Linux — Ergänzungen und Mitschriften

Ortwin Ebhardt

9. August 2002

Vorwort

Dieses Lehrskript entstand vor allem deshalb, weil es zwar eine ganze Reihe von Lehrbüchern zum Thema Linux gibt, sie diese abe nicht unbedingt am CDI-Unterricht orientieren. In erster Linie wendet es sich also an CDIler, die gerne nochmal in Ruhe nachlesen wollen, was im Unterricht drangekommen ist. Ich habe mich bemüht, die wictigsten Dinge klar und verständlich aufzuschreiben, ob mir das geungen ist, wird die Zukunft zeigen.

Nachdem ich dieses Skript vollständg überarbeitet habe, denke ich, auch andere Linux-Neulinge könnten ihren Nutzen daraus ziehen. Viele Grundlagen habe ich an anderen Stellen nicht besodners gut erklärt gefunden, und wenn, dann meistens über mehrere Bücher verstreut…

Durch die Überarbeitung hat sich die Gesamtstruktur des Ganzen etwas verändert; so war der Anhang über Sicherheitshinweise einmal ein eigenes Kapitel, der Teil über Shellskript-Programmierung beschränkte sich auf die (komentierten) Lösungen von Aufgaben, und das Anfangskapitel stand irgend wo am Ende. Das Kapitel über Dateirechte hingegen gab es gar nicht…

Nachdem der ganze Kram endlich fertig war, habe ich den (auch separat erhältlichen) Workshop dazu als Anhang angefügt. Er basiert auf dem hier geschriebenen und sollte sich mit diesen Unterlagen (und ein wenig Eigeninitiative) lösen lassen.

Ich hoffe, der geneigte (oder aufrecht stehende) Leser kann seinen Nutzen daraus ziehen. Anregungen, Verbesserungen oder Fehler nehme ich gern unter der Adresse

`2nd-System-Meister@gmx.de`

entgegen.

Einleitung

Bei diesem Text ahndelt es sich um die überarbeiteten Unterrichtsmitschriften des Linux-Kurses der CDI-Klasse CS14. Er wurde zumindest teilweise umstrukturiert, um ein besseres Lernen nach diesen Unterlage zu ermöglichen.

Die ersten drei Kapitel stellen Grundlagen der Arbeitsweise des Betriebsystems, namendlich die Arbeitsweise der Shell, Prozesse und Dateirechte, dar. Das vierte Kapitel behandelt Shellskriptprogrammierung. Im fünften Kapitel wird der Boot- und Initialisierungsvorgang beleuchtet. Das sechste Kapitel spricht einige oberflächliche Grundlagen des XWindow-Systems an. Im siebten Kapitel wird die Einrichtung von Druckern unter Benutzung des `lpr` behandelt. In Kapitel acht werden Archive und Kompressionsverfahren vorgestellt, Kapitel neun hat den Kernel und die Modulverwaltung zum Thema. Kapitel zehn und elf beschäftigen sich mit der Festplattenplatzverwaltung über Quotas und LVM.

Wärend die regulären Kapitel tatsächlich Unterrichtsstoff sind (und je nach Klasse und Kapitelumfang ein- bis mehr Tage in Anspruch nehmen können), sind die Anhänge Dinge, die eventuell in übriger Zeit zum Üben oder als Zeitfüllende Information genutzt werden können.

Bei Anhang A handelt es sich um einen auf diesen Materialien beruhenden Workshop. Er ist auch einzeln erhälltlich. Anhang Bgehörte ursprünglich als Ergänzung zu dem Workshop und beschreibt die Erstellung eines einfachen `rpm`-Paketes. Anhang C behandelt die Thematik, sich auf einem System mt unbekannten Paßwort als `root` einzuloggen.

Dieser Workshop enthällt keine Befehlsrefferenz; er beinhaltet auch keine Anleitung zur Benutzung des `vi`. Es wird davon ausgegangen, daß ein ptentieller Nutzer simple Befehle bereits beherrscht oder zumindest weiß, wo er Informationen dazu bekommen kann.

Als Hilfmttel zur Erstellung des Workshopes wurden folgende im Literaturverzeichnis nochmals aufgeführte Bücher benutzt:

- *UNIX System Administration*
- Das CDI-Skript *Linux — Grundlagen des Betriebsystems*
- *Linux in a nutshell*

Für tiefergreifendes Studium sind die ersten beiden ausgesprochen empfehlenswert. Das dritte stellt tatsächlich nichts anderes als eine Befehlsreffernz dar.

Inhaltsverzeichnis

Kapitel 1

Die Shell

Eine Shell (engl. Schale) stellt bei UNIX die Schnittstelle zwischen Benutzer und System dar. Das ist in erster Linie ein Kommandozeileninterpreter — die grafische Benutezroberfläche ist lediglich ein Aufsatz (siehe auch Kapitel 6.1, S. 31). Grob gesagt ist es also ein Programm, von welchem aus weitere Programme gestartet werden können.

In der Unix-Administration ist die Shell *die* Arbeitsumgebung schlechthin; auf vielen Servern findet sich das grafische Frontend nicht einmal; neben Resourcenverschwendung stellt es auch Sicherheitslücken dar! Aus diesem Grund ist es auch für einen Linux-Administratoren erforderlich, mit der Shell äuserst vertraut zu sein.[1]

Es gibt unter UNIX eine Reihe verschiedener Shells. Sie unterscheiden sich vor allem in ihrem Funktionsumfang, in einigen wenigen Fällen auch in ihrer Arbeitsweise. Allen ist jedoch gemein, daß sich der Benutzer einem Prompt mit Eingabeaufforderung gegenüber sieht.

Unter Linux ist die `bash` (*Bourne Again SHell*) der verbreitetste Kommandointerreter, daher wird (auch in den folgeden Kapiteln) stets von ihr als zugrunde liegende Shell ausgegangen. Die meisten Beispiele funktionieren auch uner jeder beliebigen anderen Shell.

1.1 Kommando-Arten

Die `bash` kennt drei Arten von auszuführenden Kommandos: Für die Shell definierte Funktionen, integrierte Kommandos und Programme.

Shell-Funktionen: Hierbei ahndelt es sich um vom Benutzer definierte Funktionen. Sie gelten in genau der Shell, in der sie erzeugt wurden. Auf die Definition solcher Konstrukte wird in Kapitel 4.3 auf Seite 23 näher eingenagnen.

[1] Abgesehen davon, daß einige Konfigurationen unter X gar nicht erst funktionieren...

Shell-buildin Kommandos: Hierbei handelt es sich um Kommandos, die im Befehlsumfang der Shell integriert sind. Auf einem System, welches — abgesehen von der Shell — keine weiteren ausführbaren Dinge enthällt, können dies Kommandos ausgeführt werden. Typische Vertreter sind die Befehle `echo` und `cat`.

Programme: Programme sind Dateien, die unter einer Shell ausgeführt werden können. Damit eine Datei diese Vorraussetzung erfüllt muß zum einen der Benutzer der Shell das Recht haben, sie auszuführen (siehe Kapitel 3.1, S. 11), zum anderen die Datei für die Shell findbar sein. Dies kann zum Beispiel dadurch Angabe des Pfades (sowohl absolut als auch relativ) sicher getellt werden.

Die Unterscheidung dieser drei Typen ist vor allem für das spätere Verständnis des Arbeitsablaufes (siehe 1.4, S. 4) wichtig.

1.2 Variablen

Innerhalb einer Shell können Variablen gesetzt werden. Dies geschieht durch die einfache Eingabe von *`Variable = Wert`* (also zum Beispiel `zustand = blau`). Variablen gelten nur für die Shell, in der sie definiert wurden. Es ist möglich, sich den Inhalt von Variablen mit Hilfe des Befehles `echo` anzuzegen. Da dieser Befehl normaler Weise lediglich alles ausgibt, was hinter ihm steht, ist es erforderlich, ihm mittels eines `$` zu vermiteln, daß es sich um eine Variable handelt. (Wärend der Befehl `echo zustand` einfach nur `zustand` ausgeben würde, ergäbe `echo $zustand` tatsächlich `blau`.)

Das System selbst verfügt über eine reihe von Variablen, die es zum Betrieb braucht. Eine der wichtigsten heitßt `PATH`. In ihr wird angegeben, in welchen Verzeichnissen die Shell nach ausführbaren Programmen sucht. Eine ausführbare Datei, die sich in einem in der `PATH`-Variable angegebenen Verzeichnis ebfindet, kann ohne die Angabe eines Pfades direkt ausgeührt werden. Ein Beispiel für eine solche Datei ist das Programm `less`, welches sich meistens im Verzeichnis `/usr/bin` befindet. Unter SuSE kann es wichtig sein, diese Variable mittels `PATH=$PATH:/usr/local/bin` zu erweitern, da viele selbstkompilierte Programme ihre ausführbaren Programme dorthin packen.

Eine Auflistung aller irgend wie belegten Variablen läßt sich mit dem befehl `set` ausgeben. Soll eine Variable freigegeben werden, so kann dies mit dem Befehl `unset` *`Variable`* geschehen.

1.3 Umleiten von Datenkanälen

Unter UNIX können sämtliche ein Kommando betreffende Datenströme in drei Kanäle unterteilt werden, nämlich *Eingabe*, *Ausgabe* und *Fehler*. Intern werden diese drei Arten über die Datenkanäle 0, 1 und 2 abgewickelt. Dabei ist 0 die Eingabe und ist meistens mit dem Standard-Input (`stdin` verbunden. In vielen Fällen sind 1 (Ausgabe) und 2 (Fehler) identisch, beides wrd auf der Standard-Ausgabe (`stdout`) ausgegeben.

In manchen Fällen wäre es nützlich, diese Ausgabe in einer Datei zu haben, sie mit einem anderem Programm weiter zu verarbeiten oder sie schlicht und ergreifend loß zu sein. Zu diesem Zweck können diese Kanäle umgelenkt werden. Hier gibt es zwei grundlegende Möglichkeiten: Das Schreiben von Datenströmen in Dateien (das wir auch als *redirecting* bezeichnet) und das Leiten in andere Programme (dies wird meistens als *pipeing* bezeichnet). Im Zusammenhang mit diesen Möglichkeiten gibt es zudem Hifsprogramme.

1.3.1 Redirection

Um die Ausgabe einer Operation in eine bestimmte Datei zu schreiben, wird das Sonderzeichen `>` benutzt. Die Datei wird bei der Nutzung dieses Zeichens neu erstellt, falls es sie vorher schon gab, ist sie danach zunächst leer. Danach wird die Ausgabe des Befehles in die Datei geschrieben. Nach dem Befehl `ps > prozesse` würde die Liste aller in dieser Shell laufenden Prozesse in inder Datei `prozesse` stehen. Würde der Befehl erneut aufgerufen, hätte das zur Folge, daß die Datei eine neue Prozessliste enthielte; die alte wäre gelöscht.

In manchen Fällen ist es jedoch erwünscht, die Ausgaben des Prozesses kontinuirlich zu sammeln (etwa in Log-Files). Zu diesem Zweck wird das Zeichen `>>` benutzt. Auch hier wird die Datei, so es sie nicht bereits gibt, zunächst leer. In unserem Beispiel (nun in `ps >> prozesse` geändert) würde sie zunächst die erste, später dann die erste und die zweite Prozessliste enthalten.

Um die Fehlermeldungen des Programmes zu speichern, muß Programm mitgeteilt werden, daß es den Kanal 2 umleiten soll. Dies geschieht durch die Kombination `2&>`.[2] Auch hier kann zwischen Überschreiben und Anhängen gewählt werden.

In einigen Fällen mag es auch erforderlich sein, sowohl die Ausgabe als auch den Fehlerkanal umzuleiten. Die geschieht durch ein Anfügen von `2>&1`. Nach dem Befehl `cat test > ergebnis 2>&1` würde also entweder der Inhalt der Datei `test` oder eine Fehlermeldung in `ergebnis` stehen.

Besonders interessant ist das Redirecting dadurch, daß es sich auch auf Gerätedateien anwenden läßt. Die Befehlszeile `dmesg > /dev/tty9` etwa würde, wenn der ausführende Benutzer das Schreiberecht auf diese Geräedatei hat, die Ausgabe von `dmesg` auf das virtuelle, über die Taste <ALT+F9> ereichbare Terminal umleiten. Sehr beliebt ist in manchen Fällen die Umleitung `> /dev/null 2>&1`, mit der sowohl die Standard-Ausgabe als auch

[2] Eigentlich ist die Angabe des Kanals für jede Umleitung erforderlich; im Falle der Ausgabe kann das `1&` jedoch weggelassen werden. Auch bei Eingaben wird auf das `0&` verzichtet.

die Fehler ins Daten-Nirvana gepustet werden.

Ebenso, wie sich die Ausgabe umleiten läßt, kann auch die Eingabe umgeleitet werden. Dies geschieht mit dem Zeichen `<`, falls es mehr als eine Datei ist, aus der diese Eingabe stammen soll, mit `<<`. Es gibt heute nicht mehr viele Programme, bei denen ein solches Vorgehen nötig ist. In einigen Fällen kann es sich jedoch im Zusammenhang mit Gerätedateien als ausgesprochen nützlich erweisen.

1.3.2 Piping

Dieser Begriff umschreibt das Umleiten einer Ausgabe in ein weiteres Programm, welches dann seinerseits diese Ausgabe als Eingabe nutzt. Für diese *ausgesprochen* nützliche Aktion wird das Zeichen `|` benutzt.

Eine besonders praktische Anwendung ist das Suche von Zeichenketten mit Hilfe dieses Zeichens. Der Befehl `cat hallo|grep welt` gibt nur die Zeilen der Datei `hallo` aus, in dene das Wort `welt` vorkommt. Es lassen sich beliebig viele Pipes aneinanderreihen, der Befehl `cat hallo|grep welt|wc -l` etwa gibt die Anzahl der Zeilen, in denen das Wort `welt` vorkommt, aus.

Selbsverständlich läßt sich die Ausgabe eines solchen zusammengesetzten Befehles auch in Dateien umleiten. Der Befehl, um die Anzahl der Zeilen in eine Datei `welt` zu schreiben, lautet `cat hallo|grep welt|wc -l >welt`.

1.3.3 Hilfsprogramme

Es gibt zwei Programme, die im Zusammenhang mit *Redirection* und *Piping* benutzt werden,nämlich `xargs` und `tee`.

Das Programm `xargs` wird in den seltenen Fällen notwendig, in denen ein Programm keine Eingabe von `stdin` unterstützt. Der Aufruf lautet in diesem Fall `programm1 | xargs programm2`. Ebenfalls ützlich kann es sein, wenn es darum geht, in der weiterzuleitenden Ausgabe etwas zu verändern. Näheres dazu findet sich unter `man xargs`.

Das Programm `tee` ***datei*** leitet seine Eingabe nach `stdout` weiter (sie kann dann also in ein weiteres Programm geleitet werden), schreibt sie jedoch zusätzlich in ***datei***. Dies ist für die Erstellung von Logfiles ausgesprochen günstig.

1.4 Funktionsweise einer Shell

Betrachtet wird folgende Kommandofolge:

```
REDI=">"
echo Hallo $REDI /root/kn
```

Ausgegeben wird

```
Hallo > /root/kn
```

Der beabsichtigte Effekt wäre gewesen, das Wort *Hallo* in die Datei `kn` in dem Verzeichnis `root`zu schreiben. Warum dies nict funktioniert hat, wird bei Betrachtung der einzelnen Schritte , in denen die `bash` diese Zeichenfolge abarbeitet, ersichtlich:

1. Zerlegung des Befehles in mehrere Wörter (*Token*), insgesammt sind das in diesem Fall vier: `echo`, `Hallo`, `$REDI` und `/root/kn`.

2. Auflösung von Redirection / Piping, dabei wird nach bestimmten Sonderzeichen (`>`, `>>`, `<`, `<<`, `|`) gesucht. In unserem Beispiel gibt es die noch nicht, denn die Variable wurde noch nicht aufgelößt.

3. Variablensubstitution, hier wird nach dem Metazeichen `$` gesucht. Danach lauten unsere Token `echo`, `>>`, `/root/kn`.

4. Kommandosubstitution, es wird nach dem Metazeichen `` ` `` gesucht. Wird ein Befehl innerhalb zweier solcher Zeichen gefunden, muß erst dieser Abgearbeitet werden und der Output an die entsprechende Stelle gesetzt werden. In unserem Fall gibt es keinen solchen Befehl, die Zeile ändert sich also nicht.

5. Auflösung der Wildcards, diese werden entsprechend der betroffenen Dateinamen aufgelößt. In diesem Befehl gibt es keine solchen, daher passiert da auch nichts.

6. Interpretation durch die Shell: Dieses geschicht Token für Token, wobei die Shell folgende Suchreinfolge nutzt:

   (a) Ist das vorliegende Token eine Shell-Funktion, also eine für diese Shell definierte Funktion? `echo` ist keine Funktion. Falls ja, wird es ausgeführt. `echo` ist keine Shell-Funktion. **ACHTUNG!** Dieser Schritt kommt bei der `bash` als erstes, bei anderen Shells (wie etwa der Kornshell) kommt das am Schluß.

   (b) Handelt es sich bei dem Token um ein Alias? Falls ja, wird das dahinterstehende Programm oder Skript ausgeführt.

   (c) Ist ein Hash-Eintrag enthalten? Ein Hash-Eintrag besagt, daß dieser Befehl in letzter Zeit ausgeführt wurde und eventuell auch häufiger.

   (d) Handelt es sich bei dem Token um ein sogenanntes Shell-buildin-Command, also einen Befehl, der in die Shell intergiert wurde? `echo` ist ein solcher Befehl, und somit wird er ausgeführt. Dadurch ergibt sich auch, daß er alles nachfolgende auf dem Bildschirm ausgeben soll.

   (e) Wäre das nicht der Fall gewesen, würden nun die in `PATH` angegebenen Pfade durchsucht, meistens `/usr/bin`, `/usr/sbin` und so weiter... Dabei ist zu beachten, daß das erste gefundene Ergebnis genommen, wird, es ist also entscheidend, in welcher Reinfolge die Suchverzeichnisse im Pfad stehen. (Ist also wie beim Sex — der erste Treffer zählt...)

(f) Falls das alles nichts bringt, wird eine Fehlermeldung ausgegeben.

Das oben genannte Problem wäre gelößt, wenn die Schritte 1 – 5 zwei mal durchgeführt würden. Da für einige Anwendungen dieses Verhalten ausgesprochen nützlich wäre, gibt es den Befehl `eval`. Dieser erzwing genau dieses verhalten. Im Falle der Eingabe `eval echo $REDI /root/kn` wirkt vor allem in Schritt 3 aus, da nun ein entsprechendes Metazeichen vorhanden ist (nämlich `>`). Dem zur Folge wird die Ausgabe des ersten ausführbaren Tokens (`echo`) in die Datei `/root/kn` umgeleitet.

1.5 A Shell within a shell (within a shell...)

Wie bereits am Anfang dieses Kapitels geschrieben wurde, handelt es sich bei einer Shell um ein Programm, aus dem andere Programme gestartet werden können. Das bedeutet, daß in einer Shell auch eine weitere Shell gestartet werden kann. Im Falle der `bash` geschieht dieses durch den Befehl `bash`. Um Verwirrungen vorzubeugen, wird die erste Shell im folgenden als *Elternshell*, die zweite, durch den Befehl gestartete als *Kind-* oder *Subshell* bezeichnet. Um die Kindshell zu beenden und in die Elternshell zurück zu kehren, wird der Befehl `exit` benutzt.

In der Kindshell gilt keine der eventuell innerhalb der Elternshell definierten Variablen. Falls innerhalb der Kindshell Variablen definiert werden, so gehen sie bei Rückkehr in die Elternshell verloren.

Innerhab der Elternshell gibt es eine Möglichkeit, dafür zu sorgen, daß eine Variable auch in einer Kindshell verfügbar ist. Diese besteht darin, die Variable zu exportieren. Der entsprechede Befehl dazu lautet `export` ***`Variable`***. Eine innerhalb einer Kindshellexportierte Variable ist zwar in allen Kindshells dieser Shell vorhanden, nicht jedoch in der Elternshell.

Kapitel 2

Prozesse

Prozesse und Programme sind eng miteinander verbunden. Um späteren Verwchslungen vorzubeugen, werden beide wie folgend unterschieden:

Ein **Programm** ist ein Stück compilierten Codes, der statisch[1] auf der Festplatte liegt. Ein **Prozess** hingegen beschreibt einen Vorgang innerhalb des Systemes und ist eine äußerst dynamische Sache. Er verfügt über eine Start und eine Endzeit, und er muß folgende Elemente verwalten:

- Den (statischen) Programmcode
- Eine Kopie des Programmcodes in das RAM des Rechners, das sogenannte *Textsegment*
- Die betroffenen (sprich: zu ändernden) Dateien
- Den Stack[2]
- Verschiedene Kenngrößen (auch als *Environment* oder *Prozessumgebung* bezeichnet)

Stark vereinfacht gesagt ist ein Prozess also ein tatsächlich ablaufendes Programm.

2.1 Grundlegendes

Prozesse werden anhand ihrer *Prozess-ID*, kurz *PID* unterschieden. Die PIDs werden vom System vergeben fortschreitend vergeben.[3]

[1] Das bedeutet *feststehend*.

[2] Das ist ein besonderer Teil des Speichers, er wird unter anderem für das reibungslose Ablaufen von Multitasking-nutzenden Programmen benötigt.

[3] Das bedeutet, keine Nummer kommt zwei mal vor. Wird beispielsweise eine `bash` mit der PID 666 gestartet und später beendet, wird eine neue `bash` nicht die PID 666, sondern irgend eine spätere erhalten.

In UNIX sind Prozesse streng hirarchisch angeordnet. Dabei gibt es einen Prozess `proc`. Dieser startet einen weiteren Prozess namens `init`. Dieser wird später als Wurzel des Prozessbaumes gesehen. `init` startet sämmtliche anderen Prozesse, daher wird er manchmal auch als *Vater aller Prozesse* bezeichnet (wie Hotshots — die Mutter aller Filme).

Auch die login-Vorgänge sind Prozesse, die von `init` gestartet werden. Auch die `bash`, unter der ein eingeloggter Benutzer arbeitet, ist ein von `init` gestarteter Prozess.

Diese Prozesse können ihrerseits ebenfalls Prozesse starten (etwa wenn ein Benutzer in seiner `bash` einen Befehl ausführt). Sie werden dann als *Elternprozesse*, die gestarteten als *Kindprozesse* bezeichnet. Um das ganze zu veranschulichen, findet sich in Abbildung 2.1 eine schematische Darstellung eines Prozessbaumes.

Die Kommunikation von Prozessen untereinander ist nur von Eltern zu Kinden möglich. Um einen Datenaustausch zwischen verschiedenen Prozessen zu ermöglichen, werden unter anderem die *Named Pipes* benutzt. (Andere Möglichkeiten sind Semaphore, Monitore und andere Dinge, von denen der normale Mensch garnichts wissen will…)

Ein Prozess, der beendet wurde, liefert an seinen Elternprozess einen Wert zurück. Dieser Rückgabewert ist immer dann 0, wenn der Prozess ohne Fehler terminiert. Lautet er anders, ist irgend ein Fehler aufgetreten.

2.2 Arbeiten mit Prozessen

Da jeder Vorgang auf dem System einen ihm zugehörigen Prozess besitzt, ist das Wissen über diese Prozesse für die erfolgreiche Administration unabdingbar.Prozesse können gestartet und beendet werden, andere manipulationsmöglichkeiten gibt es nicht.

2.2.1 Starten von Prozessen

In Hinblick auf Prozesse werden zwei Arten von Kommandos unterschieden Die *Shell-build-in-commands* und die *extern commands* (siehe auch Kapitel 1.1, S. 2). Erstere sind in die Shell integriert. Bei ihrem Aufruf wird kein neuer Prozess gestartet. Daher sind sie für diese Betrachtung nicht weiter von Interesse. Die anderen, externen Kommandos werden bei ihrem Aufruf als neue Prozesse gestartet.

So ein Start kann entweder ganz normal im Vordergrund geschehen (durch eine ganz normale Eingabe), wordurch wärend der Ausführung das entsprechende Terminal blockiert ist. Durch ein abschließendes `&`-Zeichen kann das Programm auch im Hintergrund gestartet werden; an der Shell kann trotzdem weitergearbeitet werden. (Auch wenn die gelegendlichen Ausgaben der Prozesse etwas störend wirken können… Um hier Abhilfe zu schaf-

```
init-+-atd
     |-bdflush
     |-cron
     |-gpm
     |-inetd
     |-kapm-idled
     |-7*[kdeinit]
     |-kdeinit-+-kdeinit
     |         |-kdeinit---bash---xjewel
     |         |-kdeinit---bash---emacs
     |         `-kdeinit---bash---xdvi.bin
     |-kdeinit---extensionproxy
     |-kdeinit---cat
     |-keventd
     |-khubd
```

Abbildung 2.1: Ausschnitt aus `pstree`

fen, können die Meldungen wie in Kapitel 1.3.1, S. 3 beschrieben umgeleitet werden.)

2.2.2 Informationen über laufende Prozesse

Um einen Prozess beenden zu können, ist es nötig, seine PID zu kennen. Außerdem ist es manchmal recht nützlich, zu sehen, welche Prozesse überhaupt auf dem System laufen. Unter Linux gibt es eine ganze Reihe von Befehlen, um diese Informationen zu erhalten. Die drei vermutlich meist-genutzten sind `ps`, `pstree` und `top`.

`ps`: Der Befehl gibt laufende Prozesse aus. Wird er ohne weitere Parameter aufgerufen, handelt es sich nur um die Prozesse, die zum einen vom aufrufenden Benutzer stammen und zum anderen auch von einer Konsole gestartet haben (also über ein sogenanntes *Kontroll-Terminal* verfügen).

Beliebte Schalter dafür sind `-a`. `-x` und `-f` (auch als `-axf`), sie sorgen dafür, daß *alle* laufenden Prozesse angezeigt und in einer annähernd baumartigen Struktur angezeigt werden.

`pstree`: Dieser Befehl stellt eine Momentaufnahme des zur Ausführungszeit des Befehles bestehenden Prozessbauems dar. Und da sich unter diesem Satz niemand was vorstellen kann, wird das in der Abbildung 2.1 mal dargestellt.

`top`: Der Befehl `top` macht sehr ähnliches wie `ps`, allerdings kommen noch ein paar mehr Informationen, und das ganze ist hübsch grafisch. Allerdings werden nur so viele Prozesse angezeigt, wie auf den Bildschirm passen.

2.2.3 Beendigung von Prozessen

Um einen Prozess zu beenden, wird der Befehl `kill` benutzt. Trotz des Namens bedeutet das nicht zwingender Maßen, daß der Prozess einfach nur beendet wird. Tatsächlich wird dem Prozess ein Signal geschickt, und je nach Signal wird entschieden, was passiert. Der vollständige Aufruf des Befehles lautet `kill -`*`signalnummer PID`*. Folgende Signalnummern sind beonders wichtig:

1	SIGHUP	Beendet den Prozess und startet ihn neu.
9	SIGKILL	Beendet den Prozess ohne Kompromisse.
15	SIGTERM	Sendet dm Prozess eine Aufforderung, sich zu beenden.

Wird `kill` ohne die Angabe einer Nummer aufgerufen, sendet er automatisch das Signal 15. In den meisten Fällen reicht das aus. Sollte sich ein Prozess als zu hartnäckig erweisen, hilft Signal 9.

In manchen Fällen soll mehr als ein Prozess vernichtet werden, etwa, wenn ein bestimmtes Programm mehrfach läuft und sich ncht beenden läßt. In diesem Fall hilf der Befehl `killall`. Hierbei wird nicht die `PID`, sondern der Name des Prozesses angegeben. (Der Befehl `killall bash` würde somit alle auf dem Rechner eingeloggten Personen ausloggen. . .)

Kapitel 3

Dateirechte unter UNIX

Nicht jeder Benutzer soll alles machen können; je mehr ein Benutzer beweriken kann, desto mehr Schaden kann er anrichten. Aus diesem Grund können Dateien und Ordner mit unterschiedlichen Rechten versehen, die regeln, was ein Benutzer eigentlich tun und lassen kann.

Bei Aufruf des Befehles `ls -l` ***datei*** wird die angegebene ***datei*** mit einer ganzen Reihe von Informationen ausgegeben. Das kann zum Beispiel so aussehen:

```
# ls -l options
-rw-rw-r--    1 atrocity users          64 Aug  3 20:35 options
```

Von rechts nach links besagt diese Aussage, daß diese Datei den Namen `options` hat (und keinen Link darstellt), daß sie am 3. August um 20:35 Uhr das letzte mahl verändert wurde, 64 Blocks groß ist (das ist im Augenblick unwichtig), daß sie der Gruppe `users` gehört und dem Benutzer `atrocity` gehört. Die darauf folgende 1 gibt die Anzahl der Hardlinks auf diese Datei an (das ist für dieses Kapitel ebenfalls nicht weiter von Interesse).

Der letzte Teil besagt, wer welche Rechte auf diese Datei hat. Dabei gibt es drei Gruppierungen, von der jede jeweils über drei Rechte verfügen kann.

3.1 Die drei Rechte

Grundsätzlich gibt es unter UNIX drei Rechte: Das Recht, eine Datei zu lesen (Kennzeichen `r`, **r**ead), das Recht, eine Datei zu schreiben (Kennzeichen `w`, **w**rite) und das Recht, eine Datei auszuführen (Kennzeichen `x`, e**x**ecute).

Bei gewönlichen Dateien besagten das Lese- und das Schreibrecht genau das, was sie sagen: Wer diese Rechte inne hat, der kann die Datei lesen oder schreiben. Das Ausführrecht besagt, daß diese Datei als Programm ausgeführt werden kann.

Bei Verzeichnissen (die ja ebenfalls ein besodnere Art von Dateien darstelen) besagt das Schreib-Recht, daß der Inhaber in diesem Ordner etwas schreiben kann. Das Lese-Recht besagt, daß der Inhaber zwar in den ordner wechseln, sich aber nicht dessen Inhalt (mittels `ls`) ansehen kann. Das Ausführrecht schließlich besagt, daß der Inhaber nicht in den Ordner wechseln kann. (Hat er aber noch das Lese-Recht, wird er sämmtliche dortdrin enthaltenen Dateien mit einer Fehlermeldung ausgegeben bekommen, wenn er `ls` ***verzeichnis*** eingibt.)

Für gewöhnlich werden diese Rechte in Form eines Trippels angegeben, etwa `r-x` für eine Datei, die zwar ausgeführt und gelesen, nicht aber überschrieben werden darf. In manchen Fällen ist es jedoch erforderlich, die sogenannte *Oktalschreibweise* zu verwenden. In dieser werden die Rechte mit Zahlen versehen. `x` entspricht dabei 1, `w` entspricht 2 und `r` entspricht 4. Durch addieren aller zutreffenden Rechte ergibt sich das Recht als Oktalzahl. `r-x` ergibt zum Beispiel 4+1=5. Das Oktalrecht 3 hingegen kann sich nur aus den Faktoren 1 und 2 zusammen setzen, es entspricht also der Kombination `-wx` (was, zugegebener Maßen, nicht sehr sinnvoll erscheint).

Zum näheren Verständnis sei noch angemerkt, daß Linux die Benutzer intern nicht über ihre Namen, sondern über Nummern, sogenannte UIDs (***U****ser* ***ID***) verwaltet. Diese werden bei der Erstellung eines Benutzers festgelegt sind einmalig. Das führt dazu, daß ein Benutzer, der gelöscht, und mit identischem Namen hinzugefügt wird, nicht automatisch seine vorherigen Rechte hat. Der Benutzer `root` hat *immer* die UID 0. Zur Laufzeit befindet sich diese Zahl in der Variable `UID`.

Wird ein Programm ausgeführt, geschieht das mit der UID des Benutzers, der es ausführt. Daraus ergeben sich auch seine Rechte. Somit kann ein normaler Benutzer ruhig ein Programm, das `root` gehört, ausführen dürfen — solange er selbst keine Rechte auf wichtige Systemdateien hat, wird auch das Programm diese nicht ändern können.

3.2 Die drei Gruppierungen

Nun sind das vor der Datei nicht drei, sondern zehn Stellen. Von links nach rechts betrachtet zeigt die erste davon an, um was für eine Datei es sich handelt; ist es eine ganz gewöhnliche Datei, dann steht hier ein `-`, Verzeichnisse sind mit einem `d` (wie ***d****irectory*) gekennzeichnet. Danach folgen drei der unter 3.1 beschriebenen Trippel.

Das erste Trippel beschreibt die Rechte des Besitzers der Datei. In dem Beispiel hat also `atrocity` das Recht zu Lesen und zu Schreiben.

Das zweite Trippel beschreibt die Rechte der Gruppe; jeder Benutzer, der Mitglied in dieser Gruppe ist, hat die hier aufgeführten Rechte.[1] In dem Beispiel haben de Mitgleider der Gruppe `users` also das gleiche Recht wie der Besitzer. Auch Gruppen werden Intern über eine Zahl verwaltet, die sogenannte GID (***G****roup* ***ID***).

[1] Es ist Möglich, Mitglied verschiedener Gruppen zu sein.

Das letzte Trippel beschreibt die Rechte aller anderen Benutzer. In dem Beispiel kann also jeder die Datei lesen.

Rechte können von jedem Benutzer, der das `w` recht der Datei hat, geändert werden. Dazu gibt es den Befehl `chmod`. Um einer Gruppierung ein Recht zuzuweisen, wird ein Gruppierungskürzel + das entsprechende Recht angegeben. Für den Besitzer der Datei lautet das Kürzel `u` (wie ***u**ser*, für die Gruppe `g` (wie ***g**roup*) und für alle anderen `o` (wie ***o**thers*). Außerdem gibt es noch das Kürzel `a` (wie ***a**ll*), um ein Recht für alle zu setzen.

Um einer Gruppierung ein Recht zu entziehen, wird das Kürzel - das Recht angegeben.

Um allen Benutzern des Systemes die Möglichkeit zu geben, `options` auszuführen, wäre der Befehl
`chmod a+x options`
um danach der Gruppe das Schreiberecht zu entziehen
`chmod g-w options`

In einigen Fällen ist es jedoch praktisch, mehrere Rechte zur Gleichen Zeit zu ändern. Unter diesen umständen wird anstelle der Gruppenkürzel und der Rechte die komplette oktalschreibweise (also drei Zahlen für die drei Trippel) angegeben. Die eben in zwei Schritten durchgeführte Operation ließe sich so mit dem einen Befehl
`chmod 766 options`
bewerkstelligen.

Für die Gruppe eines ausgeführten Programmes gilt das gleiche, wie für seinen Besitzer; es gilt immer die Gruppe des ausführenden Benutzers.

3.3 Sonderrechte

In einigen Situationen ist es erforderlich, daß ausgeführte Programme doer Verzeichnise sich nicht den genannten Regeln entsprechend verhalten, etwa wenn Programme mit `root`-Rechten ausgeführt oder ein Verzeichnis zwar von allen genutzt, die dort gespeicherten Dateien aber nur von ihren Besitzern überschrieben werden sollen. zu diesem weck gibt es die Sonderrechte SUID und SGID und das Sticky-Bit.

3.3.1 SUID und SGID

Wie bereits erwähnt hat ein Programm immer die Rechte inne, die es durch den ausführenden Benutzer (und dessen Gruppe) hat. Das ist nicht immer günstig; das Programm `passwd` etwa muß auf die Dateien `/etc/passwd` und `/etc/shadow` zugreifen. Das darf aber nur `root`. Damit aber auch einfache Benutzer ihr Paßwort ändern können, wird dm System mitgeteilt, daß dieses Programm mit den Rechten seines Besitzers (nämlich `root`) ausgeführt werden soll.

Dieses sogenannte SUID-Bit (so genannt, weil der Prozess mit der UID des Besitzers läuft) wird in den Rechten als ein `s` anstelle des `x` im Benutzertrippel angezeigt. Für den ziemlich unwahrscheinlichen Fall, daß eine Datei zwar mit den Rechten des Benutzers ausführbar ist, selbiger es aber nicht selbst ausführen kann, wird es mit einem `S` dargestellt.

Das SGID-Bit ist prinzipiell das gleiche wie das SGID-Bit, bloß bezieht es sich nicht auf den Besitzer, sondern auf die Gruppe. Es wird anstelle des `x`-Rechtes des Gruppentrippels angezeigt. Im Zusammenhang mit Vrzeichnissen kommt ihm eine zusätzliche Sonderrolle zu: Dateien, die innerhalb eines mit dem SGID-Bit versehenen Verzeichnisses erstellt werden, gehören automatisch der Gruppe des Verzeichnisses.

3.3.2 Das Sticky-Bit

Die ursprüngliche Idee des Sticky-Bites war, einen Prozess nach seiner Beendigung weiterhin im Betriebsystemspecher zu behalten (es also sozusagen darin kleben zu lassen). Damit sollte der Overhead häufig benutzer Programme klein gehalten werden. Auf heutige Systemen ist es nicht mehr nötig und wird kaum noch verwendet. Es wird mit einem `t` (für *text save mode*) anstelle des `x` im Allgementrippel (oder einem `T`, falls die Allgemeinheit das Programm nicht ausführen darf) angezeigt.

Immer noch sehr häufig wird es hingegen bei Verzeichnisen genutzt, wo es eine andere bedeutung hat: Im Falle eines temporären Verzeichnisses mag es sinnvoll sein, daß zwar alle Benutzer auf dieses Verzeichnis zugreifen und ihre Dateien dort lesen, schreiben und ausführen können, sie dieses aber eben nur für *ihre* Dateien können. Genau das bewirkt das Sticky-Bit. Das Verzeichnis `/tmp` ist ein besonders gutes Beispiel dafür, denn es hat fast immer dieses Bit gesetzt.

3.3.3 Setzen von Sonderrechten

Auch die Sonderrechte lassen sich mittels `chomd` setzen; dies geht entweder über die Kürzel `s` (beziehungweise `S`) und `t` (auch hier gibt es `T`), oder die Oktalwerte.

Fals das Recht über Kürzel gesetzt werden soll, erfolgt es auf die Bewerte Methode `chmod` ***Kürzel+Gruppierung***, wobei SUID dem Besitzer (also `u`), GID der Gruppe (also `g`) und das Sticky-Bit allen anderen (also `o`) zugeordnet ist.

Falls diese Rechte als Oktalwert gesetzt werden sollen, wird der normale Oktalwert um eine Stelle am Anfang ergänzt. Diese neue erste Stelle setzt sich ausschließlich aus den Sonderrechten zusammen, wobei das Sticky-Bit der 1, GID der 2 und SUID der 4 entspricht. Die Rechtekombination `rwsr-Sr-x` entspricht damit `6741`.

Kapitel 4

Shellskript-Programierung

Ein Shellscript ist erst einmal nichts weiter als eine Reihe von in einer Datei stehenden Befehlen. Unter DOS haben diese Dateien die Endung `.bat` und werden *Batch-Dateien* genannt. Die bekanteste ist die `autoexec.bat`.

Auf dene ersten Blick erscheinen Shellscripte nur eine weitere Verkomplizierung des ohnehin schweren Admintage zu sein; imerhin ist es erforderlich, in einem gewisen Maße zu progarmmieren!

Bei näherem Hinsehen zeigt sich, daß ein Shellscript dem Benutzer eine ganze Menge Arbeit abnehmen kann. Falls ein bestimmter Arbeitsvorgang stets identisch ist, brauchen nicht mehr alle Schritte einzeln durchgeührt werden, es reicht, das Skript aufzurufen. Durch die Möglichkeit, einem Skript Parameter zu übergeben, können die Vereinfachungen noch mehr bewirken. Unter UNIX werden Scripte sehr, sehr häufig verwendet.

Aufgrund der verschiedenen unter UNIX verfügbaren Shells gibt es eine Reihe von verschiedenen Scrip-Arten. Unter Linux ist die `bash` die Standardshell, daher werden hier vornehmlich `bash`-Skripte behandelt. Die in 4.2 genannten Beispiele dürften jedoch auch unter den meisten anderen Shells laufen. Funktionen hingegen werden von einigen sehr alten Implementationen der `sh` nicht unterstützt.

4.1 Funktionsweise von Shell-Skripten

Ein Shellskript wird für gewöhnlich über seinen Namen aufgerufen. Vorraussetzung ist jedoch, daß der ausführende Benutzer zum einen das Recht darauf hat, es auszuführen, und zum anderen das System weiß, wo die Datei zu finden ist. In dieser Beziehung verhällt sich ein Skript nicht anders als ein Programm.[1]

Bei Aufruf eines Skriptes wird normaler Weise eine Subshell (siehe Kapitel 1.5, S. 6) gestartet, in der die in der Datei angegebenen Befehle ausgeführt werden. Dies hat den Vorteil, daß eventuelle Verzeichnis-Änderun-

[1] Um genau zu sein *ist* ein Skript nichts anderes als ein Programm.

```
#!/bin/bash
rm -f `find / -name $1`
echo Alle Dateien $1 in diesem System wurden geloescht.
```

Abbildung 4.1: Das Skript `delall`

gen, Variablen und ähnliches nicht in der normalen Arbeitsshell verändert werden. Zu beachten ist, daß auch eventuelle Variablen der Eltershell nicht zur Verfügung stehen, es sei denn, sie wurden exportiert.

Soll dieser Effekt verhindert werden (etwa um mit dem Skript Variablen anzupassen), wird vor den Aufrud des Skriptes ein `.` gesetzt (also Beispielsweise `. /usr/userskripte/skript`).

4.2 Skript-Arten und die dazugehörigen Befehle

Für die im Folgenden aufgeführten Skript-Arten glt jeweils, daß alles über das vorhergesagte.

4.2.1 Einfache Shelskripte mit Parametern

Ein einfaches Shellskript besteht nur aus einer Aneinanderreihung von Befehlen. Hierzu ist vermutlich kein Beispiel nötig, und auch der Nutzen ist ziemlich eingeschränkt.

Intressanter wird es, wenn dem Skript ein Parameter übergeben wird. Innerhalb des Skriptes kann auf diese Parameter mittels der Variablen `$1` bis `$`*n* abgefragt werden, wobei *n* die Anzahl der übergebenen Argumente angibt. Die Anzahlselbs befindet sich in `$#`.

Das Skript `delall` in Abbildung 4.1 auf Seite 16 ist ein Beispiel ür so ein einfaches Skript. In der ersten Zeile wird klargestellt, mit welcher Shell es ausgeführt werden soll. Wird diese Zeile weggelassen, wird die aktuelle benutzt. In der Zweiten werden mit Hilfe des `find`-Befehles die als erstes Arguemnt übergebenen Dateien gesucht und dann mittels `rm` gelöscht. Die letzte Zeile gibt lediglich eine Bestätigung aus. Das Skript wird mittels `delall` *Datei* aufgerufen.

Eine andere Anwendung dieser ausgesprochen einfachen Shellskripte ist das Setzen wichtiger Variablen. Auch die Konfigurationsdateien des Systemes selbst sind in dieser Beziehung nichts weiter als Shellskripte. Es ist allerdings zu beachten, daß diese Skripte tunlchst mit einem `.` ausgeführt werden, da ihre Variablendefinitionen sonst nach Beendigung des Skriptes verschwinden.

Sämtliche Aktionen eines Shellskriptes lassen sich auch direkt in der Shell eingeben; anstelle der Zeilenumbrüche müssen dafür Semikolen (`;`) gesetzt werden.

4.2.2 Der `if`-Befehl

Um ein Skript bei bestimmten Vorraussetzungen zu verzweigen gibt es den Befehl `if`.

Eine `if`-Verzweigung besteht Grundlegend aus zwei Teilen, der Bedingung und der Ation, die im Falle einer zutreffenden Bedingung ausgeführt werden soll, also:

```
if Bedingung
  then Aktion
fi
```

Bei dem `fi` handelt es sich um den (notwendigen) Hinweis, daß die `if`-Konstruktion abgeschlossen ist. Die Einrückung dient der (nicht notwendigen...) Übersichtlichkeit.

Um spätere Shellskripte besser zu verstehen, ist es erforderlich, zu sehen, wie `fi` genau arbeitet, sprich: woran es erkennt, daß eine Bedingung zutrifft oder nicht. Dies entscheidet `if` anhand es Rückgabewertes des mit der Bedingung verbundenen Prozesses[2] (siehe 2.1, S. 8). Falls er 0 ist, sieht `fi` die Bedingung als erfüllt an, bei *jedem anderen* Rückgabewert nicht.

In einigen Fällen ist es erünscht, in genau diesem Fall etwas anderes als die `then`-Aktion auszuführen. Hierzu gibt es die `else`-Verzweigung. Damit sieht die konstruktion nun so aus:

```
if Bedingung
  then Aktion 1
  else Aktion 2
fi
```

Die letzte Form der Erweiterung besteht in dem Kommando `elif`. Dieses wird gebraucht, falls im Falle eines Nichtzutreffens der `if`-Bedingung eine andere Bedingung abgeprüft und die `else`-Aktion nur in diesem speziellen Falle gebraucht werden soll. Hinter `elif` können noch weitere `elif`-Verzweigungen und sogar eine `else`-Verzweigung stehen, also:

```
if Bedingung 1
  then Aktion 1
  elif Bedingung 2
    then Aktion 2
  elif Bedingung 3
    then Aktion 3
  else Aktion 4
fi
```

Es stellt sich nach wie vor die Frage, was denn nun so ein Bedingung sein kann. In den meisten Fällen wird das Programm `test` dafür benutzt. Es hat eine Vielzahl von Möglichkeiten (siehe dazu `man test`), mit Hilfe von Parametern (manchmal auch ohne...) beliebige Objekte zu überprüfen.

[2] Hiermit ist auch klar, was als Bedingung in Frage kommt — jede Aktion, die einen Rückgabewert hat.

```
if [ $UID = 0 ]
  then PS1=Weiter\#
  elif [ $UID = 666 ]
    then PS1=His\ Infernal\ Majesty\>\>
  else PS1=Bitte\ Eingabe\>\>
fi
```

Abbildung 4.2: Das Skript `prompt`

Es kann auf zweierlei Weisen aufgerufen werden: Entweder direkt als `test` *`-parameter Objekt`* oder in Form von `[` *`-parameter Objekt`* `]`. Da die zweite, kürzere Form im Zusammenhang mit Shellskripten übersichtlicher ist, wird sie im folgenden ausschließlich benutzt.

Ein Beispiel für ein solches, mittels der vorgestellten Mittel kostruierten Skriptes stellt dasSkript `prompt` in Abbildung 4.2 auf Seite 18 dar. Der Aufruf (am Besten aus der Datei `profile.local` im Verzeichnis `etc`) muß mit einem . beginnen, um das Skript in der laufenden Umgebung auszuführen und keine Subshell zu starten.

Anhand der UID des sich jeweils einloggenden Benutzers wird hier entschieden, wie sein Eingabeprompt aussieht. Falls es sich um den Benutzer `root` handelt (der ja immer die UID 0 hat, siehe Kapitel 3.1, S. 12), wird als Prompt `Weiter#` ausgegeben. Im Falle der UID 666 handelt es sich offenbar um einen höllischen Gast, daher der Promt `His Infernal Majesty>>`. Alle anderen Benutzer bekommen den Prompt `Bitte Eingabe>>`.

Dem aufmerksamen Leser wird aufgefallen sein, daß auch `test` Variablen ebenso wie `echo` in durch `$` gekennzeichneter Form benötigt.

4.2.3 Der `case`-Befehl

In Fällen, in denen je nach Inhalt einer Veraible die Aktion des Skriptes bestimmt werden soll, bietet sich der `case`-Befehl an. Dieser überprüft eldiglich, ob der Inhalt einer Variable mit einer seiner Handlungsmöglichkeiten übereinstimmt. Der Syntax lautet

```
 case Variable in
     Möglichkeit 1) Aktion 1 ;;
     Möglichkeit 2) Aktion 2 ;;
     Möglichkeit 3) Aktion 3 ;;
esac
```

Das (notwendige) `esac` beschreibt — ähnlich wie `fi` im Falle von `if` — das Ende des `case`-Konstruktes. In Abbildung 4.3 auf Seite 19 wurde das Skript aus Abbildung 4.2 auf Seite 18 nochmals mit `case` implementiert. Der `*` erfüllt als Wildcard eine ähnliche Aufgabe wie das ursprüngliche `else`.

```
case $UID in
        0) PS1=Weiter\# ;;
      666) PS1=His\ Infernal\ Majesty\>\> ;;
        *) PS1=Bitte\ Eingabe\>\> ;;
esac
```

Abbildung 4.3: Das Skript `cprompt` — `prompt` mit `case`-Anweisung

4.2.4 Interaktive Eingaben

Die Ausgabe einer Variable oder eines Textes durch ein laufendes Skript stellt kein Problem dar; Auch die Übergabe eine Parameters funktioniert. Doch wie steht es mit Eingaben? Hierfür gibt es den Befehl `read` *`Variable`*. Hierbei ist zu beachten, daß die Variable nicht mittels eines `$`-Zeichens gekennzeichnet werden muß. Ein Anwendungsbeispiel findet sich in dem Skript in Abbildung 4.4 auf Seite 20, zusammen mit der Erklärung der `while`-Schleife.

4.2.5 Die `while`-Schleife

Eine `while`-Schleife wird genau so lange durchlaufen, wie eine bestimmte Bedingung erfüllt ist. Als Bedingung kommen alle Dinge in Frage, die auch als Bedingung einer `if`-Verzweigung funktionieren. Die zu wiederholenden Aktionen werden von den Schlüsselworten `do` und `done` eingeschlossen, also:

```
 while Bedingung
   do
   Aktion
done
```

Ein Beispiel für solch eine Schleife findet sich in dem Skript `vf1000` in Abbildung 4.4 auf Seite 20. Das Skript dient der Bestimmung aller Vielfachen eines gegebenen Nenners im Bereich von Nenner bis 1000.

Hier wird als erstes dafür Sorge getragen, daß als Nenner keine 0 in Frage kommt. Solange diese Bedingung nicht erfüllt ist, wird die erste `while`-Schleife durchlaufen. (Das wäre sicherlich auch einfacher möglich gewesen, aber sicher nicht so schick...) Sobald der Nenner bestimmt (und in die Variable `nenner` geschrieben) wurde, wird die Datei `Vielfaches` durch Löschen initialisiert (der Schalter `-f` unterdrückt Fehlermeldungen, so es die Datei nicht gibt). Danach wird eine normale `while`-Schleife von 1 bis 1000 durchlaufen. Falls Laufvariable modulo `nenner` keinen Restwert ergeben, kann der Inhalt der Laufvariablen offenbar durch `nenner` geteilt werden. Also wird der Inhalt der Variable an die Datei `Vielfaches` angehängt.

```
nenner=0
while [ $nenner = 0 ]
  do echo -n 'Bitte geben Sie einen Nenner ein: '
  read nenner
done
rm -f ~/Vielfaches
i=1
while [ $i -le 1000 ]
  do
  if !(($i % nenner))
    then echo $i >>~/Vielfaches
  fi
  i=$[$i+1]
done
```

Abbildung 4.4: Das Skript `vf1000`

4.2.6 Die `for`-Schleife

Diese Schleife unterscheidet sich von der vorhergegangenen vor allemdadurch, daß hier keine Bedingung erfüllt sein muß, sondern eine Reihe von vorgegebenen Werten abgegangen wird, die eine Laufvariable annimmt. Sobald diese Werte abgearbeitet wurden, wird die Schleife beednet. Die auszuführenden Aktionen werden ejdoch auch hier von `do` und `done` eingeschlossen. Die Konstruktion sieht damit so aus:

```
for a in Wert 1 ...Wert n
  do
  Aktion
done
```

Das Skript `wochentage` in Abbildung 4.5 auf Seite 20 gibt die Wochentage und eine Meldung, der vieweilte Tag der Woche jeder Tag ist, aus.

Bei Benutzung dieser Schleife ist zu beachten, daß in derbedingungsliste verwendete Wildcards entsprechend der Dateien in dem Verzeichnis, von

```
b=1
for a in Montag Dienstag Mittwoch Donnerstag Freitag Samsag Sonntag
  do
  echo $a ist der $b. Tag der Woche
  b=$[$i+1]
done
```

Abbildung 4.5: Das Skript `wochentage`

```
for a in *
  do
  file $a
done
```

Abbildung 4.6: Das Skript `lsi`

welchem aus das Skript gesartet wurde, aufgelößt werden.

Das Skript `lsi` in Abbildung 4.6 auf Seite 21 geht alle Dateien im Verzeichnis durch und gibt Informationen über sie aus.

4.2.7 Beenden von Skripten und der Rückgabewert

In eingen Fällen ist es Sinnvoll, ein Skript vorzeitig zu beenden. Zu diesem Zweck gibt es den Befehl `exit` *`Rückgabewert`*, wobei Rückgabewert in der Regel 0 sein sollte, falls das Skript planmäßig abgelaufen, und irgend ein anderer Wert, falls ein Fehler aufgetreten ist.

Das Skript `cwrap` in Abbildung 4.7 auf Seite 22 stellt einen simplen Wrapper zum `cp`-Befehl dar.[3]

Es überprüft als erstes, ob genau ein Argument übergeben wurde. Falls das nicht der Fall ist wird die `if`-Verschachtelung beendet und ein Fehler geworfen. Anderenfalls wird überprüft, ob es sich um eine normale Datei handelt. Falls nicht, werden beide `if`-Verschachtelungen beendet und daher der Fehler geworfen. Falls alle Kriterien erfüllt sind, wird überprüft, ob das Verzeichnis vorhanden ist. Falls nicht, wird es erstellt und für alle Benutzer zugänglich gemacht. Danach wird die Datei kopiert und das Skript beendet.

Anzumerken bleibt, daß das Verzeichnis `home2` nur von `root` erstellt werden kann... Es stellt sich die Frage, wozu diese Rückgabewert überhaupt gut ist. Das Skript `mcpy` in Abbildung 4.8 auf Seite 22 benutzt ihn;

Zunächst wird mit der `if`-Verzweigung dafür gesorgt, daß, so das Skript nicht korrekt ausgeführt wird, eine Fehlermeldung ausgegeben wird. (Eigentlich ist sie unnötig, da das Skript das auch schon tut). Dies geschieht über den Rückgabe-Wert des Skriptes, der im Falle eines Fehlers 1 (unter Unix FALSE), sonst 0 (Also TRUE) ist. Danach wird eine `while`-Schleife so lange durcheführt, bis die Variable `l` den Wert 1 annimmt. Innerhalb der Schleife wird der Benutzer zur Eingabe der zu kopierenden Datei aufgefordert und auf selbige das Skript angewandt. Das Verfahren zur Fehleridentifikation ist identisch mit dem bereits beschriebenen. Danach wird in jedem Fall gefragt, ob eine weitere Datei kopiert werden soll. Im Falle einer anderen Eingabe als `j` wird die Variable `l` auf 1 gesetzt und die Schleife beendet.

[3] Ein Wrapper ist ein Skript, welches den Zweck eines anderen Befehls hat, diesen aber modifiziert.

```
if [ $# = 1 ]; then
  if [ -f $1 ]; then
    if [ ! -d /home2/save ]
        then mkdir -p /home2/save
          chmod 777 /home2/save
    fi
    cp $1 /home2/save
    exit 0
  fi
fi
echo Fehler!
exit 1
```

Abbildung 4.7: Das Skript `cwrap`, ein einfacher Wrapper

```
if ! cwrap $1
  then echo 'Da ist wohl etwas schief gegangen!'
  else echo 'Alles OK'
fi
echo -n 'Wollen Sie eine weitere Datei kopieren? [n]'
read an
if [ "$an" != 'j' ]
  then l=1
  else l=0
fi
while [ $l = 0 ]
  do echo -n 'Bitte geben Sie die Datei an! '
  read datei
  if ! cwrap $datei
    then echo 'Da ist wohl etwas schief gegangen!'
    else echo 'Alles OK'
  fi
  echo -n 'Wollen Sie eine weitere Datei kopieren? [n]'
  read an
  if [ "$an" != 'j' ]
    then l=1
  fi
done
```

Abbildung 4.8: Das Skript `mcpy`

```
dauwatch()
  {
    while true
      do
      sleep 3
      if who|grep DAU >/dev/null
        then echo 'DAU ist da'>/dev/tty4
        else echo 'DAU ist weg'>/dev/tty4
      fi
    done
  }
```

Abbildung 4.9: Das Skript `init_dauwatch` zur Deklaration der Funktion `dauwatch`

4.3 Funktionen

Eine Funktion ist im Prinzip eine Variable, die Shell-Code enthällt und von der Shell ausgeführt wird (siehe Kapitel 1.1, S. 1). Sie gilt nur für die jeweilige Shell und kann nicht exportiert werden. Gelöscht wird sie mit `unset`. Die Deklaration kann direkt von der Kommandozeile aus geschehen, die Syntax lautet ***funktion()*** { ***aktion 1*** ; ...; ***aktion n*** ; }, also zum Beispiel:
`hallo()` { `echo hallo > /dev/tty9 ;` }
Diese Funktion würde, falls sie von `root` definiet und ausgeführt würde, das Wort `hallo` auf das durch `<STRG+F9>` erreichbare virtuelle Terminal schreiben. In Zukunft wird jedoch die Schreibweise in Skriptform bevorzugt; sie ist etwas übersichtlicher. Soll eine Funktion über ein Skript defniert werden, so ist es erforderlich, dem Skrit bei seiner Ausführung einen Punkt (`.`) voran zu stellen, da die Deklaration sonst nur in der Subshell des Skriptes erfolgt (siehe 4.1, S. 15). Ein Beispiel für solch eine Skriptdefinition stellt die Funktion `dauwatch` dar, die über das Skript `init_dauwatch` in Abbildung 4.9 auf Seite `fnkt:dauwatch` deklariert wird.

Durch diese Funktion wird alle drei Sekunden eine Meldung auf `tty4` ausgegeben, falls irgend wo ein Benutzer eingelogt ist, dessen Loginname die Zeichenkette `DAU` enthällt. Dabei ist zu beachten, daß nur `root` diesen Prozess starten kann oder der Ausführende gleichzeitig auf dem Terminal eingeloggt sein muß. Sonst hat das Programm keinen Zugriff auf `/dev/tty4`.

In der `if`-Verknüpfung wird abgefragt, ob sich unter den zur Zeit auf dem Rechner eingeloggten Benutzern DAU befindet. Sollte ein Benutzer in seinem Namen die Zeichenkette DAU haben, führt das ebenfalls zur Meldung, aber wer diese Zeichenkette im Namen hat, verdient es auch, gemeldet zu werden... `who` gibt an, wer alles eingeloggt ist, `grep DAU` sucht aus diesen Ergebnissen alle heraus, welche die Zeichenkette `DAU` enthalten. Das Ergebnis wird, da es nicht von Interesse ist, ins Datennirvana geschickt; falls es kein Ergebnis gibt, liefert der Ausdruck insgesammt eine 1, also FALSE, zurück.

Eine Warnung: Falls diese Funktion im Hintergrund ausgeführt werden soll, ist es erforderlich, sich die PID zu merken; mittels `ps` wird es nicht unter seinem Namen angegeben!

Kapitel 5

Booten und Initialisieren

Der Bootvorgang eines Systemes setzt sich aus den Systemunabhängigen Startsequenzen des Rechners und der Initialisierung des Betriebsystemes zusammen.

5.1 Der Bootvorgang

Das Booten des betriebsystems läuft folgendermaßen ab:

1. BIOS, POST und so weiter werden durchlaufen
2. Das BIOS geht zur Ausführung auf das Bootmedium (Diskette, CD-ROM, Festplatte, Bootprom. . .)

Bisher hatte der Bootvorgang noch nichts mit irgend einem Betriebsystem zu tun. oBdA[1] *wird davon ausgegangen, bei dem Bootmedium handele es sich um eine Festplatte.*

3. Der Masterbootrecord (MBR) wird angesprungen. Dieser besteht aus 4×16 Bytes Partitionstabelle, der *Magic Number* $AA55$ und 446 Bytes. In letzteren findet sich der *First Stage Boot Loader* (FSBL). Bei dem FSBL handelt es sich um eine reine Prozessoranwendung (also reinen, Maschienenspezifischen Binary-Code).
4. Der FSBL springt zum *Second Stage Boot Loader* (SSBL), der auf der Platte liegt. Da der FSBL nur BIOS-Befehle zur Verfügung hat, muß der SSBL innerhalb der von BIOS verwaltbaren 1024 Zylinder liegen.

   Der SSBL befindet sich in der Datei `/dev/boot.b`
5. Der SSBL kann Menüs darstellen und Tastatureingaben verarbeiten. Entsprechend der Eingaben wird der Betriebsystemkern (*Kernel*) geladen. Dieser findet sich in der Datei `/boot/vmlinuz`. Dieser Kernel

[1] oBdA: Ohne Beschränkung der Allgemeinheit

```
boot    = /dev/hda
#vga    = 771
read-only
#menu-scheme = Wg:kw:Wg:Wg
lba32
prompt
timeout = 80
#message        = /boot/message
#append = irgendwas
restricted
password = no8tp6X
```

Abbildung 5.1: Globaler Teil einer `lilo.conf`

enthällt alle Treiber, um das Betriebsystem zum Laufen zu bekommen. Insbesondere enthällt es die Treiber für die Dateisysteme.

6. Der Kernel initialisiert das Dateisystem. Insbesondere wird das Wurzelverzeichnis `/` eingerichtet. Danach wird das Prozesssystem initialisiert. Sobald dieser Schritt abgeschlossen ist, wird eine Meldung (zur Zeit *Version 2.78 running* oder so etwas).

Alle Meldungen, die der Kernel im letzten Schritt ausgibt, können mit dem Befehl `dmesg` ausgegeben werden.

5.2 Der `lilo`

Linux bootet nicht von allein; ebenso wie Windows oder OS/2 benötigt es einen sogenannten Bootloader. Bei allen Distributionen wird der Bootloader `lilo` mitgelfert, auch wenn der `grub` diesen in manchen ablösen soll.

Um den `lilo` einzurichten, wird das Programm `/dev/lilo` ausgeführt. Anhand der Datei `/etc/lilo.conf` wird entschieden, wie und wo das geschieht. Die Datei folgt von ihrem Aufbau den Shell-Konventionen. Im Klartext bedeutet das, das Kommentare mit dem `#` begonnen werden.

Die Datei gliedert sich in zwei Teile: den *Globalen* und den *Lokalen*. Der erste kümmert sich um alle Dinge, die für *alle* Konfigurationen gilt. In Abbildung 5.1 auf Seite 26 findet sich ein Beispiel. Die erste Zeile gibt an, wo denn der FSBL hingeschrieben werden soll. In diesem Fall ist es der MBR der ersten Platte. Sollte er nun auf eine Diskette geschrieben werden, so müßte der Eintrag `boot = /dev/fd0` lauten.

Der Befehl `prompt` bewirkt, daß überhaupt gewartet wird, bis der Benutzer im Menü etwas angewählt hat. Gewartet wird 80 Sekunden — das wird mit der Zeile `timeout = 80` bewirkt.

```
  image      = /boot/vmlinuz
    label    = linux
    root     = /dev/hda7
#   password = no8tp6X

  image      = /boot/vmlinuz.suse
    label    = suse
    root     = /dev/hda7

  optional

  other  = /dev/hda1
    label  = windows
```

Abbildung 5.2: Lokaler Teil einer `lilo.conf`

Die auskommentierte `append`-Zeile dient der Übergabe von Kernelparametern. Die können zum Beispiel für die Initialisierung von PnP-Geräten.

Die letzten beiden Zeilen bewirken, daß zusätzliche Kernelparameter nur unter Eingabe eines Paßwortes (`no8tp6X`, nein, daß ist *nicht* das Paßwort *meines* `lilos`...) eingegeben werden können.Wird auf `restricted` verzichtet, muß vor *jedem* Start ein Paßwort eingegeben werden.

Der lokale Teil ist für die verschiedenen bootbaren Systeme zuständig. Ein Beispiel findet sih in Abbildung 5.2 auf Seite 27.

Linux unterscheidet zwischen zwei Arten von startbaren Systemen: Linux und anderen. Falls ein Linux gestartet werden soll, muß ein Kernelimage angegeben werden. Das ist die bereits in 5.1, S. 25 beschriebene Datei `vmlinuz`. Außerdem muß die Partition angegeben werden, die später als `/` dienen soll. Über den Befehl `label` wird angegeben, unter welchem Namen dieses System gestartet wird. **Der Name hat keine Auswirkungen auf das System; ein Linux-System kann durchaus unter dem Namen *Windows* angesprochen werden.**

Falls kein generelles Paßwort im globalen Teil eingerichtet wurde, kann hier für jedes System ein eigenes (oder keins) angegeben werden. Falls ein generelles Paßwort existiert, hier aber trotzdem eines angegeben wird, gilt das hier angegebene, lokale Paßwort für dieses System. Zu beachten ist, daß der Befehl `restricted`, so im globalen Teil angegeben, sich auch auf die lokalen Teile auswirkt. Anders ausgedrückt: Wird Global der `restricted` angegeben, können keine Paßwörter für das Starten des Systemes angegeben werden. Wird hier zusätzlich zu einem normalen, generellen Paßwort ein `restricted` angegeben, ist ebenfalls kein Paßwort zum Starten des Systems notwendig, sondern nur, falls Parameter angegeben werden sollen.

Sobald die Datei `/etc/lilo.conf` zur Zufriedenheit des Benutzers konfiguriert wurde, muß das Programm `lilo` ausgeführt werden. (Das liegt meistens im Verzeichnis `/sbin` und somit im Arbeitspfad; allerdings können nur Administratoren dieses Programm ausführen, andere können nicht auf die `lilo.conf` zugreifen.) Das Programm schreibt die FSBL und SSBL neu.

Nach jeder Änderung der Bootbetreffenden Dateien, sei es der Kernel, sei es eine der Konfigurationsdateien, muß `lilo` über den Aufruf des Befehles `/sbin/lilo` eingerichtet werden.

5.3 Die Initialisierung

Der Kernel initialisiert wie bereits erwähnt das Prozess-System. Dies geschieht durch den `init`-Prozess (der dann auch die PID 1 hat). Als nächstes wird die Datei `/etc/inittab` abgearbeitet. In dieser Datei finden sich verschiedene Einträge, die besagen, was zum Beginn eines jeden Runlevels zu geschehen hat. Runlevel gibt es insgesammt 7 verschiedene, welche die Art der Ausführung charakterisieren. (Mehr dazu an anderem Ort.)

Die Einträge in der Datei sind nach folgendem Muster aufgebaut:

`Name : Runlevel : Aktion : Prozess`

Der `Name` ist eine meist zweistellige Kennung, die den Eintrag identifiziert. Er ist vollkommen willkürlich, muß aber innrhalb der `inittab` eindeutig sein. Der `Runlevel` gibt an, für welchen Runlevel der Eintrag gilt. Die `Aktion` besagt, wie etwas passiert, und der `Prozess` besagt, was passiert. Als Beispiel wird ein Teil der `inittab` betrachtet. Das Beispiel entstammt einem SuSE-Rechner, auf anderen Distributionen kann es ein anderes Aussehen geben.

```
#
# /etc/inittab
#
# Copyright (c) 1996 SuSE GmbH Nuernberg, Germany.
# All rights reserved.
#
# Author: Florian La Roche <florian@suse.de>, 1996

# default runlevel
id:3:initdefault:

# check system on startup
# first script to be executed if not booting in emergency (-b) mode
si:I:bootwait:/etc/init.d/boot
```

Der erste Eintrag legt den Defaultrunlevel fest. In diesem Fall lautet der 3 Aufgrund dieser Funktion von `initdefualt` ist es nicht nötig (oder auch nur sinnvoll), einen auszuführenden Prozess anzugeben.

Als zweites wird definiert, welches Skript zu Beginn des Bootvorganges ausgeführt werden soll. Dieses ist bei SuSE das Skript `/etc/init.d/boot`. Hier werden unter anderem die Verzeichnisse gemountet. Das ganze geschieht noch vor dem Erreichen eines Runlevels, daher das `I`.

```
# /etc/init.d/rc takes care of runlevel handling
#
# runlevel 0  is  System halt (Do never use this for initdefault)
# runlevel 1  is  Single user mode
# runlevel 2  is  Local multiuser without remote network (e.g. NFS)
# runlevel 3  is  Full multiuser with network
# runlevel 4  is  Not used
# runlevel 5  is  Full multiuser with network and xdm
# runlevel 6  is  System reboot (Do never use this for initdefault)
#
l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
#l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
l6:6:wait:/etc/init.d/rc 6
```

Hier wird geklärt, was bei Erreichen der verschiedenen Runlevel passieren soll. Glücklicher Weise sind in diesem Fall die Runlevel auch kommentiert worden, so daß der Admin weiß, was jetzt welcher tut. Diese Runlevels können innerhalb der verschiedenen UNIXe variieren. Im Prinzip wird in allen Runleveln ein Skript (`/etc/init.d/rc`) ausgeführt, dem der Runlevel übergeben wird. `wait` sorgt dafür, daß bis Ende des Skriptes gewartet wirt, ehe irgend etwas anderes passiert.

```
# getty-programs for the normal runlevels
# <id>:<runlevels>:<action>:<process>
# The "id" field  MUST be the same as the last
# characters of the device (after "tty").
1:2345:respawn:/sbin/mingetty --noclear tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6
```

In diesem Abschnitt werden die verschiedenen virtuellen Terminals definiert. Hier werden alle Runlevel angegeben, für die der Eintrag gilt. `respawn` sorgt dafür, daß der Prozess nach Beendigung erneut gestartet wird. Der Befehl `/sbin/mingetty` ist das Programm, daß die virtuellen Terminals darstellt, `tty`*x* die Gerätedatei, die jeweils dafür zuständig ist.

Nur noch mal zur Erinnerung: Der Parameter `--noclear` vor dem `tty1` sorgt *nicht* für den häßlichen SuSE-Rahmen, er sorgt dafür, daß die Bootmeldungen stehen bleiben und nicht gelöscht werden.

Kapitel 6

Einiges zum XWindow-System

Um den egsammten Hintergrund des XWindow-Systemes wieder zu geben, wäre sehr, sehr viel Platz nötig. Dieses Kaptel beschränkt sich auf die technischen Hintergründe und die (ansatzweise) Konfiguration „von Hand".

6.1 Das Prinzip

Das hinter dem XWindow-System stehende Prinzip ist Netzwerkbasiert. Es gibt einen Client und einen Server. Client und Server können Räumlich getrennt sein, dann wird ein Netzwerkprotokol dazwischengeschaltet. Werden sie auf dem selben Rechner ausgeführt, ist das nicht nötig.

Der *XServer* kümmert sich um die gragische Darstellung — vor Ort auf dem Rechner. Die Anwendungen, also *Clients*, können an vollständg anderen Orten liegen. Das können auch Progamme sein, die aufwendige Daten berechnen. Sie bedienen sich zur Darstellung des XServers. Das führt zu der ausgesprochen perversen Situation, daß ein XClient im Keller steht und viele XServer von verschiedenen Orten aus darauf zugreifen — also genau anders herum, als das bei einem Client/Server-Modell sonst üblich ist.

6.2 Der Aufbau

Hier wird nur die „technisch" Seite betrachtet, die für die nachfolgenden Punkte wichtig sind.

Das XWindow-System wird meistens über den Befehl `startx` gestartet. Bei diesem Befehl handelt es sich zum einen um ein Programm, zum anderen um ein Script. Die Script-Variante macht nichts anderes, als alle Meldungen und Fehlermeldungen sowohl in die Datei `.X.err` umzuleiten als sie auch auf dem Schirm auszugeben.

`startx` seinerseits startet den XServer (der allein noch nichts nützt) und einen *Windowmanager*. Letzterer stellt eine Sammlung von Clients dar und kümmert sich um die grafische Oberfläche.

Der XServer wird über einen Link mit der Bezeichnung `X` im Verzeichnis `/usr/bin/X11` gestartet. Dieser Link verweist auf `/var/X11R6/bin/X`. Das wiederum verweist dann auf den entsprechenden XServer. Dieser *Kunstgriff* ist erforderlich, da `/usr/bin` für gewöhnlich schreibgeschützt ist und daher wenige Möglichkeiten bietet, den Link zu ändern.

6.3 Das Einrichten

Diese Einrichtung orientiert sich an einer Problematik, die sich auf THORBENs Rechner stellte. `SaX` wäre sicherlich gegangen, aber der richtet nur XServer der Version 3x ein. `SaX2` funktionierte wohl, aber ich habe nicht gerafft, daß der dann auf Terminal 7 läuft und wir erst hinwechseln müßten. Nun ja... Die Einrichtung orientiert sich wie gesagt am XFree86 Version 4. Immerhin ist dieses Beispiel praktisch, um die Konfigurationsdateien zu erklären.

6.3.1 Dateien

Eines der Probleme, die sich stellten, was das Finden von `X`; durch irgend welche Gründe lief der ins Leere. Die Dateien müssen aussehen, wie in 6.2, S. 32 beschrieben. Konkret läßt sich das mit den folgenden Befehlszeilen einrichten:

```
rm -f /usr/bin/X11/X /var/X11R6/bin/X
ln -s /usr/bin/X11/XFree86 /var/X11R6/bin/X
ln -s /var/X11R6/bin/X /usr/bin/X11/X
```

Die erste Zeile beseitigt die bereits vorhandenen Links. Die zweite Zeile setzt einen Link auf den XServer (Verion 4, daher auch nur ein Name ohne Zusätze), der letzte ist der Rücklink auf das `bin`-Verzeichnis. (In den meisten Fällen wird es nicht nötig sein, den Rücklink zu setzen. Hm. Das sollte vielleicht vorher Überprüft werden, etwa mit `ls -l /usr/bin/X11/X`, das sollte irgend wie ergeben, daß es ein Link ist...)

Als nächstes sollte eine `XF86Config` erstellt werden. Da wir eine vierer wollen, geht das mit dem Befehl `xf86config4`. Es geht in Ordnung, in allen Punkten auf `Enter` zu hauen; Sobald die Datei vorhanden ist, können die Werte entsprechend der im nächsten Kapitel aufgeführten Änderungen modifiziert werden.

6.3.2 Die Datei XF86Config

Hier wurden Zeilen, die irrelevant sind, und Kommentare weggelassen. Das geschah aus Platzgründen. Wer da mehr wissen möchte, kann sich an den Kommentaren orientieren. Die sind zwar auf Englisch, aber früher oder später sollte das sowieso jeder, der im IT-Bereich tätig ist, lernen. Das geht besonders gut durch das Sehen englischsprachiger Filme (nicht gerade *Das Boot*...) oder das Lesen ebensolcher Bücher.

```
# **********************************************************************
# Input devices
# **********************************************************************

# **********************************************************************
# Core keyboard's InputDevice section
# **********************************************************************

Section "InputDevice"

    Identifier  "Keyboard1"
    Driver      "Keyboard"
# For most OSs the protocol can be omitted (it defaults to "Standard").
# When using XQUEUE (only for SVR3 and SVR4, but not Solaris),
# uncomment the following line.

#    Option     "Protocol"      "Xqueue"

    Option "AutoRepeat" "500 30"
```

Hier werden Dinge zur Tastatur eingestellt, am Ende der Sektion gibt es nochmals wichtige Einträge:

```
# These are the default XKB settings for XFree86
#    Option "XkbRules"    "xfree86"
#    Option "XkbModel"    "pc101"
#    Option "XkbLayout"   "us"
#    Option "XkbVariant"  ""
#    Option "XkbOptions"  ""

#    Option "XkbDisable"

    Option "XkbRules"   "xfree86"
    Option "XkbModel"   "pc105"
    Option "XkbLayout"  "de"

EndSection
```

Hier wird das deutsche Tastatur-Layout eingerichtet. Für die meisten Leute recht wichtig.

Auch der nächste Abschnitt ist wichtig; handelt es sich doch um die Sektion, in der die Kabelratte spezifiziert wird!

```
# **********************************************************************
# Core Pointer's InputDevice section
# **********************************************************************

Section "InputDevice"

# Identifier and driver

    Identifier  "Mouse1"
    Driver      "mouse"
    Option "Protocol"    "PS/2"
    Option "Device"      "/dev/psaux"
```

Hier stellte sich das interessante Problem, daß THORBENs Rechner im Gegenzug zu meinem mit dem Treiber nichts anfangen konnte. Nachdem ein neuerer Treiber eingesetzt wurde, lief alles einwandfrei:

```
# Identifier and driver

    Identifier  "Mouse1"
    Driver      "mouse"
    Option "Protocol"    "imps/2"
    Option "Device"      "/dev/psaux"
```

Wichtig ist vor allem der Port `/dev/psaux`; diese Gerätedatei verweist auf den PS/2-Port. Bei seriellen Mäusen lautet die entsprechende Gerätedatei meistens `/dev/ttyS0`, das ist der erste der beiden seriellen Anschlüsse.

Der nächste interessante Punkt findet sich in der Monitor-Sektion:

```
# **********************************************************************
# Monitor section
# **********************************************************************

# Any number of monitor sections may be present

Section "Monitor"

    Identifier  "My Monitor"
    HorizSync   31.5 - 64.3
    VertRefresh 50-100

EndSection
```

Bei den hier angegebenen Werten handelt es sich um Standart-Werte; bessere sind möglich (etwa die exakten des Monitors...), aber hiermit geht es.

Als nächstes muß die Grafikkarte spezifiziert werden. In unserem Fall sind das zwei verschiedene: Standart-VGA (der sollte *immer* da sein) und die interne ATI-Karte. Bei VGA wurden eine Menge Kommentare weggeschnitten.

```
# **********************************************************************
# Graphics device section
# **********************************************************************

# Any number of graphics device sections may be present

# Standard VGA Device:

Section "Device"
    Identifier  "Standard VGA"
    VendorName  "Unknown"
    BoardName   "Unknown"
    Driver     "vga"
EndSection

# Device configured by xf86config:

Section "Device"
    Identifier  "ATI integrated on Intel Maui MU440EX motherboard"
    Driver      "ati"
    #VideoRam    4096
    # Insert Clocks lines here if appropriate
EndSection
```

Abgesehen von dem Eintrag bei `Driver` ist es vollkommen egal, was hier eingetragen wird.

Es folg die Sektion, in welcher der Bildschirm spezifiziert wird. Für jede Farbtiefe ist ein eigener Eintrag erforderlich. Die standart-Farbtiefe ist 16 Bit, also sechzehntausendirgendwas.

```
# **********************************************************************
# Screen sections
# **********************************************************************

# Any number of screen sections may be present.  Each describes
# the configuration of a single screen.  A single specific screen section
# may be specified from the X server command line with the "-screen"
```

```
# option.
Section "Screen"
    Identifier  "Screen 1"
    Device      "ATI integrated on Intel Maui MU440EX motherboard"
    Monitor     "My Monitor"
    DefaultDepth 16

    Subsection "Display"
        Depth       8
        Modes       "640x480" "800x600" "1024x768" "1280x1024"
        ViewPort    0 0
    EndSubsection
    Subsection "Display"
        Depth       16
        Modes       "1024x768" "640x480" "800x600" "1024x768" "1280x1024"
        Virtual     1024 768
        ViewPort    0 0
    EndSubsection
    Subsection "Display"
        Depth       24
        Modes       "640x480" "800x600" "1024x768" "1280x1024"
        ViewPort    0 0
    EndSubsection
EndSection
```

Die Reihenfolge der Zahlen gibt an, wann welche Auflösung gestartet wird. In der 16er-Sektion wird der XServer mit 1024×768 gestartet, mit `Strg+Alt++` wird als nächstes 640×480, dann 600×800 und so weiter erreicht. (Ja, ich weiß. . . diese Reihenfolge ist nicht besonders sinnvoll.)

Der Eintrag `Virtual 1024 768` setzt die Größe des virtuellen Schirmes, ie bei einer Auflösung von z.B. 640×480 kann innerhalb eines 1024×768 großen Feldes herumgescrollt werden.

Die letzten beiden Zeilen sollten standardmäßig drinn sein, zur Sicherheit gebe ich sie nochmal an:

```
    InputDevice "Mouse1" "CorePointer"
    InputDevice "Keyboard1" "CoreKeyboard"

EndSection
```

Dies ist, wie bereits gesschrieben, nicht die komplette `XF86Config`; aber es gibt ein Beispiel ab, wie der XServer konfiguriert werden kann. Mit Hilfe dieser Anleitung sollte es möglich sein, selbst in Fällen, wo die tollen SuSE-Schöpfungen `SaX(2)` versagen, eine laufende Konfiguration aufzusetzen.

Kapitel 7

Drucker (`lpr`)

Es gibt verschiedene Methoden, einen Drucker unter Linux anzusprechen. Allen ist gemein, daß sie in irgend einer Form ein Netzwerk benötigen, und sei es nur ein Loopback-Netz. In dieser Beziehung weißt es also Parallelen mit dem XWindow-System (vergleiche Kapitel 6.1, S. 31). Die für Linux wichtigsten sind `lpr` und `CUPS`. Letzteres ist wohl nach allem, was ich gehört habe, über einen Webbrowser recht einfach mittels Mausklick zu konfigurieren. Dem zur Folge findet sich in der freien Wildbahn der andere... Den `lpr` gibt es in zwei Fassungen, den `lpr-old` und den `lprng` (lpr NextGeneration).

7.1 Grundlegendes und Geschichtliches

Vor einiger Zeit erfand die Firma Adobe eine Druckersprache namens PostScript. In dieser Sprache lassen sich Schriften und ähnliches programmieren, wobei ein Schriftfont als eine Reihe von mathematischen Funktionen, nach denen das genaue Aussehen berechnet werden kann, zu sehen ist. Diese Sprache wird häufig in Laserdruckern eingerichtet.

UNIX setzt genau an diesem Punkt an; Dateien, die gedruckt werden sollen, werden erst einmal in PostScript übersetzt. Der PostScriptwust (jeder, der schon einmal eine PostScriptdatei „roh“ gesehen hat wird mir bei dieser Wordwahl zustimmen) wird dann an den Drucker gesendet.[1] Falls der Drucker kein PostScript kann, muß der Code erst einmal in eine entsprechende Druckersprache (bei HP-Druckern meistens HPGL/2) übersetzt werden, sonst besteht wenig Aussichten, mit dem Ergebis glücklich zu werden.

Für die Umwandlung wird ein sogenannter *Filter* benutzt. Ein Filter ist ein Programm, das einen Input bekommt, mit selbigen irgend etwas tut, und dann einen Output liefert. In der Befehlszeile `ps -axf|grep bash` ist `grep`

[1] Ein Drucker, der PostScript kann, ist wirklich extrem hart im Nehmen; wenn er an einer parrallelen oder seriellen Schnittstelle hängt, kann eine bestehende PostScript-Datei sogar so wie sie ist mit `cat` auf die Gerätedatei umgeleitet werden (etwa `cat text.ps > /dev/lp0`), auch das führt zu einem korrekten Ausdruck.

`bash` der Filter, der dafür sorgt, daß nur Prozesse, die den Schlüsselausdruck *bash* enthalten, ausgegeben werden. Für den Druckbetrieb sind `aps` und `magic` die verbreitetsten. Druckerfilter arbeiten mit Vorfiltern. Diese analysieren die zu druckenden Dateien und entscheiden anhand ihrer Ergebisse (etwa: ASCII, jpeg-Grafik, PS-File...), wie das über den eigentlichen Filter zu übersetzen ist.

In diesem Sinne wird auf dem Rechner selbst nicht der Drucker angelegt, sondern die sogenannte *Druckerwarteschlange*. Die dort liegenden Dateien (meistens zwei, eine Datendatei und eine Kontrolldatei) wird dann auf den Drucker geleitet.

Hier zeigt sich bei Remote-Druckern ein weiterer Unterschied: Die Filterung kann entweder vor Ort vorgenommen werden und die Datei gefiltert an den Druckerserver gesendet werden. Der muß sie dann nur noch auf seinem Port ausgeben. Das bedeutet weniger Arbeit für den Server, bei sehr vielen Aufträgen ist das sicherlich vom Vorteil. Die andere Möglichkeit besteht darin, dem Server die Filterarbeit zu überlassen. Der Vorteil hiervon liegt darin, daß der Filter ansich nur auf einem Rechner im vollen Umfang konfiguriert werden muß.

7.2 Betroffene Dateien

Abgesehen von den Filterdateien, die je nach Filter varieieren können und daher nicht weiter besprochen werden (zumindest nicht von mir), gibt es vor allem zwei Dateien, die für den Druckbetrieb verantwortlich sind. Die wichtigste ist die Datei `/etc/printcap`, in welcher die Eigenschaften der Druckerwarteschlange stehen. **Achtung!** In einigen neueren Systemen ist diese Datei leer; Eigentlich ist sie eine Abart der Datei `/dev/termcap`, eine Datei, in der Terminals, die irgend wie an den Rechner angeschlossen werden können, näher spezifiziert werden. In neuerer Zeit wird wieder dazu übergegangen, die Einträge der `printcap` wieder in die `termcap` einzutragen.

Eine andere wichtige Datei ist die `/etc/hosts`. Hier wird nicht nur die Namensauflösung nach IP-Adressen vorgenommen — genau genommen ist das nur ein angenehmer Nebeneffekt. Die hier aufgeführten IP-Adressen bezeichnen Rechner, mit denen wirklich eine Netzwerksitzung aufgebaut werden kann. Vielen Programmen ist das egal, da geht es auch ohne einen Eintrag. `lpr` ist jedoch bekannt dafür, daß er ohne Einträge gelegendlich eigenartige Dinge tut. Und wenn die Datei schon editiert wird, lohnt es sich auch, dem Drucker einen Namen zu geben; das hat den Vorteil, daß, so der Druckerserver wo anders hin wechselt, nicht alle Einträge in der `printcap`, sondern nur die in `hosts` geändert werden müssen. Eine Änderung von einem Wert gegenüber der Änderung von drei oder noch mehr Werten macht sich, so das ganze über ein Skript und auf mehr als einem Rechner laufen soll, durchaus bemerkbar!

7.3 Einrichten des Druckers

Wir gehen bei den folgenden Beispielen von unserem Schuldrucker aus; das ist ein PostScriptfähiger Netzwerkdrucker, sprich: er hat eine eigene Netzwerkkarte. Außerdem kann er PostScript.

Diese Konstellation zwingt uns zu folgender Konfiguration: Alle Daten bis auf PostScript-Dateien müssen in PostScript übersetzt werden, sie müssen an den Drucker gesendet werden und die Filterung muß auf dem eigenen Rechner geschehen — der HP selbst kann das nicht.

Zwei Möglichkeiten gibt es, das zu tun: Entweder „Quick & Dirty“ mit etwas Handarbeit (funktioniert tadelloß) oder sauber, etwas langwieriger aber dafür übersichtlicher über einen sogenannten *Bypass*.

Der Drucker wird in diesem Beispiel über den Namen `schreiber` angesprochen. Eine entsprechende Auflösung findet sich in der Datei `/etc/hosts`:

```
10.90.131.200      schreiber
```

7.3.1 Handarbeit

Auch für diese Methode wird das Programm `lprsetup` benutzt. Nach dem Start erscheint ein Menü. In diesem muß der Punkt *ENTRY — Add/Overwrite/Delete new Aspifilter* angewählt werden. Im nächsten Menü wird der Punkt *Device* angewählt. Daraufhin erscheint ein Menü, in dem eine grundlegende Druckerart angewählt werden muß:

PARALLEL Jede Art von Parallel und USB-Druckern.

SERIAL Serielle Drucker — viele gibt es davon nicht mehr.

PREFILTER Das ist die Bypasslösung für die andere Methode.

REMOTE Einrichten eines Remote-Druckers.

In unserem Fall muß der erste Punkt, **PARALLEL**, angewählt werden. Danach folgt eine Eingabemaske, bei der die Gerätedatei (meistens `/dev/lp0`) angegeben werden soll. Das ist in unserem Falle zwar egal, aber irgend etwas sollte hier eingetragen werden. Sonst mault das Programm.

Es folgt ein Menü mit den folgenden Punkten:

RETURN Zurück zum letzten Menü.

DEVICE Ändern der Gerätedatei.

PRINTER Anwahl des Druckertreibers. Viele verschiedene gibt es nicht, HP-DeskJet wird unterstützt, ebenso PostScript.

PAPER Papiergröße, in Deutshcland meistens A4.

COLOR Monochorm oder Farbe, bei den meisten für Privatpersonen erschwinglichen Laserdruckern wird das wohl nicht Farbe sein.

SPECIAL Sondereinstellungen.

RESET Zurücksetzen der Einstellungen.

ADD Undbedingt wichtiger Abschluß, fügt den Drucker hinzu. Funktioniert nur, falls es sich nicht um eine Nachkonfiguration handelt. In diesem Falle wird **OVERWRITE** genutzt.

OVERWRITE Für Nachbesserung; überschreiben der alten Konfiguration mit den neuen Werten.

DELETE Löschen des Eintrages.

Die Punkte **PRINTER** bis **COLOR** *müssen* abgegangen und eingerichtet werden. Unter **PRINTER** muß in unserem Falle ein nativer PostScript-Drucker angegeben werden. Der Rest ist selbsterklärend. Nach Abschluß der Konfiguration ***muß*** **ADD** gewählt und das Programm über die vorgesehenen Methoden beendet werden.

Nach diesem Vorgang muß die Datei `/etc/printcap` von Hand verändert werden. Hier findet sich unter anderem drei neu angelegte Einträge. Wir betrachten nur den ersten, da die hier angeführten Änderungen auch für die anderen gelten.

```
hp|lp1|PS_hp-a4-ascii-mono-600|PS_ho2 a4 ascii mono 600:\
        :lp=/dev/lp0:\
        :sd=/var/spool/lpd/PS_ho2-a4-ascii-mono-600:\
        :lf=/var/spool/lpd/PS_ho2-a4-ascii-mono-600/log:\
        :af=/var/spool/lpd/PS_ho2-a4-ascii-mono-600/acct:\
        :if=/var/lib/apsfilter/bin/PS_ho2-a4-ascii-mono-600:\
        :la@:mx#0:\
        :tr=:cl:sh:
```

Dieser Eintrag ist eigentlich nur eine einzige Zeile. Das ist jedoch arg unübersichtlich, daher wurden Zeilenumbrüche eingefügt und dem Rechner klargemacht, daß er diese nicht zu beachten hat. Das geschieht durch das abschlïesende \, getrennt werden die einzelnen Einträge durch den **:**, die Einträge haben die folgende Bedeutung:

lp= Hier wird der **l**ine**p**rinter angegeben, also die Gerätedatei. Zur Zeit ist das die erste parallele Schnittstelle.

sd= Das Spooling-Directory, das Berzeichnis, in denen die gefilterten Dateien abgelegt werden.

lf= Das Verzeichnis der Logdateien.

af= Accounting-Files, eine Art Abrechnungsdatei, nicht weiter wichtig.

if= Der Input-APS-Filter.

Der Rest Keine Ahnung... `man printcap` ist da sehr aufschlußreich. Richtig wichtig sind die Einträge jedenfalls nicht.

Da wir keine Gerätedatei benutzen, können wir diesen Teil auskommentieren oder löschen. Statt dessen brauchen wir die Angabe einer Remote Maschiene und des Portes, den der Druckerserver zum Ausdrucken benutzt. Letzterer kann in unserem Fall auch weggelassen werden.

```
:rm=schreiber:\
:rp=lp:\
```

`rm` ist die **R**emote **M**achine, `rp` der **R**emote **P**ort. Diese zwei Zeilen ersetzen in allen drei Druckereinträgen die `lp`-Zeile.

Nachdem alles abgespeichert wurde, muß der `lpd`-Dämon erst erxorziert und dann neu beschworen, mit anderen Worten, gestoppt und neu gestartet werden. Das geschieht über den Befehl `rclpd restart`.

7.3.2 Die Konfiguration über *Bypass*

Etwas komfortabler, aber langwieriger ist die vollständige Konfiguration über `lprsetup`. Die Vorgehensweise ähnelt der erstgenannten zunächst. Es wird jedoch kein Paraleler, sondern ein Remote-Drucker angelegt (mit **PARALLEL**. Unter diesem Punkt werden der Servername und der Druckerport abgefragt — wie bereits in 7.3.1 beschrieben. Weitere Dinge sollten in den nachfolgenden Punkten *nicht* konfiguriert werden; diese Konfigurationen würden Arbeitsschritte auf dem Druckerserver veranlassen. Der HP-Drucker ist dazu nicht in der Lage.

Als nächstes wird der Punkt **PREFILTER** angewählt. In einer danach erscheinenden Maske werden bereits eingetragene Remote-Drucker angezeigt (das sollte nur einer sein). Nach Anwahl des entsprechenden erscheint ein Konfigurationsmenü, welches dem Kapitel 7.3.1, S. 39 entsprechend abzuarbeiten ist. Danach und einem Neustart des `lpd`-Dämons (`rclpd restart`) sollte das alles funktionieren.

Auch wenn dieser Text kürzer und die Lösung sauberer (und meines Erachtens vorzuziehen) ist, geht die andere schneller. Zudem ist sie dem Verständnis des Druckerprinzipes zuträglich.

7.4 Drucken und Wartung

Was bringt der schönste Drucker, wenn er nciht genutzt werden kann? Generell wird ein Drucker über den Befehl `lpr` angesprochen. Die meisten Programme nutzen auch exakt den `lpr`-Befehl. Auch die restlichen Befehle zur Druckersteuerung beginnen mit `lp`. Hier eine Auflistung der wichtigsten:

`lpr -Plp1 Datei` Druckt die angegebene `Datei` auf die Druckerwarteschlange `lp1` (*kein* Leerzeichen zwischen `-P` und `lp1`!). Wird der Parameter `-P` weggelassen, wird automatisch die als `lp0` definierte Warteschlange angesprochen.

`lpq` Zeigt die aktuelle Warteschlange der zu druckenden Dateien an.

`lprm Jobnummer` Löscht die angegebene Jobnummer (über `lpq` zu beziehen) aus der Warteschlange.

`lpc` Dient der Wartung der Druckerwarteschlange. Es erscheint ein Eingabeprompt. Es gibt eine ganze Reihe von Wartungsmöglichkeiten (Aktivieren, Deaktivieren, Löschen und Ähnliches von Warteschlangen), die sich über das Kommando `help` abrufen lassen.

Abgesehen vom letzten Befehl ist allen gemein, daß sich die entsprechende Druckerwarteschlange über den Parameter `-P` ansprechen lassen (wie bei `lpr` angegeben). Für weitere Beschreibungen dienen die Manpages.

Kapitel 8

Archive und Kompressionsverfahren

Im Internet gibt es eine ganze Reihe von Linux-Materialien. Wie auch für DOS mit integrierter GUI[1] werden die entsprechenden Dinge zusammengefasst; Immerhin will ja niemand tausende von kleinen und großen Einzeldateien herunterladen.

Das in der UNIX und somit auch Linux-Welt nach wie vor verbreitetste Format ist der sogenannte ***tar**-Ball*. Er wird vor allem für Quellcode eingesetzt. Fertige Binaries werden häufig in Form von `rpm`-Pakete ausgeliefert. Diese können über den RedHat Packet Manager (`rpm`) sehr einfach nstaliert werden.

8.1 Kompressionsverfahren

Ganz egal, um welche Daten es sich handelt, wenn sie heruner geladen werden sollen, biete es sich an, sie vorher zu kompremieren. Dieses stellt nicht nur einen Platz-, sondern im Falle einer Modem- oder ISDN-Verbindung (ohne Flatrate oder gar Standleitung) auch einen realen Geldgewinn dar.

Die verbreitesten Formate unter Linux sind *GNUzip* und seid neuerem *BZip2*. Die hiermit gepackten Dateien haben die Endungen `.gz` bzw. `.bz2`. In DOS-Kreisen findet sich häufig *ZIP*, welches die Endung `.zip` aufweißt und neben der Kompression auch eine Archiv-Funktion besitzt. In anderen UNIX-Derivaten ist das etwas ältere LZW-Verfahren stark vertreten. Diese Dateien haben die Endung `.Z`.

Um eine `.gz`-Datei zu erstellen wird das Programm `gzip` ***Datei*** aufgerufen. Die ursprüngliche Datei ist danach weg. Um eine solche Datei zu dekompremieren, wird entweder `gzip -d` ***Datei*** oder `gunzip` ***Datei*** benutzt. Für `.bz2`-Dateien lauten die Programme `bzip2` ***Datei*** und `bzip2`

[1] Manche nennen das auch Windows...

`-d` ***Datei*** oder `bunzip2` ***Datei***. `.Z`-Dateien werden mit `compress` erstellt. Zum Auspacken kann `gunzip`, `gzip -d` oder `compress -d` bentzt werden. Für `.zip`-Dateien schließlich gibt es die Proramme `zip` ***Archivname Datei*** und `unzip` ***Archivname***.

8.2 Das Programm `tar`

Dieses Programm dient eigentlich dazu, Bandlaufgeräte (Streamer) anzusprechen. Es fasst Daten in Archive zusammen, ohne sie zu komprimieren. Bandlaufgeräte erfreuen sich auch heute noch gerade in Firmen großer Beliebtheit, und aufgrund seines Kommando-Charakters läßt es sich auch wunderbar in Skripten implementieren. Beispielsweise ist es möglich, alle 24 Stunden eine Sicherung bestimmter Verzeichnisse durchzuführen.

`tar` sichert die komplette Baumstruktur. Bei der Wiederherstellung wird selbiger komplett rekonstruiert; allerdings nicht bei der obersten Ebene beginnend, sondern dort, wo der `tar`-Befehl ausgeführt wird. Auf diese Eigenheit wird an späterem Ort noch eingegangen.

Nun werden die meisten Dinge nicht auf Magnetbändern ausgeliefert. Daher wurde schon recht früh ein Schalter implementiert, der anstatt auf ein Bandlaufwerk in eine Datei schreibt. Dadurch wird der Platzbedarf nicht weniger, doch die Verzeichnisstruktur bleibt erhalten. Zudem ist es, wie bereits gesagt, einfacher, eine Datei als viele Dateien herunter zu laden.

Um ein Archiv zu erstellen, wird mit dem Schalter `c` (*create*, erstelle) aufgerufen. Als weiterer Schalter ist `f` (*file*, Datei) nötig, da sonst versucht wird, das (auf den meisten Heimrechnern nicht vorhandene) Bandlaufwerk anzusprechen. Es folgt der Name des zukünftigen Archives, welches traditioneller Weise mit der Endung `.tar` ausgestattet werden sollte. Der Befehl
`tar -cf bastard.tar /home/dau`
würde an der Stelle, an der es aufgerufen würde, ein Archiv `bastard.tar` erstellen, in dem sich der Inhalt des Ordners `/home/dau` befindet.

Angenommen, dieses Archiv wurde im Verzeichnis `/root` erstellt. Dann kann es dort mit dem Schalter `x` (*extract*, entpacke) in der Form
`tar -xf bastard.tar`
wieder entpackt werden. Allerdings wird es dann nicht in `/home/dau` entpackt. Statt dessen gibt es dann allerdings einen Ordner `/root/home/dau`, indem sich die Daten finden.

Wäre `tar` allerdings aus dem Verzeichnis `/` in der Form
`tar -xf /root/bastard.tar`
aufgerufen worden, wäre es tatsächlich am richtigen Ort entpackt worden.

Natürlich können Archive mit entsprechenden Programmen komprimiert werden. Inzwischen wurden sogar Schalter implementiert, welche die Kompression direkt mit dem `tar`-Befehl kombinieren. Das ist auch bei der ursprünglichen Anwendung, also dem Schreiben von Daten auf Band, möglich.

Zwar wird hierdurch Platz eingesparrt, doch auf der anderen Seite wird die Datensicherheit dadurch gesänkt; wärend bei unkompremierten Schreiben im Falle von Bandfehlern lediglich die betroffenen Dateien nicht zurückerstellt werden können, fällt im Falle eine Kompression das gesammte Archiv aus. Dieses manko ist allerdings bei der Aufbereitun von Daten zum Verschicken oder Herunterladen zu vernachlässigen.

Die verbreitesten Formate sind *GNUzip* und seid neuerem *BZip2*. Da UNIX keine Probleme mit Dateinahmen, die mehr als einen Punkt enthalten, hat, enden die gepackten Archive meistens auf `.tar.gz` oder `.tar.bz2`. Allerdings wird an manchen Stellen auch DOS Rechnung getragen, `tar.gz` verschmilzt dann manchmal zu `.tgz`, was aber praktisch gesehen keinen Unterschied macht.

Diese Dateien lassen sich entweder manuell entpacken (siehe 8.1, S. 43) und dann wie gehabt mit Hilfe von `tar` auspacken, oder über direkte Angabe eines entsprechenden Schalters, nämlich `z` für *GNUzip* und `j` für *BZip2*.

Noch ein Wort zum `-` vor den Schaltern: Unter Linux kann es wohl weggelassen werden. Aber das klappt nicht unter allen UNIX-Systemen.

Hier nochmal die wichtigsten Schalter im Überblick:

`c` Create oder Erstelle, mit dieser Anweisung wird `tar` dazu bewegt, ein Archiv zu erstellen.

`x` Extrahiere, hiermit werden bestehende Archive entpackt.

`t` Table oder Liste, zeigt den Inhalt eines Archives an.

Die folgenden Schalter können oder müssen für alle der vorhergegangenen angegeben werden.

`f` File oder Datei, das vermutlich wichtigste, da es `tar` anweist, nicht auf ein Bandlaufwerk, sondern eine Datei zuzugreifen. Dieser Schalter ist *immer* erforderlich, falls nicht auf ein Bandlaufwerk zugegriffen werden soll.

`z` Dieser Schalter steht für die Kompression mit `gzip`. Er muß nicht nur im Zusammenhang mit `c` zur Erzeugung eines komprimierten Archives, sondern auch bei jedem Zugriff mittels `x` oder `t` auf ein komprimiertes Archiv zugegriffen werden soll.

`j` Dieser Schalter wird genauso behandelt wie `z`, allerdings gilt es für die Kompression mittels `bzip2`.

`v` Verbous oder Verbal-Modus, der Schalter sorgt dafür, daß `tar` auf dem Bildschirm ausgibt, was es gerade tut.

8.3 Der `rpm`

Der RedHat Packet Manager (RPM), der über das Programm `rpm` aufgeführt wird, wurde letzten Endes aus `tar` entwickelt. Neben kleinen Annehmlichkeiten wie abfragbaren Informationen zu den Archiven (oder *Paketen*) und festen Pfaden wurde insbesondere eine Datenbank hinzugefügt. In dieser finden sich alle installierten Programme mit ihren Informationen. Dies macht es recht einfach (für `rpm`), bei der Installlation eines neuen Paketes zu überprüfen, ob es bereits vorhanden ist oder andere Programme zu Konflikten führen werden. Zudem ist dies für die Abhängigkeiten nützlich. Die Endung eines `rpm`-Paketes lautet `.rpm`.

Der `rpm` ist extrem mächtig; er funktioniert über `ftp` (es sind also Installationen über das Netz von Internet stehenden FTP-Servern möglich), er kann zur Installation Listen abarbeiten, und natürlich kann er wunderbar in irgend welche Skripte integriert werden. Hier werden nur die wichtigsten der vielen Kommandos, die es zu ihm gibt, aufgeführt:

`-i` Installiert die nachgestellten Packet.

`-e` Löscht (erase) die nachgestellten Packet.

`-q` Query, Anfrage. Hierzu müssen weiterreichende Dinge angegeben werden. Mittels `-p` wrd das Paket, zu dem die Anfage st, speifiziert, `-i` fragt die Informationen ab, `-a` gibt alle installierten Pakete aus.

Zur Vedeutlichung folgen hier ein paar Beispiele:

Um das Programm `xjewel` zu installieren, muß zunächst eine Quelle dazu bekannt sein. Dieses Paket befindet sich im Verzeichnis `/suse/fun2` auf der vierten CD-ROM. Nach dem mounten (in unserem Beispiel auf das Verzeichnis `/mnt/cdrom` führt der Befehl:
`rpm -q -i -p /mnt/cdrom/suse/fun2/xjewel.rpm`
zu einer Ausgabe aller Informationen, die mitgeliefert wurden. Der Output sollte etwa so aussehen wie in Abbildung 8.1 auf Seite 47 gezeigt.

Mit dem Befehl

`rpm -i /mnt/cdrom/suse/fun2/xjewel.rpm`

wird das Paket installiert. Die oben genannte Anfrage (`rpm -q -i -p /mnt/cdrom/suse/fun2/xjewel.rpm`) würde dabei immernoch ergeben, es nicht installiert. Eine Anfrage `rpm -q -i xjewel` (ohne `-p`, denn hier wird ja keine Packetdatei angefragt) ergibt, daß das Packet installiert ist.

Gelöscht werden kann es mit dem Befehl

`rpm -e xjewel`

aber wer will so ein tolles Programm schon löschen?

```
Name        : xjewel                     Relocations: (not relocateable)
Version     : 1.6                        Vendor: SuSE GmbH, Nuernberg, Germany
Release     : 318                        Build Date: Wed May 16 00:10:34 2001
Install date: (not installed)            Build Host: Hygiea.suse.de
Group       : X11/Games                  Source RPM: xjewel-1.6-318.src.rpm
Size        : 76906                      License: 1990 Yoshihiro Satoh
Packager    : feedback@suse.de
Summary     : X11 game - similar to tetris
Description :
X11 game - similar to tetris. Has a flavour of its own :-)
Doku: /usr/share/doc/packages/xjewel/README
```

Abbildung 8.1: `rpm`-Informationen über das Paket `xjewel`

Kapitel 9

Kernel- und Modulverwaltung

Der Kernel ist der Kern des Betriebsystems. In ihm befinden sich auch die Gerätetreiber (oder zumindest Möglichkeiten, Treiber zu laden). Falls ein Kernel bestimmte Geräte nicht unterstützt, ein neuerer das aber tut, oder der Benutzer besonders exotische Wünsche hat (etwa *„Ich will unbedingt auf ein OS/2-Verzeichnis zugreifen, also, Admin, mach mal!“*), muß der Kernel neu kompiliert werden.

Grundsätzlich gibt es zwei Philosophien zu Linux-Kerneln: Micro-Kernel und Macro-Kernel. Der Macro-Kernel (von einigen auch als *Monolithischer Kernel* bezeichnet) enthällt alle Treiber, die er benötigt. Diese Kernel sind naturgemäß ziemlich groß. Sie haben den Vorteil, nicht auf andere Dateien angewiesen zu sein. Der UNIX-Entwickler TANNENBAUM ist Vertreter dieser Philosophie. LINUS THORWALD ist Vertreter des Micro-Kernels; Der beinhaltet nach Möglichkeit nur daß, was er unbedingt zum booten braucht, und läd das andere als Module nach.

9.1 Voraussetzungen

Um einen Kernel neu kompilieren zu können, werden zuerst die Quellen dazu gebraucht. Eine gute Adresse dafür ist `www.kernel.org`, hier finden sich regelmäßig die neusten. Es empfielt sich, Kernel der Verion `2.4.x` zu laden; die `2.5.x` sind Entwicklerkernel und nicht unbedingt stabil.

Die meisten Kernelquellen kommen in sogenannten `tar`-Balls. Diese sollten in das Verzeichnis `/usr/src` kopiert und entpackt werden (siehe Kapitel 8.2, S. 44). Danach sollte sich innerhalb des `src`-Verzeichnisses ein Verzeichnis mit dem Namen `linux`. Es empfielt sich, dieses in `linux-`*`Kernel-version`* (also zum Beispiel `linux-2.4.4`) umzubenennen und einen Link namens `linux` auf dieses Verzeichnis zu erstellen. **ACHTUNG:** Falls es solch einen Link bereits gibt, muß der **vor dem Entpacken** des Verzeichnisses gelöscht werden. Sonst wird der Inhalt des Verzeichnisses, auf das `linux` zeigt, überschrieben!

Neben den Quellen wird außerdem der GNU-C-Compiller `gcc` und das Programm `make` benötigt. Bei letzterem handelt es sich um ein extrem mächtiges Programm, welches anhand einer Datei `Makefile` heruasfindet, in welchen Konfigurationen es welche Programme ausführen soll. Diese Programm regelt die Kompilation und die Konfiguration.

9.2 Konfiguration

Sobald die Quellen da und entpackt sind, geht es an die Konfiguration. Dies geschieht über den Befehl `make` *`target`*. Der *`target`* bestimmt in diesem Fall, was genau geschehen soll. Unter anderem kann zwischen verschiedenen Konfigurationsmethoden unterschieden werden:

`make config`: Extrem spartanische, textbasierte Konfiguration, vergleichbar mit `xf86config` (siehe Kapitel 6.3.1, S. 32).

`make menuconfig` Eigentlich der Standard; eine Menübasierte Konfiguration, die im Textmodus läuft.

`make xmenuconfig` Eine menügesteuerte, grafische Konfiguration (mit anderen Worten: Die kann nur unter X ausgeführt werden).

Nun können verschiedene Bereiche konfiguriert werden, etwa zu unterstüzende Dateisysteme, Geräte und ähnliches. Der Raum reicht hier nicht aus, um jeden einzelnen Punkt anzusprechen. Doch einige grundlegende Dinge sollten auf jeden Fall angesprochen werden.

Treiber können entweder als feste Bestandteile des Kernels oder als nachladbare Module kompiliert werden. Falls Module gewählt werden, muß unter dem Punkt *Loadable Module Support* eingestellt werden, daß der Kernel das abkann. Module haben den Vorteil, daß sie nicht aufwendig nachgeladen werden müssen. Allerdings können nicht alle Treiber als Module strukturiert werden; die Treiber, die das System zum starten benötig (etwa der ATA oder der SCSI-Treiber für den Controller und die Platten und natürlich das Dateisystem, auf dem das alles liegt, meistens ext2), *müssen* im Kernel vorhanden sein. Zudem gibt es einige, zu denen die Entwickler keine Modul-Konfiguration implementiert haben.

9.3 Kompilieren

Bevor kompilliert werden soll, bietet es sich meist an, alte Abhängigkeiten von vorhergegangenen Konfigurationen aufzuheben und die aktuellen zu erstellen. Das Aufheben geschieht durch `make clean`, das Neuerstellen der Abhängigkeiten durch `make dep`. Nachdem diese Voraussetzungen erfüllt sind, kann der Kernel auf verschiedene Weisen kompiliert werden. Besprochen wird hier nur eine: `bzImage` Hierzu wird ein Image erstellt, daß mit dem bz2-Algorythmus komprimiert wird. Dieses wird im Verzeichnis `/usr/src/linux/arch/i386/boot` unter dem Namen `bzImage`.

Falls Treiber als Module konfiguriert wurden, müssen die gesondert kompilliert werden. Zu diesem Zweck wird der Befehl `make modules` eingegeben. Durch den anschließenden Befehl `make modules_install` werden sie an die Stelle geschrieben, an der der Kernel sie erwartet, nämlich das Verzeichnis `/lib/modules/`*`Kernelversion`* (also zum Beispiel `/lib/modules/2.4.4`).

9.4 Installieren

So die Konfiguration und das Kompilieren wie oben geschrieben vorgenommen wurde, muß nun der Kernel noch an eine Stelle gelegt werden, von der er gebootet werden kann. Meistens wird dazu im Verzeichnis `/boot` (siehe Kapitel 5.1, S. 25) die vorhandene Datei `vmlinuz` in `vmlinuz.old` umbenannt und die Datei `/usr/src/linux/arch/i386/boot/bzImage` nach `/boot/vmlinuz` kopiert oder verschoben.

Im Sinne der Sicherheit bietet es sich an, zusätzlich mit dem `lilo` eine weitere Boot-Option mit dem alten Kernel (also `/boot/vmlinuz-old`) zu erstellen (näheres dazu siehe Kapitel 5.2, S. 26).

Diese Schritte werden automatisch erledigt, falls zum Erstellen des Kernels `bzlilo` angegeben wurde. Hierzu muß jedoch zuvor im `Makefile` das Installationsverzeichnis freigeschaltet werden, indem in der Zeile
`#export INSTALL_PATH=/boot`
das Kommentarzeichen (`#`) herausgenommen wird.

9.5 Modul-Verwaltung

Ein Modul ist ein Treiber, der nicht in den Kernel einkompiliert wurde. Für gewöhnlich kümmert sich der Kernel selbst darum, die Module zu laden, die er benötigt. Trotzdem kann es von Zeit zu Zeit sinnvoll sein, sich anzusehen, was er da eigentlich treibt, und gelegendlich mag es auch vorkommen, daß ein Modul von Hand nachgeladen werden muß. Grundsätzlich gibt es vier Befehle, mit denen Solchartiges durchgeführt werden kann:

`lsmod` Dieser Befehl listet alle geladenen Module auf.

`insmod` Mit diesem Befehl können neue Module hinzugefügt werden.

`modprobe` Bewirkt das selbe wie `insmod`. Allerdings wird vorher überprüft, ob alle Abhängigkeiten erfüllt wurden. Falls nicht, werden entsprechend benötigte Module vorher geladen.

`rmmod` Mit diesem Befehl werden Module entfernt.

Kapitel 10

Disk-Quotas — Platzkontingente

Häufig ist es sinnvoll, Benutzern in Hinsicht auf ihren Speicherplatz Grenzen zu setzen. Immerhin wäre es ziemlich störend, wenn die Buchhaltung nicht mehr arbeiten könnte, weil einer der Entwickler sich denn Quelltext von KDE mit allen Dokumentationen heruntergeladen hat. Dafür sieht UNIX das Prinzip der Disk-Quotas vor.

Quotas werden für Partitionen angelegt. In diesem Sinne erscheint es sinnvoll, das Verzeichnis `/home` auf eine Eigene zu legen. Innerhalb dieser Partition können sowohl einzelne Benutzer als auch ganze Gruppen mit Einschränkungen versehen werden.

Die Limitation gliedert sich in ein *Soft-* und ein *Hard Limit*. Wird ersteres erreicht, bekommt der Benutzer eine Meldung. Wenn er nicht innerhalb einer als *Grace* (Gnade) bezeichneten Frist seinen Platzbedarf unter die Grenze bekommt, wird er behandelt, als habe er den *Hard Limit* überschritten. Falls das geschieht, kann er keine Dateien mehr schreiben.

Diese Zeit wird nicht für die Benutzer, sondern für die Partition eingestellt. Das bedeutet, daß für alle Benutzer die gleiche Zeit gilt, so sie auf die gleiche Partition zugreifen.

10.1 Umstellen des `/home`-Verzeichnisses

Für den Fall, daß das `/home` Verzeichnis nicht auf einer eigenen Partition liegt, aber noch genügend Platz vorhanden ist, um eine zu erstellen, kann wie folgend vorgegangen werden:

1. Der Inhalt des Verzeichnisses sollte in das Verzeichnis `/root` gesichert werden. Das geschieht am besten, wenn aus dem Verzeichnis `/home` heraus der Befehl `tar -cf /root/home.tar *` aufgerufen wird.

2. Der Inhalt des `/home`-Verzeichnisses muß gelöscht werden (etwa mit `rm -rf /home`).

3. Die neue Partition muß erstellt werden. Dies geschieht über den Befehl `fdisk /dev/hd`*x*, wobei *x* die für die Platte stehende Gerätedatei ist (also meistens `a`... Falls es sich bei dem System um ein SCSI-System handelt, lautet der Name der Gerätedatei übrigens anders, nämlich `/dev/sd`*x*).

4. Mit `m` können die verschiedenen Befehle des `fdisk` angezeigt werden. In diesem Fall ist der benötigte `n` (wie *new*, neu). Hier muß nun eine Partition erstellt werden; neben dem Startzylinder (es sollte der Angegebene verwendet werden) wird auch der Endzylinder benötigt. Inziwschen ist es auch möglich, diesen durch `+Größe`, als zum Beispiel `+100M`, bestimmen zu lassen. Im Laufe dieses Vorganges sollte gemeldet werden, wie diese Partition heißt. Der Namen dieser Gerätedatei ist wichtig.

5. Das Programm `fdisk` muß mit `w` beendet und danach ein Neustart durchgeführt werden.

6. Nach dem Neustart wird das Dateisystem mittels `mkfs -t ext2 /dev/hd`*xy*, wobei `/dev/hd`*xy* die Gerätedatei aus 4 ist. (Alternativ kann auch `mke2fs /dev/hd`*xy* benutzt werden.)

7. Der `tar`-Ball aus dem Verzeichnis `/root` kann nun wieder im Verzeichnis `/home` entpackt werden.

Sobald das geschehen ist, kann mit der gewöhnlichen Prozedur zum Einstellen von Quotas wie im Folgenden beschrieben fortgefahren werden.

10.2 Voraussetzungen

Um Quotas nutzen zu können, müssen folgende Voraussetzungen erfüllt werden: Der Kernel muß Quotas in irgend einer Weise unterstützen (der entsprechenden Punkt findet sich unter *File systems*). Des weiteren muß das Paket `quota.rpm` installiert werden. Es befindet sich auf der zweiten SuSE-CD und kann zum Beispiel mit `rpm` installiert werden (siehe Kapitel 8.3, S. 46).

Nachdem die Installation der genannten Sachen abgeschlossen ist, müssen im Verzeichnis `/home` zwei Dateien erstellt werden (etwa mit `touch`), nämlich `quota.user` und `quota.group`. Danach sollten die Rechte der Dateien mit `chmod 600 quota.*` gesetzt werden. *Auf einigen Systemen läuft eine neuere Version von Quota; diese erwartet als Benennung der Dateien* ***`aquota.user`*** *und* ***`aquota.group`***.

10.3 Einrichtung

Um die Quotas tatsächlich einzurichten, muß die entsprechende Partition in die Datei `/etc/fstab` eingetragen werden. Dabei ist es wichtig, daß bei den Dateioptionen das Wort `quotausr` erscheint. Die entsprechende Zeile könnte etwa so aussehen:

```
/dev/hda8    /home         ext2        defaults,usrquota   1   2
```

Auf die genaue Bedeutung dieser Zeile wird an dieser Stelle nicht eingegangen. Entscheident ist, daß `/dev/hda8` bei Systemstart auf `/home` gemountet wird und Quotas benutzt.

Nach diesem Eintrag muß `/home` mit dem Befehl `mount -o remount /home` neu gemountet werden. Erst danach kommt der Quota-Eintrag zum Tragen. Danach kann die Initialisierung der Begrenzungen durch den Befehl `quotacheck /home` erfolgen. Aktiviert wird der Mechanismus durch den Befehl `quotaon -a` oder einen Neustart.

10.4 Vergeben von Quotas

Die Vergabe von Quotas geschieht über den Befehl `edquota` (*edit quota*, editiere die Begrenzung). Mit diesem Befehl können einzelnen Benutzern (das ist der Standard oder der Schalter `u`) oder Gruppen (über den Schalter `g` bestimmte Kontingente zugewiesen werden. Der Aufruf öffnet einen `vi`-ähnlichen Editor; hier finden sich alle relevanten Werte. Wichtig sind dabei vor allem die beiden Einträge unter den ersten `soft` und `hard`, da hier die beiden Grenzen (die Warngrenze und die Sperrgrenze) angegeben werden.

Besonderer Augenmerk ist auf den Schalter `t` zu legen: Dieser bezieht sich nicht auf Benutzer oder Gruppen, sondern auf Partitionen. Hier kann die `Grace Periode`, die *Gnadenzeit*, eingestellt werden.

Kapitel 11

Der LVM

Die Abkürzung **LVM** steht für ***L**ogical **V**olumen **M**anager*. Die ursprüngliche Idee dabei war, verschiedene Partitionen und Platten zu einer logischen Platte zusammen zu fassen. Die auf Seite 58 stehende Abbildung 11.1 verdeutlicht ein Beispiel, indem ein solches Vorgehen durchaus Vorteile hat. In diesem Beispiel sind physikalisch drei Festplatten vorhanden, die jeweils in vier (primäre) Partitionen unterteilt wurden. Aus irgend einem Grund wurden die markierten Partitionen frei. Das trifft sich ausgesprochen gut, denn das Verzeichnis `/home` dieses Rechners ist zu klein. Eine Möglichkeit bestände darin, mehrere Verzeichnisse `/home1` bis `/home3` einzurichten. Für Skripte und Verwaltungsmeldungen ist das aber ziemlich nervig. Davon abgesehen wäre vielleicht ein kleiner Teil des neuen Platzes auch ganz nützlich für ein anderes Verzeichnis. An dieser Stelle greift der LVM.

11.1 Funktionen und Möglichkeiten

Mit dem LVM können mehrere Partitionen zu einer logischen Platte zusammengefaßt und diese dann partitioniert werden. Es hat sich eingebürgert, bei der logischen Platte von einer *virtuellen Gruppe*, bei den darauf eingerichteten Parttionen von *logischen Volumes* zu sprechen.[1]

Einer bestehenden virtuellen Gruppe können weitere (physische) Partitionen hinzugefügt werden. Das mag nützlich sein, wenn irgend wo noch eine Platte anfällt,die in das System integriert werden, aber keinen eigenen Mountpoint bekommen soll.

Ein logisches Volume kann, so in der virtuellen Gruppe noch freier Platz vorhanden ist, um diesen erweitert werden. (Mit anderen Worten: Falls das `/home`-Verzeichnis aus unserem Beispiel schon wieder zu klein wird, kann die virtuelle Gruppe zunächst um ene weitere (physische) Partition ergänzt und

[1] In wie weit das tatsächlich richtig oder glücklich ist, sei einmal dahingestellt; im Sinne der allgemeinen Gültigkeit wird diese Bezeichnung hier beibehalten.

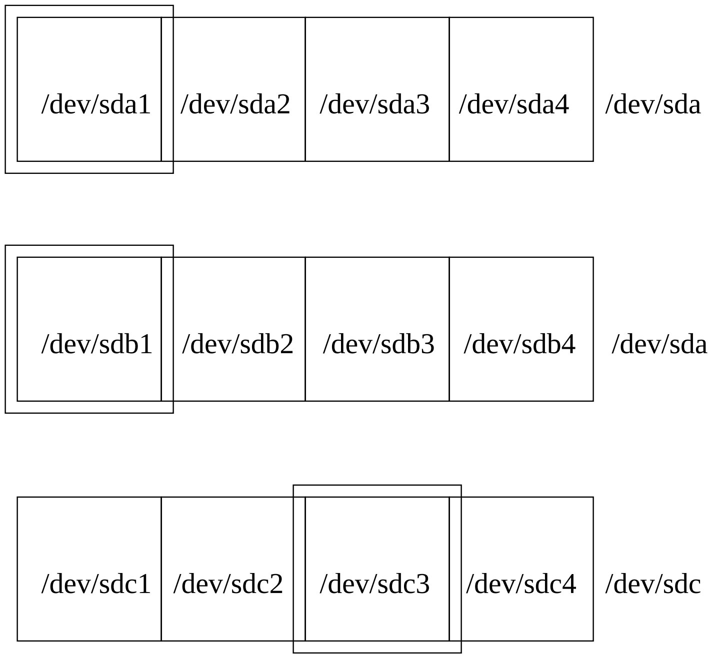

Abbildung 11.1: Drei SCSI-Platten

dieser Platz dem auf das Verzeichnis `/home` gemountete logischen Volume hinzugefügt werden.)

Seid neuerem kann LVM auch für Striping engesetzt werden.[2] Darauf wird hier jedoch nicht weiter eingegangen.

11.2 Voraussetzungen

Um den LVM überhaupt nutzen zu können, muß er vom Kernel unterstützt werden.[3] Er findet sich unter dem Menüpunkt *Multi-Device Support (RAID and LVM)*, und dort unter dem Unterpunkt *Logical volumen manager (LVM) support*. Wie die meisten Treiber kann auch dieser als Modul kompiliert werden. Die Erfahrung hat jedoch gezeigt, daß er fest in den Kernel integriert weniger Probleme bereitet.

Neben der Unterstützung durch den Kernel muß natürlich entsprechend freier Platz vorhanden sein.Außerdem ist für jedes später nutzbare logische Volume ein Mountpoint nötig. Soll das `/home`-Verzeichnis in ein logisches Volume umgewandelt werden, bietet sich das bereits für Quotas beschriebene Verfahren (siehe Kapitel 10.1, S. 53) zum Umstellen der Daten an.

---

[2] Zumindest war es neu, als diese Unterichtsschrift erstellt wurde...

[3] Der 2.4.4-GB Kernel aus der SuSE 7.2-Distribution tut das anscheinend; es ist davon auszugehen, daß auch spätere SuSE-kernel das tun.

11.3 Einrichten einer virtuellen Gruppe

Um eine virtuelle Gruppe erstellen zu können, müssen die entsprechenden Partitionen zunächst einmal vorhanden sein. In dem Beispiel der Platten in Abbildung 11.1 auf Seite 58 ist das bereits der Fall, anderenfalls können sie auch mit `fdisk` neu erstellt werden (siehe ebenfalls Kapitel 10.1, S. 54). Bei späteren Verweisen auf neu-erstellte Partitionen wird davon ausgegangen, daß sich diese über die Gerätedateien `/dev/hda8` bis `/dev/hda10`[4] ansprechen lassen.

In beiden Fällen müssen die Kennungen der Partitionen geändert werden. Dies geschieht unter `fdisk` mit dem Befehl `t`. Der angestrebte Typ *Linux LVM* hat, wie eine Auflistung aller verfügbaren Typen mit dem `fdisk`-Befehl `l` ergibt, die ID `8e`. Nachdem das eingegeben wurde, ist `fdisk` mit dem Kommando `w` zu beenden.

Um die betroffenen Partitionen später in eine virtuelle Gruppe aufnehmen zu können, müssen sie erst als sogenannte *physical volumes*, PVs, initialisiert werden. Dies geschieht in unserem Beispiel mit dem Befehl
`pvcreate /dev/sda1 /dev/sdb2 /dev/sdc3`
oder aber, im Falle der neu erstellten, mit
`pvcreate /dev/hda[8-10]`.[5]

Sobald diese Schritte abgeschlossen sind, wird die virtuelle Gruppe mit dem Befehl
`vgcreate virtuell /dev/sda1 /dev/sdb1 /dev/sc3`
oder im Falle der neu-erzeugten mit
`vgcreate virtuell /dev/hda[8-10]`
erstellt. Dabei ist `virtuell` der Name der Platte, hier könte auch jede andere zeichenfolge stehen. Nach diesem letzten Schritt st die Plate vorhanden und Einsatzbereit; ihre Daten lassen sich mit dem Befehl
`vgdisplay -v virtuell`
anzeigen.

11.4 Erstellen logischer Volumes

Die Platte `virtuell` ist nun also vorhanden, sie kann nun wie eine normale Festplatte partitioniert werden. Dies egschieht jedoh nicht mit dem Befehl `fdisk`, sondern mit `lvcreate`. Zu diesem Befehl müssen der Parameter `-L` mit der Größe des logischen Volumes und der Name der zugehörigen virtuellen Gruppe angegeben werden. Für ein 250 MByte großes Volume namens `lv1` lautete der Befehl also:
`lvcreate -L 250M -n lv1 virtuell`

[4]also Drei logische Partitionen auf einer IDE-Platte.

[5]Diese Kurzschreibweise sollte geläufig sein... Für das Beispiel ließe sich auch `pvcreate /dev/sd[ab]1 /dev/sdc3` schreiben.

Angesprochen werden diese logischen Volumes über `/dev/`*`Gruppenname`*`/`*`Volumename`*, in unserem Beispiel also `/dev/virtuell/lv1`.

Bevor ein Volume benutzt werden kann, muß es — wie jede andere Partition auch — formatiert werden. In unserem Beispiel bietet sich der Befehl `mkfs -t ext2 /dev/virtuell/lv1` (siehe auch Kapitel 10.1, S. 54) an. Ebenso wie in diesem Fall werden die Partitionen später (also beim mounten oder ähnlichen Aktionen) über ihre Gerätedateien wie ganz gewöhnlche Partitionen angesprochen.

Zur Zeit der Niederschrift dieses Kapitels war es noch nicht möglich, mit einem anderen Dateisystem als ext2 zu arbeiten. Möglicher Weise hat sich das inzwischen geändert.

11.5 Manipulieren logischer Volumes

Logische Volumes können sowohl im laufenden Betrieb als auch nachdem sie ausgekoppelt wurden in ihrer größe verändert werden. Zur Laufzeit ist das eine ausgesprochen kitzelige Angelegenhei, zudem werden die entsprechenden Programme nicht von allen Distributoren mitgeliefert. Daher wird hier nur auf die zweite, sichere Variante eingegangen.Ebenfalls außer Acht gelassen wird die Möglichkeit der Laufwerksverkleinerung; das scheint nach wie vor nicht zu funktionieren oder relativ weit jenseits der Trivialität zu liegen.[6]

Die Vergrößerung eines logischen Volumens geschieht über den Befehl `e2fsadm`.[7]

Bevor dieser Befehl angewandt werden kann, sollte die entsprechende Partition mit `umount` ausgekopet werden. Danach kann sie unter Angabe ihrer Gerätedatei vergrößert werden. Auch hier lautet der Parameter für die zunehmende Größe `-L`. Ein Befehl, um das vorhin erstellte logischeVolume um 50 MByte zu vergrößernlautet also:
`e2fsadm /dev/virtuell/lv1 -L+50M`

11.6 Erweiterung virtueller Gruppen

Wie bereits angekündigt können virtuelle Gruppen belebig erweitert werden. Eine Partiton, die als Erweiterung einer solchen Gruppe dienen soll, muß zunächst als physikalisches Volume (siehe 11.3, S. 59) eingerichtet werden.

[6]Mit anderen Worten: Der Verfasser hat sich das Dateisystem des entsprechenden logischen Laufwerkes gründlich zerschossen und geht in seiner selbstherrlichen Art undWiese davon aus, daß es also nicht ganz einfach sein kann...

[7]Falls dieser Befehl nicht vorhanden sein sollte, gibt es immer noch die ihm zu Grunde liegenden Befehle `resize2fs` und `lvextend`, zu denenglücklicher Weise `man`-Pages existieren.

Sobald das geschehen ist, kann sie mit dem Befehl `vgextend` ***Gruppenname Gerätedatei*** der Gruppe hinzugefügt werden.

Angenommen, der Gruppe `virtuell` aus unserem Beispiel sollte die Partition `hdd9` hinzugefügt werden. Dann mste diese zunächst mit der Befehlszeile
`pvcreate /dev/hdd9`
als PV initialisiert werden. Danach wäre es möglich, sie mit dem Befehl
`vgextend virtuell /dev/hdd9`
anzugliedern. Es ist in diesem Fall vollkommen Egal, ob es sich bei den restlichen Platten um SCSI- oder IDE-Platten handelt, beide Arten können gemischt in einer virtuellen Gruppe vorkommen. Sobald es eine Möglicheit gibt, USB-Platten oder sonstge Datenträger von Linux aus als Gerätedatei anzusprechen, werden auch diese eingeschlossen werden können.

11.7 Ein Fallbeispiel

In diesem Beispiel werden die Vorteile eines LVM-Volumes nochmals aufgeführt und erklärt. Es kann in der „freien Natur“ durchaus vorkommen.

11.7.1 Die Grundlage

Eine große Firma verfügt über einen Fileserver, auf dem alle Mitarbeiter ihre Arbeiten speichern. Der Fileserver verfügt dafür über ein freigegebenes Verzeichnis `/export`. Dieses liegt auf dem logischen Volume `/dev/lvmp/exp`. Die virtuelle Gruppe `lvmp` beinhaltet nur dieses logische Volume und setzt sich zur Zeit aus den vier Partitionen `/dev/sdb1` bis `/dev/sde1` zusammen, wobei jede Partition jeweils die komplette SCSI-Platte umfaßt. Das Volume `/dev/lvmp/exp` hat damit eine Größe von 120 GByte.

11.7.2 Die Aufgabe

Das besagte Volumen ist voll. Aus desem Grund wurde eine weitere SCSI-Platte in Größe von 60 GByte angeschafft. Diese soll nun dem bereits vorhandenen `/export`-Verzeichnis angegliedert werden.

11.7.3 Vorgehen

Als erstes wird die Platte angeschlossen. Sie hat die Kennung `/dev/sdf`. Mittels `fdisk` wird sie als eine komplette Partition (`/dev/sdf1` eingerichtet. Selbige erhällt die Kennung `8e`.

Nachdem diese Partition mit dem Befehl `pvcreate /dev/sdf1` als physikalisches Volume initialisiert wurde, erfolgt das Angliedern an die Gruppe `lvmp` mittels `vgextend lvmp /dev/sdf1`.

Durch den Befehl `e2fsadm /dev/lvmp/exp -L+61440M`[8] schließlich wird das Verzeichnis `/export` (welches tunlichst vorher mit dem Befehl `umount /export` ausgekoppelt worden sein sollte) um die 60 GByte erweitert. Nachdem es wieder gemountet wurde, steht der zusätzliche Platz zur Verfügung.

[8] 1 GByte sind 1024 MByte, GByte wurde bei Niederschrift noch nicht als Einheit unterstützt

Anhang A

Workshop

Bei diesem Schriftstück handelt es sich um einen Workshop, um im Unterricht (hoffendlich) erlerntes Wissen zu testen und teilweise auch in die Praxis umzusetzen. Die vorgeschlagenen Übungen und Skripte können praktisch durchgeführt werden, es mag in manchen Fällen auch ausreichen, sie theoretisch zu behandeln.

Um dem Leser einen Eindruck davon zu vermtteln, wie arg gefrustet wir alle zu diesem Zeitpunkt waren und wie sehr wir uns damals gegenseitig auf die Nerven gegangen sind (wer das heute leugne, der schönt...), habe ich das damalige von mir verfaßte Vorwort beigefügt...

A.1 Vorwort zur Original-Ausgabe

Der vorliegende Workshop entstand auf Bitten von Thorben als Vorbereitung auf den LPI. Zudem fragten auch einige andere nach so etwas. Allerdings frage ich mich langsam ernsthaft, ob diese Arbeit überhaupt irgend welchen Sinn und Zweck hat; für gewöhnlich habe ich den Eindruck, daß viele meiner Mitmenschen entweder die ganze Zeit im Unterricht gefehlt haben oder die Zetel, die ich bislang verfasst habe, zwar begeistert eingepackt, aber nicht gelesen geschweige denn verstanden haben. Zumindest würde ich viele der teilweise gestellten Fragen so werten. (Vermutlich fühlen sich — wie üblich — nun genau die falschen Personen angesprochen, wohingegen diejenigen, die es betrifft, sich keines wegs angesprochen fühlen, aber irgend wie ist mir das ziemlich egal.)

Und weil wir gerade dabei sind, irgend welche nicht ersichtlich sinnvollen, mit dem Stoff in Zusamenhang stehende Dinge aufzulisten, an dieser Stelle noch ein Danke an Olaf L. für die freundliche Unterstützung bei Anhang A und Hans-Jürgen und Thorben für die kurzen Anmerkungen zum Grundkonzept der Kapitel eins und zwei.

Ach ja, falls es irgend wen interessiert: Es gab in Kapitel zwei ursprünglich eine Aufgabe mehr, die jedoch heraus flog, weil ich die Lösung für zu kompliziert und ineffizient halte. Sollte irgend jemand entweder herausfinden, wie `john` dazu gebracht werden kann, nur die Benutzer schwacher Paßwörter auszugeben oder aber `cut`, die von `john` erzeugte Liste aufzulösen ohne dabei Klammern (gemeint sind runde, solche wie die, die diesen Einschub einschließen) als Spaltentrenner zu werten, dann teilt es mir bitte mit. Meine email Adresse lautet:

`2nd-System-Meister@gmx.de`

Danke.

A.2 Einleitung und Arbeitsanleitung

Dieser Workshop ist eigentlich als Vorbereitung für den LPI 101 gedacht. Meines Erachtens ist er auch für alle anderen nicht verkehrt, da er sich mit praktischen Übungen beschäftigt.

Der Workshop ansich besteht aus drei Teilen. Der erste enthällt fünf kurze Aufgaben, sie sich als Skripte oder ähnliches realisieren lassen. Eine Reihe von Hinweisen können bei völliger Planlosigkeit herangezogen werden. (Falls jemand ohne diese Hinweise weiterkommt, ist das natürlich besser.) Hinter den Hinweisen kommen die Lösungen. Das Kapitel schließt mit einigen Fragen, die sich nach Bearbeitung der Aufgaben ohne Nachzulesen in den Lösungen beantworten lassen sollten.

Im zweite Teil finden sich drei etwas anspruchsvollere Aufgaben. Hier gibt es Hinweise und *weiterführende* Hinweise. Ebenso wie im ersten Teil sollten die ersten Hinweise nur versucht weden, falls es ohne nicht geht, und die weiterführenden, falls die normalen Hinweise nichts bringen. Auch hier gibt es Lösungen, auf Fragen wurde verzichtet.[1]

Sowohl zu den Lösungen des ersten als auch zu denen denen des zweiten Teiles muß gesagt werden, daß es noch andere Lösungen gibt. Dieses sind lediglich welche, die mir auch tatsächlich eingefallen sind.

Im dritten Kapittel wurden nochmals die Fragen aus Teil 1 recycled, des Weiteren kamen noch welche dazu. Auch hierzu gibt es Antworten.

In allen Fällen wurde darauf geachtet, daß Hinweise, weiterführende Hinweise und Lösungen (beziehungsweise Fragen und Antworten) jeweils auf neuen Seiten anfangen.

A.3 Aufgaben I — Shorties

In diesem Teil finden sich kurze Aufgaben, zumeist Skripte oder Dinge, die sich in einer Befehlszeile tippen lassen. Er ist fünfgeteilt: Aufgaben, Tipps, Lösungen, Fragen und Antworten. Die Fragen sollten nach Möglichkeit nach Lösung der Aufgabe *ohne nochmaliges Lesen der Tipps oder der Lösungen* (auch der eigenenen) bearbeitet werden.

[1] Mit anderen Worten: Der Verfasser hate keinen Bock.

A.3.1 Aufgaben

Hier finden sich verschiedene Aufgaben. Sie lassen sich als Skript oder als Befehlszeile ausführen. Die Aufgaben sind alle lößbar!

Benutzer anlegen

Schreiben Sie ein Skript namens `multiuseradd`. Der Aufruf soll mittels
`multiuseradd` ***kennung zahl***
geschehen und ***zahl*** Benutzer namens ***kennung*** `1` bis ***kennung zahl*** (also Bei- spielsweise durch `multiuseradd bastard 3` die Benutzer `bastard 1`, `bastard 2` und `bastard 3`) anlegen. Neben den Konten sollen für die Benutzer auch die Heimverzeichnisse angelegt werden.

Hinweise zu dieser Aufgabe finden sich auf Seite 67, die Lösung steht auf Seite 68.

Benutzer löschen

Als nächstes soll ein Skript `multiuserdel` geschrieben werden, welches in der Lage ist, mit Hilfe von `multiuseradd` erstellte Benutzer zu löschen. Die Syntax soll dabei
`multiuserdel` ***kennung***
lauten (in unserem Beispiel würde `multiuserdel bastard` automatisch die Benutzer `bastard 1`, `bastard 2` und `bastard 3` löschen). Es wird davon aus- gegangen, daß sich die Heimverzeichnisse alle im Verzeichnis `/home` befinden. Die Heimverzeichnisse sollen ebenfalls gelöscht werden.

Hinweise zu dieser Aufgabe finden sich auf Seite 67, die Lösung steht auf Seite 68.

Installation von Paketen

Pakete mit dem `YaST` zu installieren ist vergleichsweise Zeitaufwendig; es werden viele Konfigurations-Skripte abgearbeitet, die eigentlich für die Installation eines Paketes gar nicht nötig sind. Da die Pakete alle im `rpm`-Format vorliegen, ist es also möglich, diese auch über den `rpm` zu installieren. Im Gegensatz zum `YaST` muß `rpm` auf die Datei selbst zugreifen, sprich: Ihre Position (Verzeichnis und CD) müssen dem Benutzer bekannt sein. Wie läßt sich dieses Problem lösen?

Hinweise zu dieser Aufgabe finden sich auf Seite 67, die Lösung steht auf Seite 68.

Der Mail-Account

Die Benutzerin Sabine soll künftig in der Lage sein, an sich einem bestimmten Rechner anzumelden und ihre Mail zu lesen. Sie soll sich nicht mit den

Unwegsamkeiten von Linux belasten. Andere Benutzer, die sich über irgend welche Tricks ihr Paßwort verschafft haben, sollen nicht in der Lage sein, über ihren Account irgend etwas anderes zu tun als Sabines Mails zu lesen. Nach Ansicht der Geschäftsführung ist es Sabine zuzumuten, die Bedienung eines verhältnismäßig interaktiven Programmes wie `pine` zu erlernen. Bitte geben Sie eine Lösungsmöglichkeit an.

Hinweise zu dieser Aufgabe finden sich auf Seite 67, die Lösung steht auf Seite 69.

Wer restriktiert...

In Ihrem System gibt es eine ihnen unbekannte Zahl von Benutzern, denen gemein ist, daß ihr Name die Zeichenfolge `dumm` enthällt. Bedauerlicher Weise finden sich auf verschiedenen Rechnern verschiedene von ihnen. Sorgen Sie dafür, daß all diese Benutzer als Shell eine restriktive bekommen.

Hinweise zu dieser Aufgabe finden sich auf Seite 67, die Lösung steht auf Seite 69.

A.3.2 Tipps

In dieser Sektion finden sich Tipps zur Lösung der Aufgaben. Meistens handelt es sich nur um eine Auflistung der vermutlich dafür gebrauchten Befehle.

Benutzer anlegen

Meines Erachtens läßt sich diese Aufgabe am Besten mit einer `while`-Schleife und dem Befehl `useradd` erledigen. Ebenfalls wichtig ist die Benennung von Übergabeparametern innerhalb des Skriptes.

Benutzer löschen

In diesem Fall empfehle ich eine `for`-Schleife; im Zusammenhang mit dem Inhalt des Verzeichnisses `/home` sollte sich die Aufgabe sehr schnell lösen lassen.

Installation von Paketen

Auf der ersten CD der Distribution gibt es eine Datei namens `INDEX.gz`. Die Lösung läßt sich am besten mit einer Pipe (—) realisieren.

Der Mail-Account

Um des geforderten Account zu erstellen und zu konfigurieren empfielt sich ein kurzer Blick auf die Funktionsweise der Datei `/etc/passwd`.

Wer restriktiert...

Da diese Benutzer alle auf verschiedenen Rechner sitzen, bietet sich die Implementation eines Skriptes an. Da es hierbei um die Ersetzung von Zeilenfolgen in der jeweiligen `passwd` handelt, bietet sich ein Skript für den Befehl `sed` an.

A.3.3 Lösungen

Zu diesen Lösungen läßt sich nur sagen, daß ich sie alle ausprobiert habe; sie funktionieren also. Sicherlich sind es nicht die einzig-wahren Lösungen, andere funktionieren sicherlich auch. Aber es sind immerhin Ansätze.

Benutzer anlegen

Eine Möglichkeit der Lösung ist das folgende Skript:

```
i=1
while [ $i -le $2 ];
  do
    useradd $1_$i -m;
    i=$[i+1];
 done
```

Wer ganz sicher gehen will, kann auch noch eine `if`-Bedingung vorschalten, um zu überprüfen, ob tatsächlich zwei Argumente übergeben wurden. Das Skript könnte dann wie folgend aussehen:

```
if [ $# != 2 ];
   then echo 'Falsche Argumentenzahl!'; exit;
fi;
i=1
while [ $i -le $2 ];
  do
    useradd $1_$i -m;
    i=$[i+1];
done
```

Benutzer löschen

Das folgende Skript erfüllt die Aufgabe bestens:

```
cd /home
for a in $1_*;
  do userdel -r $a
  done
```

Da das Skirpt in einer eigenen Shell ausgeführt wird, hat der `cd`-Befehl am Anfang des Skriptes keine Auswirkung auf die normale Arbeit des Benutzers.

Installation von Paketen

Eine Lösung ist, die Datei `INDEX.gz` auf Platte zu kopieren und dort mittels `gunzip` zu entpacken. Mit dem zusammengesetzten Befehl

`cat INDEX.gz|grep Ziel`
läßt sich das entsprechende Paket finden. Es ist auch möglich, sich das Entpacken zu sparen. Dafür wird statt `cat` der Befehl `zcat` benutzt. Er macht das selbe wie `cat`, wendet aber vorher eine Dekompression aus.

Falls die Datei in `/root` liegt, ergibt beispielsweise der Befehl
`zcat /root/INDEX.gz | grep xjewel`
das Ergebnis
`./CD4/suse/fun2/xjewel.rpm ./CD6/suse/zq2/xjewel.spm`
Da es sich bei `xjewel.spm` um ein Quellpaket handelt, findet sich also das zu installierende Paket auf CD Nummer 4 im Verzeichnis `/suse/fun2`. (Wem das `spm`-Paket nervt, der kann den Befehl um ein `grep rpm` ergänzen, also `zcat /root/INDEX.gz | grep xjewel|grep rpm`.)

Der Mail-Account

Der einfachste Weg wäre die normale Erstellung eines Accountes für Sabine. Nachdem dies geschehen ist, kann in der `/etc/passwd` als Login-Shell ein entsprechendes Programm (mit vollem Pfad!) eingetragen werden (etwa `/usr/bin/pine`). Dadurch wird beim Einloggen dieses Programm gestartet und bei Beendigung oder Abbruch des Programmes der Benutzer ausgeloggt.

Wer restriktiert...

Ein enstprechendes Skript sähe etwa so aus:

```
cp /etc/passwd /etc/passwd.sed
sed -e '/.*dumm.*/ s/\/bash/\/bash -r/' passwd.sed > passwd
rm passwd.sed
```

Die erste Zeile kopiert die `passwd` die eine temporäre Datei `passwd.sed`. Das `sed`-Kommando ersetzt nun in allen Zeilen, die mit Worten, welche die Zeichenfolge `dumm` enthalten, die Zeichenfolge `/bash` durch `/bash -r` und schreibt das Ergebnis in `passwd`. (Die Abfrage auf den etwas komplizierten und die Übersicht des Befehles nicht gerade steigernde `/bash` statt einfach `bash` ist vor allem deshalb drinn, weil ein Benutzer, der die Zeichenfolge `dumm` im Login-Namen hat, womöglich auch die Zeichenfolge `bash` drinn hat...) Die letzte Zeile löscht die temporäre Datei `passwd.sed`. Der Kunstgriff des Kopierens ist erforderlich, weil der `sed` als Filter arbeitet; er kann keine Datei überschreiben, aus der er gerade ließt.

A.3.4 Fragen

Benutzer anlegen

1. Mit welchem Parameter werden bei welchem Befehl Benutzerverzeichnisse bei der Benutzererstellung mit angelegt?

2. In welcher Variable findet sich die Anzahl der an ein Skript übergebenen Parameter?

Benutzer löschen

3. Mit welchem Parameter werden bei welchem Befehl die Benutzerverzeichnisse bei der Benutzerentfernung mit entfernt?

Installation von Packeten

4. Was bewirkt der Befehl `cat`?

5. Was bewirkt der Befehl `zcat`?

6. Was besagt normaler Weise die Endung `.rpm`?

7. Was besagt normaler Weise die Endung `.spm`?

Der Mailaccount

8. Was besagt die letzte Spalte einer Zeile in der Datei `/etc/passwd`?

Wer restriktiert

9. Wie wird die restriktive `bash` gestartet?

10. Welcher Schalter läßt `sed` Dinge ersetzen?

A.3.5 Antworten

Benutzer anlegen

1. Der Befehl, um Benutzer mit Heimverzeichnissen zu erstellen, lautet `useradd -m`.

2. Die Anzahl der an ein Skript übergebenen Parameter steht in der Variable `$#`.

Benutzer löschen

3. Der Befehl zum löschen eines Benutzers inklusive seines Heimverzeichnisses lautet `userdel -r`.

Installation von Packeten

4. Der Befehl `cat` gibt den Inhalt der danach angegebenen Datei auf dem Standard-Ausgabegerät (also meistens dem Bildschirm) aus.

5. Der Befehl `zcat` wird auf Dateien angewand, die mit `gzip` gepackt wurden. Er gibt ihren Inhalt in unkompremierter Form aus.

6. Die Endung `.rpm` kennzeichnet normaler Weise ein Installationspacket des RedHat Packet Managers (`rpm`).

7. Die Endung `.spm` kennzeichnet normaler Weise ein Quellcode-Packet, das mit dem RedHat Packet Manager installiert werden kann.

Der Mailaccount

8. Die letzte Spalte einer Zeile der Datei `passwd` gibt das Programm, das dem entsprechenden Benutzer als Eingabe-Shell dient, an.

Wer restriktiert

9. Die restriktive `bash` wird mit dem Befehl `bash -r` gestartet.

10. Der Schalter, um mit Hilfe eines `sed`-Skriptes etwas zu ersetzen, ist `s`.

A.4 Aufgaben II — Größeres

Hier stehen verschiedene Aufgaben, die weiterreichende Kombinationen von verschiedenen Linux-Bereichen erfordern. Dieser Teil läßt sich in vier Bereiche gliedern: Die Projektbeschreibung, Hinweise, weiterreichende Hinweise und Lösungen.

A.4.1 Projekte

DOOM

Vermutlich eines der sinnhaftesten und wichtigsten Projekte überhaupt... Auf der Seite von **id-Soft** (`www.idsoft.com`) lassen sich verschiedene Quellcode-Pakete herunterladen. Unter anderem auch das von *DOOM for Linux*. Bei diesem DOOM handelt es sich um eine Variante, die unter X lauffähig ist. Allerdings paßt die mitgelieferte `.wad`-Datei nicht dazu. Die Dateien aus *DOOM for Windows 95* laufen hingegen. Die sind an verschiedenen Stellen im Internet erhältlich und heißen `tnt.wad` und `plutonia.wad`.

Bei der herunterzuladenden Datei handelt es sich zunächst um eine `.zip`-Datei, doch nach dem Auspacken zeigt sich die eigentliche Datei als `.tgz`. Das ist die DOSenschreibweise für `tar.gz`. Entgegen den üblichen TARBalls muß dieser hier nicht konfiguriert werden. Statt dessen kann `make` sofort ausgeführt werden. Dies ergibt jedoch einen Fehler, da der Code für eine relativ alte Version von Linux geschrieben wurde. Damals gab es noch eine Datei namens `errnos.h`, über deren Fehlen sich der Compiler bitterlich beklagt; heutzutage läuft das alles über die Datei `errno.h`.

Diese DOOM-Variante hat den Nachteil, daß sie nur bedingt lauffähig ist; insbesondere läuft sie nur mit 8 Bit Farbtiefe.

Aufgaben:

- Finden Sie die Dateien, in denen sich der veraltete Aufruf befindet und beseitigen Sie das Problem.
- Richten Sie einen Benutzer `doom` ein, in dessen Home-Verzeichnis sich die zu DOOM gehörigen `.wad`-Dateien finden. Des weiteren soll das Ergebnis des `make`-Befehles, also die ausführbare Datei `linuxxdoom`, in dieses Heimverzeichnis geschrieben werden.
- Schreiben Sie einen sogenannten *Wrapper* namens `sx` zu `startx`. Dieser soll X mit den für `linuxxdoom` erforderlichen Parametern aufrufen.

Nach Abschluß dieser Aufgaben sollten Sie in der Lage sein, sich unter dem Namen `doom` anzumelden, mit `sx` ein für DOOM modifiziertes X hochzuschießen und mit Hilfe des Befehles `linuxxdoom -f plutonia.wad` DOOM zu starten.

- Sorgen Sie dafür, daß jeder beliebige Benutzer auf ihrem Rechner mit Hilfe von `sx` das modifizierte X starten und DOOM spielen kann.

`/home`, sweet `/home`

Die folgende Aufgabe erfordert mindestens zwei Rechner, da mit Netzwerkdateisystemen gearbeitet werden soll.

Das /home-Verzeichnis der verschiedenen Benutzer in einem Netzwerk soll — egal, wo im Netzwerk sie sich einloggen — stets ihr eigenes sein. Darüber hinaus soll es eine Plattenbegrenzung von 5 MByte für normale Benutzer und 20 MByte für bevorzugte Benutzer geben. Ferner wird davon ausgegangen, daß die Menge der Benutzer (und somit der Platzbedarf) in Zukunft extrem ansteigen kann. Implementieren Sie eine entsprechende Lösung.

(Es wird davon ausgegangen, daß auf allen Rechnern die gleichen Benutzer eingerichtet sind; das ist zwar höchst unwarscheinlich, aber die Konfiguration von Diensten wie yp ginge entschieden zu weit.)

Mit Nachdruck

In Ihrer Firma sollen Drucker eingerichtet werden. Neben einem besonders guten Drucker für die Administratoren kommen auch vier normale für das Fußvolk an. Bei diesen normalen wurden aus verschiedenen Gründen die Netzwerkkarten noch nicht mitgeliefert, sie sollen jedoch in etwa zwei Monaten nachkommen. Sie sollen diese Drucker so einrichten, daß

- der Administratorendrucker nur von den Rechnern der Administratoren benutzt werden kann,
- die anderen Drucker in vier verschiedenen Subnetzen von jeweils allen Rechnern ihrer Subnetze ansprechbar sind,
- die Umstellung möglichst einfach von statten geht (am besten gar nicht bemerkt wird).

In dieser Aufgabe wird davon ausgegangen, daß es keinen DNS gibt (das ist zwar ziemlich unwarscheinlich und macht die Saache komplizierter, sorgt aber für eine interessantere und dem LPI eher entsprechende Aufgabenstellung. . .) und für die Druckerverwaltung lpr benutzt wird. (Ich habe keine Ahnung, wie sich das Problem mit CUPS lösen läßt.)

A.4.2 Tipps

DOOM

Um die entsprechende Änderung des Quellcodes vorzunehmen, muß die entsprechnede Datei erst einmal gefunden werden. Als Idee dafür können die Aufgaben *Benutzer löschen* und *Installation von Paketen* (Lösungen siehe Kapitel A.3.3, S. 68f.) dienen.

Für den zweiten Teil der Aufgabe empfielt sich `man startx`. Um sich mit dem letzten Teil auseinander zu setzen, ist es erforderlich, in die an verschiedenen Orten liegenden README-Dateien zu sehen.

`/home`, sweet `/home`

Die verschiedenen Bereiche, mit denen das Problem gelößt werden kann, sind `export`, `LVM` und `quota`.

Mit Nachdruck

Besonders zwei Dateien und ihre Funktionsweise sind für die Lösung dieses Problemes besonders wichtig: die `/etc/hosts` und die `/etc/printcap`. Ach ja, für die Konfiguration sollte `YaST` vergessen und `lprsetup` benutzt werden.

A.4.3 Weiterreichende Tipps

DOOM

Um mit `startx` eine andere Konfiguration zu bekommen, als in der Datei `XF86Config` angegeben, wird eine andere Konfigurationsdatei und ein bestimmter Schalter, um diese anzuwählen, benötigt. Um allen Benutzern den Zugriff auf die `.wad`-Dateien zu ermöglichen, muß allen der Pfad dahin bekannt sein. `linuxxdoom` greift dazu auf die Variable `DOOMWADDIR` zu.

`/home`, sweet `/home`

Auch in der Datei `/etc/fstab` können im Netzwerk freigegebene Verzeichnisse angegeben werden. Des weiteren können LVM-Volumen freigegeben und mit Quotas versehen werden. Als letzer Hinweis mag dienen, daß Quotas auch für Gruppen möglich sind.

Mit Nachdruck

Auch an Rechnern angeschlossene Drucker können via *bypass* angesprochen werden. Weiter ist es sicherlich einfacher, die IP-Adressen von vier Rechnern, als die Konfigurations-Dateien aller anderer Rechner zu ändern.

A.4.4 Lösungen

DOOM

Der erste Teil der Aufgabe besteht in dem Aunden der zu patchenden Datei. Folgende Befehlszeile dürfte dabei helfen:

```
for a in *; do if cat a$ | grep errnos.h > /dev/null;
then echo a$; fi; done
```

Die `for`-Schleife geht sämmtliche in dem Verzeichnis befindlichen Dateien durch. Der `cat | grep` Befehl sorgt dafür, daß in den Dateien nach der Zeichenfolge `errnos.h` gesucht wird. Nervige Fehlermeldungen und Nebenausgaben landen durch `> /dev/null` im Datennirvana.

(Theoretisch wäre es auch möglich, das Ergebnis gleich weiter zu nutzen und den `sed` darauf loß zu lassen.)

Der Wrapper `sx` ist vor allem deshalb nötig, damit `XFree86` mit einer Farbtiefe von 8 Bit gestartet wird. Mit etwas anderem arbeitet `linuxxdoom` nicht zusammen.

Glücklicher Weise ist es möglich, `startx` mit einer bestimmten Konfigurationsdatei aufzurufen. Dies geschieht über den Parameter `-xf86config`. Des weiteren ist es eine gute Idee, bei dieser Gelegenheit gleich einen Windowmanager festzulegen. Je weniger Resourcen er frisst, desto besser ist das. Daher bietet sich da `twm` an.

Doch zunächst muß eine Konfigurationsdatei erstellt werden, welche eine Farbtiefe von 8 Bit einstellt. Das kann zum Beispiel durch das Kopieren der originalen `XF86Config` nach `XF86Config.doom` geschehen. Der interessante Eintrag findet sich in der Sektion `Screen` und lautet `DefaultDepth`.

Der Wrapper ansich ist denkbar einfach aufgebaut: Die Zeile
`/usr/X11/bin/startx /usr/X11/bin/twm -xf86config XF86config.doom`
reicht vollkommen aus. Es darf dabei nicht vergessen werden, die Rechte auf ausführen zu setzen (etwa mit `chmod a+x sx`). Wird `sx` danach in das Verzeichnis `/usr/bin` kopiert, kann jeder beliebige Benutzer es starten.

Um jeden beliebigen Benutzer das Ausführen von `linuxxdoom` zu ermöglichen, muß jedem die Position der `.wad`-Dateien bekannt sein. Eine Möglichkeit, dies zu erreichen, besteht in dem Erstellen eines entsprechenden Verzeichnisses (etwa `/usr/doomwads`) und dem Eintragen der Zeile
`export DOOMWADDIR=/usr/doomwads`
in die Datei `/etc/profile`.

`/home`, sweet `/home`

Als Fileserver dient uns ein Rechner mit SCSI-Platten, da sich diese leichter und in größerer Zahl nachrüsten lassen als IDE-Platen. Letzten Endes ist das aber egal, da auf den genauen Syntax der LVM-Einrichtung nicht eingegangen wird. Der Server trägt den Namen `fserver`.

Der erste Schritt wäre das Erstellen eines LVM-Volumes auf `fserver` (wir gehen hier von der virtuellen Plate `net` des Volumes `lvm` aus). Das hat den Vorteil, daß dieses Volumen bei Platzbedarf vergrößert werden kann. Wir gehen davon aus, daß dieses Volumen nach der Formatierung unter dem Namen `/home/nethome` erreichbar ist. (Dies wird durch den Eintrag
`/dev/lvm/net /home/nethome ext2 defaults,grpquota 1 2`
in die Datei `/etc/fstab` des Fileservers erreicht. Der `grpquota`-Eintrag ist erforderlich, um später Quotas für Gruppen verwalten zu können.)

Nachdem dies geschehen ist, kann eine Gruppe `befusers` angelegt werden. In die Gruppe `befusers` kommen all die Benutzer, die später 20 MByte zu Verfügung haben sollen. Diese Quotas können danach auch gleich eingerichtet werden (mit `edquota -g`). Dabei soll die Gruppe `users` 5 MByte, `befusers` wie beschrieben 20 MByte bekommen.

Um das Verzeichnis mit Lese- und Schreibrechten zu exportieren muß die folgende Zeile in die Datei `/etc/exports` eingetragen und der NFS-Server mittels `rcnfsserver restart` neu gestartet werden:
`/home/nethome *(rw)`
Auf den verschiedenen anderen Rechnern muß der Eintrag des `/home`-Verzeichnisses in `/etc/fstab` nun lauten:
`fserver:/home/nethome /home ext2 defaults,grpquota 1 2`
(Wer will, kann sich ein `sed`-Skript schreiben, um das auf den verschiedenen Rechnern zu erledigen.)

Nachdem die Einträge in `/dev/passwd`, `/dev/shadow` und `/etc/group` synchronisiert wurden sollte das System laufen.

Mit Nachdruck

Der Administratorendrucker ist am einfachsten einzurichten: Der kommt am Besten an einen der Administratoren-Rechner in der Administratoren-Stube und gut ist. Da auf diesen Rechnern kein anderer Benutzer etwas zu suchen hat ist das Problem damit erledigt.

Für die anderen Drucker ist es zunächst erforderlich, vier Rechner (in jedem der Subnetze einen) bereit zu stellen. Auf diesen werden die Drucker installiert. Als nächstes wird auf jeweils einem anderen Rechner pro Subnetz der Drucker als Bypass installiert. Zu diesem Zweck wird als erstes in die Datei `/etc/hosts` der den Drucker verwaltende Rechner unter dem Namen `printer` (oder einem anderen, leicht zu merkenden Namen) eingetragen.

Dieser Name wird bei der Bypass-Installation als Adresse des Druckers ange- geben. Danach kann die Datei `/etc/printcap` auf eine Diskette (oder ein im Netz freigegebenes Verzeichnis) kopiert werden. Folgendes Skript übernimmt danach die Konfiguration der anderen Rechner:

```
cp printcap /etc
echo IP-Adresse printer >> /etc/hosts
```

wobei `IP-Adresse` die Adresse des den Drucker verwaltenden Rechners ist.

Werden nun die Netzwerkkarten nachgeliefert, so braucht nur der Eintrag in den `/etc/hosts`-Dateien geändert zu werden. Diese Aufgabe kann beispielsweise `sed` in einem Skript erfüllen:

```
cp /etc/hosts /etc/hosts.sed
sed -e '/AIP *printer/ s/AIP /NIP /' /etc/hosts.sed > /etc/hosts
rm /etc/hosts.sed
```

wobei `AIP` die alte Adresse, also die des Rechners, der vorher den Drucker verwaltet hat, und NIP die neue Adresse, also die des Druckers, ist.

Das Ganze würde übrigens auch dann funktionieren, wenn die neuen Netzwerkkarten mit den IP-Adressen der vorher zuständigen Drucker ausgerüstet wurden. Damit würde der ganze Skript-Kram auch wegfallen.

A.5 Wissensfragen

Hier finden sich Fragen, die sich mit kurzen Sätzen beantworten lassen. Die ersten zehn davon entstammen Teil A.3.4, S. 70 und dienen nur der Wiederholung. Auch zu diesen Fragen gibt es Antworten.

A.5.1 Fragen

1. Mit welchem Parameter werden bei welchem Befehl Benutzerverzeichnisse bei der Benutzererstellung mit angelegt?

2. In welcher Variable findet sich die Anzahl der an ein Skript übergebenen Parameter?

3. Mit welchem Parameter werden bei welchem Befehl die Benutzerverzeichnisse bei der Benutzerentfernung mit entfernt?

4. Was bewirkt der Befehl `cat`?

5. Was bewirkt der Befehl `zcat`?

6. Was besagt normaler Weise die Endung `.rpm`?

7. Was besagt normaler Weise die Endung `.spm`?

8. Was besagt die letzte Spalte einer Zeile in der Datei `/etc/passwd`?

9. Wie wird die restriktive `bash` gestartet?

10. Welcher Schalter läßt `sed` Dinge ersetzen?

11. Wie kann gewährleistet werden, daß der Befehl `locate` mindestens alle Dateien findet, die vor drei Tagen auf den Rechner gekommen sind?

12. Wie kann die die Aktualität dahingehend gesteigert werden, daß mindestens alle Dateien vom Vortag gefunden werden?
13. Auf welche Weise kann der manchmal störende Maus-Treiber in der Text-Konsole zut Laufzeit[2] deaktiviert werden?
14. Welche Voraussetzungen gibt es, um eine Partition mittels des Befehles `dump` sichern zu können?
15. Wie lautet die Rechtekombination `-rwsrwSr--` in Oktalnotation?
16. Welche Datei ist für die Konfiguration von `lilo` verantwortlich?
17. Welcher Befehl bietet Informationen zum Aufbau der dieser Datei?
18. Was muß nach Änderung der Datei getan werden, um die Änderungen zu aktivieren?
19. Was bewirkt die folgende Zeile?
    `cat /var/log/boot.msg|grep gpm && killall -9 gpm`
20. Mit welchem Befehl können die jeweils aktuellen Systemnachrichten (messages) auf dem mit der Taste `<F9>` erreichbarem Schirm abgebildet werden?
21. Was besagt der Eintrag `x` in der zweiten Spalte der Datei `/etc/passwd`?
22. Was bewirkt die folgende Zeile?
    `foo & bar; sau ; fen & Leber & kaputt &`
    (Es wird davon ausgegangen, daß es sich bei allen Worten um gültige Befehle handelt. . .)
23. Welche Rechte haben neuerstellte Verzeichnisse und Dateien nach Aufruf des Befehles `umask 024`?
24. In welcher Datei stehen bei SuSE die Start-Meldungen des Rechners?
25. Welche `man`-Page-Sektion beinhaltet Administratoren-Programe?

[2]Also nachdem das System gestartet wurde.

A.5.2 Antworten

Die Antworten auf die Fragen wurden bewußt knapp gehalten, damit es einfacher ist, sich später daran zu erinnern. Die ersten zehn davon entstammen Teil A.3.5, S. 71.

1. Der Befehl, um Benutzer mit Heimverzeichnissen zu erstellen, lautet `useradd -m`.
2. Die Anzahl der an ein Skript übergebenen Parameter steht in der Variable `$#`.
3. Der Befehl zum löschen eines Benutzers inklusive seines Heimverzeichnisses lautet `userdel -r`.
4. Der Befehl `cat` gibt den Inhalt der danach angegebenen Datei auf dem Standard-Ausgabegerät (also meistens dem Bildschirm) aus.
5. Der Befehl `zcat` wird auf Dateien angewand, die mit `gzip` gepackt wurden. Er gibt ihren Inhalt in unkompremierter Form aus.
6. Die Endung `.rpm` kennzeichnet normaler Weise ein Installationspacket des RedHat Packet Managers (`rpm`).
7. Die Endung `.spm` kennzeichnet normaler Weise ein Quellcode-Packet, das mit dem RedHat Packet Manager installiert werden kann.
8. Die letzte Spalte einer Zeile der Datei `passwd` gibt das Programm, das dem entsprechenden Benutzer als Eingabe-Shell dient, an.
9. Die restriktive `bash` wird mit dem Befehl `bash -r` gestartet.
10. Der Schalter, um mit Hilfe eines `sed`-Skriptes etwas zu ersetzen, ist `s`.
11. Durch einen Eintrag in die /etc/crontab, der dafür sorgt, daß die `locate`-Datenbank alle drei Tage mitels `updatedb` erneuert wird.
12. Durch Kopieren (oder Linken) von `updatedb` nach `/etc/cron.daily`.
13. Mit dem Befehl `killall -9 gpm` wird die zeimlich nervige Maus-Unterstützung weggeschossen.
14. Der Befehl `dump` funktioniert nur mit Partitionen, die mit dem ext2-Dateisystem formatiert wurden.
15. Die Kombination `-rwsrwSr--` läßt sich wie folgend aufschlüsseln:

SUID+SGID	4		+		2	=	6
rwx	4	+	2	+	1	=	7
rw-	4	+	2			=	6
r–	4					=	4

Damit lautet die Oktal-Notation: `6764`

16. Die Datei `/etc/lilo.conf`.
17. Der Befehl `man 5 lilo.conf`, in manchen Distributionen auch nur `man lilo.conf`.
18. Der Befehl `lilo` muß aufgerufen werden, um die Änderungen zu aktivieren.
19. Durch diese Zeile wird zunächst überprüft, ob beim Systemstart der `gpm` gestartet wurde. Falls dies der Fall ist, wird wie in Frage 13 verfahren.
20. Mit dem Befehl `tail -f /var/log/messages > /dev/tty9`.
21. Das `x` besagt, daß es ein Paßwort für den entsprechenden Benutzer gibt und es sich in verschlüsselter Form in der Datei `/etc/shadow` befindet.
22. Der Befehl `foo` wird gestartet und im Hintergrund ausgeführt. Gleichzeitig wird der Befehl `bar` ausgeführt. Nach dessen Ausführung wird der Befehl `sau`, nach dessen Ende die Befehle `fen`, `Leber` und `kaputt` gestartet. Diese letzten drei laufen im Hintergrund.
23. Verzeichnisse haben die Rechte `753`, Dateien, denen standardmäßig das `x`-Recht fehlt, die Rechte 642.
24. In `/var/log/boot.msg`
25. Sie stehen in Sektion 8.

A.6 Schlußwort und Danksagungen

Hiermit ist das Ende des Workshopes erreicht. Die meisten Themenbereiche, die im Unterricht besprochen wurden, konnten hier angewandt werden.

Danken möchte ich an dieser Stelle erst einmal Thorben und Hans-Jürgen, die sehr hilfreiche Anmerkungen zum grudnkonzept der Teile A.3 und A.4 gaben. Der erstgenannte war zudem der Auslöser für die Erstellung des Workshopes, letzgenannter wieß mich später auf einen Fehler in einer der Aufgaben aus A.5 hin. Ebenfalls danken möchte ich an dieser Stelle Olaf L., der ebenfalls recht hilfreiche Anmerkungen zum Anhang B[3] lieferte.

Anmerkungen und Berichtigungen nehme ich dankend unter der Adresse
`2nd-System-Meister@gmx.de`
entgegen.

[3]Damals noch zu diesem Workshop gehörig, in dieser Zusammenfassung als eigener Anhang ausgegliedert

Anhang B

Erstellung eines rpm-Paketes

In diesem Anhang wird die Erstellung von `rpm`-Paketen anhand des Paketes `linuxxdoom.1.10-666.i386.rpm`[1] beschrieben. Das hier erstellte Paket läuft allerdings nur unter SuSE 7.2 fehlerloß...

B.1 Warum ein rpm-Paket erstellen?

Auf den ersten Blick erscheint es nicht sonderlich sinnvoll, selbst ein solches Paket zu erstellen; die meisten Programme sind bereits als `rpm` vorhanden, und falls sie das nicht sind, so geht die Installation zwar über die Kenntnisse eines normalen Benutzers hinaus, aber die sollen ja normalerweise auch nichts installieren. Es stellt sich also die Frage: Wozu der Aufwand?

Eine Antwort darauf lautet: Weil es schneller ist! Das durchschnittliche Paket erfordert es, durch irgend welche Skripte konfiguriert, kompiliert und installiert zu werden. Natürlich ist es möglich, sich dazu ein Skript zu schreiben, doch dann wäre ein `rpm`-Paket nicht wesendlich aufwendiger; tatsächlich ließe es sich sogar einfacher in skriptgesteuerte Installationen einbinden.

Hinzu kommt, daß oft Administratoren an verschiedenen Stellen gleiche Probleme zu lösen haben. Die sind sicher sehr froh, wenn ihnen das Programm als `rpm` zu Verfügung steht — dann müssen sie sich nicht erst durch kryptischen Intsallations-Code oder unvollständige Dokumentationen kämpfen. `rpm`-Pakete sind überall gleich.

B.2 Voraussetzungen

Um ein `rpm`-Paket erstellen zu können, wird zunächst einmal der Sourcecode des Programmes benötigt. Dieser sollte als `tar`-Ball vorliegen (meistens aus Platzgründen gezippt). Sollte zu dem Programm bereits ein Patch vorliegen, wäre es sicherlich keine schlechte Entscheidung, den ebenfalls zu besorgen.

[1] Dieses Paket wurde in einer Weiterführung des Projektes **DOOM** (siehe A.4.1, S. 72) erstellt.

B.3 Die Verzeichnisse

Der gesammte Vorgang findet im Verzeichnis `/usr/src/packages` statt. In diesem Verzeichnis finden sich die Unterverzeichnisse `BUILD`, `RPMS`, `SOURCES`, `SPECS` und `SRPMS`.

Für das Erstellen in unserem Beispiel werden vor allem die Verzeichnisse `SOURCES` und `SPECS` benötigt; im ersten liegt der `tar`-Ball, im zweiten stehen die Spezifikationen des Paketes.

In dem Verzeichnis `BUILD` findet sich das zwischenzeitliche Kompillat (also zunächst noch nichts), `RPMS` beinhaltet das fertige Paket (anfangs also auch nichts), in `SRPMS` liegen die `spm`-, also die Quellcode-Pakete.

B.4 Das SPEC-File

Aufgrund des `SPEC`-Files wird das `rpm`-Paket erstellt; in diesem Beispiel heißt die Datei `linuxxdoom-1.10-666.spec`, der Name setzt sich aus dem Programm-Namen, der Version und der Release-Nummer zusammen. (Na ja, 666 ist nicht wirklich die Release-Zahl... Aber irgend wie sollte diese veränderte Version sich vom Original 1.10 unterschieden. Ich hatte aber keine Ahnung, welches Release das wäre, und 666 erscheint mir irgend wie sinnvoll.) In der Datei steht einiges an Informationen:

```
Summary: Ein modifiziertes DOOM, einfacher zu handhaben
Name: linuxxdoom
Version: 1.10
Release: 666
Copyright: public domain
Source: linuxxdoom-1.10-666.tar.gz
Group: Amusement/Games
#BuildRoot: /var/tmp/%{name}-build
%description
Diese Version von DOOM basiert auf dem offiziellen Quellcode
von id-Soft. Sie enthaellt neben einem lauffaehigem DOOM, das
im /usr/bin-Verzeichnis landet, auch zwei WAD-Dateien, die
damit zusammen arbeiten und einen Wrapper zu startx namens sx
der zum Spielen von DOOM leider erforderlich ist. Diese
Fassung ist fuer SuSE, fuer andere muessen vermutlich ein paar
Dateien geaendert werden.
Viel Spass!
2nd.-System-Meister
```

Dies sind die Informationen, die rpm -qi zu Tage fördern soll.

```
%prep
%setup -n linuxxdoom-%{version}-%{release}
```

```
%build
make
%install
make install
```

Dieser Teil beschäftigt sich mit den verschiedenen Erstellungs-Abläufen. `%setup -n` setzt den Namen des Paketes. Hierbei werden die im obersten Teil gesetzten Variablen genutzt. `%build` beinhaltet die Befehle, die zum tatsächlichen Erstellen führen. Bei vielen Programmen steht zunächst noch ein `./configure` drinn, doch in diesem Fall reicht ein einfaches `make`; linuxxdoom muß immerhin nicht mehr konfiguriert werden. Unter `%install` finden sich die Routinen, die zur Installation des Paketes führen. In diesem Fall handelt es sich um ein einfaches `make install`. Statt dessen könnten die Schritte, die durch das `Makefile` mit dem Target `install` ausgeführt werden, auch an dieser Stelle stehen. Meistens handelt es sich um Kopier-Aufträge. Änderungen an Dateien können hier nicht vogenomen werden!

```
%files
/usr/local/bin/linuxxdoom
/usr/local/games/linuxxdoom
/etc/X11/XF86Config.doom
/usr/bin/sx
/usr/share/doc/packages/doom
/usr/doomwads
```

Hier sind alle auf das System kopierten Dateien mit ihren Pfaden aufgelistet. Der Eintrag ist vor allem deshalb wichtig, weil `rpm` von sich aus nicht über die Möglichkeit verfügt, diese Dateien mit zu protokollieren. Besonders wichtig ist das wegen der Deinstallation; immerhin muß `rpm` ja wissen, was er löschen darf.

```
%changelog
* Tue Jan 15 2002 - 2nd-System-Meister@gmx.de
- Erstellung
```

Diese Stelle ist in diesem Falle relativ unwichtig; hier werden die an der Datei vorgenommenen Änderungen protokolliert. Das ist vor allem für Entwickler interessant, doch auch Administratoren können sich gegebenenfalls dafür interessieren (etwa wenn da steht, wann ein bestimmter Fehler entfernt wurde).

B.5 Das `Makefile`

An dieser Stelle wird nicht das komplette `Makefile`, sondern lediglich die Installationsroutine besprochen:

```
install:
cp -P linux/linuxxdoom /usr/local/games
cp -P linuxxdoom /usr/local/bin
cp sx /usr/bin
cp XF86Config.doom /etc/X11
cp -r doomwads /usr
mkdir /usr/share/doc/packages/doom
cp doc/* /usr/share/doc/packages/doom
```

Diese Zeilen könnten ebenso gut im der `SPEC`-Datei stehen. Hier werden die verschiedenen Dateien an die ihnen zustehenden Orte kopiert. Es steckt wirklich nicht mehr dahinter.

B.6 Die Erstellung ansich

Das `rpm`-Paket ansich wird im `SPECS`-Verzeichnis durch den Befehl `rpm -bb linuxxdoom-1.10-666.spec`[2] erstellt. Das Program führt dabei neben der Erstellung auch eine Installation durch; dadurch wird sicher gegangen, daß diese überhaupt möglich ist. Am Ende findet sich das Paket im Verzeichnis `/usr/src/packages/RPMS/i386`. (Irgend wie scheint es eine Möglichkeit zu geben, `rpm`-Pakete auch für andere Architekturen zu kompilieren, das landet dann in den entsprechenden anderen Verzeichnissen.)

[2] Falls der aktuelle Pfas etwas anderes ist als das `SPECS`-Verzeichnis, muß der korrekte Pfad zu der `SPEC`-Datei natürlich angegeben werden.

Anhang C

root-Paßwort vergessen — was tun?

Es kann gelegendlich vorkommen, daß einem Administrator das `root`-Paßwort abhanden kommt oder — das ist noch wahrscheinlicher — daß er einen Rechner repareiren soll, von dem er es nicht kennt. Die beiden hier aufgeführten Methoden sollten zum Ziel führen. Sie können jedoch auc genutz werden, um sich wiederrechtlichen Zugang zu verschaffen!

C.1 Grundlagen: Die Dateien passwd und shadow

Früher gab es nur eine Datei für die Paßwörter, nämlich `/etc/passwd`. Trotzdem die Paßwörter verschlüsselt darin standen, waren sie doch für alle Benutzer frei zugänglich. Und mit einem entsprechenden Programm ließ sich jede Verschlüsselung auf Dauer knacken. Heute gibt es an der Stelle, wo früehr das Paßwort stand, nur noch drei verschiedene Zeichen:

`x` bedeutet, daß es in der Datei `/etc/shadow` ein verschlüsseltes Paßwort steht,

`*` bedeutet, daß der Benutzer sich nicht anmelden kann (der Account ist deaktiviert).

(kein Eintrag) bedeutet, daß kein Paßwort erforderlich ist.

In der `/etc/shadow` liegen die verschlüsselten Paßwörter, und auf diese Datei hat nur `root` Zugriff. (Dies ist auch der Grund, warum das Programm als SUID root ausgeführt werden muß.)

C.2 Einstieg über lilo

Falls der Administrator eines Rechners kein `lilo`-Paßwort gesetzt hat (siehe Kapitel 5.1, S. 26), ist es möglich, über den `lilo` einzudringen. Hierzu wird dem Kernel der Parameter `init` im Zusammenhang mit einer Shell übermittelt. In unserem Beispiel gehen wir von der `sh`-Shell aus, da diese im Gegensatz zur `bash` auf fast allen Systemen vorhanden ist. Dies geschieht im Startmenü. Nach Anwahl des richtigen Kernels wird dahinter `init=/bin/sh` eingegeben.

Zu beachten ist jedoch, daß zu diesem Zeitpunkt noch der amerikanische Tastatur-Treiber geladen ist. Die wichtigsten Zeichen sind erfahrungsgemäß `:`, `/` und `=`. Sie liegen auf den Tasten `ö`, `-` und `'`.

Nach Start des Systemes ist es noch nicht möglich, schreibend auf das System zuzugreifen. Mit dem Befehl `mount / -o remount,rw` wird dem Abhilfe geschaffen. Falls der Befehl `passwd` zur Verfügung steht, kann das `root`-Paßwort damit überschrieben werden. Fall nicht, bietet es sich an, in der Datei `/etc/shadow` den zweiten Eintrag in der Zeile des Benutzers `root` zu löschen. Bei dem zweiten Eintrag[1] handelt es sich um das verschlüsselte Paßwort.

Nach diesen Aktionen sollte ein `sync` durchgeführt werden, damit alle Änderungen geschrieben und ein Neustart mittels `/sbin/reboot` erfolgen.

Gesichert werden kann der Rechner auf die in Kapitel 5.1, S. 26 beschriebene Weise.

C.3 Der Einstieg über eine Rettungsdiskette

Dazu läßt sich eingentlich nur sagen, daß jedes System, auf dem gebootet werden kann, offen ist. Offen bedeutet in diesem Zusammenhang, daß ein Benutzer sich dort Rootrechte verschaffen kann. Um einen Server dagegen zu sichern gibt es verschiedene Möglichkieten. Insbesondere ist der Server für alle Benutzer unzugänglich aufzustellen. Weiter sollte nicht vergessen werden, die Möglichkeit, sich remote als `root` einzuloggen in soweit einzuschränken, daß das Paßwort auf jeden Fall abgefragt wird. Tja, und dann gibt es noch die Möglichkeit, allen potentiellen Angreifer die Hände und Füße abzuhacken...

Falls ein Benutzer den Rechner von Medien booten kann, ist es ihm möglich, mit einem eigenen Betriebsystem zu starten, in dem er selbst `root` ist. Damit können ihm die meisten Sicherheitsvorkehrungen auf dem Rechner herzlich egal sein...

[1] Die einzelnen Einträge werden mittels : von einander getrennt.

Literaturverzeichnis

[1] Frisch, Æ leen: *UNIX System Administration.* O'Reilly / International Thomsen Verlag GmbH & Co. KG, 1998. Deutsche Übersetzung von Peter Klicman.

[2] Poppele, Werner und Hans Baier: *Linux — Grundlagen des Betriebsystems.* Qualifizierungshandbuch. CDI GmbH, München, 2001.

[3] Siever, Ellen, Stepehn Spainhour, Stepehn Figgins und Jessica P. Hekman: *Linux in an nutshell.* O'Reilly & Associates, Inc., 3 Auflage, 2000.

Abbildungsverzeichnis

Index