
millin

Hans-Werner Heinl

Das Linux-Befehle-Buch

Bibliografische Information Der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über `http://dnb.ddb.de` abrufbar.
ISBN 3-938626-01-1

© 2007 Nicolaus Millin Verlag GmbH, Lohmar (http://www.millin.de)
Umschlaggestaltung: Fritz Design GmbH, Erlangen
Gesamtlektorat: Nicolaus Millin
Fachlektorat:
Satz: LaTeX
Druck: Gallus, Berlin
Printed in Germany on acid free paper.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Es ist nicht ganz einfach, den richtigen Einstieg in ein Thema wie Linux-Befehle zu finden. Die meisten Leser werden schon rudimentäre Kenntnisse über Linux haben. Dennoch wollte ich auch dem Einsteiger die Chance bieten, etwas Grundlagenwissen über die Shell, Prozesse und das Dateisystem zu erwerben. Daher beschäftigen sich die ersten drei Kapitel mit diesen Grundlagen. Im vierten Kapitel finden Sie dann eine Befehlsübersicht mit Beispielen, in denen viele Befehle mit den wichtigsten Parametern vorgestellt werden. Es gibt bestimmt einige Befehle in dieser Übersicht, die Sie nie brauchen werden. Auf der anderen Seite werden sie ggf. nach dem ein oder anderen Befehl vergeblich suchen. Ein umfassendes Werk aller Befehle zu erstellen und auf einem aktuellen Stand zu halten, dürfte kaum möglich sein. Für spezielle Anforderungen verweise ich deshalb schon hier auf die entsprechenden Manual-Seiten, die bei jeder Distribution vorhanden sind.

Wie schon erwähnt erhebt dieses Werk keinen Anspruch auf Vollständigkeit. Bei der Befehlsübersicht wurden zum Teil bewusst Option, die erfahrungsgemäß nicht oder nur sehr selten benötigt werden, ausgelassen.

Alle hier beschriebenen Befehle sind zum Großteil in der Standardinstallation der marktüblichen Linux-Distributionen enthalten oder können zumindest nachinstalliert werden. Sollten manche Befehls-Option nicht funktionieren, kann das an der Version des Kommandos liegen.

Ein paar Hinweise zur Notation und zu Schreibweisen in diesem Buch:

Bei den Beispielen können Sie am dargestellten Prompt erkennen ob der Befehl von einem normalen Benutzer aufgeführt werden kann, oder root-Rechte erfordert:

`user@linux:~ >`	ist der Prompt des normalen Benutzers.
`linux: #`	ist der Prompt des Systemverwalters root.

Befehle, Dateinamen und Bildschirmausgaben werden allgemein in einer anderen Schriftart dargestellt.
`echo Hallo` ist beispielsweise die Schriftart für Kommandos. Ausgaben der Programme werden zum Teil, auch aus Platzgründen, in einer kleineren Schrift dargestellt.

```
drwxr-xr-x  2 user users     48 2004-08-13 13:01 bin
drwxr-xr-x  2 user users     80 2004-08-13 13:01 Documents
-rw-------  1 user users   1878 2004-08-25 11:19 nohup.out
drwxr-xr-x  2 user users     80 2004-08-13 13:01 public_html
drwxr-xr-x  3 user users    656 2004-08-30 17:16 work
```

Oben sehen Sie eine Ausgabe des Befehls `ls -l`.

Bei Syntax-Beschreibungen zu Kommandos bedeuten eckige Klammern, dass die Optionen oder Werte dazwischen auch weggelassen werden können. Kursiv dargestellte Wörter sind Platzhalter für Werte, die anstelle dieser Wörter angegeben werden müssen.

Der Begriff Befehl und Kommando wird im gesamten Buch nicht unterschieden und gleichbedeutend verwendet.

Zu erwähnen ist noch, daß Linux zwischen Groß-/und Kleinschreibung unterscheidet.

Kapitel 2

Die Shell

2.1 Einführung

Je nach Anwendungsbereich haben Betriebssysteme Schnittstellen, die es dem Anwender oder Systemadministrator erlauben, verschiedene Prozesse zu starten.

Ein Videorecorder hat beispielsweise

- eine LCD-Anzeige,
- das Fernsehgerät als Ausgabegerät,
- das Videoband als Datenspeicher und
- die Fernbedienung als Eingabegerät.

Jedem Druckknopf der Fernbedienung ist eine bestimmte Anwendung (Prozess) zugeordnet. Der Benutzer kann durch Drücken der Play-Taste den eingelegten Film starten, d. h., es wird dem Videorecorder mitgeteilt, er soll das Programm zum Abspielen des Films starten. Auf der LCD-Anzeige wird der aktuelle Status des Programms angezeigt (Zeit, Modus) und auf dem Bildschirm der Film. Die Benutzerschnittstelle dieses Betriebssystems stellt uns die Möglichkeit zur Verfügung, bestimmte Kommandos an den Videorecorder zu schicken.

Bei Computersystemen stehen dem Benutzer mehr Kommandos und Ressourcen zur Verfügung als bei einem Videorecorder, allerdings ist der Ablauf ähnlich: ein Kommando wird irgendwie vom Benutzer eingegeben und anschließend vom Betriebssystem interpretiert und verarbeitet. Die meisten Computerbetriebssysteme stellen zwei verschiedene Schnittstellen für Benutzer zur Verfügung, die sich durch Darstellung und die Art der Eingabe unterscheiden.

Moderne Systeme sind in der Regel mit einer *grafischen Benutzerschnittstelle* (GUI Graphical User Interface) ausgestattet, bieten aber zusätzlich die Möglichkeit, verschiedene Kommandos über eine sogenannte *Shell* einzugeben.

Früher war die Shell das wichtigste Programm auf einem Unix-System und auch heute ist sie das wichtigste Werkzeug für den Unix-/Linux-Administrator. Im Laufe der Zeit wurden mehrere Shell-Programme entwickelt. Da wäre die `sh` (Bourne Shell), die in älteren Unix-Systemen als Standard Shell eingesetzt wurde und einfache Funktionen beinhaltet. Die `ksh` (Korne Shell), die bereits wesentlich mehr Funktionalität als die Bourne Shell zur Verfügung stellt und immer noch in vielen Unix-Systemen als Standard eingesetzt wird. Für C-Programmierer gibt es die csh (C-Shell), die einen der Korne Shell ähnlichen Funktionsumfang bietet. Die Linux Standard-Shell ist die bash (Bourne Again Shell). Sie verbindet die Einfachheit der Bourne Shell mit der Vielseitigkeit der Korne Shell und wird heute nicht nur unter Linux gerne verwendet. Im folgenden wird ausschließlich mit der bash gearbeitet.

Genau genommen ist eine Shell lediglich ein Benutzerprogramm. Sie stellt die Schnittstelle zwischen dem Benutzer und dem System dar. Sie umfaßt einen Kommandointerpreter, der Benutzereingaben analysiert und die dem Kommando zugeordneten Programme startet und Ausgaben des Kernels an den Benutzer zurückgibt. Sie verfügt aber auch über eine eigene Programmiersprache mit Sonderzeichen, Variablen und Funktionen, die es ermöglicht, sog. Shell-Skripte zu erstellen, um z. B. bestimmte administrative Aufgaben zu erleichtern. Im folgenden werden die syntaktischen Regeln und Möglichkeiten der bash ins Visier genommen.

2.2 Die Shell nutzen

2.2.1 Nach der Anmeldung

Nach der erfolgreichen Anmeldung wird für jeden Benutzer eine eigene Shell (Kommandointerpreter) geladen. Während die Shell startet, laufen noch Initialisierungsskripte für spezielle Benutzereinstellungen ab. Als Kennzeichen, dass die Shell für den Empfang von Kommandos bereit ist, sieht man einen sog. Prompt (Eingabeaufforderung) am linken Bildschirmrand, falls man sich im Textmodus angemeldet hat. Hat man sich unter einer grafische Benutzeroberfläche angemeldet, kann mit einem Terminalfenster eine Shell gestartet werden, bei dem der Prompt ebenfalls am linken Rand zu sehen ist.

Dieser Prompt kann individuell für jeden Benutzer eingestellt werden. Als Standard ist für den normalen Benutzer

```
Benutzername@Rechnername:aktuelles Verzeichnis >
```

festgelegt. Für `root` ist der Standard-Prompt

```
Rechnername:Verzeichnis #
```

Kapitel 3

Prozesse

3.1 Einführung

Ein *Prozess* ist ein Programm, das gerade ausgeführt wird. Ein Programm liegt meist in Form einer Datei auf dem Datenträger (Festplatte) vor und beschreibt eine Folge von Aktionen, die sequentiell durchgeführt werden. Erst durch das Laden des Programms in den Hauptspeicher und dessen Ausführung entsteht ein Prozess. In modernen Betriebssystemen werden immer mehrere Prozesse gleichzeitig abgearbeitet. Allerdings kann ein Prozessor nur immer einen Befehl zu einer bestimmten Zeit ausführen. Somit werden Prozesse in Ein-Prozessor-Systemen nie wirklich parallel ausgeführt. Die Zeit, die zwischen dem Umschalten von einem Prozess zum anderen vergeht, ist jedoch so gering, dass man von Pseudo-Parallelität spricht. Die Aufgabe, zu entscheiden welcher Prozess gerade Prozessorzeit erhält, übernimmt der sog. *Scheduler*, der nach folgenden Kriterien arbeitet:

- Fairness
  Jeder Prozess erhält einen gerechten Anteil an Prozessorzeit.
- Effizienz
  Der Prozessor ist immer vollständig ausgelastet.
- Kurze Antwortzeit
  Die Antwortzeit für die arbeitenden Benutzer wird minimiert.
- Großer Durchsatz
  Die Anzahl der ausgeführten Aufträge in einem bestimmten Zeitintervall wird maximiert.

Das Betriebssystem muss zu jeder Zeit wissen, welche Prozesse gerade im Speicher sind, wie der aktuelle Zustand ist, an welcher Stelle der Prozess unterbrochen wurde und welche Speicherbereiche dem Prozess zugeordnet sind. Damit

wird der Prozessverwaltung und der damit verbundenen Ressourcenverwaltung eine wichtige Rolle zugeteilt. In verschiedenen Tabellen merkt sich das Betriebssystem alle Informationen, die zu einem bestimmten Prozess gehören. Einige dieser Informationen sind:

- Register, Inhalt aller Prozessorregister
  Beim Umschalten zwischen den Prozessen müssen die Registerinhalte des Prozessors wieder hergestellt werden können.
- Befehls- bzw. Programmzähler
  Erhält ein Prozess wieder Rechenzeit, muss genau an der Stelle weitergearbeitet werden, wo er aufgehört hatte.
- Stack-Zeiger
  Jeder Prozess hat eine Ablage, wo Daten zwischengespeichert werden. Hier wird gespeichert, wo die oberste Stelle der Ablage ist.
- Prozesszustand
  Zu einer bestimmten Zeit befindet sich ein Prozess in einem bestimmten Zustand. Zustände sind:
  - Aktiv
    Der Prozess wird gerade abgearbeitet.
  - Wartend
    Der Prozess wartet auf das Ende einer Wartebedingung.
  - Bereit
    Der Prozess wartet auf die Zuteilung eines Prozesses.
  - Nicht existent
    Zustand eines Prozesses, der nicht mehr oder noch nicht im System existiert.

  Diese Zustände sind allgemeine Prozesszustände. Mit dem Kommando `ps` kann man sich in detaillierter Form anzeigen lassen, wie der aktuelle Zustand eines Prozesses ist.
- Prozessnummer
  Jedem Prozess ist eine eindeutige Nummer zugeordnet.
- Verbrauchte Prozessorzeit
  Die Zeit, die ein Prozess bereits an Rechenzeit hatte.
- Auftraggeber
  Benutzer-Nummer, des Benutzers, der den Prozess gestartet hatte.
- Ablaufpriorität
  Priorität des Prozesses, die angibt, wieviel Rechenzeit ein Prozess erhält.
- Zugriffsrechte
  Zugriffsrechte, die der Prozess hat.
- Zeiger auf Speichersegmente
  Speicherbereiche, die dem Prozess zugeordnet sind.

- Signalstatus
  Signale, die der Prozess empfängt.
- Elternprozess
  Gibt die Nummer des Elternprozesses an, der diesen Prozess erzeugt hat.
- Reale und effektive Benutzernummer
  Reale und effektive Benutzernummer, mit der dieser Prozess arbeitet.
- Reale und effektive Gruppennummer
  Reale und effektive Gruppennummer, mit der dieser Prozess arbeitet.
- Dateideskriptoren
  Referenzen auf die Dateien, die von diesem Prozess benutzt werden.

Für den Benutzer stehen verschieden Kommandos zur Verfügung, um sich die oben aufgeführten Informationen anzeigen zu lassen, mit Prozessen zu kommunizieren und Prozesse zu erzeugen. In den nachfolgenden Kapiteln werden einige dieser Kommandos vorgestellt.

3.2 Prozesshierarchie und besondere Prozesse

In Linux-Systemen wird beim Laden des Betriebssystems der *Init-Prozess* gestartet. Aufbauend auf diesen Prozess wird eine Hierarchie erstellt. Jeder Prozess kann Elternprozess eines Kindprozesses sein. Ganz oben in der Hierarchie steht jedoch immer der Init-Prozess mit der Prozessnummer 1. Diese Hierarchie ist mit einem Stammbaum zu vergleichen, der sich nach unten immer weiter gabelt. Ein Elternprozess, der Kindprozesse erzeugt, gabelt sich. Vom Betriebssystem wird zur Erzeugung eines Kindprozesses der Befehl `fork` (engl. Gabel) verwendet. Für den Benutzer heißt das, wenn er ein Kommando an der Shell eingibt, entsteht ein neuer Prozess. Dieser neue Prozess ist das Kind der Shell, in der er das Kommando eingegeben hat. Mit dem Kommando `pstree` können Sie sich die Prozesshierarchie ansehen. Der Init-Prozess (`init`) steht bei der Ausgabe immer an der linken Seite ganz oben. Wenn Sie der Prozesshierarchie folgen, sehen Sie auch das eingegebene Kommando `pstree` als Kindprozess der bash.

3.2.1 Der Init-Prozess

Nachdem der Linux-Kernel gestartet ist, überprüft dieser die Gerätekonfiguration und erzeugt den SWAP-Prozess, der für das Scheduling verantwortlich ist, sowie den Init-Prozess, der in unserer Hierarchie die oberste Stelle einnimmt. Gesteuert wird dieser Prozess durch die Datei `/etc/inittab`. In `/etc/inittab` befinden sich Informationen zum Start, d. h. welche Startskripte ausgeführt werden und in welchem Run-Level das System startet. Das Programm des Init-Prozesses befindet sich im Verzeichnis `/sbin`. Init startet das

```
[1]+  8472 Running                 sleep 100 &
user@linux:/home/user >
[1]+  Done                    sleep 100
```

Um einen Job wieder in den Vordergrund zu holen oder nachträglich einen Prozess in den Hintergrund zu stellen, werden die Kommandos `fg` *`jobnummer`* (foreground) und `bg` *`jobnummer`* (background) verwendet. Bevor ein Prozess in den Hintergrund gestellt werden kann, muss er mit dem Signal SIGTSTP durch die Tastenkombination (Strg)(Z) gestoppt werden.

```
user@linux:/home/user > sleep 100 &
[1] 8562
user@linux:/home/user > fg 1
sleep 100
(Strg)(z)

[1]+  Stopped                 sleep 100
user@linux:/home/user > bg 1
[1]+ sleep 100 &
user@linux:/home/user > jobs
[1]+  Running                 sleep 100 &
```

3.6 Übersicht: Prozesse und Jobs steuern

Tabelle 3.2: Prozesse und Jobs steuern

Kommando	Kurzbeschreibung	Details
`at`	Jobs zur einmaligen Ausführung zu einem bestimmten Zeitpunkt in eine Warteschlange stellen	Seite 119
`atq`	Die `at`-Jobs in einer Warteschlange anzeigen lassen	Seite 119
`atrm`	Die `at`-Jobs aus einer Warteschlange löschen	Seite 119
`batch`	Jobs zu einem späteren Zeitpunkt (in Abhängigkeit von der Systemauslastung) ausführen lassen	Seite 119
`bg`	Angehaltene Programme im Hintergrund fortsetzen	Seite 121
`crontab`	periodische Aufträge, die durch den cron-Daemon gestartet werden, verwalten	Seite 139
`fg`	Angehaltene Programme im Vordergrund fortsetzen	Seite 165
`fuser`	Ausgabe der PID, die angegebene Dateien bzw. Dateisysteme verwenden	Seite 173

Tabelle 3.2 – Fortsetzung

Kommando	Kurzbeschreibung	Details
`jobs`	Aktuelle Hintergrundprozesse der Shell anzeigen	Seite 194
`kill`	Signale an Prozesse anhand der Prozessnummer senden	Seite 197
`killall`	Mit Prozessen anhand des Namens Signale senden	Seite 197
`ldd`	Anzeige der dynamischen Abhängigkeiten eines Programms	Seite 200
`nice`	Kommando mit einer bestimmten Priorität (einem bestimmten Nice-Wert) ausführen	Seite 228
`nohup`	Ignorieren der Signale SIGINT, SIGQUIT und SIGHUP	Seite 231
`ps`	Prozess-Status anzeigen	Seite 238
`pstree`	Aktive Prozesse in Baumstruktur anzeigen	Seite 242
`renice`	Priorität eines bereits laufenden Prozesses manipulieren	Seite 249
`time`	Dauer von Kommandoausführung messen und anzeigen lassen	Seite 276
`times`	Benutzer- und Systemzeiten für Prozesse der aktiven Shell anzeigen lassen	Seite 277
`top`	CPU-intensive Prozesse ausgeben	Seite 277
`trap`	Signale abfangen	Seite 284

Kapitel 4

Dateisystem

4.1 Die Linux-Verzeichnishierarchie

Mit der Installation ihres Linux-Systems werden bereits viele Programme und Konfigurationsdateien auf Ihren Computer kopiert. Alle diese Dateien sind hierarchisch in Verzeichnissen geordnet. Nachfolgende Tabelle 4.1 enthält die wichtigsten Verzeichnisse, die nach einer Installation vorhanden sind. Die Schrägstriche (/) bei den Verzeichnisnamen haben eine besonderer Bedeutung. Befindet sich der Schrägstrich am Anfang eines Verzeichnisnamens zeigt das an, dass das Verzeichnis direkt unterhalb des Hauptverzeichnisses (root) ist. Trennt der Schrägstrich Verzeichnisnamen wie in `/usr/bin` zeigt das an, dass sich das Verzeichnis mit dem Namen `bin` in der Hierarchie unter dem Verzeichnis mit dem Namen `usr` befindet.

Als Benutzer werden Sie hauptsächlich in ihrem Heimat-Verzeichnis (auch Home-Verzeichnis genannt) `/home/<Anmeldename>` arbeiten und dort ihre eigene Verzeichnishierarchie aufbauen, ihre Dateien speichern. Alle anderen Verzeichnisse enthalten Dateien, die vom System benötigt werden.

Tabelle 4.1: Die Linux-Verzeichnishierarchie

Verzeichnis	Inhalt
`/`	Hauptverzeichnis (sog. *root*) für alle Benutzer, oberste Ebene der Hierarchie.
`/boot`	Enthält die Start-Dateien (Kernel) des Systems.
`/bin`	(binary) Verzeichnis für häufig benutzte Programme.
`/dev`	(device) Verzeichnis für Geräte-Dateien.

Tabelle 4.1 – Fortsetzung

Verzeichnis	Inhalt
`/etc`	Verzeichnis für Dateien zur Administration und Konfiguration des Systems.
`/home`	Enthält die Benutzerverzeichnisse (Heimat-Verzeichnisses), der dem System bekannten Benutzer.
`/lib`	(library) Verzeichnis für Programmbibliotheken
`/media`	Enthält Verzeichnisse, die mit bestimmten Datenträgern, wie Diskettelaufwerk und CD, verbunden sind.
`/opt`	(option) Verzeichnis für Programmepakete, z. B. KDE, Gnome, Open-Office, die optional installiert wurden.
`/proc`	Eigentlich kein wirkliches Verzeichnis, enthält Laufzeit-Informationen zum System und zum Speicher.
`/tmp`	(temporary) Platz für temporäre Dateien, die vom System, verschieden Programmen oder Benutzern erstellt werden.
`/usr`	(unix special resource) Platz für Verzeichnisse, die verschiedene Programme, Bibliotheken und Dokumentationen enthalten.
`/usr/bin`	Programme für alle Benutzer.
`/usr/include`	Include-Dateien für die C / C++ Programmierung.
`/usr/lib`	Bibliotheken für verschiedene Programmiersprachen (je nach Installation).
`/usr/share`	Verzeichnis für gemeinsam genutzte Dateien und Dokumentationen.
`/usr/share/man`	(manual) Enthält die Manual-Pages, die mit `man` aufgerufen werden.
`/usr/sbin`	(special binary) Enthält Programme zur Administration des Systems, die nicht für alle Benutzer verwendbar sind.
`/usr/src`	(source) Quellcode des Systems, falls dieser installiert wurde.
`/sbin`	(special binary) Programme zur Administration, die nur für bestimmte Benutzer (root oder Administratoren) zur Verfügung stehen.

Tabelle 4.1 – Fortsetzung

Verzeichnis	Inhalt
`/var`	(variable) Verzeichnis für variable Daten wie Drucker-Warteschlangen, Log-Dateien, Mails und Prozessnummern verschiedener Dienste.

4.2 Dateiarten

In Linux- und Unix-Systemen gibt es verschiedene Arten von Dateien. Im Grunde wird alles als Datei bezeichnet, von dem aus Daten gelesen oder wohin Daten geschrieben werden können. Beispielsweise werden Netzwerkverbindungen (Stream), Terminals, Pipes, die Tastatur und der Bildschirm aus Sicht des Betriebssystems als Dateien betrachtet. Die Tabelle 4.2 gibt Auskunft über die Dateiarten, die Linux benutzt.

Tabelle 4.2: Dateiarten unter Linux

Art	Verwendung
Reguläre Datei	Dies ist der gebräuchlichste Dateityp. Dies sind Programmdateien, Datenbankdateien, Bilddateien, Videodateien, Klangdateien, normale Textdateien und alle Dateien, die ein Benutzer mit Büroanwendungen erstellt.
Verzeichnisse	Dies sind Dateien, die zur Verwaltung dienen (Ordner) und können beliebige Dateien enthalten.
Zeichenorientierte Geräte	Dies sind Spezialdateien und repräsentieren serielle Geräte wie Terminals, Drucker oder das Netzwerk. Sie erlauben eine eigene Pufferung durch den zugeordneten Treiber.
Blockorientierte Geräte	Spezialdateien für Geräte wie Harddisklaufwerke, die dem Betriebssystem die Pufferung überlassen.

Tabelle 4.3 – Fortsetzung

Kommando	Kurzbeschreibung	Details
`updatedb`	Aktualisieren der locate-Datenbank	Seite 294

4.4 Datei-Berechtigungen steuern

In Mehrbenutzersystemen können mehrere Anwender auf das Dateisystem zugreifen. Aus diesem Grund werden Berechtigungen vergeben, die regeln, welche Dateien von welchen Benutzern gelesen, bearbeitet oder ausgeführt werden dürfen. Linux- und Unix-Systeme kennen drei Rechte:

- das *Leserecht* (`r`) (engl. ***r**ead*),
- das *Schreibrecht* (`w`) (engl. ***w**rite*) und
- das *Ausführrecht* (`x`) (engl. *e**x**ecute*).

Die Vergabe dieser Rechte erfolgt jeweils für den

- *Eigentümer* (`u`) (engl. ***u**ser*),
- einer *Gruppe* (`g`) (engl. ***g**roup*), der die Datei zugeordnet ist, und
- die *anderen* (`o`) (engl. ***o**thers*), die nicht Eigentümer oder Mitglied der Gruppe sind.

Zusätzlich können jeder Datei noch Funktionen zugeteilt werden, die sich auf das Verhalten von ausführbaren Dateien, auf die Weitergabe von Rechten innerhalb von Verzeichnissen und auf die Nutzung des Speichers auswirken. Details finden sich in den Befehlsbeschreibungen zu `chattr` auf Seite 126 und `lsattr` auf Seite 210.

All diese Rechte und Sonderfunktionen werden in der I-Node-Tabelle einer Datei gespeichert. Hiefür stehen 12 Bit zur Verfügung. Ist ein Recht vergeben, ist das jeweilige Bit wie ein Schalter auf 1 (an) gesetzt.

Die Tabelle 4.4 beschreibt die einzelnen Rechte und Funktionen.

Tabelle 4.4: Rechte und Funktionen

Kürzel	Beschreibung
`r`	*read (Lese-Recht)* Dieses Bit ermöglicht das Lesen einer Datei. Für Verzeichnisse bewirkt es, dass der Inhalt des Verzeichnisses mit `ls` gelesen werden kann.
`w`	*write (Schreib-Recht)* Dieses Bit ermöglicht das Schreiben in eine Datei. Bei Verzeichnissen ist das Umbenennen, Erstellen oder Löschen von Dateien nur über die Kombination aus den Rechten `w` und `x` möglich.

Tabelle 4.4 – Fortsetzung

Kürzel	Beschreibung
x	*execute (Ausführ-Recht)* Dieses Bit ermöglicht das Ausführen einer Datei. Allerdings muss auch das Lese-Recht (`r`) vorhanden sein, um die Datei auszuführen. Nur wenn dieses Recht für ein Verzeichnis vergeben ist, kann in das Verzeichnis mit `cd` gewechselt werden. Außerdem können Dateien und Unterverzeichnisse in einem Verzeichnis nur angelegt oder gelöscht werden, wenn außer dem Schreib-Recht auch dieses Recht vorhanden ist.
s	*SUID-Bit (set user ID)* Ist diese Bit für eine ausführbare Datei gesetzt, wird der Prozess nicht mit der Benutzer-ID des aufrufenden Benutzers, sondern mit der Benutzer-ID des Datei-Eigentümers gestartet. Beispielsweise ist das Programm `/bin/passwd` mit diesem Bit ausgestattet. Erst dadurch ist es möglich, dass jeder Benutzer sein eigenes Passwort ändern kann, denn nur root hat die Rechte, in die Dateien zur Benutzerverwaltung, `/etc/passwd` und `/etc/shadow`, zu schreiben. Es versteht sich von selbst, dass diese Funktion nur mit äußerster Vorsichtig eingesetzt werden soll.
g	*SGID-Bit (set group ID)* Für Dateien ist die Funktionsweise dieselbe wie bei SUID, nur auf Gruppen bezogen. Ist es für ein Verzeichnis gesetzt, werden alle in diesem Verzeichnis angelegten Dateien nicht der Gruppe des Ersteller (owner) zugeordnet, sondern erhalten die gleiche Gruppenzugehörigkeit wie das Verzeichnis selbst. Interessanterweise wirkt sich das SGID-Bit auch die Unterverzeichnisse aus. Wird ein Unterverzeichnis in einem Verzeichnis mit SGID-Bit erstellt, so wird bei diesem das SGID-Bit automatisch gesetzt.
t	*Sticky-Bit* Der ursprüngliche Einsatzbereich dieses Bits lag darin, die Ladezeit von Programmen zu verringern. Ein Programm mit gesetzten Sticky-Bit (Klebe-Bit) wurde nach dem ersten Start im Speicher gehalten, um es bei erneutem Starten nicht mehr vom Datenträger holen zu müssen. Auch heute ist diese Funktionsweise noch vorhanden, wird allerdings wegen der inzwischen performanteren Hardware kaum noch eingesetzt. Wird dieses Bit auf ein Verzeichnis angewendet, so kann ein Benutzer nur die Dateien in diesem Verzeichnis löschen, deren Eigentümer er ist. Ausgenommen ist der Benutzer root, der alle Dateien löschen kann. Dieses Verhalten wird insbesondere bei Verzeichnissen für temporäre Dateien `/tmp` genutzt. Jeder Benutzer darf in dieses Verzeichnis Dateien stellen, allerdings darf sie nur der Eigentümer der Datei umbenennen oder löschen.

Folgende Darstellung zeigt die 12 Rechte-Bits und ihre Zuordnung:

Zuordnung	Funktion			Eigentümer			Gruppe			Andere		
Kürzel	s	g	t	r	w	x	r	w	x	r	w	x
Oktalwert	4	2	1	4	2	1	4	2	1	4	2	1
Bitposition	12	11	10	9	8	7	6	5	4	3	2	1

Alle Rechte, Funktionen und die Zuordnung einer Datei zum Eigentümer und zur Gruppe dürfen vom Eigentümer der Datei und vom Benuzter root verändert werden. Bei der Vergabe von Rechten wird in Linux oft auf die oktale Schreibweise zurückgegriffen. Dies ergibt sich aus der Konstellation, dass jeweils 3 Rechte oder Funktionen zur Verfügung stehen. Zur Darstellung der größten Ziffer des oktalen Zahlensystems (7) benötigt man auch 3 Bit (1 + 2 + 4 = 7).

Hierzu ein paar Beispiele:

Die Rechte `---rwxrw-r--` bedeuten Oktal `0754`:

	Funktion			Eigentümer			Gruppe			Andere		
Kürzel	-	-	-	r	w	x	r	-	x	r	-	-
Bits	0	0	0	1	1	1	1	0	1	1	0	0
Oktalwert	-	-	-	4	2	1	4	-	1	4	-	-
addiert		0			7			5			4	

Für eine Datei bedeutet diese Rechte-Konstellation, dass der Eigentümer alles darf, die Gruppe nur lesen und ausführen, alle anderen nur lesen dürfen. Für ein Verzeichnis bedeuten diese Rechte, dass der Eigentümer alles, die Gruppe nur das Verzeichnis mit `ls` lesen (`r`) und mit `cd` in das Verzeichnis wecheln (`x`) darf. Für alle anderen bedeuten diese Rechte, dass sie zwar das Verzeichnis lesen dürfen jedoch nicht in das Verzeichnis wechseln.

Die Rechte `---rw-rw----` bedeuten oktal `0640`:

	Funktion			Eigentümer			Gruppe			Andere		
Kürzel	-	-	-	r	w	-	r	-	-	-	-	-
Bits	0	0	0	1	1	0	1	0	0	0	0	0
Oktalwert	-	-	-	4	2	-	4	-	-	-	-	-
addiert		0			6			4			0	

Für eine Datei bedeuten diese Rechte, dass der Eigentümer nur lesen und schreiben darf (allerdings darf der Eigentümer die Rechte seiner Dateien jederzeit verändern, er darf ihnen also selbst fehlende Rechte geben). Die Gruppe darf die Datei nur lesen. Alle anderen sehen zwar die Datei mit `ls`, wenn die entprechenden Verzeichnisrechte vergeben wurden, können aber weder lesend noch schreibend auf diese Datei zugreifen. Auf ein Verzeichnis bezogen bedeuten diese Rechte, dass der Eigentümer das Verzeichnis lesen darf, allerdings nicht in das Verzeichnis wechseln darf (fehlendes `x`). Das Schreibrecht ist demzufolge überflüssig, da nur die Kombination von Schreib- und Ausführ-Recht das Schreiben, Löschen

und Erstellen innerhalb des Verzeichnisses erlauben. Die Gruppe und alle anderen haben hier somit effektiv die gleichen Rechte wie der Eigentümer.

Das Rechte `-gtrwxrwxr-x` bedeuten oktal `3775`:

	Funktion			Eigentümer			Gruppe			Andere		
Kürzel	-	g	t	r	w	x	r	w	x	r	-	x
Bits	0	1	1	1	1	1	1	1	1	1	0	1
Oktalwert	-	2	1	4	2	1	4	2	1	4	-	1
addiert		3			7			7			5	

Das letzte Beispiel bedeutet für eine Datei, dass der Eigentümer und die Gruppenmitglieder alles dürfen. Alle anderen dürfen die Datei nur lesen und ausführen. Die Datei wird nach dem ersten Starten im Speicher gehalten (`t`) und muss nicht neu vom Datenträger geladen werden. Diese Datei wird mit den Rechten der ihr zugeordneten Gruppe ausgeführt (`g`) und nicht mit den Rechten der Gruppe der ein ausführender Benutzer angehört.

Für ein Verzeichnis bedeuten diese Rechte, dass alle Dateien, die im Verzeichnis erstellt werden, die Gruppenzugehörigkeit des Verzeichnisses erhalten, nicht die des erstellenden Benutzers (`g`). Außerdem darf ein Benutzer nur die Dateien löschen und umbenennen, die er selbst erstellt hat (`t`). Alle dürfen in das Verzeichnis wechseln (`x`) und den Inhalt des Verzeichnisses mit `ls` ansehen (`r`). Nur der Eigentümer und die Gruppe dürfen Dateien im Verzeichnis erstellen (`w`).

4.4.1 Rechte verändern mit `chmod`

Der Eigentümer eine Datei und der Benutzer root können die Rechte mit dem Kommando `chmod` verändern.

`chmod [`*`Optionen`*`] `*`Rechte Dateien`*

Das Kommando `chmod` erlaubt die Angabe von Rechten in der oktalen Schreibweise und mit Kürzeln. Die oktale Schreibweise bietet den Vorteil, dass die ursprünglichen Rechte durch die neuen Rechte überschrieben werden, ist also immer eindeutig.

Folgende Beispiele zeigen die oktale Rechtevergabe mit `chmod`:

```
chmod 4754 d
```

Setzt für die Datei `d` die Rechte `s--rwxr-xr--`, d. h. es wird das SGID-Bit gesetzt, für den Eigentümer werden alle Rechte gesetzt, für die Gruppe werden die Rechte Lesen und Ausführen gesetzt, für die anderen wird nur das Lese-Recht gesetzt.

```
chmod 644 d
```

Setzt für die Datei `d` die Rechte `---rw-r--r--`, d. h. für den Eigentümer werden die Rechte lesen und ausführen gesetzt, für die Gruppe und alle anderen wird nur das Recht lesen gesetzt.

Will man nur bestimmte Rechte entziehen oder vergeben, ohne andere Rechte zu

Kapitel 5

Systemverwaltung

Im diesem Kapitel werden wichtige Linux-Befehle zur Syystemmverwaltung mit Optionen und kleinen Beispielen beschrieben. Zunächst erfolgt eine Übersicht der Befehle nach Themen. Da eine genaue Zuordnung eines Befehls nicht immer machbar ist tauchen manche Kommandos in mehreren der folgenden Tabellen auf. Danach werden kurz ein paar Erläuterungen zur syntaktischen Darstellung in den Befehls-Beschreibungen gegeben.

5.1 Laufzeitumgebung und Sitzungsinformationen verwalten

5.1.1 Einführung

Die Shell ist über Variablen konfigurierbar. Eine Variable ist ein kurzer Name für einen Wert. Ein Wert ist eine Zahl, ein Audruck, ein Text oder ein Kommando. Variablen sind flüchtig, d.h. bei jedem Neustart der Shell werden die Variablen neu initialisiert. Veränderungen oder neu erstellte Variablen gehen also verloren, wenn sie nicht permanent in einer Datei gespeichert werden. Meist erfolgt das Speichern von Variablen in den dafür vorgesehenen Startskripten. Diese Startskripte setzen die Variablen für das System, die Benutzerumgebung und für Programme. Unterschieden werden sog. Shell-Variablen und Umgebungsvariablen (Environment-Variable).

5.1.1.1 Shell-Variablen

Variablen, die nur innerhalb der aktuellen Shell gültig sind, werden als Shell-Variable bezeichnet. Erstellt wird eine Shell-Variable, indem man einem Variablennamen durch das Gleichheitszeichen (=), einen Wert zuweist. Das Komman-

do `summe=0` weist der Variablen *summe* den Wert *0* zu. Bei der Zuweisung darf kein Leerzeichen zwischen dem Variablennamen und dem Wert stehen. Enthält der Wert Leerzeichen, muss er mit Hilfe von Quoting-Zeichen zugewiesen werden (`text='a b c d'`). Der Zugriff auf den Inhalt einer Variablen erfolgt durch das $-Zeichen, das dem Variablennamen vorangestellt wird.

Um den Inhalt der Variablen *summe* auszugeben oder zu verwenden, gibt man demnach `$summe` ein. Die Variable hinter dem $-Zeichen wird durch Ihren Wert ersetzt (Variablensubstitution). Die folgenden Beispiele verdeutlichen den Umgang mit Variablen.

Zunächst werden den Variablen *vorname* und *nachname* die Werte *Felix* und *Krull* zugewiesen und anschließend mit dem Kommando `echo` ausgegeben. Es findet eine Ersetzung der Variablen durch die Werte statt:

```
user@linux:/home/user > vorname=Felix
user@linux:/home/user > nachname=Krull
user@linux:/home/user > echo $vorname $nachname
Felix Krull
```

Fehlt das $-Zeichen, wird nur der Variablenname ausgegeben:

```
vorname
```

Gibt man nur `$vorname` ohne die Verwendung des Kommandos `echo` ein, wird die Variable ersetzt und die Shell versucht, den Wert zu interpretieren. Die Kommandos `wer=who` und `echo $wer` verdeutlichen dieses Verhalten.

```
user@linux:/home/user > $vorname
bash: Felix: command not found
user@linux:/home/user > wer=who
user@linux:/home/user > $wer
user   :0           Jul 28 07:29 (console)
```

Die Variablen *vorname* und *nachname* werden im folgenden einer Variablen *name* zugewiesen. Da zwischen den beiden substituierten Werten ein Leerzeichen stehen soll, müssen die Variablen zusammen mit dem Leerzeichen in Quoting-Zeichen eingeschlossen werden. Verwendet man anstelle der doppelten Anführungszeichen (`" "`) nur einfache Anführungszeichen (`' '`)[1], verliert das $-Zeichen seine besondere Bedeutung und die Variablen werden nicht durch ihre Werte ersetzt:

```
user@linux:/home/user > vorname=Felix
user@linux:/home/user > nachname=Krull
user@linux:/home/user > name="$vorname $nachname"
user@linux:/home/user > echo $name
```

[1] bei vielen Tastaturen das Zeichen, das man mit Shift + # eingeben kann

```
Felix Krull
user@linux:/home/user > name='$vorname $nachname'
user@linux:/home/user > echo $name
$vorname $nachname
```

Wird der Wert einer Variablen als Bestandteil einer Zeichenfolge benötigt, so muss der Variablenname in geschweifte Klammern gesetzt werden, da sonst kein Trennzeichen (Leerzeichen) zwischen Variablenname und Text vorhanden ist und die Shell beide zusammen als einen Variablennamen sieht.

```
user@linux:/home/user > verz1=/home/user/
user@linux:/home/user > echo ${verz1}privat
/home/user/privat
```

Wenn Sie das Ergebnis eines Kommandos einer Variablen zuweisen wollen, müssen Sie die Quoting-Zeichen für Kommandosubstitution verwenden. Der Variablen *`datum`* wird das Ergebnis des Kommandos `date` zugewiesen und nicht das Kommando `date`. Während sich die Uhrzeit bei der Eingabe von `date` verändert, bleibt der Inhalt der Variablen *`datum`* unverändert:

```
user@linux:/home/user > datum=‘date‘
user@linux:/home/user > echo $datum
Mi Jul 28 11:05:14 CEST 2008
user@linux:/home/user > date
Mi Jul 28 11:13:22 CEST 2008
user@linux:/home/user > echo $datum
Mi Jul 28 11:05:14 CEST 2008
```

Mit dem Kommando `set` wird eine Liste aller definierten Variablen auf der Konsole angezeigt. Löschen kann man eine Variable mit dem Kommando `unset`.

```
user@linux:/home/user > vorname=Felix
user@linux:/home/user > echo $vorname
Felix
user@linux:/home/user > unset vorname
user@linux:/home/user > echo $vorname

user@linux:/home/user >
```

5.1.1.2 Umgebungsvariablen

Mit *Umgebungsvariablen* oder Environment-Variablen wird die Umgebung der Shell gesteuert. Diese Variablen werden in Startdateien, die beim Starten der bash ausgeführt werden, mit Werten initialisiert. Zur besseren Übersicht sind die Variablennamen meist in Großbuchstaben geschrieben. Eine Umgebungsvariable zeichnet sich dadurch aus, dass sie an Kind-Prozesse weitergegeben wird. Mit

Kapitel 6

Befehlsübersicht a-z

6.1 Hinweise zu Befehlsoptionen

Allgemein haben Linux--Befehle das Format:

```
Kommando [Option[Arugment]][...]
```

Optionen beginnen meist mit einem vorangestelltem Minus-Zeichen (`-`) und können ggf. Argumente besitzen. Folgende Optionen gibt es bei vielen Befehlen.

Tabelle 6.1: Allgemeine Optionen

Option	Beschreibung
`-h`, `--help`	Mit dieser Option werden eine kurze Syntaxbeschreibung und die möglichen Optionen des Befehls ausgegeben
`-v`, `--verbose`	Mit dieser Option wird der Befehl meist veranlaßt, eine sog. wortreiche Ausgabe durchzuführe. Man kann auch sagen, der Befehl ist mit dieser Option „geschwätziger“.
`-r`, `-R`	steht für rekursiv und weist den Befehl an, auch Unterverzeichnisse in seine Aktivitäten einzubeziehen. Befehle, die mit Dateien und Verzeichnissen arbeiten, besitzen diese Option.

6.1.1 Erläuterungen zur syntaktischen Beschreibung

Die Überschrift zu jedem Befehl enthält den Namen und die Sektionsnummer der man-Pages. Jede Befehlsbeschreibung enthält folgende Bereiche:

- ❙ *Einsatz:*
  ob es sich um ein Anwender- , Administratoren-Kommando oder einen Dienst handelt
- ❙ *Anwendung:*
  eine kurze Erklärung, was der Befehl macht
- ❙ *Syntax:*
  eine syntaktische Beschreibung des Befehls
- *Tabelle mit Optionen:*
  eine Beschreibung der Optionen, sofern vorhanden
- ❙ *Beispiel:*
  ein oder mehrer Beispiele zur Anwendung des Befehls
- ❙ *Besonderheiten:*
  beschreibt Besonderheiten des Befehls und Hinweise zur Ausführung (nicht bei jedem Befehl vorhanden)
- *Ausgabebeschreibung:*
  beschreibt die Ausgabe des Befehls (nicht bei jedem Befehl vorhanden)
- *Dateien:*
  beschreibt Dateien die der Befehl nutzt oder verändert (nicht bei jedem Befehl vorhanden)
- ❙ *Siehe auch:*
  enthält eine Liste von Kommandos, die im gleichen Kontext verwendet werden.

Folgende Tabelle 6.2 beschreibt die im weitern verwendeten Notationen bei der syntaktischen Beschreibung eines Kommandos.

Tabelle 6.2: Eingesetzte Notation

Notation	Bedeutung
`[A]`	Angaben in eckigen Klammern `[ ]` sind optional (Die Klammern sind nicht einzugeben).
`[A[B]]`	`A` und `B` sind optional. A kann ohne B angegeben werden oder beide zusammen. B alleine ist jedoch nicht erlaubt.
`[...]`	Mehrere Angaben des vorangehenden Typs können optional wiederholt werden.
`A | B [A | B]`	Bei Angaben, die durch einen Strich `|` (ODER) getrennt sind, darf wahlweise nur eine Angabe gemacht werden. (ODER-Zeichen sind nicht einzugeben)

Tabelle 6.2 – Fortsetzung

Notation	Bedeutung
`ABC +x`	Begriffe, Zeichen und Buchstaben in **Fettschrift** sind Schlüsselwörter und exakt wie dargestellt einzugeben.
`Wort N`	Begriffe und Buchstaben in *Kursivschrift* sind durch die jeweils möglichen Angaben zu ersetzen.
(DEL) (Return)	Tastenbezeichnung in Versalien sind Sondertasten oder Tastenkombinationen, z. B. Delete-Taste, Enter-Taste.
(CTRL)+(V)	Tastenkombination (Taste (CTRL) gedrückt halten und Taste (V) betätigen).
(t)(Z)	Tastenfolge (zuerst Taste (t) betätigen, anschließend Taste (Z)).

Die Reihenfolge der Angaben ist in den meisten Fällen zwingend einzuhalten. Linux unterscheidet zwischen Groß- und Kleinschreibung. Die Begriffe Befehl und Kommando werden in dieser Referenz gleichwertig im Sinne von Betriebssystem-Kommando verwendet. Der Begriff Kommando wird auch für Steueranweisungen in interaktiven Programmen verwendet.

Bei den Beispielen unterscheidet sich der Prompt je nach dem, ob es sich um ein Kommando handelt das jeder Benutzer starten darf (`user@linux:~ >`) oder ob das Kommando vom Benutzer root auszuführen ist (`linux: #`).

Beispiel:

```
shutdown  -h|-r [Optionen]  [-t sec] time  [warning-message]
```

Mit dem Befehl `shutdown` muss entweder die Option `-h` (halt) **oder** `-r` (reboot) angegeben werden. Weitere Optionen sind optional. Die Option `-t` braucht jedoch die Angabe einer Sekundenanzahl (es ist die Anzahl der Sekunden, die zwischen dem Versenden des SIGTERM-Signals und dem darauffolgenden SIGKILL-Signals verstreichen soll). Wird diese Option samt Sekundenangabe weggelassen, so greift der voreingestellte Standardwert. Die Angabe *`time`* muss unbedingt gemacht werden, um den Zeitpunkt des eigentlichen Herunterfahrens des Systems zu spezifizieren. Die *`warning-message`* ist wiederum optional zu der ohnehin erfolgenden Ausgabe, dass das System heruntergefahren wird. Mit den Optionen können damit den Benutzern ergänzende Informationen geliefert werden. Ein tatsächlich abgesetzter Befehl kann wie folgt aussehen:

```
shutdown -h +10 Dringende Wartungsarbeiten
```

Das Herunterfahren des Systems soll in 10 Minuten durchgeführt werden. Als *warning-message* erscheint „Dringende Wartungsarbeiten“.

```
shutdown -rf now
```

Das Herunterfahren des Systems soll sofort (statt `now` kann auch `+0` angegeben werden) und als `reboot` durchgeführt werden (`-r`), wobei jedoch ein sog. *fast reboot* (`-f`) durchgeführt wird, d. h. beim Hochfahren des Systems wird die Überprüfung des Dateisystems ausgelassen. Es ist keine *warning-message* angegeben.

6.1.2 Linux-Befehle

`alias` (bash-builtin)

Einsatz: Anwender

Anwendung: `alias` ist eine interne Shell-Funktion (kein eigener Prozess). Abkürzungen für Befehle definieren. Die Shell gestattet es, für Befehle und deren Parameter Alias-Namen zu definieren. Vor allem bei häufig gebrauchten bzw. komplexen Befehlen ist dies sinnvoll. Mit Alias-Namen kann man nicht nur bestehende Kommandos verändern, man kann auch neue Kommandos definieren.

Syntax: `alias neuerBefehl="alter Befehl"`

Besonderheiten: Die Shell wertet zuerst die Alias-Namen aus. Wird der eingegebene Befehl nicht unter den Alias-Namen gefunden, so wird nach dem passenden Kommando im Dateisystem gesucht. Alias-Namen haben damit auch Vorrang vor gleichnamigen Befehlen. Wenn man ein Alias mit dem Befehl `alias` definiert, so gilt dieser Alias nur solange, bis die Shell beendet oder die Definition mit dem Befehl `unalias` wieder rückgängig gemacht wird. Soll ein Alias-Name für einen Benutzer dauerhaft definieren werden, so muss die Alias-Anweisung in eine der Benutzerstartdateien (`.profile`) im Heimatverzeichnis des Benutzers eingetragen werden. Wird die Alias-Anweisung in die Datei `.bashrc` im Heimatverzeichnis des Benutzers eingetragen, so gilt der Alias auch in jeder Sub-Shell der Login-Shell.

Beispiel:

```
user@linux:~ > alias rm="rm -i"
```

Mit dieser Eingabe wird festgelegt, dass beim Ausführen des Löschbefehls `rm` der Befehl `rm -i` ausgeführt wird. Es wird also vor dem Löschen erst noch einmal nachgefragt.

Siehe auch:: `unalias` auf Seite 291.

`apropos` (1)

Einsatz: Anwender

Anwendung: Erleichtert das Durchsuchen von man-pages zu einem bestimmten Thema. Das Kommando `apropos` durchsucht die Titel der man-pages und die in den man-pages entaltenen Kurzbeschreibungen nach dem angegebenen Stichwort. Wird eine Übereinstimmung gefunden, so wird der Name der man-Page, die Sektion und eine Kurzbeschreibung des Befehls ausgegeben. Standardmäßig wird der Suchbegriff als regulärer Ausdruck behandelt. Das Stichwort kann auch Wildcards enthalten (`-w`). Man kann die Suche auch auf das exakte Stichwort beschränken (`-e`).

Syntax: `apropos [-e|-w|-r] Stichwort`

Tabelle 6.3: Optionen zu `apropos`

Option	Beschreibung
`-r`, `--regex`	Jedes Stichwort wird als regulärer Ausdruck verstanden, was auch dem Standardverhalten entspricht.

```
user@linux:~ > cd ..
```

Das Verzeichnis `/etc` wird zum neuen Arbeitsverzeichnis:

```
user@linux:~ > cd /etc
```

Es wird ins vorherige Arbeitsverzeichnis zurückgewechselt:

```
user@linux:~ > cd -
```

cfdisk (8)

Einsatz: root

Anwendung: Partitionierung der Festplatte. Der Befehl `cfdisk` erfüllt die gleiche Funktion wie der Befehl `fdisk`, nur ist er übersichtlicher und komfortabler, da er über die Pfeil-Tasten und ein Menü bedient werden kann. Wird `cfdisk` aufgerufen, so liefert er zunächst eine Übersicht über die aktuelle Partitionierung der Festplatte (gefundenen Partitionen und jeweilige Größe). Mit den Pfeil-nach-oben/unten Tasten kann die Partition ausgewählt werden, mit den Pfeil-nach-rechts/links Tasten können aus dem vorhandenen Menü die Befehle ausgewählt werden.

Syntax: **`cfdisk [Optionen] [Gerät]`**

Tabelle 6.8: Optionen zu `cfdisk`

Option	Beschreibung
`-a`	Als Cursor wird ein Pfeil, anstelle einer invertierten Zeile dargestellt.
`-g`	Verwendet nicht die Plattengemoetrie des Festplatten-Treibers
`-z`	Startet das Programm mit einer leeren Partitionstabelle, Partitionen auf der Festplatte bleiben zunächst erhalten
`-c` *`cyl`*	überschreibt BIOS-Zylinder mit *`cyl`*
`-h` *`hd`*	überschreibt BIOS-Köpfe mit *`hd`*
`-s` *`sec`*	überschreibt BIOS-Sektoren pro track *`sec`*
`-P` *`opt`*	Gibt die Paritionstabelle im angegebenen Format *`opt`* aus. Formate sind: „`r`“ (raw-Daten-Format), „`s`“ (sector-order-Format),“`t`“ (partion-table-raw-format)

Besonderheiten: Sollte der Befehl `cfdisk` auf einem System nicht zur Verfügung stehen, so verwenden Sie das dialogorientierte nicht ganz so komfortable `fdisk`. Es versteht sich, dass dieses Kommando mit äußerster Vorsicht verwendet werden muss!

Beispiel:

```
linux:~ # cfdisk
```

Es wird die aktuelle Partitionierung von `/dev/hda` ausgegeben (Standard).

Siehe auch: `fdisk` auf Seite 164.

chage (1)

Einsatz: root

Anwendung: Einstellungen zu Benutzerpasswörtern vornehmen, (chage = change age). Dies betrifft vor allem Passwort-Lebensdauer und Passwort-Verfallsdatum. Sämtliche Zeiteinstellungen zu den Passwörtern werden in der Datei `/etc/shadow` gespeichert. Die Einstellungen in der Datei

/etc/shadow haben Vorrang vor den Einstellungen in der Datei /etc/passwd, d. h. ist ein Passwort tatsächlich abgelaufen, so schafft auch das Ausschalten der Passwortabfrage in der /etc/passwd (das „x“ im Passwort-Feld (2. Feld) entfernen) keine Abhilfe.

Syntax: **chage [-m *min_days*] [-M *max_days*] [-d *last_day*] [-I *inactive*] [-E *expire_date*] [-W *warn_days*] *Benutzer***
chage -l *Benutzer*

Tabelle 6.9: Optionen zu chage

Option	Beschreibung
-m *min_days*	Der als min_days angegebene Wert gibt die Mindestanzahl an Tagen an, die das Passwort nicht geändert werden darf. Setzt man hier den Wert 0 ein, so darf der Benutzer sein Paßwort jederzeit ändern. (4. Feld der Datei /etc/shadow).
-M *max_days*	Der mit max_days angegebene Wert bezeichnet die maximale Anzahl an Tagen die das Passwort, gerechnet ab der letzten Passwort-Änderung, gültig ist. Der Benutzer muss sein Passwort vor Ablauf dieser Frist ändern, will er das Ablaufen seines Passworts und das anschließende Sperren seines Kontos verhindern. (5. Feld der Datei /etc/shadow). Sinnvollerweise wird ein „Warn-Countdown“ (Option -W) und eine Gnadenfrist (Option -I) eingestellt.
-d *last_day*	Der Wert last_day bezeichnet das Datum der letzten Passwort-Änderung. Er kann entweder als Anzahl der Tage seit dem 1.1.1970 oder als Datum mit dem Format JJJJ-MM-TT angegeben werden (3. Feld der Datei /etc/shadow).
-E *expire_date*	Dies ist das absolute Verfallsdatum des Benutzerkontos. An dem hier angegebenen Datum wird das Konto unabhängig aller anderen Einstellungen gesperrt. Der Wert expire_date kann entweder als Anzahl der Tage seit dem 1.1.1970 oder als Datum im Format JJJJ-MM-TT angegeben werden (8. Feld der Datei /etc/shadow).
-I *inactive*	Der Wert inactive bezeichnet die Anzahl der Tage, die das Passwort nach Ablauf der maximalen Passwort-Gültigkeitsdauer immer noch gültig ist. Wird hier der Wert 0 eingesetzt, so kann kein regelmäßiges Ändern des Passworts durch den Benutzer erzwungen werden (7. Feld der Datei /etc/shadow).
-W *warn_days*	Der Wert warn_days bezeichnet die Anzahl der Tage, ab denen der ”Warn-Countdown” läuft. Beim Einloggen erhält der Benutzer ab diesem Tag bei jeder Anmeldung einen Hinweis, in wie vielen Tagen sein Passwort abläuft (6. Feld der Datei /etc/shadow).
-l *Benutzer*	(list) die Abfrageoption ermöglicht allen Usern, ihre Passwort-Einstellungen abzufragen.

Besonderheiten: Ohne Angabe einer Option wird ein interaktives Programm gestartet, welches ermöglicht, die aktuellen Werte aller Felder der Datei /etc/shadow der Reihe nach zu bearbeiten. Die aktuellen Werte sind in eckigen Klammern angegeben. Gibt man einen neuen Wert ein, so wird er übernommen, gibt man nichts ein, so gilt weiterhin der aktuelle Wert.

Statt mit dem Kommando chage können sämtliche Passwort-Einstellungen durch Editieren der Datei /etc/shadow vorgenommen werden. Zum Aufbau der /etc/shadow (siehe man 5 shadow). Feldangaben in den folgenden Ausführungen beziehen sich immer auf die Datei /etc/shadow.

Soll aus Sicherheitsgründen ein regelmäßiges Ändern des Passwortes erzwungen werden, muss neben der Mindest-Lebensdauer und der maximalen Lebensdauer unbedingt auch eine Angabe gemacht werden, wie lange das Passwort nach Ablauf der maximalen Lebensdauer noch gültig ist, es muss ein Wert >=1 angegeben werden!

Will man verhindern, dass ein Benutzer sein Passwort selbst verändern kann, so genügt es, die Mindest-Passwort-Lebensdauer höher einzustellen als die maximale Passwort-Lebensdauer.

Beispiel:

```
linux:~ # chage -M 20 peter
```

Die Maximale Passwort-Lebensdauer für den Benutzer `peter` wird auf 20 Tage festgelegt.

```
linux:~ # chage -E 2008-07-31 peter
```

Das absolute Verfallsdatum für das Konto des Benutzers `peter` wird auf den 31. Juli 2008 festgelegt.

```
linux:~ # chage peter
Ändere Shadow-Informationen für peter.
        Minimales Passwortalter [0]: 20
        Maximales Passwortalter [99999]: 50
        Password Expiration Warning [7]:
        Passwort Inaktive [-1]: 2
        Letzte Passwortänderung (JJJJ-MM-TT) [2004-08-14]:
        Account Expiration Date (YYYY-MM-DD) [1969-12-31]: 2004-09-30
Aging-Informationen geändert.
```

Hier wurde das interaktive Programm zur Bearbeitung der Passwort-Einstellungen gestartet. Bei den ersten beiden und beim letzten Eintrag wurden die aktuellen Werte überschrieben, der Rest wurde durch Drücken der Enter-Taste beibehalten. Die neuen Einstellungen können wie folgt abgefragt werden:

```
linux:~ # chage -l peter
Minimum:        20
Maximum:        50
Warning:        7
Inactive:       2
Last Change:            Aug 14, 2004
Passwort verfällt:      Okt 03, 2004
Passwort Inaktive:      Okt 05, 2004
Account verfällt:       Sep 30, 2004
```

`chattr` (1)

Einsatz: Anwender

Anwendung: Änderung von Dateiattributen (chattr = change attributes) in einem Linux-Dateisystem vom Typ ext2 oder ext3. Sämtliche Verwaltungsinformation zu Dateien wird im jeweiligen Inode gespeichert, so auch die Dateiattribute, die meist nicht näher erläutert werden. Man sollte jedoch zumindest ihr Konzept verstanden haben, selbst wenn man nicht so oft in die Verlegenheit kommen wird, sie anzuwenden. Die Attribute stellen sozusagen eine tiefergehende Ebene der Dateirechte dar. Vor allem für einige wichtige Systemdateien ist das Setzen dieser Attribute sinnvoll. Mit dem Befehl `lsattr` (= list attributes) kann überprüft werden ob und welche Attribute für eine Datei gesetzt sind (siehe Eintrag zu `lsattr` auf Seite 210).

Index