# Kapitel 1
# Einleitung

*»Das Beste, was wir von der Geschichte haben,*
*ist der Enthusiasmus, den sie erregt.«*
*– Johann Wolfgang von Goethe*

Das fängt ja gut an. Da will man ein Buch schreiben und weiß nicht einmal, wie man das Thema grob umreißen soll. Dabei könnte alles so einfach sein – wir schreiben doch nur über ein Betriebssystem, das eigentlich keines ist. Aber wir schreiben eben auch über einen Begriff, der nicht mehr nur Technik, sondern mittlerweile so etwas wie eine Philosophie umschreibt.

Neugierig? Zu Recht! Kurz gesagt steht der Begriff Linux heute für ein sehr stabiles, schnelles, freies, Unix-ähnliches Betriebssystem – obwohl Linux streng genommen eigentlich nur der Kern (der sogenannte »Kernel«) dieses Betriebssystems ist.

Doch eins nach dem anderen.

Zunächst einmal ist ein **Betriebssystem** die grundlegende Software auf einem Computer. In den Worten von Andrew Tanenbaum, einer Koryphäe der Betriebssystemforschung, klingt dies so: Ein Betriebssystem ist eine Software, die die Aufgabe hat, *vorhandene Geräte zu verwalten und Benutzerprogrammen eine einfache Schnittstelle zur Hardware zur Verfügung zu stellen*. Ein Betriebssystem ermöglicht es demnach, Hardware anzusteuern, Dateien zu lesen und zu speichern und Programme zu installieren, zu starten und zu verwenden. Ein Betriebssystem verwaltet den Speicher Ihres Computers inklusive sämtlicher Medien und ermöglicht Netzwerk- und Internetzugriff. Betriebssysteme haben dabei ganz unterschiedliche Einsatzgebiete (siehe Kasten).

Doch nun zurück zum Betriebssystem dieses Buches: Linux! Wie erwähnt, können Sie Linux auf einem riesigen Mainframe laufen lassen, aber eben auch auf einem Microcontroller, einem Smart-Home-Gerät oder einem Smartphone.

**Einsatzgebiete von Betriebssystemen**

Im Standardwerk zu Betriebssystemen von Andrew Tanenbaum und Herbert Bos (*Moderne Betriebssysteme*, 2016 erschienen bei Pearson Studium) werden Einsatzgebiete von Betriebssystemen unterschieden. Linux kann die meisten dieser Einsatzgebiete völlig problemlos abdecken. Gehen wir diese Einsatzgebiete einmal durch:

**Mainframe-Betriebssysteme** sind für Großrechner ausgelegt. Großrechner können besonders hohe Anforderungen an den Datendurchsatz und die Speicherkapazität (bspw. mit Tausenden von Festplatten) sowie Zuverlässigkeit (Ausfallsicherheit) gewährleisten. Ihr Einsatzgebiet sind große (Web-)Server, Business-to-Business-Anwendungen, Systeme für Flugbuchungen und große Banken, die Tausende von Transaktionen gleichzeitig abwickeln müssen. Ein *Typischer Vertreter* ist neben Linux auch z/OS.

**Server-Betriebssysteme** müssen ähnliche Anforderungen wie Mainframe-Betriebssysteme erfüllen, allerdings in geringerer Ausprägung. Sie dienen primär der Erbringung von Netzwerkdiensten, etwa zur Bereitstellung eines Webservers (also eines Dienstes zur Auslieferung von Webinhalten). *Typische Vertreter* sind Linux, Unix, BSD und Windows Server.

**Betriebssysteme für PCs und Workstations** kommen auf kleinen Heimcomputern und ihren leistungsfähigeren Geschwistern – den Workstations – zum Einsatz. Sie dienen im privaten und Geschäftsumfeld der Bearbeitung sämtlicher digital unterstützter Aufgaben, von der Bearbeitung einfacher E-Mails bis hin zum computergestützten Entwurf von Bauteilen oder zur Durchführung wissenschaftlicher Simulationen. *Typische Vertreter* sind Windows, macOS, Linux, BSD und Unix.

**Echtzeitbetriebssysteme** werden hauptsächlich in der Fertigung (Steuerung von Produktionsanlagen samt Robotern) und sonstigen Bereichen der Automation eingesetzt, etwa in der Gebäudeautomation. Sie steuern zeitkritische

Prozesse. Wenn beispielsweise sichergestellt werden muss, dass zehnmal pro Sekunde Temperaturwerte elektronisch gemessen werden, dann käme ein Echtzeitbetriebssystem dafür infrage. *Typische Vertreter* sind VxWorks, QNX und Linux (nur bestimmte echtzeitfähige Varianten).

**Betriebssysteme für eingebettete Systeme** kommen auf Microcontrollern und Einplatinensystemen (z. B. Raspberry Pi), Smartphones oder Smart-Home-Steuerungen zum Einsatz. Sie müssen in ressourcenbeschränkten Umgebungen (wenig Speicher, geringe Rechenleistung, kurze Batterielaufzeit) kontextspezifische Dienste (etwa Überwachung und Steuerung eines Gebäudes) ermöglichen. *Typische Vertreter* sind verschiedene Linux-Distributionen, iOS, Android (ebenfalls Linux-basiert).

**Betriebssysteme für Container** sind zwar kein gesonderter Punkt im oben genannten Buch von Tanenbaum und Bos, aber mittlerweile sehr relevant. Containerbetriebssysteme sind auf die Anforderungen moderner Cluster-Systeme wie Kubernetes ausgerichtet. Sie bringen möglichst wenig Ballast mit und sind oft *immutable*, können also nach dem Deployment nicht mehr modifiziert werden. *Typische Vertreter* sind Atomic und CoreOS von Red Hat oder Talos.

**Betriebssysteme für Sensorknoten** werden auf winzigen Sensoren untergebracht, die miteinander und mit einer Basisstation kommunizieren (per Funk). Sie sind batteriebetrieben und müssen besonders energiesparsam sein. Ihr Ziel ist der wartungsarme Langzeitbetrieb (oft im Außenbereich). Ein Beispiel sind Temperatursensoren, die mit einer Smart-Home-Zentrale kommunizieren. *Typische Vertreter* sind TinyOS, RIOT und Contiki.

**Betriebssysteme für Chipkarten** müssen unter extremer Ressourcenknappheit (geringe Rechenleistung und minimale Speicherkapazität) auf Chipkarten laufen. Eingesetzt werden diese Systeme etwa beim elektronischen Bezahlverkehr, weshalb sie trotz ihrer eingeschränkten Umgebung rechenintensive Verschlüsselungsverfahren durchführen müssen. *Typische Vertreter* sind bspw. Security Card Operating System (SECCOS) für die EC-Karte in Deutschland sowie STARCOS und CardOS.

## 1.1 Warum Linux?

Vielleicht stellen Sie sich gerade diese Frage: Warum Linux? Sicher – wenn Sie mit Ihrem gewohnten Betriebssystem zufrieden und einfach »Nutzerin« oder »Nutzer« sind, ist die Motivation gering, hier überhaupt Zeit zu investieren und das Betriebssystem zu wechseln. Wer wäre also ein typischer Linux-Nutzer – und warum?

### 1.1.1 Man muss kein Informatiker sein ...

Linux hat in den Jahrzehnten seit seinem Erscheinen erhebliche Fortschritte hinsichtlich Benutzerfreundlichkeit und Ergonomie gemacht, sodass man kein Informatikstudium absolviert haben muss, um das System bedienen zu können. Freie Software fristet mittlerweile kein Nischendasein mehr, sondern erobert allerorten Marktanteile.

Im Web ist Linux ein Quasistandard für Internetserver aller Art. Wer ein leistungsfähiges und günstiges Hosting der eigenen Website, des eigenen Webshops oder anderer Dienste will, wird hier fündig. Viele kleine, schicke Netbooks werden bereits ab Werk mit Linux geliefert, sind sehr performant und unschlagbar günstig durch die schlichte Abwesenheit jeglicher Lizenzkosten. Aber egal, ob Server- oder Anwendersystem: Die eigentliche Software setzt sich mittlerweile auch ganz unabhängig von Linux auf anderen Betriebssystemen durch. Auch so kommen viele Anwenderinnen und Anwender auf die Idee, sich näher mit Linux auseinanderzusetzen. Es ist nur ein kleiner Schritt von Firefox, Thunderbird und LibreOffice unter Windows zu einem komplett freien Betriebssystem wie Linux.

Außerdem sind einst komplizierte Einstiegshürden heute leicht zu nehmen; es gibt Anleitungen und Hilfe allerorten, professionellen kommerziellen Support genauso wie zahlreiche Websites und Internetforen. Und natürlich gibt es auch dieses Buch. Um Linux effektiv nutzen zu können, muss man heute wirklich kein halber Informatiker mehr sein.

### 1.1.2 ... aber es hilft

Trotzdem ist Linux vor allem für Anwenderinnen und Anwender zu empfehlen, die einfach »mehr« wollen: mehr Möglichkeiten, mehr Leistung oder schlicht mehr Freiheiten. Linux ist eine offene Plattform, bevorzugt genutzt von vielen Entwicklern, Systemadministratorinnen und sonstigen interessierten Power-Usern.

Die Faszination der Technik macht sicherlich einen Teil der Beliebtheit aus. Aber ist das alles? Was haben wohl Google, Instagram, Facebook, YouTube und Co. gemeinsam? Richtig, ihre Server laufen unter Linux. Nach der Lektüre dieses Buches haben Sie zumindest eine Ahnung davon, welche Möglichkeiten Linux besitzt, die Ihnen andere Betriebssysteme so nicht bieten können.

Im nächsten Abschnitt machen wir Sie jedoch erst einmal mit den wichtigsten und gängigsten Begriffen vertraut.

## 1.2 Grundbegriffe: Kernel, Distribution, Derivat

Der Begriff *Linux* bezeichnet dabei eigentlich kein ganzes Betriebssystem, sondern nur die Kernkomponente, den sogenannten *Kernel*. Damit man mit Linux etwas anfangen kann, benötigt man zusätzlich zum Kernel System- und Anwendersoftware.

Diese zusätzliche Software wird daher zusammen mit dem Kernel und einer mehr oder weniger ansprechenden Installationsroutine von sogenannten *Distributoren* zu *Distributionen* zusammengepackt. Zu den bekanntesten Distributionen zählen Linux Mint, (open)SUSE, Fedora, Red Hat Enterprise Linux (RHEL), Gentoo, Debian und Ubuntu.

Ist eine Distribution von einer anderen abgeleitet, so spricht man von einem *Derivat*[1]. So ist beispielsweise das bekannte Ubuntu ein Debian-Derivat, wohingegen Fedora ein Red-Hat-Derivat ist. Derivate bzw. abgespaltete Projekte sind in der Open-Source-Welt recht häufig und eine wichtige Innovationsquelle.

1 Lateinisch von *derivare*, »ableiten«, deutsch: Abkömmling

### 1.2.1 Bekannte Distributionen und Derivate

Im Folgenden werden wir Ihnen einen kleinen Einblick in die aktuelle Welt der Distributionen und Derivate geben. Der Rest des Buches geht dann nur noch in wichtigen Fällen auf Besonderheiten einzelner Distributionen und Derivate ein, da wir Ihnen Wissen vermitteln möchten, mit dem Sie unter jedem System zum Ziel kommen.

In Abschnitt 1.3, »Die Entstehungsgeschichte von Linux«, erfahren Sie mehr über die ersten Derivate und Distributionen.

### 1.2.2 Arten von Distributionen

Es gibt Distributionen, die direkt von einer CD, DVD oder einem USB-Stick gebootet werden können und mit denen Sie *ohne* vorhergehende Installation auf einer Festplatte arbeiten können. Man nennt diese Distributionen *Livesysteme*. Hierzu zählt beispielsweise Knoppix, das die grafische Oberfläche LXDE sowie viele Zusatzprogramme enthält. Neben Knoppix als »reinem« Livesystem bieten auch viele gängige Distributionen wie bspw. Ubuntu Installationsmedien an, die sich auch als Livesystem starten lassen.

Dann wiederum gibt es *Embedded*-Distributionen. Dabei handelt es sich um stark minimierte Systeme, bei denen alle unnötigen Programme und Kernel-Features deaktiviert wurden, um Speicherplatz und Rechenbedarf einzusparen. Sinn und Zweck solcher Systeme ist es, eine Distribution auf sogenannten *eingebetteten Systemen* lauffähig zu machen, die teilweise nur über sehr wenig Hauptspeicher und Rechenleistung verfügen. Es gibt hierfür übrigens auch speziell minimierte C-Bibliotheken, die Sie beispielsweise auf *kernel.org* finden.

Verwendung finden Embedded-Distributionen unter anderem im Router-Bereich. Man kann mit Distributionen wie OpenWRT oder FreeWRT auf diese Weise z. B. Linux-Firewalls auf handelsüblichen Routern installieren.

Die wichtigsten Distributionen sind für den Allzweck-Einsatz auf Heimanwender-Desktops, professionellen Workstations und Servern ausgelegt (und dementsprechend in verschiedenen Ausführungen zu haben). Distributio-

nen wie openSUSE, Fedora, Ubuntu und Gentoo zählen zu diesem Bereich. Sie umfassen sowohl eine Vielzahl von Paketen für das Arbeiten mit verschiedensten Oberflächen-Systemen als auch Serversoftware, Entwicklerprogramme, Spiele und was man sonst noch alles gebrauchen kann.

Darüber hinaus gibt es Security-Distributionen/-Derivate, die speziell darauf ausgelegt sind, eine besonders sichere Umgebung für sensible Daten oder den Schutz von Netzwerken zu bieten. Hierzu zählen Distributionen wie Hardened Gentoo, die im Unterschied zu anderen Distributionen oft modifizierte Kernels und eine minimalistische Softwareauswahl zur Reduktion der Angriffsoberfläche anbieten. Solche Distributionen sind auch für den Einsatz als Firewall/VPN-System geeignet, doch es gibt auch spezielle Distributionen, die hierfür optimiert sind und beispielsweise keine gehärteten Kernels benutzen. Hierzu zählen das bereits erwähnte OpenWRT und seine Derivate.

Im Security-Kontext gibt es auch Distributionen wie Kali Linux, die vor allem Anwendungen zur Identifikation von Sicherheitslücken, zur Durchführung von Penetrationstests und zur digitalen Forensik mitbringen.

Es gibt noch viele weitere spezialisierte Linux-Distributionen. Beispielsweise werden spezielle Distributionen mit wissenschaftlichen Programmen für den Forschungsbereich erstellt. Schauen Sie sich bei Interesse doch einmal die Distribution Scientific Linux unter *www.scientificlinux.org* an.

Unter *distrowatch.com* finden Sie Übersichten zu einer Vielzahl bekannter Distributionen und Derivate.

Es gibt also offensichtlich viel Auswahl. Aber was ist die richtige Distribution für Sie? Für einen allerersten Eindruck eignet sich oft ein Livesystem – werfen Sie also vielleicht einen Blick auf Ubuntu, Fedora oder Linux Mint. Sie sind herzlich zum Ausprobieren eingeladen!

## 1.3 Die Entstehungsgeschichte von Linux

Linux übernahm diverse Konzepte und Eigenschaften des Betriebssystems *Unix*. Daher beschäftigen wir uns an dieser Stelle zunächst einmal mit der Entstehungsgeschichte von Unix. Wir beginnen dazu mit einem Rückblick auf die graue Frühzeit der Informatik.

### 1.3.1 Die Entstehung von Unix

In den 1960er Jahren wurden die ersten großen Softwaresysteme gebaut.[2] Zu dieser Zeit, nämlich im Jahr 1965, begannen BELL, General Electric und das MIT, an einem System namens MULTICS (**Mult***iplexed* **I***nformation and* **C***omputing* **S***ystem*) zu arbeiten. Als allerdings feststand, dass dieses Vorhaben scheitern würde, stieg BELL aus.

#### Die Raumfahrt

Als 1969 das Apollo-Raumfahrtprogramm der USA im Mittelpunkt der Aufmerksamkeit stand, begann Ken Thompson (BELL) mit der Entwicklung einer MULTICS-Variante für zwei Benutzer. Dieses System entwickelte er für den Computer PDP-7 des Herstellers DEC. Sein Ziel war es, raumfahrtbezogene Programme zu entwickeln, um Orbit-Berechnungen für Satelliten, Mondkalender und Ähnliches zu realisieren. Das Grundprinzip von MULTICS wurde dabei übernommen und so bekam das daraus resultierende Betriebssystem beispielsweise ein hierarchisches Dateisystem.

Brian Kernighan nannte dieses System spöttisch *UNICS* (von *uniplexed*). Erst später benannte man es aufgrund der Begrenzung für die Länge von Dateinamen auf der Entwicklungsplattform GECOS in *UNIX* bzw. *Unix* um.[3]

Ursprünglich waren alle Unix-Programme in der Programmiersprache Assembler geschrieben. Ken Thompson entschied sich später, eine Un-

2 Vgl. W. Brenner et al.: Auf dem Weg zu einer Informatik neuer Prägung in Wissenschaft, Studium und Wirtschaft. In: Informatik-Spektrum, Vol. 40(6), S. 602–606, Springer, 2017.

3 Es sind tatsächlich beide Schreibweisen – »UNIX« sowie »Unix« – gleichermaßen korrekt und werden auch benutzt. Aus Gründen der Lesbarkeit haben wir uns in diesem Buch für die Schreibweise »Unix« entschieden.

terstützung für die Sprache FORTRAN zu entwickeln (ein sogenannter *FORTRAN-Compiler*), da Unix seiner Meinung nach ohne eine solche wertlos wäre. FORTRAN ist (wie C) eine Programmiersprache der dritten Generation und erlaubt, verglichen mit Assembler, das Programmieren auf einer höheren Abstraktionsebene. Nach kurzer Zeit entschied er sich allerdings, eine neue Programmiersprache namens B zu entwickeln, die stark von der Sprache BCPL (*Basic Combined Programming Language*) beeinflusst wurde.

#### Aus B wird C

Da das Team 1971 ein PDP11-System bekam, das byteadressiert arbeitete, entschloss sich der amerikanische Programmierer Dennis Ritchie, aus der wortorientierten Sprache B eine byteorientierte Sprache mit dem schlichten Namen »C« zu entwickeln, indem er unter anderem Typen hinzufügte. Tatsächlich gab es zwischen B und C noch einen Zwischenschritt in Form der Sprache *New B* (NB).[4]

1973 wurde der Unix-Kernel komplett neu in C geschrieben. Dieses neue Unix (mittlerweile in der Version 4) wurde damit auf andere Systeme portierbar. Noch im selben Jahr wurde Unix zu einem Mehrbenutzer-Mehrprozess-Betriebssystem (Multiuser-Multitasking) weiterentwickelt und der Öffentlichkeit vorgestellt. Auf diesem System konnten nun mehrere Benutzer gleichzeitig unterschiedliche Programme laufen lassen. Da C gleichzeitig eine sehr portable, aber auch systemnahe Sprache war, konnte Unix recht gut auf neuen Plattformen implementiert werden, um dann auch dort performant zu laufen. Die Vorteile einer Hochsprache wurden hier deutlich: Man braucht nur einen Übersetzer auf einer neuen Hardwareplattform, und schon kann der Code mit nur wenigen Änderungen übernommen werden.

Ein Jahr später, also 1974, erschien ein gemeinsamer Artikel mit dem Titel *The UNIX Time-Sharing System* von Dennis Ritchie und Ken Thompson im Fachblatt *Communications of the ACM*, in dem sie auf die Entstehungsgeschichte und die Wurzeln von Unix eingingen, die eben nicht völlig neu waren, sondern auf bestehenden Systemen basierten. So stammen Grundkonzepte etwa aus dem Berkeley Timesharing System (sogenanntes *Forking*, d. h. das

4 Vgl. D. Ritchie: The Development of the C Language. In: ACM SIGPLAN Notices, Vol. 28(3), S. 201–208, ACM, 1993.

Konzept zur Erzeugung neuer Prozesse), dem bereits erwähnten MULTICS (Konzepte der Systemaufrufe und der Shell) und TENEX oder wurden zumindest durch diese beeinflusst:

»Der Erfolg von Unix liegt nicht so sehr in [...] Innovationen als vielmehr darin, dass sorgfältig ausgewählte Ideen zur vollen Blüte gebracht wurden.« (eigene Übersetzung aus *The UNIX Time-Sharing System*)

Ken Thompson und Dennis Ritchie erhielten 1998 von Bill Clinton die *National Medal of Technology* der USA für die Entwicklung von Unix und C. 1983 erhielten Thompson und Ritchie den Turing Award – die bedeutendste Auszeichnung der Informatik und für Informatiker prinzipiell so wertig wie der Nobelpreis. Weitere wichtige Bücher mit Beteiligung dieser Autoren sind beispielsweise *The Unix Programming Environment* von Rob Pike und Brian Kernighan sowie *The Unix time-sharing system* von Dennis Ritchie. Zudem haben Ritchie und Kernighan mit ihrem Buch *The C Programming Language* eines der bedeutendsten Werke der Informatik verfasst.

#### Die Entstehung der Unix-Derivate

1977 nahm man dann auch die erste Implementierung auf einem Nicht-PDP-System vor, nämlich auf einem Interdate 8/32. Dies regte weitere Unix-Portierungen durch Firmen wie HP und IBM an, und die Unix-Entwicklung begann, sich auf viele Abkömmlinge, sogenannte *Derivate*, auszuweiten.

Die Unix-Variante von AT&T wurde 1981 mit derjenigen von BELL zu einem einheitlichen »Unix-System III« kombiniert. 1983 kündigte BELL das »System V« an, das primär für den Einsatz auf VAX-Systemen an Universitäten entwickelt wurde. Im Jahr darauf annoncierte AT&T die zweite Version von System V. Die Anzahl der Unix-Installationen war bis dahin auf ca. 100.000 angestiegen. 1986 erschien System V, Release 3. Schließlich wurde 1989 System V Release 4 (SVR4) freigegeben, das noch heute als Unix-Standard gilt.

### 1.3.2 BSD wird ins Leben gerufen

Neben SVR4-Unix gab es eine Entwicklung von BSD-Unix, auf deren Darstellung wir hier natürlich keineswegs verzichten möchten. Schließlich haben wir der BSD-Implementierung der sogenannten *TCP/IP-Protokolle* (so

bezeichnet man die Kommunikationsprotokolle für das Internet) mehr oder weniger das heutige Internet zu verdanken.

Bereits 1974 verteilte AT&T Quellcodelizenzen an einige Forschungsinstitute. Auch das *Computing Sciences Research Center* (CSRC) der Bell Labs bekam solch eine Lizenz.

In Berkeley entwickelte ein Kreis von Programmierern der dortigen Universität in den folgenden Jahren einen neuen Code und nahm Verbesserungen gegenüber AT&T-Unix vor, wonach 1977 *1BSD*, die erste *Berkeley Software Distribution*, von Bill Joy zusammengestellt wurde. Im darauffolgenden Jahr wurde *2BSD* veröffentlicht, das über neue Software und Verbesserungen verfügte.

1979 beauftragte die *Defense Advanced Research Projects Agency* (DARPA) der amerikanischen Regierung die *Computer Systems Research Group* (CSRG) der University of California, Berkeley, die Unix-Referenzimplementierung der Protokolle für das ARPANET, den Vorläufer des Internets, zu entwickeln. Die CSRG veröffentlichte schließlich das erste allgemein verfügbare Unix namens 4.2BSD, das unter anderem eine Integration der Kommunikationsprotokolle für das Internet (der oben erwähnten TCP/IP-Protokolle) aufwies: Damit konnte bereits *sehr* ähnlich über ein Netzwerk kommuniziert werden, wie es heutige Rechner nach wie vor tun.[5] Außerdem wurde ein neues Dateisystem eingeführt, nämlich das *Berkeley Fast Filesystem* (FFS).

Somit kann dieses BSD-Derivat als Urvater des Internets angesehen werden. Durch die Integration von TCP/IP und der Berkeley-Socket-API (das ist eine Programmierschnittstelle für die Netzwerkkommunikation) wurden Standards geschaffen bzw. geschaffene Standards umgesetzt, die für das spätere Internet essenziell sein sollten. Wenn man bedenkt, dass selbst heute noch ebendiese Berkeley-Socket-API als Standard in allen netzwerkfähigen Betriebssystemen implementiert ist, wird erst das volle Ausmaß der Bedeutung dieser Entwicklungen deutlich.

1989 entschloss man sich dazu, den TCP/IP-Code in einer von AT&T unabhängigen Lizenz als *Networking Release 1* (Net/1) zu vertreiben. Net/1 war die erste öffentlich verfügbare Version. Viele Hersteller benutzten den Net/1-Code, um

5 Vgl. S. Wendzel: IT-Sicherheit für TCP/IP- und IoT-Netzwerke. Springer, 2. Aufl., 2021.

TCP/IP in ihre Systeme zu integrieren. In *4.3BSD Reno* wurden 1990 noch einmal einige Änderungen am Kernel und an den Socket-APIs vorgenommen, um OSI-Protokolle zu integrieren.

Im Juni 1991 wurde *Net/2* herausgegeben, das komplett neu und unabhängig vom AT&T-Code entwickelt wurde. Die wichtigsten Neuerungen von Net/2 waren eine komplette Neuimplementierung der C-Bibliothek und vieler Systemprogramme sowie die Ersetzung des AT&T-Kernels bis auf sechs Dateien. Nach einiger Zeit hatte William Frederick Jolitz auch die letzten sechs Dateien neu entwickelt. Er stellte ein vollständiges, bootbares Betriebssystem zum freien FTP-Download zur Verfügung. Es trug den Namen *386/BSD* und lief auf Intel-Plattformen.

Die *Berkeley Software Design, Inc.* (BSDI) brachte 1991 mit *BSD/OS* eine kommerzielle Weiterentwicklung von 386/BSD auf den Markt. Diese Version konnte für den Preis von 999 USD erworben werden. BSD/OS konnte sich über mehr als zehn Jahre auf dem Markt halten, insbesondere als Betriebssystem für Server.

1992 entstand außerdem das freie *NetBSD*-Projekt, das es sich zum Ziel setzte, 386/BSD als nicht kommerzielles Projekt weiterzuentwickeln und es auf möglichst vielen Plattformen verfügbar zu machen.

Nachdem die Unix System Laboratories, eine Tochtergesellschaft von AT&T, BSDI wegen einer Urheberrechtsverletzung verklagt hatten, mussten einige Veränderungen am Net/2-Code vorgenommen werden. Daher mussten 1994 alle freien BSD-Projekte ihren Code auf den von 4.4BSD-Lite (auch als Net/3 bezeichnet) umstellen. Mit der Veröffentlichung von *4.4BSD-Lite2* im Jahr 1995 wurde die CSRG aufgelöst. Eingestellt wurde auch die Entwicklung von BSD/OS – allerdings erst Anfang des neuen Jahrtausends. Die vier großen zu diesem Zeitpunkt existierenden BSD-Derivate *FreeBSD*, *OpenBSD* sowie das bereits erwähnte NetBSD (und ihre jeweiligen Ableger) werden bis heute gepflegt und ständig weiterentwickelt.

### 1.3.3 Die Geburtsstunde von Linux

Wir schreiben das Jahr 1991, und Linus Torvalds kann die Version Linux 0.02 bereits in der Newsgroup `comp.os.minix` posten. Hier die Originalnachricht:

```
From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: What would you like to see most in minix?
Date: 25 Aug 91 20:57:08 GMT

Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big
and professional like gnu) for 386(486) AT clones. This has been
brewing since april, and is starting to get ready. I'd like any
feedback on things people like/dislike in minix, as my OS
resembles it somewhat (same physical layout of the file-system
(due to practical reasons) among other things).

I've currently ported bash(1.08) and gcc(1.40), and things seem
to work. This implies that I'll get something practical within
a few months, and I'd like to know what features most people
would want.  Any suggestions are welcome, but I won't promise
I'll implement them :-)

                Linus (torvalds@kruuna.helsinki.fi)

PS.  Yes - it's free of any minix code, and it has a multi-
threaded fs. It is NOT protable (uses 386 task switching etc),
and it probably never will support anything other than AT-
harddisks, as that's all I have :-(.
```

**Listing 1.1** Linus Torvalds' Posting an comp.os.minix

Zu diesem Zeitpunkt liefen bereits wichtige Programme wie der GNU-C-Compiler (`gcc`, dient der Übersetzung von Softwarequellcode der Programmiersprache C in ein ausführbares Programm), die `bash` (dient der Eingabe von Befehlen) und `compress` (dient zum Komprimieren von Dateien) auf diesem System.

Im Folgejahr veröffentlichte Torvalds Version 0.12 auf einem öffentlichen FTP-Server, wodurch die Anzahl derjenigen stieg, die an der Systementwicklung mitwirkten. Im selben Jahr wurde das Diskussionsforum `alt.os.linux` gegründet. Es handelt sich dabei um eine sogenannte *Newsgroup* – so hießen

die Foren im *Usenet*.[6] So wie das Internet mit BSD groß wurde, ist Linux also ein Kind des Internets.

Im Jahr 1994 wurde Version 1.0 veröffentlicht. Der Kernel verfügte zu diesem Zeitpunkt schon über Netzwerkfähigkeit. Außerdem portierte das XFree86-Projekt seine grafische Oberfläche – das X-Window-System – auf Linux. Das wohl wichtigste Ereignis in diesem Jahr war jedoch, dass Torvalds auf die Idee kam, den Kernelcode unter der GNU General Public License zu veröffentlichen. Zwei Jahre später war Linux 2.0 zu haben. Erste Distributionen stellten ihre Systeme nun auf die neue Version um, darunter auch Slackware mit dem »96«-Release.

1998 erschien die Kernelversion 2.2. Von da an verfügte Linux auch über Multiprozessorsupport. Im Jahr 2001 erschien schließlich Version 2.4 und im Dezember 2003 Version 2.6. 2011 kam Version 3.0 heraus. Nach einer Meinungsumfrage auf der Plattform *Google+* wurde die Version im Jahr 2015 schließlich von 3.19 nicht auf 3.20, sondern auf 4.0 erhöht. Linux 5.0 erschien schließlich 2019 und Linux 6.0 Ende 2022. Die zum Zeitpunkt des Schreibens aktuelle Version 6.9.1 wurde im Mai 2024 veröffentlicht.

**Empfehlenswerte Bücher zur Geschichte der Betriebssysteme**

- A. S. Tanenbaum, H. Bos: **Moderne Betriebssysteme**. 4. Auflage, Pearson Studium, 2016.
- B. Hansen (Hrsg.): **Classic Operating Systems. From Batch Processing to Distributed Systems**. Springer, 2001. (Dieser Titel ist für anspruchsvolle Leserinnen und Leser geeignet, die selektierte englische Originalaufsätze einiger Koryphäen lesen möchten.)

### 1.3.4 Die Kernelversionen

Das Versionsschema der Kernelversionen ist seit Kernel 3.0 wie folgt: Alle paar Monate wird die erste Stelle nach dem Punkt (3.**x**) erhöht, kleine Ände-

6 Wir haben in den ersten zehn Jahren unserer Linux-Nutzung tatsächlich sehr viel Zeit in derlei Foren (und dem Internet Relay Chat) verbracht.

rungen (Fehlerbehebungen und Sicherheitsupdates) werden mit der zweiten Stelle hinter dem Punkt angegeben (3.x.**y**).

Entwicklerversionen des Kernels gibt es nur in einem separaten Entwicklungszweig und der Entwicklungsprozess läuft dabei folgendermaßen ab: Es gibt ein Zeitfenster, innerhalb dessen neue Features in den Kernel eingebaut werden. Anschließend werden diese Features optimiert und auf ihre korrekte Funktionsweise hin überprüft. Steht fest, dass alle neuen Features ordentlich funktionieren, wird schließlich eine neue Kernelversion nach dem oben genannten Schema herausgegeben.

Sollten Sie mal jemanden treffen, der Ihnen von irgendwelchen komischen Versionen à la »Linux 20.0« erzählen will, haben Sie ein seltenes Exemplar der Spezies Mensch gefunden, die offensichtlich die falschen Bücher liest. Diese bringen nämlich die Versionen der Distributionen und des Kernels durcheinander.

Aber keine Angst: Aktuelle Distributionen enthalten natürlich immer die Stable-Version des Kernels. Einige Anbieter von Distributionen beschäftigen auch Kernelentwickler, die die Features des (eigenen) Kernels erweitern, um den Anwenderinnen und Anwendern beispielsweise zusätzliche Treiber zur Verfügung zu stellen.

Wie bereits erwähnt, gibt es Distributionen, die einen modifizierten Kernel enthalten, und solche, die den unmodifizierten Kernel nutzen. Dieser unmodifizierte Kernel ohne zusätzliche Patches wird auch als *Vanilla-Kernel* bezeichnet.

Auf *kernel.org* erfahren Sie zu jedem Zeitpunkt etwas über die aktuellen Versionen des Linux-Kernels.

#### Das Linux-Maskottchen

Da Linus Torvalds ein Liebhaber von Pinguinen ist, wollte er einen als Logo für Linux haben. Larry Erwing entwarf mit dem Grafikprogramm GIMP einen Pinguin (siehe Abbildung 1.1). Er gefiel Torvalds, und fertig war *Tux*, der übrigens für **T**orvald's **Uni**x steht.

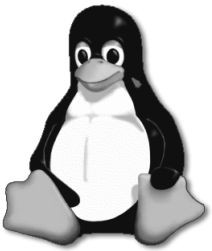

**Abbildung 1.1** Tux

### 1.3.5 Stallman und das GNU-Projekt

Im Jahre 1992 wurde Linux unter die GNU General Public License (GPL) gestellt, die 1989 von Richard Stallman erarbeitet worden war. Stallman gründete 1983 das GNU-Projekt, das freie Software und Kooperationen zwischen den Entwicklerinnen und Entwicklern befürwortet. Außerdem ist Stallman Entwickler von bekannten Programmen wie dem Emacs-Editor oder dem GNU-Debugger.

Stallman ist noch heute einer der wichtigsten – wenn nicht *der* wichtigste – Verfechter des Open-Source-Gedankens. Er arbeitete in den 70er Jahren am Massachusetts Institute of Technology (MIT) in einem Labor für künstliche Intelligenz und kam dort zum ersten Mal mit Hackern in Kontakt. Die dortige Arbeitsatmosphäre gefiel ihm so gut, dass er die spätere Auflösung die Aufspaltung des Labors sehr bedauerte. Zudem wurde Software immer mehr in binärer Form und weniger durch Quelltexte vertrieben, was Stallman ändern wollte. Aus diesem Grund schuf er das GNU-Projekt, dessen Ziel die Entwicklung eines kompletten freien Betriebssystems war.[7] Den Kern dieses Betriebssystems bildet heutzutage meistens Linux. Umgekehrt sind die wichtigsten Komponenten der Userspace-Software von Linux seit Beginn GNU-Programme wie der `gcc`. Richard Stallman versuchte daher später, den Namen GNU/Linux durchzusetzen, was ihm aber nur bedingt gelang.

Dass Linux selbst eigentlich nur den Kernel umfasst, haben wir bereits angesprochen. Die für den Betrieb nötige Systemsoftware kommt in erster Linie

7 Die Abkürzung GNU steht für »GNU is not Unix« und ist rekursiv.

vom bereits erwähnten GNU-Projekt (*http://www.gnu.org*). Diese Initiative gibt es seit 1984 und damit viel länger als Linux selbst. Das Ziel war von Anfang an, ein völlig freies Unix zu entwickeln, und mit Linux hatte das Projekt seinen ersten freien Kernel. Somit ist auch die Bezeichnung GNU/Linux für das Betriebssystem als Ganzes gebräuchlich.

Was aber ist eigentlich *freie Software*? Wenn man ein Programm schreibt, so besitzt man an seinem Quelltext ein Urheberrecht wie ein Buchautor. Die resultierende Software kann verkauft werden, indem man den Käuferinnen und Käufern durch eine Lizenz gewisse Nutzungsrechte einräumt. Alternativ kann man aber auch festlegen, dass das eigene Programm von anderen kostenlos benutzt werden kann. Gibt man sogar den eigenen Quellcode frei, so spricht man von *offener Software*.

Im Linux- und BSD-Umfeld gibt es unterschiedliche Lizenzen, die mit teilweise besonderen Bestimmungen ihr jeweils ganz eigenes Verständnis von »Freiheit« verdeutlichen.

#### Die GPL

Linux steht wie alle GNU-Projekte unter der *GNU Public License*, der GPL. Laut dieser Lizenz muss der Quellcode eines Programms frei zugänglich sein. Das bedeutet jedoch nicht, dass GPL-Software nicht verkauft werden darf. Mehr dazu finden Sie unter *www.gnu.org/philosophy/selling.de.html*.

Selbst bei kommerziellen Distributionen zahlt man allerdings oft nicht für die Software selbst, sondern für die Zusammenstellung der Software, die eventuell vorhandenen Handbücher und den (Installations-)Support.

> Die GPL stellt damit Programme unter das sogenannte *Copyleft*: Verändert man ein entsprechendes Softwareprojekt, so muss das veränderte Ergebnis wieder frei sein. Man darf zwar Geld für ein GPL-basiertes Produkt nehmen, muss aber den Sourcecode samt den eigenen Änderungen weiterhin frei zugänglich halten.

Somit bleibt jede einmal unter die GPL gestellte Software immer frei – es sei denn, alle jemals an einem Projekt beteiligten Entwicklerinnen und Ent-

wickler stimmen einer Lizenzänderung zu. Bei großen Softwareprojekten wie dem Linux-Kernel mit vielen Tausend Beteiligten ist das undenkbar.

#### Die BSD-Lizenz

Im Unterschied zu der im Linux-Umfeld verbreiteten GPL verzichtet die von BSD-Systemen verwendete Lizenz auf ein Copyleft. Man darf zwar den ursprünglichen Copyright-Vermerk nicht entfernen, doch darf entsprechend lizenzierte Software durchaus Ausgangspunkt für proprietäre, kommerzielle Software sein. Die BSD-Lizenz ist also weniger streng als die GPL, aber aufgrund der möglichen freien Verteilbarkeit und Veränderbarkeit immer noch freie Software.

#### Weitere freie Projekte

Natürlich gibt es freie Software nicht nur vom GNU-Projekt oder von dem BSD-Entwicklerteam. Jeder kann für eigene Softwareprojekte die GPL oder die BSD-Lizenz verwenden. Natürlich kann man – wie beispielsweise das Apache-Projekt – auch eigene Open-Source-Lizenzen mit besonderen Bestimmungen entwickeln. Jedoch haben bekannte Lizenzen den Vorteil, dass sie in der Community auch anerkannt sind und einen guten Ruf genießen oder – wie die GPL – sogar bereits von einem deutschen Gericht in ihrer Wirksamkeit bestätigt wurden.

### 1.3.6 Geschichte der Distributionen

#### Bootdisk und Rootdisk

Ursprünglich war nur der Quellcode des Linux-Kernels verfügbar, der von erfahrenen Unix-Anwenderinnen und -Anwendern übersetzt und gebootet werden konnte. Mit einem blanken, bootbaren Kernel konnte man aber nicht sonderlich viel anfangen, wenn man nicht wusste, wie die zugehörigen Benutzerprogramme, mit denen man dann etwa seine Mails lesen konnte, installiert werden. Aus diesem Grund stellte Linus Torvalds zunächst zwei Disketten-Images im Internet zur Verfügung, die besonders Anwendern alter Slackwareversionen bekannt sein dürften: die Boot- und die Rootdisk. Von der Bootdisk war es möglich, den Linux-Kernel beim Start des Rechners zu

laden. War der Ladevorgang abgeschlossen, musste man die Rootdisk einlegen. Diese enthielt Basisanwendungen und machte das Linux-System für Anwenderinnen und Anwender ohne größere Vorkenntnisse zugänglich.

#### SLS

Die erste halbwegs benutzbare Linux-Distribution nannte sich SLS (*Softlanding Linux System*) und wurde 1992 von Peter McDonald erstellt. Da SLS viele Fehler enthielt, entwickelten zwei Personen basierend auf SLS jeweils eine neue Distribution, die beide die ältesten heute noch aktiven Distributionsprojekte darstellen.

#### Slackware und Debian

Der erste Entwickler war Patrick J. Volkerding, der im Juli 1993 Slackware 1.0.0 veröffentlichte. Ian Murdock gab im August 1993 die erste Debian-Version frei. Auf Debian und Slackware basieren zahlreiche der heute aktiven Distributionen (etwa Zenwalk oder Ubuntu).

Die letzte Version von Slackware (15.0) wurde im Februar 2022 veröffentlicht und ist damit zwar noch nicht eingestellt, aber doch eher etwas für Enthusiasten. Debian dagegen hat eine sehr aktive Community, und es werden kontinuierlich drei verschiedene Varianten gepflegt: *stable, testing* und *unstable*. Die *stable*-Release enthält nur stabile Pakete, die über einen längeren Zeitraum mit Updates versorgt werden. Oft nutzt man Pakete aus diesem Zweig für die Serverinstallation, da hier Sicherheit in der Regel vor Aktualität geht. Im *testing*-Zweig findet man alle Pakete, die in das zukünftige *stable*-Release eingehen sollen. Hier können die Pakete ausführlich getestet und für die Veröffentlichung vorbereitet werden. Der *unstable*-Zweig ist trotz seines Namens nicht zwangsläufig instabil. Stattdessen findet man hier immer die aktuellen Pakete, die so oder anders frühestens in das übernächste Debian-Release Einzug halten werden. Aufgrund der Aktualität können wir trotz manchmal auftretender Probleme diesen Zweig vor allem für Workstation-Installationen empfehlen.

### Red Hat und Fedora

Im November 1994 wurde die Distribution Red Hat begründet, die auf Slackware basierte, aber ein eigenes Paketformat (RPM) bekam. Auf ihr basieren die heutigen Distributionen Red Hat Enterprise Linux und Fedora.

### SuSE, openSuSE und SUSE

Ebenfalls 1994 wurde die Distribution SuSE Linux veröffentlicht. SuSE Linux war jahrelang die in Deutschland populärste Linux-Distribution, zusammengestellt durch die Software- und System-Entwicklungsgesellschaft mbH aus Nürnberg. Mit ihr gab es (neben Red Hat Linux) eine einfach zu bedienende Distribution mit großer Paketauswahl. Für den deutschen Markt war zudem die ISDN-Unterstützung sehr bedeutsam. Später wurde die Firma von Novell übernommen und der Name SuSE komplett in Großbuchstaben geschrieben, also »SUSE«. Heute gibt es die von der Community mitgepflegte Variante openSUSE sowie die Enterprise-Varianten SLES und SLED (*SUSE Linux Enterprise Server/Desktop*) für Unternehmen.

### Knoppix

Knoppix von Klaus Knopper war die erste wirklich bekannte Distribution, die sich direkt von CD starten und ohne weitere Installation komplett benutzen ließ. Diese Distribution wird nach wie vor aktiv weiterentwickelt und setzt den LXDE-Desktop ein.

### Gentoo

Gentoo Linux basiert auf einem BSD-artigen Ports-System (man spricht bei Gentoo allerdings nicht von Ports, sondern von *Ebuilds*), also einem System, bei dem Software erst kompiliert werden muss. Die Hauptvorteile von Gentoo liegen in der großen Anpassbarkeit und der Performance der für den eigenen Prozessor optimierten Software. Gentoo richtet sich eher an fortgeschrittene User und bietet mittlerweile neben dem Linux-Kernel auch einen FreeBSD-Kernel an (dies gilt übrigens auch für einige andere Distributionen).

#### Ubuntu

Eine der mittlerweile populärsten Linux-Distributionen ist das auf Debian basierende Ubuntu. Das Ubuntu-Projekt verfolgt das Ziel, eine möglichst einfach zu bedienende, an den Anwenderinnen und Anwendern orientierte Distribution zu schaffen. Die Versionsnummern von Ubuntu setzen sich übrigens aus dem Erscheinungsjahr und -monat zusammen. Die Version 24.04 erschien entsprechend im April 2024. Der Distributor gibt zudem sogenannte *LTS-Versionen* (*Long-Term Support*) heraus, die besonders lang unterstützt werden.

#### Linux Mint

Linux Mint ist eine populäre, auf dem jeweils aktuellen LTS-Release von Ubuntu basierende Distribution. Sie ist in mehreren Varianten (»Editionen«) verfügbar, die sich hauptsächlich in der standardmäßig installierten Desktop-Umgebung unterscheiden. Neben dem normalen Linux Mint gibt es auch eine auf Debian basierende Variante, LMDE (*Linux Mint Debian Edition*).

#### Arch Linux und Manjaro Linux

Arch Linux und das darauf basierende, etwas einsteigerfreundlicher ausgelegte Manjaro Linux benutzen ein Rolling-Release-Modell ohne feste Versionszyklen. Manjaro bietet dabei im Gegensatz zu Arch Linux wieder diverse Editionen mit unterschiedlichen vorinstallierten Desktop-Umgebungen an. Manjaro ist aktuell eine der populärsten Linux-Distributionen.

## 1.4 Zusammenfassung

In diesem Kapitel haben Sie grundlegende Begrifflichkeiten und die Geschichte rund um Linux gelernt. Sie wissen, dass ein Kernel allein noch kein Betriebssystem macht. Sie kennen die wichtigsten Distributionen und ihre Geschichte.

## 1.5 Aufgaben

### Kernel.org

Besuchen Sie die Website *http://kernel.org*. Informieren Sie sich. Was ist die letzte stabile Version des Linux-Kernels?

### Ubuntu-Dokumentation

Finden Sie die offizielle Dokumentation zur freien Linux-Distribution Ubuntu. Stöbern Sie etwas.

# Kapitel 7
# Werkzeuge für die Konsole

*»... the Linux philosophy is laugh in the face of danger. Oops. Wrong one. Do it yourself. That's it. «*
*»... die Linux-Philosophie ist es, der Gefahr ins Gesicht zu lachen. Ach nee, falsch. Bau es selbst. Das ist es.«*
*– Linus Torvalds*



In diesem Kapitel werden wir wichtige Tools für die Konsole besprechen. Diese sehr unterschiedlichen Tools können nicht nur in Shellskripten eingesetzt werden, sondern erleichtern auch die tägliche Arbeit mit Dateien, insbesondere Textdateien (z. B. Konfigurationsdateien oder Textdateien mit Messwerten). Die in diesem Kapitel vorgestellten Kommandos ergänzen die Kommandos der vorherigen Kapitel.

## 7.1 touch – Zeitstempel von Dateien setzen

Mit dem `touch`-Kommando können Sie die Zeitstempel, sogenannte *Timestamps*, von Dateien anpassen. Genauer gesagt werden der letzte Zugriff auf die Datei sowie die letzte Änderung am Dateiinhalt auf einen neuen Zeitpunkt gesetzt. Ein Zeitpunkt wird dabei in der Form `-t MMDDhhmm` (also Monat, Tag, Stunde, Minute) übergeben. Optional können das Jahr vorangestellt und Sekunden durch einen Punkt getrennt angehängt werden. Der letzte Zugriff wird über den Parameter `-a` (*last access*), die Modifizierung via `-m` (*last modification*) gesetzt.

Das folgende Beispiel illustriert die Nutzung. Achten Sie nur auf das sich ändernde Datum der Datei, das wir neben weiteren Informationen über den `ls`-Parameter `-l` erhalten (die ersten Spalten enthalten Zugriffsrechte sowie Informationen über den Eigentümer und die Dateigröße).

```
$ ls -l Datei.txt
-rw-rw-r-- 1 sw sw 1000 Jan 24 18:03 Datei.txt
$ touch -t 01020304 Datei.txt // Januar, 2. Tag d. Monats, 03:04 Uhr
$ ls -l Datei.txt
-rw-rw-r-- 1 sw sw 1000 Jan  2 03:04 Datei.txt
```

**Listing 7.1** Setzen des »letzten Zugriffs« auf eine Datei

Ein nettes Feature von `touch` ist, dass Sie eine noch nicht existierende, leere Datei erzeugen können, indem Sie `touch <Dateiname>` aufrufen. Dies ist in der Systemadministration ein nettes Mittel, um »mal schnell« eine leere Logdatei zu erstellen.

## 7.2 cut – Dateiinhalte abschneiden

Dateien (und auch Eingaben über die Standardeingabe) können Sie mit dem Programm `cut` auf die eigenen Bedürfnisse zurechtstutzen. Besonders bei der Erstellung von Shellskripten spielen solche Funktionalitäten eine wichtige Rolle, da oftmals Datenströme angepasst werden müssen, um bestimmte Ausgaben zu erreichen.

Doch zurück zu `cut`. `cut` kann die Eingabedaten auf zwei Weisen »abschneiden«: mithilfe von Spalten (`-c`) und mithilfe von Feldern (`-f`). Dabei werden die Nummern der jeweiligen Einheit über Kommata getrennt bzw. mit einem `--` verbunden.

Nun mag dies etwas verwirrend erscheinen, doch wozu gibt es Beispiele? Im Folgenden wollen wir die Datei */etc/hosts* an unsere Bedürfnisse anpassen. Sie besteht aus drei Spalten, die jeweils durch ein Leerzeichen voneinander getrennt sind. (Dies muss nicht zwangsläufig so sein.) Die erste Spalte gibt die IP-Adresse eines Computers, die zweite seinen vollen Domainnamen und die dritte seinen bloßen Hostnamen an. Wir interessieren uns nun ausschließlich für die IP-Adresse und den Hostnamen.

Da die einzelnen Felder durch ein Leerzeichen getrennt sind, geben wir dies über den `-d`-Parameter als »Trennungszeichen« für die Spalten an. Da es sich beim Leerzeichen um ein nicht druckbares Zeichen handelt, »escapen« wir es, indem wir »\ « (beachten Sie das Leerzeichen) schreiben.

```
user$ cut -d\  -f 1,3 /etc/hosts
127.0.0.1 localhost
192.168.0.1 merkur
192.168.0.2 venus
192.168.0.3 erde
192.168.0.4 mars
192.168.0.5 jupiter
192.168.0.6 saturn
192.168.0.7 uranus
192.168.0.8 neptun
```

**Listing 7.2** Beispielanwendung für cut



## 7.3 paste – Dateien zusammenfügen

Nein, `paste` ist nicht – wie einige Leute glauben – das Gegenstück zum `cut`-Programm. `cut` schneidet die Teile, die Sie benötigen, aus einem Text heraus. `paste` fügt jedoch keine Teile ein, sondern fügt ganze Dateien zusammen.

Das Zusammenfügen erfolgt zeilenweise über ein Trennzeichen, das Sie angeben können. Schauen wir uns einmal die Ausgabe des obigen Aufrufs von `cut` an. Dort geben wir die IP-Adressen und die Hostnamen der Rechner im Netzwerk aus. In der folgenden fiktiven Situation gehen wir davon aus, dass die IP-Adressen in der Datei *IPAdressen* und die Hostnamen in der Datei *Hostnames* untergebracht sind. Wir möchten nun eine zeilenweise Zuordnung erstellen, wobei die einzelnen Spalten durch einen Doppelpunkt voneinander getrennt sein sollen.

```
user$ paste -d : IPAdressen Hostnames
127.0.0.1:localhost
192.168.0.1:merkur
192.168.0.2:venus
192.168.0.3:erde
192.168.0.4:mars
192.168.0.5:jupiter
```

**Listing 7.3** Beispiel für das Zusammenfügen zweier Dateien

## 7.4 tac – Dateiinhalt umdrehen

Es könnte vorkommen, dass eine Datei in einer Form vorliegt, die umgekehrt werden muss, beispielsweise als eine Tabelle, die Benutzerdaten von User-ID 1000 bis 10000 enthält, jedoch mit 10000 statt mit 1000 beginnt. In diesem Fall hilft `tac` sehr einfach weiter:

```
user$ tac /etc/hosts
192.168.0.5 jupiter.sun jupiter
192.168.0.4 mars.sun mars
192.168.0.3 erde.sun erde
192.168.0.2 venus.sun venus
192.168.0.1 merkur.sun merkur
127.0.0.1 localhost.sun localhost
```

**Listing 7.4** tac dreht unsere hosts-Datei um.

## 7.5 column – Ausgaben tabellenartig formatieren

Wie es sich aus dem Namen von `column` bereits erahnen lässt, werden mit diesem Programm Ausgaben spaltenweise dargestellt. Übergibt man ihm eine simple Zahlenfolge, so sieht das Ergebnis folgendermaßen aus:

```
$ seq 1 10 | column
1    3    5    7    9
2    4    6    8    10
```

**Listing 7.5** column stellt Zahlen in Spalten dar.

`column` kann in dieser Form beispielsweise dazu verwendet werden, Ausgaben, die zeilenweise aus einer Pipe kommen, spaltenweise darzustellen. Das lässt sich leicht verdeutlichen, wenn wir die Ausgabe von `ls`, die normalerweise auch in Spaltenform zu sehen ist, durch eine Pipe schicken:

```
$ mkdir dir; cd dir; touch a b c d
$ /bin/ls
a  b  c  d
$ /bin/ls | more
a
b
```

```
c
d
$ /bin/ls | more | column
a    b    c    d
```

**Listing 7.6** Aus zeilenweiser Darstellung wieder eine Spaltendarstellung machen

Ausgaben lassen sich auch in tabellarischer Form (also ein Datensatz pro Zeile) darstellen. Dazu muss der Parameter `-t` verwendet werden. Mit `-s` kann, ähnlich wie bei `awk -F`, ein Zeichen angegeben werden, das die einzelnen Attribute teilt.



```
$ head -4 /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
$ head -4 /etc/passwd | column -t -s :
root    x  0  0  root    /root      /bin/bash
daemon  x  1  1  daemon  /usr/sbin  /bin/sh
bin     x  2  2  bin     /bin       /bin/sh
sys     x  3  3  sys     /dev       /bin/sh
```

**Listing 7.7** Die passwd als Tabelle darstellen

Dem Parameter `-s` können Sie auch mehr als ein Zeichen übergeben, da er eine *Zeichenmenge* entgegennimmt. Schreiben Sie diese Zeichen dann ohne Trennung direkt hintereinander.

## 7.6 colrm – Spalten entfernen

Haben Sie mit `column` soeben gelernt, wie Ausgaben tabellenartig dargestellt werden können, so lernen Sie nun, wie Sie mit `colrm` einzelne Spalten von Texten entfernen. Obwohl dieses Programm seit Urzeiten zum Standardumfang eines Linux-Systems gehört, ist es recht unbekannt – ein Grund mehr, es in diesem Buch zu besprechen.

Eingelesen werden Daten immer über die Standardausgabe; sie werden dabei automatisch als in Spalten aufgeteilter Text interpretiert. Für `colrm`

sind Spalten keine durch Leerzeichen oder Tabulatoren getrennten Bereiche. Stattdessen bildet jedes Zeichen eine neue Spalte. Ein Dateiinhalt wie etwa »abcd« besteht somit aus vier Spalten. Ein Tabulatorzeichen setzt die Spaltennummer für das darauf folgende Zeichen auf das nächsthöhere Vielfache von acht (also genauso, wie es ein Texteditor üblicherweise auf dem Bildschirm darstellt, wenn Sie ein Zeichen mit dem Tabulator verschieben).

Als Parameter übergeben Sie `colrm` in jedem Fall die Spaltennummer, ab der gelöscht werden soll. Ein optionaler zweiter Parameter dient zur Angabe der Spalte, ab der nicht weiter gelöscht werden soll. Wie so oft ist auch hier ein Beispiel angebracht.

```
$ cat testfile
abcdefg
ABCDEFG
QWERTZX
$ cat testfile | colrm 3
ab
AB
QW
$ cat testfile | colrm 3 6
abg
ABG
QWX
```

**Listing 7.8** Spalten abschneiden mit colrm

Um nur eine einzelne Spalte zu entfernen, können Sie als Start- und Stoppspalte denselben Wert angeben:

```
$ cat testfile | colrm 3 3
abdefg
ABDEFG
QWRTZX
```

**Listing 7.9** Nur die dritte Spalte einer Eingabe löschen

Sollten Sie mehrere einzelne Spalten löschen wollen, dann müssen Sie `colrm` mehrfach aufrufen. Bedenken Sie dann allerdings, dass sich die Zeichennummerierungen durch die vorhergehende Manipulation bereits verändert

haben. Wenn Sie z. B. von sechs Eingabespalten zunächst die zweite und dann die fünfte löschen wollen, müssen Sie `colrm` erst die zweite und anschließend die vierte (und nicht die fünfte) löschen lassen, da es nach dem Löschen der zweiten Spalte nur noch vier Spalten in der neuen Eingabe gibt. Fehlerfreier ist da die Lösung von »hinten«. Das heißt: Sie löschen erst die fünfte Spalte und dann die zweite, müssen also nicht rechnen. Das folgende Listing zeigt beide Fälle:



```
$ echo "123456" | colrm 2 2 | colrm 4 4
1346
$ echo "123456" | colrm 5 5 | colrm 2 2
1346
```

**Listing 7.10** Die zweite und die fünfte Spalte löschen

## 7.7 nl – Zeilennummern für Dateien

Oft soll der Quellcode eines Programms oder auch eines Shellskripts im Usenet oder in Foren gepostet oder dort erklärt werden. An dieser Stelle (aber auch bei jeglicher Form von Tabelle und Plaintext-Datenbank) sind Zeilennummerierungen ein sehr hilfreiches Mittel, um dem Empfänger bzw. der Empfängerin oder dem verarbeitenden Programm die Arbeit mit der Datei zu erleichtern.

An genau dieser Stelle setzt `nl` an und fügt der angegebenen Datei die Zeilennummern hinzu. Die Datei selbst wird dabei jedoch nicht manipuliert: Die Ausgabe erfolgt auf der Standardausgabe.

## 7.8 wc – Zählen von Zeichen, Zeilen und Wörtern

Mit diesem Programm können Sie ganz einfach die Wörter (`-w`) eines Textes (sofern dieser im ASCII-Format vorliegt), die Zeilen (`-l`) des neuesten Quellcodes oder dessen Zeichen (`-c`) zählen.

```
user$ wc -l kap??.tex
     714 kap01.tex
     439 kap02.tex
     831 kap03.tex
    1268 kap04.tex
     716 kap05.tex
    1111 kap06.tex
    2636 kap07.tex
    3632 kap08.tex
     662 kap09.tex
    1241 kap10.tex
    1964 kap11.tex
    1813 kap12.tex
     501 kap13.tex
     385 kap14.tex
     648 kap15.tex
   18561 total
```

**Listing 7.11** Zeilen der Buchdateien zählen

## 7.9 od – Dateien zur Zahlenbasis x ausgeben

Möchten Sie einmal eine Binärdatei verstehen? Nun, dazu genügt manchmal schon ein einfacher Hexeditor oder das Dump-Kommando `od`. Mithilfe dieses netten Programms können Dateien in ASCII-, dezimaler, oktaler und hexadezimaler Darstellungsweise ausgegeben werden.

Die oktale Schreibweise wird über den Parameter `-b`, die ASCII-Ausgabe mit `-c` erzielt. Dabei wird jeweils ein Byte pro Spalte dargestellt.

Bei der Ausgabe in Hexform (`-x`) und in dezimaler Form (`-d`) werden jeweils zwei Byte der Datei ausgegeben.

```
# od -x /vmlinuz
0000000 5a4d 07ea c000 8c07 8ec8 8ed8 8ec0 31d0
0000020 fbe4 befc 0040 20ac 74c0 b409 bb0e 0007
0000040 10cd f2eb c031 16cd 19cd f0ea 00ff 00f0
0000060 0000 0000 0000 0000 0000 0000 0082 0000
0000100 7355 2065 2061 6f62 746f 6c20 616f 6564
```

```
0000120 2e72 0a0d 520a 6d65 766f 2065 6964 6b73
0000140 6120 646e 7020 6572 7373 6120 796e 6b20
0000160 7965 7420 206f 6572 6f62 746f 2e2e 0d2e
0000200 000a 4550 0000 8664 0004 0000 0000 0000
0000220 0000 0001 0000 00a0 0206 020b 1402 9ca0
0000240 006d 0000 0000 9160 00f5 4610 0000 0200
...
```

**Listing 7.12** Hexdump des Kernels mit od



## 7.10 split – Dateien aufspalten

Ein Tool, das große Dateien in kleinere Stücke aufteilt, ist `split`. Die Aufteilung erfolgt entweder durch Zeilen (`-l`) oder aber durch Bytes (`-b`).

Gehen wir einmal davon aus, dass Sie eine riesige Backup-Datei auf mehrere große Speichermedien aufteilen wollen, dann wäre es doch gut, diese Datei in Teile zu zerlegen. Weil wir Retro-Computing mögen und weil die Dateigröße bei der Erklärung von `split` letztlich keine Rolle spielt, nehmen wir spaßeshalber an, Sie würden Ihr Backup auf Disketten kopieren. Eine Diskette bietet 1.440 KB Speicherplatz, wir benötigen von der Backup-Datei also 1.440 KB große Teile, um eine effiziente Speichernutzung auf den Backup-Medien zu erzielen.

Die Datei selbst hat eine Größe *x*. `split` erstellt nun so lange 1.440 KB große Dateien, bis die komplette Backup-Datei aufgeteilt ist.

Wenn die Datei nicht die Größe eines Vielfachen von 1.440 KB hat, wird die letzte Datei natürlich nur die verbleibenden Restdaten und damit das Ende der Backup-Datei enthalten und aus diesem Grund auch nicht den kompletten Speicherplatz belegen.

Bei der Aufteilung in Byte können folgende Suffixe verwendet werden: `b` für Blockeinheiten zu je 512 Byte, `k` für Kilobyte sowie `m` für Megabyte.

```
user$ split -b 1440k backup.tgz
user$ ls xa?
xaa xab xac xad xae xaf xag
```

**Listing 7.13** Aufteilen der Backup-Datei in 1.440 KB große Teile

Die Dateien xaa, xab, xac ... sind die neu erstellten Teildateien. Doch wie fügt man sie »nu' wieder 'zam«? Nun, es gibt viele Möglichkeiten, dies zu tun, die einfachste dürfte jedoch ein Aufruf von `cat` in Verbindung mit einer Ausgabeumlenkung sein.

```
$ cat x?? > backup.tgz
```

**Listing 7.14** Zusammenfügen der Dateien

## 7.11 script – Terminal-Sessions aufzeichnen

Mit `script(1)` stellte seinerzeit 3.0BSD erstmals eine Möglichkeit zur Aufzeichnung von Aus- und Eingaben einer Terminal-Session bereit, und das Tool landete letztlich auch in Linux. Diese Textströme werden dabei in eine Datei mit dem Namen *typescript* im aktuellen Arbeitsverzeichnis geschrieben. Darin finden Sie übrigens auch alle Sonderzeichen der Eingabe in Binärform vor – zum Beispiel die Betätigung der ↵-Taste bei jedem Kommandoabschluss.

```
$ cd /tmp
$ script
Script started, output file is typescript
$ ...
...
$ exit
Script done, output file is typescript
```

**Listing 7.15** Das Tool script nutzen

Die Session finden Sie in der *typescript*-Datei, die Sie sich beispielsweise mit einem Editor oder mit `cat` ansehen können:

```
$ cat /tmp/typescript
Skript gestartet auf 2021-05-06 15:07:58+02:00
[TERM="xterm-256color" TTY="/dev/pts/1"
COLUMNS="90" LINES="24"]
swendzel> ls -l /boot/*lin*
lrwxrwxrwx 1 root root      24 Apr 16 08:18 /boot/vmlinuz
...
swendzel> exit
```

```
exit

Skript beendet auf 2024-05-06 15:08:18+02:00
[COMMAND_EXIT_CODE="0"]
```

**Listing 7.16** typescript

## 7.12 bc – der Rechner für die Konsole



Bei `bc` handelt es sich um ein Rechenprogramm. Es kann Ihnen zwar nicht wie die meisten Grafiktaschenrechner eine grafische Funktionsausgabe bieten, in Verbindung mit dem Tool `gnuplot` ist aber auch dies kein Problem. `bc` verfügt dafür aber im Gegensatz zu den Produkten von Casio und Co. über eine hübsche Syntax für die Programmierung.

Doch wollen wir uns hier nicht zu sehr auf die Programmierung des `bc` stürzen, sondern primär zeigen, wie man `bc` überhaupt verwendet. Denn programmieren können Sie ja schließlich bereits mit `awk` und bald auch mit Shellskripten.

Gestartet wird der Rechner über seinen Namen, beendet wird er mit der Tastenkombination Strg + D oder dem Befehl `quit`. Sie füttern ihn einfach mit einer Rechenaufgabe, worauf er meistens das gewünschte Ergebnis ausgibt.

```
$ bc
4+4
8
1049*(5-4)-1
1048
^D
```

**Listing 7.17** Beispielrechnungen mit dem Rechner bc

Die Betonung liegt nun allerdings auf dem Wort *meistens*. Dazu sei gesagt, dass der `bc` Ergebnisse von Divisionen, wenn man es ihm nicht explizit anders befiehlt, nur als Ganzzahlen ausgibt. Um dies zu ändern, muss die Anzahl der Nachkommastellen mit dem Befehl `scale` angegeben werden.

```
$ bc
7/2
3
scale=1
7/2
3.5
scale=10
1904849/103941494
.0183261652
quit
```

**Listing 7.18** bc mit scale

Übergibt man dem `bc` beim Start den Parameter `-l`, lädt er die mathematische Bibliothek, wodurch Funktionen wie `s()` (Sinus), `c()` (Kosinus), `e()` (Exponentialfunktion) und `l(x)` (Logarithmus) zur Verfügung stehen. Diese Funktionen verwenden Sie, indem Sie sie – ähnlich wie in `awk` oder C – in der Form `Funktion(Wert)` aufrufen.

```
$ bc -l
s(3.141592)
.00000065358979323841
```

**Listing 7.19** bc -l: Sinus von einem Wert nahe Pi berechnen

Das Ergebnis der letzten Rechnung wird in eine neue Berechnung durch einen Punkt (`.`) eingebunden. Möchten Sie beispielsweise den obigen Sinus-Wert zu der Zahl 1 addieren, so addieren Sie den Punkt und 1:

```
.+1
1.00000065358979323841
.+1
2.00000065358979323841
.+3
5.00000065358979323841
.*0
0.00000000000000000000
```

**Listing 7.20** Das letzte Ergebnis erneut verwenden

Wie wir bereits erwähnt haben, werden wir an dieser Stelle nicht näher auf die Programmierung mit dem bc eingehen. Doch sei zumindest gesagt, dass Ihnen darin Schleifen und die Möglichkeit, eigene Funktionen zu implementieren, zur Verfügung stehen.

```
$ bc
for(x=0;x<6;++x){
   "Der Wert von x ist: "; x;
}
Der Wert von x ist: 0
Der Wert von x ist: 1
Der Wert von x ist: 2
Der Wert von x ist: 3
Der Wert von x ist: 4
Der Wert von x ist: 5
```

**Listing 7.21** Ein einfaches Programmierbeispiel im bc

Zudem können Sie im bc sehr einfach Variablen verwenden:

```
a=3
b=100
a*b
300
x=(a*b)*394.13+4919489
x
5037728.00
```

**Listing 7.22** Variablen im bc

Im bc können Sie auch mehrere Befehle aneinanderreihen, was eine übersichtliche Ergebnisdarstellung ermöglicht. Dazu trennen Sie in derselben Zeile einfach mehrere Anweisungen durch ein Semikolon, also etwa `x=4;x+99;.+99`.

## 7.13 Der Midnight Commander

Vielleicht gehören Sie wie wir zu einer Generation von Computernutzern, die vor vielen, vielen Jahren mit dem *Norton Commander* unter MS-DOS ge-

arbeitet haben. Für Linux gibt es seit vielen Jahren einen freien Klon dieses Programms. Dieser Klon trägt den Namen *Midnight Commander* und sieht nicht nur wie der Norton Commander aus, sondern funktioniert auch wie dieser. Ein ähnliches Programm, dessen Bedienkonzept jedoch ein wenig anders funktioniert, ist `ranger`.

Der Midnight Commander wird über `mc` gestartet und bietet je nach Terminal ein buntes oder ein schwarz-weißes Bild.

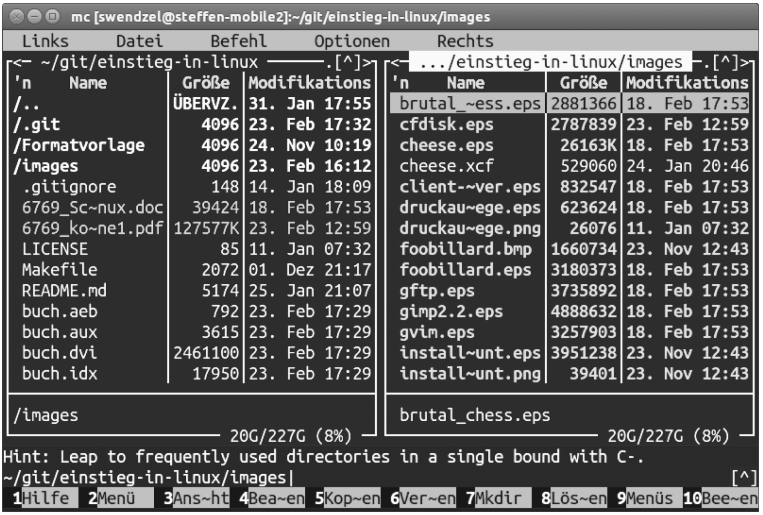


**Abbildung 7.1** Der Midnight Commander

Erscheint der Midnight Commander unter X11 nur in Schwarz-Weiß, setzen Sie die `TERM`-Variable auf den Wert »xterm-color«.

### 7.13.1 Bedienung

Generell bewegt man sich mit den Cursor-Tasten zwischen den Dateien hin und her. Für Bildsprünge im Dateibaum können Sie die Bild↓ und Bild↑-Tasten verwenden.

Kommandos können Sie direkt unten im Prompt des Bildschirms eingeben. Diese werden dann mit der ↵-Taste ausgeführt. Ist jedoch kein Kommando eingegeben, dient die ↵-Taste zum Start von grün markierten, ausführbaren Dateien.

Zwischen den zwei Bildschirmhälften wechseln Sie mit der ⇆-Taste.

Das Menü des Commanders rufen Sie mit F9 auf, den internen Datei-Viewer mit F3 und den Editor mit F4. Zudem können Sie Dateien kopieren (F5), verschieben und umbenennen (F6), Verzeichnisse erstellen (F7) und diverse andere Dateisystem-Operationen durchführen.



### 7.13.2 Verschiedene Ansichten

Was eine der beiden Seiten anzeigen soll, können Sie frei auswählen. Dazu wählen Sie im LEFT- bzw. RIGHT-Menü eine Anzeigeart aus.

- **Listing mode** legt fest, welche Attribute der Dateien im Dateilisting angezeigt werden sollen.
  - **Full file list** zeigt den Namen, die Größe und die Modifikationszeit der Datei an.
  - **Brief file list** zeigt zwei Dateispalten an, in denen ausschließlich die Dateinamen zu sehen sind.
  - **Long file list** zeigt die gleiche Ausgabe wie `ls -l`, verwendet dazu aber den kompletten Darstellungsbereich des Midnight Commanders.
- **info** gibt genauere Informationen über die auf der gegenüberliegenden Seite markierte Datei an (Abbildung 7.2).
- **Quick View** zeigt ähnlich wie `head` den Kopfbereich des Dateiinhalts an.
- **Tree** zeigt einen Verzeichnisbaum inklusive Unterverzeichnissen an. Verzeichnisse werden durch ↵ geöffnet, und ihre Dateien werden auf der gegenüberliegenden Seite gezeigt.

Hilfe zum Midnight Commander erhalten Sie, wenn Sie F1 drücken.

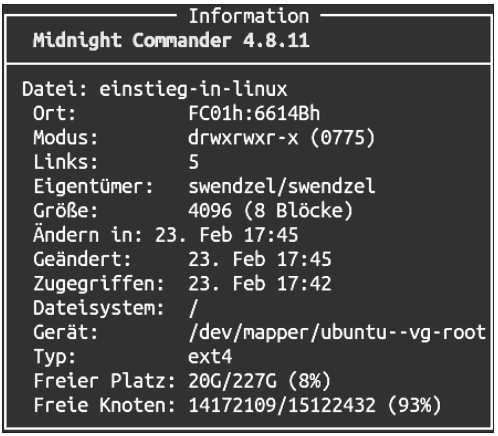


**Abbildung 7.2** Info-View

Leider steht einem zur Arbeit mit dem Dateisystem nicht immer der Midnight Commander zur Verfügung, oder aber er stößt an seine Grenzen. In diesem Buch lernen Sie deshalb natürlich, alles, was der Midnight Commander kann, auch via Shell zu bewerkstelligen.

## 7.14 Zusammenfassung

In diesem Kapitel haben wir Ihnen einige Konsolentools gezeigt, die Ihnen insbesondere die Arbeit mit Dateien erleichtern können, etwa im Rahmen von Shellskripten, die wir im nächsten Kapitel besprechen. Dateien können wir zerlegen, in Spalten aufteilen, Spalten aus Dateien entfernen, Zeitstempel anpassen, mit Zeilennummern versehen und noch viel mehr. Außerdem haben Sie gelernt, wie Sie mit `script` Shellinteraktionen aufzeichnen und mithilfe des `bc` auf der Konsole rechnen können.

## 7.15 Aufgaben

**Schon so spät!**

Erstellen Sie eine Datei mit dem Timestamp 03. April, 17:23.

**Dateisystemübersicht**

Erstellen Sie eine Liste aller eingehängten Dateisysteme, und formatieren Sie sie übersichtlich. Nutzen Sie das Programm `df -h` zusammen mit `cut`, um sich nur die prozentuale Belegung anzuzeigen.



**Script**

Zeichnen Sie eine Shellsitzung mit `script` auf, und zählen Sie anschließend die Anzahl der Ausgabezeilen mit `wc`.

# Kapitel 10
# Grundlegende Administration

*»Der Computer rechnet mit allem –*
*nur nicht mit seinem Besitzer.«*
*– Dieter Hildebrandt*



Um ein Linux-System in seinen gesamten Facetten nutzen zu können, muss man es notwendigerweise administrieren können. Und mit genau dieser Problematik befasst sich dieses Kapitel. Es macht Sie zwar nicht zu einem zertifizierten Linux-Admin, aber Sie werden lernen, die wichtigsten anfallenden Aufgaben zu erledigen.

## 10.1 Benutzerverwaltung

Ein guter Punkt, um mit der Arbeit zu beginnen, ist die Verwaltung von Benutzern. Bevor wir aber irgendwelche Lösungen und Arbeitsweisen beschreiben, wollen wir diese Problematik zuerst in Ihren bisherigen Wissenskontext einordnen.

### 10.1.1 Das Verwalten der Benutzerkonten

Bei den meisten Distributionen wird man bereits während der Installation dazu gebracht, sich einen Benutzernamen auszusuchen und entsprechende Passwörter festzulegen. Wenn Sie aber Ihren Computer nicht allein nutzen oder vielleicht sogar ein größeres Mehrbenutzersystem aufsetzen möchten, kommen Sie schnell in die Situation, dass Sie neue Benutzerkonten anlegen müssen.

### adduser und useradd

Für dieses Vorgehen gibt es zwei Programme, die eigentlich eines sind. Neue Benutzer werden über das Programm `useradd` mit vielen Kommandozeilenoptionen angelegt. Das Programm `adduser` ist dabei nur ein hübsches Frontend für dieses Programm, das von vielen Distributionen schon passend vorkonfiguriert ist.

```
# adduser mploetner
Lege Benutzer mploetner an...
Lege neue Gruppe mploetner (1002) an.
Lege neuen Benutzer mploetner (1002) mit Gruppe mploetner an.
Erstelle Homeverzeichnis /home/mploetner.
Kopiere Dateien aus /etc/skel
Enter new Unix password: <tippsel>
Retype new Unix password: <tippsel>
passwd: password updated successfully
Ändere Benutzerinformationen für mploetner
Geben Sie einen neuen Wert an oder ENTER für den Standardwert
        Name []: Maria Plötner
        Raum []:
        Telefon geschäftlich []:
        Telefon privat []:
        Sonstiges []:
Sind die Informationen korrekt? [j/n] j
#
```

**Listing 10.1** Das Anlegen eines Benutzers mit adduser

Was macht aber `adduser` genau? Es nimmt den neuen Benutzernamen als Argument von der Kommandozeile entgegen und sucht sich eine neue UID aus. Da `adduser` der schlauere der beiden Befehle zum Anlegen neuer User ist, sucht er sich eine Benutzer-ID passend zum aktuellen Systemstatus heraus. Viele Distributionen nutzen beispielsweise erst die IDs ab 500 oder 1000 für Benutzeraccounts und reservieren die unteren für Systemaccounts. Daher nimmt `adduser` einfach die erste *freie* ID ab dieser Grenze.

Danach wird der Benutzer erst einmal angelegt, also ein Eintrag für ihn in der */etc/ passwd* vorgenommen. Diese Datei sieht ungefähr so aus:

```
root:x:0:0:root:/root:/bin/bash
…
jploetner:x:1000:1000:Johannes Plötner:/home/jploetner:/bin/bash
…
mploetner:x:1002:1002:Maria Plötner:/home/mploetner:/bin/bash
```

**Listing 10.2** Die /etc/passwd

Hier kann man viele wichtige Informationen über die Benutzer sehen, daher ist die Datei für alle lesbar. Aus diesem Grund sind hier auch keine verschlüsselten Passwörter zu sehen. Das `x` in der zweiten Spalte sagt nämlich aus, dass das Passwort für diesen Benutzer bzw. diese Benutzerin in der */etc/shadow* zu finden ist. (Das war früher anders. Da stand das verschlüsselte Passwort noch in der Datei */etc/passwd*. Es wurde schließlich aber aus Sicherheitsgründen in die Datei */etc/shadow* ausgelagert.) Der Benutzername steht dabei in der ersten Spalte, die Benutzer-ID gefolgt von der Gruppen-ID in der dritten bzw. vierten Spalte. Das nächste Feld ist frei belegbar, wird aber meist für den vollen Namen und weitere Informationen genutzt. Danach folgen das Homeverzeichnis und die Standardshell des Benutzers. Es gibt alternative Shells, die hier eingetragen werden können. Man kann aber auch statt eines Kommandozeileninterpreters jedes andere Programm hier eintragen. Allerdings kann sich der Benutzer dann unter Umständen nicht mehr richtig einloggen.



Diese Einträge werden also von `adduser` sinnvoll vorgenommen. Zudem wird eine neue Gruppe allein für diesen Benutzer erstellt, und er wird, je nach Einstellung des Administrators bzw. der Administratorin, zusätzlich den schon existierenden Gruppen zugeteilt.

Als Nächstes erfolgt ein wichtiger Schritt: Das Homeverzeichnis für den neuen Benutzer wird erstellt. Dazu wird eine Art Skelett, die Vorlage eines Homeverzeichnisses, an die entsprechende Stelle kopiert und mit den richtigen Berechtigungen und Eigentumsverhältnissen versehen. Diese Vorlage liegt unterhalb von */etc/skel/* und kann vom Administrator noch nach seinen Wünschen bearbeitet werden. So können Sie beispielsweise erreichen, dass das System nach der Erstellung eines neuen Users für diesen schon fix und fertig konfiguriert ist.

Während `adduser` also seine Arbeit erledigt, fragt es fleißig alle Informationen ab, die es nicht selbst bestimmen kann. Dazu gehört natürlich das Passwort für den neuen Benutzer bzw. die neue Benutzerin. Das Passwort kann man während der Eingabe nicht sehen, daher muss man es zur Kontrolle zweimal eingeben. Danach wird das neue Passwort in die nur für `root` zugängliche */etc/shadow* geschrieben. Schließlich werden noch die Angaben für das Optionenfeld in der */etc/passwd* abgefragt. In unserem Beispiel wurde aber nur der volle Name des Benutzers angegeben.

#### deluser und userdel

Dieses Spiel funktioniert beim Löschen eines Benutzeraccounts ähnlich. Auch hier ist die `deluser`-Variante ein Frontend für den Befehl `userdel`.

```
# deluser [--remove-home] [--remove-all-files] [--backup] user
```

**Listing 10.3** Die deluser-Syntax

Bei `deluser benutzer` gibt es nur eine Sache zu beachten: Standardmäßig wird zwar der Benutzer gelöscht, das Homeverzeichnis bleibt allerdings erhalten. Möchten Sie es löschen, benutzen Sie die Option `--remove-home` bzw. `--remove-all-files`, die entweder nur das Homeverzeichnis oder gleich alle Dateien des Benutzers löscht. Wählen Sie dazu noch die `--backup`-Option, werden die entsprechenden Dateien vorher noch als gepacktes Archiv ins aktuelle Verzeichnis gelegt.

#### Und das Ganze mit Gruppen

Um den Reigen zu vollenden, gibt es auch für Gruppen die entsprechenden Befehle, die konsequenterweise `addgroup` bzw. `delgroup` heißen. Die Benutzung dieser Befehle erfolgt eigentlich analog zu ihren Geschwistern; bei Fragen und Problemen hilft hier die Manpage sehr gut weiter.

Es gibt natürlich auch hier einige Sachen, die Sie beachten müssen: zum Beispiel, dass eine Gruppe nach dem Erstellen *leer* ist, also kein Benutzer in dieser Gruppe ist. Darüber hinaus kann eine Gruppe nicht gelöscht werden, solange sie noch einem Benutzer als primäre Gruppe zugewiesen ist, sie also für ihn in der */etc/passwd* vermerkt ist.

### 10.1.2 Benutzer und Gruppen

Jetzt möchten wir die frisch erstellten Gruppen und Benutzer natürlich noch zusammenbringen. Wenn Sie sich im letzten Abschnitt gefragt haben, warum in der */etc/passwd* nur eine einzige Gruppe festgelegt wird, dann dürfen Sie sich selbst auf die Schulter klopfen, weil Sie so gut mitgedacht haben.

#### Die /etc/group

Benutzer können selbstverständlich Mitglied in mehreren Gruppen sein – und um den Umfang der */etc/passwd* nicht zu sprengen, wurden diese Einstellungen einfach in die */etc/group* ausgelagert.



```
root:x:0:
…
users:x:100:jploetner,mploetner
…
usb:x:106:jploetner,mploetner
jploetner:x:1000:jploetner
…
mploetner:x:1002:mploetner
```

**Listing 10.4** Auszug aus der /etc/group

Wie Sie sehen, ist die Datei ähnlich wie die */etc/passwd* aufgebaut. An erster Stelle steht wieder der Gruppenname, gefolgt von einem `x`. Dieses `x` steht wieder dafür, dass das Gruppenpasswort verschlüsselt in die */etc/gshadow* ausgelagert wurde, die wiederum nur `root` lesen kann. Als nächstes Feld folgt die GID, und schließlich kommen die durch Kommata separierten Mitglieder der Gruppe.

Im obigen Beispiel gibt es also die Gruppe `users` mit der GID `100` und den Mitgliedern `jploetner` und `mploetner`. Die Gruppe `jploetner` hat dagegen nur ein Mitglied, nämlich den gleichnamigen User.

Wenn man sich */etc/passwd* und */etc/group* anschaut, fällt außerdem auf, dass gleichnamige Gruppen und Benutzer dieselbe GID bzw. UID haben. Das ist allerdings reiner Zufall, da beim Erstellen der Benutzer ja auch ihre spezi-

ellen Gruppen erstellt worden sind und in der Zwischenzeit keine weiteren Gruppen per Hand hinzugefügt wurden.

#### Benutzer zu weiteren Gruppen hinzufügen

Wenn Sie bereits existierende Benutzer zu bestehenden Gruppen hinzufügen möchten, nutzen Sie einfach das `adduser`-Kommando:

```
# adduser jploetner projektX
Adding user jploetner to group projektX...
Done.
```

**Listing 10.5** Benutzer zu Gruppen hinzufügen

#### Ein Beispiel

Werden wir also mal praktisch. Nehmen wir an, Sie hätten Ihre gesamte Musiksammlung digitalisiert, und Sie möchten sie allen Benutzerinnen und Benutzern des Systems zur Verfügung stellen.

Also sollten Sie eine Gruppe `users` haben, die aller Wahrscheinlichkeit nach schon auf Ihrem System existiert. Eventuell müssen Sie in diese Gruppe nur noch alle Benutzer eintragen, falls `adduser` das nicht schon für Sie übernommen hat. Als Nächstes fehlt Ihnen noch ein schönes Verzeichnis mit den entsprechenden Rechten. Weil uns nichts Besseres einfällt, erstellen wir einfach eines unterhalb von */home* und nennen es ganz unkreativ auch noch *musik*:

```
# cd /home
# mkdir musik
# chown root:users musik
# chmod 2770 musik
```

**Listing 10.6** Unser Musikverzeichnis

An dieser Stelle haben wir dann gleich auch die entsprechenden Rechte gesetzt. Das Verzeichnis gehört `root` bzw. der Gruppe `users`, also haben alle Benutzer über die Gruppenrechte schon mal Zugriff. Jetzt setzen wir mit dem `chmod`-Aufruf aber nicht nur die normalen Zugriffsrechte für die Gruppen, sondern wir setzen auch noch das SetGID-Bit. In diesem Kontext veranlassen wir damit, dass alle Dateien, die in diesem Verzeichnis erstellt bzw. in es

hineinkopiert werden, automatisch der Gruppe `users` gehören – folglich hat jeder auch auf neue Dateien Zugriff, und Ihre Mitbenutzerinnen und -benutzer können die Sammlung ohne Probleme erweitern.

## 10.2 Installation neuer Software

Wenn man seine Benutzer so halbwegs im Griff hat, möchte man sich als Nächstes natürlich um sein System kümmern. Dazu gehören die ständige Aktualisierung der Software und das Installieren neuer Programme. Da viele Distributionen zur Vereinfachung solcher Probleme ausgefeilte Paketsysteme mitbringen, wollen wir an dieser Stelle die wichtigsten näher erläutern. Ein Paket repräsentiert dabei in der Regel ein Softwaresystem, beispielsweise den Firefox-Browser.



Kommen wir aber zuerst einmal generell zu Paketsystemen und -managern. Sie verfügen im Allgemeinen mehr oder weniger über die folgenden Eigenschaften:

- **Abhängigkeiten**
  Wenn ein gewünschtes Paket noch Abhängigkeiten (beispielsweise Softwarebibliotheken oder andere Grundsoftware) hat – ein Office-Programm braucht bspw. eine grafische Oberfläche und ein MP3-Player eine Audiobibliothek –, wird diese Abhängigkeit angezeigt bzw. meistens auch gleich aufgelöst, indem die fehlende Software einfach mitinstalliert wird. Sie werden es sich bereits gedacht haben: Die Abhängigkeiten sind wiederum selbst Pakete.
- **Saubere Deinstallation**
  Für jedes Paket sollte sich das Paketsystem bei der Installation merken, *wo* es *welche* Dateien *wie* installiert. Dann kann bei der Deinstallation der Systemzustand, wie er vor der Installation war, wiederhergestellt werden.
- **Konfiguration**
  Manche Paketsysteme erlauben auch das Ausführen bestimmter Skripte während der Installation bzw. während der Deinstallation eines Pakets, sodass auf diese Weise die Software sogar noch konfiguriert werden kann.

- **Sauberes Update**
  Genau wie die Installation und Deinstallation soll auch ein Update reibungslos funktionieren. Wenn das Paketsystem gut ist, kann es sogar mit wechselnden Abhängigkeiten bei neuen Versionen der Pakete gut umgehen. Leider ist gerade das nicht immer unproblematisch.

### 10.2.1 Welches Paketsystem nutzen?

Im Folgenden möchten wir die wichtigsten Paketsysteme näher erläutern, ohne jedoch auf die genaue Benutzung der Software einzugehen. Sie sollen allerdings eine Vorstellung davon bekommen, was man mit den einzelnen Distributionen bzw. Paketsystemen so anstellen kann – und wo vielleicht deren Vor- oder Nachteile liegen. Die populärsten Paketsysteme sind das DEB- und das RPM-Paketsystem, weshalb wir diese in den Fokus nehmen werden. Anschließend werden wir die Installation von Software ohne vorhandene Pakete bzw. Nutzung eines Paketsystems beschreiben, damit Sie in jedem Fall zum Ziel kommen – unabhängig von der verwendeten Distribution.

### 10.2.2 Das DEB-Paketsystem

Das DEB-Paketsystem stammt ursprünglich von der Debian-Distribution und wurde auch von anderen Distributionen wie Ubuntu übernommen. Es ist bekannt dafür, auch nach vielen Updates einer Installation noch sauber zu funktionieren. Zum Verwalten der Pakete und zur Administration wird dabei vor allem das APT-System eingesetzt, von dem es auch schon Klone für andere Paketsysteme gibt.

Das Prinzip von APT ist dabei einfach: Man trägt an einer zentralen Stelle im Dateisystem, der */etc/apt/sources.list*, alle Orte ein, an denen aktuelle Pakete gefunden werden können. Das können beispielsweise Server im Internet mit jeweils den aktuellsten Paketversionen sein (sogenannte *Repositories*).

#### Die Pakete an sich

Die Pakete des DEB-Paketsystems erkennen Sie ganz einfach an ihrer Endung *.deb*. Die Pakete selbst sind ja eigentlich Dateien und haben meist etwas kryp-

tisch wirkende Namen, die aber sehr viele nützliche Informationen enthalten. Hier ein Beispiel:

```
$ ls *.deb
nethack-common_3.6.0-4_amd64.deb
```

**Listing 10.7** Ein typischer Paketname

Der Paketname ist folgendermaßen aufgebaut: Zuerst kommt der Name der Software, der bis zum ersten Unterstrich reicht, in diesem Fall also `nethack-common`. Dann folgen die Version der Software selbst (`3.6.0`) und die distributionsinterne Releasenummer des Pakets (`4`). Die Releasenummer wird immer dann benötigt, wenn neue Pakete für die gleiche Softwareversion erstellt werden. Nach dem letzten Unterstrich folgt schließlich die Architektur, in unserem Fall ein 64-Bit-Desktop-Prozessor. Software wird immer für spezielle Prozessoren kompiliert. Wenn dort also `arm` stünde, wäre das Paket für einen ARM-Prozessor ausgelegt.



Das Paket an sich ist dabei nur eine gepackte Archivdatei mit einer sehr speziellen internen Struktur. Es gibt in diesem Archiv bestimmte, streng festgelegte Dateien und Verzeichnisse, die beispielsweise beschreiben, welche Abhängigkeiten so ein Paket hat. Ein Paketmanager – unter Debian das Programm `dpkg` – packt dann nur das Archiv aus und verarbeitet diese Daten entsprechend.

### Software-Installation mit apt

Da das `dpkg`-Programm zumindest für Anfänger recht kompliziert zu bedienen ist, wurde ein einfacheres System zur Paketverwaltung entwickelt – das APT-System. APT steht für *Advanced Package Tool* und ist verantwortlich für das Management der zugrunde liegenden Tools wie `dpkg` bzw. der zu handhabenden DEB-Pakete. Dabei ist APT allerdings »nur« eine Art Frontend für `dpkg`, also kein eigenes Paketsystem. Es gibt auch viele grafische Programme zur Paketverwaltung, die mitunter noch einmal auf APT aufsetzen.

Es gibt verschiedene Tools, die mit APT interagieren und die zur Paketverwaltung geeignet sind: `apt`, `apt-get` und `aptitude` sind sicher die wichtigsten Beispiele – die teilweise sogar identisch zu bedienen sind. Wir wollen im Folgenden die Paketverwaltung mit dem »neuesten« Tool, `apt`, vorstellen (eine

Weiterentwicklung von `apt-get`) und anschließend auch kurz auf `aptitude` eingehen.

Software können Sie systemweit immer nur als `root` installieren. Falls Sie mit einem normalen Nutzer arbeiten, können Sie aber auf `sudo` zurückgreifen, das Sie den unten gezeigten Befehlen voranstellen.

Jedes Mal, wenn Sie beispielsweise ein neues Programm installieren möchten – nehmen wir an, es handelt sich um die X11-Version des beliebten Adventures `nethack` –, reicht ein Kommandozeilenaufruf des Programms `apt` mit der `install`-Option. Nun werden alle benötigten Pakete aus dem Netz geladen, entpackt, konfiguriert und komplett installiert:

```
# apt install nethack-x11
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  nethack-common
The following NEW packages will be installed:
  nethack-common nethack-x11
0 upgraded, 2 newly installed, 0 to remove and 244 not upgraded.
Need to get 1,604 kB of archives.
After this operation, 4,640 kB of additional disk space will be used.
Do you want to continue? [Y/n]
Get:1 http://... amd64 nethack-common amd64 3.6.0-4 [535 kB]
Get:2 http://... amd64 nethack-x11 amd64 3.6.0-4 [1,069 kB]
Fetched 1,604 kB in 1s (3,004 kB/s)
Preconfiguring packages ...
...
Unpacking nethack-common (3.6.0-4) ...
Unpacking nethack-x11 (3.6.0-4) ...
...
Setting up nethack-common (3.6.0-4) ..
Setting up nethack-x11 (3.6.0-4) ...
...
```

**Listing 10.8** Software-Installation mit apt

Wie Sie merken, gibt man hier nicht den vollständigen Paketnamen an, sondern lediglich den Namen der Software. Alle Pakete werden ja von einem Server verwaltet und immer in der aktuellen Version vorgehalten. Zudem wird Ihre Prozessorarchitektur (beispielsweise `amd64`) automatisch bestimmt.

Sie müssen weder irgendwelche Dateien im Internet suchen noch irgendwie anderweitig aktiv werden. Die DEB-Pakete werden automatisch vom Server geladen, entpackt, und alle Dateien (ausführbare Programme, Konfigurationsdateien, Manpages etc.) werden an die richtigen Stellen im Dateisystem installiert. Sie müssen einzig und allein wissen, wie das Paket heißt, das Sie installieren möchten.



Haben Sie ein Paket trotzdem per Hand heruntergeladen, beispielsweise weil es noch nicht Teil der offiziellen Debian-Distribution ist, können Sie die gleiche Syntax verwenden. Geben Sie nur anstatt des Paketnamens den Pfad zur DEB-Datei an, um das Paket zu installieren:

```
# apt install ./super-tool_1.0.0-1_amd64.deb
```

**Listing 10.9** DEB-Pakete direkt installieren

### Pakete finden

Es kommt aber oft vor, dass man nur eine ungefähre Ahnung von dem hat, was man installieren will, und deshalb keinen konkreten Paketnamen kennt. Für diesen Fall gibt es mit der `search`-Option des `apt`-Tools eine Möglichkeit, alle Pakete der Distribution sowie ihre Beschreibungen zu durchsuchen:

```
# apt search nethack
…
nethack-x11/bionic 3.6.0-4 amd64
  dungeon crawl game - X11 interface
…
```

**Listing 10.10** Pakete suchen mit apt

`apt search` arbeitet dabei auf einem lokalen Abbild (*Cache*) aller vorhandenen Pakete – es wird also nicht im Internet gesucht! Aus diesem Grund sollten Sie immer, wenn Sie mit Servern aus dem Internet arbeiten und lange kein Update mehr vorgenommen haben, einmal kurz `apt update` aufrufen und durchlaufen lassen, um diesen Cache zu aktualisieren. Sollten nämlich

einmal Cache und Server nicht mehr übereinstimmen, beispielsweise weil auf dem Server schon neuere Versionen der Pakete vorhanden sind, kann es beim Installieren zu Problemen kommen. `apt` meldet dann, dass einzelne Pakete nicht mehr gefunden werden können, wenn diese auf dem Server bereits in einer neueren Version vorliegen.

Wenn Sie allerdings nichts Spezielles suchen, sondern einfach nur mal stöbern möchten, bieten sich Programme wie `aptitude` an. Mit ihnen können Sie komfortabel die Liste aller verfügbaren Pakete durchsehen und gegebenenfalls gleich neue Pakete installieren.

### Deinstallation

Möchten Sie ein Paket wieder loswerden, müssen Sie sich entscheiden, ob Sie eventuell vorhandene Konfigurationsdateien behalten möchten (`remove`) oder nicht (`purge`).

Unsere Empfehlung ist an dieser Stelle, für eine komplette und saubere Deinstallation von Paketen die Option `purge` zu benutzen:

```
# apt purge nethack-x11 nethack-common
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages will be REMOVED:
  nethack-common nethack-x11
0 upgraded, 0 newly installed, 2 to remove and 244 not upgraded.
After this operation, 4,640 kB disk space will be freed.
Do you want to continue? [Y/n]
(Reading database ... 169075 files and directories currently installed.)
Removing nethack-x11 (3.6.0-4) ...
Removing nethack-common (3.6.0-4) ...
...
Purging configuration files for nethack-common (3.6.0-4) ...
Purging high-scores and save-files for Nethack... done.
Purging configuration files for nethack-x11 (3.6.0-4) ...
...
#
```

**Listing 10.11** Pakete deinstallieren mit apt

Möchten Sie Konfigurationsdateien etc. bewusst behalten, rufen Sie `apt` stattdessen mit der Option `remove` auf. Die Konfigurationsdateien verbleiben dann auf dem System und stehen bei einer erneuten Installation wieder zur Verfügung.

### Das System aktuell halten mit Updates

Natürlich möchte man sein System auf dem Laufenden halten, und das am liebsten mit möglichst wenig Aufwand.

Selbstverständlich hilft Ihnen da auch das Tool `apt` weiter: Rufen Sie `apt update` zum Aktualisieren des lokalen Paketcaches gefolgt von `apt upgrade` bzw. `apt full-upgrade` auf, je nachdem, ob Sie nur neuere Versionen installieren möchten (`upgrade`) oder auch auf deren eventuell geänderte Abhängigkeiten (`full-upgrade`) eingehen wollen.



### aptitude

Wie wir bereits erwähnt haben, gibt es auch eine Alternative mit einer hübschen Oberfläche für die Kommandozeile: `aptitude` (siehe Abbildung 10.1).


**Abbildung 10.1** aptitude

Dieses Tool bietet prinzipiell die gleichen Funktionen wie die `apt`-Tools. Bedient wird es dabei außer mit den obligatorischen Cursortasten zur Navigation sowie ↵ für eine Detailansicht zum gerade ausgewählten Paket über folgende, recht intuitive Tastenkürzel:

| Befehl | Aktion |
|---|---|
| `u` | Führt ein Update durch (siehe auch `apt-get update`). |
| `U` | Markiert alle upgradefähigen Pakete für ein Upgrade. |
| `+` | Markiert ein nicht installiertes Paket zur Installation. |
| `-` | Markiert ein installiertes Paket für die Deinstallation. |
| `_` | Markiert ein Paket zur vollständigen Deinstallation (*purge*). |
| `/` | Öffnet einen Dialog zum Suchen von Paketen. |
| `n` | Springt zum nächsten Suchergebnis. |
| `g` | Führt alle geplanten Aktionen aus (*go*). |
| `q` | Je nach Kontext wird entweder das Tool beendet oder von der Detailansicht der Pakete wieder zur Paketliste zurückgesprungen. |

**Tabelle 10.1** Mögliche Aktionen

Wie versprochen, haben wir Sie nicht mit Details zur Administration eines Debian-Systems gelangweilt, aber hoffentlich doch einige interessante Ausblicke gegeben. Wenn Sie sich näher für diese Programme oder für Debian interessieren, gibt es im Internet sehr viel frei verfügbare Literatur. Dort wird auch alles noch einmal sehr ausführlich und mit Beispielen erklärt. Fragen Sie einfach die Suchmaschine Ihrer Wahl, oder starten Sie am besten gleich auf einer der vielen Seiten zur Open-Source-Software.

### 10.2.3 Das RPM-Paketsystem

Das RPM-Paketsystem ist neben Debian eines der ältesten. Es stammt ursprünglich von der Red-Hat-Distribution und wurde im Laufe der Zeit unter anderem von openSUSE sowie SUSE Enterprise Linux übernommen – und letztendlich hat es sogar die ursprüngliche Red-Hat-Distribution überlebt und ist nun für Red Hat Enterprise Linux, Fedora und Co. im Einsatz.

#### Die Pakete an sich

Die Pakete des RPM-Paketsystems haben die Endung *.rpm* und ähnlich komplizierte Namen wie die des DEB-Paketsystems. Im Prinzip sind aber auch RPM-Dateien wie ihre Debian-Äquivalente nur gepackte Archive mit einer speziellen internen Struktur, und das Programm, das sie verarbeitet, heißt `rpm` – *Red Hat Package Manager*. Das `rpm`-Programm lässt sich dabei am besten mit `dpkg` von Debian vergleichen.



Zum Management von RPM-Paketen auf der Kommandozeile hat sich das Tool `yum` etabliert. Es ist mit den APT-Tools von Debian-basierten Distributionen vergleichbar – sogar die Kommandozeilenoptionen ähneln sich.

#### RPM-Pakete managen mit yum

Konfiguriert werden die Repository-Server, die die RPM-Pakete der Distributionen bereitstellen, im Verzeichnis */etc/yum.repos.d/*. Neue Paketquellen, die zusätzliche Pakete außerhalb der Kern-Distribution bereitstellen, können dort mit einer neuen Datei konfiguriert werden. Schauen wir uns die wichtigsten Optionen von `yum` an:

- **yum install PAKET**
  lädt das angeforderte RPM-Paket inklusive aller Abhängigkeiten vom Repository-Server der Distribution, entpackt und installiert das Paket
- **yum remove PAKET**
  entfernt ein installiertes Paket inklusive aller Abhängigkeiten und Konfigurationsdateien (Dies entspricht im Wesentlichen einem `apt purge`.)
- **yum search TEXT**
  durchsucht die Paketnamen und Kurzbeschreibungen nach dem angegebenen Text und zeigt die gefundenen Pakete an

- **yum update**
  aktualisiert alle installierten Pakete inklusive eventuell geänderter Abhängigkeiten (analog einem `apt update && apt upgrade`.)

Im Vergleich zu `apt` hat `yum` eine etwas vereinfachte Syntax. Unter den verschiedenen Linux-Distributionen sind diese Tools auf jeden Fall der bevorzugte Weg, Software zu installieren und das System insgesamt aktuell zu halten.

### 10.2.4 Snaps

Snaps wurden von Canonical (das Unternehmen hinter Ubuntu) eingeführt. Ein *Snap* ist ein Softwarepaket, das seine sämtlichen Abhängigkeiten enthält. Man spricht dabei auch von einem *Bundle*. Durch die Bundles bleiben Softwarepakete samt ihrer Abhängigkeiten verschachtelt, werden also bspw. gleichzeitig aktualisiert. Aus Sicherheitssicht haben Snaps zudem den Vorteil, dass sie in einer sogenannten *Sandbox* laufen und somit vom restlichen System isoliert sind.

**Sandboxing**

Eine Sandbox kann auch herkömmlich mit Tools wie bspw. `chroot` oder über Container- und Virtualisierungslösungen (siehe auch Kapitel 12.8 zum Containersystem *Docker*) aufgesetzt werden, Snaps sind also keine Notwendigkeit, um Sandboxing zu betreiben. Zugriffsrestriktionen, bspw. auf ein Mikrofon oder ein Verzeichnis, lassen sich über fortgeschrittene Systeme wie AppArmor und SELinux setzen, die wir an dieser Stelle jedoch nicht besprechen.

Ein weiterer Sicherheitsvorteil von Snaps liegt darin, dass sie hochfrequent (je nach Konfiguration mehrfach am Tag) prüfen, ob Sicherheitsupdates vorhanden sind, und diese Updates im Hintergrund und unbemerkt vom Nutzer installieren. Updates sind atomar (es kommt zu keinen Konflikten im laufenden Betrieb) und reversibel.

Mithilfe eines Daemons (`snapd`), der unter diversen Linux-Distributionen lauffähig ist, können Softwarepakete so ohne weitere Anpassung verteilt wer-

den. Dies ist insbesondere für kommerzielle Software interessant, da diese auf möglichst vielen Systemen lauffähig sein muss. Snaps können über den *Snap Store* von Ubuntu frei heruntergeladen bzw. käuflich erworben werden.

Auf aktuellen Ubuntu-System ist bereits alles für die Nutzung von Snaps vorbereitet. Generell informiert ein Versionstest von `snap` darüber, ob Snaps direkt verwendet werden können oder ob man `snapd` zunächst nachinstallieren muss.

```
$ snap version
snap    2.62
snapd   2.62
series  16
ubuntu  22.04
kernel  5.15.0-102-generic
```

**Listing 10.12** Version von snapd abfragen



#### Welche Snaps sind bereits installiert?

Eine Übersicht über die aktuell installierten Snaps erhalten Sie mit `snap list`. Dabei ist neben dem Paketnamen auch jeweils ersichtlich, in welcher Version und Paketrevision ein Snap vorliegt, ob es sich um ein stabiles (oder bspw. in Betaphase befindliches) Paket handelt, von wem ein Snappaket erstellt und veröffentlicht wurde (*Publisher*) und ob es entsprechende Notizen gibt (`base` bedeutet, ein Snap gehört zum Basissystem, `snapd` bedeutet, dass das Snap zum Snap-Daemon gehört).

```
$ snap list
Name                Version           Rev    Tracking          Publisher     Notes
bare                1.0               5      latest/stable     canonical     base
chromium            123.0.6312.105    2811   latest/stable     canonical     -
core18              20231027          2812   latest/stable     canonical     base
core20              20240227          2264   latest/stable     canonical     base
core22              20240111          1122   latest/stable     canonical     base
cups                2.4.7-8           1041   latest/stable     openprinting  -
firefox             124.0.2-1         4090   latest/stable/... mozilla       -
gnome-3-38-2004     0+git.efb213a     143    latest/stable/... canonical     -
gnome-42-2204       0+git.510a601     172    latest/stable     canonical     -
gtk-common-themes   0.1-81-g442e511   1535   latest/stable/... canonical     -
signal-desktop      7.4.0             642    latest/stable     snapcrafters  -
snapd               2.62              21465  latest/stable     canonical     snapd
zoom-client         5.17.10.3512      225    latest/stable     ogra           -
```

**Listing 10.13** Aktuell installierte Snaps

### Snaps finden, installieren und deinstallieren

Möchten Sie Pakete installieren, müssen Sie sie zunächst ausfindig machen. Nehmen wir an, wir möchten den Browser *Firefox* installieren. Dazu suchen wir ihn mit `snap find`:

```
$ snap find "firefox"
Name           Version    Publisher Notes  Summary
firefox        124.0.2-1  mozilla   -      Mozilla Firefox web browser
firefox-kiosk  0.1        scout208  -      firefox example kiosk
...
```

**Listing 10.14** Firefox snap suchen

Vermutlich ist `firefox` unser gewünschtes Programm. Mit `snap info` können wir dies verifizieren:

```
$ snap info firefox
name:      firefox
summary:   Mozilla Firefox web browser
publisher: Mozilla
store-url: https://snapcraft.io/firefox
contact:   https://support.mozilla.org/kb/file-bug-report-or-
           feature-request-mozilla
license:   unset
description:
  Firefox is a powerful, extensible web browser with support for
  modern web application technologies.
commands:
  - firefox
  - firefox.geckodriver
snap-id:      3wdHCAVyZEmYsCMFDE9qt92UV8rC8Wdk
tracking:     latest/stable/ubuntu-22.04
refresh-date: 13 days ago, at 09:36 CEST
channels:
  latest/stable:    124.0.2-1     2024-04-02 (4090) 281MB -
  latest/candidate: 125.0-1       2024-04-11 (4136) 282MB -
  latest/beta:      125.0b8-1     2024-04-11 (4111) 282MB -
  latest/edge:      126.0a1       2024-04-16 (4153) 302MB -
  esr/stable:       115.9.1esr-1  2024-03-22 (4032) 256MB -
```

```
  esr/candidate:    115.10.0esr-1 2024-04-11 (4126) 256MB -
  esr/beta:         -
  esr/edge:         -
installed:          124.0.2-1               (4090) 281MB -
```

**Listing 10.15** Informationen über den Firefox-Snap abfragen

Die Installation erfolgt dann mit einem simplen `snap install firefox`. Eine Deinstallation von Snaps gelingt analog mit `snap remove Programmname`. Möglich ist zudem die Installation von Betaphasen-Software, also Software, die nicht ausgereift ist, aber in der Regel mehr Funktionalitäten bereitstellt. Dazu fügen Sie `--channel=beta` an.



```
$ snap install firefox
...
$ snap remove firefox
...
```

**Listing 10.16** Snaps installieren und deinstallieren

#### Snaps aktualisieren und Aktualisierungen zurücknehmen

Updates suchen und Installieren ist ebenso einfach und gelingt mit `snap refresh`. Im unteren Fall wurden der Chromium-Browser und die Gnome-Desktop-Umgebung aktualisiert.

```
$ snap refresh
chromium 123.0.6312.122 from Canonical refreshed
gnome-42-2204 0+git.510a601 from Canonical refreshed
```

**Listing 10.17** Snaps aktualisieren

Spezifische Updates eines Snaps können mit Angabe des Snaps erledigt werden, zum Beispiel: `snap refresh chromium`. Die Rücknahme eines Updates erfolgt hingegen mit `snap revert chromium`, womit Ihnen wieder die vorherige Version zur Verfügung steht.

### 10.2.5 Software-Installation ohne Pakete

Manchmal findet man im Internet ein nettes Programm, für das aber noch niemand Pakete erstellt hat. Dann bleibt einem oft nichts anderes übrig, als

das Programm »von Hand« zu installieren. Da Linux ja selbst ein Produkt der Open-Source-Gemeinde ist, sind auch sehr viele Programme *frei*. Das bedeutet, dass Sie alle Quelltexte der entsprechenden Programme bekommen können, und oft werden Ihnen auch nur diese vorgesetzt. Für Einsteigerinnen und Einsteiger ist das natürlich oft ein großes Hindernis, auch weil eine gewisse psychologische Hürde besteht, mit Sourcen zu arbeiten. Diese verschwindet jedoch relativ schnell, sobald Sie einigermaßen wissen, was Sie tun. Dieses Wissen wollen wir Ihnen im Folgenden vermitteln.

### Das Standardvorgehen

Zuallererst brauchen Sie entsprechende Software, um solche Quellcodes zu übersetzen. In den Installationsroutinen verschiedener Distributionen können Sie dazu meist Punkte wie DEVELOPMENT oder Ähnliches auswählen. Als Nächstes entpacken Sie die oft gepackt gelieferten Sourcen einfach irgendwohin:

```
$ tar -xzf quellen.tgz
```

**Listing 10.18** Entpacken eines Archivs

Im Normalfall haben Sie jetzt ein neues Verzeichnis, in das Sie einfach mit dem `cd`-Kommando wechseln können. Dann können Sie sich in aller Ruhe eventuell vorhandene *README*- und *INSTALL*-Dateien durchlesen. Im allgemeinen Fall wird sich die Kompilierung jedoch auf die folgende simple Befehlsfolge reduzieren:

```
$ cd quellen-x.y
$ ./configure
$ make
$ sudo make install # alternativ: su und make install
```

**Listing 10.19** Kompilieren von Quellcode

Dabei konfigurieren Sie zuerst das Paket mittels `./configure`, übersetzen es dann mit `make` und müssen als `root` die fertigen Programme anschließend noch an die richtige Stelle des Dateisystems kopieren. Das erledigt im Normalfall ein `make install` für Sie (entweder über `sudo` aufgerufen oder durch zwei separate Befehle: `su` und `make install`).

Standardmäßig werden selbst erstellte Programme nicht in die »normalen« Verzeichnisse wie */usr/bin* usw. kopiert, sondern in eine Extrahierarchie unter */usr/local*. Die ausführbaren Programmdateien würden sich dann beispielsweise unter */usr/local/bin*, die Manpages in */usr/local/man* befinden. Das hat den Vorteil, dass das Deinstallieren der Programme von Hand recht einfach geht. Zudem trennt man so Distributionsspezifisches sauber von selbst Hinzugefügtem.

Leider ist jede Software anders, und je nachdem, wie der oder die Autorinnen bzw. Autoren das Paket designt haben, kann der Installationsvorgang auch einmal anders aussehen. Allerdings sind das Vorgehen sowie alle Voraussetzungen meistens im Detail beschrieben, sodass das Kompilieren auch dann keine so große Hürde mehr darstellt.



#### Das Vorgehen bei Problemen

Allerdings klappt natürlich nicht immer alles so reibungslos. Im Folgenden wollen wir kurz die wichtigsten Fehler und ihre Ursachen behandeln.

Mit `configure` konfigurieren wir Code. Das heißt, es wird zum Beispiel geprüft, welche Bibliotheken (kurz: Libs für Engl. *libraries*) in welchen Versionen vorhanden sind und ob bestimmte Voraussetzungen erfüllt sind. Eigentlich macht `configure` nichts anderes, als die Abhängigkeiten, wie sie bei Paketen von den Distributoren eingestellt werden, vor dem Kompilieren zu überprüfen.

In allererster Linie sollten Sie also bei einem Fehler von `configure` nicht in Panik geraten und sofort aufgeben, sondern sich lieber in aller Ruhe die Fehlermeldung durchlesen. In der Regel erkennt selbst ein Laie, der sich mit Programmierung gar nicht auskennt, welche Bibliothek fehlt.

Oft reicht es dann aus, wenn Sie diese Bibliothek einfach nachinstallieren, also einfach mal `apt install lib_die_fehlt` probieren. Die entsprechenden Pakete haben in der Regel Namen, die mit »lib« beginnen, also beispielsweise »libsqlite3« für die SQLite3-Bibliothek (eine kleine Datenbank). Für die Übersetzung eines Programmes mit der entsprechenden Bibliothek gibt es oft noch eine Entwicklungsversion, kurz »dev«-Version, also etwa »libsqlite3-dev«. Sofern vorhanden, installieren Sie am besten beide Varianten. Unter Debian gibt es auch das Programm `auto-apt`, das zwar erst nachinstalliert

werden muss, dann aber solche fehlenden Libs automatisch nachladen soll. Dazu übergeben Sie schlicht den Aufruf von `configure` als Argument.

Der zweithäufigste Fehler, den `configure` liefert, ist das Fehlen sogenannter *Include-* oder *Headerdateien*. Diese Dateien sind eine Art Inhaltsverzeichnis für Softwarebibliotheken. Dass `configure` sie nicht findet, hat meist zwei Gründe:

- **Falsches Verzeichnis**
  Include-Dateien werden standardmäßig in bestimmten Verzeichnissen vermutet. Manchmal befinden sich aber durch Versionsnummern oder den Fakt, dass es sich bei Ihrem System um eine andere Distribution als bei den Programmierern handelt und die Verzeichnisse etwas anders strukturiert sind, die Include-Dateien woanders. Setzen Sie doch einfach einen Link, oder kopieren Sie alles entsprechend an die richtige Stelle. Vielleicht muss auch nur eine Shellvariable gesetzt werden? Manchmal hilft bei solchen Problemen, die im Übrigen eher selten auftreten, auch die *README*-Datei oder eine FAQ weiter.
- **Falsche Bibliotheksversion**
  Manche Distributionen (beispielsweise Debian) unterscheiden in ihren Paketen teilweise zwischen Bibliotheken für normale Systeme und Bibliotheken zum Programmieren. Das hat den Vorteil, dass eine normale Installation so erheblich kleiner wird, da Include-Dateien wirklich nur zum Übersetzen gebraucht werden. Allerdings kann man manchmal schon verzweifeln, wenn man die entsprechende Bibliothek zwar wirklich installiert hat, aber trotzdem nichts funktioniert. In so einem Fall versuchen Sie einfach mal ein `apt install libXYZ-dev` für die Entwicklungsversion.

  Manchmal benötigt ein Programm vielleicht eine ältere Bibliothek, die nicht mehr auf dem System installiert ist. In so einem Fall ist allerdings Fingerspitzengefühl gefragt, damit Sie nichts Wichtiges kaputtmachen. Wie so oft gilt: Wenn man weiß, was man tut, kann man getrost fortfahren, ansonsten sollte man besser aufhören – und zwar sofort.

Tritt beim Kompilieren ein Fehler auf, ist entweder eine Bibliothek nicht vorhanden oder es liegt ein Programmierfehler vor. Letzteres ist allerdings eher selten und normalerweise können Sie dann auch nichts machen (oder

müssen sehr viel Zeit in die Analyse des Quellcodes investieren). In so einem Fall hilft einfach nur das Warten auf eine neue Version oder eine Mail an das Entwicklerteam.

Wenn `make install` einen Fehler liefert, liegt das meist an fehlenden Rechten und seltener an unfähigen Programmierern. Falls Sie wirklich `root` sind, können Sie ja versuchen, aus den Fehlermeldungen schlau zu werden – eine allgemeine Lösung kann man hier leider nicht bieten. Allerdings gilt wie so oft: Wer keine Angst vor Fehlern hat und sich wirklich Mühe gibt, sie zu verstehen, der kann das Problem mit hoher Wahrscheinlichkeit ganz einfach lösen.[1]



## 10.3 Backups erstellen

Egal, ob Sie zu Hause nur einen kleinen PC haben oder der Administrator bzw. die Administratorin eines mittleren Firmennetzwerks sind, Sie werden an der Backup-Problematik nicht vorbeikommen. Aber wie immer erst mal der Reihe nach.

### 10.3.1 Die Sinnfrage

Nein, an dieser Stelle wollen wir nicht klären, »woher wir kommen, wohin wir gehen und warum wir hier sind«. Viel eher lauten die Fragen hier: »Wohin damit?« und »Was soll das?«. In der Tat sehen viele Menschen die Notwendigkeit von Backups nicht ein und bereuen das später oft ganz bitterlich.

#### Was ist ein Backup?

Ein Backup ist eine Art *Sicherheitskopie* wichtiger Daten. So eine Kopie wird entweder vom ganzen System oder nur von wirklichen Nutzdaten wie Datenbanken oder Ähnlichem erstellt – in allererster Linie, um einem Datenverlust vorzubeugen. Da es ziemlich selten vorkommen sollte, dass man wichtige Daten mal so eben aus Versehen löscht, ist der Hauptgrund für Datenverlust

1 Wir wiederholen uns, aber die psychologische Hemmschwelle ist nicht zu unterschätzen.

oft ein defekter Datenträger. Hin und wieder werden Daten auch aus Versehen überschrieben.

Es ergibt also wenig Sinn, Daten auf demselben Datenträger zu sichern, wie es leider von sehr vielen Laien oft praktiziert wird. Mit anderen Worten: Es nützt wirklich nichts, wenn Sie Ihre wichtigen Daten einfach nur in ein zweites Verzeichnis kopieren. Geht Ihre Festplatte kaputt, sind die vermeintlich gesicherten Daten nämlich auch futsch.

**Zur richtigen Zeit am richtigen Ort ...**

... sollten Sie ein Backup machen. Wann das ist und worauf es gespeichert wird, ist dabei allerdings sehr von den Gegebenheiten abhängig, denen Sie sich zu unterwerfen haben.

Sie sollten des Weiteren beachten, dass eine ordinäre Sicherheitskopie erst zum wirklichen Backup wird, wenn sie regelmäßig von einem vordefinierten Datenbestand gezogen wird. Wenn Sie also alle Jubeljahre mal ein paar Dateien sichern, weil die eventuell noch wichtig sein könnten, macht das noch lange kein Backup.

Der verrückte Bücher schreibende Informatikstudent hingegen, auf dessen Zweit- oder Drittserver jede Nacht vollautomatisch ein Backup mit seinen wichtigsten Daten durchgeführt wird, macht es schon eher richtig. Aber wie immer gilt: Jeder muss seinen Weg finden, und jeder Weg ist anders. Vielleicht macht Ihnen ein Datenverlust ja auch gar nichts aus, weil Sie Ihren PC nur zum Surfen nutzen und beispielsweise E-Mails über ein Webinterface lesen. In solchen Fällen müssen Sie natürlich genau überlegen, ob Sie überhaupt ein Backup brauchen oder ob Sie die Zeit, die Sie dafür zu investieren hätten, nicht sinnvoller nutzen könnten.

### 10.3.2 Backup eines ganzen Datenträgers

Wie wir schon erwähnt haben, ist eine Art des Backups ein vollständiges System-Backup. Mit dieser Methode erstellt man einfach nur ein Image, also eine bitweise Kopie der Festplatte bzw. anderer Datenträger, die im Notfall einfach wieder »drübergebügelt« wird – alle in der Zwischenzeit neu hinzugekommenen Daten sind natürlich wie bei jedem Backup verloren. (Dieser

Fakt impliziert natürlich, dass Sie regelmäßig Backups machen sollten, um im Notfall nicht wieder ins letzte Jahrtausend zurückgeworfen zu werden.) Allerdings haben Sie im Fall eines Totalausfalls wirklich keine Arbeit außer der, den Kopiervorgang zu initialisieren. Sie müssen kein neues System installieren und nichts konfigurieren – der Zustand zum Zeitpunkt des Backups wird komplett wiederhergestellt.

Demnach hat ein solches Backup natürlich den Vorteil, dass man am wenigsten Arbeit damit hat. Allerdings ist so ein Backup häufig wirklich *sehr* groß, sodass es für Privatanwenderinnen und -anwender oft nicht praktikabel ist, ihre gesamte Festplatte auf diese Art zu sichern. Man bräuchte, würde man so ein Image nicht packen, genauso viel Platz, wie die alte Festplatte groß ist – sprich die gleiche Platte noch einmal. Und das ist schlicht zu teuer, außerdem gibt es bessere Alternativen.



Aber sehen wir uns einmal ein einfaches Beispiel an. Wir benutzen dazu das Programm `dd`, das solche bitweisen Kopien herstellen kann. Sie geben dazu einfach ein Device bzw. eine Datei als Eingabe und als Ausgabe an. `dd` liest dann blockweise von der Eingabe und schreibt die Daten auf die Ausgabe.

In unserem Beispiel wollen wir eine alte Festplatte verkaufen und sie vorher gründlich löschen. Weil heutzutage selbst jeder Laie weiß, dass das Löschen von Dateien diese nicht wirklich von der Festplatte entfernt, möchten wir die ganze Platte wirklich leeren. Wir lesen dazu von */dev/zero*, einem Device, das beim Lesen immer nur Nullen zurückgibt. Wir beschreiben unsere Festplatte also mit Nullen, sodass eventuell böswillige Käufer nicht in unsere Privatsphäre eindringen können, indem sie versuchen, Daten wiederherzustellen.

```
# dd if=/dev/zero of=/dev/sdb
```

**Listing 10.20** Festplatte löschen mit dd

Sie fragen sich jetzt bestimmt, was dieses komische Beispiel mit einem Backup zu tun hat. Nun, das ist kreativer Umgang mit Technik – um unser Ziel zu erreichen, leiten wir einfach das Zero-Device auf die zweite Festplatte */dev/sdb*. Die wichtigsten Optionen für `dd` sind dabei `if=DEVICE` und `of=DEVICE`, die die Quelle bzw. das Ziel unserer Kopiererei festlegen.

In unserem zweiten Beispiel wollen wir den *Master Boot Record* (MBR) sichern:

```
# dd if=/dev/sda of=MBR.txt count=1 ibs=512
1+0 records in
1+0 records out
# ls -l MBR.txt
-rw-r--r--  1 root root   512 Oct  8 18:03 MBR.txt
```

**Listing 10.21** Backup des MBR

Hier haben wir die Blockgröße bei der Input-Datei mit der Option `ibs` auf 512 Byte, also einen Sektor, gesetzt. Anschließend kopieren wir genau einen Block (`count=1`) – und der erste Sektor der Festplatte ist schließlich der MBR mit Bootloader und Partitionstabelle.

Die resultierende Datei ist natürlich genau 512 Byte groß – so viel haben wir ja auch kopiert. Wir haben jetzt zwar immer noch keinen gesamten Datenträger kopiert, aber sicherlich ist das auch nicht nötig.

Sie kennen jetzt die wichtigsten Optionen von `dd` und haben eine ungefähre Vorstellung davon, was man damit anstellen kann. Mit etwas Logik und Forscherdrang könnten Sie versuchen, auf ganz ähnliche Weise ein Image Ihrer alten Computerspiele (noch auf verstaubter Floppy, CD oder DVD auf dem Dachboden gelagert) zu erstellen oder eine Partition zu sichern. Oder Sie erstellen eine ganze Sammlung unterschiedlicher Distributionen für den Raspberry Pi, die Sie auf unterschiedliche SD-Karten speichern.

### 10.3.3 Backup ausgewählter Daten

Möchte man nur ausgewählte Daten sichern, so befinden sich diese oft in einem bestimmten Verzeichnis oder einer besonderen Datei. In einem solchen Fall bietet es sich an, die entsprechenden Daten zu komprimieren, um Platz zu sparen.

Wo Sie beispielsweise unter Windows die Daten einfach »zippen« würden, ist unter Linux ein etwas anderes Vorgehen angebracht. Es wird nämlich zwischen einem Archiv und einer gepackten Datei unterschieden: Ein Archiv

enthält mehrere Dateien und Verzeichnisse, während eine gepackte Datei einfach nur eine einzige komprimierte Datei darstellt.

### Archive mit tar

Damit Rechte und andere Dateiattribute erhalten bleiben, werden mehrere Dateien also vor dem Packen jeweils in ein Archiv gesteckt. Das hat den Vorteil, dass bei Änderungen an den Dateiattributen nicht jedes einzelne Komprimierungsprogramm neu geschrieben werden muss.

Das Archivierungsprogramm der Wahl ist unter Linux so gut wie immer `tar`, der **T***ape* **Ar***chiver*. Wie Sie dem Namen entnehmen können, stammt das Programm aus einer Zeit, als Backups noch auf große Magnetbänder geschrieben wurden.



```
$ tar -c Verzeichnis > Verzeichnis.tar
$ ls *.tar
Verzeichnis.tar
```

**Listing 10.22** Ein Archiv mit tar erstellen

`tar` schreibt die binären Daten standardmäßig einfach auf die Standardausgabe, also in unserem Fall auf den Bildschirm. Das ist nützlich für Pipes, siehe Kapitel 4, »Grundlagen der Shell«. Weil wir sie aber nicht auf dem Bildschirm, sondern lieber in einer Datei haben wollen, müssen wir die Ausgabe mit dem >-Operator in eine Datei *umlenken*.

Möchten wir das Ganze auch noch packen, dann müssen wir zur Option `-c` für **create** auch noch ein `-z` packen, um das Resultat noch zu *gzippen*. Das erspart uns den Aufruf eines Extraprogramms, und so war es also nicht ganz richtig, als wir am Anfang sagten, dass Archivierer und Packer streng voneinander getrennt sind. Das Resultat ist allerdings: Es handelt sich um ein gepacktes `tar`-Archiv.

```
$ tar -cz Verzeichnis > Verzeichnis.tar.gz
$ ls *.gz
Verzeichnis.tar.gz
```

**Listing 10.23** Ein komprimiertes Archiv mit tar erstellen

Jetzt haben wir alle Dateien in `Verzeichnis` gepackt. Wir drücken die Tatsache, dass wir den Inhalt von `Verzeichnis` erst gepackt und dann komprimiert haben, durch die Endung `.tar.gz` aus, die oft auch als `.tgz` abgekürzt wird.

Möchten wir so ein Archiv wieder entpacken, nutzen wir statt `-c` für *create* einfach die Option `-x` für *extract*. Handelt es sich um ein `gzip tar`-Archiv, packen wir wie beim Erstellen einfach noch das `-z` dazu.

```
$ tar -xz Verzeichnis.tar.gz
```

**Listing 10.24** Ein Archiv mit tar entpacken

Was Sie mit `tar` noch so alles anstellen können, erfahren Sie wie immer auf der Manpage, die Sie mit `$ man tar` lesen. Dort finden Sie unter anderem Informationen darüber, wie bestehende Archive modifiziert werden können oder wie Sie sich deren Inhalt ausgeben lassen können.

#### Komprimieren mit gzip, bzip2 und compress

Wie wir bereits erwähnt haben, gibt es unterschiedliche Werkzeuge, die alle unterschiedliche Komprimierungsverfahren einsetzen und daher mehr oder weniger effektiv sind. Je mehr eine Datei komprimiert wird, umso länger muss in der Regel diese Komprimierung berechnet werden.

Im folgenden Listing stellen wir einfach die entsprechenden Komprimierungsprogramme gegenüber. Dazu haben wir unser Buchverzeichnis gepackt, das im Original zum aktuellen Zeitpunkt stolze 13 MByte groß ist. Wie Sie sehen, sind die Ergebnisse für verschiedene Kompressoren doch sehr unterschiedlich. Das GNU-`gzip`-Programm in Verbindung mit `tar` liefert neben dem `zip`-Programm das größte Ergebnis. Mit Abstand die kleinste Datei hat `bzip2` erzeugt, allerdings hat das Programm dafür auch am längsten gebraucht. Das ist ein Grund dafür, dass man in der Unix-Welt oft auf den Kompromiss `gzip` zurückgreift.

```
$ ls -lh Buch*
-rw-r--r-- 1 jploetne users  2.2M  Oct9  Buch.tar.bz2
-rw-r--r-- 1 jploetne users  3.7M  Oct9  Buch.tar.gz
-rw-r--r-- 1 jploetne users  3.8M  Oct9  Buch.zip
-rw-r--r-- 1 jploetne users   13M  Oct9  Buch.tar
```

**Listing 10.25** Vergleich der Komprimierungsprogramme

Sie sehen auch gut, dass tar die Daten standardmäßig nicht packt – das .tar-Archiv ist nämlich genauso groß wie das Verzeichnis selbst. Nun möchten wir die Programme einzeln kurz vorstellen und ihre Bedienung erläutern.

**compress**

Das Programm compress hat mittlerweile nur noch historische Bedeutung und wird in der Praxis kaum noch eingesetzt. Die Programme zum Packen bzw. Entpacken heißen compress und uncompress, und die entsprechenden Dateien werden dabei einfach auf der Kommandozeile übergeben. Weil compress kaum noch genutzt wird, ist auf vielen Systemen oft nur uncompress vorhanden. So haben Sie zwar die Möglichkeit, noch vorhandene alte Archive zu entpacken, jedoch nicht die Möglichkeit, neue zu erstellen.



**bzip2**

Das effektivste unter den Komprimierungsprogrammen wird durch die Kommandos bzip2 und bunzip2 gesteuert, an die sich der Dateiname der zu komprimierenden Datei anschließt. Optional können Sie beim Packen beispielsweise noch das Verhältnis von Geschwindigkeit und Effektivität durch die Parameter -1 bis -9 steuern, wobei -1 die schnellste und -9 die effektivste Art zu packen bezeichnet.

Haben Sie eine gepackte Textdatei, so können Sie sie auch lesen, ohne sie gleich entpacken zu müssen. Für diesen Fall gibt es nämlich das praktische Programm bzcat, das den Inhalt der Datei einfach ausgibt:

```
$ bzcat Readme.bz2
Das ist
   ein Test
.
$
```

**Listing 10.26** Gepackte Dateien lesen: bzcat

**gzip**

Die einfache Steuerung durch zwei Programme zum Packen und Entpacken gibt es natürlich auch – gzip und gunzip. Allerdings gibt es für das beliebte und populärste Packprogramm weitaus mehr Tools als nur ein zcat wie bei bzip2. Es sind nämlich unter anderem diff, less und grep jeweils durch ein vorangestelltes z auch für so gepackte Dateien verfügbar.

#### deja-dup macht Backups einfach

Sofern Sie einen Laptop- oder Desktop-Rechner mit grafischer Oberfläche nutzen, können Sie auch auf grafische Tools wie `deja-dup` zurückgreifen. Diese Tools müssen Sie gegebenenfalls zunächst installieren. `deja-dup` erlaubt die einfache Selektion der zu sichernden Verzeichnis, führt die Backups automatisiert regelmäßig durch, unterstützt die Sicherung auf Remote-Systemen, etwa über SFTP, und kann Backups auch direkt vom Sicherungssystem zurückspielen.

## 10.4 Logdateien und dmesg

Nachdem wir unsere Benutzer und die Software verwalten sowie wichtige Daten sichern können, wollen wir jetzt das System erst einmal so am Laufen halten. Linux erleichtert diese Arbeit dadurch, dass für alle wichtigen Ereignisse *Logdateien* erstellt werden.

Eine Logdatei ist eine Datei, in der wichtige Vorgänge im System verzeichnet werden. Viele Softwarepakete haben eigene Logdateien. Sie werden in diesem Abschnitt die wichtigsten kennenlernen. Diese Dateien sind im Allgemeinen unter */var/log* oder seinen Unterverzeichnissen gespeichert.

### 10.4.1 /var/log/messages

Die wichtigste Logdatei eines Systems enthält so ziemlich alles, was der Kernel oder einige Systemprozesse mitzuteilen haben. So schreibt der Kernel eigene Meldungen in diese Datei, aber auch Anwendungen ohne eigene Logdateien haben die Möglichkeit, Nachrichten hineinzuschreiben. Sehen wir uns einfach mal einen Auszug an:

```
…
Oct 12 11:44:44 localhost kernel: eth0: no IPv6     \
  routers present
…
Oct 12 14:59:00 localhost /USR/SBIN/CRON[2011]:     \
  CMD ( rm -f /var/spool/cron/lastrun/cron.hourly)
Okt 12 15:29:09 localhost su: FAILED SU (to root)
```

```
  jploetner on /dev/pts/4
Okt 12 15:29:16 localhost su: (to root) jploetner   \
  on /dev/pts/4
Okt 12 15:29:16 localhost su: pam_Unix2: session    \
  started for user root, service su
Okt 12 15:31:42 localhost su: pam_Unix2: session    \
  finished for user root, service su
```

**Listing 10.27** Auszug aus /var/log/messages

Hier wird auch schon der generelle Aufbau deutlich, den Logdateien oft haben: Das Datum und die genaue Uhrzeit werden vor der eigentlichen Nachricht angegeben. Im Fall der */var/log/messages* folgt nach der Zeit der Name des Rechners, der die Meldung verursachte, dann der Name des aufrufenden Programms, gefolgt von der eigentlichen Meldung.



In unserem Fall enthält die Datei ausschließlich Meldungen eines einzigen Computers, der im Logfile als `localhost` identifiziert wird. (Der Name `localhost` bezeichnet immer den aktuellen, also eigenen Rechner. In einem Netzwerk können aber unterschiedliche Hostnamen vergeben werden, siehe Kapitel 11, »Netzwerke unter Linux«.) In unserem Beispiel haben wir Meldungen von den Programmen `cron`, `su` und dem Kernel sowie Einträge des Logging-Systems selbst, die wir aber getrost ignorieren können.

An der Ausgabe erkennen wir zum Beispiel, dass der Benutzer `jploetner` einmal vergeblich versucht hat, sich per `su`-Kommando die `root`-Identität zu verschaffen, um als Systemadministrator Aufgaben zu übernehmen. Ein paar Sekunden später hat er es aber dann doch geschafft und ist für ein paar Minuten in einer Session als `root` aktiv.

Aus den Meldungen des Kernels haben wir eine beliebige herausgegriffen, in diesem Fall eine Meldung vom Bootvorgang, die beim Initialisieren der Netzwerkschnittstelle *eth0* aufgetreten ist. Es werden also schwerwiegende Fehler, Warnungen sowie schlichte Informationen in dieser Logdatei angezeigt.

Eine typische Information ist der Eintrag des `cron`-Programms. Es handelt sich dabei um einen Dienst, der regelmäßig Programme beispielsweise zu Administrationszwecken ausführt. Eine solche Logdatei ist ein komfortabler

Weg, an Rückmeldungen über die gestarteten Programme zu kommen – in diesem Fall wird einfach nur eine Statusdatei mit dem Kommando `rm` gelöscht, um die Festplatte nicht mit unnützen Daten zu verstopfen.

### 10.4.2 /var/log/wtmp

Die `wtmp`-Datei enthält Informationen über die letzten Login-Versuche bzw. die letzten Logins für jeden einzelnen Nutzer. Leider ist die Datei in keinem lesbaren Textformat gespeichert. Daher sollten Sie lieber das Programm `lastlog` nutzen, um Informationen aus dieser Datei auszulesen und anzuzeigen:

```
$ lastlog
Username  Port  From       Latest
root      tty2             Don Sep 18 16:35:01 +0200 2003
bin                        **Never logged in**
…
jploetner :0    console    Son Okt 12 11:46:30 +0200 2003
swendzel  pts/5 jupiter.wg Son Okt 12 20:05:22 +0200 2003
…
```

**Listing 10.28** Die letzten Logins mit lastlog

Beim einfachen Aufruf von `lastlog` werden alle Benutzer mit den entsprechenden Daten ausgegeben. Zu diesen Daten gehört natürlich auf der einen Seite der Zeitpunkt, auf der anderen Seite beispielsweise aber auch der Ort, von dem aus sich der entsprechende Benutzer bzw. die Benutzerin eingeloggt hat. Das kann eine lokale Konsole (beispielsweise `tty2`) oder auch ein fremder Rechner (`jupiter.wg`) sein, der eine virtuelle Konsole (`pts/5`) zum Einloggen nutzt.

### 10.4.3 /var/log/Xorg.log

Die Logdatei der grafischen Oberfläche ist ein typisches Beispiel für eine Logdatei einer Anwendung, die ja vom System getrennt ist. An dieser Stelle möchten wir nur erwähnen, dass es diese Datei gibt, da man nach unerklärlichen Abstürzen des XServers vielleicht hier einen Hinweis auf die Ursache findet.

### 10.4.4 syslogd

Der `syslogd` hat nur bedingt etwas mit Logdateien zu tun – er ist ein sogenannter *Daemonprozess* und verwaltet die Logdateien. Wenn dieser Dienst läuft, haben auch andere Rechner im Netzwerk die Möglichkeit, auf diesem Rechner ihre Logdateien anzulegen. Aus diesem Grund ist in der */var/log/messages* ein Feld für den Rechnernamen reserviert – so können Sie die einzelnen Systeme auseinanderhalten. Sinnvoll ist der Einsatz eines Logging-Servers oft nur in größeren Netzwerken, wo beispielsweise Hacker daran gehindert werden sollen, auf gehackten Systemen durch das Löschen von Logdateien ihre Spuren zu verwischen.



### 10.4.5 logrotate und DoS-Angriffe

Logdateien werden mit der Zeit immer größer, und übergroße Logdateien können ein System potenziell außer Gefecht setzen, indem sie irgendwann die ganze Festplatte füllen und das System somit keine neuen Daten mehr ablegen kann. Man nennt einen derartigen Angriff einen *DoS-Angriff* (*Denial of Service*), weil der Rechner vom Angreifer so überlastet wird, dass er schließlich den Dienst verweigert.[2]

Zur Lösung des Problems überfüllter Platten gibt es das Programm `logrotate`. Wird eine Logdatei zu groß oder ist ein bestimmter Zeitabschnitt vorüber, wird die Logdatei gepackt bzw. gelöscht und eine neue, leere Logdatei erstellt.

`logrotate` ist kein Daemonprozess, und das hat auch seine Gründe. Es ist nicht notwendig, dass `logrotate` die gesamte Zeit im Hintergrund läuft und Speicher sowie Rechenzeit frisst – `logrotate` läuft als `cron`-Job, wird also vom `cron`-Daemon in einem regelmäßigen Intervall gestartet.

In vielen Distributionen ist `logrotate` schon als `cron`-Job sinnvoll vorkonfiguriert, und wenn Sie in Ihrem Logverzeichnis */var/log* ein paar durchnummerierte und gepackte Dateien finden, ist `logrotate` bereits am Werk.

2 Tatsächlich gibt es diverse Formen von DoS-Angriffen, insbesondere in Netzwerken, siehe hierzu bspw. S. Wendzel: *IT-Sicherheit für TCP/IP- und IoT-Netzwerke*, 2. Aufl., Springer, 2021.

```
$ ls /var/log/messages*
/var/log/messages
/var/log/messages-20230510.gz
/var/log/messages-20230629.gz
```

**Listing 10.29** logrotate im Einsatz

### 10.4.6 tail und head

Wenn Sie einmal einen Blick in */var/log* werfen, werden Sie feststellen, dass dort viel mehr Dateien liegen als nur die paar, die wir Ihnen hier vorgestellt haben. Zudem sind Logdateien trotz `logrotate` mitunter sehr groß, und ein Anschauen mit `cat <Datei>` macht da nicht wirklich Spaß.

Als Lösung bieten sich wie immer ein paar mysteriöse Shellprogramme an: `head` und `tail`. Beide Programme haben eine ähnliche Syntax und zeigen jeweils Teile einer Datei, doch mit dem Unterschied, dass `head` die ersten und `tail` entsprechend die letzten Zeilen einer Datei anzeigt.

```
$ head Makefile
all : .
        latex buch
        makeindex -s gp97.ist buch
        latex buch
        makeindex -s gp97.ist buch
        latex buch
        dvips -P gp -j0 -o buch.ps buch
…

$ tail Makefile
view:
        evince buch.ps

backup :
        cp -r *.tex ../working_backup/
        sync
…
```

**Listing 10.30** head und tail

In diesem Beispiel haben wir uns einfach das Makefile für unser Buch angeschaut. Ein Makefile ist eine Art Skript, das normalerweise eher für die Programmierung genutzt wird. Wir haben jedoch ein Skript geschrieben, das unser Buch automatisch *übersetzt*, sichert oder in einem Extra-Betrachter anzeigt. Aber eigentlich ist das nebensächlich, denn es geht schlicht um den Fakt, dass `head` wie versprochen den Anfang und `tail` das Ende der Datei ausgibt. Per Default sind das jeweils 10 Zeilen, es kann aber mit der Option `-n` eine andere Zeilenzahl angegeben werden.

Da in Logdateien die neuesten Einträge sinnvollerweise immer unten stehen, ist sicherlich `tail` für Sie das Programm der Wahl. Mit der Option `-f` können Sie auch aktuelle Änderungen mitverfolgen. Das Programm bleibt aktiv, und wenn neue Einträge in die Datei geschrieben werden, werden sie auch gleichzeitig auf dem Bildschirm ausgegeben.



## 10.5 Weitere nützliche Programme

In ein Kapitel zur grundlegenden Administration gehört noch eine Reihe kleiner, aber doch wichtiger Programme, die wir nun im Einzelnen vorstellen wollen.

### 10.5.1 Speicherverwaltung

Sie werden es von anderen Systemen vielleicht weniger gewohnt sein, Kontrolle über Ihren Hauptspeicher zu besitzen, um eventuelle Spitzen in der Auslastung abfangen zu können.

Die Kontrolle über den Hauptspeicher fängt schon beim Sticky-Bit an und erstreckt sich bis zur Laufzeitkontrolle über Swap-Bereiche auf der Festplatte. Sie merken schon: An dieser Stelle fängt es so langsam an, sich auszuzahlen, wenn man das Buch bisher intensiv gelesen und nicht einfach Abschnitte überblättert hat.