es gibt 64 verschiedene Signale, die entweder durch ihre Nummer (ab `1` aufwärts) oder durch ihren symbolischen Namen (`SIGx`, wobei `x` der Name des Signals ist) identifiziert werden. Die 32 „oberen“ Signale (`33` bis `64`) sind Echtzeit-Signale, deren Behandlung die Grenzen dieses Kapitels übersteigt. Für jedes dieser Signale kann der Prozess seine eigene Reaktion definieren, ausser für zwei spezielle Signale: Signal Nummer 9 (`KILL`) und Nummer 19 (`STOP`).

Signal `9` beendet einen Prozess unwiderruflich, ohne ihm Zeit für eine saubere Beendigung zu geben. Dieses Signal sendet man an Prozesse, die hängen geblieben sind oder andere Probleme zeigen. Eine komplette Liste der Signale bekommt man mit dem Befehl `kill -l`.

6.2 Informationen über Prozesse: ps und pstree

Diese beiden Befehle zeigen die aktuell auf dem System laufenden Prozesse in einer Liste an, deren Kriterien Sie festlegen können.

6.2.1 ps

`ps` ohne jegliche Argumente zeigt Ihnen nur die Prozesse, die Sie gestartet haben und die mit dem von Ihnen benutzten Terminal verbunden sind:

```
$ ps
      PID TTY          TIME CMD
    18614 pts/3    00:00:00 bash
    20173 pts/3    00:00:00 ps
```

Wie bei vielen UNIX®-Werkzeugen üblich, hat auch `ps` einige Optionen, deren meist genutzte wir hier vorstellen:

- `a`: zeigt Prozesse an, die von allen Usern gestartet wurden;
- `x`: zeigt Prozesse an, die entweder keinem Kontrollterminal oder einem von Ihnen nicht benutzten Terminal zugeordnet sind;
- `u`: zeigt für jeden Prozess den Benutzer an, der ihn gestartet hat sowie die Zeit, wann er gestartet wurde.

Es gibt noch einige weitere Optionen. Sehen Sie sich dazu den Handbuchauszug ps(1) an.

Die Ausgabe des Befehls ist in mehrere Felder eingeteilt: das Interessanteste davon ist das Feld PID, das die Prozessnummer anzeigt. Das Feld CMD zeigt den Namen des ausgeführten Befehls an. Eine gebräuchliche Kommandozeile des Befehls ps lautet etwa so:

```
$ ps ax | less
```

Damit bekommen Sie eine Liste aller zur Zeit laufenden Prozesse, aus der Sie bei Bedarf einen oder mehrere problematische Prozesse heraussuchen und beenden können.

6.2.2 pstree

Der Befehl pstree zeigt Prozesse in der Baumstruktur an. Einer der damit verbundenen Vorteile ist, dass Sie sofort die Elternprozesse eines Prozesses sehen können: wenn Sie beispielsweise eine ganze Gruppe von Prozessen beenden wollen, die alle Kindprozesse der gleichen Eltern sind, reicht es, den Elternprozess zu beenden. Am Besten lassen Sie sich dabei mit der Option -p die PID jedes Prozesses anzeigen sowie mit der Option -u auch den Namen des Benutzers, der den Prozess gestartet hat. Da die Baumstruktur aller Prozesse normalerweise sehr lang ist, sollten Sie pstree auf die folgende Art aufrufen:

```
$ pstree -up | less
```

Das gibt Ihnen einen Überblick über die gesamte Prozess-Baumstruktur.

6.3 Signale an die Prozesse: kill, killall und top

6.3.1 kill, killall

Diese beiden Befehle werden dazu benutzt, Signale an die Prozesse zu schicken. Der Befehl kill benötigt als Argument eine Prozessnummer, während killall nach dem Prozessnamen verlangt.

Beiden Befehlen kann optional auch die Nummer des zu sendenden Signals als Argument mitgegeben werden. Standardmäßig senden beide das Signal 15 (TERM) an den entsprechenden Prozess. So lautet beispielsweise der Befehl zur sofortigen Beendigung des Prozesses mit der PID 785:

```
$ kill 785
```

Wenn Sie dabei das Signal 19 (STOP) senden wollen, geben Sie ein:

```
$ kill -19 785
```

Nehmen wir an, Sie wollen einen Prozess beenden, dessen Namen Sie kennen. Anstatt mit Hilfe von ps die Prozessnummer herauszufinden, können Sie das Kommando zum Beenden mit dem Namen des Prozesses versehen:

```
$ killall -9 mozilla-bin
```

Was Sie auch immer eingeben, Sie werden immer nur Ihre eigenen Prozesse beenden (es sei denn, Sie sind root). Also brauchen Sie sich auf einem Multiuser-System nicht um die Prozesse anderer Anwender zu sorgen, sie werden nicht angerührt.

6.3.2 ps und kill gemeinsam: top

top ist ein Programm, das die Funktionen von ps und kill beide erfüllt. Zusätzlich kann es noch zur Echtzeit-Überwachung von Prozessen benutzt werden, wobei es Informationen zur CPU- und Speicherauslastung, Laufzeit und mehr ausgibt (siehe Abbildung 6-1).

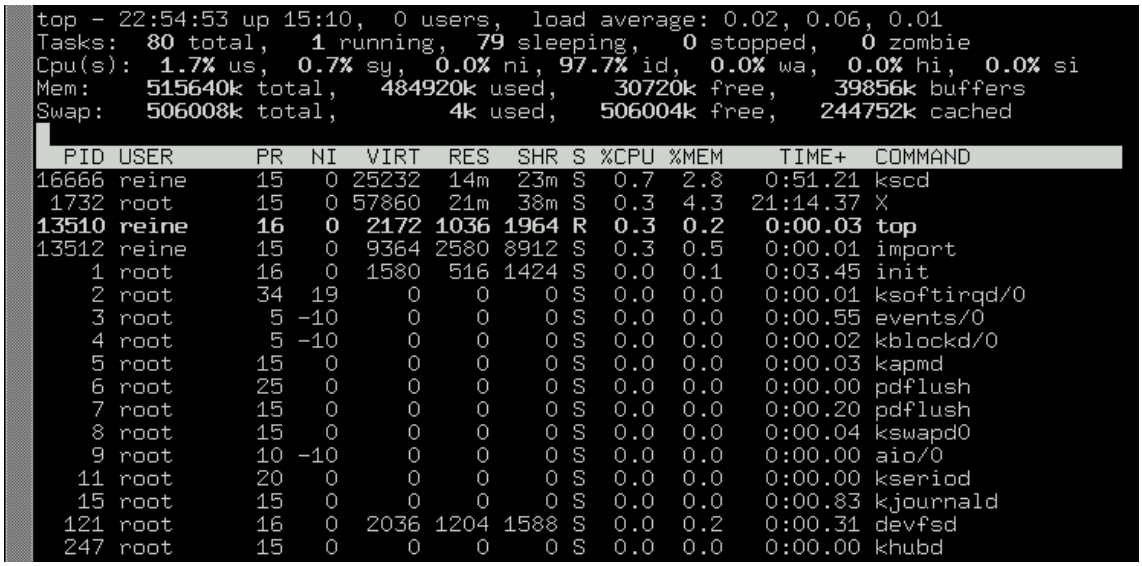


Abbildung 6-1: Prozessüberwachung mit top

Das Programm `top` wird vollständig mit der Tastatur bedient. Einen Hilfetext bekommen Sie mit der Taste **h** angezeigt. Die wichtigsten Befehle sind:

- **k**: mit diesem Kommando wird ein Signal zum Prozess geschickt. `top` wird dann nach der Prozess PID fragen, gefolgt von der Nummer oder dem Namen des zu sendenden Signals (standardmäßig `TERM` oder `15`);
- **M**: mit diesem Kommando sortieren Sie die Anzeige von `top` nach der Speicherbelegung (Feld `%MEM`);
- **P**: damit wird die Ausgabe nach der verbrauchten CPU-Zeit sortiert (Feld `%CPU`; dies ist die Standard-Sortiermethode);
- **u**: zeigt die Prozesse eines benannten Users an, `top` fragt in diesem Fall nach einem Usernamen. Sie müssen den **Namen** des Benutzers angeben, nicht die UID. Falls kein Name angegeben wird, werden alle Prozesse angezeigt;
- **i**: standardmäßig werden alle Prozesse, auch schlafende, angezeigt. Mit diesem Kommando beschränken Sie die Anzeige auf die zur Zeit laufenden Prozesse (Prozesse, deren Feld `STAT` ein `R` für *Running* anzeigen). Mit dem gleichen Kommando kehren Sie zur Standardanzeige (alle Prozesse) zurück.
- **r**: Mit diesem Kommando ändern Sie die Priorität ausgewählter Prozesse.

6.4 Prioritäten zuteilen: nice, renice

Jeder Prozess des Systems läuft mit definierter Priorität, auch „*nice value*“ genannt, die von -20 (höchste Priorität) bis 19 (niedrigste Priorität) reichen kann. Falls nicht explizit definiert, hat jeder Prozess die Standardpriorität 0 (die „Grund“-Priorität). Prozesse mit höherer Priorität (niedrigere „nice value“ bis zu -20) werden bei der Vergabe der CPU-Zyklen öfter berücksichtigt als andere Prozesse mit niedrigerer Priorität (höherer „nice value“). Dadurch bekommen Sie mehr Prozessor-Zyklen. Benutzer (ausser `root`) können ihre Prozesse nur innerhalb der Werte 0 bis 19 herabstufen. `root` kann die Priorität jedes beliebigen Prozesses auf jede beliebige Stufe setzen.

6.4.1 renice

Wenn ein oder mehrere Prozesse zuviele Systemressourcen beanspruchen, kann man ihre Priorität herabsetzen anstatt sie sofort zu beenden. Dazu wird das Programm `renice` benutzt. Die Syntax ist wie folgt:

```
renice priority [[-p] pid ...] [[-g] pgrp ...] [[-u] user ...]
```

Hier ist `priority` der Wert der Priorität, `pid` die Prozess-ID (benutzen Sie für mehrere Prozesse die Option `-p`), `pgrp` die Prozessgruppen-ID (benutzt mit der Option `-g`) und `user` (`-u` für mehr als einen) der Benutzername des Prozessbesitzers.

Nehmen wir an, Sie haben einen laufenden Prozess mit der PID 785, der eine lange wissenschaftliche Berechnung durchführt. Während dieser Zeit wollen Sie ein Spiel spielen, das freie Systemressourcen benötigt. In diesem Fall geben Sie ein:

```
$ renice +15 785
```

Dadurch wird zwar der Prozess wesentlich länger zur Durchführung der Berechnung benötigen, er belegt aber keine CPU-Zeit, die von anderen Prozessen benötigt wird.

Falls Sie der Systemadministrator sind und feststellen, dass ein Benutzer zu viele Prozesse laufen lässt, die zu viele Systemressourcen verbrauchen, können Sie die Priorität der Prozesse dieses Benutzers mit einem einzigen Befehl ändern:

```
# renice +20 -u birgit
```

Nach diesem Befehl haben alle Prozesse von birgit die niedrigste Priorität und werden Prozesse anderer Benutzer nicht behindern.

6.4.2 nice

Jetzt, da Sie wissen, dass Sie die Priorität von Prozessen ändern können, werden Sie nach einem Kommando fragen, mit dem Sie einen Befehl mit einer definierten Priorität starten können. Dafür benutzen Sie den Befehl `nice`.

In diesem Fall müssen Sie Ihren Befehl als Option von `nice` angeben. Die Option `-n` wird zum Setzen des Prioritätswertes benutzt. Standardmäßig setzt `nice` diesen Wert auf 10.

Sie wollen beispielsweise ein ISO-Image einer Mandrakelinux-Installations-CD-ROM anfertigen:

```
$ dd if=/dev/cdrom of=~/mdk1.iso
```

Auf manchen Systemen mit Standard-IDE-Geräten kann das Kopieren von großen Dateien zu viel Systemressourcen belegen. Um nun das Blockieren anderer Prozesse durch den Kopierprozess zu vermeiden, starten Sie ihn mit einer niedrigen Priorität:

```
$ nice -n 19 dd if=/dev/cdrom of=~/mdk1.iso
```

und fahren dann mit der normalen Arbeit fort.

Kapitel 7. Datei-Organisation

Heutzutage ist ein UNIX®-System groß – richtig groß! Das gilt besonders für GNU/Linux: die Menge der verfügbaren Software wäre ohne Regeln für das Dateisystem und den Verzeichnisbaum weder überschaubar noch verwaltbar.

Der anerkannte Standard ist der FHS (*Filesystem Hierarchy Standard*), dessen Version 2.3 im Januar 2004 verabschiedet wurde. Das entsprechende Dokument finden Sie in verschiedenen Formaten im Internet auf der The Pathname Website (`http://www.pathname.com/fhs/`). Dieses Kapitel bietet nur ein kurze Zusammenfassung der Regeln, sollte aber durchaus genügen, Ihnen aufzuzeigen, in welchem Verzeichnis Sie eine gesuchte Datei üblicherweise finden können oder wo Sie eine Datei dem Standard entsprechend ablegen sollten.

7.1 Einzeln / Gemeinsam genutzte, statische / variable Daten

Daten in einem UNIX®-System können nach den folgenden Kriterien klassifiziert werden: freigegebene Daten können allen Computern im Netzwerk zur Verfügung stehen und gemeinsam genutzt werden, im Gegensatz zu nicht freigegebenen, nur einzeln genutzten Daten. Statische Daten werden im normalen Betrieb nicht verändert, im Gegensatz zu variablen Daten. Während wir uns die Baumstruktur ansehen, werden wir die verschiedenen Verzeichnisse nach diesen Kategorien klassifizieren.

Bedenken Sie bitte, dass diese Einteilungen nur eine Empfehlung sind. Man muss der Empfehlung zwar nicht folgen, diese Regeln können Ihnen aber bei der Verwaltung Ihres Systems eine große Hilfe sein. Die Unterscheidung in statische/variable Daten betrifft nur den normalen Betrieb des Systems, nicht die Konfiguration. Bei jeder Installation eines Programmes werden Sie unweigerlich „normalerweise“ statische Verzeichnisse (etwa `/usr`) modifizieren müssen.

7.2 Die Verzeichnisbaumwurzel: /

Die Verzeichnisbaumwurzel enthält die gesamte System-Hierarchie. Sie kann nicht klassifiziert werden, da ihre Unterverzeichnisse sowohl statisch und einzeln genutzt als auch das Gegenteil sein können. Es folgt eine Liste der üblichen Verzeichnisse und Unterverzeichnisse mit ihrer jeweiligen Klassifizierung:

- `/bin`: wichtige Binärdateien. Dieses Verzeichnis enthält die grundlegenden Befehle, die von allen Anwendern des Systems benutzt werden und für den Betrieb des Systems nötig sind: `ls`, `cp`, `login`, usw. Statisch, Einzelnutzung.
- `/boot`: enthält die für den GNU/Linux Betriebssystemstarter (GRUB oder LILO für **Intel**-Systeme, yaboot für PPC, usw.) notwendigen Dateien. Es kann den Kern enthalten, muss aber nicht. Wenn der Kern nicht hier liegt, muss er in der Verzeichnisbaumwurzel liegen. Statisch, Einzelnutzung.
- `/dev`: System-Geräte-Dateien (`dev` für engl. *DEVices*). Einige Dateien in `/dev` sind vorgeschrieben, etwa: `/dev/null`, `/dev/zero` und `/dev/tty`. Statisch, Einzelnutzung.
- `/etc`: alle Konfigurationsdateien für diesen Rechner. Dieses Verzeichnis darf keine Binärdateien enthalten. Statisch, Einzelnutzung.
- `/home`: enthält alle persönlichen Verzeichnisse der auf dem System eingerichteten Benutzerkonten. Dieses Verzeichnis kann zur gemeinsamen Nutzung freigegeben werden, muss aber nicht. In manchen großen Netzwerken wird es via NFS freigegeben. In diesem Verzeichnis sind auch die Konfigurationsdateien Ihrer meist genutzten Anwendungen (wie etwa Browser, Mail, etc.) untergebracht. Diese Dateien beginnen alle mit einem Punkt („.“). So liegen die Konfigurationsdateien des Mozilla etwa im Unterverzeichnis `.mozilla`. Variabel, gemeinsame Nutzung möglich.
- `/lib`: dieses Verzeichnis enthält die für das System wichtigen Bibliotheken und die Kernmodule, die im Unterverzeichnis `/lib/modules/KERN_VERSION/` untergebracht sind. Alle Bibliotheken für die Programme in den Verzeichnissen `/bin` und `/sbin` müssen hier gespeichert werden, gemeinsam mit dem optionalen Laufzeit-Linker `ld*` und der dynamisch gelinkten C-Bibliothek `libc.so`. Statisch, Einzelnutzung.

- /mnt: das Verzeichnis, in dem die Einhängepunkte für temporär eingehängte Dateisysteme wie etwa /mnt/cdrom, /mnt/floppy, usw. untergebracht sind. Im Verzeichnis /mnt werden auch temporäre Verzeichnisse (/mnt/removable für eine USB-Karte) eingehängt. Variabel, Einzelnutzung.
- /opt: hier werden Programme installiert, die nicht wesentlich für den Betrieb des Systems sind. Es wird empfohlen, die statischen Dateien (Programmdateien, Bibliotheken, Dokumente, etc.) im Unterverzeichnis /opt/paket_name und die speziellen Konfigurationsdateien der Programme im Unterverzeichnis /etc/opt zu installieren.
- /root: persönliches Verzeichnis des Superusers root. Variabel, Einzelnutzung.
- /sbin: enthält für den Systemstart essentielle Programmdateien. Die meisten dieser Programme können nur von root ausgeführt werden. Manche der Programme können von normalen Anwendern ausgeführt werden, bieten dann aber nicht alle möglichen Optionen. Statisch, Einzelnutzung.
- /tmp: das Verzeichnis für temporäre Dateien, die von manchen Programmen während des Betriebs erstellt werden. Variabel, Einzelnutzung.
- /usr: nähere Angaben im Kapitel */usr: Das Große Verzeichnis*, Seite 56. Statisch, gemeinsame Nutzung möglich.
- /var: enthält Dateien, die von verschiedenen Programmen (etwa: E-Mail-Server, Auditprogramme, Printserver, usw.) während der Laufzeit verändert werden. Variabel. Die einzelnen Unterverzeichnisse können sowohl zur Einzelnutzung als auch zur gemeinsamen Nutzung bestimmt werden.

7.3 /usr: Das Große Verzeichnis

Das Verzeichnis /usr ist das Hauptverzeichnis zur Ablage von Anwendungen. Da die Binärdateien dieses Verzeichnisses nicht zum Systemstart benötigt werden, kann (und wird) die Hierarchie /usr oft auf einem separaten Dateisystem (d.h., Partition) untergebracht. Aufgrund seiner Größe besitzt /usr gewöhnlich eine eigene Hierarchie von Unterverzeichnissen, von denen wir hier einige aufzählen:

- /usr/X11R6: die gesamte X Window System-Hierarchie. Hier müssen alle Programmdateien und Bibliotheken zum Betrieb von X (inklusive X-Server) abgelegt werden. Das Verzeichnis /usr/X11R6/lib/X11 wiederum enthält alle für die Konfiguration von X notwendigen und für alle Computer geltenden Dateien. Spezielle, von Modell zu Modell variierende Dateien finden Sie in /etc/X11.
- /usr/bin: enthält die große Mehrzahl der Programmdateien. Hier sollten **alle** Programmdateien, die weder zur Systemverwaltung noch Systemreparatur dienen, zu finden sein. Ausgenommen sind nur die Programme, die Sie selbst kompilieren und installieren und daher in /usr/local ablegen.
- /usr/lib: enthält alle Bibliotheken, die zum Betrieb der Programme in /usr/bin und /usr/sbin notwendig sind. Hier gibt es auch einen Symlink /usr/lib/X11, der auf das Verzeichnis /usr/X11R6/lib zeigt, in dem die X Window System-Bibliotheken untergebracht sind (natürlich nur, wenn X installiert wurde)[1].
- /usr/local: hier installieren Sie Programme, die Sie selbst aus Quellcode erstellt haben. Das Installationsprogramm sollte die notwendige Hierarchie automatisch anlegen.
- /usr/share: dieses Verzeichnis enthält alle unveränderbaren architektur-unabhängigen Daten der Programme im Verzeichnis /usr. Unter Anderem finden Sie hier auch die Zeitzonen- und Landesinformationen (zoneinfo und locale).

Zu erwähnen sind natürlich auch die Verzeichnisse /usr/share/doc und /usr/share/man, die die Dokumentationen und „Handbuchauszüge“ (engl. *Man-Pages*) enthalten.

7.4 /var: Zur Laufzeit veränderliche Dateien

Das Verzeichnis /var enthält alle operativen Daten der laufenden Programme. Im Gegensatz zu den temporären Dateien in /tmp müssen diese Daten immer konsistent sein, falls es zu einem unvorhergesehenen Rechnerneustart kommt. Hier folgt die Beschreibung einiger der vielen Unterverzeichnisse:

1. Bitte beachten Sie, dass Mandrakelinux jetzt Xorg anstelle von X Window System als Standard X Window-System benutzt.

- `/var/log`: enthält die Logdateien des Systems, die zur Problembehebung sehr wichtig sind (`/var/log/messages` und `/var/log/kernel/errors`, um nur zwei davon zu nennen).
- `/var/run`: hier werden alle seit dem Systemstart laufenden Prozesse beobachtet, um bei einem Wechsel im System entsprechend reagieren zu können ***Runlevel*** (siehe *Die Startdateien: sysv initialisieren*, Seite 81).
- `/var/spool`: enthält die Arbeitsdateien der Systemdienste, die auf eine Verarbeitung oder eine Aktion warten. Beispielsweise finden Sie in `/var/spool/cups` die Arbeitsdateien des Drucker-Servers CUPS während `/var/spool/mail` die Arbeitsdateien des Mailservers beinhaltet (d.h., die ein- und ausgehende E-Mails).

7.5 /etc: Konfigurationsdateien

`/etc` ist eines der unter UNIX® wichtigsten Verzeichnisse, da es alle grundlegenden Konfigurationsdaten enthält. Dieses Verzeichnis dürfen Sie **in keinem Fall** wegen Platzersparnis löschen! Desgleichen dürfen Sie, wenn Sie Ihr System über mehrere Partitionen verteilen, das Verzeichnis `/etc` nie auf eine gesonderte Partition legen: es wird für die Initialisierung des Systems benötigt und muss zum Systemstart im Wurzelverzeichnis des Systems liegen.

Hier einige wichtige Dateien:

- `passwd` und `shadow`: diese Textdateien enthalten alle Benutzerkennzeichen und ihre verschlüsselten Passwörter. Sie finden die Datei `shadow` natürlich nur dann vor, wenn Shadow-Passwörter benutzt werden (Standard-Option aus Sicherheitsgründen).
- `inittab`: die Konfigurationsdatei für `init`, welches eine fundamentale Rolle beim Start des Systems spielt.
- `services`: eine Auflistung existierender Netzwerk-Dienste.
- `profile`: die Konfigurationsdatei der shell. Manche shells benutzen andere Dateien, beispielsweise `.bashrc` für die bash.
- `crontab`: die Konfigurationsdatei für `cron`, das Programm zur periodischen Ausführung von Befehlen.

Für manche Programme, die eine größere Anzahl von Konfigurationsdateien benötigen, existieren entsprechende Unterverzeichnisse. Dies gilt etwa für X Window System, das alle notwendigen Dateien im Verzeichnis `/etc/X11` speichert.

Kapitel 8. Dateisysteme und Einhängpunkte

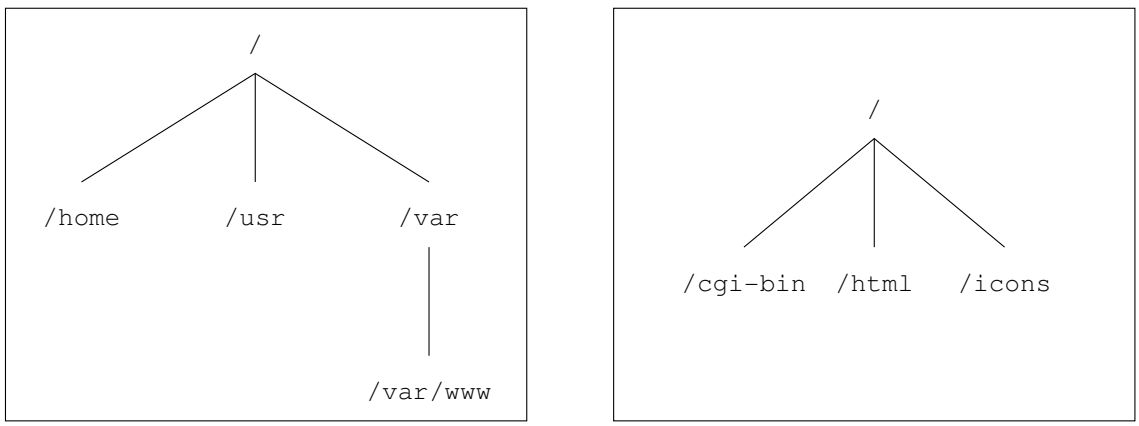
Am leichtesten versteht man die Art, „wie es funktioniert“, indem man sich einen praktischen Fall anschaut. Und das ist genau das, was wir hier machen werden. Nehmen wir an, Sie haben sich gerade eine brandneue Festplatte gekauft, die noch keinerlei Partitionen enthält. Ihre Mandrakelinux-Partition auf der alten Festplatte ist nahezu voll und Sie haben sich dafür entschieden, keine neue Installation durchzuführen sondern einen kompletten Bereich des vorhandenen Verzeichnisbaums auf die neue Festplatte zu übertragen. Da die neue Festplatte eine recht hohe Kapazität aufweist, werden Sie das umfangreichste Verzeichnis darauf unterbringen: `/usr`. Aber sehen wir uns zuerst ein wenig in der Theorie um.

8.1 Grundlagen

Jede Festplatte ist in mehrere Partitionen unterteilt und jede davon enthält ein Dateisystem. Während Windows® jedem dieser Dateisysteme (aber nur den Systemen, die es erkennt!) einen Buchstaben zuteilt, besitzt GNU/Linux eine einzige Baumstruktur, in die jedes Dateisystem an einem Punkt ***eingehängt*** wird.

So ähnlich wie Windows® ein „Laufwerk C:“ benötigt, muss GNU/Linux das Wurzelverzeichnis (`/`) dieser Baumstruktur auf der Partition finden, die das ***Root Dateisystem*** enthält. Wenn einmal dieses System eingehängt ist, kann man weitere Dateisysteme in diese Baumstruktur an beliebigen ***Einhängepunkten*** einhängen. Jedes Verzeichnis unterhalb des Wurzelverzeichnisses kann als Einhängepunkt dienen und Sie können sogar das gleiche Dateisystem mehrere Male an verschiedenen Punkten einhängen.

Dadurch erhalten Sie eine große Flexibilität in Ihrer Konfiguration. Wenn Sie beispielsweise einen Webserver aufsetzen, können Sie dem Verzeichnis mit den Webserver-Daten eine ganze Partition zuteilen. Das Verzeichnis, das normalerweise diese Daten enthält, also `/var/www`, dient in diesem Fall als Einhängepunkt für diese Partition. Wenn Sie eine große Menge an Daten aus dem Internet herunterladen wollen ist etwa eine eigene große Partition für das Verzeichnis `/home` von Vorteil. Sie sehen in Abbildung 8-1 und Abbildung 8-2 wie das System vor und nach dem Einhängen des Dateisystems aussieht.



Root Dateisystem
(bereits eingehängt)

Dateisystem mit Dateien
des Verzeichnisses "/var/www"
(noch nicht eingehängt)

Abbildung 8-1: Vor dem Einhängen des Dateisystems

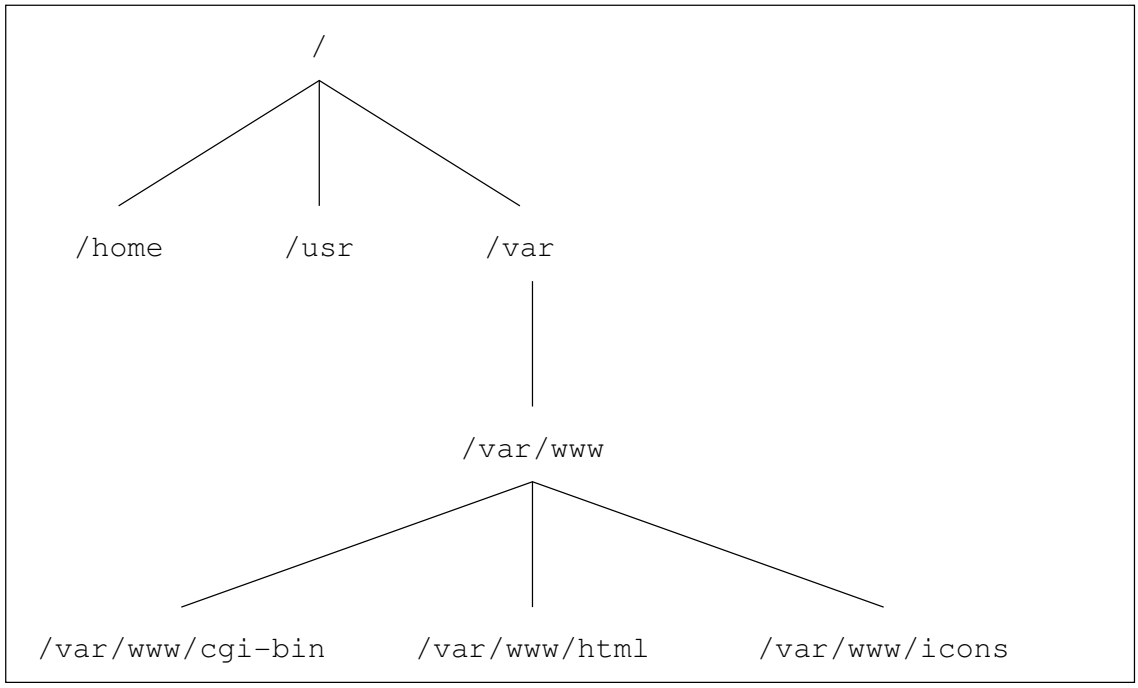


Abbildung 8-2: Jetzt ist das Dateisystem eingehängt

Das bietet natürlich eine Menge Vorteile: die Baumstruktur bleibt immer erhalten, egal, ob sie sich über ein Dateisystem oder über ein Dutzend Dateisysteme erstreckt. Diese Flexibilität erlaubt es Ihnen auch, einen wesentlichen Teil des Verzeichnisbaums bei Bedarf auf eine andere Partition zu verschieben und das ist genau das, was wir hier machen werden.

Zwei Dinge sollten Sie über Einhängepunkte wissen:

1. Das Verzeichnis, das als Einhängepunkt dienen soll, muss bereits vorhanden sein.
2. Und dieses Verzeichnis **sollte besser leer sein**: wenn ein Verzeichnis als Einhängepunkt bestimmt wird, das bereits Dateien und Unterverzeichnisse enthält, so werden diese Dateien und Unterverzeichnisse von dem neu eingehängten Dateisystem „verdeckt“. Die Dateien und Unterverzeichnisse werden nicht gelöscht, sie sind nur solange nicht erreichbar bis das eingehängte Dateisystem wieder ausgehängt wird.

Tatsächlich gibt es eine Möglichkeit, auf die von einem neu eingehängten Dateisystem „verdeckten“ Dateien zuzugreifen. Dazu muss man das neue Dateisystem mit der Option `--bind` einhängen. Wenn Sie beispielsweise gerade ein Dateisystem in `/hidden/directory/` eingehängt haben und auf den ursprünglichen Inhalt unter `/new/directory` zugreifen wollen, sollten Sie das Folgende eingeben:

```
mount --bind /hidden/directory/ /new/directory
```

8.2 Partitionierung einer Festplatte und Formatieren einer Partition

Beim Durcharbeiten dieses Kapitels sollten Sie zwei Dinge beachten: eine Festplatte wird in Partitionen eingeteilt und jede dieser Partitionen enthält ein Dateisystem. Auf Ihrer brandneuen Festplatte gibt es bisher nichts davon, also werden wir mit der Partitionierung beginnen. Für diese Aufgabe müssen Sie als `root` angemeldet sein.

Als Erstes müssen Sie den „Namen“ Ihrer Festplatte herausfinden (d.h., durch welche Gerätedatei sie repräsentiert wird). Angenommen, die neue Festplatte ist die zweite Platte (Slave) am ersten IDE-Port, so wird sie als `/dev/hdb` bezeichnet.[1] Bitte lesen Sie im Kapitel *Ihre Partitionen verwalten* des *Starter Handbuch*, wie eine

1. Die Bezeichnungsregeln für Festplatten finden Sie im *Installationshandbuch*.

Festplatte partitioniert wird. DiskDrake legt auch die Dateisysteme an, sodass wir nach der Partitionierung und dem Anlegen der Dateisysteme gleich fortfahren können.

8.3 Die Befehle zum Ein- und Aushängen

Nach dem Anlegen eines Dateisystems kann die Partition eingehängt werden. Am Anfang ist sie natürlich leer, da das System noch keinen Zugriff auf die Partition hatte. Der Befehl zum Einhängen von Dateisystemen lautet `mount` und die Syntax ist wie folgt:

```
mount [optionen] <-t type> [-o mount optionen] <gerät> <einhängpunkt>
```

In unserem Fall wollen wir unsere neue Partition nur temporär im Verzeichnis `/mnt` einhängen (oder in jeden beliebigen anderen Einhängpunkt – denken Sie daran, dass dieser Einhängpunkt bereits vorhanden sein muss). Der entsprechende Befehl lautet also:

```
$ mount -t ext3 /dev/hdb1 /mnt
```

Die Option `-t` wird benutzt um den Typ des Dateisystems der einzuhängenden Partition zu spezifizieren. Die meist benutzten Dateisysteme sind ext2FS (das GNU/Linux Dateisystem) bzw. ext3FS (eine verbesserte Version von ext2FS mit Journal-Funktionalität), VFAT (für alle DOS/Windows® Partitionen: FAT 12, 16 oder 32) und ISO9660 (CD-ROM Dateisystem). Wenn Sie keinen Typ angeben, wird `mount` versuchen, den richtigen Typ aus dem Superblock der Partition herauszulesen.

Die Option `-o` wird benutzt um eine oder mehrere Einhängoptionen festzulegen. Die jeweils möglichen Optionen ergeben sich aus dem benutzten Dateisystem. Lesen Sie dazu die Handbuchauszüge mount(8).

Nachdem Sie nun Ihre neue Partition eingehängt haben wird es Zeit, das gesamte Verzeichnis `/usr` dorthin zu kopieren:

```
$ (cd /usr && tar cf - .) | (cd /mnt && tar xpvf -)
```

Wenn der Kopiervorgang abgeschlossen ist, kann die Partition wieder ausgehängt werden. Dazu benutzen wir den Befehl `umount`, dessen Syntax recht einfach ist:

```
umount <einhängpunkt|gerät>
```

Also lautet der Befehl in unserem Beispiel:

```
$ umount /mnt
```

oder:

```
$ umount /dev/hdb1
```

Manchmal kann es vorkommen, dass das auszuhängende Gerät (meist das CD-ROM) besetzt ist. In diesem Fall lösen die meisten Benutzer das Problem mit einem Reboot. Wenn beispielsweise der Befehl `umount /dev/hdc` versagt können Sie die Option „*lazy*“ des Befehls `umount` versuchen. Auch dessen Syntax ist einfach:

```
umount -l  <einhängpunkt|gerät>
```

Dieser Befehl hängt das Gerät aus und schließt, wenn möglich, alle Verbindungen zum Gerät. Normalerweise kann man die CD mit dem Kommando `eject <einhängpunkt|gerät>` auswerfen. Wenn also der Befehl `eject` keine Wirkung zeigt und Sie einen Neustart vermeiden wollen, versuchen Sie es mit der Option „*lazy*“.

Da diese Partition unser Verzeichnis `/usr` aufnehmen soll, müssen wir das dem System mitteilen. Dazu editieren wir die Datei `/etc/fstab`. Mit Hilfe dieser Datei wird das automatische Einhängen von bestimmten

Dateisystemen gesteuert, insbesondere während des Systemstarts. Sie besteht aus einer Anzahl von Zeilen mit den Beschreibungen der Dateisysteme, der Einhängepunkte und anderer Optionen. Ein Beispiel einer gebräuchlichen `/etc/fstab`:

```
/dev/hda1   /           ext2    defaults        1 1
/dev/hda5   /home       ext2    defaults        1 2
/dev/hda6   swap        swap    defaults        0 0
none        /mnt/cdrom  supermount dev=/dev/scd0,fs=udf:iso9660,ro,-- 0 0
none        /mnt/floppy supermount dev=/dev/fd0,fs=ext2:vfat,--,sync,umask=0 0 0
none        /proc       proc    defaults        0 0
none        /dev/pts    devpts  mode=0622       0 0
```

Jede Zeile enthält:

- das Gerät (die Partition), das das Dateisystem enthält;
- den Einhängepunkt;
- den Typ des Dateisystems;
- die Einhängeoptionen;
- das passende ***Flag*** (Markierung) für das Backup-Programm `dump`;
- die Kriterien für die Reihenfolge für `fsck` (engl. *FileSystem ChecK*).

Eine Zeile, die Sie **immer** in dieser Datei finden werden, ist die Zeile für die Verzeichnisbaumwurzel. Die swap Partitionen sind etwas spezieller Natur, da sie innerhalb der Verzeichnisstruktur nicht aufgeführt werden. Der Einhängepunkt für diese Partition enthält das Schlüsselwort `swap`. Das hier auch aufgeführte Dateisystem `/proc` wird im Kapitel *Das Dateisystem /proc*, Seite 75 näher beschrieben. Ein weiteres besonderes Dateisystem ist `/dev/pts`.

In unserem Beispiel haben wir die gesamte Hierarchie `/usr` auf die Partition `/dev/hdb1` verschoben und diese Partition soll nun bei Neustart des Rechners automatisch als `/usr` eingehängt werden. Dazu fügen wir den folgenden Eintrag in die Datei `/etc/fstab` ein:

```
/dev/hdb1       /usr        ext2    defaults  1 2
```

Jetzt wird die Partition bei jedem Systemstart eingehängt und (falls nötig) auf Fehler geprüft.

Es gibt zwei besondere Optionen: `noauto` und `user`. Die Option `noauto` bewirkt, dass das Dateisystem nicht automatisch während des Systemstarts eingehängt wird sondern nur dann, wenn Sie es wünschen. Die Option `user` gibt an, dass das Dateisystem von jedem normalen Benutzer ein- und ausgehängt werden kann. Diese beiden Optionen werden üblicherweise bei CD-ROM- und Disketten-Laufwerken benutzt. Es gibt noch einige andere Optionen, die Sie im Handbuchauszug (fstab(5)) nachlesen können.

Ein Vorteil der Benutzung der Datei `/etc/fstab` ist der leichtere Umgang mit der Befehlssyntax des Befehls `mount`. Um ein in der Datei bereits beschriebenes Dateisystem oder Gerät einzuhängen reicht der Bezug auf den Einhängepunkt oder das Gerät. Um beispielsweise ein Diskettenlaufwerk einzuhängen, reicht der Befehl:

```
$ mount /mnt/floppy
```

oder:

```
$ mount /dev/fd0
```

Zum Schluss unseres „Partitions-Umzuges“ lassen Sie uns rekapitulieren, was wir bisher getan haben. Wir haben die Hierarchie `/usr` kopiert und die Datei `/etc/fstab` dahingehend geändert, dass die neue Partition beim Systemstart automatisch eingehängt wird. Jedoch befinden sich im Moment noch die alten Dateien und Verzeichnisse von `/usr` auf ihrem alten Platz und müssen gelöscht werden. Schließlich war unser ursprüngliches Ziel die Schaffung von mehr freiem Platz auf der alten Festplatte. Dazu müssen Sie zuerst mit dem Befehl `telinit 1` auf der Kommandozeile in den Einzelbenutzer-Modus (Runlevel 1) wechseln.

- Als Nächstes löschen Sie alle Dateien und Verzeichnisse im Verzeichnis `/usr`. Hier ist immer noch das „alte“ Verzeichnis gemeint, da das neue, größere, noch nicht eingehängt ist. Geben Sie also ein: `rm -Rf /usr/*`.
- Schließlich und endlich hängen wir mit dem Befehl `mount /usr/` das neue Verzeichnis `/usr` in den Verzeichnisbaum ein.

Das wars. Jetzt wechseln Sie zurück in den Mehrbenutzer-Modus (mit `telinit 3` in den Textmodus oder `telinit 5` in die grafische Umgebung) und falls Sie nicht noch weitere administrative Aufgaben erledigen wollen, sollten Sie auch den `root` Account wieder verlassen.

Kapitel 9. GNU/Linux Dateisysteme

Ihr GNU/Linux-System befindet sich natürlich innerhalb eines Dateisystems auf Ihrer Festplatte. In diesem Kapitel beschäftigen wir uns mit den verschiedenen Dateisystemen von GNU/Linux und deren Optionen.

9.1 Vergleich einiger Dateisysteme

Während der Installation können Sie für Ihre Partitionen verschiedene **Dateisysteme** auswählen mit deren Algorithmen diese Partitionen dann formatiert werden.

Wenn Sie nicht gerade ein Spezialist sind, ist die Auswahl des richtigen Dateisystems nicht gerade selbstverständlich. Lassen Sie uns einen kurzen Blick auf verschiedene aktuelle Dateisysteme werfen, die alle in Mandrakelinux enthalten sind.

9.1.1 Verschiedene gebräuchliche Systeme

9.1.1.1 Ext2

Das **Second Extended Filesystem** (abgekürzt ext2 oder einfach `ext2`) war einige Jahre lang das Standard-Dateisystem unter GNU/Linux. Es ersetzte das Extended File System (daher der Namensteil „Second“). Das „neue“ Dateisystem korrigierte einige Probleme und Limitierungen des Vorgängers.

Ext2 hält sich an die üblichen Standards für Unix-Dateisysteme. Seit seiner Einführung hat es sich natürlich weiter entwickelt aber dabei seine Stabilität und gute Leistung beibehalten.

9.1.1.2 Ext3

Wie es schon der Name ahnen lässt ist das **Third Extended File System** der Nachfolger von Ext2. Es ist kompatibel zu seinem Vorgänger, enthält aber eine neue Eigenschaft: **das Journal**.

Eine der größten Schwächen der „traditionellen“ Dateisysteme wie Ext2 ist ihre Empfindlichkeit bei plötzlichen Systemausfällen (Stromunterbrechung, Software-Absturz, usw.). Im Allgemeinen folgt dann nach einem Neustart eine recht ausgedehnte Prüfung der Systemstruktur und Versuche der Fehlerkorrektur, die meistens weitere Schäden des Dateisystems verursachen. Dabei entsteht nicht selten ein teilweiser oder sogar vollständiger Datenverlust.

Ein ständiges Journal ist die Lösung dieses Problems. Vereinfacht dargestellt, wird eine Aktion (wie etwa das Speichern einer Datei) gespeichert **bevor** die Aktion tatsächlich ausgeführt wird. Man kann das mit der Tätigkeit eines Kapitäns vergleichen, der ein Logbuch zur Aufzeichnung der täglichen Ereignisse führt. Das Ergebnis: ein ständig kohärentes Dateisystem. Bei eventuellen Problemen geschieht eine Prüfung und Reparatur sehr schnell und sicher. Dadurch hängt die Dauer der Prüfung eines Dateisystems nicht mehr von der Größe sondern von der tatsächlichen Nutzungsart ab.

Ext3 bietet also die Journal-Technologie zusätzlich zur robusten Struktur von Ext2 und ist somit voll kompatibel zum Vorgänger-System. Daher ist ein Umstieg von Ext2 auf Ext3 oder zurück kein Problem und lässt sich leicht durchführen.

9.1.1.3 ReiserFS

ReiserFS wurde dagegen komplett neu entwickelt. Es ist wie Ext3 auch ein journalisierendes Dateisystem aber die interne Struktur ist vollkommen anders, da es zur Journalisierung ein aus dem Datenbank-Bereich stammendes Konzept benutzt.

9.1.1.4 JFS

JFS ist ein journalisierendes Dateisystem, das von IBM entwickelt und benutzt wird. Es war zu Anfang proprietäre Software bis sich IBM dazu entschloss, es der Freien Software Bewegung zur Verfügung zu stellen. Die interne Struktur ähnelt der des ReiserFS.

9.1.1.5 XFS

XFS ist das von SGI entwickelte und in IRIX benutzte journalisierende Dateisystem. Auch dieses Dateisystem war zu Anfang proprietäre Software und wurde ebenfalls später von SGI freigegeben. Die interne Struktur hat viele besondere Eigenschaften, wie etwa Unterstützung für Echtzeit-Übertragung und Cluster-Dateisysteme (allerdings nicht in der freien Version).

9.1.2 Unterschiede zwischen den Dateisystemen

	Ext2	Ext3	ReiserFS	JFS	XFS
Stabilität	Exzellent	Gut	Gut	Medium	Gut
Wiederherstellung gelöschter Dateien	Ja (komplex)	Ja (komplex)	Nein	Nein	Nein
Reboot-Zeit nach Crash	Lang, sogar sehr lang	Schnell	Sehr schnell	Sehr schnell	Sehr schnell
Status der Daten nach einem Crash	Allgemein gut, aber hohes Risiko von teilweise oder komplettem Verlust	Sehr gut	Medium. a	Sehr gut	Sehr gut
ACL-Unterstützung	Ja	Ja	Nein	Nein	Ja
Bemerkungen: a. Durch das Einfügen der Option *data=journal* in die /etc/fstab kann man bessere Resultate bei der Wiederherstellung nach einem Systemcrash erreichen. Dadurch werden nicht nur die **Metadaten** sondern auch die **Daten** im Journal erfasst.					

Tabelle 9-1. Eigenschaften der Dateisysteme

Die maximale Dateigröße hängt von einer Reihe von Parametern ab (etwa Block-Größe bei ext2/ext3) und erhöht sich gewöhnlich abhängig von der Kernversion und Architektur. Die Begrenzung durch das Dateisystem liegt zur Zeit nahe oder etwas über 2 TeraByte (TB, 1 TB=1024 GB) und kann bei JFS bis zu 4 PetaByte (PB, 1 Pb=1024 TB) betragen. Leider unterliegen diese Werte auch den Grenzen der maximalen Größe eines Block-Gerätes, die beim aktuellen Kern `2.4.X` (nur für `X86`-Systeme) bei 2TB liegt[1], sogar im RAID-Modus. In der 2.6-er Kernreihe kann diese Grenze durch die Kompilierung der Unterstützung von Large Block Device (CONFIG_LBD=y) überschritten werden. Weitere Informationen erhalten Sie auf den Seiten Adding Support for Arbitrary File Sizes to the Single UNIX Specification (`http://ftp.sas.com/standards/large.file/x_open.20Mar96.html`), Large File Support in Linux (`http://www.suse.com/~aj/linux_lfs.html`) und Large Block Devices (`http://www.gelato.unsw.edu.au/IA64wiki/LargeBlockDevices`).

9.1.3 Und die Leistung?

Ein Leistungsvergleich zwischen Dateisystemen ist immer recht schwierig. Die Tests haben ihre Schwachstellen und die Ergenisse sollten vorsichtig interpretiert werden. Zur Zeit ist ext2 ziemlich ausgereift, entwickelt sich aber nur langsam weiter. Auch ext3 und `reiserfs` haben heute ihre Kinderkrankheiten hinter sich. `reiserfs` bietet in ReiserFS4 einige neue Eigenschaften.[2]. XFS dagegen hat eine Menge Features und mit der Zeit funktionieren die erweiterten Features immer besser unter Linux. JFS geht etwas anders an die Sache heran,

1. Wie kann man solche Werte mit Festplatten erreichen, die gerade einmal 320-400GB an Kapazität bieten? Nehmen Sie beispielsweise einen RAID-Kontroller, der 8 Festplatten zu je 250GB per RAID-Striping ansteuert, so bekommen Sie eine Gesamtleistung von 2TB. Die Kombination mehrerer RAID-Kontroller mit Linux Software-RAID oder LVM ermöglicht sogar das Überschreiten der 2TB-Grenze.
2. Zum Zeitpunkt der Entstehung dieses Handbuchs ist `reiserfs4` moch nicht in 2.6-er Kernen enthalten

hier werden einzelne Features nach und nach in Linux integriert. Das verlangsamt zwar den gesamten Prozess, ergibt aber einen jeweils sauberen Code. Von diesen Entwicklungen her sind bereits Vergleiche veraltet, die nur ein paar Monate oder Wochen zurück liegen. Auch die modernen Geräte (denken Sie an heutige Festplatten-Kapazitäten) machen einen großen Unterschied aus. Zur Zeit kann man unter XFS mit großen Dateien die beste Performance beobachten.

Jedes System hat seine vor- und Nachteile. Es kommt eigentlich darauf an, wozu und wie Sie Ihren PC benutzen. Für einen einfachen Desktop mögen Sie mit Ext2 zufrieden sein während bei einem Server ein journalisierendes Dateisystem wie Ext3 vorzuziehen ist. ReiserFS eignet sich hingegen wegen seiner Entwicklungsgeschichte am besten für Datenbank-Server und JFS ist da die beste Wahl, wo es auf die Geschwindigkeit des Datendurchsatzes ankommt. XFS schließlich ist für diejenigen interessant, die die erweiterten Features benötigen, die dieses System bietet.

Für den „normalen“ Gebrauch zeigen alle 4 Dateisysteme annähernd gleiche Leistungen. ReiserFS ist zwar bei sehr kleinen Dateien schneller, verliert aber bei der Arbeit mit größeren Dateien mit vielen MegaBytes. In den meisten Fällen wird die Leistung der Journal-Fähigkeiten des ReiserFS die Nachteile aufwiegen. Beachten Sie bitte, dass das ReiserFS normalerweise ohne die Option `notail` benutzt wird. Diese Option ist eine Optimierung für den Umgang mit kleinen Dateien. Ohne diese Option werden auch große Dateien mit „normaler“ Geschwindigkeit verarbeitet.

9.2 Alles ist eine Datei!

Im Starter-Handbuch haben Sie die Besitz- und Zugriffsrechte von Dateien kennengelernt, aber das wirkliche Verständnis des UNIX® ***Dateisystem*** (und das gilt auch für das Linux Dateisystem) erfordert eine neue Beantwortung der Frage „Was ist eine Datei?“.

In diesem Fall bedeutet „alles“ **wirklich** alles. Eine Festplatte, eine Partition darauf, ein Parallelport, eine Internetverbindung, eine Ethernet-Karte: all das sind Dateien. Sogar Verzeichnisse sind Dateien. Linux kennt ausser den Standarddateien und Verzeichnissen noch eine ganze Menge anderer Dateitypen. Beachten Sie bitte, dass wir hier mit Dateityp nicht den Dateityp meinen, der durch den **Inhalt** einer Datei festgelegt wird. GNU/Linux und jedes andere UNIX®-System betrachtet eine Datei, egal ob es ein Bild, eine Programmdatei oder was auch immer ist, als Ansammlung von Bytes, deren Verwendung und Typ allein durch die entsprechende Anwendung bestimmt wird.

9.2.1 Die Dateitypen

Sie werden sich erinnern: beim Resultat des Befehls `ls -l` bezeichnet das erste Zeichen vor den Zugriffsrechten den Dateityp. Sie kennen bereits zwei verschiedene Typen: normale Dateien (`-`) und Verzeichnisse (`d`). Beim Durchsuchen des Verzeichnisbaums und der Verzeichnisinhalte werden Sie noch weitere Dateitypen finden:

1. ***Zeichenmodus-Dateien***: diese Dateien sind zeichen-orientiert. Es handelt sich entweder um spezielle Systemdateien (wie etwa `/dev/null`, die wir schon behandelt haben) oder periphere Anschlüsse (seriellle oder parallele Ports). Allen gemeinsam ist, dass ihr Inhalt (falls vorhanden) nicht ***gepuffert*** wird (d.h., er wird nicht im Arbeitsspeicher gehalten). Diese Dateien werden identifiziert durch den Buchstaben `c`.
2. ***Blockmodus-Dateien***: diese Dateien sind Geräte und anders als bei den Zeichenmodus-Dateien wird ihr Inhalt im Speicher **gepuffert**. Übliche Geräte dieser Art sind: Festplatten, Partitionen auf Festplatten, Disketten- und CD-ROM-Laufwerke, usw. Dateien mit Bezeichnungen wie `/dev/hda` oder `/dev/sda5` sind block-orientierte Dateien und werden in der Ausgabe von `ls -l` mit dem Buchstaben `b` gekennzeichnet.
3. ***Symbolische Links***: diese Dateien findet man oft, sie werden gerade beim Start-Prozess eines Mandrakelinux-Systems häufig verwendet (siehe Kapitel *Die Startdateien: sysv initialisieren*, Seite 81). Wie der Name schon andeutet, ist der Zweck dieser Dateien die symbolische Verknüpfung von Dateien. D.h., ihr Inhalt ist nur der Verweis auf den Pfad einer anderen Datei. Das muss nicht unbedingt eine existierende Datei sein. Sie werden oft auch ***„Soft Links“*** genannt und werden durch den Buchstaben `l` gekennzeichnet.
4. ***Named pipes***: falls Sie sich wundern, ja, diese Dateien sind ähnlich den in den shell-Befehlen benutzten Pipes, nur mit dem Unterschied, dass diese hier gemeinten Dateien Namen besitzen. Sie sind jedoch recht selten und Sie werden bei Ihrer Reise durch den Verzeichnisbaum kaum eine davon entdecken. Falls doch,

erkennen Sie sie an der Kennzeichnung p. Weitere Informationen finden Sie im Kapitel *„Anonyme“ Pipes und Named Pipes*, Seite 69.

5. ***Sockets***: dies ist der Dateityp aller Netzwerk-Verbindungen, von denen allerdings nur wenige einen Namen besitzen. Weiterhin gibt es mehrere Arten von Sockets und nur eine davon kann verknüpft werden. Aber das geht weit über die Aufgabe dieses Buches hinaus. Die Sockets werden durch den Buchstaben s gekennzeichnet.

Im Folgenden finden Sie ein Beispiel für jeden Dateityp:

```
$ ls -l /dev/null /dev/sda  /etc/rc.d/rc3.d/S20random /proc/554/maps \
/tmp/ssh-franz/ssh-510-agent
crw-rw-rw-    1 root     root       1,   3 May  5  1998 /dev/null
brw-rw----    1 root     disk       8,   0 May  5  1998 /dev/sda
lrwxrwxrwx    1 root     root           16 Dec  9 19:12 /etc/rc.d/rc3.d/
  S20random -> ../init.d/random*
pr--r--r--    1 franz  franz         0 Dec 10 20:23 /proc/554/maps|
srwx------    1 franz  franz         0 Dec 10 20:08 /tmp/ssh-franz/
  ssh-510-agent=
$
```

9.2.2 Inodes

Inodes sind, genauso wie der Grundsatz „Alles ist eine Datei“, ein fundamentaler Teil jeden UNIX®-Dateisystems. Das Wort ***„Inode“*** ist die Kurzfassung von *Information NODE*.

Inodes werden auf der Festplatte in einer **Inode-Tabelle** gespeichert. Es gibt sie für alle Dateitypen, die in einem Dateisystem gespeichert werden, inklusive Verzeichnisse, Named Pipes, usw. Das bringt uns zu dem nächsten Merksatz: „Der Inode ist die Datei!“. UNIX® benutzt Inodes zur eindeutigen Identifikation einer Datei.

Ja, Sie haben richtig gelesen: unter UNIX® **identifizieren Sie eine Datei nicht durch ihren Namen** sondern durch die Inode-Nummer. [3] Der Grund dafür ist die Möglichkeit, dass eine einzelne Datei mehrere Namen oder auch garkeinen Namen haben kann. Ein Dateiname ist in UNIX® nur ein Eintrag in einem Verzeichnis-Inode. Dieser Eintrag wird **Link** genannt. Diese Links schauen wir uns jetzt genauer an.

9.3 Links

Am besten kann man Links mit einem Beispiel erklären. Erstellen Sie also eine (normale) Datei:

```
$ pwd
/home/franz/example
   $ ls
   $ touch a
   $ ls -il a
   32555 -rw-rw-r--    1 franz     franz            0 Dec 10 08:12 a
```

Die Option `-i` des Befehls `ls` gibt die Inode-Nummer aus, die am Anfang der Ausgabezeile steht. Wie Sie sehen können, waren vor der Erstellung der Datei `a` keine Dateien im Verzeichnis vorhanden. Das nächste interessante Feld ist das dritte Feld, in dem die Anzahl der Datei-Links (besser: Inode-Links) aufgelistet ist.

Der Befehl `touch a` kann in zwei Aktionen aufgegliedert werden:

- Erstellung eines Inode, dem vom Betriebssystem eine Nummer (32555) zugeteilt wird und dessen Dateityp der einer normalen Datei ist;

3. Wichtig: Inode-Nummern sind nur innerhalb **eines Dateisystems** eineindeutig, d.h., in einem anderen Dateisystem kann die gleiche Inode-Nummer vorhanden sein. Damit kommen wir zum Unterschied zwischen „on-disk“ Inodes und „in-memory“ Inodes. Während zwei Inodes auf einer Festplatte die gleiche Nummer besitzen können, solange sie in verschiedenen Dateisystemen existieren, haben Speicher-Inodes eine einzigartige Nummer innerhalb des gesamten Systems. Eine mögliche Lösung zur Erlangung einzigartiger Nummern ist, sie mit dem Bezeichner des Block-Gerätes (der Partition) zu verbinden.

- und das Anlegen eines Links zu diesem Inode, genannt `a` im aktuellen Verzeichnis `/home/franz/example`. Somit ist die Datei `/home/franz/example/a` ein Link zum Inode Nummer 32555, und zwar der zur Zeit einzige Link, wie man an der `1` im dritten Feld sehen kann.

Wenn wir aber das Folgende eingeben:

```
$ ln a b
$ ls -il a b
32555 -rw-rw-r--    2 franz     franz           0 Dec 10 08:12 a
32555 -rw-rw-r--    2 franz     franz           0 Dec 10 08:12 b
$
```

erstellen wir einen zweiten Link zum gleichen Inode. Wie Sie unschwer sehen können, haben wir keine Datei mit Namen `b` erstellt sondern einen zweiten Link auf den Inode Nummer 32555 im gleichen Verzeichnis angelegt und diesem neuen Link den Namen `b` gegeben. Daher steht jetzt in der Ausgabe von `ls -l` eine 2 im dritten Feld anstelle der 1.

Jetzt machen wir folgendes:

```
$ rm a
$ ls -il b
32555 -rw-rw-r--    1 franz     franz           0 Dec 10 08:12 b
$
```

und stellen fest, dass, obwohl wir die „ursprüngliche Datei“ gelöscht haben, der Inode immer noch existiert. Allerdings hat er jetzt nur noch einen Link und zwar den auf die Datei `/home/franz/example/b`.

Daher hat eine Datei unter UNIX® keinen Namen sondern einen oder mehrere ***Link(s)*** in einem oder mehreren Verzeichnis(sen).

Auch Verzeichnisse sind in Inodes gespeichert, ihre Links entsprechen der Anzahl von Unterverzeichnissen, die sie besitzen. Daher hat jedes Verzeichnis mindestens zwei Links: der des Verzeichnisses selbst (`.`) und der des übergeordneten Verzeichnisses (`..`).

Typische Vertreter von Dateien ohne Links (d.h. ohne Namen) sind Netzwerk-Verbindungen; Sie werden, egal wo Sie suchen, nie eine Datei finden, die Ihre Internet-Verbindung zur Mandrake Linux Website (`www.mandrakelinux.com`) anzeigt. Gleiches gilt für eine ***Pipe*** in der shell, der Inode für die Pipe existiert, ist aber nicht gelinkt. Weitere Beispiele für Inodes ohne Namen sind temporäre Dateien. Sie erstellen eine temporäre Datei, arbeiten damit und löschen sie wieder. Solange die Datei geöffnet ist existiert sie auch. Sie kann aber von niemandem sonst benutzt werden, da sie keinen Namen besitzt. Daher wird die temporäre Datei auch automatisch entfernt falls die Anwendung einmal abstürzt.

9.4 „Anonyme“ Pipes und Named Pipes

Lassen Sie uns zum Beispiel der Pipes zurückkehren, da es erstens sehr interessant und zweitens eine gute Beschreibung des Link-Prinzips ist. Wenn Sie eine Pipe in der Kommandozeile benutzen, wird diese von der shell erstellt und so benutzt, dass der Befehl vor der Pipe in die Pipe schreibt während der Befehl danach aus ihr liest. Alle Pipes, sowohl anonyme (wie die der shell) als auch benannte (siehe unten) agieren wie FIFOs (*„First In, First Out“*). Wir haben bereits Beispiele für den Gebrauch von Pipes in der shell gesehen, aber lassen Sie uns noch einmal genauer hinsehen:

```
$ ls -d /proc/[0-9] | head -5
/proc/1/
/proc/2/
/proc/3/
/proc/4/
/proc/5/
```

Ein wichtiger Aspekt bleibt in diesem Beispiel verborgen (der Ablauf ist zu schnell): Schreibaktionen in die Pipe blockieren die Pipe. Das bedeutet, dass nach einem Schreibvorgang des Befehls `ls` in die Pipe diese solange blockiert ist, bis ein Prozess am anderen Ende wieder daraus liest. Um diesen Effekt besser begreifen zu können, sollten sie einmal eine Named Pipe erstellen, die - anders als die Pipes der shell - Namen besitzen. [4] Der Befehl zur Erstellung einer Named Pipe lautet `mkfifo`:

4. Es gibt weitere Unterschiede zwischen den beiden Arten von Pipes, diese hier auszuführen würde aber den Rahmen dieses Buches sprängen.

```
$ mkfifo eine_pipe
$ ls -il
total 0
    169 prw-rw-r--    1 franz     franz        0 Dec 10 14:12 eine_pipe|
  #
  # Sie sehen, dass der Linkzähler auf 1 steht und die Ausgabe zeigt,
  # dass die Datei eine Pipe ist ('p').
  #
  # Auch hier können Sie ln benutzen:
  #
$ ln eine_pipe die_selbe_pipe
$ ls -il
total 0
    169 prw-rw-r--    2 franz     franz        0 Dec 10 15:37 eine_pipe|
    169 prw-rw-r--    2 franz     franz        0 Dec 10 15:37 die_selbe_pipe|
$ ls -d /proc/[0-9] >a_pipe
  #
  # Der Prozess ist blockiert, da es keinen Lesebefehl am anderen Ende gibt.
  # Geben Sie Strg-z ein, um den Prozess anzuhalten...
  #
zsh: 3452 suspended  ls -d /proc/[0-9] > eine_pipe
  #
  # ...Dann stellen Sie ihn in den Hintergrund:
  #
$ bg
[1]  + continued  ls -d /proc/[0-9] > eine_pipe
  #
  # jetzt lesen Sie von der Pipe...
  #
$ head -5 <die_selbe_pipe
  #
  # ...und der Schreibprozess beendet sich
  #
[1]  + 3452 done       ls -d /proc/[0-9] > eine_pipe
/proc/1/
/proc/2/
/proc/3/
/proc/4/
/proc/5/
#
```

Genauso blockieren natürlich auch Lesevorgänge den Prozess. Wenn Sie die obigen Befehle in umgekehrter Reihenfolge eingeben, werden Sie feststellen, dass `head` blockiert und auf einen Prozess wartet, der ihm etwas zum Lesen gibt:

```
$ head -5 <eine_pipe
  #
  # Programm blockiert, geben Sie Strg-z ein
  #
zsh: 741 suspended  head -5 < eine_pipe
  #
  # Stellen Sie ihn in den Hintergrund...
  #
$ bg
[1]  + continued  head -5 < eine_pipe
  #
  # ...und füttern Sie ihn :)
  #
$ ls -d /proc/[0-9] >die_selbe_pipe
$ /proc/1/
/proc/2/
/proc/3/
/proc/4/
/proc/5/
[1]  + 741 done       head -5 < eine_pipe
$
```

Sie werden bei diesem Beispiel auch einen unerwünschten Effekt feststellen: der Befehl `ls` wird beendet bevor der Befehl `head` übernommen hat. Als Folge kehren Sie sofort zum Prompt zurück. Der Befehl `head` agiert aber erst später, sodass Sie zur Ansicht der Ausgabe noch einmal Return drücken müssen.

9.5 „Spezielle“ Dateien: Zeichenorientierte- und Blockorientierte Dateien

Wie bereits bemerkt, werden solche Dateien entweder vom System erstellt oder von Geräten Ihres PCs. Wir haben auch bereits geschrieben, dass blockorientierte Geräte gepuffert werden, zeichenorientierte Dateien jedoch nicht. Um das zu veranschaulichen legen Sie eine Diskette in Ihr Laufwerk ein und schreiben Sie zweimal den folgenden Befehl:

```
$ dd if=/dev/fd0 of=/dev/null
```

Sie sollten nun folgendes festgestellt haben: beim der ersten Befehlseingabe wurde der gesamte Inhalt der Diskette gelesen. Beim zweiten Mal fand keinerlei Zugriff auf die Diskette statt. Das geschah weil der Inhalt der Diskette nach dem ersten Lesen bereits im Speicher gepuffert wurde und danach keine Änderung des Inhaltes vorgenommen wurde.

Wenn Sie nun aber eine große Datei auf diese Art ausdrucken wollen (ja, das funktioniert):

```
$ cat /eine/riesige/Datei >/dev/lp0
```

benötigt der Befehl zur Ausführung immer die gleiche Zeit, egal wie oft Sie ihn hintereinander eingeben. Das geschieht; da `/dev/lp0` eine zeichenorientierte Datei ist, dessen Inhalt nicht gepuffert wird.

Der Umstand, dass blockorientierte Dateien gepuffert werden, hat einen netten Seiteneffekt: es werden nicht nur Lesezugriffe sondern auch Schreibaktionen gepuffert. Das ermöglicht ein asynchrones Schreiben auf Festplatten: wenn Sie den Befehl zum Schreiben geben, findet die Aktion nicht sofort statt sondern dann, wenn Linux das möchte.

Schließlich hat jede spezielle Datei je eine sog. ***Major***- und eine ***Minor***-Nummer. Bei der Ausgabe des Befehls `ls -l` erscheinen diese Nummern anstelle der Größenangabe (die Größe ist bei speziellen Dateien irrelevant):

```
ls -l /dev/ide/host0/bus0/target0/lun0/disc /dev/printers/0
brw-------    1 root     root       3,   0 dic  31  1969 /dev/ide/host0/bus0/target0/lun0/disc
crw-rw----    1 lp       sys        6,   0 dic  31  1969 /dev/printers/0
```

Hier sind die Major und Minor von `/dev/ide/host0/bus0/target0/lun0/disc` 3 und 0, während sie für `/dev/printers/0` 6 und 0 betragen. Beachten Sie, dass diese Nummern nur innerhalb einer Datei-Kategorie eindeutig sind, d.h., es kann eine zeichenorientierte Datei mit Major 6 und Minor 0 geben (diese Datei gibt es tatsächlich: `/dev/pty/s0`) und gleichzeitig eine blockorientierte Datei mit Major 6 und Minor 0. Diese Nummern existieren, damit der Kern der Datei die richtigen Aktionen zuweisen kann (eigentlich den Geräten, auf die diese Dateien verweisen): ein Disketten-Laufwerk erfordert schließlich andere Aktionen als etwa eine SCSI-Festplatte.

9.6 Symbolische Links und die Schwächen von „Hard“-Links

Hier wenden wir uns einem oft (sogar unter UNIX®-Anwendern) vorkommenden Missverständnis zu, das hauptsächlich dadurch entsteht, dass Links so wie wir sie bisher kennen lernten (fälschlicherweise „Hard“ Links genannt) nur mit normalen Dateien verknüpft waren (wir haben aber auch gesehen, dass das nicht der Fall ist – sogar symbolische Links sind „verknüpft“). Zuerst müssen wir jedoch erklären, was eigentlich symbolische Links („Soft“ Links oder auch „Symlinks“) sind.

Symbolische Links sind Dateien eines speziellen Typs, deren einziger Inhalt eine beliebige Zeichenkette ist, die vom System als Pfad zur eigentlichen Datei interprretiert werden. Diese muss nicht notwendigerweise existieren. Wenn Sie in der Kommandozeile oder in einem Programm einen symbolischen Link verwenden, so benutzen Sie in Wirklichkeit die damit verknüpfte Datei, falls sie existiert. Ein Beispiel:

```
$ echo Hallo >myfile
$ ln -s myfile mylink
$ ls -il
total 4
    169 -rw-rw-r--    1 franz     franz            6 Dec 10 21:30 myfile
    416 lrwxrwxrwx    1 franz     franz            6 Dec 10 21:30 mylink
-> myfile
$ cat myfile
Hallo
$ cat mylink
Hallo
```

Sie sehen, dass der Dateityp von `mylink` ein `l`, also ein symbolischer *Link* ist. Die Zugriffsrechte für symbolische Links sind uninteressant: sie sind immer `rwxrwxrwx`. Sie können ebenfalls feststellen, dass der Link **wirklich** eine andere Datei als `myfile` ist, da die Inode-Nummern differieren. Der Link verweist aber symbolisch auf die Datei, also geben Sie bei dem Befehl `cat mylink` in Wirklichkeit den Inhalt der Datei `myfile` aus. Zur Anschauung, dass ein symbolischer Link eine beliebige Zeichenkette enthalten kann, geben Sie folgendes ein:

```
$ ln -s "Ich bin keine existierende Datei" anotherlink
$ ls -il anotherlink
    418 lrwxrwxrwx    1 franz     franz           20 Dec 10 21:43 anotherlink
-> Ich bin keine existierende Datei
$ cat anotherlink
cat: anotherlink: No such file or directory
$
```

Symbolische Links existieren, um verschiedene Schwachpunkte der normalen („Hard“) Links zu vermeiden:

- Sie können keinen Link auf einen Inode in einem Verzeichnis erstellen, das in einem anderen Dateisystem liegt als der bewusste Inode. Der Grund ist einfach: Der Linkzähler ist im Inode gespeichert und Inodes können nicht zwischen Dateisystemen gemeinsam benutzt werden. Mit Symlinks funktioniert das jedoch;
- Sie können Verzeichnisse nicht innerhalb eines Dateisystems verlinken, aber Sie können einen Symlink erstellen, der auf ein Verzeichnis verweist und diesen Symlink dann wie ein Verzeichnis nutzen.

Symbolische Links sind daher bei vielen Gelegenheiten nützlich und zahlreiche Anwender tendieren dazu, Symlinks zu verwenden obwohl bei manchen Situationen auch normale Links funktionieren würden. Ein Vorteil der normalen Links ist jedoch, dass eine Datei nicht verloren geht, wenn der Originallink gelöscht wird.

Schließlich und endlich noch eine Bemerkung zur Größe eines Links: Sie haben vermutlich bemerkt, dass die Größe eines sybolischen Links der Größe der enthaltenen Zeichenfolge entspricht.

9.7 Dateiattribute

Genauso wie FAT (Archiv, Systemdatei, Unsichtbar) haben auch GNU/Linux-Dateisysteme ihre eigenen Attribute, die aber sehr verschieden sind. Wir werden sie hier der Vollständigkeit halber kurz behandeln, obwohl sie sehr selten genutzt werden. Wenn Sie jedoch an einem wirklich sicheren System interessiert sind sollten Sie hier weiterlesen.

Es gibt zwei Befehle zur Bearbeitung von Dateiattributen: `lsattr(1)` und `chattr(1)`. Das ist einfach: `lsattr` „LiStet“ Attribute, während `chattr` sie ändert (*CHange*). Die Attribute können nur bei Verzeichnissen und normalen Dateien verwendet werden. Sie lauten wie folgt:

1. `A` (*no **A**ccess time*): wenn eine Datei oder Verzeichnis dieses Attribut besitzt, wird die Zugriffszeit (zum Öffnen, Lesen oder Schreiben) nicht aktualisiert. Das ist dann hilfreich, wenn Dateien oder Verzeichnisse sehr oft zum Lesen geöffnet werden; besonders deshalb, weil dieser Parameter die einzige Veränderliche eines Inodes einer „Nur Lesen“-Datei ist.
2. `a` (***a**ppend only*): bei einer Datei mit diesem Attribut können bei einem Schreibvorgang nur Daten zum ursprünglichen Inhalt hinzugefügt werden. Bei einem Verzeichnis bedeutet das, dass nur neue Dateien hinzugefügt werden können, aber keine Dateien gelöscht oder umbenannt werden dürfen. Nur `root` kann dieses Attribut setzen oder entfernen.
3. `d` (*no **d**ump*): `dump (8)` ist in UNIX® das Standardwerkzeug für Backups. Es sichert alle Dateisysteme, die in `/etc/fstab` eine 1 aufweisen (siehe Kapitel *Dateisysteme und Einhängpunkte*, Seite 59). Wenn jedoch eine Datei oder ein Verzeichnis dieses Attribut aufweist wird es bei einer Datensicherung übergangen. Beachten Sie, dass das bei Verzeichnissen auch alle Unterverzeichnisse und deren Inhalte einschließt.
4. `i` (***i**mmutable*): eine Datei oder ein Verzeichnis mit diesem Attribut kann in keinster Weise verändert werden, sie darf nicht umbenannt werden, es darf kein Link auf sie gesetzt werden [5] und sie darf nicht gelöscht werden. Nur `root` kann dieses Attribut setzen oder entfernen. Da hiermit auch Änderungen der Zugriffszeit vermieden werden brauchen Sie bei Verwendung von `i` das Attribut `A` nicht zusätzlich zu setzen.

5. Denken Sie daran, was „Hinzufügen eines Links“ tatsächlich bedeutet, sowohl für eine Datei als auch für ein Verzeichnis!

5. `s` (***s**ecure deletion*): bei einer Löschung einer Datei oder eines Verzeichnisses mit diesem Attribut wird der bisher davon belegte Festplattenbereich mit Nullen überschrieben.
6. `S` (***S**ynchronous mode*): Bei einer Änderung einer Datei oder eines Verzeichnisses mit diesem Attribut werden die Änderungen sofort auf die Festplatte zurück geschrieben.

Sie könnten nun zum Beispiel bei wichtigen Systemdateien das Attribut `i` setzen um böse Überraschungen zu vermeiden. Auch ein `A` für die „man“ Dateien wäre angebracht, da Sie damit eine Menge Festplattenaktivität und bei Laptops dadurch auch Batterie-Lebensdauer einsparen.

Kapitel 10. Das Dateisystem /proc

Das Dateisystem `/proc` ist eine GNU/Linux-Spezialität. Es ist ein virtuelles Dateisystem, also belegen die Dateien in diesem Verzeichnis keinen realen Platz auf Ihrer Festplatte. Es ist eine sehr hilfreiche Möglichkeit Systeminformationen zu erlangen, insbesondere da die meisten Dateien dieses Verzeichnisses für den Benutzer lesbar sind (nun ja, zumindest mit ein wenig Nachhilfe). Tatsächlich holen sich viele Programme Informationen aus den Dateien in `/proc`, formatieren sie in ihrem eigenen Format und geben das Resultat aus. Einige Programme machen das wenn sie Informationen über Prozesse ausgeben (etwa `top` oder `ps`). `/proc` ist aber auch eine gute Informationsquelle über Ihre Hardware. So sind genau wie die Programme zur Ausgabe von Prozessinformationen auch Programme vorhanden, die nur als „Aufbereiter" für Informationen aus dem Verzeichnis `/proc` dienen.

Das spezielle Unterverzeichnis, `/proc/sys` ermöglicht es, Kernparameter aufzulisten und sie zu ändern, wobei die Änderungen sofort umgesetzt werden.

10.1 Informationen über Prozesse

In der Liste des Verzeichnisinhaltes von `/proc` werden Sie viele Unterverzeichnisse mit numerischem Namen finden. Das sind Verzeichnisse, die alle Informationen über die aktuell auf dem System laufenden Prozesse enthalten:

```
$ ls -d /proc/[0-9]*
/proc/1/    /proc/302/  /proc/451/  /proc/496/  /proc/556/  /proc/633/
/proc/127/  /proc/317/  /proc/452/  /proc/497/  /proc/557/  /proc/718/
/proc/2/    /proc/339/  /proc/453/  /proc/5/    /proc/558/  /proc/755/
/proc/250/  /proc/385/  /proc/454/  /proc/501/  /proc/559/  /proc/760/
/proc/260/  /proc/4/    /proc/455/  /proc/504/  /proc/565/  /proc/761/
/proc/275/  /proc/402/  /proc/463/  /proc/505/  /proc/569/  /proc/769/
/proc/290/  /proc/433/  /proc/487/  /proc/509/  /proc/594/  /proc/774/
/proc/3/    /proc/450/  /proc/491/  /proc/554/  /proc/595/
```

Bedenken Sie, dass Ihnen als Anwender (logischerweise) nur die Informationen der von Ihnen gestarteten Prozesse zugänglich sind, nicht die der anderen Benutzer. Wechseln Sie also in den Kontext des priv. Kennzeichens `root` um zu sehen, welche Informationen über den Prozess 127 vorhanden sind:

```
$ su
Password:
$ cd /proc/127
$ ls -l
total 0-9
-r--r--r--    1 root     root            0 Dec 14 19:53 cmdline
lrwx------    1 root     root            0 Dec 14 19:53 cwd -> //
-r--------    1 root     root            0 Dec 14 19:53 environ
lrwx------    1 root     root            0 Dec 14 19:53 exe -> /usr/sbin/apmd*
dr-x------    2 root     root            0 Dec 14 19:53 fd/
pr--r--r--    1 root     root            0 Dec 14 19:53 maps|
-rw-------    1 root     root            0 Dec 14 19:53 mem
lrwx------    1 root     root            0 Dec 14 19:53 root -> //
-r--r--r--    1 root     root            0 Dec 14 19:53 stat
-r--r--r--    1 root     root            0 Dec 14 19:53 statm
-r--r--r--    1 root     root            0 Dec 14 19:53 status
$
```

Jedes Verzeichnis enthält die gleichen Einträge. Hier folgt eine kurze Erklärung einiger dieser Einträge:

1. `cmdline`: Dies ist eine (Pseudo-)Datei, die die gesamte Befehlszeile zum Start des Prozesses enthält. Da diese Zeile nicht formatiert ist, also keine Leerfelder zwischen Programmname und Argumenten und keinen Zeilenumbruch am Ende enthält, muss man sich die Ansicht mit dem folgenden Befehl etwas aufbessern: `perl -ple 's,\00, ,g' cmdline`.
2. `cwd`: dieser symbolische Link deutet auf das aktuelle Arbeitsverzeichnis des Prozesses (daher auch der Name des Links).
3. `environ` Diese Datei enthält alle für den Prozess definierten Umgebungsvariablen, aufgelistet in der Form: `VARIABLE=wert`. Genau wie bei `cmdline` ist auch diese Ausgabe nicht formatiert. Also sollte man zur Aufbereitung der Ansicht den folgenden Befehl benutzen: `perl -pl -e 's,\00,\n,g' environ`.

4. exe: ein symbolischer Link, der auf die für den laufenden Prozess benutzte Programmdatei zeigt („EXEcutable“).
5. fd: dieses Unterverzeichnis enthält die Liste der aktuell vom Prozess geöffneten Datei-Deskriptoren (siehe weiter unten).
6. maps: wenn Sie sich den Inhalt dieser *Named Pipe* anzeigen lassen (beispielsweise mit cat), können Sie die Teile des vom Prozess belegten Adressraums sehen, die aktuell in der Datei abgebildet werden. Die Felder enthalten, von links nach rechts: den Adressraum dieser Speicherabbildung, die zugeteilten Rechte, den Offset vom Start der Speicherabbildung, die Major- und Minor-Nummer (in Hexadezimal-Notation) des Gerätes auf dem die abgebildete Datei liegt, die Inode-Nummer der Datei und schließlich den Namen der Datei. Wenn das Gerät die Nummer 0 hat und weder Inode-Nummer noch Dateiname vorhanden sind, handelt es sich um ein Anonymes Mapping (siehe mmap(2)).
7. root: dies ist ein symbolischer Link, der auf das vom Prozess benutzte Wurzelverzeichnis zeigt. Im Normalfall ist das / (siehe aber auch chroot(2)).
8. status: diese Datei enthält verschiedene Informationen über den Prozess: den Namen der Programmdatei, den aktuellen Status, seine PID und PPID, seine echte und effektive UID und GID, seine Speicherbelegung und weitere Daten. Beachten Sie bitte, dass die bisher benutzten Dateien stat und statm nicht mehr verwendet werden. Die früher darin enthaltenen Informationen liegen jetzt alle in status.

Eine Auflistung des Verzeichnisses fd für den Prozess 127 zeigt das folgende Bild:

```
$ ls -l fd
total 0
lrwx------    1 root     root           64 Dec 16 22:04 0 -> /dev/console
l-wx------    1 root     root           64 Dec 16 22:04 1 -> pipe:[128]
l-wx------    1 root     root           64 Dec 16 22:04 2 -> pipe:[129]
l-wx------    1 root     root           64 Dec 16 22:04 21 -> pipe:[130]
lrwx------    1 root     root           64 Dec 16 22:04 3 -> /dev/apm_bios
lr-x------    1 root     root           64 Dec 16 22:04 7 -> pipe:[130]
lrwx------    1 root     root           64 Dec 16 22:04 9 ->
/dev/console
$
```

Dies ist die Liste der vom Prozess geöffneten Datei-Deskriptoren. Jeder geöffnete Deskriptor wird als symbolischer Link angezeigt, wobei der Name die Deskriptor-Nummer ist. Der Link zeigt auf die bestimmte Datei, die vom jeweiligen Deskriptor geöffnet wurde[1]. Beachten Sie die Zugriffsrechte der Links: hier ist der einzige Ort, wo sie einen Sinn ergeben. Sie entsprechen den Rechten, mit denen die dem Deskriptor entsprechende Datei geöffnet wurde.

10.2 Informationen über die Hardware

Neben den den verschiedenen Prozessen zugeordneten Verzeichnissen enthält /proc eine Unmenge von Informationen über die Hardware Ihres Rechners. Eine Auflistung des Verzeichnisses /proc sieht etwa so aus:

```
$ ls -d [a-z]*
apm      dma          interrupts  loadavg  mounts      rtc       swaps
bus/     fb           ioports     locks    mtrr        scsi/     sys/
cmdline  filesystems  kcore       meminfo  net/        self/     tty/
cpuinfo  fs/          kmsg        misc     partitions  slabinfo  uptime
devices  ide/         ksyms       modules  pci         stat      version
$
```

Wenn wir uns beispielsweise die Datei /proc/interrupts ansehen, stellen wir fest, dass sie eine Liste der aktuell im System benutzten Interrupts enthält, zusammen mit der Angabe, welche Geräte diese Interrupts benutzen. In gleicher Weise enthält ioports die Liste der gerade belegten Ein-/Ausgabe-Adressen und schließlich dma Informationen über die DMA-Kanäle. Also sollten Sie bei Hardwareproblemen auf jeden Fall den Inhalt dieser drei Dateien die Suche nach einer Konfliktlösung einbeziehen:

```
$ cat interrupts
           CPU0
  0:     127648          XT-PIC  timer
  1:       5191          XT-PIC  keyboard
```

1. Erinnern Sie sich daran, was im Kapitel *Umleitungen und Pipes*, Seite 27 beschrieben wurde, dann wissen Sie, was die Deskriptoren 0, 1 und 2 bedeuten.

```
  2:          0          XT-PIC  cascade
  5:       1402          XT-PIC  xirc2ps_cs
  8:          1          XT-PIC  rtc
 10:          0          XT-PIC  ESS Solo1
 12:       2631          XT-PIC  PS/2 Mouse
 13:          1          XT-PIC  fpu
 14:      73434          XT-PIC  ide0
 15:      80234          XT-PIC  ide1
NMI:          0
$ cat ioports
0000-001f : dma1
0020-003f : pic1
0040-005f : timer
0060-006f : keyboard
0070-007f : rtc
0080-008f : dma page reg
00a0-00bf : pic2
00c0-00df : dma2
00f0-00ff : fpu
0170-0177 : ide1
01f0-01f7 : ide0
0300-030f : xirc2ps_cs
0376-0376 : ide1
03c0-03df : vga+
03f6-03f6 : ide0
03f8-03ff : serial(auto)
1050-1057 : ide0
1058-105f : ide1
1080-108f : ESS Solo1
10c0-10cf : ESS Solo1
10d4-10df : ESS Solo1
10ec-10ef : ESS Solo1
$ cat dma
 4: cascade
$
```

Oder Sie benutzen den Befehl `lsdev`, der die Informationen aus diesen drei Dateien holt und sie nach Gerät sortiert, was für den Benutzer zweifellos bequemer ist.[2]:

```
$ lsdev
Device            DMA   IRQ  I/O Ports
------------------------------------------------
cascade             4     2
dma                          0080-008f
dma1                         0000-001f
dma2                         00c0-00df
ESS                          1080-108f 10c0-10cf 10d4-10df 10ec-10ef
fpu                       13 00f0-00ff
ide0                      14 01f0-01f7 03f6-03f6 1050-1057
ide1                      15 0170-0177 0376-0376 1058-105f
keyboard                   1 0060-006f
Mouse                     12
pic1                         0020-003f
pic2                         00a0-00bf
rtc                        8 0070-007f
serial                       03f8-03ff
Solo1                     10
timer                      0 0040-005f
vga+                         03c0-03df
xirc2ps_cs                 5 0300-030f
$
```

Eine umfassende Auflistung der Dateien würde unseren Rahmen sprengen. Daher folgen hier nur die Erklärungen einiger dieser Dateien:

- `cpuinfo`: wie der Name schon andeutet, enthält diese Datei Informationen über den/die Prozessor(en) des PCs.
- `modules`: hier finden Sie eine Liste aller aktuell vom Kern benutzten Module, versehen mit Zählern der Aufrufe des jeweiligen Moduls. Es ist die gleiche Information, die man auch mit dem Befehl `lsmod` bekommt.

2. `lsdev` ist Teil des Paketes `procinfo`.

- `meminfo`: Auslastung des Arbeitsspeichers zur Zeit des Aufrufs der Datei. Eine besser lesbare Darstellung bekommen Sie mit dem Befehl `free`.
- `apm`: falls Sie einen Laptop benutzen zeigt Ihnen der Inhalt dieser Datei den Status der Batterie des Laptops. Es wird dargestellt, ob das Steckernetzteil angeschlossen ist, wie hoch der Ladezustand der Batterie ist, ob das BIOS Ihres Laptops APM tatsächlich unterstützt (leider ist das nicht generell der Fall) und die verbleibende Batterieladung in Minuten. Auch der Inhalt dieser Datei ist nicht besonders aufbereitet, sodass Sie besser den Befehl `apm` zur Darstellung der Informationen benutzen sollten.

  Beachten Sie bitte, dass moderne Rechner eher ACPI anstelle von APM benutzen (siehe unten).

- `bus`: in diesem Verzeichnis finden Sie Informationen über alle aktuell an die verschiedenen Busse angeschlossenen Geräte. Diese Informationen sind im Allgemeinen nicht lesbar und müssen mit externen Werkzeugen (`lspcidrake`, `lspnp`, usw.) aufbereitet werden.
- `acpi`: Einige der in diesem Verzeichnis enthaltenen Dateien sind speziell für Laptop-Besitzer interessant. Sie können auf diese Weise verschiedene Stromspar-Optionen einstellen. Allerdings kann man diese Einstellungen bequemer mit höher angesiedelten Werkzeugen durchführen, wie etwa den in `acpid` und `kacpi` enthaltenen Programmen.

  Die interessantesten Einträge sind:

  `battery`

  zeigt an, wieviele Batterien im Laptop vorhanden sind, deren aktuelle Kapazität, die maximale Kapazität, usw.

  `button`

  Einstellung der Aktionen verschiedener „spezieller“ Bedienelemente wie etwa Power- und Sleep-Knopf, Verschluss des Laptopdeckel, usw.

  `fan`

  Zeigt den Status der Ventilatoren im Rechner an, d.h., ob aktiv oder nicht. Hier können Sie den/die Ventilator(en) nach gewissen Kriterien einstellen, jedenfalls soweit es das Motherboard erlaubt.

  `processor`

  Es gibt für jeden im Computer aktiven Prozessor ein Unterverzeichnis. Die Kontrolloptionen variieren, je nach Prozessortyp, wobei spezielle Mobilprozessoren erweiterte Möglichkeiten besitzen:

  - Benutzung verschiedener Leistungsstufen, je nach Leistungsbedarf und Stromverbrauch.
  - Einstellung der Taktfrequenz der CPU zur Minderung der Stromaufnahme.

  Beachten Sie bitte, dass nicht alle Prozessoren diese Fähigkeiten besitzen.

  `thermal_zone`

  Informationen über die Temperatur der CPU und des Systems.

10.3 Das Unterverzeichnis `/proc/sys`

Die Aufgabe dieses Unterverzeichnisses ist es, verschiedene Kernparameter aufzulisten und die interaktive Änderung einiger dieser Parameter zu ermöglichen. Im Unterschied zu allen anderen Dateien in `/proc` können einige der Dateien in diesem Verzeichnis verändert werden, allerdings nur von `root`.

Eine Auflistung aller Eigenschaften dieser Verzeichnisse und Dateien wäre zu umfangreich, insbesondere, da dieser Umfang systemabhängig ist und die meisten dieser Dateien nur ganz spezielle Anwendungen betreffen. Es gibt jedoch zwei Bereiche, für die diese Unterverzeichnisse sehr hilfreich sind:

1. Routing erlauben: Selbst wenn der Standardkern in Mandrakelinux das Routing ermöglicht, müssen Sie es explizit aktivieren. Dazu müssen Sie als `root` den folgenden Befehl eingeben:

   ```
   $ echo 1 >/proc/sys/net/ipv4/ip_forward
   ```

   Ersetzen Sie die 1 durch 0 falls Sie das Routing nicht erlauben wollen.

2. IP-Spoofing verhindern: Das IP-Spoofing spiegelt dem System vor, dass ein von aussen kommendes Datenpaket tatsächlich von dem Gerät stammt, das es empfängt. Diese Technik wird sehr oft von ***Crackern*** eingesetzt [3]. Sie können den Kern dazu bringen, diese Art des Eindringens zu verhindern. Geben Sie die folgende Befehlszeile ein:

   ```
   $ echo 1 >/proc/sys/net/ipv4/conf/all/rp_filter
   ```

   und diese Angriffsmöglichkeit ist nicht mehr vorhanden.

Diese Änderungen bleiben nur zur Laufzeit des Systems erhalten. Bei einem Neustart werden die Werte wieder auf den Standard zurückgesetzt. Um die Änderungen dauerhaft zu behalten, schreiben Sie die vorher auf der Konsole eingegebenen Befehle untereinander in die Datei `/etc/rc.d/rc.local`. Eine Alternative wäre eine Änderung der Datei `/etc/sysctl.conf` (siehe sysctl.conf(5)).

3. Nicht aber von ***Hackern***!

Kapitel 11. Die Startdateien: sysv initialisieren

Das „System V“ Modell ist ein Erbe des **AT&T** UNIX® und eines der traditionellen UNIX® Systemstart-Modelle. Das System V bewirkt den Start oder die Beendigung von Systemdiensten und bringt so das System in den jeweils gewünschten Status. Das umfasst alle für den jeweils gewünschten Systemstatus notwendigen Dienste. Diese reichen dabei von der Benutzer-Authentifizierung bis zum lokalen Grafik-Server oder den Internet-Diensten.

Das Ihnen für das manuelle Starten oder Stoppen der Systemdienste zur Verfügung stehende Mandrakelinux-Werkzeug nennt sich `drakxservices`. Sie finden es im Bereich „System“ des Mandrakelinux Control Center.

11.1 Am Anfang steht „init“

Wenn beim Systemstart der Kern alles erkannt und eingerichtet hat und die Verzeichnisbaumwurzel eingehängt wurde, wird der Befehl `/sbin/init` ausgeführt[1]. `init` ist der „Vater“ aller Prozesse des Systems und dafür verantwortlich, dass das System in den gewünschten ***Runlevel*** startet. Wir werden die Runlevel später noch behandeln (siehe *Runlevel*, Seite 81).

Die `init`-Konfigurationsdatei heißt `/etc/inittab`. Sie hat einen eigenen Handbuchauszug (inittab(5)), daher werden wir hier nur einige der möglichen Einstellungen wiedergeben.

Die erste Zeile, die Sie beachten sollten, ist diese:

```
si::sysinit:/etc/rc.d/rc.sysinit
```

Sie weist `init` an, das Skript `/etc/rc.d/rc.sysinit` sofort nach der Initialisierung des Systems auszuführen (`si` steht für *System Init*). `init` sucht anschließend den voreingestellten Runlevel, den es in der Zeile mit dem Schlüsselwort `initdefault` findet:

```
id:5:initdefault:
```

In diesem Fall stellt `init` fest, dass der Standard-Runlevel 5 ist. Es weiß ferner, dass zum Erreichen des Runlevels 5 der folgende Befehl ausgeführt werden muss:

```
l5:5:wait:/etc/rc.d/rc 5
```

Wie Sie sehen, ist die Syntax für jeden Runlevel gleich.

`init` ist weiterhin für den Neustart (`respawn`) einiger Programme verantwortlich, die von keinem anderen Prozess gestartet werden können. So werden beispielsweise alle Logins der sechs virtuellen Konsolen durch `init` gestartet.[2]. Die zweite virtuelle Konsole wird hier mit der folgenden Zeile identifiziert:

```
2:2345:respawn:/sbin/mingetty tty2
```

11.2 Runlevel

Alle Dateien, die beim Systemstart gebraucht werden, befinden sich im Verzeichnis `/etc/rc.d`. Es folgt eine Liste der Dateien:

```
$ ls /etc/rc.d
init.d/  rc0.d/  rc2.d/  rc4.d/  rc6.d/            rc.local*    rc.sysinit*
rc*      rc1.d/  rc3.d/  rc5.d/  rc.alsa_default*  rc.modules*
```

Wie bereits erwähnt. ist `rc.sysinit` die erste Datei, die vom System ausgeführt wird. Durch sie werden die grundlegenden Teile des Systems eingerichtet: Tastaturtyp, Einrichtung verschiedener Geräte, Prüfung der Dateisysteme, usw.

1. Daher ist es keine gute Idee, `/sbin` auf ein anderes Dateisystem zu verlegen. An diesem Punkt hat der Kern noch kein anderes Dateisystem eingehängt und wäre nicht in der Lage `/sbin/init` zu finden.
2. Wenn Sie mit der Anzahl der virtuellen Konsolen nicht zufrieden sind, können Sie in dieser Datei welche hinzufügen oder entfernen. Sie können bis zu maximal 64 Konsolen einrichten. Vergessen Sie dabei aber nicht, dass X auch auf einer dieser Konsolen läuft und Sie daher eine virtuelle Konsole dafür reservieren müssen.

Danach wird das Skript `rc` mit dem gewünschten Runlevel als Argument gestartet. Wie Sie sehen konnten ist der Runlevel eine einfache Zahl. Für jeden vorhandenen Runlevel `<x>` muss ein korrespondierendes Verzeichnis `rc<x>.d` existieren. In einem typischen Mandrakelinux-System finden Sie also sechs Runlevel:

- 0: vollständiger Halt des PCs.
- 1: ***single-user*** („Einzelbenutzer-Modus“). Wird zur Behebung von schweren Problemen oder zum Wiederherstellen des Systems benutzt.
- 2: ***multi-user*** („Mehrbenutzer-Modus“), ohne Netzwerk.
- 3: Mehrbenutzer-Modus, mit Netzwerk
- 4: unbenutzt.
- 5: wie Runlevel 3, allerdings mit Start des grafischen Anmeldeschirms.
- 6: Neustart.

Lassen Sie uns einen Blick auf den Inhalt des Verzeichnisses `rc5.d` werfen:

```
$ ls rc5.d
K15postgresql@  K60atd@      S15netfs@   S60lpd@        S90xfs@
K20nfs@         K96pcmcia@   S20random@  S60nfs@        S99linuxconf@
K20rstatd@      S05apmd@     S30syslog@  S66yppasswdd@  S99local@
K20rusersd@     S10network@  S40crond@   S75keytable@
K20rwhod@       S11portmap@  S50inet@    S85gpm@
K30sendmail@    S12ypserv@   S55named@   S85httpd@
K35smb@         S13ypbind@   S55routed@  S85sound@
```

Wie Sie sehen sind alle Dateien in diesem Verzeichnis nur symbolische Links mit einer sehr spezifischen Form:

```
<S|K><order><service_name>
```

Das `S` steht für *Start* des entsprechenden Dienstes und das `K` bedeutet *Kill* (Stopp) für den entsprechenden Dienst. Die Skripte werden in aufsteigender numerischer Reihenfolge ausgeführt. Wenn zwei Skripte die gleiche Nummer haben gilt die alphabetische Reihenfolge. Jeder der symbolischen Links zeigt auf ein vorhandenes Skript im Verzeichnis `/etc/rc.d/init.d` (mit Ausnahme von `local`, das einen speziellen Dienst kontrolliert).

Wenn das System in einen bestimmten Runlevel startet werden zuerst alle `K`-Links der Reihe nach aufgerufen und deren korrespondierende Skripte mit dem Argument `stop` gestartet. Dann werden die `S`-Skripte in der gleichen Art, allerdings mit dem Parameter `start` ausgeführt.

Also können wir, ohne jetzt alle vorhandenen Skripte zu behandeln, sehen, dass das System im Runlevel 5 zuerst den Befehl `K15postgresql` (also `/etc/rc.d/init.d/postgresql stop`) ausführt. Danach kommt `K20nfs`, dann `K20rstatd`, usw., bis zum letzten Befehl. Anschließend werden alle `S`-Skripte gestartet: zuerst `S05apmd` (also `/etc/rc.d/init.d/apmd start`), und so weiter.

Mit diesen Informationen versorgt, können Sie in ein paar Minuten Ihren eigenen Runlevel erstellen (beispielsweise den Runlevel 4) oder den einen oder anderen Dienst durch Entfernen des entsprechenden Links davon abhalten, gestartet oder gestoppt zu werden. Es gibt für diese Aufgabe auch einige Programme, speziell drakxservices (siehe *DrakXServices: Einrichtung der Dienste beim Systemstart* im *Starter Handbuch*) mit einem grafischen Interface oder chkconfig für den Textmodus.

Sie können auch den Befehl `chkconfig` zur Auflistung sowie zum Hinzufügen oder Löschen von Systemdiensten in einem bestimmten Runlevel benutzen. Siehe chkconfig(8).

Kapitel 12. Kompilieren und Installieren Freier Software

Wir werden oft gefragt, wie man freie Software aus den Quellen installiert. Das selbständige Kompilieren der Software ist sehr einfach, da in den meisten Fällen der Arbeitsablauf immer gleich ist, egal um welche Software es sich handelt. Das Ziel dieses Kapitels ist, den Neuling in diesem Bereich durch die einzelnen Schritte des Vorgangs zu begleiten und jeden dieser Schritte zu erklären. Wir gehen davon aus, dass der Leser eine minimale Kenntnis des UNIX®-Systems mitbringt (beispielsweise von `ls` oder `mkdir`).

Dieses Dokument ist nur eine Anleitung, keine Referenz. Zur Beantwortung weiterführender Fragen geben wir Ihnen daher am Ende einige Links. Vermutlich kann die Anleitung noch verbessert werden. Wir nehmen deshalb jeden Kommentar und jede Verbesserung etwaiger Fehler gerne an.

12.1 Einleitung

Einer der großen Unterschiede zwischen freier und proprietärer Software ist die Verfügbarkeit des Quellcodes *Source Code*)[1]. Das hat zur Folge, dass freie Software in Form von Archiven mit Code-Dateien vertrieben wird. Für die meisten Einsteiger ist das eine ungewöhnliche Situation, da sie sich als Benutzer von freier Software die Programme zuerst selbst durchbauen müssen, bevor sie sie benutzen können.

Es gibt aber auch fertig kompilierte Versionen des größten Teils der existierenden freien Software. Ein eiliger Benutzer muss sich nur diese kompilierten Binärdateien installieren. Einige Programme der freien Software werden jedoch nicht in dieser fertigen Form verbreitet, auch frühe Versionen kann man selten in binärer Form bekommen. Des Weiteren werden Sie für ein exotisches Betriebssystem oder eine ungewöhnliche Hardware-Architektur auch kaum dafür kompilierte Software finden. Aber der weitaus interessantere Aspekt der eigenen Software-Kompilierung ist die Möglichkeit, dabei nur die für Sie interessanten Optionen zu aktivieren oder die Funktionalität der Software zu erweitern, so dass Sie ein genau für Ihre Bedürfnisse zugeschnittenes Programm erhalten.

12.1.1 Voraussetzungen

Zum Durchbauen der Software benötigen Sie:

- einen Rechner mit einem laufenden Betriebssystem,
- allgemeine Kenntnis des von Ihnen benutzten Systems,
- etwas freien Platz auf Ihrer Festplatte,
- einen Compiler (im Allgemeinen für C) und ein Archivierungsprogramm (z.B. `tar`),
- etwas Nahrung (in schwierigen Fällen kann es lange dauern). Ein echter Hacker nimmt Pizza zu sichs – keine Burger!
- etwas Trinkbares (aus den gleichen Gründen). Ein echter Hacker trinkt Cola – wegen des Koffeins!
- die Telefonnummer Ihres technisch begabten Freundes, der sowas jede Woche macht,
- viel Geduld!

Das Durchbauen eines Quellcodes bringt im Allgemeinen kaum Probleme mit sich. Als Neuling werden Sie jedoch von der kleinsten Hürde aufgehalten. Dieses Dokument soll Ihnen beim Überwinden dieser Hürden helfen.

1. Das ist nicht ganz korrekt, da es einige proprietäre Programme gibt, deren Quellcode offen gelegt wird. Allerdings darf ihn der Anwender – anders als bei freier Software – nicht benutzen oder nach seinem Willen verändern.

12.1.2 Kompilierung

12.1.2.1 Das Prinzip

Um einen Quellcode in ein binäres Programm umzuwandeln müssen Textdateien (normalerweise C oder C++, die die meist verwendeten Sprachen der (UNIX®) Freien Software Gemeinschaft) ***kompiliert*** werden. Einige Programme sind in Sprachen geschrieben, die kein Durchbauen benötigen (etwa perl oder die shell). Diese müssen aber zumindest eingerichtet werden.

Das Durchbauen von C-Quellen wird logischerweise von einem C-Compiler durchgeführt. Meist wird dafür der `gcc`, der freie Compiler des GNU-Projektes (`http://www.gnu.org/`) verwendet. Vollständige Softwarepakete durchzubauen ist eine komplexe Aufgabe, die aus dem nacheinander kompilieren einzelner Quellcode-Dateien besteht (aus vielerlei Gründen ist es für Entwickler einfacher, die verschiedenen Module eines Projektes in einzelne Dateien zu speichern). Für Sie wird diese Aufgabe durch ein Hilfsprogramm namens `make` erleichtert.

12.1.2.2 In vier Schritten zum Programm

Zum Verständnis des Kompilierungsvorganges (und um in der Lage zu sein, dabei auftauchende Probleme zu beheben) sollten Sie etwas über die einzelnen dabei zu durchlaufenden Phasen wissen. Das Ziel ist die schrittweise Umwandlung einer in einer für einen ausgebildeten Menschen lesbaren Sprache geschriebenen Textdatei in eine maschinenlesbare Sprache umzuwandeln (lesbar auch für manche **sehr gut** ausgebildete Menschen). `gcc` führt dabei vier Programme nacheinander aus. Jedes davon ist für einen Schritt zuständig:

1. `cpp`: Der erste Schritt besteht darin, Direktiven im Quellcode (***Präprozessoren***) durch reine C-Instruktionen zu ersetzen. Im Allgemeinen bedeutet das, einen Header (`#include`) einzufügen oder ein Makro zu definieren (`#define`). Nach Beendigung dieses Schrittes besteht die Datei aus reinem C-Code.
2. `cc1`: In diesem Schritt wird der C-Code in ***Assembler*** umgewandelt. Der generierte Code ist von der verwendeten Architektur abhängig.
3. `as`: Jetzt wird aus dem Assemblercode ein ***Objektcode*** (oder auch ***Binärcode***) erzeugt. Übrig bleibt am Ende des Vorgangs eine `.o`-Datei.
4. `ld`: Im letzten Schritt (***Verlinkung***) werden alle Objekt-Dateien (`.o`) und ihre entsprechenden Bibliotheken miteinander verbunden und es entsteht eine ausführbare Datei.

12.1.3 Struktur eines Programm-Paketes

Ein korrekt aufgebautes Programmpaket der freien Software hat immer die gleiche Struktur:

- Eine Datei `INSTALL`, die den Installationsprozess beschreibt.
- Eine Datei `README`, die allgemeine Informationen zum Programm enthält (kurze Beschreibung, Autor, URL der Website, Dokumentation, hilfreiche Links, usw.). Falls keine Datei `INSTALL` vorhanden ist, findet man eine kurze Installationsbeschreibung in der Datei `README`.
- Eine Datei `COPYING`, die die Lizenz oder die Herausgabebedingungen des Programms enthält. Manchmal wird statt dessen auch eine Datei `LICENSE` mit gleichem Inhalt benutzt.
- Eine Datei `CONTRIB` oder `CREDITS`, in der einige Leute aufgelistet werden, die an der Entstehung der Software beteiligt waren (aktive Mitarbeit, sachdienliche Kommentare, Programme von Drittanbietern, usw.).
- Eine Datei `CHANGES` (oder weniger häufig `NEWS`), die neueste Änderungen und Bugfixes enthält.
- Eine Datei `Makefile` (siehe *Make*, Seite 89), die den Kompilierungsablauf steuert (notwendig für `make`). Falls diese Datei nicht bereits existiert, muss sie in einem vor der Kompilierung stattfindenden Konfigurationsprozess generiert werden.
- Sehr oft findet man eine Datei `configure` oder `Imakefile`, die die Erstellung einer neuen, auf die Erfordernisse eines bestimmten Systems abgestimmte Datei `Makefile` ermöglicht (siehe *Konfiguration*, Seite 87).
- Ein Verzeichnis, das die Quellcode-Dateien enthält und wo normalerweise am Ende des Durchbauens die neue Binärdatei abgelegt wird. Meistens trägt dieses Verzeichnis den Namen `src`.

- Ein Verzeichnis mit der Programm-Dokumentation (meist als Handbuchauszug oder Texinfo-Datei), gewöhnlich mit dem Namen `doc`.
- Manchmal findet sich auch ein Verzeichnis, das programmspezifische Dateien enthält (Konfigurationsdateien, Fall-Beispiele oder sonstige Hilfsdateien).

12.2 Entpacken

12.2.1 Das tar.gz-Archiv

Das standardmäßige Kompressionsformat[2] der UNIX®-Systeme ist das vom GNU-Projekt entwickelte `gzip`. Es wird als eines der besten allgemeinen Kompressionstools angesehen.

`gzip` wird oft mit dem Hilfsprogramm `tar` zusammen benutzt. `tar` ist ein Überbleibsel aus den grauen Vorzeiten, als die Rechnernutzer ihre Daten noch auf Bändern speicherten. Heute haben CD-ROM und DVD die Bänder längst ersetzt aber `tar` wird immer noch zur Erstellung von Archiven verwendet. So kann man beispielsweise alle Dateien eines Verzeichnisses sehr einfach in einer einzigen Datei zusammenfassen und diese dann mit `gzip` komprimieren.

Daher begegnet man freier Software oft in Form von `tar`-Archiven, die mit `gzip` komprimiert wurden und die Endung `.tar.gz` (oder kurz `.tgz`) besitzen.

12.2.2 Anwenden von GNU Tar

Zum Entpacken eines solchen Archivs wird `gzip` und anschließend `tar` benutzt. Da die GNU-Version von `tar` (`gtar`) auch *„on-the-fly“* mit `gzip` umgehen kann, ist es möglich, ein Archiv ohne bewusste Ausführung des Zwischenschrittes zu entpacken (und ohne den zusätzlichen Platz dafür zu benötigen).

Die Benutzung von `tar` folgt dieser Syntax:

```
tar [Option(en)] archiv_datei [Datei(en)]
```

Die Option `[Datei(en)]` ist nicht unbedingt nötig. Wird sie ausgelassen, wird der Befehl auf das ganze Archiv angewendet. Dieses Argument muss beim Entpacken eines `.tar.gz`-Archivs nicht spezifiziert werden.

Ein Beispiel:

```
$ tar xvfz guile-1.3.tar.gz
-rw-r--r-- 442/1002      10555 1998-10-20 07:31 guile-1.3/Makefile.in
-rw-rw-rw- 442/1002       6668 1998-10-20 06:59 guile-1.3/README
-rw-rw-rw- 442/1002       2283 1998-02-01 22:05 guile-1.3/AUTHORS
-rw-rw-rw- 442/1002      17989 1997-05-27 00:36 guile-1.3/COPYING
-rw-rw-rw- 442/1002      28545 1998-10-20 07:05 guile-1.3/ChangeLog
-rw-rw-rw- 442/1002       9364 1997-10-25 08:34 guile-1.3/INSTALL
-rw-rw-rw- 442/1002       1223 1998-10-20 06:34 guile-1.3/Makefile.am
-rw-rw-rw- 442/1002      98432 1998-10-20 07:30 guile-1.3/NEWS
-rw-rw-rw- 442/1002       1388 1998-10-20 06:19 guile-1.3/THANKS
-rw-rw-rw- 442/1002       1151 1998-08-16 21:45 guile-1.3/TODO
...
```

Einige Optionen von `tar`:

- `v` macht `tar` „geschwätzig“, d.h., es gibt die Namen aller im Archiv gefundenen Dateien auf dem Bildschirm aus. Wird diese Option weggelassen, ist die Ausgabe stumm.
- `f` ist eine vorgeschriebene Option. Ohne sie würde `tar` versuchen, ein Bandlaufgerät anstelle der Archivdatei anzusprechen (etwa das Gerät `/dev/rmt0`).
- `z` ermöglicht die Verarbeitung eines „gziped“ Archivs (mit der Dateierweiterung `.gz`). Ohne diese Option wird `tar` eine Fehlermeldung ausgeben. Bei einem nicht komprimierten Archiv muss diese Option hingegen weggelassen werden.

2. Der Standard bei den heutigen GNU/Linux-Systemen ist `bzip2`. Dieses Programm ist zwar effizienter bei der Behandlung von Textdateien, verschlingt aber auch mehr Rechenleistung. Bitte sehen Sie sich *Bzip2*, Seite 86 an, das sich speziell mit diesem Programm beschäftigt.

`tar` lässt Sie verschiedene Aktionen auf eine Archivdatei anwenden (entpacken, lesen, erstellen, erweitern...). Diese Aktionen werden mit einer Option bestimmt:

- `x`: entpackt Dateien aus dem Archiv.
- `t`: zeigt den Inhalt des Archivs an.
- `c`: erzeugt ein neues Archiv. Diese Option brauchen Sie um beispielsweise Ihre Daten zu sichern.
- `r`: fügt Dateien an ein existierendes Archiv an. Diese Option kann nicht bei komprimierten Archiven verwendet werden.

12.2.3 Bzip2

Obwohl das Format `bzip2` das ältere `gzip` im allgemeinen Gebrauch weitgehend ersetzt hat, wird aus Kompatibilitätsgründen mit älteren Systemen auch `gzip` noch sehr häufig verwendet. Nahezu die gesamte freie Software wir heute in `.tar.bz2`-Archiven vertrieben.

`bzip2` wird, was `tar` betrifft, wie `gzip` benutzt. Es muss dazu nur die Option `z` durch `j` ersetzt werden. Ein Beispiel:

```
$ tar xjvf foo.tar.bz2
```

Eine andere Möglichkeit (als Befehlskette verständlicher, aber länger!):

```
$ tar --use-compress-program=bzip2 -xvf foo.tar.bz2
```

Dazu muss `bzip2` vor der Verwendung mit `tar` auf Ihrem System in einem Verzeichnis installiert sein, das in der `PATH`-Umgebungsvariablen enthalten ist.

12.2.4 Frisch ans Werk!

12.2.4.1 Der einfachste Weg

Bevor Sie nun das Archiv entpacken sollten Sie daran denken, dies als Administrator (`root`) durchzuführen. Sie werden bei dem ganzen Unterfangen Schritte durchführen, die nur `root` erlaubt sind. Einige Schritte können zwar als normaler Anwender durchlaufen werden aber es ist einfacher, sich für den gesamten Vorgang als `root` anzumelden (auch wenn dies nicht gerade den Sicherheitsregeln entspricht).

Zuerst wechseln Sie in das Verzeichnis `/usr/local/src` und kopieren das Archiv dorthin. Dadurch werden Sie, auch Verlust des installierten Softwarepaketes, zumindest das Archiv immer wiederfinden. Wenn der verfügbare Platz zu eng bemessen ist, können Sie das Archiv nach der Installation des Programms auch auf einer Diskette speichern oder sogar ganz löschen, wenn Sie sicher sind, es im Internet wiederzufinden.

Im Normalfall wird bei der Dekomprimierung eines `tar`-Archivs ein neues Verzeichnis erstellt (Sie können das mit der Option `t` vorher testen). Gehen Sie in dieses neue Verzeichnis und schon sind Sie bereit für den nächsten Schritt.

12.2.4.2 Der sicherste Weg

UNIX®-Systeme (wozu auch GNU/Linux und FreeBSD® gehören) können als sichere Systeme gelten. Das bedeutet, dass beim Arbeiten mit nichtprivilegierten Benutzerkennzeichen, die Anwender weder Abläufe starten dürfen, die das ganze System gefährden können (etwa eine Festplatte formatieren) noch Dateien anderer Kennzeichen ändern können. Dadurch wird das System auch gegen Viren weitgehend resistent.

Dem gegenüber darf `root` alles - auch ein „böswilliges“ Programm starten. Durch den Einblick in den Quellcode der Programme können Sie – bei entsprechendem Wissen – diesen Quellcode nach verderblichem Code sowie Viren und Trojanern durchsuchen. In dieser Hinsicht ist es immer gut, vorsichtig zu sein[3].

Eine Möglichkeit ist, mit Hilfe des Befehls `adduser` ein besonderes Kennzeichen für administrative Aufgaben (etwa `free` oder `admin`) zu erstellen. Dieses Kennzeichen muss Schreibrechte in den Verzeichnissen

3. Ein Sprichwort aus der BSD-Welt sagt: „Traue keinem Programm, dessen Quellcode Du nicht hast.“

`/usr/local/src`, `/usr/local/bin` und `/usr/local/lib` sowie in allen Unterverzeichnissen von `/usr/share/man` besitzen (er mag auch berechtigt sein, Dateien an andere Stellen zu kopieren). Wir empfehlen, dieses Kennzeichen zum Eigentümer der Verzeichnisse zu machen oder noch besser eine spezielle Gruppe zu erstellen (der das Kennzeichen angehört) und dieser Gruppe Schreibrechte für die Verzeichnisse zu geben.

Nachdem diese Vorsichtsmaßnahmen getroffen wurden, kann der Vorgang wie in *Der einfachste Weg*, Seite 86 beschrieben fortgesetzt werden.

12.3 Konfiguration

Für die technisch Interessierten: der Grund, warum Entwickler ihre Software als Quellcode weitergeben, ist die ***Portierbarkeit*** der Software. Für ein UNIX®-System entwickelte Software kann mit mehr oder weniger Änderungen auf allen existierenden UNIX®-Systemen (frei oder proprietär) eingesetzt werden. Daher ist vor dem Durchbauen eine Konfiguration erforderlich.

Es existieren mehrere verschiedene Konfigurationssysteme. Sie müssen jeweils das System verwenden, das der Autor der Software vorschreibt (das können manchmal auch mehrere sein). Im Allgemeinen können Sie:

- AutoConf verwenden (siehe *Autoconf*, Seite 87), wenn sich eine Datei `configure` im Hauptverzeichnis des Programmpaketes befindet.
- imake verwenden (siehe *Imake*, Seite 89), wenn sich eine Datei `Imakefile` im Hauptverzeichnis des Programmpaketes befindet.
- Ein Shell-***Script*** verwenden (etwa `install.sh`), falls das in der Datei `INSTALL` oder `README` so angewiesen wird.

12.3.1 Autoconf

12.3.1.1 Das Prinzip

AutoConf wird zur korrekten Einrichtung des Programms benutzt. Es erstellt die für das Durchbauen des Programms nötigen Dateien (u.A. `Makefile`) und kann unter Umständen den Quellcode direkt beeinflussen (beispielsweise durch eine Datei `config.h.in`).

Die Arbeitsweise von AutoConf ist einfach:

- Der Entwickler weiß genau, welche Tests zur Konfiguration seiner Software benötigt werden (etwa: „welche Version dieser oder jener ***Bibliothek*** benutzen Sie?“). Er schreibt diese Tests nach einer präzisen Syntax in eine Datei namens `configure.in`.
- Dann führt er das Programm AutoConf aus, das aus dieser Datei ein Konfigurationsscript `configure` erstellt. Dieses Script führt die nötigen Tests bei der Konfiguration der Software durch.
- Der Benutzer startet das Script und AutoConf konfiguriert nun alles, was zum Durchbauen notwendig ist.

12.3.1.2 Beispiel

Ein Beispiel der Arbeitsweise von AutoConf:

```
$ ./configure
loading cache ./config.cache
checking for gcc... gcc
checking whether the C compiler (gcc  ) works... yes
checking whether the C compiler (gcc  ) is a cross-compiler... no
checking whether we are using GNU C... yes
checking whether gcc accepts -g... yes
checking for main in -lX11... yes
checking for main in -lXpm... yes
checking for main in -lguile... yes
checking for main in -lm... yes
checking for main in -lncurses... yes
checking how to run the C preprocessor... gcc -E
```

```
checking for X... libraries /usr/X11R6/lib, headers /usr/X11R6/include
checking for ANSI C header files... yes
checking for unistd.h... yes
checking for working const... yes
updating cache ./config.cache
creating ./config.status
creating lib/Makefile
creating src/Makefile
creating Makefile
```

Zur besseren Kontrolle über das, was `configure` generiert, können über die Kommandozeile oder die Umgebungsvariablen einige Optionen hinzugefügt werden. Beispiel:

```
$ ./configure --with-gcc --prefix=/opt/GNU
```

oder (durch die shell – unter der bash etwa):

```
$ export CC=‘which gcc‘
$ export CFLAGS=-O2
$ ./configure --with-gcc
```

oder:

```
$ CC=gcc CFLAGS=-O2 ./configure
```

12.3.1.3 Was tun... wenn es nicht funktioniert?

Eine typische Fehlermeldung sieht so aus: `configure: error: Cannot find library guile` (die meisten Fehlermeldungen des `configure`-Skripts sehen so aus).

Das bedeutet, dass das `configure`Skript eine Bibliothek nicht finden konnte (im Beispiel die Bibliothek `guile`). Vereinfacht gesagt, kompiliert das `configure`-Skript ein kurzes Testprogramm, das diese Bibliothek benötigt. Kann das Testprogramm nicht kompliliert werden, wird auch das Durchbauen der Software misslingen. Daher wird eine Fehlermeldung ausgegeben.

- Den Grund für die jeweilige Fehlermeldung finden Sie am Ende der Datei `config.log`, in der alle Schritte der Konfiguration protokolliert werden. Da der C-Compiler in seinen Meldungen sehr klar und deutlich ist, wird die Logdatei Ihnen normalerweise bei der Behebung des Problems helfen können.
- Prüfen Sie, ob die gesuchte Bibliothek richtig installiert ist. Falls nicht, installieren Sie sie (aus dem Quellcode oder einer Binärdatei) und starten Sie `configure` neu. Sie können bei der Suche nach der Bibliothek effizient vorgehen indem Sie nach der Datei mit den Symbolen der Bibliothek suchen (eine Datei mit dem Namen `lib<name>.so`). Geben Sie z.B. ein

  ```
  $ find / -name 'libguile*'
  ```

  oder:

  ```
  $ locate libguile
  ```

- Prüfen Sie ob die Bibliothek zugänglich für den Compiler ist. D.h., sie sollte in den Verzeichnissen `/usr/lib`, `/lib` oder `/usr/X11R6/lib` liegen (oder in denen, die in der Umgebungsvariablen `LD_LIBRARY_PATH` angegeben sind, siehe *Was tun ... wenn es nicht funktioniert?*, Seite 91, 5.b). Prüfen Sie, ob diese Datei eine Bibliothek ist: `file libguile.so`.
- Prüfen Sie, ob die zu der Bibliothek passenden Header an der richtigen Stelle installiert sind (normalerweise in `/usr/include`, `/usr/local/include` oder `/usr/X11R6/include`). Sind Sie nicht sicher, welche Header Sie benötigen, prüfen Sie einfach, ob Sie die Entwicklungs-Version der entsprechenden Bibliothek installiert haben (beispielsweise `libgtk+2.0-devel` anstelle von `libgtk+2.0`). Diese Entwicklungs-Version beinhaltet die „include“-Dateien der Bibliothek, die für die Kompilierung benötigt werden.
- Schließlich prüfen Sie noch, ob auf der Festplatte noch genug freier Platz vorhanden ist (das `configure`-Skript benötigt etwas Platz für temporäre Dateien). Lassen Sie sich mit `df -h` die Partitionen Ihres Systems anzeigen und achten Sie auf belegte oder fast belegte Partitionen.

Falls Sie die Fehlermeldungen der Datei `config.log` nicht verstehen, wenden Sie sich vertrauensvoll an die Benutzergemeinschaft der freien Software, wo Sie jederzeit Hilfe finden (siehe *Technische Unterstützung*, Seite 96).

Falls `configure` darauf besteht, dass eine Bibliothek nicht vorhanden ist, sollten Sie trotzdem prüfen, ob die entsprechende Datei tatsächlich nicht exisitert (es wäre z.B. sehr ungewöhnlich, dass auf Ihrem System keine `curses`-Bibliothek vorhanden ist). In diesem Fall kann es auch sein, dass die Variable `LD_LIBRARY_PATH` unvollständig oder falsch ist!

12.3.2 Imake

imake ermöglicht die Konfiguration freier Software durch die Erstellung einer `Makefile`-Datei nach einfachen Regeln. Diese Regeln legen fest, welche Dateien zur Kompilierung der Binärdatei benötigt werden. imake generiert die entsprechende `Makefile`-Datei. Die Regeln werden in einer Datei namens `Imakefile` festgelegt.

Das Interessante an imake ist seine Fähigkeit, Informationen zu verwerten, die ***site-dependent*** (architekturabhängig) sind. Das ist sehr hilfreich, für Anwendungen unter X Window System. imake wird jedoch auch für Anwendungen auf anderen Systemen benutzt.

Man nutzt imake auf die einfachste Art, indem man in das Hauptverzeichnis des entpackten Archivs wechselt und dort das Skript `xmkmf` aufruft. Dieses startet das imake-Programm:

```
$ xmkmf -a
$ imake -DUseInstalled -I/usr/X11R6/lib/X11/config
$ make Makefiles
```

Falls ein Fehler auftritt, rekompilieren und installieren Sie X11R6 neu!

12.3.3 Verschiedene Shell-Skripte

Die nötigen Informationen finden Sie in den Dateien `INSTALL` oder `README`. Normalerweise müssen Sie ein Skript mit dem Namen `install.sh` oder `configure.sh` starten. Das dann ablaufende Skript ist entweder autonom (es sucht sich selbst, was es braucht) oder interaktiv, d.h., es fragt Sie nach Informationen über Ihr System (z.B. nach Pfaden).

Wenn Sie die Datei, die gestartet werden soll, nicht finden, geben Sie in der bash einfach die Zeichen `./` ein und drücken dann zweimal auf **TAB**. Die bash wird (in der Standardeinstellung) dann den angefangenen Befehl automatisch mit einer möglichen ausführbaren Datei (also auch einem möglichen Konfigurationsskript) aus dem aktuellen Verzeichnis vervollständigen. Falls es in dem Verzeichnis mehrere ausführbare Dateien existieren bekommen Sie eine Liste gezeigt, aus der Sie nur noch die richtige Datei auswählen müssen.

Ein weiterer Sonderfall ist die Installation von perl-Modulen. Das geschieht durch die Ausführung eines in perl geschriebenen Konfigurationsskripts. Normalerweise lautet der dazu nötige Befehl:

```
$ perl Makefile.PL
```

12.3.4 Alternativen

Es gibt auch Pakete mit freier Software, die schlecht strukturiert sind, speziell in den ersten Phasen der Entwicklung (der Benutzer wird jedoch davor gewarnt!). Sie verlangen manchmal das „manuelle“ Ändern der Konfigurationsdateien. Meist sind das die Dateien `Makefile` (siehe *Make*, Seite 89) und `config.h` (dieser Name ist nur eine Konvention).

Wir raten Ihnen von diesen Manipulationen ab, es sei denn, Sie wissen genau, was Sie tun. Dazu gehört eine Menge Wissen und etwas Motivation, aber auch hier macht die Übung den Meister.

12.4 Kompilierung

Jetzt, nachdem die Software richtig konfiguriert wurde, muss sie nur noch kompiliert werden. Diese Phase ist normalerweise sehr einfach und beinhaltet keine ernsthaften Probleme.

12.4.1 Make

Das bevorzugte Werkzeug der freien Software-Gemeinde für das Durchbauen von Quellcode ist `make`. Es hat zwei Vorteile:

- Der Entwickler spart Zeit, da `make` einen effizienten Ablauf des Durchbauens eines Projektes ermöglicht.
- Der Benutzer kann (selbst ohne jegliche Programmier-Vorkenntnisse) die Software mit ein paar Kommandozeilen erstellen und installieren.

Die auszuführenden Aktionen zum Durchbauen des Quellcodes sind in einer Datei mit namens `Makefile` oder `GNUMakefile` gespeichert. Wenn der Befehl `make` aufgerufen wird, liest er diese Datei – wenn sie im aktuellen Verzeichnis existiert. Falls nicht, kann die Datei mittels der `make`-Option `-f` spezifiziert werden.

12.4.2 Regeln

`make` arbeitet unter Berücksichtigung eines Systems von ***Abhängigkeiten***. Also besteht die Kompilierung einer Binärdatei (eines „ ***Target*** “) aus dem Abarbeiten mehrerer Phasen („Abhängigkeiten“). Um beispielsweise die (imaginäre) Binärdatei `glloq` zu erstellen, müssen die Objektdateien `main.o` und `init.o` (Zwischenstufen der Kompilierung) kompiliert und danach verbunden werden. Diese Objektdateien sind ebenfalls *Targets*, deren Abhängigkeiten in den entsprechenden Quellcode-Dateien bestehen.

Dieses Dokument ist nur eine kurze Einführung zum Überleben in der gnadenlosen Welt von `make`. Eine umfassende Dokumentation finden Sie in *Managing Projects with Make*, 2. Ausgabe, **O'Reilly**, von Andrew Oram und Steve Talbott.

12.4.3 Los gehts!

Im Allgemeinen folgt die Benutzung von `make` mehreren Konventionen. Beispiele:

- Ohne Argument führt `make` nur die Kompilierung des Programms ohne anschließende Installation aus.
- Der Befehl `make install` bewirkt die Kompilierung des Programms (aber nicht immer) und die anschließende Installation der benötigten Dateien an den richtigen Stellen des Dateisystems. Manche Dateien werden nicht immer korrekt installiert (`man`, `info`) und müssen vom Benutzer selbst an die entsprechenden Stellen kopiert werden. Manchmal muss `make install` nochmals in Unterverzeichnissen durchgeführt werden. Das ist gewöhnlich bei Software von Drittanbietern der Fall.
- Der Befehl `make clean` entfernt alle temporären Dateien, die während der Kompilierung erstellt wurden, sowie in den meisten Fällen auch die ausführbare Datei.

Der erste Schritt ist die Kompilierung des Programms, also (für eine imaginäres Beispiel):

```
$ make
gcc -c glloq.c -o glloq.o
gcc -c init.c -o init.o
gcc -c main.c -o main.o
gcc -lgtk -lgdk -lglib -lXext -lX11 -lm glloq.o init.o main.o -o glloq
```

Gut, die Binärdatei wurde korrekt erstellt. Wir können also zum nächsten Schritt übergehen: die Installation der Dateien des Programm-Pakets (Binärdateien, Datendateien, usw.). Siehe *Installation*, Seite 95.

12.4.4 Erläuterungen

Wenn Sie neugierig genug sind, sich näher mit dem `Makefile` zu beschäftigen, werden Sie darin sowohl bekannte Befehle (`rm`, `mv`, `cp`, usw.) als auch ungewohnte Ausdrücke, wie z.B. `$(CFLAGS)`, finden.

Das sind ***Variablen***, die gewöhnlich am Anfang der Datei `Makefile` definiert werden und denen später ein entsprechender Wert zugewiesen wird. Variablen sind sehr hilfreich, wenn Sie die gleichen Optionen mehrfach hintereinander benutzen wollen.

Zur Ausgabe der Zeichenfolge „`foo`“ auf dem Bildschirm unter Verwendung von `make all` schreibt man:

```
TEST = foo
all:
```

```
echo $(TEST)
```

In den meisten Fällen werden folgende Variablen gesetzt:

1. CC: Der Compiler. Normalerweise ist das cc, in den meisten Systemen ein Synonym für gcc. Im Zweifelsfall sollten Sie hier gcc einsetzen.
2. LD: Das Programm, das in der letzten Kompilierungsphase eingesetzt wird (siehe *In vier Schritten zum Programm*, Seite 84). Standardmäßig ist das ld.
3. CFLAGS: Die zusätzlichen Argumente, die dem Compiler in den ersten Kompilierungsphasen übergeben werden. Dazu gehören u.A.:
   - -I<path>: Informiert den Compiler, wo er nach zusätzlichen Header-Dateien suchen soll (beispielsweise bewirkt -I/usr/X11R6/include die Integration der Header-Dateien im Verzeichnis /usr/X11R6/include).
   - -D<symbol>: Definiert ein zusätzliches Symbol, hilfreich für Programme, deren Kompilierung von definierten Symbolen abhängt (Beispiel: die Datei string.h wird durch HAVE_STRING_H eingebunden).

   Oft sieht man bei der Kompilierung Zeilen wie die folgende:
   ```
   $(CC) $(CFLAGS) -c foo.c -o foo.o
   ```
4. LDFLAGS (oder LFLAGS): Diese Argumente werden im letzten Schritt der Kompilierung verwendet. Dazu gehören (u.A.):
   - -L<path>: Spezifiziert einen zusätzlichen Pfad für die Suche nach Bibliotheken (z.B.: -L/usr/X11R6/lib).
   - -l<library>: Spezifiziert eine zusätzliche Bibliothek, die im letzten Kompilierungsschritt verwendet werden soll.

12.4.5 Was tun ... wenn es nicht funktioniert?

Keine Panik! Das kann Jedem passieren. Die am häufigsten auftretenden Gründe:

1. `glloq.c:16: decl.h: No such file or directory`

   Der Compiler konnte die entsprechende Header-Datei nicht finden. Eigentlich hätte dieser Fehler bereits bei der Konfiguration auftreten sollen. Die Lösung des Problems:

   - Prüfen Sie, ob die Header-Datei in einem der folgenden Verzeichnisse existiert: /usr/include, /usr/local/include, /usr/X11R6/include oder in einem ihrer Unterverzeichnisse. Falls nicht, suchen Sie die Datei auf der gesamten Festplatte (benutzen Sie dazu find oder locate). Falls sie auch jetzt nicht gefunden wird, prüfen Sie, ob die entsprechende Bibliothek installiert ist. Beispiele für die Anwendung von find und locate finden Sie in den entsprechenden Handbuchauszügen.
   - Prüfen Sie, ob die Header-Datei auch lesbar für den Compiler ist (testen Sie das mit dem Befehl less <path>/<file>.h).
   - Falls sich die Datei in einem Verzeichnis wie /usr/local/include oder /usr/X11R6/include befindet, muss man dem Compiler manchmal ein neues Argument übergeben. Öffnen Sie dazu die entsprechende Datei Makefile mit Ihrem bevorzugten Editor, z.B. Emacs, Vi, usw. (beachten Sie, dass Sie die richtige Datei öffnen, nämlich die Datei in dem Verzeichnis, in dem die Kompilierung fehlschlug[4]). Suchen Sie die fehlerhafte Zeile und fügen Sie die Zeichenkette -I<path> hinter dem Aufruf des Compilers (gcc oder manchmal auch $(CC)) ein, wobei <path> der Pfad zum Verzeichnis ist, in dem die gesuchte Header-Datei liegt. Falls Sie nicht wissen, wo Sie diese Option einfügen sollen, stellen Sie sie an den Anfang der Datei, direkt hinter die Zeilen CFLAGS=<irgendwas> oder CC=<irgendwas>.

4. Sehen Sie sich die Fehlermeldung von make genau an. Normalerweise enthalten die letzten Zeilen den Namen eines Verzeichnisses (z.B.: `make[1]: Leaving directory '/home/franz/Project/foo'`). Wählen Sie das Verzeichnis in der Zeile mit der höchsten Nummer. Prüfen Sie, ob Sie das richtige Verzeichnis gewählt haben, durch einen erneuten Aufruf von make in diesem Verzeichnis. Es sollte die gleiche Fehlermeldung erscheinen.

- Starten Sie `make` erneut. Falls es immer noch nicht funktioniert, prüfen Sie, ob diese Option (siehe letzter Absatz) während der Kompilierung in der fehlerhaften Zeile hinzugefügt wurde.
- Wenn sich jetzt immer noch kein Erfolg einstellt, suchen Sie Hilfe bei Ihrem örtlichen GNU/Linux-Guru oder bei der Gemeinschaft der Nutzer freier Software (siehe *Technische Unterstützung*, Seite 96).

2. `glloq.c:28: ‘struct foo’ undeclared (first use this function)`

   In der Struktur werden spezielle Datentypen verwendet, die von allen Programmen benutzt werden. Viele davon werden vom System in Header-Dateien definiert. Das bedeutet, dass das Problem höchstwahrscheinlich durch eine fehlende oder falsch benutzte Header-Datei verursacht wird. Die korrekte Vorgehensweise zur Lösung dieses Problems ist:

   - Versuchen Sie festzustellen, ob der in Frage kommende Konstrukt vom Programm selbst oder vom System definiert wird. Dabei hilft Ihnen der Befehl grep, mit dem Sie die Header-Dateien durchsuchen können.

     Angenommen, Sie befinden sich im Hauptverzeichnis des Programm-Paketes:

     ```
     $ find . -name '*.h'| xargs grep 'struct foo' | less
     ```

     Daraufhin können unter Umständen sehr viele Zeilen auf dem Bildschirm erscheinen (etwa eine Zeile für jedes Mal, bei dem eine Funktion definiert wird, die den Konstrukt benutzt). Sehen Sie sich die Header-Dateien an, die Ihnen grep als Resultat liefert und suchen Sie – falls sie existiert – die Zeile, in der der Konstrukt definiert wird.

     Die Definition eines solchen Konstrukts sieht so aus:

     ```
     struct foo {
         <inhalt des konstrukts>
     };
     ```

     Prüfen Sie, ob die Zeile dem entspricht, was Sie suchen. Falls ja, bedeutet das, dass die Header-Datei nicht in der defekten `.c`-Datei genannt wird. Für diesen Fall gibt es zwei Lösungen:

     - fügen Sie die Zeile `#include "<dateiname>.h"` am Anfang der `.c`-Datei ein,
     - oder kopieren Sie per Kopieren/Einfügen die Definition selbst am Anfang der Datei ein (das ist nicht gerade sauberer Stil, es funktioniert aber im Allgemeinen).

   - Wenn Sie die gesuchte Stelle nicht finden, setzen Sie die Suche in gleicher Weise bei den System-Header-Dateien fort (die im Allgemeinen in den Verzeichnissen `/usr/include`, `/usr/X11R6/include` oder `/usr/local/include` liegen). Dabei benutzen Sie als Suchkriterium jetzt die Zeile `#include <<filename>.h>`.
   - Falls dieser Konstrukt auch an diesen Stellen nicht existiert, so versuchen Sie herauszufinden, in welcher Bibliothek (d.h., in welcher Funktionen-Sammlung) er definiert werden sollte (in den Dateien `INSTALL` oder `README` sehen Sie, welche Bibliotheken in welchen Versionen vom Programm benutzt werden). Falls die vom Programm benötigte Version nicht auf Ihrem System installiert ist, müssen Sie sie installieren.
   - Falls Sie damit auch keinen Erfolg haben, stellen Sie fest, ob das Programm überhaupt auf der von Ihnen benutzten Architektur lauffähig ist. Einige Programme wurden noch nicht auf alle UNIX®-Systeme portiert. Prüfen Sie nochmals nach, ob Sie während der Konfiguration (`configure`) das Programm richtig für die von Ihnen benutzte Architektur eingerichtet haben.

3. `parse error`

   Dieses Problem ist schwer zu lösen, da es oft durch eine bestimmte Zeile ausgelöst wird, nachdem der Compiler diese Zeile wieder verlassen hat. Manchmal liegt es einfach an einem nicht definierten Datentyp. Wenn Sie eine Fehlermeldung wie diese erhalten:

   ```
   main.c:1: parse error before ‘glloq_t
   main.c:1: warning: data definition has no type or storage class
   ```

   besteht das Problem darin, dass der Datentyp `glloq_t` nicht definiert wurde. Die Lösung dieses Problems gleicht mehr oder weniger der Lösung des vorangegangenen Problems.

In den alten curses-Bibliotheken, gab es einen `parse error` .

4. `no space left on device`

   (Deutsch: „Kein Platz mehr auf dem Gerät verfügbar“). Dieses Problem kann leicht gelöst werden: es besteht darin, dass auf der Festplatte kein ausreichender freier Platz zur Erzeugung der Binärdatei vorhanden ist. Die Lösung ist: schaffen Sie freien Platz auf der Partition, die das Installationsverzeichnis enthält (löschen Sie temporäre Dateien oder Quellcodes, de-installieren Sie Programme, die Sie nicht benutzen, usw., falls Sie das Programm-Paket in `/tmp` entpackt haben, anstatt `/usr/local/src` zu benutzen, was eine unnötige Unordnung auf der `/tmp`-Partition vermeidet. Prüfen Sie, ob es auf Ihrer Festplatte `core`-***Dateien***[5] gibt. Wenn ja, löschen Sie sie oder veranlassen Sie die Löschung, falls die Dateien einem anderen Benutzer gehören.

5. `/usr/bin/ld: cannot open -lglloq: No such file or directory`

   Das bedeutet klar und deutlich, dass das Programm `ld` (aufgerufen von `gcc` während der letzten Phase der Durchbauens) eine Bibliothek nicht finden kann. Zum Einbinden einer Bibliothek sucht `ld` nach einer Datei, deren Name in den Argumenten des Typs `-l<library>` genannt wird. Diese Datei ist `lib<library>.so`. Wenn `ld` sie nicht finden kann, wird eine Fehlermeldung erzeugt. Das Problem kann mit den folgenden Maßnahmen beseitigt werden:

   a. Vergewissern Sie sich mit dem Befehl `locate`, dass die Datei auf Ihrer Festplatte vorhanden ist. Normalerweise liegen die grafischen Bibliotheken im Verzeichnis `/usr/X11R6/lib`. Ein Beispiel für die Suche:

      ```
      $ locate libglloq
      ```

      Wenn diese Suche erfolglos ist, versuchen Sie es mit dem Programm `find` (etwa: `find /usr -name "libglloq.so*"`). Falls das auch kein Resultat bringt müssen Sie die Bibliothek installieren.

   b. Wenn die Bibliothek gefunden wird, prüfen Sie nach, ob sie von dem Befehl `ld` gelesen werden kann: die Datei `/etc/ld.so.conf` spezifiziert die Verzeichnisse, in denen diese Bibliotheken liegen. Fügen Sie das bei der Suche gefundene Verzeichnis am Ende der Datei hinzu (möglicherweise müssen Sie danach Ihren Rechner neu starten um dem System die Änderung bekannt zu machen). Sie können das Verzeichnis auch in die Umgebungsvariable `LD_LIBRARY_PATH` integrieren. Wenn das gefundene Verzeichnis beispielsweise `/usr/X11R6/lib` heißt, geben Sie ein:

      ```
      export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/X11R6/lib
      ```

      (wenn Sie die bash als Shell benutzen).

   c. Wenn Sie jetzt immer noch keinen Erfolg haben, prüfen Sie mit dem Kommando `file` ob die Bibliothek eine ausführbare Datei ist (oder ELF). Falls es ein symbolischer Link ist prüfen Sie, ob der Link korrekt ist und auf eine existierende Datei zeigt (mit dem Befehl `nm libglloq.so` beispielsweise). Auch die Berechtigungen können falsch gesetzt sein (wenn Sie nicht den `root`-Account benutzen und die Bibliothek beispielsweise gegen das Lesen geschützt ist).

6. `glloq.c(.text+0x34): undefined reference to ‘glloq_init’`

   Dies ist ein durch ein Symbol verursachtes Problem, das während der letzten Kompilierungsphase nicht gelöst wurde. Normalerweise ist das ein Bibliotheken-Problem für das es mehrere mögliche Gründe geben kann:

   - Das Erste, was man hier machen sollte, ist herauszufinden, ob das Symbol normalerweise in eine Bibliothek **gehört**. Wenn es zum Beispiel ein Symbol ist, dessen Name mit `gtk` beginnt, so gehört es in

5. Dateien, die vom System erzeugt werden, wenn ein Prozess einen Speicherbereich belegen will, den er nicht belegen darf. Diese Dateien werden dazu verwendet, den Grund des falschen Verhaltens zu finden und den Fehler zu beseitigen.

eine gtk-Bibliothek. Wenn der Name der in Frage kommenden Bibliothek so einfach zu identifizieren ist wie z.B. frobnicate_foobar, können Sie die Symbole der Bibliothek mit dem Befehl nm auflisten:

```
$ nm libglloq.so
0000000000109df0 d glloq_message_func
000000000010a984 b glloq_msg
0000000000008a58 t glloq_nearest_pow
0000000000109dd8 d glloq_free_list
0000000000109cf8 d glloq_mem_chunk
```

Mit der Option -o des Befehls nm wird der Name der Bibliothek in jeder Zeile angezeigt, was die Suche sehr erleichtert. Angenommen, wir suchen das Symbol bulgroz_max, so kann eine simple Lösung so aussehen:

```
$ nm /usr/lib/lib*.so | grep bulgroz_max
$ nm /usr/X11R6/lib/lib*.so | grep bulgroz_max
$ nm /usr/local/lib/lib*.so | grep bulgroz_max
/usr/local/lib/libfrobnicate.so:000000000004d848 T bulgroz_max
```

Wunderbar! Das Symbol bulgroz_max wird in der Bibliothek frobnicate definiert (der Großbuchstabe T steht vor dem Namen). Nun brauchen Sie nur die Zeichenfolge -lfrobnicate in die Zeile des Makefiles einzufügen: hängen Sie sie an das Ende der Zeile, in der LDFLAGS oder LFGLAGS (oder im Schlimmsten Fall CC) definiert werden, oder an die Zeile, die die endgültige Binärdatei erstellt.

- Die Kompilierung der Software wurde mit einer Version der Bibliothek versucht, die für diese Software nicht zugelassen ist. In der Datei README oder INSTALL des Programm-Pakets sehen Sie, welche Version benutzt werden muss.
- Es wurden nicht alle Objekt-Dateien des Programm-Pakets sauber eingebunden. Es fehlt die Datei, in der diese Funktion definiert wird. Geben Sie nm -o *.o ein und prüfen Sie, welche Datei fehlt. Fügen Sie die entsprechende .o-Datei in die Kompilierungszeile ein, falls sie dort fehlt.
- Die problematische Funktion oder Variable existiert möglicherweise nicht. Versuchen Sie, sie zu entfernen: editieren Sie dazu die entsprechende Quellcode-Datei, deren Name am Anfang der Fehlermeldung genannt wird. Dies ist eine verzweifelte Lösung, deren Konsequenz sehr wahrscheinlich in einem chaotischen Verhalten des Programms bestehen wird (etwa ein Speicherzugriffsfehler – ***segfault*** – beim Start, usw.).

7. Segmentation fault (core dumped)

   Manchmal hängt der Compiler sofort nach Beginn und gibt diese Fehlermeldung aus. Dazu habe ich keinen Rat ausßer dem Vorschlag, eine neuere Version des Compilers zu installieren.

8. no space on /tmp

   Die Kompilierung benötigt temporären freien Platz während der verschiedenen Phasen. Ist dieser nicht vorhanden so misslingt das Vorhaben. Also sollten Sie die Partition säubern. Aber beachten Sie dabei, dass einige laufende Programme (X-Server, Pipes, etc) durch das entfernen benötigter Dateien abgebrochen werden können. Sie müssen also genau wissen, was Sie tun! Falls /tmp nicht auf einer eigenen Partition liegt (beispielsweise auf der Root-Partition) können Sie eventuell vorhandene core-Dateien suchen und entfernen.

9. make/configure in unendlicher Wiederholung

   Oft besteht das Problem in der Zeit des Systems. make muss tatsächlich die Zeit des Rechners kennen sowie die Zeitstempel der benötigten Dateien. Es vergleicht die Zeitstempel und weiß durch das Resultat, ob etwa das *Target* jünger als die abhängige Datei ist.

   Ein paar Zeitprobleme können make dazu bringen, sich endlos immer wieder neu zu erstellen (oder Unterverzeichnisse in unendliche Wiederholung anzulegen). In solch einem Fall ist touch die Lösung. Man gibt damit den entsprechenden Dateien den aktuellen Zeitstempel, womit normalerweise das Problem gelöst ist.

   Ein Beispiel:

   ```
   $ touch *
   ```

Oder auch (simpel, aber wirksam):

```
$ find . | xargs touch
```

12.5 Installation

12.5.1 Mit Make

Nun, da das Erstellen erledigt ist, müssen die Dateien in die passenden Verzeichnisse kopiert werden (üblicherweise in ein Unterverzeichnis von `/usr/local`).

Meistens kann `make` diese Aufgabe übernehmen. Eines der Targets von `make` ist `install`. Der Befehl `make install` führt die Installation der benötigten Dateien aus.

Normalerweise wird der Vorgang in der Datei `INSTALL` oder `README` beschrieben. Aber es kommt vor, dass der Entwickler keine solche Datei liefert. In diesem Fall müssen Sie alles selbst installieren.

Kopieren Sie:

- die ausführbaren Dateien (Programme) in das Verzeichnis `/usr/local/bin`.
- die Bibliotheken (`lib*.so`-Dateien) in das Verzeichnis `/usr/local/lib`.
- die Header-Dateien (`*.h`) in das Verzeichnis `/usr/local/include` (achten Sie darauf, die Originale nicht zu löschen).
- Die Datendateien gehören üblicherweise in das Verzeichnis `/usr/local/share`. Wenn Sie die Installationsprozedur nicht kennen, versuchen Sie das Programm zu starten ohne die Datendateien zu kopieren. Das Programm wird Sie zu gegebener Zeit danach fragen (zum Beispiel in einer Fehlermeldung wie dieser: `Cannot open /usr/local/share/glloq/data.db`). Dann kopieren Sie die Dateien dorthin, wo das Programm sie sucht.
- Mit der Dokumentation muss ein wenig anders verfahren werden:
  - Die Handbuchauszüge gehören üblicherweise in ein Unterverzeichnis von `/usr/local/man`. Im Allgemeinen besitzen diese Dateien das `troff`- oder `groff`-Format und eine Ziffer als Namenserweiterung. Der Name ist der Name eines Befehls (etwa `echo.1`). Ist die Ziffer `n`, so kopieren Sie die Datei in das Verzeichnis `/usr/local/man/man<n>`.
  - Die `info`-Dateien werden in das Verzeichnis `/usr/info` oder `/usr/local/info` kopiert.

Fertig! Gratulation! Jetzt können Sie ein komplettes Betriebssystem durchbauen!

12.5.2 Probleme

Wenn Sie gerade eine freie Software installiert haben, beispielsweise GNU `tar`, und es wird beim Aufruf des Programms ein anderes Programm gestartet oder das Programm benimmt sich nicht so, wie während des Tests direkt aus dem Quell-Verzeichnis, so kann das ein Problem der Variablen `PATH` sein, die ein anderes, gleichlautendes Programm in einem Verzeichnis findet, das vor demjenigen aufgeführt ist, in dem Sie die neue Software installiert haben. Prüfen Sie das mit dem Befehl `type -a <program>`.

Zur Lösung des Problems kann man das Verzeichnis der installierten Software in der `PATH`-Variablen höher ansiedeln und/oder die Dateien, die ungewollt aufgerufen wurden, löschen oder umbenennen und/oder Ihr neues Programm umbenennen (in diesem Beispiel `gtar`), so dass es zu keiner Verwechslung kommt.

Wenn die von Ihnen benutzte Shell dafür geeignet ist, können Sie auch ein Alias einrichten (so dass `tar` dann z.B. `/usr/local/bin/gtar` heißt).

12.6 Support

12.6.1 Dokumentation

Verschiedene Quellen:

- HOWTOs, kurze Dokumente zu einzelnen Punkten (normalerweise nicht das, was wir hier brauchen, aber manchmal recht hilfreich). Sie finden eine Sammlung – sofern installiert – auf Ihrer Festplatte im Verzeichnis `/usr/share/doc/HOWTO` (nicht immer, manchmal liegen sie woanders. Suchen Sie mit `locate HOWTO` danach.),
- Die Handbuchauszüge. Geben Sie `man <command>` ein und die Dokumentation zum Befehl `<command>` wird angezeigt,
- Spezielle Literatur zum Thema. Einige große Verlage haben Bücher über freie Software herausgegeben (besonders über GNU/Linux). Bücher sind gerade für Einsteiger sehr hilfreich, wenn Sie noch nicht alle Begriffe dieser Dokumentation verstehen.

12.6.2 Technische Unterstützung

Falls Sie eine „offizielle“ Mandrakelinux-Distribution mit Support-Option gekauft haben, können Sie die angebotenen Support-Leistungen in Anspruch nehmen.

Sie können sich auf jeden Fall auf die Hilfe der Benutzergemeinschaft der freien Software verlassen:

- ***Newsgroups*** (im Usenet) comp.os.linux.* (`news:comp.os.linux.*`) beantworten alle Fragen zu GNU/Linux. Die Diskussionsforen der `comp.os.bsd.*`-Hierarchie beschäftigen sich mit BSD-Systemen. Es gibt noch einige Newsgroups, die sich mit anderen UNIX®-Systemen beschäftigen. Beachten Sie bitte, dass Sie erst einige Zeit in den Newsgroups lesen sollten, bevor Sie dort Fragen stellen.
- Einige Vereine und Gruppen von Enthusiasten der freien Software-Gemeinschaft bieten freiwillige Unterstützung an. Diese finden Sie am einfachsten, indem Sie die einschlägigen Webseiten und Newsgroups durchforsten.
- Mehrere ***IRC-Kanäle*** bieten eine sofortige (aber anonyme) Hilfe, manchmal sogar von richtigen ***Gurus***. Da ist zum Beispiel der Kanal `#linux` auf den meisten IRC-Servern sowie `#linuxhelp` bei IRCNET.
- Als allerletzten Ansprechpartner kann man auch den Entwickler der Software selbst kontaktieren (so er denn seinen Namen und seine ***Mail***Adresse in einer Datei des Software-Pakets hinterlassen hat) wenn man sicher ist, einen Bug gefunden zu haben. Allerdings kann der Fehler auch an der Architektur liegen, aber eigentlich sollte freie Software portierbar sein, nicht wahr?

12.6.3 Wie findet man freie Software?

Auf diese Frage geben eine Menge Links eine ausgiebige Antwort:

- die unerschöpfliche FTP-Site sunsite.unc.edu (`sunsite.unc.edu`) oder einer ihrer Spiegelserver
- die folgenden Websites bieten einen Katalog mit vielen freien Programmen für UNIX®-Systeme (allerdings finden Sie dort auch proprietäre Programme):
  - FreshMeat (`http://www.freshmeat.net/`) ist vermutlich die umfassendste Sammlung,
  - SourceForge.net (`http://sourceforge.net/`) ist die weltweit größte Website für Open Source Software-Entwicklung mit der umfangreichsten Sammlung von Open Source Code und Programmen im Internet.
  - GNU Software (`http://www.gnu.org/software/`) bietet eine umfassende Sammlung aller GNU-Programme. Natürlich sind das alles freie Programme und die meisten unterliegen der GPL.

- Sie können natürlich auch eine Suche mit einer Suchmaschine wie Google™/ (`http://www.google.com/`) und Lycos/ (`http://www.lycos.com/`) mit Suchbegriffen wie `+<software>+download` oder `"download software"` starten.

12.7 Danksagungen

- Proof-Reader und Lieferanten kritischer Kommentare (in alphabetischer Reihenfolge): Sébastien Blondeel, Mathieu Bois, Xavier Renaut und Kamel Sehil.
- ***Beta-Tester***: Laurent Bassaler
- Englische Übersetzung: Fanny Drieu; Englische Überarbeitung: Hoyt Duff

Kapitel 13. Konfigurieren und Installieren neuer Kerne

Neben dem Einhängen von Dateisystemen und dem Erstellen von Programmen aus dem Quellcode ist das eigene Kompilieren eines GNU/Linux-Kerns zweifellos die schwierigste Hürde für Einsteiger. Das Kompilieren eines neuen Kerns ist normalerweise nicht erforderlich da der Kern, der mit Mandrakelinux installiert wird, Unterstützung für die wichtigsten Geräte (es sind mehr, als Sie jemals brauchen werden) enthält und noch durch einen Set von verlässlichen Patches erweitert wurde. Dennoch ...

Es kann sein, dass Sie es machen wollen, nur um zu „sehen, was es bewirkt“. Abgesehen davon, dass Ihr PC und Ihre Kaffeemaschine ein wenig heftiger als gewöhnlich arbeiten, nicht viel. Die Gründe, warum ein Kern neu kompiliert werden muss, reichen von der Deaktivierung einer Option bis zum Aufbauen eines experimentellen Kerns. Wie auch immer, das Ziel dieses Kapitels ist, das Ihre Kaffeemaschine auch noch nach dem Kompilieren funktioniert!

Es gibt auch andere triftige Gründe zum Rekompilieren des Kerns. Wenn etwa Ihr aktueller Kern einen sicherheitsrelevanten Fehler einen sog. ***Bug*** enthält, der erst in einer aktuelleren Version korrigiert wuerde, oder der neue Kern die Unterstützung von Peripheriegeräten bietet, die der alte nocht nicht kannte. In diesem Falle könnten Sie natürlich auf eine aktualisierte Version von Mandrakelinux warten. Das aktualisieren der Kernquellen und ein eigenes Rekompilieren stellt jedoch meist die schnellere Lösung dar.

Was immer Sie tun, legen Sie sich erst einen Vorrat an Kaffee an!

13.1 Aktualisieren des Kerns mit binären Paketen

Bevor wir ganz tief in die Kommpilierung des Kern-Quellcodes eintauchen, schauen wir uns nur kurz die einfache Prozedur zur Aktualisierung des Kerns durch ein für Ihre Version von Mandrakelinux gepacktes RPM-Paket an. Wir nehmen für dieses Beispiel an, der neue Kern sei `kernel-2.6.8.5mdk` und der alte (aktuell benutzte) hat die Version `kernel-2.6.8.1mdk`.

1. **Installation des neuen Kerns.** Geben Sie das Folgende Kommando in einem Terminalfenster ein: `urpmi kernel-2.6.8.5mdk`. Falls Sie die genaue Version nicht kennen, geben Sie einfach `urpmi kernel` ein und wählen Sie den für Ihr System passenden Kern aus der Vorschlagsliste aus.
2. **Prüfen Sie, ob er funktioniert.** Der gerade installierte Kern wird zum Standardkern. In Ihrem Bootmenü (LILO, GRUB, ELILO...) erscheint ein neuer Eintrag, ähnlich diesem: `268-5`. Führen Sie einen Neustart Ihres Rechners durch und wählen Sie den neuen Eintrag zum Starten Ihres neuen Kerns aus. Führen Sie ausgiebige Tests durch, bis Sie davon überzeugt sind, dass der neue Kern ordnungsgemäß funktioniert.
3. **De-Installieren des alten Kerns (Optional).** Wenn Sie sich vom richtigen Arbeiten des neuen Kerns überzeugt haben wollen Sie vielleicht die zum alten Kern gehörenden Dateien entsorgen. Dazu geben Sie den Befehl `urpme kernel-2.6.8.1mdk` in ein Terminal-Fenster ein. Die entsprechenden Dateien werden entfernt und die Bootloader-Konfiguration entsprechend angepasst.

13.2 Aus den Kernquellen

Grundsätzlich können Sie die Kernsourcen von zwei Stellen bekommen:

1. **Der offizielle Mandrakelinux-Kernel.** In dem `SRPMS`-Verzeichnis eines jeden beliebigen `Cooker` Mirrors (`http://www.mandrakelinux.com/en/cookerdevel.php3`) können Sie die folgenden Pakete finden:

   kernel-2.6.??.?mdk-?-?mdk.src.rpm

   Die Kernquellen zum Kompilieren des Kerns, der in der Distribution enthalten ist. Diese sind wegen zahlreicher zusätzlicher Funktionen sehr stark modifiziert.

   kernel2.6-linus-2.6.??-?mdk.src.rpm

   Der Grundkern, wie er von den GNU/Linux-Entwicklern herausgebracht wurde.

   Der offizielle Mandrakelinux Kern wird empfohlen: Holen Sie das Quellcode-RPM, installieren Sie es (als `root`) und gehen Sie zum Kapitel *Konfiguration des Kernes*, Seite 100.

2. **Die offizielle Linux Kernel Website.** Die Haupt-Kernelquellen befinden sich unterftp.kernel.org (`ftp://ftp.kernel.org`). Es gibt allerdings eine große Anzahl von Spiegelservern, alle mit dem Namen ftp.xx.kernel.org (`ftp://ftp.xx.kernel.org`), wobei `"xx"` den ISO Ländercode repräsentiert. Nach der offiziellen Verlautbarung der Verfügbarkeit des Kernes sollten Sie mindestens zwei Stunden abwarten, ehe alle Spiegelservern aktualisiert wurden.

   Auf all diesen FTP Server befinden sich die Kernquellen im Verzeichnis `/pub/linux/kernel`. Als nächstes gehen Sie in das Verzeichnis mit der Version, die Sie interessiert: es wird sehr wahrscheinlich `v2.6` sein. Natürlich hält Sie Nichts davon ab, auch die experimentellen Versionen oder die älteren 2.4-Versionen zu verwenden. Die Datei, welche die Kernquellen enthält, heißt `linux-<version>.tar.bz2`, also etwa `linux-2.6.8.tar.bz2`.

   Sie können auch Patches zur kontinuierlichen Aktualisierung des Kerns verwenden: wenn Sie etwa die Kernquellen Version 2.6.6 verwenden und zu Kern 2.6.8 aktualisieren wollen, brauchen Sie nicht die gesamten 2.6.8-Quellen zu holen, sondern können einfach die ***Patches*** `patch-2.6.7.bz2` und `patch-2.6.8.bz2` herunterladen. Im Allgemeinen ist dies eine gute Idee, da die gesamte Quelldatei mittlerweile mehrere Dutzend MB groß ist.

13.3 Entpacken der Quellen, Patchen des Kernes (falls notwendig)

Kernquellen sollten im Verzeichnis `/usr/src` platziert werden. Gehen Sie in dieses Verzeichnis und entpacken Sie dort die Quellen:

```
$ cd /usr/src
$ mv linux linux.old
$ tar xjf /pfad/zu/linux-2.6.6.tar.bz2
```

Das Kommando `mv linux linux.old` ist notwendig, falls Sie immer noch Quellen für eine andere Version des Kernes haben. Dieser Befehl stellt sicher, daß Sie den nicht überschreiben. Nachdem das Archiv entpackt wurde, haben Sie ein Verzeichnis `linux-<version>` (in dem `<version>` die Version des Kernes bezeichnet) mit den neuen Kernquellen. Sie können bei Bedarf einen Link mit `ln -s linux-<version> linux`) anlegen.

Nun zu den Patches. Wir nehmen an, dass Sie von Version 2.6.6 auf 2.6.8 ***patchen*** wollen und die nötigen Patches heruntergeladen haben: Gehen Sie in das neu erstellte Verzeichnis `linux` und spielen Sie die Patches ein:

```
$ cd linux
$ bzcat /pfad/zu/patch-2.6.7.bz2 | patch -p1
$ bzcat /pfad/zu/patch-2.6.8.bz2 | patch -p1
$ cd ..
```

Allgemein gesagt erfordert das aktualisieren von einer Version `2.6.x` zu einer Version `2.6.y` alle Patches in der Reihenfolge `2.6.x+1`, `2.6.x+2`, ..., `2.6.y-1`, `2.6.y`. Um von einer Version `2.6.y` wider zu Version `2.6.x` zurückzukehren, wiederholen Sie exakt diesee Prozedur, verwenden die Patches jedoch in umgekehrter Reihenfolge und mit der Option `-R` für den Befehl `patch` (`R` steht für *Reverse*). Also lautet der Befehl um vom Kern `2.6.8` zum Kern `2.6.6` zurückzukehren:

```
$ bzcat /pfad/zu/patch-2.6.8.bz2 | patch -p1 -R
$ bzcat /pfad/zu/patch-2.6.7.bz2 | patch -p1 -R
```

Wenn Sie vor dem tatsächlichen Patchen prüfen wollen ob ein Patch korrekt funktionieren wird, dann benutzen Sie dazu die Option `--dry-run` des Befehls `patch`.

Als Nächstes, um der Klarheit willen (und damit Sie sehen, um welche Version es sich handelt), können Sie `linux` umbenennen, um die Kern-Version wiederzugegeben und dann einen symbolischen Link erstellen:

```
$ mv linux linux-2.6.8
$ ln -s linux-2.6.8 linux
```

Anschließend beginnt die Arbeit der Konfiguration.

13.4 Konfiguration des Kernes

Zu Anfang dieses Schrittes gehen Sie in das Verzeichnis `/usr/src/linux` und wechseln Sie zu `root`.

Zuerst ein kleiner Trick: Sie können, wenn Sie möchten, die Version Ihres Kernes anpassen. Die Kernversion ist in der ersten vier Zeilen der Datei `Makefile` festgelegt:

```
$ head -4 Makefile
VERSION = 2
PATCHLEVEL = 6
SUBLEVEL = 8
EXTRAVERSION = -1mdkcustom
```

Weiter unten können Sie im `Makefile` sehen, dass die Kernelversion erstellt wurde:

```
KERNELRELEASE=$(VERSION).$(PATCHLEVEL).$(SUBLEVEL)$(EXTRAVERSION)
```

Alles was Sie nun tun müssen, ist eines dieser Felder zu verändern. Am Besten ändern Sie nur EXTRAVERSION. Sagen wir, Sie setzen es auf `-foo`, dann wird Ihre neue Kernversion 2.6.8-foo sein. Zögern Sie nicht, bei jedem erneuten Durchbauen des Kernes dieses Feld zu verändern. So können Sie verschiedene Optionen testen, während die alten Versuche erhalten bleiben.

Nun zur Konfiguration. Wählen Sie zwischen:

- `make xconfig` für eine grafische, auf `qt` basierende Benutzeroberfläche;
- `make gconfig` für eine grafische, auf `gtk+` basierende Benutzeroberfläche;
- `make menuconfig` für eine Benutzeroberfläche basierend auf`ncurses`;
- `make config` für eine mehr rudimentäre Benutzeroberfläche, Zeile für Zeile, Bereich nach Bereich;
- `make oldconfig` entspricht der vorherigen Variante jedoch basierend auf Ihrer vorherigen Konfiguration. Siehe *Speichern und Wiederverwenden Ihrer Kernkonfiguration*, Seite 102.

Sie werden dann Abschnitt für Abschnitt durch die Konfiguration geleitet. Sie können aber auch Abschnitte auslassen und zu denen springen, die Sie interessieren, wenn Sie `menuconfig`, `xconfig` oder `gconfig` benutzen. Die Optionen sind **y** für Yes (Funktion wird in den Kern eingebunden), **m** für Module (Funktion wird als Modul kompiliert) oder **n** für No (nicht benötigt).

Die Werkzeuge `make xconfig`, `make menuconfig` und `make gconfig` enthalten die Optionen gebündelt in hierachischen Gruppen. Zum Beispiel steht die `Prozessor-Familie` unter `Prozessortyp und Features`.

Im Dialogfenster von `xconfig` bzw. `gconfig` wird die Schaltfläche Main Menu benutzt, um aus einer Untergruppe in das Hauptmenü zurückzukehren. Mit Next gelangt man zur nächsten Gruppe von Optionen und mit Prev zur vorherigen Gruppe zurück. Bei `menuconfig` benutzen Sie **Enter** um einen Abschnitt auszuwählen und **y**, **m** oder **n** um zwischen den Optionen zu wählen. Mit **Enter** kommen Sie auch in die Einzelmenüs zur Auswahl von Mehrfach-Optionen. Mit Exit verlassen Sie einen Abschnitt oder (falls Sie im Hauptmenü sind) die Konfiguration. Und dann gibt es natürlich noch Help.

Wir wollen hier nicht alle Optionen aufzählen, da es mehrere Hunderte davon gibt. Zudem, wenn Sie dieses Kapitel erreicht haben, werden Sie wahrscheinlich wissen, was zu tun ist. Also browsen Sie ruhig durch die Konfiguration und setzen die Optionen nach Ihrem Gutdünken. Im Folgenden lesen Sie jedoch eine Anleitung, die verhindern soll, dass Sie am Ende mit einem unbrauchbaren Kern dastehen:

1. Wenn Sie nicht gerade eine Ramdisk verwenden (siehe auch `initrd`, kompilieren Sie **niemals** die Treiber zum Einhängen des Root-Dateisystems (Hardware-Treiber und Dateisystem-Treiber) als Module! Wenn Sie eine Ramdisk benutzen, sagen Sie `Y` zur ext2FS-Unterstützung, da dies das Dateisystem ist, das für Ramdisks benutzt wird. Sie brauchen dann außerdem die Unterstützung für `initrd`;
2. Haben Sie Netzwerkkarten in Ihrem System, dann kompilieren Sie deren Treiber als Module. Dadurch können Sie festlegen, welche Karte die erste und welche die zweite usw. ist, indem Sie entsprechende Aliases in `/etc/modules.conf` eintragen. Wenn Sie die Treiber in den Kern kompilieren, ist die Reihenfolge, in der diese geladen werden, abhängig von der Verknüpfungsanordnung, welche nicht die gewünschte Reihenfolge sein mag.
3. Zum Abschluss: Wenn Sie nicht wissen wozu eine Option dient, lesen Sie erst den Hilfetext! Wenn Sie dort keine Inspiration finden, lassen Sie die Option so, wie sie ist (bei `config` und `oldconfig` drücken Sie **?** um zur Hilfe zu gelangen).

Et voilà! Die Konfiguration ist nun vorüber. Speichern Sie Ihre Änderungen ab und beenden Sie das Programm.

13.5 Speichern und Wiederverwenden Ihrer Kernkonfiguration

Die Kernkonfiguration wird in der Datei `/usr/src/linux/.config` gespeichert. Es gibt eine Sicherungskopie in `/boot/config-<version>`, die man als Referenz behalten sollte. Aber speichern Sie auch Ihre eigene Konfigurationen für verschiedene Kerne ab, da das nur eine Frage der unterschiedlichen Namensgebung für die Konfigurationsdateien ist.

Eine Möglichkeit ist die Benennung der Konfigurationsdatei nach der Kernversion. Wenn Sie beispielsweise Ihre Kernversion wie bei *Konfiguration des Kernes*, Seite 100 gezeigt ändern, könnten Sie folgendermaßen vorgehen:

```
$ cp .config /root/config-2.6.8-foo
```

Wenn Sie sich etwa dazu entschlossen haben, auf die Version `2.6.9` zu aktualisieren, können Sie diese Datei wiederverwenden, da die Differenz zwischen den Konfigurationen dieser beiden Kerne sehr gering sein wird. Benutzen Sie einfach die Sicherheitskopie:

```
$ cp /root/config-2.6.8-foo .config
```

Das Zurückkopieren der Dateien bedeutet jedoch nicht, dass der Kern nun bereit zum Durchbauen ist. Sie müssen wieder `make menuconfig` (oder was Sie normalerweise benutzen) starten, weil einige Dateien, die für eine erfolgreiche Kompilierung benötigt werden dadurch erst erstellt und/oder modifiziert werden.

Sie könnten so allerdings, abgesehen von der Arbeit, nochmals durch alle Menüs zu gehen, einige interessante neue Optionen verpassen. Dies vermeiden Sie, indem Sie den Befehl `make oldconfig` benutzen. Das hat zwei Vorteile:

1. er ist schnell;
2. Wenn eine neue Option im Kern erscheint und diese nicht in Ihrer Konfigurationsdatei steht, hält der Prozess an und wartet, bis Sie eine Auswahl getroffen haben.

Nachdem Sie die `.config` Datei in das Home-Verzeichnis von `root` kopiert haben (wie oben vorgeschlagen), starten sie den Befehl `make mrproper`. So wird sichergestellt, dass keine Dateien verbleiben, die noch mit der alten Konfiguration erstellt wurden, und Sie somit einen sauberen Kern bekommen.

Es folgt die Kompilierung

13.6 Kompilieren des Kerns und der Module, Installation des gesamten „Pakets“

Eine Kleinigkeit zu Beginn: Wenn Sie einen Kern mit exakt identischer Version neu durchbauen wollen, die schon in Ihrem System vorhanden ist, müssen die alten Module zuerst gelöscht werden. Wenn Sie also die Version `2.6.8` rekompilieren, dann löschen Sie zuerst das Verzeichnis `/lib/modules/2.6.8`.

Das Kompilieren des Kernes und der Module und das Installieren der Module geschieht durch folgende Zeilen:

```
make clean
make all
make modules_install install
```

Eine kleine Begriffserklärung: Die benutzten Argumente `clean`, `all`, etc., werden **Targets** („Ziele“) genannt. Mit dem Kern `2.6` wurde das Target `all` eingeführt. Es beinhaltet die Ausführung der von älteren Kernen her bekannten Targets (bei x86-Systemen) `bzImage` und `modules`. Diese neue Option erstellt die vorgegebenen Targets unter jeder vorhandenen Architektur während vor der Kernserie `2.6` jede Architektur ihre eigenen verschiedenen Argumente zum Durchbauen des Kernes hatte. Wenn Sie verschiedene Ziele für `make` angeben

(wie oben beschrieben), dann werden Sie in der Reihenfolge ihrer Auflistung ausgeführt. Wenn dabei ein Ziel nicht erledigt werden kann, macht `make` nicht weiter [1].

Lassen Sie uns die unterschiedlichen Ziele betrachten und sehen, was diese bewirken:

- `bzImage`:dies erstellt den Kern. Beachten Sie, dass dieses Ziel nur für **x86** und **x86_64** Prozessoren gültig ist. Dieses Ziel erstellt auch die `System.map` für den Kern. Wir werden später sehen, wozu diese Datei gebraucht wird;
- `modules`: dieser Befehl generiert Module für den Kern, den Sie gerade erstellt haben. Wenn Sie keine Module verwenden, bewirkt dieser Befehl nichts.
- `all`: mit diesem Ziel erstellen Sie den gewünschten Kern (entsprechend der vorhandenen Architektur) sowie die Module;
- `modules_install`: dies installiert die Module. Standardmäßig weden Module im Verzeichnis `/lib/modules/<kernel-version>` installiert. Dieses Ziel berechnet auch die Modulabhängigkeiten;
- `install`: dieser letzte Befehl kopiert den Kern und die Module schließlich zum richtigen Patz und modifiziert die Konfiguration Ihres Betriebssystemstarters so, dass der neue Kern zur Boot-Zeit verfügbar ist. Benutzen Sie dieses Target nicht, wenn Sie eine manuelle Installation wie in *Manuelle Installation des neuen Kernes*, Seite 103 bevorzugen.

Es ist wichtig, die Zielreihenfolge `modules_install install` einzuhalten, so dass die Module zuerst installiert werden. Anderenfalls wäre `initrd` nicht korrekt und der Bootvorgang würde fehlschlagen.

An diesem Punkt ist nun alles kompiliert und korrekt installiert, bereit zum Testen! Starten Sie Ihren Rechner neu und wählen Sie den neuen Kern im Startmenü aus. Beachten Sie, dass der alte Kern erreichbar bleibt, so dass Sie ihn bei Problemen mit dem neuen Kern wiederverwenden können. Sie können sich aber auch für eine manuelle Installation entscheiden und das Startmenü manuell verändern. Wir werden das im nächsten Abschnitt behandeln.

13.7 Manuelle Installation des neuen Kernes

Die Vorgehensweise in diesem Kapitel bezieht sich auf die **x86**-Architektur. Falls Sie eine andere Architektur benutzen können sowohl die Speicherorte der Dateien als auch die zu installierenden Dateien differieren.

Der Kern befindet sich in der Datei `arch/i386/boot/bzImage`. Das Standardverzeichnis,in dem Kerne installiert werden, ist `/boot`. Sie müssen außerdem die Datei `System.map` kopieren, um sicherzugehen, dass einige Programme (beispielsweise `top`) korrekt funktionieren. Eine gute Vorgehensweise ist auch hier: benennen Sie diese Dateien nach der Kernversion. Nehmen wir an, Ihre Kernversion ist `2.6.8-foo`. Die Befehls-Sequenz lautet in diesem Fall:

```
$ cp arch/i386/boot/bzImage /boot/vmlinuz-2.6.8-foo
$ cp System.map /boot/System.map-2.6.8-foo
```

Nun müssen Sie dem Betriebssystemstarter mitteilen, dass ein neuer Kern vorhanden ist. Ihre Distributooopn bietet Ihnen zwei Starter: GRUB oder LILO. Beachten Sie, dass Mandrakelinux standardmäßig mit LILO installiert wird.

1. Wenn die Kompilierung an dieser Stelle fehlschlägt, haben Sie einen Bug im Kern gefunden... Ist dies der Fall, melden Sie es bitte!

13.7.1 Aktualisierung von LILO

Der einfachste Weg, LILO zu aktualisieren, ist die Benutzung von drakboot (siehe Kapitel Ändern Ihrer Startkonfiguration im *Starter Handbuch*). Alternativ können Sie die Konfigurationsdatei manuell bearbeiten:

Die Konfigurationsdatei für LILO ist `/etc/lilo.conf`. So sieht etwa eine typische `lilo.conf` aus:

```
boot=/dev/hda
map=/boot/map
default="linux"
keytable=/boot/de-latin1.klt
prompt
nowarn
timeout=50
message=/boot/message
menu-scheme=wb:bw:wb:bw
image=/boot/vmlinuz
        label="linux"
        root=/dev/hda1
        read-only
        initrd=/boot/initrd.img
        append="devfs=mount resume=/dev/hda5"
other=/dev/fd0
        label="floppy"
        unsafe
```

Eine `lilo.conf` enthält eine allgemeine Sektion, gefolgt von je einem Abschnitt für jedes Betriebssystem. Im vorhergehenden Beispiel wird der allgemeine Abschnitt durch die folgenden Anweisungen gebildet:

```
boot=/dev/hda
map=/boot/map
default="linux"
keytable=/boot/de-latin1.klt
prompt
nowarn
timeout=50
message=/boot/message
menu-scheme=wb:bw:wb:bw
```

Die Anweisung `boot=` teilt LILO mit, wo sein Bootsektor installiert werden soll. In diesem Falle ist es der MBR (*Master Boot Record*) der ersten IDE-Festplatte. Wenn Sie eine LILO - Diskette erstellen wollen, dann ersetzen Sie einfach `/dev/hda` mit `/dev/fd0`. Die Anweisung `prompt` teilt LILO mit, dass ein Startmenü erscheinen soll. Da ein Timeout gesetzt wurde, startet LILO das Standardsystem in 5 Sekunden (`timeout=50`). Wenn Sie die Timeout-Anweisung entfernen, wird LILO beim Systemstart warten, bis Sie etwas eingeben.

Dann kommt der Bereich `linux`:

```
image=/boot/vmlinuz
        label="linux"
        root=/dev/hda1
        initrd=/boot/initrd.img
        append="devfs=mount resume=/dev/hda5"
        read-only
```

Ein Abschnitt, der ein GNU/Linux Betriebssystem startet, beginnt immer mit einer `image=` Anweisung, gefolgt von dem vollständigen Pfad zu einem gültigen GNU/Linux Kern. Wie jede andere Sektion beinhaltet es eine `label=` Anweisung zur eindeutigen Identifizierung, hier `linux`. Die `root=` Anweisung teilt LILO mit, welche Partition die Verzeichnisbaumwuzel beherbergt, das Dateisystem für diesen Kern. Das kann, je nach Konfiguration, unterschiedlich sein. Die `read-only` Anweisung teilt LILO mit, dass es dieses Dateisystem beim Start nur zum Lesen einhängen soll: wenn diese Anweisung nicht gesetzt wurde, bekommen Sie eine Warnung angezeigt. Mit der `append` Anweisung gibt man dem Kern Optionen mit.

Nun folgt der Bereich für die Startdiskette:

```
other=/dev/fd0
        label="floppy"
        unsafe
```

Sektionen, die mit `other=` beginnen, werden von LILO benutzt um andere Betriebssysteme zu starten. Das Argument dieser Anweisung ist die Lage des Bootsektors dieses Systems - in diesem Fall handelt es sich um eine Start-Diskette.

Jetzt ist es an der Zeit, dass wir eine Sektion für unseren neuen Kern hinzufügen. Sie können diese Sektion überall hinter den allgemeinen Abschnitt schreiben, nur nicht innerhalb eines anderen Abschnitts. Das könnte also so aussehen:

```
image=/boot/vmlinuz-2.6.8-foo
label="foo"
root=/dev/hda1
read-only
append="devfs=mount resume=/dev/hda5"
```

Selbstverständlich müssen Sie das auf Ihre Konfiguration anpassen!

So sollte Ihre `lilo.conf` nach der Änderung aussehen, versehen mit einigen passenden Kommentaren (diese beginnen mit #), die von LILO ignoriert werden:

```
#
# Allgemeiner Bereich
#
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
message=/boot/message
# Was soll standardmäßig gestartet werden? Nehmen wir unseren neuen Kern:
default="foo"
# Prompt anzeigen...
prompt
# ... 5 Sekunden warten
timeout=50
#
# Unser neuer Kern:
#
image=/boot/vmlinuz-2.6.8-foo
        label="foo"
        root=/dev/hda1
        read-only
        append="devfs=mount resume=/dev/hda5"
#
# Der Orginalkern:
#
image=/boot/vmlinuz
        label="linux"
        root=/dev/hda1
        read-only
        append="devfs=mount resume=/dev/hda5"
#
# Starten von Diskette:
#
other=/dev/floppy
        label="floppy"
        unsafe
```

So könnte also Ihre Datei `lilo.conf` aussehen, aber denken Sie noch einmal daran, sie Ihrer eigenen Konfiguration anzupassen.

Nun, da die Datei entsprechend geändert wurde, müssen Sie - im Unterschied zu GRUB - LILO noch anweisen, den Bootsektor zu ändern:

```
$ lilo
Added foo *
Added linux
Added floppy
$
```

Auf diesem Wege können Sie so viele Kern kompilieren, wie Sie wollen und sie jeweils in einen neuen Abschnitt eintragen. Alles was Sie jetzt noch zu tun haben ist, den Rechner neu zu starten um den neuen Kern zu testen. Beachten Sie, dass LILO bei irgendeiner Fehlermeldung während der Installation nichts verändert.

13.7.2 Aktualisierung von Grub

Sie sollten die Möglichkeit, den aktuell benutzten Kern zu starten, unbedingt beibehalten! Der einfachste Weg, GRUB zu aktualisieren, ist die Benutzung von drakboot (siehe Kapitel Ändern Ihrer Startkonfiguration in *Starter Handbuch*). Alternativ können Sie die Konfigurationsdatei auch manuell wie Folgt ändern:

Sie müssen die Datei `/boot/grub/menu.lst` editieren. So sollte eine typische `menu.lst` nach der Installation Ihrer Mandrakelinux Distribution und vor einer Veränderung aussehen:

```
timeout 5
color black/cyan yellow/cyan
i18n (hd0,4)/boot/grub/messages
keytable (hd0,4)/boot/de-latin1.klt
default 0

title linux
kernel (hd0,4)/boot/vmlinuz root=/dev/hda5

title failsafe
kernel (hd0,4)/boot/vmlinuz root=/dev/hda5 failsafe

title Windows
root (hd0,0)
makeactive
chainloader +1

title floppy
root (fd0)
chainloader +1
```

Diese Datei besteht aus zwei Teilen: der Kopf mit allgemeinen Optionen (die ersten fünf Zeilen) und die Images, jedes davon für einen GNU/Linux Kern oder ein anderes BS. `timeout 5` legt die Zeit in Sekunden fest, in der GRUB auf eine Eingabe wartet, bevor das Standardimage gestartet wird. Dieses wird in den allgemeinen Optionen mit der Anweisung `default 0` (das erste Image) definiert. Die Anweisung `keytable`, wenn vorhanden, bestimmt, wo das Layout für Ihre Tastatur zu finden ist. In diesem Beispiel ist es ein deutsches Layout. Ist keine solche Anweisung vorhanden, wird es als normales `QWERTY`-Keyboard angesprochen. Alle Anweisungen in der Form von `hd(x,y)`, die Sie sehen, beziehen sich auf Partitionsnummer `y` auf Festplatte Nummer `x`, vorgegeben vom BIOS.

Es folgen die verschiedenen Images. In diesem Fall sind vier Images definiert: `linux`, `failsafe`, `windows` und `floppy`.

- Die Sektion `linux` startet mit der Übergabe des zu ladenden Kernes an GRUB (`kernel hd(0,4)/boot/vmlinuz`), gefolgt von den Optionen, die an den Kern übergeben werden sollen. In diesem Fall teilt `root=/dev/hda5` dem Kern mit, dass die Verzeichnisbaumwurzel sich auf `/dev/hda5` befindet. Tatsächlich ist `/dev/hda5` das Äquivalent zu GRUB's `hd(0,4)`, aber der Kern kann sich auch auf einer anderen Partion befinden als derjenigen, auf der die Verzeichnisbaumwurzel liegt.
- Der Abschnitt `failsafe` ist dem vorhergehenden sehr ähnlich, abgesehen davon, dass wir ein Argument an den Kern übergeben (`failsafe`), das ihn anweist, in den „Single-" oder „Rescue-" Modus zu wechseln.
- Der Abschnitt `Windows` übermittelt GRUB den Bootsektor der ersten Partition zu laden, der vermutlich einen Windows® Bootsektor enthält.
- Der Abschnitt `floppy` startet das auf der Diskette befindliche System. Das kann sowohl eine Windows®- als auch eine GNU/Linux-Startdiskette sein, oder ein ganz anderes Betriebssystem;

Abhängig von der Sicherheitsebene, die Sie auf Ihrem System benutzen, können einige der hier beschriebenen Einträge in Ihrer Konfigurationsdatei fehlen.

Nun zur Sache: Wir müssen einen weiteren Abschnitt hinzufügen, um GRUB über den neuen Kern zu informieren. In diesem Beispiel soll er vor den anderen Einträgen stehen, Sie können ihn aber auch an anderer Stelle zu platzieren:

```
title foo
kernel (hd0,4)/boot/vmlinuz-2.6.8-foo root=/dev/hda5
```

Vergessen Sie nicht, die Datei an Ihre Konfiguration anzupassen! Das GNU/Linux Root-Dateisystem ist hier auf `/dev/hda5` zu finden. Auf Ihrer Festplatte kann es sich aber auch an anderer Stelle befinden.

Das war's! Anders als bei LILO (wie wir weiter oben gesehen haben), ist hier nichts weiter zu tun. Starten Sie Ihren Rechner neu und Ihr neu definierter Eintrag wird erscheinen. Wählen Sie ihn im Menü aus und der neue Kern wird gestartet.

Wenn Sie den neuen Kern mit der Framebuffer-Option kompiliert haben, werden Sie ihn vermutlich verwenden wollen: in diesem Fall müssen Sie eine Anweisung zum Kern hinzufügen, die ihm mitteilt, in welcher Auflösung gestartet werden soll. Die Liste der möglichen Einstellungen finden Sie in `/usr/src/linux/Documentation/fb/vesafb.txt` (nur im Falle eines VESA-Framebuffers! Anderenfalls lesen Sie die entsprechende Datei). Für eine Auflösung von 800x600 mit 32 Bit Farbtiefe[2] ist die Modusnummer `0x315`. Fügen Sie also diese Anweisung hinzu:

```
vga=0x315
```

und Ihr Eintrag sieht jetzt so aus:

```
title foo
kernel (hd0,4)/boot/vmlinuz-2.6.8-foo root=/dev/hda5 vga=0x315
```

Weitere Informationen über GRUB erhalten Sie auf dessen Info-Seiten (`info grub`).

2. 8 Bits bedeuten 2^8 Farben, also 256; 16 Bits bedeuten 2^{16} Farben, also 64k oder 65536; in 24 Bits wie in 32 Bits bekommen Sie 2^{24} mögliche Farben, in anderen Worten 16M oder exakt ein wenig mehr als 16 Millionen mögliche Farben dargestellt.

Anhang A. Glossar

Abhängigkeiten, die
Ein von `rpm` verwendetes System, um die Verwaltung installierter Software zu vereinfachen. Jedes Paket enthält ein Skript, das dem lokalen `rpm` mitteilt, welche anderen Pakete es zur Ausführung benötigt. Sind die erforderlichen Pakete nicht installiert, wird die Installation abgebrochen. Gleiches gilt für die Deinstallation.

Account
In einem UNIX®-System die Kombination von Name, persönlichem Verzeichnis, Passwort und shell; ermöglicht einem Benutzer den Zugang zum System.

Alias, der
Ein Shell -Mechanismus, um eine Zeichenkette durch einen anderen zu ersetzen, bevor ein Kommando ausgeführt wird. Einen Überblick über alle definierten Aliasse erhalten Sie mit dem Kommando `alias`.

anpassbare Oberfläche, die
Ein grafisches Programm hat eine anpassbare Oberfläche (denglisch auch: *ist themeable*), falls sein Aussehen in Echtzeit geändert werden kann. Viele Fenstermanager haben diese Fähigkeit.

APM, das
Advanced Power Management („Fortgeschrittenes Energiemanagement“). Wird von einigen BIOS-sen dazu benutzt, die Maschine nach einer Periode von Inaktivität in einen energiesparenden Ruhezustand zu bringen. Auf Laptops ist APM auch dafür zuständig, Informationen über den Batteriestatus bzw. die verbleibende Arbeitszeit bereit zu stellen.

Arbeitsfläche
Wenn Sie das X window system verwenden, ist die Arbeitsfläche der Teil des Bildschirms, auf dem Sie arbeiten und auf dem Ihre Symbole und Fenster dargestellt werden. Manchmal wird auch lachs vom „Hintergrund“ gesprochen.
Siehe auch: virtuelle Arbeitsfläche, die.

architekturspezifisch
bedeutet, dass Programme, wie imake und `make` für ihre Ausführung Informationen über den Rechner, auf dem sie ausgeführt werden, benötigen. So ist es etwa wichtig, wo sich Bibliotheken oder Hilfswerkzeuge wie `sed` befinden.

ARP, das
Address Resolution Protocol. („Protokoll zur Adress-Auflösung“). Das Protokoll um IP-Adressen dynamisch in physikalische (Hardware-)Adressen umzuwandeln. Nur möglich auf direkt verbundenen Netzwerken mit Hardware-Broadcasting.

ASCII
Abkürzung für *American Standard Code for Information Interchange*. Es handelt sich um den meistverwendeten 7-Bit-Kode zur Darstellung von Ziffern, Buchstaben und Sonderzeichen. Der ASCII-Code ist mittlerweile per „normativer Kraft des Faktischen“ durch den ISO Standard 8859-1 abgelöst.
Siehe auch: .

Assembler, der
Maschinensprache. Vorstufe des Objekt-Kodes. Programme werden meist in sogenannten Hochsprachen (wie C) geschrieben, die dann per Kompilation in Assembler umgewandelt werden.

ATAPI
Abkürzung für *AT Attachment Packet Interface* (engl. für „AT-Geräte Paket-Schnittstelle“) Es handelt sich um eine Erweiterung der ATA Spezifikation. (*Advanced Technology Attachment*), bekannter unter dem Namen IDE, *Integrated Drive Electronics*). Dieser Standard bietet zusätzliche Befehle für das Ansprechen von CD-ROM -, und Band-Laufwerken. IDE -Geräte, die diesen Standard unterstützen werden auch als EIDE-Geräte (*Enhanced IDE*) bezeichnent.

ATM, der
Abkürzung für *Asynchronous Transfer Mode* (engl für. „Aysynchroner Transfermodus“. Ein ATM-Netzwerk packt Daten in Blöcke einheitlicher Größe (53 Bytes: 48 Datenbytes, 5 Verwaltungsbytes), um effiziente Punkt-zu-Punkt Verbindungen erreichen zu können. ATM ist eine geswitchte Netzwerktechnologie, die für optische Hochgeschwindigkeitsnetzwerke konzipiert wurde.

atomar
Eine Gruppe von Operationen, die nach außen als eine Einheit, als nicht weiter aufteilbar, angesehen wird.

ausführlicher Modus, der
Befehle produzieren im ausführlichen Modus Zusatzinformationen, die auf die Satandardausgabe oder Standard-Fehlerausgabe augegeben werden. Einige Programme bieten die Möglichkeit, einzustellen wie „redselig“ sie sein sollen.

Ausgabe, die (Target, das)
In diesem Zusammenhang ist das durchzubauende Objekt, etwa ein Programm, das von einem Kompilierer erzeugt wird, gemeint.

Batch-Betrieb, der
Es handelt sich um einen Verarbeitungsmodus, in dem die Aufgaben *en block* an den Prozessor übergeben werden, und von diesem der Reihe nach abgearbeitet werden. Erst wenn der letzte Prozess abgearbeitet ist, kann wieder ein Stapel neuer Aufgaben weitergereicht werden.

Beep, der
Akustische Warn- und Fehlermeldung.

Befehlsmodus, der
In Vi (oder einem Klon) ein Modus, der nicht für die Eingabe von Text sondern von Textbearbeitungskommandos dient. Dies ist der initiale Modus beim Start von Vi. U.a. mit der **i**-Taste gelangen Sie in den Einfügemodus.

benannte Pipe, die
Eine Unix Ein- und Ausgabeumleitung (*Pipe*), die im Dateisystem sichtbar wird.
Siehe auch: Link, der, Pipe, die („Röhre“, die).

Benutzerkennzeichen, das
Der Name, mit dem Sie sich an einem Unix-System anmelden. Zu einem Benutzerkennzeichen gehört ein Heim-Verzeichnis, eine Standard-Shell sowie ein Passwort.

Besitzergruppe, die
In Unix eine Gruppe von Benutzern, denen bestimmte Rechte zukommen. Die Mitgliedschaft in einer Benutzergruppe wird in der Datei `/etc/group` festgelegt.

Beta-Test, der
Software durchläuft verschiedene Stadien, unter anderem die Alpha- und die Betaphase. Die Betaphase bezeichnet das Stadium eines Produkts, das seiner offiziellen Veröffentlichung vorausgeht und dient dazu, das Programm in einer breiten Öffentlichkeit zu testen.

Betriebssystem, das
ein Prozess, der ständig im Hintergrund läuft und grundlegende Operationen zur Verfügung stellt. So muss ein Betriebssystem beispielsweise dafür sorgen, dass die Betriebsmittel des Rechners zur Verfügung stehen. Auf einem GNU/Linux-System wird dies durch den Kern und ladbare Module gewährleistet. Es gibt aber auch noch AmigaOS, MacOS, FreeBSD, OS/2, Unix, Windows NT und Windows 9x.

Betriebssystemstarter, der
ein Programm zum Starten des Betriebssystems. Viele solche Starter geben Ihnen die Möglichkeit unter mehreren Betriebssystemen dasjenige auszuwählen, das gestartet werden soll. Starter wie grub sind daher sehr populär und nützlich bei Systemen mit zwei oder mehr Betriebssystemen.

Bibliothek, die
Siehe „Programmbibliothek“.

Binden, das
Letzter Schritt des Kompilationsprozesses. In ihm werden alle Objektdateien zusammengebunden, um daraus ein ausführbares Programm zu machen.

Bit, das
Steht für *BInary uniT* („zweiwertige Einheit“). Eine einzelne Ziffer, die die Werte „0“ oder „1“ annehmen kann. Berechnungen erfolgen im Binärsystem.

blockorientierte Datei, die
Dateien, deren Inhalt zwischengespeichert wird. Jede Schreib-/Leseoperation dieser Dateien geht durch Zwischenspeicher (Puffer), welche asynchrones Schreiben auf die zugrundeliegende Hardware erlauben und Lesezugriffe beschleunigen.
Siehe auch: Puffer, der, Puffer-Cache, der, zeichenorientierte Datei, die.

Bootdiskette, die
Siehe „Startdiskette“.

Booten, das
Startvorgang, beinhaltet Erkennung und Konfiguration der Hardware und das Laden des Betriebssystems in den flüchtigen Speicher.

BSD
Berkeley Software Distribution. Eine Unix-Variante, entwickelt an der Universität von Berkeley, USA. Freie BSD-Versionen ähneln GNU/Linux in vielen Bereichen, die bekannteste ist FreeBSD.

Bug, der
Unlogisches, unzusammenhängendes Programmverhalten oder ein Verhalten, das von dem dokumentierten abweicht. Oft ausgelöst durch neue Fähigkeiten des Programms.

Byte, das
Acht aufeinanderfolgende Bits, im Dualsystem als Nummer zwischen „0“ und „255“ interpretiert.
Siehe auch: Bit, das.

CHAP
Challenge-Handshake Authentication Protocol: Wird von ISPs benutzt, um Klienten zu authentifizieren. In diesem Schema wird dem sich anmeldenden Gastrechner ein Wert gesendet. Dieser errechnet einen hash aus diesem Wert und schickt ihn zurück zum Server. Der Server schließlich vergleicht diesen hash mit dem, den er selbst errechnet hat.
Siehe auch: PAP.

CIFS
Common Internet FileSystem Nachfolger des SMB-Dateisystems.

Cookie, der
Cookies („Kekse“) sind temporäre Dateien, die von einem Web-Server auf die lokale Festplatte eines Anwenders geschrieben werden. Mit ihrer Hilfe werden Benutzerpräferenzen gepeichert.

Datagram, das
Ein Datagram ist ein spezielles Paket aus Daten und einem Verwaltungskopf. In diesem Anfangsbereich stehen Adressesn, und zwar genau die, die für eine Übertragung über IP-Netzwerke notwendig sind. Man verwendet häufig auch den Begreiff „Paket“, wenn man eigentlich ein Datagram meint.

Dateisystem, das
Wird benutzt, um Dateien konsistent auf physischen Medien zu speichern. Beispiele sind FAT (Windows), ext2fs (GNU/Linux), iso9660 usw.

DHCP
Dynamic Host Configuration Protocol. Ein Protokoll mit dessen Hilfe Maschinen auf einem lokalen Netzwerk dynamisch IP-Adressen zugewiesen werden können.

diskreter Wert, der
Für sich selbst stehender, unterscheidbarer Wert.

Diskussionsforen, die
Diskussions- und Nachrichten-Bereiche, die mit einem News- oder USENET-Klienten gelesen und geschrieben werden können. Die Foren sind in hierarchische Gruppen unterteilt, so ist etwa `de.os.linux.mandrake` eine deutschsprachiges (`de`) Forum, dass sich mit Betriebssystemen beschäftigt (`os`), speziell GNU/Linux (`linux`) und zwar der Distribution **Mandrake Linux** (`mandrake`). Diese Hierarchien wurden eingerichtet, um das Suchen zu erleichtern.

Distribution, die
Der Begriff, mit dem die einzelnen GNU/Linux-Versionen eines Herstellers bezeichnet werden. Eine Distribution besteht aus dem Linux-Kern, den Anwendungen, einem Installationsprogramm, Programmen anderer Firmen und manchmal auch proprietären Programmen.

DLCI, der
Abkürzung für *Data Link Connection Identifier*. er wird für eine explizite virtuelle Punkt-zu-Punkt Verbindung über ein sog. *Frame Relay* Netzwerk verwendet. Die DLCIs werden normalerweise vom *Frame Relay* Netzwerkanbieter festgelegt.

DMA
Direct Memory Access. Eine Möglichkeit auf der Intel-Architektur, welche Peripheriegeräten unter Umgehung der CPU direkten Zugriff auf den Systemspeicher erlaubt. PCI-Geräte benutzen ein anderes Verfahren, das sog. Bus-Mastering, und brauchen daher kein DMA.

DNS
Domain Name System (engl. für „Domänennamen-System“) ist der Mechanismus, der im Internet benutzt wird um Namen und Adressen bekannt zu machen. Es ist hiermit möglich einen Domänennamen auf eine IP-Adresse abzubilden, was Ihnen erlaubt auf Webseiten zuzugreifen, ohne die IP-Adresse des betreffenden Rechners zu kennen.

DPMS
Display Power Management System. Energiesparprotokoll, wird von allen modernen Monitoren verwendet. Solche Monitore werden auch „grüne Monitore“ genannt.

Durchquerung, die
Auf Unix Systemen bedeutet das für Ordner, dass Besitzer gewisser Benutzerkennzeichen die Erlaubnis haben, in einen gewissen Ordner zu wechseln, möglicherweise (falls Unterordner existieren) diesen sogar zu „durchqueren“. Der Besitzer des Ordners muss hierfür die notwendigen Rechte erteilen.

E-Mail, die
Electronic Mail, Elektronische Post. Eine Möglichkeit, um Nachrichten zwischen Personen durch ein Netzwerk zu verschicken. Genau wie bei der normalen Post (auch *Snail Mail*, „Schneckenpost“ genannt) wird sowohl ein Empfänger benötigt („empfänger@empfänger.domäne“), als auch ein Absender (s.o.). E-Mail ist ein sehr schnelles System um zwischen Personen zu kommunizieren. Es braucht im Schnitt lediglich einige Minuten bis zur Auslieferung, egal wie groß die Entfernung zwischen Sender und Empfänger ist. Um eine E-Mail verschicken zu können, benötigen Sie einen E-Mail-Klienten, wie etwa die Textprogramme pine oder `mutt`, oder grafische Versionen wie kmail.

echo
Ein Unix-Kommando, das nur seine Argumente ausgibt. Allgemein die Ausgabe auf dem Schirm.

Eingabemodus, der
In Vi (und seinen Klonen), der Modus, in dem Text in eine Datei geschrieben werden kann. Sie gelangen in den Kommandomodus mittels der Taste **Esc** (oder der Kombination **Strg-[**).

eingehängt
Ein Medium ist eingehängt, wenn es über das GNU/Linux Dateisystem erreichbar ist. Sie können sich dann den Inhalt des Mediums anzeigen lassen. Durch Verwendung von „Supermount“ wird es überflüssig, das Medum explizit ein- bzw. auszuhängen – die Wechselmedien werden dynamisch vom System verwaltet.
Siehe auch: Mount-Punkt, der.

ELF
Executable and Linking Format. Das heute von GNU/Linux verwendete Binärformat für Programme und Bibliotheken.

Expandierung, die
Das Ersetzen von regulären Ausdrücken durch konkrete Begriffe. So werden in der Shell etwa Dateinamen aus regulären Ausdrücken expandiert.
Siehe auch: Suchmuster, das.

ext2
kurz für: „second extended filesystem“, engl. für *zweites erweitertes Dateisystem*. Es stellt das hauseigene Dateisystem von GNU/Linux dar. Der Vorteil gegenüber älteren Systemen wie FAT oder FAT32 ist der höhere Durchsatz, lange Dateinamen, Dateirechte und höhere Fehlertoleranzen.

FAQ, die
Frequently Asked Questions: Dokument, das häufig gestellte Fragen (und natürlich auch die Antworten darauf) zu einem speziellen Thema enthält. Entstanden ist diese Dokumentform in den Diskussionsforen,

doch sie hat sich mittlerweile auf Web -Sites und sogar kommerzielle Produkte ausgeweitet. Sie sind oft sehr gute Informationsquellen.

FAT, die
File Allocation Table. Von DOS und Windows verwendetes Dateisystem.

FDDI
Fiber Distributed Digital Interface. Ein Hochgeschwindigkeitsnetzwerk, welches Glasfasern zur Kommunikation benutzt. Aufgrund des horrenden Preises nur in großen Netzwerken verwendet.

FHS
Filesystem Hierarchy Standard. Ein Dokument, das Richtlinien für die Einrichtung des Verzeichnisbaums auf Unix -Systemen enthält. **Mandrake Linux** hält diesen Standard (größtenteils) ein.

FIFO
First In, First Out. Eine Datenstruktur oder ein Hardwarepuffer aus denen die Dinge in der Reihenfolge entnommen werden, in die sie hineingelangt sind (Beispiel: Unix -Pipes).

Firewall, die
Eine Maschine, die den einzigen Verbindungspunkt zwischen einem lokalen und einem externen Netzwerk darstellt. Sie filtert den Datenverkehr und sichert die lokalen Rechner vor Angriffen von außerhalb.

Fokus, der
Das Fenster, welches die Tastatureingaben und Mausklicks empfängt, hat den Fokus.

Framebuffer, der
Projektion des Videospeichers in den Systemspeicher. Dies erlaubt Anwendungen auf den Videospeicher zuzugreifen, ohne das spezifische Protokoll der Karte kennen zu müssen. Highend-Workstations im Grafikbereich benutzen Framebuffer.

FTP
File Transfer Protocol. Ein Internet -Protokoll, um Dateien zwischen verschiedenen Maschinen zu transferieren.

Gateway, das
Verbindungsknoten zwischen zwei IP Netzwerken.

GFDL, die
Die *GNU Free Documentation License*. Es handelt sich um die Lizenz, unter der die Dokumentation der **Mandrake Linux** Distributionen veröffentlicht werden.

GIF
Graphics Interchange Format. Ein Dateiformat für Bilddateien. GIF Dateien können komprimiert oder animiert sein. Aufgrund lizenzrechtlicher Probleme wird die Verwendung von PNG anstelle von GIF empfohlen.

GNU
GNU's Not Unix. Von Richard Stallman Anfang der 80er gegründet, hat sich GNU zum Ziel gesetzt, auf der Grundlage von Unix ein neues, freies Betriebssystem zu entwickeln. Während sich der Kern dieses Systems, HURD, immer noch in der Entwicklung befindet, verrichten die fertigen GNU -Programme unentbehrliche Dienste auf jedem GNU/Linux -System (sowie mittlerweile auf einer Menge anderer Unix -Varianten). Die von GNU entworfene Lizenz, die GPL, dient als rechtlicher Schutz für viele GNU/Linux -Programme einschließlich des Kerns.
Siehe auch: GPL, die.

GPL, die
Die *GNU General Public License*. Die Lizenz, unter der die meisten Programme in GNU/Linux, einschließlich des Kerns, veröffentlicht werden. Sie legt u.a. Folgendes fest: Sie haben das Recht, zu jeder Software den Quellcode zu bekommen, Sie haben das Recht, diese Software zu verändern und weiter zu vertreiben, solange Sie erneut den Quellcode beilegen und den Empfängern die selben Rechte einräumen, die Ihnen eingeräumt wurden. Sie dürfen für diese Dienste Geld verlangen: die Software ist vor allem frei im Sinne der freien Rede, und erst dann im Sinne von Freibier **:-)**

GUI
Graphical User Interface. Die grafische Benutzeroberfläche mit ihrer Gesamtheit aus Menüs, Feldern, Farben, Schriftarten usw. Unter GNU/Linux Aufgabe eines X -Servers und eines Windowmanagers.

Hacker, der
Ein Programmierer.

Handbuchauszug, der
Digital vorliegender Handbuchauszug, der die Aufrufparameter eines Kommandos sowie eine Beschreibung, Verwendungszweck und mehr enthält. Wird aufgerufen mit `man` (in Anlehnung an das englische *Manual*). Häufig das erste, was Sie konsultieren sollten, wenn Sie Probleme mit einem Befehl haben **:-)**

Hardwareadresse, die
Diese Adresse identifiziert einen Rechner in einem Netzwerk eindeutig auf der Hardware-Ebene, dem sog. *Media Access Layer*. Man spricht daher auch von der MAC-Adresse.

Heim-Verzeichnis, das
Persönliches Verzeichnis eines Benutzers.
Siehe auch: Account.

Hintergrund, der
In der Shell läuft ein Prozess im Hintergrund, wenn er das Terminal, von dem er gestartet wurde, nicht mehr blockiert.
Siehe auch: Job, der, Vordergrund, der.

Host, der
Der Begriff wird meist dort verwendet, wo ein Rechner Teil eines Netzwerks ist.

HTML
HyperText Markup Language. Eine Formatierungssprache zur Erstellung von Web -Dokumenten.

HTTP
HyperText Transfer Protocol. Das Protokoll, das zur Übertragung von HTML -Dokumenten entwickelt wurde.

IDE
Integrated Drive Electronics. Der meistverwendete Festplatten-Bus in der PC -Welt. An einen IDE -Bus können bis zu zwei Geräte angeschlossen sein. Die Geschwindigkeit des Busses wird von dem Gerät bestimmt, das die langsamere Befehlskette hat.
Siehe auch: ATAPI.

Inode, der
Bildet den Eingang zum Inhalt einer Datei auf einem Unix -Dateisystem. Ein Inode wird durch eine Nummer, die Inodennummer, identifiziert, und enthält generelle Informationen über die Datei wie Zugriffszeiten, Typ und Größe (aber nicht den Namen!).

Internet, das
Großes Netzwerk, dass Rechner überall auf der Welt miteinander verbindet.

IP-Adresse, die
Numerische Adresse, bestehend aus vier Teilen, die einen Rechner im Internet eindeutig identifiziert. Sie sind hierarchisch, nach obersten, bzw. nationalen Domänen, Domänen, Subdomänen und persönliche Adressen. IP -Adressen haben die Form wie: `192.168.0.1`. Die Adresse einer Maschine kann entweder statisch oder dynamisch sein. Statische IP -Adressen ändern sich nicht, dynamische können hingegen bei jeder Anbindung an das Internet (beispielsweise bei Benutzung eines Modems) neu vergeben werden.

IP-Maskierung, die
wird benutzt um bei der Anbindung ans Internet über eine Firewall die wahre IP -Adresse Ihres Rechners nach außen hin zu verdecken. Normalerweise erhalten dann alle Verbindungen nach draußen die IP -Adresse der Firewall. Dieses Konzept könnte bei einer schnellen Anbindung, die auf eine IP -Adresse beschränkt ist sinnvoll sein, falls Sie mit mehreren Rechnern raus wollen.

IRC
Internet Relay Chat. Einer der wenigen Internet -Standards für die Übertragung dynamischer Daten in Echtzeit. Erlaubt durch die Erstellung von sog. Kanälen Privatgespräche und Datenaustausch. Die Kanäle werden von Administratoren geleitet, die im Bedarfsfall die Rechte bestimmter Benutzer einschränken dürfen. Es existieren mehrere IRC -Netzwerke, wie etwa **Undernet**, **DALnet**, **EFnet**.

IRC Kanal, der
Das ist der „Platz" auf einem IRC -Server, wo Sie mit anderen Personen *chaten* können. Kanalnamen beginnen mit einem Doppelkreuz (#).

ISA
Industry Standard Architecture. Der allererste Bus auf dem PC. Heute weitgehend ersetzt durch den PCI -Bus. Einige wenige Hardwarehersteller stellen noch Karten für ihn her. Leider ist es noch immer üblich, dass Scannern, CD -Brennern u.ä. SCSI -Karten im ISA -Format beigelegt werden.

ISDN
Integrated Services Digital Network. Ein Set von Kommunikationsstandards, die es erlauben über eine Leitung verschiedene Datentypen wie Stimme, Video, Netzwerkdaten usw. zu übertragen. Soll die herkömmlichen, analogen Telefonsysteme ersetzen. Technisch gesehen ist ISDN ein geswitchtes Netzwerk.

ISO
International Standards Organization. Eine Gruppe von Firmen, Beratern, Universitäten und anderen, welche Standards für unterschiedliche Bereiche ausarbeitet. Die Standardpapiere sind numeriert. Der Standard 9660 beispielsweise beschreibt das Dateisystem auf CD-ROM s.

ISO 8859
Der Standard ISO 8859 enthält verschiedene 8-bit Erweiterungen des ASCII Zeichensatzes. Besondere Bedeutung kommt ISO 8859-1, dem "Latin Alphabet No. 1", zu, da dieses oft implementiert wurde und nunmehr per „normativer Kraft des Faktischen" den ASCII Standard abgelöst hat.
ISO 8859-1 unterstützt die folgenden Schriften: afrikaans, baskisch, dänisch, deutsch, englisch, färöerisch, finnisch, französisch, gallisch, holländisch, isländisch, irisch, italienisch, katalanisch, norwegisch, portugisisch, schottisch, schwedisch und spanisch.
ISO 8859-1 entspricht den ersten 256 Zeichen von ISO 10646 (Unicode). Jedoch enthält dieser Zeichensatz kein Euro-Symbol. Er deckt auch nicht vollständig den finnischen und französischen Zeichen-Umfang ab. Daher wurde ein neuer Zeichensatz eingeführt, der diesen Mangel behebt: ISO 8859-15.
Siehe auch: ASCII.

ISP
Internet Service Provider. Eine Firma, die Internet -Zugänge zur Verfügung stellt.

Job, der
Im Kontext der Shell bezeichnet Job einen im Hintergrund laufenden Prozess. Es können mehrere Jobs auf der selben Shell laufen.
Siehe auch: Vordergrund, der, Hintergrund, der.

JPEG
Joint Photographic Experts Group. Ein sehr populäres Bilddateiformat. JPEG findet vor allem bei Fotografien Verwendung. Die Kompression ist nicht verlustfrei.

Kern, der
Hauptbestandteil des Betriebssystems. Er ist verantwortlich für die Bereitstellung der Betriebsmittel und das Trennen von Prozessen. Es verwaltet alle grundlegenden Operationen, die Prozesse ausführen können, um auf die Hardware zugreifen zu können.

Kill-Ring, der
In Emacs bezeichnet dieser Begriff die Menge der Textabschnitte, die seit Beginn der Sitzung ausgeschnitten oder kopiert wurden und wieder verwendet werden können. Sie sind in Form eines (virtuellen) Rings organisiert.

Klient, der
Programm oder Rechner, die zeitweilig auf andere Programme oder Rechner zugreifen, um sie Befehle ausführen zu lassen oder Informationen abzufragen. Ist eine der Komponenten eines **Klienten-/Server-Systems**.

Klienten-/Server-System
System oder Protokoll, das aus einem **Server** und einem (oder mehreren) **Klienten** besteht.

Kommandozeile, die
Wird von einer Shell bereitgestellt und erlaubt es dem Benutzer, Befehle einzugeben. Auch Gegenstand eines immerwährenden „Heiligen Krieges“ zwischen ihren Befürwortern und den Mausfetischisten **:-)**

Kompilation, die
Der Übersetzungsvorgang der Quellen (also lesbarer Textdateien), die in einer Programmiersprache geschrieben sind (etwa C) in eine Binärdatei, die vom Prozessor interpretiert werden kann.
Siehe auch: Binden, das.

Komplettierung, die
Ein Shell -Mechanismus, der die automatische Vervollständigung von Zeichenfolgen zulässt (meist Datei- oder Befehlsnamen). Wird ausgelöst durch das Drücken der **TAB**-Taste.

Kompression, die
Ein Weg, um Dateien zu packen, bzw. die Anzahl Zeichen, die über eine Kommunikationsleitung gesendet werden zu verringern. Zu diesen Programmen gehören u.a. `compress`, `zip`, `gzip` und `bzip2`.

Konsole, die
früher: Terminal. Auf GNU/Linux -Systemen gibt es sogenannte virtuelle Konsolen, mit denen Sie auf einem Monitor mehrere, voneinander unabhängige Sitzungen laufen lassen können. Standardmäßig haben Sie sechs virtuelle Konsolen, die Sie über die Tasten **ALT-F1** bis **ALT-F6** erreichen können. Die siebte virtuelle Konsole (**ALT-F7**) erlaubt es Ihnen eine Verbindung zum X window system aufzubauen. Um von ihr wieder auf eine Text-konsole zu wechseln, drücken Sie **CTRL-ALT-F1** bis **CTRL-ALT-F6**.
Siehe auch: virtuelle Konsole, die.

LAN
Local Area Network. Name für ein Netzwerk von Maschinen, die durch ein gemeinsames Kabel verbunden sind.

LDP
Linux Documentation Project. Eine Organisation, die sich um die Dokumentation von GNU/Linux kümmert (da wären Sie jetzt von selber nicht drauf gekommen, was? **:-)**), wie die HOWTOs, eine Anzahl von FAQ s und eine Reihe von Büchern.

Link, der
Referenz zu einem Inode in einem Verzeichnis, gibt dem Inode einen (Datei)namen.

Linux, das
Ein Unix nachempfundenes BS, dass für etliche verschiedene Rechnerarchitekturen existiert. Sein Hauptvorteil gegenüber anderen BS en ist, dass die Quellen jedermann zur Verfügung stehen, um sie einzusehen, Fehler zu finden und zu beheben. Der Kern von Linux wurde von Linus Torvalds geschrieben. Die meisten Programme stammen aus dem GNU -Projekt.

Major-Nummer, die
Nummer der Geräteklasse.

Markierung, die
Eine Anzeige (meistens ein Bit), die verwendet wird, um einen zweiwertigen Zustand an ein Programm zu übergeben. So gibt es etwa eine Markierung bei Dateisystemen, die angibt, dass es beim automatisierten Erstellen von Sicherungskopien mit gesichert wird.

Minor-Nummer, die
Nummer für das Gerät an sich.

Mount-Punkt, der
das Verzeichnis, an das eine Partition oder ein Gerät in das GNU/Linux -Dateisystem eingehängt wird. Ihr CD-ROM -Laufwerk wird beispielsweise in das Verzeichnis `/mnt/cdrom` eingehangen, in dem Sie dann den Inhalt der CD sehen können.

Multi-User System, das
„Mehrbenutzer“. Unix -Systeme sind von Natur aus Mehrbenutzer-Systeme, im Unterschied etwa zu Windows.

Nullzeichen, das
Das Zeichen oder die Bytenummer 0, es steht in C am Ende jeder Zeichenkette (String). Der technische Name ist `NULL`.

Nur-Lese-Modus, der
Öffnen Sie eine Datei in diesem Modus, dürfen Sie sie nur lesen, nicht jedoch verändern, bzw. die Datei löschen.
Siehe auch: Schreib-Lese-Modus, der.

Objektkode ,der
Maschinencode, generiert von Quellcode durch einen Kompilierer.
Siehe auch: Kompilation, die, Binden, das.

Pager, der
ein Programm zum blidschirmweisen Anzeigen von Textdateien mit der Möglichkeit einfach vor- und zurück zu blättern, sowie eine Suche nach Zeichenketten durchzuführen. Wir möchten Ihnen an dieser Stelle das Programm `less` ans Herz legen.

PAP
Password Authentication Protocol: Protokoll, das von ISP s zur Identifikation ihrer Klienten benutzt wird. Der Klient sendet dabei ein unverschlüsseltes Name/Passwortpaar an den Server.
Siehe auch: CHAP.

Passwort, das
ein geheimes Wort, bestehend aus Kombinationen von Wörtern und Buchstaben um etwas abzusichern. Passwörter werden im Zusammenhang mit der Anmeldung von Benutzern an Mehrbenutzersysteme, FTP -Servern, Web -Seiten, usw. verwendet. Passwörter sollten schwierig zu erraten sein, also niemals Wörter sein, die man in einem Wörterbuch finden kann und am besten auch Sonderzeichen und Ziffern beihalten. Passwörter sollen gewährleisten, das sich niemand unter Ihrem Namen irgendwo anmelden kann – ausser Ihnen selbst natürlich.

Patch, der; patchen
Datei, die eine Reihe von beabsichtigten Veränderungen an einer Quelldatei enthält, etwa um Programmfehler zu beseitigen oder neue Fähigkeiten hinzuzufügen. Das Patchen bezeichnet den Prozess des Anbringens des Patches an der oder den Quelldateien. Verwendet wird meist das Programm `patch`.

Pipe, die (Röhre, die)
Ein besonderer Unix Dateityp. Ein Programm schreibt Daten in die Röhre, die ein anderes Programm ausliest. Unix -Pipes sind FIFO -Dateien, d.h. die Daten werden in der Reihenfolge gelesen, in der sie geschrieben wurden.
Siehe auch: benannte Pipe, die.

POP
Post Office Protocol, engl. für „Postamt-Protokoll“. Das weit verbreitete Protokoll, das von vielen ISP verwendet wird, um E-Mail zu verschicken.

Portierung, die
Die Möglichkeit, ein Programm auf verschiedenen Betriebssystemen und Rechnerarchitekturen laufen zu lassen. Während die Portierung auf verwandte Betriebssysteme (wie etwa innerhalb der Unix -Familie) relativ einfach ist, erfordert dies bei Systemen unterschiedlicher Familien (wie etwa von einem Unix -System auf ein DOS -System) einen erheblichen Aufwand.

PPP
Point to Point Protocol. Das Protokoll um Daten über serielle Verbindungen zu transportieren. Es wird häufig benutzt, um IP -Pakete über das Internet zu verschicken, kann aber auch mit anderen Protokollen, wie etwa dem IPX -Protokoll von Novell verwendet werden.

Präprozessor, der
Der Präprozessor arbeitet Befehle ab, bevor der Kompilierer seine Arbeit aufnimmt. Beispiele für Anweisungen, die der C Präprozessor bearbeitet, sind `#include`, `#define`, etc.

prompt
In der Shell bezeichnet dies eine Eingabeaufforderung.

Prozess, der
In Unix ist ein Prozess eine Programminstanz mit ihrer Umgebung.

Puffer, der
Ein kleiner Speicherbereich fester Größe, der mit einer Blockdatei, einer Systemtabelle einem Prozess o.ä. assoziert wird. Der **Puffer-Cache** ist für die Kohärenz und die Verwaltung zuständig.
Siehe auch: Puffer-Cache, der.

Puffer-Cache, der
Wichtiger Bestandteil eines Betriebssystemkerns. Sorgt für die Aktualität aller Puffer, die Anpassung der Cache-Größe, die Freigabe unbenutzter Puffer u.ä.
Siehe auch: Puffer, der.

Pulldown-Menü, das
Es handelt sich um ein Menü, dass wie ein Rolladen an einer Stelle des Bildschirms heruntergelassen wird. Sie haben die Möglichkeit eine Zeile, quasi eine Lamelle des Rolladens zu markieren und dadurch eine Aktion auszulösen.

regulärer Ausdruck, der
Eine Methode um möglichst viele verschiedene Zeichenketten in einer möglichst kurzen Zeichenfolge abzubilden. Wird von vielen Unix -Programmen unterstützt, wie z.B. `sed`, awk, `grep`, perl u.v.a.m.

RFC
Request For Comments. RFC s sind offizielle Internet -Standard-Dokumente. Sie beschreiben alle verwendeten Protokolle, ihren Gebrauch, ihre Voraussetzungen usw. Wenn Sie mehr Informationen über ein Protokoll brauchen, lesen Sie im entsprechenden RFC !

Root, der
der allmächtige Benutzer eines Unix -Systems. Root (auch System-Administrator oder ausgezeichnetes Benutzerkennzeichen genannt) ist für die Instandhaltung und Überwachung des Systems verantwortlich. Er hat zu allem Zugang und darf alles machen.

Runlevel, der
Ein Zustand in dem sich Ihr System befinden kann. Anhand dieses Zustands werden verschiedenen Dienste und Server angeboten. Die verschiedenen Runlevel werden in der Datei `/etc/inittab` definiert. Normalerweise gibt es acht Standard-Runlevel: 0, 1, 2, 3, 4, 5, 6 und S. Zum Umschalten zwischen diesen kann das Privilegierte Benutzerkennzeichen `root` mit den Befehlen `init` und `telinit`.

Schreib-Lese-Modus, der
Öffnen Sie eine Datei in diesem Modus, haben Sie sowohl das Recht, den Inhalt der Datei zu lesen, als auch ihn zu verändern. Falls Sie eine Datei in diesem Modus öffnen dürfen, bedeutet das, Sie dürfen sie auch löschen.
Siehe auch: Nur-Lese-Modus, der.

SCSI
Small Computers System Interface. Ein leistungsfähiger Bus, der im Gegensatz zu IDE nicht durch die Schnelligkeit der Befehlsfolge der Peripheriegeräte begrenzt wird, parallele Zugriffe auf den Bus zulässt und Mechanismen zur Fehlerkontrolle bereithält. Für gewöhnlich wird eine Adapterkarte gebraucht, es gibt jedoch auch Hauptplatinen, die einen SCSI -Controller integrieren. SCSI -Geräte sind aufgrund der aufwendigeren Architektur meist ein gutes Stück teurer als ihre IDE -Pendants.

Shell-Skript ,das
Datei, die Anweisungen für die Shell enthält. Ein Shell-Skript muss les- und ausführbar sein.

Sicherungskopie, die
eine Methode um Ihre wichtigen Daten auf ein sicheres Medium an einem sicheren Ort zu sichern. Sicherungskopien sollten regelmäßig durchgeführt werden, insbesondere bei sicherheitsrelevanten Informationen und Konfigurations-Dateien (die wichtigsten Verzeichnisse sind also `/etc`, `/home`, und `/usr/local`). Traditionsgemäß benutzen viele Anwender `tar` zusammen mit `gzip`, bzw. `bzip2` um Sicherungskopien durchzuführen. Ebensogut können Sie aber auch die Programme `dump` und `restore` verwenden oder auch andere freie oder kommerzielle Lösungen.

Single-User System, das
„Einzelbenutzer“. Da Unix -Systeme von Natur aus Multi-User-Systeme sind, bezeichnet der Single-User-Modus den Runlevel 1, der für Reparaturarbeiten am System genutzt werden kann.

Socket, der
Dateityp, der eine Netzwerkverbindung repräsentiert.

Standard-Ausgabe, die
Dateideskriptor Nummer 1. Wird von jedem Prozess geöffnet, um darüber Daten auszugeben. Standardkanal ist das aktuelle Terminal.
Siehe auch: Standard-Fehlerausgabe, die , Standard-Eingabe, die.

Standard-Eingabe, die
Dateideskriptor Nummer 0. Wird von jedem Prozess geöffnet, um darüber Daten zu empfangen. Standardkanal ist die Tastatur.
Siehe auch: Standard-Fehlerausgabe, die , Standard-Ausgabe, die.

Standard-Fehlerausgabe, die
Dateideskriptor 2. Wird von jedem Prozess geöffnet, um Fehlermeldungen auszugeben. Standardkanal ist der Terminalschirm.
Siehe auch: Standard-Eingabe, die, Standard-Ausgabe, die.

Suchmuster, das
Eine Zeichenkette aus normalen und Sonderzeichen. Die Sonderzeichen werden von der Shell interpretiert und expandiert.

SVGA
Super Video Graphics Array: Video-Standard, definiert von VESA. Die Auflösung liegt bei 800x 600 bei 16 Farben.

Symbol, das
Unter einem (Arbeitsflächen-) Symbol (engl. *Icon*) versteht man ein kleines Bild (normalerweise 16x 16, 32x 32, 48x 48 oder 64x 64 Punkte), das als Repräsentant für ein Programm, ein Gerät oder eine Datei stehen kann.

symbolischer Verweis, der
Spezielle Datei, die nichts weiter als eine bestimmte Zeichenfolge enthält. Jeder Zugriff auf diese Datei wird vom System interpretiert als Zugriff auf die Datei, die in der Zeichenfolge referenziert wird.

TCP
Transmission Control Protocol. *Übertragungskontroll-Protokoll*. Das am weitesten verbreitete und stabilste Protokoll das IP verwendet um Pakete über ein Netzwerk zu schicken. TCP, im Gegensatz zu UDP legt eine verbindungsorientierte Übertragungsschicht über IP, um sicherzustellen, dass die Pakete auch ankommen.

Telnet, das
öffnet eine Verbindung zu einem entfernten Rechner und erlaubt es Ihnen sich an diesem Rechner anzumelden, sofern Sie dort ein Benutzerkennzeichen haben. Telnet ist das am weitesten verbreitet Protokoll dafür, es gibt aber bessere und vor allem sicherere, wie etwa `ssh`.

Texteditor, der
Programme, mit denen man Textdateien ändern kann. Die bekanntesten Editoren unter GNU/Linux sind Emacs und Vi.

Umgebung, die
Ist der Ausführungs-Kontext eines Prozesses. Sie einthält alle notwendigen Informationen die das BS benötigt, um den Prozess zu verwalten bzw. korrekt auszuführen.
Siehe auch: Prozess, der.

Umgebungsvariable, die
Teil der Prozessumgebung. Umgebungsvariablen können von der Shell aus direkt eingesehen werden.
Siehe auch: Prozess, der.

URL
Uniform Resource Locator. Bezeichnung für eine speziell formatierte Zeichenfolge, die zur Identifikation einer Datenressource im Internet oder einem anderen Netzwerk dient. Die Syntax ist wie folgt definiert:
`protocol://server.name[:port]/Pfad/`
Ist nur ein Maschinenname und das Protokoll `http://` angegeben, wird meist die Datei `index.html` der Maschine abgerufen.

Variable, die
Zeichenketten, die in Skripten und `Makefile`s verwendet werden. Sie werden durch Werte ersetzt, die ihnen irgendwo (meist am Anfang des Skripts oder des `Makefile`s) zugewiesen werden. Mit ihrer Hilfe soll die Verwaltung der Dateien vereinfacht werden.
Siehe auch: Umgebungsvariable, die.

versteckte Datei, die
In Unix werden Dateien, die mit einem . beginnen, normalerweise nicht angezeigt, es sei denn, man wünscht dies ausdrücklich. Diese Dateien sind meist Konfigurationsdateien und befinden sich im Heim-Verzeichnis des Benutzers.
Siehe auch: Heim-Verzeichnis, das.

Verzeichnis, das
Teil der Struktur des Dateisystems. Innerhalb eines Verzeichnisses werden Dateien und andere Verzeichnisse abgelegt. Mit allen Unterverzeichnissen stellt es einen Verzeichnisbaum dar. Wollen Sie den Inhalt eines Verzeichnisses ansehen, müssen Sie entweder dahin wechseln, oder es von einer anderen Stelle aus anzeigen lassen. Dateien in einem Verzeichnis heißen auch Blätter, Unterverzeichnisse Äste. Verzeichnisse unterliegen den gleichen Einschränkungen wie Dateien, ihre Rechte haben allerdings eine etwas andere Bedeutung. Die besonderen Verzeichnisse „.“ und „..“ bezeichnen das aktuelle, bzw. das übergeordnete Verzeichnis.

Verzeichnisbaumwurzel, die
das oberste Verzeichnis eines Dateisystems. Dieses Verzeichnis hat kein Vater-Verzeichnis, also zeigt „..“ wieder auf sich selbst. Die Verzeichnisbaumwurzel wird als „/“ geschrieben.

virtuelle Arbeitsfläche, die
Im X window system stellt Ihnen Ihr der Windowmanager möglicherweise mehrere Arbeitsflächen zur Verfügung. Diese nützliche Tatsache hilft Ihnen Ordnung zu halten. Anstelle von Dutzenden unterschiedlichster Fenster, die sie übereinanderlegen, können sie diese auf die verschiedenen Arbeitsflächen verteilen. Stellen Sie sich vor, Sie hätten verschiedene Monitore, zwischen diene sie hin und her wechseln können (die Art und Weise, wie Sie zwischen diesen virtuellen Arbeitsflächen wechseln können, hängt stark von der verwendeten Arbeitsumgebung ab).
Siehe auch: Windowmanager, der, Arbeitsfläche.

virtuelle Konsole, die
Sie stellt eine Möglichkeit dar, über ein physikalisches Terminal mehrere verschiedene Terminals laufen zu lassen. Auf PC s ist das physikalische Terminal die Tastatur und der Bildschirm.
Siehe auch: Konsole, die.

Vollbild, das
Eine Anwendung, die den ganzen Schirm einnimmt, im Gegensatz zu einer zeilenorientierten, wie z.B. der Zeileneditor `ed`.

Vordergrund, der
In Bezug auf die Shell ist der Prozess im Vordergrund derjenige, der gerade läuft. Sie müssen das Ende eines solchen Prozesses abwarten, um auf dieser Konsole neue Kommandos geben zu können.
Siehe auch: Job, der, Hintergrund, der.

weicher Verweis, der
Siehe „Symbolischer Verweis“.

Wertigkeit, die
Gibt die Auswertungsreihenfolge von Operanden und Ausdrücken an. So wird etwa `4 + 3 * 2` zu 10 ausgewertet, da die Multiplikation eine höhere Wertigkeit hat, als die Addition. Wollen Sie zuerst die Addition ausgewertet bekommen, müssen Sie diesen Teil klammern: `(4 + 3) * 2`. Jetzt erhalten Sie 14, da die Klammern eine höhere Wertigkeit als die Multiplikation besitzt.

Windowmanager, der
: Das Programm, das X eine Seele gibt. Verantwortlich für die Darstellung und Verwaltung von Fenstern, Menüs, Hintergrundbildern u.v.a. Es gibt Dutzende von verschiedenen Windowmanagern für GNU/Linux: AfterStep, WindowMaker, E, Fvwm, Icewm, Kwm, Blackbox usw. KDE und GNOME sind **keine** Windowmanager, sondern Umgebungen, die als Grundlage für Windowmanager dienen.

zeichenorientierte Datei, die
: Dateien deren Inhalt nicht gepuffert wird. Alle Ein-/Ausgabeoperationen werden soort durchgeführt. Sie entsprechen Datenströmen.
Siehe auch: blockorientierte Datei, die.

Stichwortverzeichnis