

KAPITEL 4

Grundlegende Linux-Befehle und -Konzepte

Die aktuellen grafischen Benutzeroberflächen von *Linux* sind denen von *Windows* sehr ähnlich. Sie ermöglichen jedoch nicht die volle Nutzung der vom System bereitgestellten Leistungen. Daher wird hier der Umgang mit den Kommandozeilenbefehlen beschrieben.

Anmelden am System

Der erste Schritt zum Kontakt mit einem Mehrbenutzersystem ist es, sich anzumelden. Die Gründe und die Vorgehensweise dafür finden Sie im folgenden Abschnitt.

Konzepte eines Mehrbenutzersystems

Linux hat von *Unix* die Idee von Benutzern und Gruppen geerbt. Jeder, der mit *Linux* arbeiten will, muss sich als Benutzer anmelden. Benutzer werden durch Namen unterschieden, die für jeden Namen unterschiedliche Berechtigungen zuweisen. Benutzer werden unterschieden, um unterschiedliche Anforderungsprofile an die Arbeitsumgebung zu ermöglichen. Dies entspricht dem Aufräumen, Anordnen und Gestalten des eigenen Schreibtischs. Jeder Benutzer bekommt einen privaten, von vornherein nur ihm zugänglichen Arbeitsbereich zugewiesen. Gruppen werden benutzt, um nicht für jeden Benutzer alle möglichen Einstellungen machen zu müssen, sondern Gemeinsamkeiten abzubilden, die dann einfach durch den Beitritt zu einer Gruppe erworben werden. Jeder Benutzer ist Mitglied mindestens einer Gruppe. Gruppen können ohne Mitglieder existieren, einen, mehrere oder alle Benutzer als Mitglied haben. Gruppen werden auch durch Namen unterschieden.

Das Einloggen

Lassen Sie uns annehmen, dass die Installation völlig ohne Probleme verlief und Sie jetzt folgenden Prompt auf Ihrem Bildschirm sehen:

```
Linux login:
```

Nur ganz wenige *Linux*-Benutzer haben Pech; sie müssen zunächst einige Einstellungen vornehmen, wenn das System gerade erst installiert wurde oder sich im Einzelbenutzer-Modus (*single user mode*) befindet. Im Augenblick wollen wir uns allerdings mit dem Einloggen in ein funktionierendes *Linux*-System befassen.

Das Einloggen sorgt natürlich dafür, dass die Benutzer voneinander unterschieden werden. Es lässt mehrere Personen gleichzeitig auf einem System arbeiten oder auch eine Person in unterschiedlichen so genannten Sitzungen und stellt sicher, dass Sie – und nur Sie – Zugang zu Ihren Dateien haben.

Vielleicht haben Sie *Linux* bei sich zu Hause installiert und denken jetzt: »Was soll's, außer mir arbeitet niemand auf diesem System, also brauche ich mich auch nicht einzuloggen«. Das Einloggen unter einem persönlichen Account (Zugangsberechtigung) bietet aber auch ein gewisses Maß an Schutz – unter Ihrem *Account* haben Sie keine Möglichkeit, wichtige Systemdateien zu zerstören oder zu entfernen. Solche heiklen Dinge werden unter dem *Account* des Systemverwalters erledigt (den wir im nächsten Kapitel besprechen).

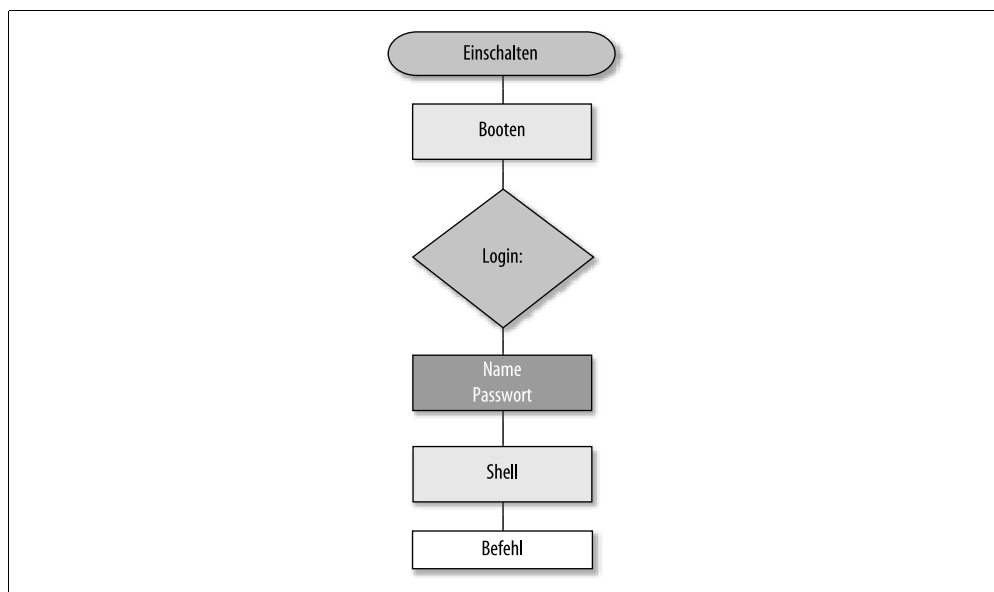


Abbildung 4-1: Vorgänge bis zum Einschalten

Hierarchie von Benutzerkonten

Unix führt zwei grundsätzliche Kategorien ein: den Systemverwalter und alle anderen. Dieses Konzept hat *Linux* übernommen. Der Systemverwalter darf alles, er kann aber auch alles zerstören. Der Rest der Benutzer hat eingeschränkte Rechte, darf nicht alle Programme ausführen und auch nicht auf alle Verzeichnisse zugreifen. Sein Schadenspotential ist begrenzt.

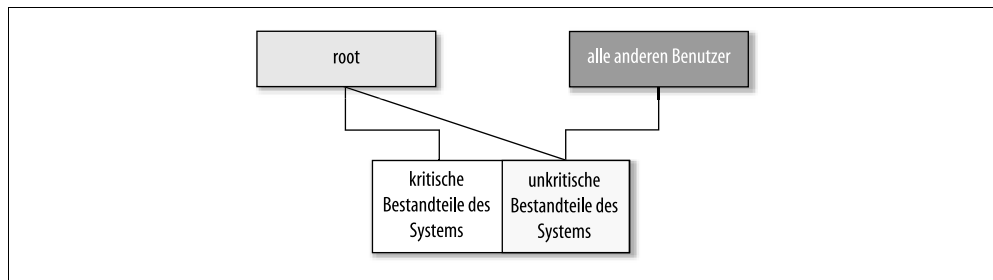


Abbildung 4-2: Zugriffe für verschiedene Benutzerkonten

Vergabe von Passwörtern

Das Passwort für den Nutzer *root* ist besonders wichtig, weil dieser Nutzer wirklich alles kann. Hier sollte also bei einem Computer, an den mehr als eine Person heran kann, besondere Sorgfalt verwendet werden.

root

Falls Sie noch kein Passwort festgelegt haben, empfehlen wir Ihnen, das jetzt zu tun. Geben Sie dazu einfach den Befehl `passwd` ein. Das Programm wird Sie nach dem Passwort fragen, das Sie verwenden wollen, und Sie dann auffordern, das Passwort noch einmal einzugeben, um sicherzustellen, dass Sie sich nicht vertippt haben.

Es gibt Richtlinien, die sicherstellen sollen, dass Passwörter nicht ohne weiteres von anderen Leuten erraten werden können. Einige Systeme prüfen die Passwörter und lehnen sie ab, wenn gewisse minimale Anforderungen nicht erfüllt sind. So wird zum Beispiel oft verlangt, dass ein Passwort aus mindestens sechs Zeichen besteht. Außerdem sollten Groß- und Kleinbuchstaben gemischt werden, oder es sollten außer Buchstaben und Ziffern noch andere Zeichen enthalten sein.

Um das Passwort zu ändern, geben Sie einfach den Befehl `passwd` noch einmal ein. Das Programm fragt Sie nach Ihrem alten Passwort (um sicherzustellen, dass Sie wirklich Sie sind) und lässt Sie dann das Passwort ändern.

Andere Nutzer

Das Passwort von anderen Nutzern kann auf zwei Arten geändert werden: Vom Benutzer selber oder vom Benutzer *root*. Das Ändern durch den Benutzer *root* ist insbesondere dann wichtig, wenn der Benutzer sein Passwort vergessen hat, oder ein Konto von einem andern Benutzer verwendet werden soll. Der Benutzer selber ruft zur Änderung seines Passwortes einfach den Befehl `passwd` auf, und muss je nach Distribution das alte Passwort noch einmal eingeben, oder kann gleich das neue Passwort eingeben. Zur Sicherheit muss die Eingabe des neuen Passwortes immer wiederholt werden.

Passwörter werden sehr häufig aus Sicherheitsgründen nicht auf dem Bildschirm dargestellt, so dass das Passwort blind eingegeben werden muss.

Der Nutzer *root* ändert das Passwort eines anderen Nutzers, indem er den Befehl `passwd` *Benutzername* aufruft und – immer ohne Abfrage des alten Passwortes – das neue Passwort zweimal eingibt.

Virtuelle Konsolen

Leistungssteigerung und Preisverfall für Computer machen es heutzutage möglich, das Vielfache der Rechenleistung eines historischen Rechenzentrums, das früher Riesensäle füllte, einfach unter oder auf den Schreibtisch zu stellen.

Geschichte von Konsolen

Während man um 1970 herum froh war, überhaupt interaktiv arbeiten zu können, das heißt unmittelbar auf einen Befehl eine Antwort zu erhalten, können heute alle Komponenten eines kompletten Computers in ein Gehäuse eingebaut werden. Die Trennung von Rechnen und Anzeigen ist aber in den Strukturen immer noch fest verwurzelt, so dass viele Programme sich immer noch so verhalten, als ob Berechnung und Anzeige- und Eingabevorrichtung (Terminal) getrennt wären.

Wenn Ihr Computer mit dem Internet verbunden ist (auch wenn dies nur temporär durch Einwahl über ein Modem oder eine ISDN-Adapter geschieht), sollten Sie auf jeden Fall nicht-triviale Passwörter für jedes Benutzerkonto (Account) verwenden. Benutzen Sie in Ihren Passwörtern auch Sonderzeichen und Zeichenketten, die nicht in irgendwelchen tatsächlich existierenden Wörtern oder Namen vorkommen.

Beachten Sie, dass die meisten Distributionen gleich einen so genannten grafischen Login-Manager installieren, so dass Sie wahrscheinlich nicht von dem etwas farblosen `login:-` Prompt in weißen Buchstaben auf dunklem Hintergrund begrüßt werden, sondern von einem grafisch ausgestalteten Anmeldebildschirm, der Ihnen möglicherweise sogar die auf dem System vorhandenen Benutzerkonten anzeigt und verschiedene grafische Umgebungen anbietet. Das grundlegende Anmeldeverfahren ist aber fast immer das gleiche, wie hier beschrieben wurde; Sie müssen immer noch Ihren Benutzernamen und Ihr Passwort angeben.

Bei der Installation von *Linux* wurden Sie wahrscheinlich aufgefordert, einen persönlichen *Login-Account* für sich einzurichten. Wenn Sie einen solchen *Account* haben, geben Sie am Prompt `Linux login:` den gewählten *Namen* ein. Falls Sie noch keinen Account angelegt haben, geben Sie *root* ein, weil dieser Account auf jeden Fall existiert. Einige Distributionen benutzen eventuell auch eine Zugangsberechtigung unter dem Namen *install* oder sonst einen Namen für die ersten Gehversuche im neuen System.

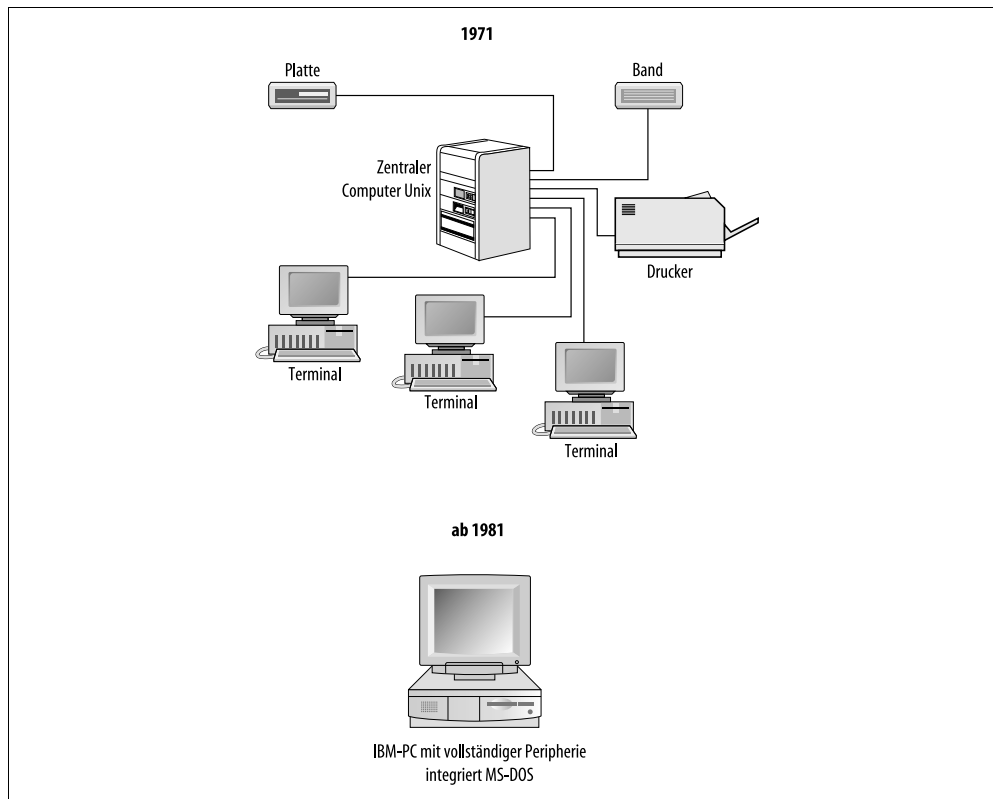


Abbildung 4-3: Fortschritt in der Integration

Nachdem Sie Ihren Account eingegeben haben, sehen Sie:

Password:

Sie müssen dann das gültige Passwort eingeben. Das Terminal wird dieses Passwort nicht auf dem Bildschirm anzeigen, damit niemand in Ihrer Nähe es mitlesen kann. Wenn Sie diesen Prompt nicht erhalten, sollten Sie die Passwortabfrage einrichten, um sich vor den flinken Fingern anderer Leute zu schützen. Wir kommen weiter unten noch darauf zurück.

Übrigens müssen sowohl der Login-Name als auch das Passwort in der richtigen Groß-/Kleinschreibung eingegeben werden. Achten Sie auf die *CAPS-LOCK*- oder Feststelltaste, denn die Eingabe von *ROOT* statt *root* bringt Sie nicht weiter.

Nach dem erfolgreichen Einloggen werden Sie einen Prompt sehen. Wenn Sie sich als *root* eingeloggt haben, kann das ein # sein. Für andere Benutzer wird in der Regel ein Dollar-Zeichen \$ angezeigt. Der Prompt kann außerdem den Namen enthalten, den Sie Ihrem System zugeordnet haben, oder auch das Verzeichnis, in dem Sie sich gerade befinden. Ganz egal, was hier angezeigt wird – Sie können jetzt Befehle eingeben. Man sagt, dass Sie sich auf der *Shell-Ebene* befinden, und der Prompt, den Sie sehen, ist der *Shell-*

Prompt. Der Name stammt von dem Programm, das zu diesem Zeitpunkt läuft und Ihre Eingaben verarbeitet und das Shell genannt wird. Im Moment brauchen wir uns um die Shell nicht zu kümmern, aber später in diesem Kapitel werden wir feststellen, dass die Shell eine Reihe nützlicher Dinge für uns tun kann.

Wenn wir in diesem Kapitel Befehle vorstellen, werden wir den Prompt einfach als \$ darstellen. Wenn Sie also Folgendes sehen:

```
$ pwd
```

bedeutet das, dass die Shell \$ anzeigt und dass Sie `pwd` eingeben müssen.

Umsetzung des Konsolenkonzepts unter Linux

Linux verhält sich so, als ob die Tastatur und der Monitor ein serielles Terminal wären. Es werden, obwohl keine entsprechende physikalische Schnittstelle Verwendung findet, Übertragungsrate und so weiter angegeben. (Diese Parameter können mit dem Befehl `stty` angezeigt werden). Dies ist für den durchschnittlichen Benutzer aber ohne Bedeutung. Es wird von *Linux* so getan, als ob das angeschlossene *virtuelle* Terminal, das heißt Monitor und Tastatur ein *DEC-VT102* Terminal wären. Man kann tatsächlich serielle Terminals an einen Computer, auf dem *Linux* läuft anschließen und hat keinen Unterschied in der Bedienung zu berücksichtigen, so als ob man direkt an der angeschlossenen Tastatur und vor dem Monitor säße.

Virtuelle Konsolen

Als Multitasking-System bietet *Linux* eine Reihe von interessanten Möglichkeiten, mehrere Dinge gleichzeitig zu tun. Sie können mit einer langwierigen Softwareinstallation beginnen und währenddessen E-Mails lesen oder ein Programm kompilieren.

Die meisten *Linux*-Benutzer werden das *X-Window*-System einsetzen, wenn sie das Multiprocessing nutzen wollen. Aber auch ohne die Installation von *X* können Sie mithilfe von virtuellen Konsolen einen ähnlichen Effekt erzielen. Diese Eigenschaft gibt es auch in einigen anderen *Unix*-Versionen, sie ist aber nicht allgemein verfügbar.

Nutzen mehrerer Konsolensitzungen

Halten Sie die linke ALT-Taste gedrückt, und drücken Sie dazu eine der Funktionstasten F1 bis F8, wenn Sie die virtuellen Konsolen ausprobieren möchten. Mit jeder Funktionstaste schalten Sie auf einen anderen Bildschirm mit einem neuen Login-Prompt um. Sie können so tun, als seien Sie mehrere Personen, und können sich so auf verschiedenen virtuellen Konsolen einloggen. Außerdem haben Sie die Möglichkeit, zwischen den Konsolen hin- und herzuschalten, um mehrere Dinge gleichzeitig zu erledigen. Sie können sogar auf jeder virtuellen Konsole eine *X*-Sitzung starten. Das *X-Window*-System läuft von Haus aus auf einer virtuellen Konsole, meist der Konsole 7. Wenn Sie also *X* gestartet haben und dann auf eine virtuelle Konsole gewechselt sind, kommen Sie durch Drücken von ALT+F7 zu *X* zurück.

Wenn die Kombination aus ALT- und Funktionstaste ein X-Menü oder eine andere Funktion auslöst, dann verwenden Sie stattdessen STRG+ALT+Funktionstaste.

Was ist ein Befehl

Wir haben bereits erwähnt, dass *Unix* eine enorme Anzahl an Befehlen kennt und dass Sie neue Befehle hinzufügen können. Was sind also *Unix*-Befehle, und wie werden sie gespeichert?

Rolle der Shell

Die Shell ist der Mittler zwischen dem Benutzer und den Programmen, die zur Verfügung stehen. Sie wartet auf die Eingabe- oder Return-Taste, um die eingegeben Zeichen zu analysieren und ein Programm aufzurufen. Einige Kommandos sind in der Shell eingebaut und müssen daher nicht durch ein externes Programm ausgeführt werden. Es ist jedoch möglich, die Shell zu zwingen, ein externes Programm auszuführen, obwohl eine gleichnamige Funktion in der Shell zur Verfügung steht, indem man beispielsweise den vollständigen Programmnamen mit Pfad eingibt.

Befehl und Parameter

Unter *Unix* ist ein Befehl einfach eine Datei. Der Befehl `ls` zum Beispiel ist eine Datei im Binärformat und liegt im Verzeichnis `/bin`. Statt `ls` einzugeben, könnten Sie also auch den kompletten Pfadnamen `/bin/ls` eingeben (auch *absoluter Pfadname* genannt):

```
$ /bin/ls
```

Das verleiht *Unix* Flexibilität und macht es zu einem mächtigen Betriebssystem. Ein Systemverwalter, der ein neues Utility bereitstellen möchte, kann dieses einfach in einem der Verzeichnisse installieren, in denen Befehle stehen. Es lassen sich auch verschiedene Versionen eines Befehls installieren – etwa eine neue Version zu Testzwecken in einem bestimmten Verzeichnis, während die alte Version an einer anderen Stelle verbleibt. Die Benutzer können dann selbst entscheiden, mit welcher Version sie arbeiten möchten. Hier ergibt sich häufig ein Problem: Manchmal geben Sie einen bekannten Befehl ein, aber das System antwortet mit einer Meldung wie `-bash: command not found`. Das Problem könnte darin bestehen, dass der Befehl in einem Verzeichnis steht, das von der Shell nicht durchsucht wird. Man nennt die Liste aller Verzeichnisse, in denen die Shell nach Befehlen sucht, den Pfad (*path*). Mit folgendem Befehl können Sie Ihren Pfad anzeigen lassen (denken Sie an das Dollar-Zeichen, ansonsten bekommen Sie nicht den Inhalt der Umgebungsvariablen zu sehen, sondern ihren Namen, den Sie ohnehin schon kennen!):

```
$ echo $PATH
/usr/local/bin:/usr/bin:/usr/X11R6/bin:/bin:/usr/lib/java/bin:\
/usr/games:/usr/bin/TeX:.
```

Hier müssen Sie genau hinsehen. Die Anzeige stellt eine Reihe von Pfadnamen dar, die durch Doppelpunkte getrennt sind. Der erste Pfadname in diesem Fall ist `/usr/local/bin`, der zweite `/usr/bin` usw. Wenn zwei Versionen eines Befehls vorhanden sind, die in `/usr/local/bin` und `/usr/bin` stehen, wird der Befehl in `/usr/local/bin` ausgeführt. Der letzte Pfadname in diesem Beispiel ist einfach ein Punkt; dieser bezeichnet das aktuelle Verzeichnis. Anders als die Eingabeaufforderung von Windows durchsucht *Linux* absichtlich nicht automatisch das aktuelle Verzeichnis, sondern Sie müssen es ausdrücklich dazu anweisen – so wie wir es hier gezeigt haben. Das automatische Ausführen von Befehlen im Arbeitsverzeichnis ist aus Sicht der Systemsicherheit eine schlechte Idee. (Ein Eindringling, der in Ihren Account vordringt, könnte ein böses Programm in eines Ihrer Arbeitsverzeichnisse kopieren.) Das betrifft aber hauptsächlich den Systemverwalter; normale Benutzer müssen sich wenig Sorgen darüber machen.

Wenn ein Befehl nicht gefunden wird, müssen Sie herausfinden, wo im System er sich befindet, damit Sie das betreffende Verzeichnis in Ihren Pfad einfügen können. Die Manpage sollte Ihnen sagen, wo der Befehl steht. Nehmen wir an, dass er sich im Verzeichnis `/usr/sbin` befindet, wo auch andere Befehle für die Systemverwaltung installiert sind. Ihnen ist klar, dass Sie Zugang zu den Befehlen für die Systemverwaltung brauchen, deshalb geben Sie folgendes ein (beachten Sie, dass das erste `PATH` ohne, das zweite dagegen mit einem Dollar-Zeichen geschrieben wird):

```
$ export PATH=$PATH:/usr/sbin
```

Dieser Befehl fügt `/usr/sbin` zum Suchpfad hinzu, und zwar als das Verzeichnis, das als letztes durchsucht wird. Der Befehl heißt soviel wie: »Definiere meinen Pfad als den alten Pfad plus `/usr/sbin`.«

Der eben vorgestellte Befehl funktioniert übrigens nicht in allen Shells. Die meisten *Linux*-Benutzer, die mit einer Bourne-kompatiblen Shell wie *bash* arbeiten, sollten keine Probleme damit haben. Wenn Sie allerdings *csh* oder *tcsh* benutzen, müssen Sie stattdessen diesen Befehl eingeben:

```
set path = ( $PATH; /usr/sbin )
```

Abschließend wollen wir noch auf ein paar Befehle hinweisen, die nicht als eigenständige Programmdateien existieren. `cd` ist einer davon. Die meisten dieser Befehle wirken sich auf die Shell selbst aus und müssen deshalb von der Shell verstanden und ausgeführt werden. Weil sie ein Teil der Shell sind, nennt man sie »interne Befehle« (built-in commands).

Befehle, die in einer Terminalsitzung eingegeben werden, können sehr flexibel den Wünschen angepasst werden. Zum Beispiel kann beim Befehl `ls` durch eine einzige Eingabe bestimmt werden, ob alle gewöhnlichen Dateien aufgelistet werden `ls`, oder nur Verzeichnisse `ls -d` oder auch solche Dateien, die mit einem Punkt im Namen anfangen `ls -a`. Diese Modifizierer für das Verhalten von Befehlen heißen Parameter, sie kommen fast immer in zwei Formen, als traditionelle Unix-Befehlsparameter `ls -a` oder in der GNU-Version `ls --all`, wobei in diesem Fall das Verhalten genau gleich ist, die GNU-Version ist aber leichter zu merken.

Berechtigungen und Pfade

Wie Windows und praktisch jedes andere moderne Betriebssystem auch, organisiert *Linux* Dateien in einer hierarchischen Verzeichnisstruktur. *Linux* gibt dabei nicht vor, wo Dateien stehen müssen, aber im Laufe der Jahre haben sich bestimmte Konventionen herausgebildet. Es gibt ein Standardisierungskomitee, das die *Linux-Standard-Base (LSB)* herausgibt. Sie werden zum Beispiel unter *Linux* ein Verzeichnis namens */home* vorfinden, in dem die Dateien der Benutzer gespeichert sind. Jeder Benutzer hat ein Unterverzeichnis unter */home*. Wenn Ihr Login-Name also *mdw* ist, stehen Ihre persönlichen Dateien im Verzeichnis */home/mdw*. Dies wird Ihr Home-Verzeichnis genannt. Selbstverständlich können Sie in Ihrem Home-Verzeichnis weitere Unterverzeichnisse anlegen.

Wenn Sie bis jetzt mit einem Windows-System gearbeitet haben, dann sieht der Schrägstrich (*/*) als Pfadtrenner natürlich sehr merkwürdig aus, weil Sie an den rückwärts gerichteten Schrägstrich (**) gewohnt sind. Am Schrägstrich ist aber nichts Besonderes. Schrägstriche wurden schon lange als Pfadtrenner verwendet, bevor überhaupt irgend jemand anfang, an MS-DOS oder Windows zu denken. Der rückwärts gerichtete Schrägstrich hat unter *Unix* eine andere Bedeutung (er schaltet die Sonderbedeutung des nächsten Zeichens ab, sofern eine vorhanden ist).

Wie Sie sehen, werden die Bestandteile eines Verzeichnisses durch Schrägstriche (Slashes) getrennt. Diese durch Schrägstriche getrennte Liste wird oft als *Pfadname* bezeichnet.

In welchem Verzeichnis befindet sich aber */home*? Natürlich im Verzeichnis */*. Dies wird das Root-Verzeichnis genannt – wir haben es bei der Einrichtung der Dateisysteme bereits erwähnt. Besondere Aufmerksamkeit erfordert die mehrfache Verwendung des Namens *root* als Namen des Systemverwalters wie auch als Ursprung des Dateisystems.

Linux unterscheidet normale (nicht ausführbare) Dateien von Programmen (nicht ausführbare Dateien) nicht durch den Namen, sondern durch Berechtigungen, die für jede Datei und jedes Verzeichnis vergeben werden. Ob nun eine Datei gültige Kommandos oder nur Informationen enthält, wird durch die Berechtigungs-Bits bestimmt. Daher kann zu Testzwecken sehr einfach ein Programm deaktiviert werden, indem die Ausführbarkeit abgeschaltet wird.

Häufig benutzte Befehle

Ein typisches *Linux*-System kennt so viele Befehle, dass man mit einer Beschreibung derselben mehrere hundert Seiten füllen kann. Außerdem haben Sie noch die Möglichkeit, eigene Befehle zu definieren. Wir wollen hier nur die Befehle vorstellen, mit denen Sie sich durch das System bewegen können, um es zu erkunden.

Befehle zum Bearbeiten von Texten

Sehr häufig hat man es mit Informationen zu tun, die in Listen abgelegt werden wie zum Beispiel Fußballergebnisse oder Verkaufszahlen. Um die Liste umzuordnen, also einmal

alphabetisch nach Verein oder nach Spieldatum, werden Befehle zum Bearbeiten von Texten eingesetzt:

- `cat`: Nimmt die Dateien, die als Parameter übergeben wurde und gibt sie »ohne Punkt und Komma« wieder aus. Verwendung erfährt dieser Befehl, um die beispielsweise die folgenden Befehle auf mehr als eine Datei anzuwenden.
- `sort`: Sortiert alphabetisch die Datei, die ihr als Parameter übergeben wurde.
- `uniq`: Gibt an, ob zwei aufeinander folgende Zeilen gleich sind, wird vor allem zusammen mit `sort` verwendet.
- `wc`: Zählt Zeichen, Wörter und Zeilen.

Befehle zum Verwalten von Dateien

Dateien können umbenannt, verschoben, gelöscht und angelegt werden

- `rm`: Löschen einer Datei
- `rmdir`: Löschen eines Verzeichnisses
- `mkdir`: Erzeugen eines Verzeichnisses
- `mv`: Verschieben und Umbenennen von Dateien
- `touch`: Erzeugen von Dateien und Setzen des Zugriffsdatums
- `ls`: Auflisten von Dateien

Symbolische Verknüpfungen

Manchmal empfiehlt es sich, eine Datei an mehr als einer bestimmten Stelle zu speichern, aber gleichzeitig dafür zu sorgen, dass alle Kopien dieselbe Datei darstellen. Dies wird meistens von Systemverwaltern angewendet, nicht von Benutzern. Eventuell möchten Sie mehrere Versionen eines Programms zur Verfügung haben, die dann *prog.0.9*, *prog.1.1* usw. heißen. Die gerade benutzte Version des Programms soll immer *prog* heißen. Vielleicht haben Sie eine Datei auch in einer bestimmten Partition abgelegt, weil dort gerade Platz war, aber das Programm, das diese Datei benutzt, verlangt, dass sich die Datei an einer anderen Stelle befindet, weil der Pfadname fest im Programm steht.

Linux benutzt *Links* (Verknüpfungen, Verweise), um mit solchen Situationen umzugehen. In diesem Abschnitt werden wir die *symbolic links* (symbolischen Verknüpfungen) vorstellen, die auch die am häufigsten benutzten und flexibelsten *Links* sind. Ein *symbolischer Link* ist so etwas wie eine Attrappe einer Datei, die nichts weiter tut, als auf eine andere Datei zu verweisen. Wenn Sie einen *symbolischen Link* zum Editieren, Lesen oder Ausführen aufrufen, wird das Betriebssystem dem Verweis folgen und Ihnen die tatsächlich existierende Datei präsentieren. *Symbolische Links* ähneln den Verknüpfungen von Windows 95/98/ME, sind aber viel mächtiger.

Wir wollen das am Beispiel der Datei **prog** darstellen. Wenn Sie eine symbolische Verknüpfung **prog** mit der eigentlichen Datei namens **prog.1.1** herstellen wollen, geben Sie Folgendes ein:

```
$ ln -s prog.1.1 prog
```

Sie haben damit eine Datei mit dem Namen **prog** angelegt, die aber nur eine Attrappe (ein Verweis) ist – wenn Sie **prog** aufrufen, starten Sie in Wirklichkeit **prog.1.1**. Lassen Sie uns nachsehen, was **ls -l** uns zu dieser Datei mitteilt:

```
$ ls -l prog
lrwxrwxrwx  2 mdw      users          8 Nov 17 14:35 prog -> prog.1.1
```

Das **l** am Anfang der Zeile zeigt an, dass **prog** ein Link ist, und der kleine Pfeil (**>**) verweist auf die eigentliche Datei.

Symbolische Links sind sehr einfach zu verstehen, wenn man sich einmal an die Vorstellung gewöhnt hat, dass eine Datei auf eine andere verweist. Bei der Installation von Software werden Ihnen ständig Links begegnen.

Dateien auflisten

Mit **ls** erfahren Sie, was sich in einem Verzeichnis befindet. Ohne Argument wird der Inhalt des aktuellen Verzeichnisses ausgegeben. Durch die Angabe eines Arguments sehen Sie den Inhalt eines anderen Verzeichnisses:

```
$ ls /home
```

Einige Systeme benutzen einen schicken **ls**-Befehl, der besondere Dateien – etwa Verzeichnisse und ausführbare Dateien – in Fettdruck oder sogar in verschiedenen Farben anzeigt. Wenn Sie die voreingestellten Farben ändern möchten, editieren Sie die Datei **/etc/DIR_COLORS**, oder erzeugen Sie eine Kopie davon unter dem Namen **.dir_colors** in Ihrem Home-Verzeichnis, und editieren Sie diese. Wie die meisten **Linux**-Befehle kann auch **ls** durch Optionen erweitert werden, die mit einem Bindestrich (**-**) eingeleitet werden. Achten Sie darauf, dass Sie vor dem Bindestrich eine Leerstelle lassen. Eine nützliche Option zu **ls** ist **-a** für »alles« – damit erfahren Sie Dinge, die Sie in Ihrem Home-Verzeichnis nie vermutet hätten:

```
$ cd
$ ls -a
.                .bashrc          .fwm2rc
..               .emacs           .xinitrc
.bash_history    .exrc
```

Ein einzelner Punkt steht für das aktuelle Verzeichnis und zwei Punkte für das direkt darüber liegende. Aber wozu braucht man die anderen Dateien, deren Name mit einem Punkt beginnt? Es handelt sich um versteckte Dateien. Der Punkt am Anfang des Dateinamens bewirkt, dass diese Dateien mit dem einfachen Befehl **ls** nicht angezeigt werden können. Viele Programme benutzen versteckte Dateien für die Konfiguration durch den Benutzer – als Voreinstellungen, die Sie vielleicht einmal ändern möchten. Sie können zum Beispiel in der Datei **.xdefaults** Befehle eintragen, die das Verhalten von Program-

men unter dem *X-Window*-System beeinflussen. Sie werden diese Dateien die meiste Zeit nicht benötigen, Sie werden sie aber benutzen, wenn Sie Ihr System konfigurieren. Wir wollen weiter unten einige dieser Dateien vorstellen.

Berechtigungen (3 für den Eigner, 3 für die Gruppe, 3 für Sonstige)	Eigner	Gruppe	Datum und Zeit der letzten Änderung			Name
-rw-r--r--	1 mdw	users	2321	Mar 15	1994	Fontmap
-rw-r--r--	1 mdw	users	139836	Aug 11	09:11	Index.whole
drwxr-xr-x	2 mdw	users	1024	Jan 25	1994	Xfonts
drwxr-xr-x	3 mdw	users	1024	Sep 20	07:40	bin
-rw-r--r--	1 mdw	users	124408	Nov 2	10:53	bitgif.tar.gz
drwxr-xr-x	2 mdw	users	2048	Jan 21	1994	bitmaps

Dateityp
(„d“ steht für Verzeichnis)

Anzahl der echten Links

Größe in Bytes
(bei Verzeichnissen die Anzahl der Bytes, die Informationen über das Verzeichnis enthalten)

Abbildung 4-4: Ausgabe von `ls -l`

Eine andere nützliche Option zu `ls` ist `-l` für »lang«. Damit erhalten Sie zusätzliche Informationen über die Dateien. Abbildung 4-4 zeigt eine typische Ausgabe dieses Befehls und die Bedeutung der einzelnen Felder. Mit der Option `-h` (für »human«) bekommen Sie die Dateigrößen geeignet skaliert und damit leichter lesbar angezeigt.

Wir werden die Felder *Permissions*, *Owner* und *Group* (Zugriffsrechte, Eigner und Gruppe) später noch besprechen. Der Befehl `ls` zeigt auch die Größe der Dateien sowie das Datum der letzten Änderung an.

Wenn wir schon einmal dabei sind, könnten wir auch gleich ein Verzeichnis namens `~/programs` anlegen. Geben Sie in Ihrem Home-Verzeichnis entweder

```
$ mkdir programs
```

ein oder den kompletten Pfadnamen:

```
$ mkdir /home/mdw/programs
```

Wechseln Sie jetzt in das neue Verzeichnis:

```
$ cd programs
$ pwd
/home/mdw/programs
```

Mit der Zeichenfolge `..` wird das »unmittelbar übergeordnete Verzeichnis« angesprochen. Sie können also in Ihr Home-Verzeichnis zurückgelangen, indem Sie Folgendes eingeben:

```
$ cd ..
```

Dateien mit `more` oder `less` anzeigen

Eine Möglichkeit, eine Datei anzuzeigen, ist das Starten eines Editors:

```
$ emacs .bashrc
```

Wenn Sie allerdings nur kurz in die Datei hineinschauen wollen, ohne den Inhalt zu ändern, sind andere Befehle geeigneter. Der einfachste davon hat den merkwürdigen Namen `cat` – der Name kommt von *concatenate* (aneinander hängen), weil man damit auch mehrere Dateien aneinander hängen kann:

```
$ cat .bashrc
```

Weil eine lange Datei so schnell über den Bildschirm wandert, dass ein Mitlesen nicht möglich ist, benutzen die meisten Leute stattdessen den Befehl `more`:

```
$ more .bashrc
```

Damit bekommen Sie jeweils eine Bildschirmseite voll Text angezeigt und müssen dann die Leertaste drücken, um die Anzeige fortzusetzen. `more` kennt eine ganze Reihe von mächtigen Optionen. So können Sie zum Beispiel in der Datei nach einem String suchen. Geben Sie dazu einen Schrägstrich (`/`) ein, dann den **Suchstring**, und drücken Sie abschließend die Eingabetaste.

Eine beliebte Variante von `more` ist der Befehl `less`. Er hat noch weitaus mächtigere Optionen, beispielsweise können Sie damit eine bestimmte Stelle in einer Datei markieren und später dorthin zurückspringen.

Beim Einloggen bringt das System Sie in Ihr `/home`-Verzeichnis. Um das zu prüfen, geben Sie den Befehl für »print working directory« (oder `pwd`) ein:

```
$ pwd
/home/mdw
```

Das System bestätigt, dass Sie sich in `/home/mdw` befinden.

Sie werden nicht lange Spaß daran haben, sich nur in einem Verzeichnis aufzuhalten. Versuchen Sie jetzt, mit dem Befehl `cd` in ein anderes Verzeichnis zu wechseln:

```
$ cd /usr/bin
$ pwd
/usr/bin
$ cd
```

Wo sind wir jetzt? Der Befehl `cd` ohne Argumente bringt uns in unser Home-Verzeichnis zurück. Übrigens wird das Home-Verzeichnis oft als Tilde (`~`) dargestellt. Die Angabe `~/programs` bedeutet also, dass `programs` sich direkt unterhalb Ihres Home-Verzeichnisses befindet.

Das Gegenstück zum Befehl `mkdir` ist `rmdir`, mit dem Sie Verzeichnisse entfernen:

```
$ rmdir programs
```

In ähnlicher Weise werden mit dem Befehl `rm` Dateien gelöscht. Wir wollen das hier nicht demonstrieren, weil wir noch nicht einmal gezeigt haben, wie man Dateien erzeugt. Im

Allgemeinen werden Sie dafür einen der Editoren `vi` oder `Emacs` benutzen, aber auch einige der Befehle, die wir später in diesem Kapitel besprechen, erzeugen Dateien. Mit der `-r` Option löscht `rm` ein ganzes Verzeichnis und alles, was es enthält (daher ist diese Option mit äußerster Vorsicht zu genießen!).

Befehle zur Verwaltung

Der wichtigste Befehl zur Verwaltung ist der zum geordneten Abschalten des Computers: `shutdown`, sehr häufig in der Form `shutdown -h now` eingegeben.

Die meisten *Linux*-Distributionen verwenden das von *Unix System V* übernommene Konzept von `Runlevels` (Betriebsarten). Zwischen diesen wird mit dem Befehl `init` oder `telinit` gewechselt. Dies wird in der Praxis vor allem zum Ein- (`init 5`) oder Ausschalten (`init 3`) der grafischen Benutzeroberfläche verwendet, sowie zum Ausschalten (`init 2`) oder Einschalten (`init 3`) der Netzwerkfunktionalität.

Shells

Wir haben bereits erwähnt, dass beim Anmelden im Konsolenmodus eine Shell für Sie geöffnet wird. Wenn Ihr System so konfiguriert ist, dass eine grafische Oberfläche gestartet wird, sehen Sie nach dem Anmelden diese Oberfläche und können dort ein `xterm`-Fenster (oder Ähnliches) aufmachen, um eine Shell zu bekommen. Die Shell interpretiert Ihre Befehle und führt sie aus. Wir wollen uns zunächst ein wenig mit den verschiedenen Shells befassen, weil sie auch Einfluss auf einige Punkte haben, die wir weiter unten im Text besprechen.

Falls Sie es verwirrend finden, dass es unter *Unix* so viele verschiedene Shells gibt, betrachten Sie das einfach als ein Zeichen der Evolution. Glauben Sie uns: Sie wären bestimmt nicht glücklich, wenn Sie mit der ersten für *Unix* entwickelten Shell arbeiten müssten. Obwohl sie zu ihrer Zeit (Mitte der 70er-Jahre) eine tolle Benutzerschnittstelle war, fehlten ihr doch einige nützliche Anwendungen für die interaktive Arbeit – unter anderem die Dinge, auf die wir in diesem Abschnitt noch eingehen. Deshalb sind im Laufe der Zeit andere Shells entwickelt worden, und das gibt Ihnen die Gelegenheit, die Shell auszuwählen, die Ihrer Arbeitsweise am besten entspricht.

Mit dem Befehl `chsh` können Sie eine andere Shell auswählen:

```
$ chsh
Enter Password: Geben Sie hier aus Sicherheitsgründen Ihr Passwort ein.
Changing the login shell for kalle.
Enter the new value, or press return for the default
```

```
Login Shell [/bin/sh]: /bin/bash
```

Bevor ein Benutzer eine bestimmte Shell als Login-Shell auswählen kann, muss diese installiert und vom Systemadministrator durch Eintragen in `/etc/shells` freigegeben worden sein.

Die Unterschiede zwischen den Shells lassen sich aus verschiedenen Blickwinkeln betrachten. Einerseits kann man *Bourne*-Shell-kompatiblen von *C*-Shell-kompatiblen Schnittstellen unterscheiden. Dieser Gesichtspunkt ist interessant, wenn es um die Programmierung der Shell geht (das Schreiben von Shell-Skripten). Die Bourne- und die C-Shell benutzen unterschiedliche Konstrukte für die Shell-Programmierung. Die meisten *Linux*-Anwender halten die Bourne-Shell für die bessere Alternative, und es gibt viele *Unix*-Utilities, die nur unter der Bourne-Shell laufen.

Andererseits kann man Shells auch danach einteilen, ob sie das Editieren der Kommandozeile erlauben oder nicht (alle neueren Shells bieten diese Möglichkeit). Sowohl der `sh` als auch der `csh` fehlt diese nützliche Erweiterung.

Wenn Sie diese beiden Kriterien – Kompatibilität mit der Bourne-Shell und das Editieren der Befehlszeile – miteinander kombinieren, bleiben die Shells `bash`, `ksh` und `zsh` als sinnvolle Alternativen übrig. Sie sollten verschiedene Shells ausprobieren, bevor Sie sich endgültig entscheiden. Es kann von Vorteil sein, mehr als eine Shell zu kennen, wenn Sie einmal vor einem System sitzen, auf dem nicht alle Shells vorhanden sind.

bash

Bourne-Again-Shell. Sie ist die unter *Linux* am häufigsten benutzte und mächtigste Shell. Sie entspricht dem POSIX-Standard, ist zur Bourne-Shell kompatibel und wird vom GNU-Projekt der Free Software Foundation entwickelt und vertrieben. Sie ermöglicht das Editieren der Befehlszeile sowie History-Substitution.

Geben Sie folgenden Befehl ein, wenn Sie herausfinden möchten, mit welcher Shell Sie arbeiten. Sie erhalten den vollen Pfadnamen zu Ihrer Shell. Vergessen Sie nicht, das Dollar-Zeichen mit einzugeben:

```
$ echo $SHELL
```

Die Wahrscheinlichkeit ist groß, dass Sie mit `bash` arbeiten, der Bourne-Again-Shell. Falls Sie sich in einer anderen Shell wiederfinden, ist dies vielleicht eine gute Gelegenheit, auf `bash` umzusteigen. Die `bash` ist äußerst umfangreich, folgt dem POSIX-Standard, wird gut unterstützt und ist unter *Linux* sehr beliebt.

sh

ist meistens ein Link auf die tatsächlich als Standard-Shell zu verwendende Shell.

zsh

Z-Shell. Die neueste der Shells. Kompatibel zur Bourne-Shell. Das Editieren der Befehlszeile ist möglich.

csh

C-Shell. Sie wurde in Berkeley entwickelt. Sie ist bei der interaktiven Arbeit weit gehend kompatibel zur Bourne-Shell, bietet aber eine andere Schnittstelle bei der Shell-Programmierung. Das Editieren der Befehlszeile ist nicht möglich, aber es gibt einen ausgefeilten Ersatz namens History-Substitution dafür. Unter Linux ist csh nur ein Aliasname für die neuere tcsh.

tcsh

Erweiterte C-Shell mit der Möglichkeit, die Befehlszeile zu editieren.

ksh

Korn-Shell. Sie ist vielleicht die beliebteste Unix-Shell in der *IBM*-Welt und war die erste, die die Bourne-Shell um einige Merkmale moderner Benutzerschnittstellen erweiterte (einige davon wurden von der C-Shell übernommen). Sie ist kompatibel zur Bourne-Shell und bietet Editiermöglichkeiten in der Befehlszeile.

Ein-/Ausgabe-Umleitung

Wenn beispielsweise einer der klassischen *Linux*-Befehle zur *Textbearbeitung* (Achtung, nicht *Textverarbeitung*!) ohne irgendeinen Parameter aufgerufen wird, so verarbeitet er alle folgenden Tastatureingaben als Daten, ohne der Shell die Herrschaft über die Terminalsitzung zurückzugeben. Wenn also *wc* ohne eine Parameter aufrufen wird, so passiert scheinbar gar nichts – es ist nur kein Eingabeaufforderung (Shell-Prompt) zu sehen. Jetzt kann beliebiger Text eingetippt werden, der so, wie er eingegeben wird, am Bildschirm erscheint. Sobald man jedoch das Steuerzeichen STRG-D (das ist das Standard-Steuersymbol für Dateiende) eingibt, erscheinen nacheinander in einer Zeile die Anzahl der nach *wc* eingegebenen Zeilen, Worte und Zeichen. Es besteht kein Unterschied im Verhalten von *wc*, ob der zu verarbeitende Zeichenstrom aus einer schon gespeicherten Datei stammt oder direkt von einem Gerät wie der Tastatur erzeugt wird. Ebenso wird der Bildschirm als Anzeige und eine Datei zur Speicherung des Ausgabe-Zeichenstroms gleich behandelt. Dieses Verhalten kann dazu genutzt werden, Befehle wie *wc* dazu zu bringen, auszuwertende Daten von der Platte zu holen (*wc < /var/log/messages*) oder auch das Ergebnis auf Platte zu schreiben (*wc /var/log/messages > ./Logdateigröße*) oder um mehrere Befehle hintereinander zu schalten, um ohne Zwischenspeichern von Zwischenresultaten direkt ein Ergebnis zu erhalten (*sort /var/log/messages | uniq | less*). *Linux*-Befehle für die Kommandozeile lesen grundsätzlich von der Standardeingabe (Datei, Gerät wie Tastatur) und schreiben auf die Standardausgabe (Bildschirm, Datei). Resultate werden auf die Standard-Fehlerrückmeldung geschrieben, meist den Bildschirm. Alle drei Standardschnittstellen eines *Linux*-Kommandos lassen sich umstellen (I/O-Redirection).

Das Speichern von Befehlsausgaben

Systemverwalter (und auch andere Menschen) sehen eine ganze Reihe von wichtigen Systemmeldungen über den Bildschirm huschen. Oft ist es wichtig, dass man diese Nachrichten aufbewahrt, um sie später in Ruhe auswerten zu können oder um sie an jemanden zu schicken, der mit ihrer Hilfe herausfindet, was schief gelaufen ist. In diesem Abschnitt wollen wir Ihnen deshalb etwas zum Thema Umleitung (*redirection*) erzählen, einer weiteren äußerst nützlichen Eigenschaft von *Linux*-Shells im Besonderen und *Linux*-Programmen im Allgemeinen. Falls Sie aus der *Windows*-Welt kommen, haben Sie wahrscheinlich bereits eine ähnliche, wenn auch eingeschränkte Art der Umleitung in der dortigen Eingabeaufforderung kennen gelernt.

Wenn Sie ein Größer-als-Zeichen (>) und einen Dateinamen hinter einem beliebigen Befehl angeben, bewirken Sie damit, dass die Ausgaben des Befehls in diese Datei geschrieben werden. Wenn Sie zum Beispiel die Ausgabe von `ls` abspeichern wollen, geben Sie Folgendes ein:

```
$ ls /usr/bin > ~/Binaries
```

Die Liste der Dateien in `/usr/bin` wird damit in einer Datei namens `Binaries` in Ihrem Home-Verzeichnis abgelegt. Falls es dort schon eine Datei `Binaries` gibt, wird das > die alte Datei löschen und die Ausgabe des Befehls `ls` unter diesem Namen speichern. Das Überschreiben einer Datei passiert recht häufig. Wenn Sie mit einer der Shells `csh` oder `tcsh` arbeiten, können Sie sich folgendermaßen gegen unbeabsichtigtes Überschreiben schützen:

```
$ set noclobber
```

In der `bash` erzielen Sie denselben Effekt mit:

```
$ noclobber=1
```

Es muss keine 1 sein, ein beliebiges Zeichen reicht.

Man kann auch die neue Ausgabe an eine bestehende Datei anhängen. Nehmen wir an, dass Sie bereits eine Auflistung von `/usr/bin` gespeichert haben und dass Sie jetzt den Inhalt von `/bin` an diese Datei anhängen möchten. Mit zwei Größer-als-Zeichen können Sie die neue Auflistung an die bestehende Datei `Binaries` anhängen:

```
$ ls /bin >> ~/Binaries
```

Die Umleitung von Befehlsausgaben ist sehr nützlich, wenn Sie ein Utility wiederholt starten und die Ausgaben zwecks Fehlersuche abspeichern.

Die meisten *Linux*-Programme erzeugen zwei Ausgaben (*output streams*). Die eine bezeichnet man als Standardausgabe (*standard output*), die andere als Standardfehlerausgabe (*standard error*). Die C-Programmierer unter den Lesern werden dies wiedererkennen: Die Standardfehlerausgabe ist der Dateiverweis `stderr`, in den Sie Systemmeldungen schreiben lassen.

Das >-Zeichen lenkt nicht die Standardfehlerausgabe um – Sie benutzen es, um die regulären Ausgaben zu speichern, ohne die Datei mit Fehlermeldungen zu überladen. Was aber passiert, wenn Sie gerade die Fehlermeldungen speichern wollen? Während der Fehlersuche kommt das häufig vor. Die Lösung ist ein Größer-als-Zeichen >, gefolgt von

einem Kaufmanns-Und (&). (Diese Konstruktion funktioniert in allen modernen *Linux*-Shells.) Damit lenken Sie sowohl die Standardausgabe als auch die Standardfehlerausgabe um. Ein Beispiel:

```
$ gcc invinitjig.c >& error-msg
```

Dieser Befehl speichert alle Nachrichten des Compilers gcc in der Datei error-msg. In der Bourne-Shell und in bash können Sie das Gleiche so formulieren:

```
$ gcc invinitjig.c &> error-msg
```

Lassen Sie uns jetzt alle Register ziehen: Wir nehmen an, dass Sie die Fehlermeldungen, aber nicht die normalen Ausgaben speichern wollen – also die Standardfehlerausgabe ohne die Standardausgabe. In der Bourne-Shell und dazu kompatiblen Shells erledigen Sie das mit:

```
$ gcc invinitjig.c 2> error-msg
```

Die Shell ordnet selbstständig der Standardausgabe die 1 und der Standardfehlerausgabe die 2 zu. Deshalb speichern Sie mit diesem Befehl nur die Fehlermeldungen.

Zum Schluss wollen wir annehmen, dass Sie die Standardausgabe nicht brauchen – Sie wollen Ihren Bildschirm sauber halten. Die Lösung ist eine Umleitung in eine spezielle Datei namens /dev/null. (Vielleicht haben Sie schon einmal jemanden sagen hören: »Richten Sie Ihre Kritik an /dev/null.« Jetzt wissen Sie also, woher dieser Spruch kommt.) Das Verzeichnis /dev ist der Platz, an dem alle *Unix*-artigen Systeme besondere Dateien ablegen, die den Zugriff auf Terminals, Bandlaufwerke und andere Geräte regeln. Die Datei /dev/null ist allerdings einzigartig – alles, was Sie dorthin schicken, verschwindet in einem schwarzen Loch. Der folgende Befehl speichert beispielsweise die Standardfehlerausgabe und lässt die Standardausgabe verschwinden:

```
$ gcc invinitjig.c 2>error-msg >/dev/null
```

Damit sollten Sie in der Lage sein, genau die Ausgaben zu erzeugen, die Sie sehen möchten.

Falls Sie sich schon gefragt haben, ob auch das Kleiner-als-Zeichen (<) in der Shell eine Bedeutung hat – die Antwort lautet: ja. Damit erreichen Sie, dass Befehle ihre Eingaben aus einer Datei lesen. Da die meisten Befehle sowieso die Möglichkeit bieten, auf der Befehlszeile Eingabedateien zu definieren, wird diese »Eingabeumleitung« selten genutzt.

Manchmal möchte man erreichen, dass ein Hilfsprogramm die Ausgaben eines anderen Programms benutzt. Ein Beispiel: Mit dem Befehl sort können Sie die Ausgaben eines anderen Befehls in eine sinnvolle Reihenfolge bringen. Es wäre ziemlich umständlich, wenn Sie die Ausgaben eines Befehls zunächst in eine Datei umlenken und anschließend diese Datei mit sort bearbeiten würden, etwa so:

```
$ du > du_output  
$ sort -nr du_output
```

Linux bietet allerdings eine viel schnellere und effizientere Methode, dies zu bewerkstelligen, nämlich die *Pipe*. Geben Sie zwischen den beiden Befehlen einfach einen senkrechten Strich ein:

```
$ du | sort -nr
```

Die Shell schickt alle Ausgaben des Programms *du* weiter an das Programm *sort*.

Im obigen Beispiel steht *du* für »disk usage« (Plattenbelegung) und gibt an, wie viele Blöcke jede einzelne Datei im aktuellen Verzeichnis belegt. In der Regel erfolgt die Ausgabe in eher zufälliger Reihenfolge:

```
$ du
10      ./zoneinfo/Australia
13      ./zoneinfo/US
9       ./zoneinfo/Canada
4       ./zoneinfo/Mexico
5       ./zoneinfo/Brazil
3       ./zoneinfo/Chile
20      ./zoneinfo/SystemV
118     ./zoneinfo
298     ./ghostscript/doc
183     ./ghostscript/examples
3289    ./ghostscript/fonts
.
```

Wir haben deshalb beschlossen, die Ausgabe durch *sort* mit den Optionen *-n* und *-r* bearbeiten zu lassen. Dabei sorgt *-n* dafür, dass »numerisch sortiert« wird statt der üblichen ASCII-Sortierung. *-r* bewirkt, dass die »Sortierreihenfolge umgekehrt« wird, so dass die größten Zahlen zuerst erscheinen. Die Ausgabe lässt schnell erkennen, welche Dateien und Verzeichnisse den meisten Speicherplatz beanspruchen:

```
$ du | sort -rn
34368  .
16005  ./emacs
16003  ./emacs/20.4
13326  ./emacs/20.4/lisp
4039   ./ghostscript
3289   ./ghostscript/fonts
.
```

Weil so viele Dateien vorhanden sind, sollten wir noch eine zweite Pipe einsetzen, um die Ausgabe durch den Befehl *more* zu leiten (eine der häufigsten Anwendungen für Pipes):

```
$ du | sort -rn | more
34368  .
16005  ./emacs
16003  ./emacs/20.4
13326  ./emacs/20.4/lisp
4039   ./ghostscript
```

```
3289      ./ghostscript/fonts
          .
          .
```

Eine Alternative zu `more` wäre hier die Verwendung des Befehls `head`, der nur die ersten paar Zeilen (10 ist der Default) anzeigt. Wenn es einen `head`-Befehl gibt, muss es natürlich auch einen `tail`-Befehl geben, der nur die letzten paar Zeilen anzeigt.

Hilfreiche Sondertasten und der Umgang mit Ihnen

Es gibt viele weitere Möglichkeiten, die Befehlszeile zu editieren, aber schon mit den hier gezeigten einfachen Tricks können Sie recht weit kommen. Wenn Sie mit dem Editor `emacs` arbeiten, werden Sie feststellen, dass viele `emacs`-Befehle auch in der Shell funktionieren. Falls Sie ein Fan von `vi` sind, können Sie Ihre Shell so einrichten, dass sie mit `vi`-Befehlen statt mit `Emacs`-Befehlen arbeitet. Geben Sie dazu in der `bash`, `ksh` oder `zsh` Folgendes ein:

```
$ export VISUAL=vi
```

In der `tcsh` geben Sie ein:

```
$ setenv VISUAL vi
```

Die Shell behält eine Anzahl von Befehlen, die während einer Terminalsitzung eingegeben wurde im Gedächtnis. Mit den Cursorastasten (die »Pfeiltasten«) können alte Befehle zurückgeholt werden. Dies ist insbesondere dann interessant, wenn man

- nachvollziehen will, was man getan hat,
- dasselbe Vorgehen wiederholen möchte
- ein ähnliches Vorgehen, wie schon einmal durchgeführt, mit geringen Änderungen und geringem Aufwand durchführen will.

`BildAuf`, `PgUp` nach dem Gedrückt-Halten der Umschalt-Taste für Groß-Klein-Schreibung ermöglicht es, aus dem sichtbaren Bereich gelaufene Teile der Terminalsitzung wieder anzusehen, sofern nicht zwischendurch ein Umschalten der virtuellen Konsolen vorgenommen wurde.

Nützliche Tastenkombinationen

Wenn Sie einen Befehl eingeben, sollte das Betätigen der `BACKSPACE`-Taste das letzte Zeichen wieder löschen. Mit `STRG-U` wird die Zeile von der Cursor-Position bis zum Zeilenende gelöscht; wenn sich der Cursor am Zeilenanfang befindet, löscht diese Tastenkombination also die ganze Zeile.¹ Wenn Sie einen Befehl fertig eingegeben haben und ihn ausführen, sollten Sie ihn mit `STRG-C` abbrechen und mit `STRG-Z` unterbrechen können. Wenn Sie das unterbrochene Programm wieder aufnehmen wollen, geben Sie `fg` für »foreground« (Vordergrund) ein.

¹ `STRG-U` bedeutet: Halten Sie die `STRG`-Taste gedrückt, und drücken Sie gleichzeitig `u`.

STRG-S hält die Terminalausgabe solange an, bis sie mit STRG-Q wieder eingeschaltet wird. Dies ist heutzutage wahrscheinlich nicht mehr so nützlich, weil die meisten Terminal-emulationen ohnehin das Scrolling unterstützen, aber Sie sollten diese Tastenkombination trotzdem kennen, denn wenn Sie aus Versehen STRG-S gedrückt haben, reagiert Ihr Terminal auf einmal nicht mehr. Drücken Sie dann einfach STRG-Q, um wieder Gehör zu finden; das Terminal hat nur auf Sie gewartet.

Falls irgendeine dieser Tastenkombinationen nicht funktioniert, ist Ihr Terminal nicht richtig konfiguriert. Mit dem Befehl `stty` lässt sich das beheben. Geben Sie dazu ein:

```
stty funktion taste
```

wobei *funktion* das bezeichnet, was Sie tun möchten, während *taste* die Taste (oder Tastenkombination) ist, die Sie dazu drücken müssen. Definieren Sie eine STRG-Tastenkombination, indem Sie der Taste einen Zirkumflex (^) voranstellen.

Hier zeigen wir Beispiele für die Funktionen, die wir weiter oben schon erwähnt haben:

```
$ stty erase ^H
$ stty kill ^U
$ stty intr ^C
$ stty susp ^Z
```

Die erste Tastenkombination, ^H, steht für den ASCII-Code, der von der BACKSPACE-Taste erzeugt wird.

Mit `stty -a` können Sie übrigens eine Liste der Einstellungen für Ihr Terminal auf den Bildschirm holen. Das heißt allerdings nicht, dass Sie diese Liste auch verstehen werden! `stty` ist ein sehr komplexer Befehl mit vielen Anwendungsmöglichkeiten, von denen einige profunde Kenntnisse zum Thema Terminals erfordern.

Die Kompletterung von Namen und Befehlen

Lassen Sie uns zuerst einen einfachen Trick ausprobieren, der Ihnen viel Zeit ersparen kann. Geben Sie Folgendes ein, ohne RETURN zu drücken:

```
$ cd /usr/inc
```

Drücken Sie jetzt die TAB-Taste. Die Shell wird lude ergänzen und den vollen Pfadnamen `/usr/include` anzeigen. Wenn Sie dann RETURN drücken, wird der Befehl ausgeführt.

Die Kompletterung erfolgt nach dem »Minimalprinzip«: Sie müssen nur so viele Zeichen eingeben, bis der Name von allen anderen Namen im betreffenden Verzeichnis eindeutig unterschieden werden kann. Die Shell sucht und komplettiert dann den Namen – einschließlich des folgenden Schrägstrichs, wenn es sich um einen Verzeichnisnamen handelt.

Sie können die Kompletterung auch auf Befehle anwenden. Wenn Sie zum Beispiel

```
$ emacs
```

eingeben und dann die TAB-Taste drücken, wird die Shell das `cs` hinzufügen, und Sie erhalten `emacs` (solange kein anderer Befehl in Ihrem Pfad mit `ema` beginnt).

Was passiert, wenn es mehr als eine Datei gibt, deren Name auf Ihre Eingabe passt? Wenn Dateinamen mit denselben Buchstaben beginnen, ergänzt die Shell Ihre Eingabe bis zu der Stelle, an der die Namen sich unterscheiden. Für die meisten Shells ist die Hilfestellung damit erledigt. `bash` kennt allerdings eine nette Erweiterung hierzu: Wenn Sie die `TAB`-Taste zweimal drücken, werden alle Möglichkeiten der Komplettierung angezeigt. Wenn Sie zum Beispiel

```
$ cd /usr/l
```

eingeben und dann zweimal `TAB` drücken, erhalten Sie so etwas wie:

```
lib          local
```

Automatische Erweiterung von Dateinamen

Eine andere Möglichkeit der Zeitersparnis ist die Benutzung von speziellen Zeichen, um Dateinamen abzukürzen (Wildcards). Sie können mithilfe dieser Zeichen mehrere Dateien gleichzeitig ansprechen. Windows bietet ein paar einfache Möglichkeiten dieser Art: Sie können das Fragezeichen als Platzhalter für ein »beliebiges Zeichen« einsetzen, und ein Stern steht für eine »Zeichenkette«. *Linux* kennt solche Wildcards ebenfalls, ist aber in deren Anwendung wesentlich flexibler.

Nehmen wir an, in einem Verzeichnis stehen folgende Dateien mit C-Quellcode:

```
$ ls
inv1jig.c  inv2jig.c  inv3jig.c  invinitjig.c  invpar.c
```

Um nur die Dateien aufzulisten, deren Namen sieben Buchstaben lang sind und sich nur im vierten Buchstaben unterscheiden, könnten Sie

```
$ ls inv?jig.c
inv1jig.c  inv2jig.c  inv3jig.c
```

eingeben. Die Shell sucht nach Dateinamen, in denen ein Zeichen das Fragezeichen ersetzt. Deshalb werden *inv1jig.c*, *inv2jig.c* und *inv3jig.c* aufgelistet, jedoch nicht *invinitjig.c*, weil dieser Name zu viele Zeichen enthält.

Falls Sie die zweite Datei nicht gebrauchen können, benutzen Sie eckige Klammern, um nur die für Sie interessanten Dateien aufzulisten:

```
$ ls inv[13]jig.c
inv1jig.c  inv3jig.c
```

Falls eines der Zeichen innerhalb der eckigen Klammern auf einen Dateinamen passt, wird dieser angezeigt. Sie können auch einen Zeichenbereich definieren:

```
$ ls inv[1-3]jig.c
inv1jig.c  inv2jig.c  inv3jig.c
```

Diesmal werden wieder alle drei Dateien aufgelistet. Der Bindestrich bedeutet: »Suche nach Treffern für alle Zeichen zwischen 1 und 3 einschließlich«. Sie können beliebige Ziffern mit der Angabe 0-9 ansprechen und alle Buchstaben mit der Angabe [a-zA-Z]. In diesem Fall müssen Sie zwei Bereiche definieren, weil die Shell die Groß- und Kleinschreibung

berücksichtigt. Die Sortierreihenfolge, die bei diesen Definitionen benutzt wird, ist übrigens diejenige des *ASCII*-Zeichensatzes.

Nehmen wir an, dass Sie nun auch die *init*-Datei auflisten wollen. Dazu können Sie den Stern einsetzen, weil er eine beliebige Anzahl von Zeichen zwischen dem *inv* und dem *jig* ersetzt:

```
$ ls inv*jig.c
inv1jig.c inv2jig.c inv3jig.c invinitjig.c
```

Der Stern steht tatsächlich für »kein oder mehrere Zeichen«, so dass auch eine Datei mit dem Namen *invjig.c* angezeigt worden wäre, hätte sie existiert.

Anders als die Eingabeaufforderung von *Windows* erlauben die *Linux*-Shells eine beliebige Kombination aus normalen Zeichen und Zeichen mit besonderer Bedeutung. Sie können zum Beispiel alle Quelldateien (*.c*) und Objektdateien (*.o*) auflisten, deren Name eine Ziffer enthält. Das folgende Suchmuster kombiniert alle Möglichkeiten der Namensweiterung, die wir hier vorgestellt haben:

```
$ ls *[0-9]*.[co]
```

Die Erweiterung von Dateinamen kann in Shell-Skripten sehr nützlich sein, wenn Sie nicht genau wissen, mit wie vielen Dateien Sie es zu tun haben. Vielleicht haben Sie zum Beispiel eine Reihe von Logdateien namens *log001*, *log002* usw. zu bearbeiten. Egal wie viele dieser Dateien es gibt – mit dem Suchstring *log** erfassen Sie sie alle.



Die Erweiterung von Dateinamen ist nicht dasselbe wie reguläre Ausdrücke (regular expressions), die von vielen *Utilities* benutzt werden, um Gruppen von Dateien zu bezeichnen. Die Besprechung von regulären Ausdrücken würde den Rahmen dieses Buches sprengen, aber jedes Buch über *Unix-Utilities* geht auch auf dieses Thema ein.

Das Arbeiten mit dem Befehlsspeicher

Durch Drücken der Cursor-Taste Pfeil-nach-oben holen Sie den zuletzt getippten Befehl noch einmal auf den Bildschirm. Der Pfeil nach oben lässt Sie rückwärts durch die bereits eingegebenen Befehle blättern, mit dem Pfeil nach unten blättern Sie vorwärts. Wenn Sie auf der aktuellen Zeile etwas ändern wollen, bewegen Sie sich dazu mit den Tasten Pfeil-nach-rechts und Pfeil-nach-links an die betreffende Stelle.

Dazu sehen wir uns ein Beispiel an. Nehmen wir an, Sie wollten

```
$ mroe .bashrc
bash: mroe: command not found
```

ausführen. Sie haben dabei *mroe* statt *more* getippt. Holen Sie mit Pfeil nach oben den Befehl noch einmal auf den Bildschirm. Drücken Sie dann die Pfeil-links-Taste, bis der Cursor auf dem *o* in *mroe* steht. Sie könnten jetzt das *o* und das *r* löschen und dann korrekt eingeben; aber es gibt eine noch einfachere Lösung: Drücken Sie- STRG T. Damit tauschen Sie die Position der beiden Buchstaben und können anschließend mit RETURN den Befehl ausführen lassen.

Shortcuts für schnelle Tipper

Falls Sie die in dieser Einführung beschriebenen Schritte am Terminal direkt ausprobiert haben, haben Sie es vielleicht schon satt, immer wieder dieselben Eingaben zu machen. Besonders ärgerlich kann das werden, wenn man sich vertippt hat und noch einmal anfangen muss. Dies ist die Stelle, an der die Shell Ihnen das Leben wesentlich erleichtern kann. *Linux* ist dadurch noch nicht so einfach zu bedienen wie eine mausgesteuerte Benutzeroberfläche, aber auf diese Art können Sie Ihre Befehle sehr schnell eingeben.

In diesem Abschnitt besprechen wir das Editieren der Befehlszeile. Diese Tipps gelten für *bash*, *ksh*, *tcsh* und *zsh*. Die Idee hinter der editierbaren Befehlszeile ist ein Speicher, der zum Beispiel die letzten 50 Befehlszeilen enthält, die Sie eingegeben haben. Diesen Speicher können Sie wie ein Textdokument bearbeiten. Sobald Sie RETURN drücken, wird die aktuelle Befehlszeile ausgeführt.

Manpages

In einem *Linux*-System ist alles dokumentiert. Das Wichtigste, was Sie zu wissen haben ist, wie, wo und womit Sie selbst etwas herausfinden können. Wir wollen dieses Konzept beibehalten und stellen Ihnen deshalb jetzt die Online-Hilfe vor, die mit *Linux*-Systemen ausgeliefert wird. Man nennt sie Manual-Pages (Handbuchseiten) oder kurz Manpages.

Die Manpages sind für einen Einsteiger noch nicht das Gelbe vom Ei. Das liegt daran, dass sie kurz gehalten sind und eine ganze Menge *Linux*-Wissen voraussetzen. Jede Manpage behandelt einen bestimmten Befehl, sagt aber nur selten etwas darüber aus, warum Sie gerade diesen Befehl benutzen sollten. Trotzdem sind die Manpages wichtig. Die Befehle auf verschiedenen *Linux*-Systemen können leicht unterschiedlich sein, und dann sind die Manpages die zuverlässigste Informationsquelle zu dem, was auf Ihrem System passiert. (Man muss dem LDP ein großes Lob aussprechen für die unglaublich vielen Stunden, die man dort auf die Erstellung der Manpages verwendet hat.) Informationen zu einem bestimmten Befehl erhalten Sie zum Beispiel mit:

```
$ man ls
```

Die Manpages sind in verschiedene Abschnitte (sections) eingeteilt – abhängig von ihrem Einsatzzweck. Die Benutzerbefehle finden sich in Abschnitt 1, die Systemaufrufe von *Linux* sind in Abschnitt 2 dokumentiert usw. Die Abschnitte 1, 4 (Dateiformate) und 8 (Systemverwaltung) werden Sie am meisten interessieren. Wenn Sie die Manpages online lesen, spielen die Abschnitte keine Rolle, sie können aber als Option bei der Suche nach einem Befehl mit angegeben werden:

```
$ man 1 ls
```

Wenn Sie allerdings einmal ein gedrucktes Handbuch in die Hand nehmen, werden Sie feststellen, dass es nach diesem Schema in Abschnitte eingeteilt ist. Manche Einträge tauchen unter demselben Namen in verschiedenen Abschnitten auf. (So ist zum Beispiel *chmod* sowohl ein Befehl als auch ein Systemaufruf.) Deshalb werden Sie gelegentlich den

Namen einer Manpage zusammen mit der Nummer des entsprechenden Abschnitts in Klammern dahinter sehen, etwa `ls(1)`.

Es gibt allerdings eine Situation, in der Sie die Abschnittsnummer auch auf der Kommandozeile benötigen, nämlich wenn es mehrere Manpages für das gleiche Schlüsselwort gibt (beispielsweise für einen Befehl und eine Systemfunktion gleichen Namens). Wenn Sie zum Beispiel eine Bibliotheksfunktion nachschlagen wollen, der Befehl `man` Ihnen aber Informationen zum Befehl liefert, dann können Sie mit der Abschnittsnummer sagen, was Sie genau sehen möchten.

Sehen Sie sich den Anfang der Manpage an. Die erste Überschrift lautet **NAME**. Darunter steht eine einzeilige Beschreibung. Diese Beschreibungen sind eine wertvolle Hilfe, wenn Sie nicht genau wissen, wonach Sie suchen. Finden Sie einen Begriff, der mit Ihrem Thema zusammenhängt, und geben Sie diesen Begriff zusammen mit dem Befehl `apropos` ein.

```
$ apropos edit
```

Mit diesem Befehl erhalten Sie eine Liste der Manpages, die mit Ihrem Suchbegriff zusammenhängen. Der Algorithmus ist einfach: `apropos` zeigt alle **NAME**-Zeilen an, in denen Ihr Suchbegriff vorkommt.

Viele andere Hilfsprogramme, insbesondere solche für die im Kapitel über die *X-Arbeitsoberfläche anpassen* beschriebenen Desktops, stellen Manpages attraktiv dar.

Auch Manpages werden, genau wie Befehle, manchmal an weniger üblichen Stellen installiert. Es kann zum Beispiel sein, dass Sie einige spezielle Programme im Verzeichnis `/usr/local` installiert haben und die Manpages dazu in `/usr/local/man`. Der Befehl `man` wird nicht automatisch in `/usr/local/man` suchen, so dass Sie statt einer Manpage nur die Meldung »No manual entry« erhalten. Korrigieren Sie das, indem Sie alle Verzeichnisse, in denen Manpages stehen, in der Variable `MANPATH` angeben. Ein Beispiel:

```
$ export MANPATH=/usr/man:/usr/local/man
```

Die Syntax ist dieselbe wie bei `PATH`, die wir weiter oben in diesem Kapitel besprochen haben. Die einzelnen Verzeichnisse werden durch Doppelpunkte getrennt. Falls Sie mit `csh` oder `tcsh` arbeiten, müssen Sie Folgendes eingeben:

```
$ setenv MANPATH /usr/man:/usr/local/man
```

Eine weitere Umgebungsvariable, die sich zu setzen lohnen kann, ist `MANSECT`. Diese bestimmt, in welcher Reihenfolge die Abschnitte der Manpages nach einem Eintrag durchsucht werden. Ein Beispiel:

```
$ export MANSECT="2:3:1:5:4:6:7:8:n:9"
```

sucht zuerst im Abschnitt 2.

Haben Sie bereits einige Manpages gelesen und sind immer noch verwirrt? Die Manpages sind nicht als Einführung in ein neues Thema gedacht. Benutzen Sie die Manpages wieder, wenn Sie sich mit dem System vertraut gemacht haben. Sie werden dann auf diese Online-Hilfe nicht mehr verzichten wollen.

Manpages sind nicht die einzige Informationsquelle. Programme aus dem GNU-Projekt haben oft *Info*-Seiten, die mit dem Programm `info` angezeigt werden können. Um beispielsweise die Info-Seiten zum Befehl `find` anzuzeigen, würden Sie folgenden Befehl verwenden:

```
$ info find
```

Das Programm `info` ist ziemlich kryptisch und hat eine Vielzahl von Optionen. Um es zu erlernen, geben Sie am besten im Programm STRG-H ein und lesen den Hilfetext. Glücklicherweise gibt es auch Programme, die einfacher zu bedienen sind, darunter `tkinfo` und `kdehelp`. Diese verwenden das *X-Window*-System für ihre grafische Oberfläche. Außerdem können Sie Info-Seiten mit dem Programm Emacs oder dem in manchen *Linux*-Distributionen zur Verfügung stehenden Programm `pinfo` lesen, das mehr wie der Webbrowser `lynx` funktioniert. In jüngerer Zeit wird mehr und mehr Dokumentation in Form von HTML-Seiten zur Verfügung gestellt. Diese können Sie mit jedem Webbrowser lesen. Im Webbrowser Konqueror wählen Sie beispielsweise »Datei öffnen...« aus dem Menü »Datei« oder klicken auf den Button mit dem Ordnersymbol, worauf ein ganz gewöhnlicher Dateiauswahldialog geöffnet wird, in dem Sie die gewünschte Datei auswählen können.

Dateiberechtigungen

Eigner- und Zugriffsrechte sind zentrale Punkte der Systemsicherheit. Es ist wichtig, dass Sie diese Berechtigungen richtig vergeben – auch wenn Sie der einzige Benutzer sind –, weil andernfalls merkwürdige Dinge geschehen können. Mit den Dateien, die Benutzer erzeugen und ständig bearbeiten, gibt es in der Regel keine Probleme (trotzdem kann es nicht schaden, wenn Sie die dahinter stehenden Konzepte kennen). Für den Systemverwalter ist die Lage komplizierter. Wenn Sie nur einmal falsche Eigner- oder Zugriffsrechte vergeben, kann das zur Folge haben, dass Sie beispielsweise Ihre E-Mails nicht mehr lesen können. Im Allgemeinen deutet die Meldung

```
Permission denied
```

darauf hin, dass jemand Eigner- und Zugriffsberechtigungen restriktiver als von Ihnen gewünscht vergeben hat.

Die Bedeutung der Eigner- und Zugriffsrechte

Zugriffsrechte geben an, in welcher Weise jemand eine Datei benutzen kann. *Linux* kennt drei Stufen der Berechtigung: *Read* permission (Leseberechtigung) heißt, dass Sie den Inhalt einer Datei ansehen dürfen.

- *Write* permission (Schreibberechtigung) heißt, dass Sie eine Datei ändern und löschen dürfen.
- *Execute* permission (Ausführberechtigung) heißt, dass Sie eine Datei als Programm ausführen dürfen.

Sobald eine Datei erstellt wird, vergibt das System einige Standardberechtigungen, die in den meisten Fällen genügen. So wird zum Beispiel meist die Schreibberechtigung für alle anderen Benutzer gelöscht, so dass Sie die Datei zwar schreiben und lesen können, andere jedoch sie nur lesen können. Falls Sie einen Grund sehen, besonders vorsichtig zu sein, können Sie das System so einrichten, dass andere Leute überhaupt keine Zugriffsrechte bekommen. Einige Utilitys vergeben Berechtigungen, die von den Voreinstellungen abweichen. Wenn zum Beispiel der Compiler ein ausführbares Programm erzeugt, weist er ihm automatisch die Ausführberechtigung zu.

Es gibt allerdings auch Fälle, in denen die voreingestellten Berechtigungen nicht funktionieren. Wenn Sie zum Beispiel ein Shell-Skript oder ein Perl-Programm schreiben, müssen Sie selbst dafür sorgen, dass das Skript oder Programm ausführbar gemacht wird. Wir werden Ihnen weiter unten in diesem Abschnitt zeigen, wie das geschieht, nachdem wir die grundlegenden Konzepte behandelt haben.

Für Verzeichnisse haben die Zugriffsrechte eine andere Bedeutung:

- Leseberechtigung heißt, dass Sie den Inhalt des Verzeichnisses auflisten dürfen.
- Schreibberechtigung heißt, dass Sie in diesem Verzeichnis Dateien hinzufügen und löschen dürfen.
- Ausführberechtigung heißt, dass Sie Zugriff auf die Dateien haben, solange Sie den Dateinamen kennen.

Machen Sie sich über den Unterschied zwischen Lese- und Ausführberechtigung bei Verzeichnissen keine Gedanken – im Grunde genommen sind die beiden identisch. Vergeben Sie entweder beide Berechtigungen oder keine.

Beachten Sie, dass Benutzer mit Schreibberechtigung für ein Verzeichnis Dateien hinzufügen und auch Dateien löschen dürfen – mit der Vergabe der Schreibberechtigung eröffnen Sie dem Benutzer beide Möglichkeiten. Trotzdem gibt es eine Methode, verschiedenen Benutzern den Zugriff auf ein gemeinsames Verzeichnis zu gestatten, ohne dass jeder die Dateien des anderen löschen kann.

Auf einem *Linux*-System gibt es weit mehr Dateitypen als die einfachen Dateien und Verzeichnisse, die wir bisher besprochen haben. Das sind zum Beispiel Gerätedateien (device files), Sockets, symbolische Links usw. Jeder Dateityp hat seine eigenen Regeln in Bezug auf die Zugriffsrechte, aber es ist nicht notwendig, dass Sie alle Details zu jedem Dateityp kennen.

Eigner und Gruppen

Wem werden eigentlich diese Berechtigungen zugestanden? Damit verschiedene Benutzer auf einem System arbeiten können, unterscheidet *Linux* bei den Berechtigungen drei Benutzergruppen: Eigner, Gruppe und Sonstige. Die Benutzergruppe »Sonstige« umfasst alle Benutzer, die Zugang zum System haben, aber nicht Eigner oder Mitglied der Gruppe sind.

Die Idee hinter der Einrichtung einer Gruppe ist, dass man zum Beispiel einem Team von Programmierern den Zugang zu einer Datei gewährt. So könnte eine Programmiererin sich selbst die Schreibberechtigung für ihren Quellcode erteilen, während die Mitglieder ihres Teams mittels Gruppenberechtigung den lesenden Zugriff bekommen. Die Benutzergruppe »Sonstige« bekommt vielleicht gar keine Zugriffsrechte, damit Leute außerhalb Ihres Teams nicht in Ihrem Quellcode herumschnüffeln können. (Glauben Sie wirklich, dass Ihre Programme *so* gut sind?) Zu jeder Datei gehören ein Eigner und eine Gruppe. Der Eigner ist in der Regel der Benutzer, der die Datei angelegt hat. Jeder Benutzer gehört standardmäßig zu einer Gruppe, und diese Gruppe wird jeder Datei zugeordnet, die dieser Benutzer erstellt. Sie können beliebig viele Gruppen einrichten, und jeder Benutzer kann mehreren Gruppen angehören. Indem Sie die Gruppe ändern, die einer Datei zugeordnet ist, können Sie einer beliebig zusammengesetzten Gruppe von Benutzern Zugriff auf die Datei gewähren. Wir werden im Abschnitt »Die Datei group«, genauer auf Gruppen eingehen.

Damit haben wir alle Elemente für unsere Sicherheitsvorkehrungen zusammen: drei Arten der Berechtigung (lesen, schreiben, ausführen) und drei Grade bei den Benutzern (Eigner, Gruppe, Sonstige). Lassen Sie uns einen Blick auf die Berechtigungen für einige typische Dateien werfen.

Die Abbildung zeigt ein typisches ausführbares Programm. Wir haben `ls` mit der Option `-l` eingegeben, um diese Anzeige zu erzeugen.

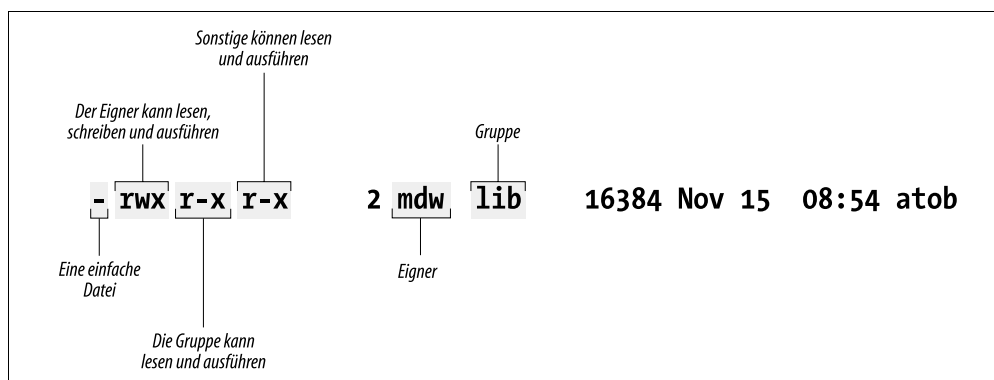


Abbildung 4-5: Eignerschaft und Berechtigungen anzeigen

Zwei nützliche Informationen fallen sofort auf: Der Eigner der Datei ist der Autor dieses Buches und Ihr zuverlässiger Begleiter, `mdw`, und die Gruppe ist `lib` (vielleicht ein Team von Programmierern, die Libraries erstellen). Die wirklich wichtigen Informationen über die Berechtigungen sind allerdings in der Buchstabenfolge links in der Anzeige versteckt.

Das erste Zeichen ist ein Bindestrich und zeigt eine normale Datei an. Die nächsten drei Stellen beziehen sich auf den Eigner – wie man erwarten konnte, hat `mdw` alle drei Berechtigungen. Die folgenden drei Stellen beziehen sich auf die Mitglieder der Gruppe – sie können die Datei lesen (`r`; nicht besonders hilfreich bei einer Binärdatei) und ausführen

(x), aber sie haben keinen schreibenden Zugriff, weil das Feld, das ein w enthalten sollte, stattdessen einen Bindestrich aufweist. Die letzten drei Stellen zeigen die Berechtigungen für »Sonstige« – in diesem Fall haben »Sonstige« dieselben Berechtigungen wie die Mitglieder der Gruppe.

Noch ein Beispiel: Angenommen, Sie fordern ein langes Listing einer C-Quelldatei an, dann könnte das so aussehen:

```
$ ls -l
-rw-rw-r-- 2 kalle kalle 12577 Apr 30 13:13 simc.c
```

Der einzige Unterschied hier besteht darin, dass der Eigner Lese- und Schreibrechte (rw) hat, genau wie die Gruppe. Alle anderen im System haben nur Leserechte.

Nehmen wir jetzt an, dass wir die Datei in ein ausführbares Programm kompilieren. Der Compiler gcc erzeugt die Datei simc:

```
$ gcc -o simc simc.c
$ ls -l
total 36
-rwxrwxr-x 1 kalle kalle19365 Apr 30 13:14 simc
-rw-rw-r-- 1 kalle kalle12577Apr 30 13:13 simc.c
```

Neben den Lese- und Schreib-Bits hat gcc für die ausführbare Datei das »ausführbare« Bit (x) für den Benutzer, die Gruppe und alle anderen gesetzt. Das ist auch sinnvoll so, damit das Programm ausgeführt werden kann:

```
$ ./simc
(Programmausgabe)
```

Als weiteres Beispiel zeigen wir noch ein typisches Verzeichnis:

```
drwxr-xr-x 2 mdw lib 512 Jul 17 18:23 perl
```

An der Stelle ganz links zeigt jetzt ein d an, dass es sich um ein Verzeichnis handelt. Die Ausführberechtigungen sind wieder da, so dass alle Benutzer sich den Inhalt des Verzeichnisses anzeigen lassen können.

Dateien können noch andere, obskure Zustände annehmen, die wir hier nicht besprechen wollen. Lesen Sie die grausigen Details in der Manpage zu ls nach. Für uns wird es jetzt Zeit, die Eigner- und Zugriffsrechte zu ändern.

Eigner, Gruppe und Berechtigungen ändern

Wir haben bereits darauf hingewiesen, dass Sie in der Regel mit den Sicherheitsvorkehrungen auskommen, die das System bietet. Allerdings gibt es auch Ausnahmen, insbesondere für Systemverwalter. Ein einfaches Beispiel: Nehmen wir an, dass Sie für einen neuen Benutzer unterhalb von /home ein Verzeichnis anlegen. Sie müssen diese Aufgabe als root erledigen, aber später den Benutzer zum Eigner des Verzeichnisses machen. Wenn Sie das nicht tun, wird der Benutzer nicht in diesem Verzeichnis arbeiten können! (Glücklicherweise sorgt der Befehl adduser automatisch für die richtige Eignerschaft.)

In ähnlicher Weise haben bestimmte Utilities wie UUCP und News ihre eigenen Benutzer. Niemand wird jemals als *UUCP* oder *News* einloggen, aber diese Benutzer und Gruppen müssen auf dem System vorhanden sein, damit die Utilities sicher funktionieren. Im Allgemeinen besteht der letzte Schritt bei der Installation darin, dass man Eigner, Gruppe und Berechtigungen so anpasst, wie es die Dokumentation vorgibt. Mit dem Befehl `chown` ändern Sie die Eignerschaft einer Datei und mit `chgrp` die Gruppe. Unter *Linux* kann `chown` nur von *root* verwendet werden, um die Eigentümerschaft einer Datei zu ändern, aber jeder Benutzer kann seine Gruppe in eine andere Gruppe ändern, zu der er gehört.

Nachdem Sie also zum Beispiel die Software `sampsoft` installiert haben, können Sie mit folgenden Befehlen sowohl den Eigner als auch die Gruppe auf `bin` ändern:

```
# chown bin sampsoft
# chgrp bin sampsoft
```

Das geht auch in einem Schritt, wenn die Punkt-Notation verwendet wird:

```
# chgrp bin.bin sampsoft
```

Die Syntax für die Änderung der Berechtigungen ist komplizierter. Man nennt die Berechtigungen auch den »Modus« einer Datei, und diesen ändern Sie mit dem Befehl `chmod`. Lassen Sie uns diesen Befehl anhand eines einfachen Beispiels erkunden. Nehmen wir an, dass Sie in Perl oder Tcl ein nettes kleines Programm namens `header` geschrieben haben, das Sie anschließend ausführen wollen.

```
$ chmod +x header
```

Das Pluszeichen bedeutet: »Füge eine Berechtigung hinzu«, und das `x` zeigt an, welche Berechtigung gemeint ist.

Wenn Sie jemandem die Ausführberechtigung entziehen möchten, setzen Sie statt des Plus `+` ein Minuszeichen `-` ein:

```
$ chmod -x header
```

Der eben gezeigte Befehl vergibt die Berechtigung auf allen Ebenen – an den Eigner, die Gruppe und Sonstige. Lassen Sie uns annehmen, dass Sie insgeheim ein Sammler von Software sind, der seine Programme für sich behalten möchte. (Nein, das wäre zu hart. Wir wollen stattdessen annehmen, dass Ihr Skript noch nicht einwandfrei funktioniert und dass Sie andere Leute vor Schaden bewahren möchten, bis das Skript fehlerfrei ist.) Mit folgendem Befehl können Sie die Ausführberechtigung nur für sich selbst vergeben:

```
$ chmod u+x header
```

Alle Angaben vor dem Pluszeichen bezeichnen die Benutzerebene, auf der Sie Berechtigungen vergeben. Die Angaben hinter dem Pluszeichen geben die Art der Berechtigung an. Mit `g` vergeben Sie Rechte an die Gruppe und mit `o` an Sonstige (*others*). Wenn Sie die Ausführberechtigung an sich selbst und die Gruppe erteilen wollen, geben Sie Folgendes ein:

```
$ chmod ug+x header
```

Sie können auch mehrere Berechtigungen gleichzeitig erteilen:

```
$ chmod ug+rxw header
```

Es gibt noch die eine oder andere Abkürzung, die Sie in der Manpage zu `chmod` nachlesen können, falls Sie jemanden beeindrucken möchten, der Ihnen über die Schulter sieht. Allerdings bleibt die Funktionalität des Befehls auf das beschränkt, was wir hier vorgestellt haben.

Obwohl die Syntax zur Angabe des Dateimodus schon ziemlich obskur ist, gibt es noch eine andere, kompliziertere Syntax. Aus verschiedenen Gründen müssen wir sie hier trotzdem beschreiben. Erstens gibt es Situationen, in denen die gerade vorgestellte Syntax, die *symbolischer Modus* genannt wird, nicht ausreicht. Zweitens benutzen die Leute oft die andere Syntax, die *absoluter Modus* genannt wird, in ihrer Dokumentation. Außerdem könnte es ja passieren, dass Sie die absolute Schreibweise einfach bequemer finden.

Um den absoluten Modus zu verstehen, müssen Sie sich auf die Bitebene und die oktale Schreibweise einlassen. Aber keine Bange – so schwierig wird das nicht. Der typische Modus wird durch drei Zeichen dargestellt, die den drei Benutzerebenen entsprechen (Eigner, Gruppe und Sonstige). Diese Ebenen sind in der Abbildung dargestellt. Innerhalb jeder Ebene bezeichnen drei Bits die Berechtigung zum Lesen, Schreiben und Ausführen.

Eigner			Gruppe			Sonstige		
lesen	schreiben	ausführen	lesen	schreiben	ausführen	lesen	schreiben	ausführen
400	200	100	40	20	10	4	2	1

Abbildung 4-6: Die Bits im absoluten Modus

Nehmen wir an, dass Sie sich selbst die Leseberechtigung und niemandem sonst irgendwelche Rechte erteilen wollen. Sie möchten also nur das Bit ansprechen, das mit der Nummer 400 bezeichnet ist. Der `chmod`-Befehl würde dann so aussehen:

```
$ chmod 400 header
```

Um jedermann die Leseberechtigung zu geben, wählen Sie das entsprechende Bit für alle Benutzerebenen: 400 für den Eigner, 40 für die Gruppe und 4 für Sonstige. Der Befehl lautet dann:

```
$ chmod 444 header
```

Das entspricht dem Modus `+r` – abgesehen davon, dass Sie mit diesem Befehl gleichzeitig alle Berechtigungen zum Schreiben und Ausführen aufheben. (Um es genau zu sagen: Es entspricht dem Modus `=r`, den wir weiter oben nicht erwähnt haben. Das Gleichheitszeichen bedeutet: »Vergib diese und nur diese Berechtigung, also hebe alle anderen Rechte auf.«)

Wenn Sie allen Benutzern die Lese- und Ausführberechtigung zuweisen wollen, müssen Sie die Lese- und Ausführbits addieren. Ein Beispiel: 400 plus 100 ist 500. Der komplette Befehl lautet also:

```
$ chmod 555 header
```

und das entspricht dem Modus `=rx`. Wenn jemand vollen Zugriff erhalten soll, steht an der entsprechenden Stelle eine 7 – nämlich $4 + 2 + 1$.

Einen Trick wollen wir Ihnen noch verraten, nämlich wie Sie den Modus voreinstellen können, der jeder Datei zugeordnet wird, die Sie erzeugen (mit einem Texteditor, der Umleitung `>` usw.). Führen Sie dazu entweder den Befehl `umask` aus, oder fügen Sie ihn in die Startdatei Ihrer Shell ein. Diese Datei heißt wahrscheinlich `.bashrc`, `.cshrc` oder so ähnlich – je nachdem, mit welcher Shell Sie arbeiten. (Wir werden die Startdateien im nächsten Abschnitt besprechen.)

Der Befehl `umask` bekommt einen Parameter mit auf den Weg, so wie `chmod` den absoluten Modus mitbekommt. Allerdings ist die Bedeutung der Bits gerade umgekehrt. Sie müssen für den Eigner, die Gruppe und Sonstige festlegen, welche Berechtigungen Sie vergeben wollen, und dann jede einzelne Ziffer von 7 subtrahieren. Das Ergebnis ist eine dreistellige Maske.

Nehmen wir an, dass Sie sich selbst alle Rechte zugestehen wollen (7), die Gruppe soll Lese- und Ausführberechtigung haben (5), und Sonstige erhalten gar keinen Zugriff (0). Ziehen Sie diese Werte von 7 ab, und Sie erhalten 0 für sich selbst, 2 für die Gruppe und 7 für Sonstige. Der Befehl in Ihrer Startdatei muss also lauten:

```
umask 027
```

Eine merkwürdige Vorgehensweise, aber sie funktioniert. Der Befehl `chmod` berücksichtigt die Maske, wenn er Ihren Modus interpretiert. Ein Beispiel: Wenn Sie für eine Datei bei der Erzeugung die Ausführberechtigung vergeben, wird `chmod` Ihnen und der Gruppe die Ausführberechtigung zuteilen, aber alle anderen werden davon ausgeschlossen, weil die Maske für diese Benutzer keine Ausführberechtigung angibt.