Linux

Das umfassende Handbuch

Kapitel 1
Was ist Linux?

Um die einleitende Frage zu beantworten, erkläre ich in diesem Kapitel zuerst einige wichtige Begriffe, die im gesamten Buch immer wieder verwendet werden: Betriebssystem, Unix, Distribution, Kernel etc. Ein knapper Überblick über die Merkmale von Linux und die verfügbaren Programme macht deutlich, wie weit die Anwendungsmöglichkeiten von Linux reichen.

Es folgt ein kurzer Ausflug in die Geschichte von Linux. Von zentraler Bedeutung ist dabei natürlich die *General Public License* (kurz GPL), die angibt, unter welchen Bedingungen Linux weitergegeben werden darf. Erst die GPL macht Linux zu einem freien System, wobei »frei« mehr heißt als einfach »kostenlos«.

1.1 Einführung

Linux ist ein Unix-ähnliches Betriebssystem. Der wichtigste Unterschied gegenüber historischen Unix-Systemen besteht darin, dass Linux zusammen mit dem vollständigen Quellcode frei kopiert werden darf.

Betriebssystem

Ein Betriebssystem ist ein Bündel von Programmen, mit denen die Grundfunktionen eines Rechners realisiert werden. Dazu zählen die Verwaltung von Tastatur, Bildschirm, Maus sowie der Systemressourcen (CPU-Zeit, Speicher, SSDs etc.). Sie benötigen ein Betriebssystem, damit Sie ein Anwendungsprogramm überhaupt starten und eigene Daten in einer Datei speichern können. Populäre Betriebssysteme sind Windows, Linux, macOS, Android und iOS.

Unix versus Linux

Schon lange vor Windows, Linux und den Smartphones gab es Unix. Dieses Betriebssystem war technisch gesehen seiner Zeit voraus: echtes Multitasking, eine Trennung der Prozesse voneinander, Zugriffsrechte für Dateien etc. Allerdings bot Unix anfänglich nur eine spartanische Benutzeroberfläche, stellte hohe Hardware-Anforderungen und lief nur auf teuren Workstations.

Inzwischen hat Linux Unix verdrängt: Große Teile des Internets werden von Linux-Servern getragen. Linux läuft in Form von Android auf Smartphones, in Routern und NAS-Festplatten sowie in Supercomputern: Die 500 schnellsten Rechner der Welt laufen heute *alle* unter Linux (*https://top500.org/statistics/list*).

Kernel Genau genommen bezeichnet der Begriff Linux nur den Kernel: Er ist der innerste Teil (also der Kern) eines Betriebssystems mit ganz elementaren Funktionen, wie Speicherverwaltung, Prozessverwaltung und Steuerung der Hardware. Die Informationen in diesem Buch beziehen sich auf den Kernel 6.*n*.

1.2 Hardware-Unterstützung

Linux unterstützt beinahe die gesamte gängige PC-Hardware und läuft darüber hinaus auch auf anderen Hardware-Plattformen, z. B. auf Smartphones oder Embedded Devices. Dennoch müssen Sie beim Kauf eines neuen Rechners aufpassen. Es gibt einige Hardware-Komponenten, die im Zusammenspiel mit Linux oft Probleme machen:

- **Grafikkarten:** Fast alle auf dem Markt vertretenen Grafikkarten bzw. in die CPU integrierten Grafik-Cores funktionieren unter Linux. Für viele Linux-Anwender ohne besondere Anforderungen an das Grafiksystem sind Intel- oder AMD-CPUs mit eingebautem Grafik-Core die optimale Lösung. Grafikkarten von NVIDIA erfordern hingegen oft Zusatztreiber, damit die Karte perfekt genutzt werden kann. Die Installation dieser Treiber kann Probleme bereiten.
- **WLAN- und Netzwerkadapter:** WLAN- und LAN-Controller machen selten Probleme. Nur ganz neue Modelle werden von Linux mitunter noch nicht unterstützt.
- **Energiesparfunktionen:** Gerade neue Notebooks haben unter Linux oft deutlich kürzere Akku-Laufzeiten als unter Windows. Dieses Ärgernis resultiert daraus, dass das Zusammenspiel diverser Energiesparfunktionen optimale Treiber voraussetzt, die für Linux oft gar nicht oder erst ein, zwei Jahre nach der Markteinführung verfügbar sind.

Stellen Sie also *vor* dem Kauf eines neuen Rechners bzw. einer Hardware-Erweiterung sicher, dass alle Komponenten von Linux unterstützt werden. Auch eine Internetsuche nach *linux <hardwarename>* kann nicht schaden. Lesenswert sind außerdem Testberichte der Zeitschrift c't: Deren Redakteure machen sich bei den meisten Geräten die Mühe, auch die Linux-Kompatibilität zu testen.

Lieber etwas älter

Um ganz neue Notebooks mache ich beim Kauf in der Regel einen großen Bogen, auch wenn die Spezifikationen noch so verlockend sind. Sie verursachen allzu oft Treiberprobleme und verursachen Zusatzarbeit bei Installation und Konfiguration. Der sparen Geld und vermeiden es, sich beim Konfigurieren zu ärgern, wenn Sie sich für ein Vorjahresmodell entscheiden!

1.3 Distributionen

Noch immer ist die einleitende Frage – Was ist Linux? – nicht ganz beantwortet. Viele Anwender interessiert der Kernel nämlich herzlich wenig. Für sie umfasst der Begriff Linux, wie er umgangssprachlich verwendet wird, neben dem Kernel auch das riesige Bündel mitgelieferter Programme: Dazu zählen unzählige Kommandos, ein Desktop-System (z. B. KDE oder Gnome), LibreOffice, Firefox, GIMP sowie zahllose Programmiersprachen und Server-Programme (Webserver, Mail-Server etc.).

Als Linux-Distribution wird die Einheit bezeichnet, die aus dem eigentlichen Betriebssystem (Kernel) und den vielen Zusatzprogrammen gebildet wird. Eine Distribution ermöglicht eine rasche und bequeme Installation von Linux. Die meisten Distributionen können kostenlos aus dem Internet heruntergeladen werden.

Distributionen unterscheiden sich vor allem durch folgende Punkte voneinander:

- **Umfang, Aktualität:** Die Anzahl, Auswahl und Aktualität der mitgelieferten Programme und Bibliotheken variiert stark. Manche Distributionen setzen bewusst auf etwas ältere, stabile Versionen – z. B. Debian.
- **Installations- und Konfigurationswerkzeuge:** Die mitgelieferten Programme zur Installation, Konfiguration und Wartung des Systems helfen dabei, die Konfigurationsdateien einzustellen. Das kann viel Zeit sparen.
- **Konfiguration des Desktops (KDE, Gnome):** Manche Distributionen lassen dem Anwender die Wahl zwischen KDE, Gnome und anderen Desktop-Systemen. Auch die Detailkonfiguration und optische Gestaltung variiert je nach Distribution.
- **Hardware-Unterstützung:** Linux kommt mit den meisten PC-Hardware-Komponenten zurecht. Dennoch gibt es im Detail Unterschiede zwischen den Distributionen, insbesondere wenn es darum geht, Nicht-Open-Source-Treiber (z. B. für NVIDIA-Grafikkarten) in das System zu integrieren.
- **Updates:** Sie können eine Linux-Distribution nur so lange sicher betreiben, wie Sie Updates bekommen. Danach sollten Sie auf eine neue Version der Distribution wechseln. Deswegen ist es bedeutsam, wie lange es für eine Distribution Updates gibt. Hier gilt meist die Grundregel: je teurer der kommerzielle Support, desto länger der Zeitraum. Einige Beispiele (Stand: Sommer 2023):

Debian:	3 Jahre (mit Einschränkungen 5)
Fedora:	13 Monate
openSUSE:	ca. 18 bis 24 Monate
Red Hat Enterprise Linux (RHEL):	10 Jahre (mit Einschränkungen sogar 13 Jahre)
RHEL-Klone:	bis zu 10 Jahre
SUSE Enterprise Server:	10 Jahre (mit Einschränkungen sogar 13 Jahre)
Ubuntu LTS:	3 bis 5 Jahre (mit Pro-Upgrade: 10 Jahre)
Ubuntu (sonstige Versionen):	9 Monate

- **Rolling Release:** Alle oben aufgezählten Distributionen unterscheiden explizit zwischen Versionen. Ubuntu 23.10 enthält also andere Versionen von Gnome, LibreOffice und GIMP als Ubuntu 24.04.

  Es gibt aber auch Distributionen, die das *Rolling-Release*-Modell anwenden, z. B. Arch Linux oder openSUSE Tumbleweed: Dort erhalten Sie mit Updates stets die neueste Version *jeder* installierten Software-Komponente. Das klingt praktisch, kann aber zu Stabilitätsproblemen führen. Deswegen sind Rolling-Release-Distributionen im Server-Bereich nicht üblich. Sie sprechen eher fortgeschrittene Linux-Anwender an, die Software entwickeln oder Systeme administrieren und die kein Problem damit haben, nach einem Update die eine oder andere Konfigurationsdatei anzupassen, wenn etwas nicht mehr funktioniert.
- **Live-System:** Viele Distributionen ermöglichen den Linux-Betrieb direkt von einem USB-Stick. Das ermöglicht ein einfaches Ausprobieren. Außerdem bieten derartige Live-Systeme eine gute Möglichkeit, um ein defektes Linux-System zu reparieren bzw. die betreffende Distribution neu zu installieren.
- **Zielplattform (CPU-Architektur):** Viele Distributionen sind nur für Intel- und AMD-kompatible Prozessoren erhältlich. Es gibt aber auch Distributionen für andere Prozessorplattformen (ARM, SPARC etc.).
- **Support:** Wenn Sie sich eine kommerzielle Distribution leisten, erhalten Sie Hilfe bei der Installation und im Betrieb.
- **Lizenz:** Die meisten Distributionen sind kostenlos erhältlich. Bei einigen Distributionen gibt es hier aber Einschränkungen: Beispielsweise ist bei den Enterprise-Distributionen von Red Hat und SUSE ein Zugriff auf das Update-System nur für registrierte Kunden möglich. Sie zahlen hier nicht für die Software an sich, wohl aber für das Service-Angebot rundherum.

Linux Standard Base (LSB)

Das Linux-Standard-Base-Projekt (LSB) definiert Regeln, um einen gemeinsamen Nenner zwischen den Distributionen zu schaffen. Die meisten Distributionen sind LSB-konform:

https://wiki.linuxfoundation.org/lsb/start

Gängige Linux-Distributionen

Der folgende Überblick über die wichtigsten verfügbaren Distributionen soll Ihnen eine erste Orientierungshilfe geben. Die Liste ist alphabetisch geordnet und erhebt keinen Anspruch auf Vollständigkeit.

AlmaLinux

AlmaLinux ist ein RHEL-Klon, also eine zu Red Hat Enterprise Linux kompatible Distribution. AlmaLinux hat zusammen mit Rocky Linux die Nachfolge von CentOS Linux angetreten.

Android ist eine von Google entwickelte Plattform für Mobilfunkgeräte und Tablets. Android hat damit Linux zu der Weltdominanz verholfen, über die Linux-Entwickler in der Vergangenheit gescherzt haben. Android ist aber ungeeignet für eine PC-Installation und insofern keine »echte« Distribution. **Android**

Arch Linux ist eine für technische Anwender optimierte Rolling-Release-Distribution. Wegen der relativ komplizierten, im Textmodus durchzuführenden Installation machen Einsteiger zumeist einen großen Bogen um Arch Linux. Dafür zählen *https://wiki.archlinux.org* und *https://wiki.archlinux.de* zu den besten Quellen für Linux-Konfigurationsdetails im Netz. **Arch Linux**

Arch-Linux-Derivate wie **Manjaro** und **EndeavourOS** mit grafischen Installations- und Konfigurationsprogrammen haben Arch Linux zuletzt sogar in die Top-10-Liste von *distrowatch.com* gebracht.

CentOS war eine kostenlose Variante zu Red Hat Enterprise Linux (RHEL) und hatte eine riesige Installationsbasis. Allerdings hat Red Hat im Dezember 2020 das Ende von CentOS in seiner bisherigen Form verkündet. **CentOS**

CentOS Stream soll die Nachfolge von CentOS antreten. Diese Variante unterscheidet sich aber in zwei wichtigen Details vom ursprünglichen CentOS: Zum einen ist der Wartungszeitraum wesentlich kürzer und beträgt nur 4 bis 5 Jahre anstelle von bisher 10 Jahren. **CentOS Stream**

Zum anderen werden die meisten Paket-Updates (ausgenommen sind Sicherheits-Updates, die einem *Non-disclosure Agreement* unterliegen) zuerst für CentOS freigegeben, bevor sie für RHEL zum Einsatz kommen. Das scheint auf den ersten Blick ein Vorteil zu sein. Tatsächlich geht damit aber die vollständige Kompatibilität zu RHEL verloren. Außerdem werden CentOS-Nutzer damit zu Beta-Testern für Updates. CentOS Stream ist für den längerfristigen Produktiveinsatz ungeeignet.

Das **Chrome OS** wird wie Android von Google entwickelt. Es ist für Notebooks optimiert und setzt zur Nutzung eine aktive Internetverbindung voraus. Die Benutzeroberfläche basiert auf dem Webbrowser Google Chrome. Chrome OS spielt aktuell in Europa keine große Rolle, wohl aber auf dem Bildungsmarkt in den USA: Dort werden billige Chrome-Books (also Notebooks mit Chrome OS) häufig in Schulen eingesetzt. **Chrome OS**

Debian ist die älteste vollkommen freie Distribution. Sie wird von engagierten Linux-Entwicklern zusammengestellt, wobei die Einhaltung der Spielregeln »freier« Software eine hohe Priorität genießt. Die strikte Auslegung dieser Philosophie hat in der Vergangenheit mehrfach zu Verzögerungen geführt. **Debian**

Debian richtet sich an fortgeschrittene Linux-Anwender und hat einen großen Marktanteil bei Server-Installationen. Im Vergleich zu anderen Distributionen ist Debian

stark auf maximale Stabilität hin optimiert und enthält deswegen oft nicht die neuesten Programmversionen. Dafür steht Debian für neun Hardware-Plattformen zur Verfügung. Viele andere Distributionen sind von Debian abgeleitet, z. B. Raspberry Pi OS und Ubuntu.

Fedora **Fedora** ist der kostenlose Entwicklungszweig von Red Hat Linux. Die Entwicklung wird von Red Hat unterstützt und gelenkt. Für Red Hat ist Fedora eine Art Spielwiese, auf der neue Funktionen ausprobiert werden können, ohne die Stabilität der Enterprise-Versionen zu gefährden. Programme, die sich unter Fedora bewähren, werden später in die Enterprise-Versionen integriert. Bei technisch interessierten Linux-Fans ist Fedora beliebt, weil diese Distribution oft eine Vorreiterrolle spielt: Neue Linux-Funktionen finden sich oft zuerst in Fedora und erst später in anderen Distributionen. Neue Fedora-Versionen erscheinen alle sechs Monate. Updates werden einen Monat nach dem Erscheinen der übernächsten Version eingestellt, d. h., die Lebensdauer ist mit 13 Monaten sehr kurz.

Kali Linux Das auf Debian basierende **Kali Linux** enthält eine riesige Sammlung von Hacking- und Pen-Testing-Werkzeugen. Die Distribution gilt als *der* Werkzeugkasten für Hacker und Sicherheitsexperten.

openSUSE **openSUSE** ist eine kostenlose Linux-Distribution, die auf den Enterprise-Versionen von SUSE basiert, sich aber speziell an Privatanwender und Entwickler wendet. Beachten Sie, dass ich in diesem Buch oft einfach von »SUSE« schreibe, wenn ich sowohl die kommerziellen Angebote von SUSE als auch das Community-Angebot openSUSE meine. In vielen Details verhalten sich die Distributionen einheitlich.

Oracle Oracle bietet unter dem Namen **Oracle Linux** eine Variante zu Red Hat Enterprise Linux (RHEL) an. Das ist aufgrund der Open-Source-Lizenzen eine zulässige Vorgehensweise. Technisch gibt es nur wenige Unterschiede zu RHEL, die Oracle-Variante ist aber billiger und ohne Support sogar kostenlos verfügbar.

Raspberry Pi OS **Raspberry Pi OS** ist die Standarddistribution für den beliebten Minicomputer Raspberry Pi. Raspberry Pi OS basiert auf Debian, wurde für den Raspberry Pi aber speziell adaptiert und erweitert.

Red Hat Die 2018 von IBM übernommene Firma **Red Hat** ist das international bekannteste und erfolgreichste Linux-Unternehmen. Red-Hat-Distributionen dominieren insbesondere den amerikanischen Markt. Die Paketverwaltung auf der Basis des *Red Hat Package Formats* (RPM) wurde von vielen anderen Distributionen übernommen.

Red Hat ist überwiegend auf Unternehmenskunden ausgerichtet. Die Enterprise-Versionen (RHEL = **Red Hat Enterprise Linux**) sind vergleichsweise teuer. Sie zeichnen sich durch hohe Stabilität und einen zehnjährigen Update-Zeitraum aus. Für Linux-Enthusiasten und -Entwickler, die ein Red-Hat-ähnliches System zum Nulltarif

suchen, bieten sich AlmaLinux, CentOS Stream, Fedora, Oracle Linux oder Rocky Linux an.

Rocky Linux

Nach dem Ende von CentOS Linux in seiner bisherigen Form kämpfen neue RHEL-Klone um dessen Nachfolge. Rocky Linux zählt neben AlmaLinux zu den wichtigsten Anwärtern und hat zuletzt stark an Popularität sowie an Unterstützung durch namhafte Firmen gewonnen.

SUSE

SUSE ist nach diversen Übernahmen die wichtigste deutsche Linux-Firma. Ihr Hauptprodukt, **SUSE Enterprise**, ist vor allem im europäischen Markt verankert. Seit 2021 ist SUSE als Aktienunternehmen an der Börse vertreten. Im August 2023 gab der Finanzinvestor EQT allerdings bekannt, die Firma wieder von der Börse nehmen zu wollen.

Ubuntu

Ubuntu ist die zurzeit populärste Distribution für Privatanwender. Ubuntu verwendet als Basis Debian, ist aber besser für Desktop-Anwender optimiert (Motto: *Linux for human beings*). Die kostenlose Distribution erscheint im Halbjahresrhythmus. Für gewöhnliche Versionen werden Updates über neun Monate zur Verfügung gestellt. Für die alle zwei Jahre erscheinenden Versionen mit Long Time Support (LTS) gibt es sogar 3 bis 5 Jahre lang Updates.

Für kommerzielle Kunden bietet die Firma Canonical diverse Support-Angebote, unter anderem **Ubuntu Pro**: Dieses zahlungspflichtige Upgrade erweitert den Update-Zeitraum von LTS-Versionen auf zehn Jahre. Canonical hat sich damit vor allem im Server- und Cloud-Sektor zu den weltweit wichtigsten Linux-Firmen entwickelt.

Ubuntu-Derivate

Zu Ubuntu gibt es eine Menge offizieller und inoffizieller Varianten. Etabliert und weit verbreitet sind **Ubuntu Server**, **Kubuntu**, **Xubuntu**, **Ubuntu MATE** und **Linux Mint**. Die amerikanische Firma System76 pflegt mit **Pop!_OS** eine Ubuntu-Variante, die speziell für Notebooks optimiert ist und NVIDIA-Grafikkarten besonders gut unterstützt. Interessant ist auch **KDE Neon**: Diese Distribution kombiniert Ubuntu LTS mit stets aktuellen KDE-Paketen und ist insofern bei KDE-Fans beliebt.

Andere Distributionen

Neben den oben aufgezählten »großen« Distributionen gibt es im Internet zahlreiche Zusammenstellungen von Miniatursystemen. Sie sind vor allem für Spezialaufgaben konzipiert, etwa für Wartungsarbeiten (Emergency-Systeme) oder um ein Linux-System ohne eigentliche Installation verwenden zu können (Live-Systeme). Populäre Vertreter dieser Linux-Gattung sind **Parted Magic**, **Slax** und **TinyCore**.

Einen ziemlich guten Überblick über alle momentan verfügbaren Linux-Distributionen, egal ob kommerziellen oder anderen Ursprungs, finden Sie im Internet auf der folgenden Seite:

https://distrowatch.com

Die Qual der Wahl

Eine Empfehlung für eine bestimmte Distribution ist schwierig. Für Linux-Einsteiger ist es zumeist von Vorteil, sich vorerst für eine weitverbreitete Distribution wie Debian, Fedora, openSUSE oder Ubuntu zu entscheiden. Eine gute Wahl ist auch Linux Mint. Zu diesen Distributionen sind sowohl im Internet als auch im Buch- und Zeitschriftenhandel viele Informationen verfügbar. Bei Problemen ist es vergleichsweise leicht, Hilfe zu finden.

Kommerzielle Linux-Anwender bzw. Server-Administratoren müssen sich entscheiden, ob sie bereit sind, für professionellen Support Geld auszugeben. In diesem Fall spricht wenig gegen die Marktführer Red Hat, Ubuntu und SUSE. Andernfalls sind AlmaLinux, Debian, Oracle Linux, Rocky Linux und Ubuntu attraktive kostenlose Alternativen.

1.4 Open-Source-Lizenzen (GPL & Co.)

Die Grundidee von »Open Source« besteht darin, dass der Quellcode von Programmen frei verfügbar ist und von jedem erweitert bzw. geändert werden darf. Allerdings ist damit auch eine Verpflichtung verbunden: Wer Open-Source-Code zur Entwicklung eigener Produkte verwendet, muss den gesamten Code ebenfalls wieder frei weitergeben.

Die Open-Source-Idee verbietet übrigens keinesfalls den Verkauf von Open-Source-Produkten. Auf den ersten Blick scheint das ein Widerspruch zu sein. Tatsächlich bezieht sich die Freiheit in »Open Source« mehr auf den Code als auf das fertige Produkt. Zudem regelt die freie Verfügbarkeit des Codes auch die Preisgestaltung von Open-Source-Produkten: Nur wer neben dem Kompilat eines Open-Source-Programms weitere Zusatzleistungen anbietet (Handbücher, Support etc.), wird überleben. Sobald der Preis in keinem vernünftigen Verhältnis zu den Leistungen steht, werden sich andere Firmen finden, die es günstiger machen.

General Public License (GPL)

Das Ziel der Open-Source-Entwickler ist es, Software zu schaffen, deren Quellen frei verfügbar sind und es auch bleiben. Um einen Missbrauch auszuschließen, sind viele Open-Source-Programme durch die *GNU General Public License* (kurz GPL) geschützt. Hinter der GPL steht die *Free Software Foundation* (FSF). Diese Organisation wurde von Richard Stallman gegründet, um hochwertige Software frei verfügbar zu machen. Richard Stallman ist übrigens auch der Autor des Editors Emacs, der in Kapitel 17 beschrieben wird.

Die Kernaussage der GPL besteht darin, dass zwar jeder den Code verändern und sogar die resultierenden Programme verkaufen darf, dass aber gleichzeitig der Anwender/Käufer das Recht auf den vollständigen Code hat und diesen ebenfalls verändern und wieder kostenlos weitergeben darf. Jedes GNU-Programm muss zusammen mit dem vollständigen GPL-Text weitergegeben werden. Die GPL schließt damit aus, dass

jemand ein GPL-Programm weiterentwickeln und verkaufen kann, *ohne* die Veränderungen öffentlich verfügbar zu machen. Jede Weiterentwicklung ist somit ein Gewinn für *alle* Anwender. Den vollständigen Text der GPL finden Sie hier:

https://gnu.org/licenses/gpl.html

Das Konzept der GPL ist recht einfach zu verstehen, im Detail treten aber immer wieder Fragen auf. Viele davon werden hier beantwortet:

https://gnu.org/licenses/gpl-faq.html

Wenn Sie glauben, dass Sie alles verstanden haben, sollten Sie das GPL-Quiz ausprobieren:

https://gnu.org/cgi-bin/license-quiz.cgi

Lesser General Public License (LGPL)

Neben der GPL existiert noch die Variante LGPL (Lesser GPL). Der wesentliche Unterschied zur GPL besteht darin, dass eine derart geschützte Bibliothek auch von kommerziellen Produkten genutzt werden darf, deren Code *nicht* frei verfügbar ist. Ohne die LGPL könnten GPL-Bibliotheken nur wieder für GPL-Programme genutzt werden, was in vielen Fällen eine unerwünschte Einschränkung für kommerzielle Programmierer wäre.

Andere Lizenzen

Durchaus nicht alle Teile einer Linux-Distribution unterliegen den gleichen Copyright-Bedingungen! Obwohl der Kernel und viele Tools der GPL unterliegen, gelten für manche Komponenten und Programme andere rechtliche Bedingungen:

- **MIT- und BSD-Lizenz:** Die MIT- und BSD-Lizenzen erlauben die kommerzielle Nutzung des Codes *ohne* die Verpflichtung, Änderungen öffentlich weiterzugeben. Die Lizenzen sind damit wesentlich liberaler als die GPL und eher mit der LGPL vergleichbar.
- **Doppellizenzen:** Für einige Programme gelten Doppellizenzen. Beispielsweise können Sie den Datenbank-Server MySQL für Open-Source-Projekte, auf einem eigenen Webserver bzw. für die innerbetriebliche Anwendung gemäß der GPL kostenlos einsetzen. Wenn Sie hingegen ein kommerzielles Produkt auf der Basis von MySQL entwickeln und samt MySQL verkaufen möchten, ohne Ihren Quellcode zur Verfügung zu stellen, dann kommt die kommerzielle Lizenz zum Einsatz. Die Weitergabe von MySQL wird in diesem Fall kostenpflichtig.
- **Kommerzielle Lizenzen:** Einige Programme unterstehen zwar einer kommerziellen Lizenz, dürfen aber dennoch kostenlos genutzt werden. Der Quellcode bleibt dann ein Firmengeheimnis. Zwei Beispiele dafür sind Microsoft Teams und Skype.

Manche Distributionen kennzeichnen die Produkte, bei denen die Nutzung oder Weitergabe eventuell lizenzrechtliche Probleme verursachen könnte. Bei Debian befinden sich solche Programme in der Paketquelle *non-free*.

Das Dickicht der zahllosen, mehr oder weniger »freien« Lizenzen ist schwer zu durchschauen. Groß ist die Bandbreite zwischen der manchmal fundamentalistischen Auslegung von »frei« im Sinne der GPL und den verklausulierten Bestimmungen mancher Firmen, die ihr Software-Produkt zwar frei nennen möchten (weil dies gerade modern ist), in Wirklichkeit aber uneingeschränkte Kontrolle über den Code behalten möchten. Eine gute Einführung in das Thema finden Sie hier:

https://opensource.org
https://heise.de/-221957

Lizenzkonflikte zwischen Open- und Closed-Source-Software

Open-Source-Lizenzen für Entwickler

Wenn Sie Programme entwickeln und diese zusammen mit Linux bzw. in Kombination mit Open-Source-Programmen oder -Bibliotheken verkaufen möchten, müssen Sie sich in die bisweilen verwirrende Problematik der unterschiedlichen Software-Lizenzen tiefer einarbeiten. Viele Open-Source-Lizenzen erlauben die Weitergabe nur, wenn auch Sie Ihren Quellcode im Rahmen einer Open-Source-Lizenz frei verfügbar machen. Auf je mehr Open-Source-Komponenten mit unterschiedlichen Lizenzen Ihr Programm basiert, desto komplizierter wird die Weitergabe.

Es gibt aber auch Ausnahmen, die die kommerzielle Nutzung von Open-Source-Komponenten erleichtern: Beispielsweise gilt für Apache und PHP sinngemäß, dass Sie diese Programme auch in Kombination mit einem Closed-Source-Programm frei weitergeben dürfen.

GPL-Probleme mit Hardware-Treibern

Manche proprietäre Treiber für Hardware-Komponenten (z. B. für NVIDIA-Grafikkarten) bestehen aus einem kleinen Kernelmodul (Open Source) und diversen externen Programmen oder Bibliotheken, deren Quellcode nicht verfügbar ist (Closed Source). Das Kernelmodul hat nur den Zweck, eine Verbindung zwischen dem Kernel und dem Closed-Source-Treiber herzustellen.

Diese Treiber sind aus Sicht vieler Linux-Anwender eine gute Sache: Sie sind kostenlos verfügbar und ermöglichen es, diverse Hardware-Komponenten zu nutzen, zu denen es entweder gar keine oder zumindest keine vollständigen Open-Source-Treiber für Linux gibt.

Die Frage ist aber, ob bzw. in welchem Ausmaß die Closed-Source-Treiber wegen der engen Verzahnung mit dem Kernel, der ja der GPL untersteht, diese Lizenz verletzen. Viele Open-Source-Entwickler dulden die Treiber nur widerwillig. Eine direkte Weitergabe mit GPL-Produkten ist nicht zulässig, weswegen der Benutzer die Treiber in der Regel selbst herunterladen und installieren muss.

1.5 Die Geschichte von Linux

1982: GNU

Da Linux ein Unix-ähnliches Betriebssystem ist, müsste ich an dieser Stelle eigentlich mit der Geschichte von Unix beginnen – aber dazu fehlt hier der Platz. Stattdessen beginnt diese Geschichtsstunde mit der Gründung des GNU-Projekts durch Richard Stallman. GNU steht für *GNU is not Unix*. In diesem Projekt wurden seit 1982 Open-Source-Werkzeuge entwickelt. Dazu zählen der GNU-C-Compiler, der Texteditor Emacs sowie diverse GNU-Utilities wie `find` und `grep` etc.

1989: GPL

Erst sieben Jahre nach dem Start des GNU-Projekts war die Zeit reif für die erste Version der *General Public License*. Diese Lizenz stellt sicher, dass freier Code frei bleibt.

1991: Linux-Kernel 0.01

Die allerersten Teile des Linux-Kernels (Version 0.01) entwickelte Linus Torvalds. Er gab seinen Code im September 1991 über das Internet frei. Schnell fanden sich weltweit Programmierer, die an der Idee Interesse hatten und Erweiterungen dazu schrieben. Als der Kernel von Linux die Ausführung des GNU-C-Compilers erlaubte, stand auch die gesamte Palette der GNU-Tools zur Verfügung. Weitere Komponenten waren das Dateisystem Minix, Netzwerk-Software von BSD-Unix, das X Window System des MIT und dessen Portierung XFree86 etc.

Linux ist also nicht nur Linus Torvalds zu verdanken. Hinter Linux stehen vielmehr eine Menge engagierter Menschen, die in ihrer Freizeit, im Rahmen ihres Studiums oder bezahlt von Firmen wie Google, IBM oder HP freie Software produzieren.

1994: Erste Distributionen

Informatik-Freaks an Universitäten konnten sich Linux und seine Komponenten selbst herunterladen, kompilieren und installieren. Eine breite Anwendung fand Linux aber erst mit Linux-Distributionen, die Linux und die darum entstandene Software auf Disketten bzw. CD-ROMs verpackten und mit einem Installationsprogramm versahen. Vier der zu dieser Zeit entstandenen Distributionen existieren heute noch: Debian, Red Hat, Slackware und SUSE.

1996: Pinguin

1996 wurde der Pinguin Tux zum Linux-Logo.

1998: Microsoft nimmt Linux wahr

Mit dem rasanten Siegeszug des Internets stieg auch die Verbreitung von Linux, vor allem auf Servern. Gewissermaßen zum Ritterschlag für Linux wurde der legendäre Ausspruch von Steve Ballmer: *Microsoft is worried about free software* … Ein Jahr später ging Red Hat spektakulär an die Börse.

2000er-Jahre: Linux wird zum Fundament für das Internet und die Cloud

Der gemeinsame Nenner von Amazon (gegründet 1994), Google (1998), Wikipedia (2001), Facebook (2006) und Dropbox (2007) besteht darin, dass all diese Firmen bzw. Organisationen für ihren Server-Betrieb auf Linux setzen. Das Internet, wie es sich seit den 2000er-Jahren entwickelt, und die daraus entwachsene Cloud-Infrastruktur sind ohne Linux undenkbar.

2009: Android

Mit der Android-Plattform brachte Google Linux zuerst auf das Handy (2009), danach auch auf Tablets und in TV-Geräte.

2012: Raspberry Pi

2012 eroberte der Minicomputer Raspberry Pi die Herzen von Elektronikbastlern. Für nur rund 40 EUR können Sie mit dem Raspberry Pi selbst Hardware-Experimente durchführen, in die Welt der Heimautomation einsteigen, ein Medien-Center oder einen Home-Server betreiben. Der Raspberry Pi macht Embedded Linux zu einem Massenphänomen.

2018: IBM kauft Red Hat, Microsoft umarmt Open Source und Linux

2018 erwarb IBM die Fima Red Hat für stattliche 34 Mrd. US-Dollar. Gleichzeitig wandte sich Microsoft unter der Führung von Satya Nadella immer stärker der Open-Source-Idee und Linux zu: Das *Windows Subsystem for Linux* erlaubt die Ausführung von Linux direkt in Windows. Mit dem Kauf von GitHub, der ebenfalls 2018 über die Bühne ging, beherrscht Microsoft nun das wichtigste Repository für Open-Source-Projekte.

2021: SUSE geht an die Börse

Im Mai 2021 wurde SUSE zu einem börsennotierten Unternehmen. Die anfängliche Bewertung des Unternehmens fällt mit knapp 6 Mrd. EUR durchaus beachtlich aus. Der langfristige Erfolg bleibt allerdings aus. Schon im Sommer 2023 wurde der Rückzug von der Börse angekündigt.

Kapitel 8
Arbeiten im Terminal

Vom Desktop ins Terminal

Bis jetzt habe ich Ihnen Linux in erster Linie als Desktop-System präsentiert. Sie haben diverse Programme kennengelernt, die vielleicht ein wenig anders aussehen als unter Windows oder macOS, aber letztlich denselben Zweck erfüllen und ähnlich zu bedienen sind. Der Umgang mit Linux endet allerdings nicht an dieser Stelle. Es gibt quasi noch eine andere Seite von Linux, die auf den ersten Blick abschreckend wirken mag: Erfahrene Linux-Anwender führen in Terminalfenstern bzw. Textkonsolen Kommandos aus und erhalten die Resultate wiederum in Textform. Die Maus spielt nur noch eine Nebenrolle, grafische Benutzeroberflächen sind passé.

Wenn Sie einmal mit der Arbeit in Terminalfenstern vertraut sind, können Sie dort viele Aufgaben effizient ausführen: Sie können Linux-Kommandos miteinander verknüpfen, im Hintergrund ausführen, in kleinen Programmen (Scripts) automatisieren etc. All diese Möglichkeiten stehen Ihnen auch dann zur Verfügung, wenn Sie nicht lokal am Rechner sitzen, sondern nur über eine Netzwerkverbindung verfügen.

Reine Büroanwender werden für Terminalfenster seltener Verwendung finden als Programmierer oder Netzwerkadministratoren. Auf jeden Fall gehört die Arbeit im Terminal zum elementaren Handwerkszeug jedes Anwenders, der Linux richtig kennenlernen will. Das merken Sie spätestens dann zum ersten Mal, wenn das Grafiksystem wegen einer Fehlkonfiguration nicht funktioniert oder wenn Sie Ihren externen Root-Server administrieren möchten.

Querverweise

Dieses Kapitel gibt lediglich einen ersten Überblick über Arbeitstechniken in Terminalfenstern bzw. Konsolen. Für die Ausführung der Programme im Terminal ist eine sogenannte Shell verantwortlich. Unter Linux stehen mehrere Shells zur Auswahl. Am häufigsten kommt die `bash` zum Einsatz, deren Grundfunktionen Thema des nächsten Kapitels sind. Alternativ dazu spricht die `zsh` fortgeschrittene Benutzer an (siehe Kapitel 10).

Die weiteren Kapitel stellen dann diverse Linux-Kommandos näher vor. Diese dienen beispielsweise zur Verwaltung des Dateisystems (`ls`, `cp`, `mv`, `ln`, `rm` etc.), zur Suche nach Dateien (`find`, `grep`, `locate`) oder zur Steuerung von Netzwerkfunktionen (`ping`, `ip`, `ssh`) etc. Nebenbei werden Sie eine Menge Linux-Grundlagen lernen.

8.1 Textkonsolen und Terminalfenster

Textkonsolen

Microsoft Windows nutzen Sie ausschließlich im Grafikmodus. Linux können Sie dagegen auch in sogenannten Textkonsolen nutzen. Bei den meisten Distributionen stehen sechs Textkonsolen zur Verfügung. Sofern der Textmodus bereits aktiv ist, wechseln Sie mit Alt+F1 in die erste Konsole, mit Alt+F2 in die zweite etc. Alt+← bzw. Alt+→ springen in die vorige bzw. nächste Konsole.

Wenn der Rechner dagegen im Grafikmodus läuft, führt Strg+Alt+F# in die #-te Textkonsole. Allerdings gibt es unter den Distributionen keine Einigkeit, welche Konsole für welche Aufgabe vorgesehen ist. Bei vielen Distributionen läuft in der ersten Konsole der Display-Manager mit der Login-Box und in der zweiten Konsole das Desktop-System. Damit sind die ersten beiden Konsolen blockiert und Strg+Alt+F3 führt in die erste freie Textkonsole.

Zurück in das Desktop-System gelangen Sie bei den meisten Distributionen mit Alt+F2, je nach Distribution kann aber auch Alt+F1 oder Alt+F7 zum Ziel führen. Probieren Sie es einfach aus!

Bevor Sie in einer Textkonsole arbeiten können, müssen Sie sich einloggen. Wenn Sie mit der Arbeit fertig sind oder wenn Sie sich unter einem anderen Namen anmelden möchten, müssen Sie sich wieder ausloggen. Dazu drücken Sie einfach Strg+D.

Tastenkürzel	Funktion
Strg+Alt+F#	vom Grafikmodus in die Textkonsole # wechseln
Alt+F#	von einer Textkonsole in eine andere Textkonsole # wechseln
Alt+F2	zurück in den Grafikmodus wechseln (je nach Distribution auch Alt+F1 oder Alt+F7)
Alt+→ / +←	in die vorige/nächste Textkonsole wechseln
⇧+Bild↑/Bild↓	vorwärts/rückwärts durch die Textausgaben der Konsole blättern
Strg+Alt+Entf	Linux beenden (nur in Textkonsolen, führt `shutdown` aus, Vorsicht!)

Tabelle 8.1 Tastenkürzel zum Aktivieren von Textkonsolen

Sie können in der einen Konsole ein Kommando starten, und während dieses Kommando läuft, können Sie in der zweiten Konsole etwas anderes erledigen. Sie können sich auch in einer Konsole als `root` anmelden, um administrative Aufgaben zu erledi-

gen, während Sie in der anderen Konsole unter Ihrem normalen Login-Namen eine Datei editieren. Jede Konsole läuft also vollkommen unabhängig von den anderen.

Mit ⇧+Bild↑ und ⇧+Bild↓ scrollen Sie den Bildschirminhalt einer Textkonsole auf und ab. Auf diese Weise können Sie die Ergebnisse der zuletzt ausgeführten Programme nochmals ansehen, auch wenn sie bereits aus dem sichtbaren Bildschirmbereich hinausgeschoben wurden.

Terminalfenster

Üblicherweise arbeiten Sie nur dann in Textkonsolen, wenn es keine andere Möglichkeit gibt – beispielsweise auf einem Server, wo gar kein Desktop-System zur Verfügung steht, oder bei Konfigurationsproblemen, wenn sich das Grafiksystem nicht starten lässt.

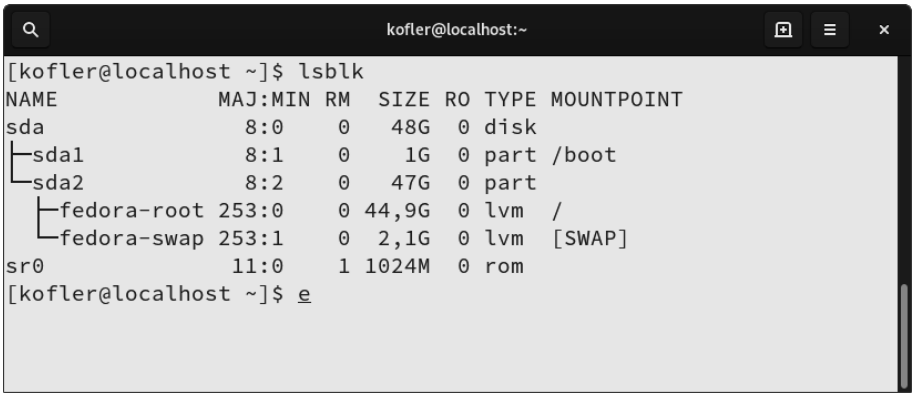

Abbildung 8.1 Ein Terminalfenster

Der Normalfall sieht dagegen so aus, dass Sie sich zuerst in Ihr Desktop-System einloggen und dann zum Ausführen von Kommandos ein sogenanntes *Terminalfenster* öffnen (siehe Abbildung 8.1). Je nach Distribution und Desktop-System stehen unterschiedliche Terminalprogramme zur Auswahl. Am populärsten sind `gnome-terminal` oder `kgx` (beide Gnome) bzw. `konsole` (KDE). Auch das Menükommando zum Starten eines Terminalfensters variiert je nach Distribution – hier zwei Beispiele:

Distributionen mit Gnome: ⊞ `terminal`
Distributionen mit KDE: Anwendungen • System • Terminal

In Terminalfenstern können Sie grundsätzlich wie in einer Textkonsole arbeiten. Der größte Vorteil besteht darin, dass Sie dank einer Bildlaufleiste bequemer durch die bisherigen Ausgaben scrollen können.

Wichtige Tastenkürzel

Innerhalb von Textkonsolen bzw. Terminalfenstern helfen diverse Tastenkürzel bei der effizienten Eingabe von Kommandos. Tabelle 8.2 fasst die wichtigsten Kürzel zusammen. Sie gelten nur, wenn Sie die `bash` in der Standardkonfiguration als Shell verwenden, was bei den meisten Distributionen der Fall ist. Wenn Sie unter Gnome in einem Terminalfenster arbeiten, sollten Sie BEARBEITEN • TASTENKOMBINATIONEN ausführen und die Option ALLE MENÜKÜRZELBUCHSTABEN AKTIVIEREN deaktivieren.

Tastenkürzel	Funktion
Strg+A	Cursor an den Zeilenanfang (wie Pos1)
Strg+C	Programm abbrechen
Strg+E	Cursor an das Ende der Zeile (wie Ende)
Strg+K	Zeile ab Cursor löschen
Strg+Y	zuletzt gelöschten Text wieder einfügen
Strg+Z	Programm unterbrechen (Fortsetzung mit `fg` oder `bg`)
⇆	Datei- und Kommandonamen vervollständigen
↑ / ↓	durch die bisher ausgeführten Kommandos blättern

Tabelle 8.2 Tastenkürzel zur Kommandoeingabe in der bash

Insbesondere die Kommandoerweiterung mit ⇆ spart eine Menge Tipparbeit. Sie brauchen nur die ersten Buchstaben eines Kommandos oder einer Datei angeben. Anschließend drücken Sie ⇆. Wenn der Dateiname bereits eindeutig erkennbar ist, wird er vollständig ergänzt, sonst nur so weit, bis sich mehrere Möglichkeiten ergeben. Ein zweimaliges Drücken von ⇆ bewirkt, dass eine Liste aller Dateinamen angezeigt wird, die mit den bereits eingegebenen Anfangsbuchstaben beginnen. Im Detail ist dieser Mechanismus in Abschnitt 9.3, »Kommandoeingabe«, beschrieben.

Menü im »gnome-terminal« ein- und ausblenden

In `gnome-terminal` können Sie mit ANSICHT • MENÜLEISTE das Menü ausblenden. Häufig ist das Programm standardmäßig so konfiguriert. Aber wie blenden Sie das Menü wieder ein? Dazu klicken Sie das Terminal mit der rechten Maustaste an oder drücken ⇧+F10. Im nun sichtbaren Kontextmenü finden Sie das Kommando MENÜLEISTE ANZEIGEN.

Zwischenablage

Da im Terminal eine Menge Tastenkürzel mit Strg eine vorreservierte Bedeutung haben, müssen Sie beim Kopieren und Einfügen von Text umdenken: Um ein zuvor mit der Maus markiertes Textsegment in die Zwischenablage zu kopieren,

drücken Sie ⇧+Strg+C. Um Text wieder einzufügen, drücken Sie entsprechend ⇧+Strg+V.

Einige weitere Tastenkürzel sind vom jeweiligen Terminalprogramm abhängig. Deswegen gelten manche der in Tabelle 8.3 zusammengefassten Kürzel nur im Programm `gnome-terminal`.

Tastenkürzel	Funktion
⇧+Strg+C	markierten Text in die Zwischenablage kopieren
⇧+Strg+N	neues Terminalfenster öffnen
⇧+Strg+T	neues Dialogblatt (Tab) öffnen
⇧+Strg+V	Text aus der Zwischenablage einfügen
Strg+Bild↑	in das vorige Dialogblatt wechseln
Strg+Bild↓	in das nächste Dialogblatt wechseln
Strg++ / +-	Schriftgröße ändern

Tabelle 8.3 Tastenkürzel zur Steuerung des Terminalprogramms (zum Teil Gnome-spezifisch)

Maus

Die Maus spielt in Textkonsolen bzw. in Terminalfenstern nur eine untergeordnete Rolle. Sie können sie *nicht* dazu verwenden, um die aktuelle Cursorposition zu verändern! Das gelingt nur mit den Cursortasten. Die Funktion der Maus beschränkt sich darauf, mit der linken Maustaste Text zu kopieren und diesen dann mit der mittleren Maustaste bzw. durch Drücken des Mausrads an der aktuellen Cursorposition wieder einzufügen. Das ist ungemein praktisch und effizient.

Bei einer Maus ohne Mausrad bzw. bei einem Touchpad ohne mittlere Taste müssen Sie wie in anderen Betriebssystemen den Text zum Einfügen zuerst mit einem Tastenkürzel in die Zwischenablage kopieren und mit einem zweiten Kürzel einfügen. Wie umständlich! Jetzt verstehen Sie vielleicht, warum viele Linux-Freaks beim Kauf eines Notebooks darauf achten, dass das Touchpad über eine dritte Taste verfügt.

Die Maus in Textkonsolen

In Textkonsolen, also beim Arbeiten *ohne* grafische Benutzeroberfläche, kann eigentlich keine Maus verwendet werden – es sei denn, Sie installieren und starten das Programm `gpm`. Dann können Sie die Maus auch in Textkonsolen zum Kopieren von Text nutzen.

Farben Die meisten Terminalprogramme sind so vorkonfiguriert, dass der Hintergrund dunkel und die Schrift hell ist. Das können Sie in den Einstellungen Ihres Terminalprogramms natürlich ändern. Dabei sollten Sie aber beachten, dass manche im Terminal ausgeführten Kommandos Farben verwenden und sich darauf verlassen, dass der Hintergrund dunkel ist. Stellen Sie nun einen augenfreundlichen hellen Hintergrund ein, kann es passieren, dass manche Ausgaben schlecht lesbar sind.

Kommandos ausführen

Zum Ausführen von Kommandos geben Sie in der Textkonsole oder im Terminalfenster einfach den Kommandonamen, eventuell einige Parameter und schließlich ↵ ein. Das Kommando `ls` liefert eine Liste der Dateien und Unterverzeichnisse im aktuellen Verzeichnis:

```
user$  ls -l
-rw-------  1 user users   506614 29. Jun 12:11 angebot-katzbauer.pdf
drwxrwxr-x  3 user users     4096 13. Apr 11:31 bak
drwxrwxr-x  2 user users     4096 18. Jul 15:03 bin
-rw-r--r--  1 user users   243571  3. Jul 09:14 DB20078.jpg
drwxr-xr-x  2 user users     4096  7. Apr 10:59 Desktop
-rw-------  1 user users 17708403 19. Mai 10:35 DN.pdf
...
```

Aus dem obigen Beispiel geht hervor, wie in diesem Buch die Kommandoeingabe und das Ergebnis dargestellt werden: `user$` am Beginn der ersten Zeile bedeutet, dass das Kommando von einem gewöhnlichen Benutzer mit dem Namen `user` ausgeführt wurde. Wenn in der ersten Textspalte stattdessen `root#` angegeben ist, wurde das Kommando hingegen von `root` ausgeführt (also vom Systemadministrator). `user$` bzw. `root#` gilt als Eingabeprompt. Diese Zeichen werden am Beginn jeder Eingabezeile automatisch angezeigt. Sie dürfen diese Zeichen *nicht* mit eingeben!

Generell gilt in diesem Buch, dass nur die fett hervorgehobenen Zeichen einzugeben sind! Auf Ihrem Rechner wird statt `user$` bzw. `root#` möglicherweise ein anderer Text angezeigt, der oft das aktuelle Verzeichnis und/oder den Rechnernamen enthält. Auf diese Angaben verzichte ich in diesem Buch aus Gründen der Übersichtlichkeit.

Manchmal reicht der Platz in diesem Buch nicht aus, um ein Kommando in einer einzigen Zeile abzudrucken. In solchen Fällen wird das Kommando über mehrere Zeilen verteilt, die durch das Zeichen \ getrennt sind. Das sieht dann beispielsweise so aus:

```
user$  gconftool-2 --set "/apps/panel/toplevels/top_panel_screen0/monitor" \
                   --type integer "0"
```

Sie können dieses Kommando nun ebenfalls zweizeilig eingeben – dann müssen Sie die erste Zeile wie im Buch mit \ abschließen. Sie können die zwei Zeilen aber auch einfach zusammenziehen: Dann entfällt das Zeichen \.

Kommandos im Hintergrund ausführen

Sie können Kommandos auch im Hintergrund ausführen. Das bedeutet, dass Sie nicht auf das Programmende zu warten brauchen, sondern sofort weiterarbeiten können. Dazu geben Sie am Ende der Kommandozeile das Zeichen `&` an. Diese Vorgehensweise empfiehlt sich vor allem, wenn Sie aus einer Konsole heraus ein Programm mit grafischer Benutzeroberfläche starten (z. B. `firefox &`).

Arbeiten als root

Es ist unter Linux unüblich, als `root` zu arbeiten, also mit Systemadministratorrechten. Wenn Sie als gewöhnlicher Benutzer eingeloggt sind, gibt es verschiedene Wege, einzelne Kommandos mit `root`-Rechten auszuführen. Bei den meisten Distributionen führen Sie im Terminalfenster einfach `sudo -s` aus. Nach der Eingabe Ihres Passworts verfügen Sie über `root`-Rechte. Nun können Sie die gewünschten Kommandos ausführen. `exit` oder Strg+D führt wieder zum ursprünglichen User zurück.

Tipps dazu, wie Sie die Kommandoausführung vom Vordergrund in den Hintergrund verschieben, wie Sie eine Liste aller aktiven Kommandos (Prozesse) ermitteln etc., folgen in Kapitel 12, »Prozessverwaltung«. Details zu `sudo` behandle ich in Abschnitt 12.3, »Prozesse unter einer anderen Identität ausführen (sudo)«.

8.2 Textdateien anzeigen und editieren

less

Natürlich können Sie unter KDE oder Gnome Textdateien in einem Editor öffnen. Wenn Sie hingegen in einer Textkonsole oder in einem Terminalfenster arbeiten, verwenden Sie zum Betrachten von Dateien am besten das Kommando `less`. Sie können das Kommando auch hinter andere Kommandos stellen, um deren oft sehr lange Ausgaben in Ruhe seitenweise zu lesen oder darin nach Text zu suchen (siehe Tabelle 8.4):

```
user$ less datei       (seitenweise Anzeige der Datei)
user$ ls -l | less     (seitenweise Anzeige des Dateiverzeichnisses)
user$ ps ax | less     (seitenweise Anzeige der Prozessliste)
```

Auf manchen Mini-Linux-Systemen, z. B. in Embedded-Geräten wie NAS-Festplatten, fehlt das Kommando `less`. Möglicherweise ist stattdessen der `less`-Vorgänger `more` installiert. Andernfalls können Sie mit `cat` den gesamten Inhalt einer Textdatei ausgeben, also ohne seitenweises Blättern. Wenn Sie nur die letzten Zeilen lesen möchten, z. B. bei einer Logging-Datei, verwenden Sie `tail`.

Das Terminal zeigt nur noch merkwürdige Zeichen ...

Wenn Sie in einer Textkonsole eine Datei anzeigen, die statt Text binäre Daten enthält, kann es passieren, dass die Daten als Sonderzeichen interpretiert werden und die Konsole dabei durcheinanderkommt. In diesem Fall werden nur noch seltsame Zeichen auf dem Bildschirm bzw. im Terminalfenster angezeigt, weil die Zuordnung des Zeichensatzes nicht mehr stimmt. Abhilfe schafft zumeist das Kommando `reset`.

Tastenkürzel	Funktion
Cursortasten	Text nach oben oder unten verschieben
Pos1, Ende	an den Beginn / das Ende des Textes springen
G, ⇧+G	an den Beginn / das Ende des Textes springen
/ muster ↵	vorwärts suchen
? muster ↵	rückwärts suchen
N	Suche vorwärts wiederholen (*next*)
⇧+N	Suche rückwärts wiederholen
Q	beenden (*quit*)
H	Hilfetext mit weiteren Tastenkürzeln anzeigen

Tabelle 8.4 less-Tastenkürzel

Texteditoren

Unter KDE oder Gnome stehen mit `kate`, `gedit` oder `gnome-text-editor` komfortable Texteditoren mit intuitiver Bedienung zur Verfügung. In einer Textkonsole sind diese Programme aber nicht verwendbar – Sie brauchen einen Editor, der im Textmodus läuft. Dieser Abschnitt stellt die populärsten Vertreter dieser Zunft vor. Welcher der Editoren bei Ihnen standardmäßig installiert ist, hängt von Ihrer Distribution ab.

nano

`nano` ist im Befehlsumfang bescheiden, aber dafür einfach zu bedienen. Die beiden unteren Bildschirmzeilen geben eine Übersicht der zur Verfügung stehenden Kommandos (siehe Abbildung 8.2).

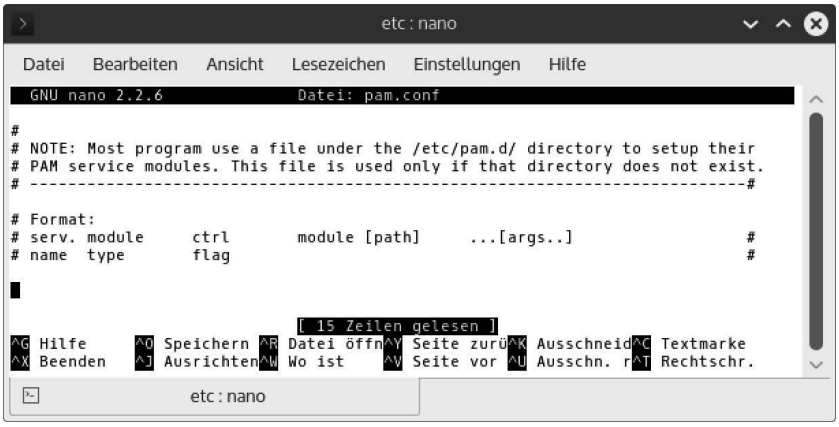

Abbildung 8.2 Der Editor nano in einem Terminalfenster

Emacs, jove, jed, jmacs

Eine Sonderrolle unter den Editoren nimmt der GNU Emacs ein. Das Programm lässt sich aus dem Alltag vieler Administratoren kaum wegdenken. Im Detail stelle ich Ihnen diesen Editor in Kapitel 17 näher vor. Vorweg fasst Tabelle 8.5 die wichtigsten Tastenkürzel zusammen. Sie gelten auch für die Editoren `jove`, `jed` und `jmacs`. Dabei handelt es sich um Minimalversionen des Emacs, die in den Grundfunktionen kompatibel sind.

Tastenkürzel	Funktion
Strg+X, Strg+F	lädt eine neue Datei.
Strg+X, Strg+S	speichert die aktuelle Datei.
Strg+X, Strg+W	speichert die Datei unter einem neuen Namen.
Strg+G	bricht die Eingabe eines Kommandos ab.
Strg+K	löscht eine Zeile.
Strg+X, U	macht das Löschen rückgängig (Undo).
Strg+X, Strg+C	beendet den Emacs (mit Rückfrage zum Speichern).

Tabelle 8.5 Emacs-Tastenkürzel

Vi, Vim und Elvis

Ebenfalls ein Urgestein der Unix-Geschichte ist der Editor Vi, der unter Linux zumeist durch das dazu kompatible Programm Vim vertreten ist, seltener durch den ebenfalls kompatiblen Editor Elvis. Der Original-Vi ist aus urheberrechtlichen Gründen nicht Teil von Linux. Das Kommando `vi` kann aber dennoch ausgeführt werden und führt dann zum Start von Vim oder Elvis.

Der Vi bietet fast genauso viele Funktionen wie der Emacs, die Bedienung ist aber noch schwieriger zu erlernen. Dafür ist der Vi vergleichsweise kompakt und steht zumeist auch auf Notfallsystemen zur Verfügung. Das Programm stellt einen inoffiziellen Unix/Linux-Standard dar und wird von diversen Programmen automatisch als Editor aufgerufen.

Der wichtigste fundamentale Unterschied zu anderen Editoren besteht darin, dass der Vi zwischen verschiedenen Modi unterscheidet. Die Texteingabe ist nur im Insert-Modus möglich (siehe Tabelle 8.6).

Die Eingabe der meisten Kommandos erfolgt im Complex-Command-Modus, der mit : aktiviert wird (siehe Tabelle 8.7). Vorher muss gegebenenfalls der Insert-Modus durch Esc verlassen werden. Die Cursorbewegung ist natürlich auch mit den Cursortasten möglich. Dem Vi in seiner Erscheinungsform Vim ist Kapitel 16 gewidmet.

Tastenkürzel	Funktion
I	wechselt in den Insert-Modus.
Esc	beendet den Insert-Modus.
H / L	bewegt den Cursor nach links/rechts.
J / K	bewegt den Cursor ab/auf.
X	löscht ein Zeichen.
D D	löscht die aktuelle Zeile.
P	fügt die gelöschte Zeile an der Cursorposition wieder ein.
U	macht die letzte Änderung rückgängig (Undo).
:	wechselt in den Complex-Command-Modus.

Tabelle 8.6 Vi-Tastenkürzel

Tastenkürzel	Funktion
:w name	speichert den Text unter einem neuen Namen.
:wq	speichert und beendet den Vi.
:q!	beendet den Vi, ohne zu speichern.
:help	startet die Online-Hilfe.

Tabelle 8.7 Vi-Kommandos im Complex-Command-Modus

joe

`joe` ist ein einfacher Editor. Die Tastenkürzel sind dem aus DOS-Zeiten stammenden Textverarbeitungsprogramm Wordstar nachempfunden. Eine umfassende Beschreibung aller Kommandos erhalten Sie, wenn Sie in einer Konsole `man joe` ausführen. Strg+K, H zeigt einen Hilfetext mit den wichtigsten Tastenkürzeln an.

Das Programm kann auch unter den Namen `jmacs` oder `jpico` gestartet werden. Es gelten dann andere Tastenkürzel, die zum Emacs bzw. zu `nano` kompatibel sind. Mir persönlich hat es vor allem `jmacs` angetan. Das ist ein minimalistischer Emacs-Klon, der für die meisten Zwecke ausreicht. Deswegen ist `joe` oft das erste Paket, das ich auf einem neuen Server installiere.

Standardeditor einstellen

Einige Programme starten zum Ansehen oder Editieren von Dateien selbstständig einen Editor, standardmäßig zumeist den Editor Vi. Wenn Sie einen anderen Editor wünschen, müssen Sie in `/etc/profile` oder `.profile` die Umgebungsvariablen `EDITOR` und `VISUAL` einstellen:

```
# Ergänzung in /etc/profile oder in ~/.profile
export EDITOR=/usr/bin/jmacs
export VISUAL=$EDITOR
```

Ergänzend dazu bieten viele Distributionen die Möglichkeit, mit dem Kommando `alternatives` ein Defaultprogramm bei mehreren von der Funktion her gleichartigen Programmen (Editoren, Java-Interpretern etc.) einzustellen – siehe Abschnitt 20.10, »Verwaltung von Parallelinstallationen (alternatives)«.

8.3 man und info

Kommandos wie `ls`, `cp` oder `top`, die Sie üblicherweise in einem Terminalfenster ausführen, reagieren weder auf F1 noch verfügen sie über ein HILFE-Menü. Es gibt aber natürlich auch für diese Kommandos Hilfetexte, die durch verschiedene Kommandos gelesen werden können:

- `kommando --help` liefert bei sehr vielen Kommandos eine Liste aller Optionen samt einer kurzen Erklärung zu ihrer Bedeutung.
- `man kommando` zeigt bei vielen Kommandos den `man`-Hilfetext an. Durch den meist mehrseitigen Text können Sie mit den Cursortasten blättern. Mit Q beenden Sie die Hilfe.
- `help kommando` funktioniert nur bei sogenannten Shell-Kommandos, z. B. `cd` oder `alias`.
- `info kommando` ist eine Alternative zu `man`. Das `info`-System eignet sich vor allem für sehr umfangreiche Hilfetexte. Ob der Hilfetext im `man`- oder `info`-System vorliegt, hängt ganz einfach davon ab, für welches Hilfesystem sich die Programmentwickler entschieden haben. `man` ist aber deutlich populärer.

man

`man` ist ein Kommando zur Anzeige der Dokumentation vieler elementarer Kommandos wie `ls` oder `cp`. `man` wird in der Form `man kommando` aufgerufen, um den Hilfetext zu `kommando` zu lesen.

Die optionale Angabe eines Bereichs (`man bereich kommando`) schränkt die Suche nach `man`-Texten auf einen Themenbereich ein. Beispielsweise liefert `man 3 printf` die Syntax der C-Funktion *printf*. Diese Einschränkung ist dann notwendig, wenn mehrere gleichnamige `man`-Texte in unterschiedlichen Themenbereichen existieren. `man` zeigt in diesem Fall nur den ersten gefundenen `man`-Text an.

Wenn Sie alle gleichnamigen `man`-Texte aus allen Bereichen lesen möchten, müssen Sie `man` mit der Option `-a` verwenden. Sobald Sie den Text gelesen haben und `man` mit Q beenden, erscheint der `man`-Text zum nächsten Abschnitt.

	Thema		Thema
1	Benutzerkommandos	6	Spiele
2	Systemaufrufe	7	Diverses
3	C-Funktionen	8	Kommandos zur Systemadministration
4	Dateiformate, Device-Dateien	9	Kernelfunktionen
5	Konfigurationsdateien	n	neue Kommandos

Tabelle 8.8 man-Themengruppen

In vielen Unix- und Linux-Büchern werden zusammen mit den Kommandos gleich die man-Nummern angegeben – etwa find(1). Damit wissen Sie sofort, wie Sie man aufrufen müssen. man kennt üblicherweise die Themenbereiche 1 bis 9 und n (siehe Tabelle 8.8). Manchmal werden die Kommandos von Programmiersprachen in zusätzlichen Bereichen mit anderen Buchstaben eingeordnet.

Die Darstellung der Hilfetexte erfolgt intern durch das Programm less. Deswegen gelten für die Navigation im Hilfetext die Tastenkürzel aus Tabelle 8.4. Aus welchen Verzeichnissen man die Hilfetexte liest, können Sie wahlweise durch die Steuerungsdatei /etc/manpath.config oder durch die Umgebungsvariable MANPATH einstellen.

Unter KDE und Gnome können Sie man-Texte auch mit den jeweiligen Help- oder Webbrowsern lesen. Die folgenden Beispiele zeigen, wie Sie die man-Seite zu ls und ein Inhaltsverzeichnis aller man-Seiten anzeigen können:

```
user$  gnome-help man:ls
user$  khelpcenter man:ls
user$  khelpcenter 'man:(index)'
```

help Zu manchen Kommandos erhalten Sie Hilfe nicht mit man, sondern mit help. Das betrifft alle Kommandos, die direkt von der Shell ausgeführt werden. (Die Shell ist der Kommandointerpreter, der Ihre Eingaben entgegennimmt. Ausführliche Informationen zur Linux-Standard-Shell bash folgen im nächsten Kapitel.)

info man-Hilfetexte haben den Nachteil, dass sie nur schwer strukturierbar sind. Der gesamte Text muss in einem einzigen – mitunter schier endlosen – Dokument abgebildet werden. Das alternative info-Format bietet hier deutlich bessere Möglichkeiten, weswegen vor allem umfangreiche Hilfetexte häufig nur in diesem Format vorliegen.

info wird üblicherweise in der Form info kommandoname (z. B. info ls) aufgerufen und über diverse Tastenkürzel gesteuert (siehe Tabelle 8.9). Wird das Kommando ohne Parameter gestartet, zeigt das Programm eine Übersicht der verfügbaren Hilfethemen an.

Tastenkürzel	Funktion
Leertaste	Text nach unten scrollen
←	Text nach oben scrollen
B, E	zum Anfang/Ende der Info-Einheit springen (*beginning/end*)
⇆	Cursor zum nächsten Querverweis bewegen
↵	Querverweis zu anderer Info-Einheit verfolgen
N	nächste Info-Einheit derselben Hierarchiestufe (*next*)
P	vorige Info-Einheit derselben Hierarchiestufe (*previous*)
U	eine Hierarchieebene nach oben (*up*)
L	zurück zum zuletzt angezeigten Text (*last*)
H	ausführliche Bedienungsanleitung (*help*)
?	Kommandoübersicht
Q	beendet `info` (*quit*).

Tabelle 8.9 info-Tastenkürzel

Leider erweist sich der Vorteil der klareren Strukturierung rasch als Nachteil: Die Navigation in `info`-Texten ist unübersichtlich, außerdem fehlt ein Suchmechanismus, der über die gerade aktuelle Seite hinausreicht.

Kapitel 16
Vim

Im Mittelpunkt dieses Kapitels stehen der Editor Vi und dessen Open-Source-Implementierung Vim (*Vi Improved*). Diese Editoren sind – ebenso wie der im nächsten Kapitel vorgestellte Editor Emacs – relativ schwer zu erlernen. Dieser Aufwand lohnt sich, wenn Sie häufig Text, Programmcode, Konfigurationsdateien etc. bearbeiten, wenn ein Texteditor also ein ständiges und unverzichtbares Werkzeug für Sie ist.

Im Vergleich zu dem im vorigen Kapitel vorgestellten Editor VSCode haben Vi und Emacs den Vorteil, dass die Programme bei Bedarf im Textmodus in einer Konsole ausgeführt werden können, also auch in einer SSH-Session. Deswegen ziehen gerade Administratoren Vi oder Emacs modernen Editoren vor, die zwingend eine grafische Benutzeroberfläche erfordern.

Bleibt noch die Königsfrage: Vi oder Emacs? Die Programme sind Urgesteine der Unix/Linux-Geschichte. Beide bieten unzählige Spezialfunktionen, z. B. die automatische Syntaxhervorhebung für zahllose Programmiersprachen und Dokumenttypen oder das Suchen und Ersetzen mit regulären Ausdrücken. Über die Frage, welches Programm nun besser ist, wurden im Internet endlose Diskussionen geführt. Wirklich objektiv kann ich die Frage nicht beantworten: Da ich sämtliche Auflagen dieses Buchs mit Emacs-Varianten verfasst habe (dieses Kapitel natürlich ausgenommen, so viel Vi muss sein!), ist mir der Emacs vertrauter als irgendwelche Vi-Varianten.

Persönlich erscheint mir der Editor Emacs intuitiver zu bedienen und einfacher zu erlernen. Beim Vi treibt einen die Unterscheidung zwischen dem Standard- und dem Einfügemodus anfänglich leicht zum Wahnsinn. Für Vi & Co. spricht andererseits, dass das Programm ein De-facto-Standard unter Unix/Linux ist. Es beansprucht wesentlich weniger Ressourcen und steht selbst auf minimalen Rescue-Systemen zur Verfügung. Echte Unix/Linux-Freaks sollten ohnedies beide Editoren in ihren Grundfunktionen beherrschen – und viel mehr vermittle ich in diesem Buch nicht.

Der ursprüngliche Editor Vi ist ein kommerzielles Programm und steht daher unter Linux nicht zur Verfügung. Vim ist dagegen ein Open-Source-Programm, das zu Vi kompatibel ist und darüber hinaus zahllose Verbesserungen und Erweiterungen bietet. Das Programm kann wahlweise mit den Kommandos `vi` oder `vim` gestartet werden.

Vim im Grafikmodus

Grundsätzlich wird Vim in einer Textkonsole bzw. in einem Konsolenfenster ausgeführt. Wenn Sie ein richtiges Menü und ordentliche Bildlaufleisten bevorzugen, sollten Sie einen Blick auf `gvim` werfen (siehe Abbildung 16.1). Diese grafische Variante zu Vim muss eventuell extra installiert werden, wobei der Paketname oft `vim-gnome` oder `vim-gtk3` lautet.

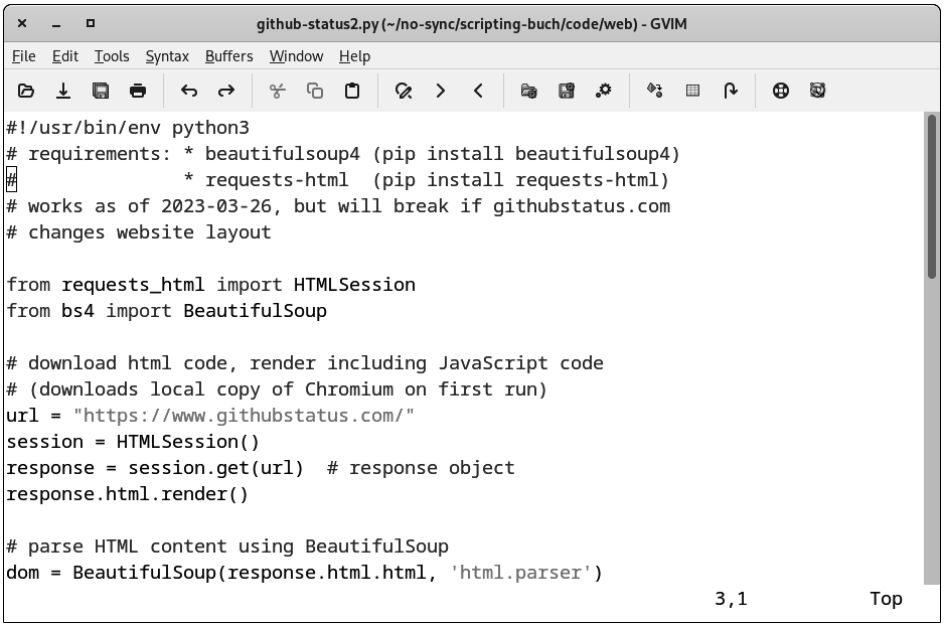

Abbildung 16.1 Die grafische Variante des Editors Vim

Links

Aus Platzgründen kann dieses Kapitel nur eine Einführung zu Vim geben. Für Einsteiger sehr hilfreich ist das Tutorial, das Sie durch das Kommando `vimtutor` starten. (Dadurch wird Vim gestartet und ein deutschsprachiger Hilfetext geladen, der eine Einführung sowie Beispiele zum Ausprobieren enthält.) Die folgenden Links verweisen auf Seiten mit weiterführenden Informationen:

https://www.vim.org	(Homepage)
https://vimhelp.org/vim_faq.txt.html	(FAQ)
https://fprintf.net/vimCheatSheet.html	(Tastenkürzel)
https://truth.sk/vim/vimbook-OPL.pdf	(500-seitiges Vim-Buch)

Vim ist Charityware

Der Hauptentwickler Bram Moolenaar bezeichnet Vim als »Charityware«: Vim ist kostenlos unter einer GPL-kompatiblen Lizenz verfügbar. Wer Vim regelmäßig nutzt, wird aber gebeten, sich in Form einer Spende zu bedanken, die einer Kinderhilfsorganisation in Uganda zugutekommt. Weitere Informationen liefert das Vim-Kommando Esc : `help iccf` ↵.

16.1 Schnelleinstieg

Sie starten Vim üblicherweise in der Form `vim dateiname` innerhalb einer Textkonsole oder in einem Konsolenfenster. Die zu ändernde Datei wird direkt in der Konsole angezeigt.

Standardmodus

Bevor Sie darauflosschreiben können, müssen Sie sich allerdings mit einer Eigenheit auseinandersetzen: Das Programm unterscheidet zwischen unterschiedlichen Bearbeitungsmodi (siehe Tabelle 16.1). Der Standardmodus dient nicht zur Eingabe von Text, sondern zur Ausführung von Kommandos. Wenn Sie im Standardmodus beispielsweise [L] eingeben, bewegen Sie damit den Cursor um ein Zeichen nach rechts. [D], [W] löscht ein Wort, [P] fügt es an der aktuellen Cursorposition wieder ein etc.

Einfügemodus

Um Text einzugeben, müssen Sie mit [I] (*insert*) oder [A] (*append*) in den Einfügemodus wechseln. `vim` zeigt in der untersten Zeile ganz links den Text `-- EINFÜGEN --` an. Im Einfügemodus können Sie Text eingeben, den Cursor bewegen und einzelne Zeichen löschen ([Entf] und [←]). Der Unterschied zwischen [I] und [A] besteht darin, dass die Eingabe bei [I] an der aktuellen Cursorposition beginnt, bei [A] beim Zeichen dahinter.

Bevor Sie wieder ein Kommando eingeben können, müssen Sie mit [Esc] zurück in den Standardmodus wechseln. Dieser Modus wird nicht extra gekennzeichnet. Der linke Teil der letzten Zeile ist jetzt also leer.

Moduswechsel ohne Änderung der Cursorposition

Beim Wechsel vom Einfüge- in den Standardmodus bewegt sich der Cursor um ein Zeichen nach links – es sei denn, er steht bereits am Beginn einer Zeile. Dieses merkwürdige Verhalten ist laut Vim-FAQ beabsichtigt und kann nicht verhindert werden. Um ein einzelnes Kommando auszuführen, ohne den Einfügemodus zu verlassen – und damit auch ohne die aktuelle Cursorposition zu verändern –, leiten Sie die Kommandoeingabe mit [Strg]+[O] ein.

Text löschen

Im Einfügemodus können Sie mit [Entf] und [←] wie üblich einzelne Zeichen löschen. Wenn Sie Wörter, Zeilen oder ganze Bereiche löschen möchten, wechseln Sie zuerst mit [Esc] in den Standardmodus. Anschließend löscht [D], [W] ein Wort und [D], [D] eine ganze Zeile. Wenn Sie eine Zahl voranstellen, wird das Löschkommando entsprechend oft wiederholt. [5], [D], [D] löscht also fünf Zeilen. [.] wiederholt das zuletzt ausgeführte Kommando.

[P] (*put*) fügt den zuletzt gelöschten Text hinter der aktuellen Cursorposition ein, [⇧]+[P] davor. [U] (*undo*) widerruft die letzten Änderungen, [Strg]+[R] (*redo*) stellt die Änderungen wieder her.

Tastenkürzel	Funktion
I	aktiviert den Einfügemodus.
A	aktiviert den Einfügemodus. Die Texteingabe beginnt beim nächsten Zeichen.
Esc	aktiviert den Standardmodus bzw. bricht die Kommandoeingabe ab.
Kommandos im Standardmodus	
D, W	löscht ein Wort.
D, D	löscht die aktuelle Zeile.
n D, D	löscht *n* Zeilen.
P	fügt den zuletzt gelöschten Text hinter der Cursorposition ein.
⇧+P	fügt den zuletzt gelöschten Text vor der Cursorposition ein.
.	wiederholt das letzte Kommando.
U	macht die letzte Änderung rückgängig (Undo).
⇧+U	widerruft alle Änderungen in der aktuellen Zeile.
Strg+R	macht Undo rückgängig (Redo, ab Vim 7).
: w	speichert die Datei.
: q	beendet Vim.
: q!	beendet Vim auch dann, wenn es nicht gespeicherte Dateien gibt.
Kommandos im Einfügemodus	
Strg+O kommando	führt das Kommando aus, ohne den Einfügemodus zu verlassen.

Tabelle 16.1 Elementare Kommandos

Speichern und beenden

Um die geänderte Datei zu speichern, wechseln Sie mit Esc in den Standardmodus und geben dann das Kommando : w ↵ (*write*) ein. : q ↵ (*quit*) beendet den Editor, sofern alle offenen Dateien gespeichert sind. Mit : q! ↵ erzwingen Sie ein Ende selbst dann, wenn es nicht gespeicherte Änderungen gibt. : wq ↵ kombiniert das Speichern und das Programmende.

Hilfe

Vim stellt eine umfassende Online-Hilfe in englischer Sprache zur Verfügung. Zur Startseite des Hilfesystems gelangen Sie von jedem Modus aus mit F1. Alternativ führen im Standardmodus : help bzw. : help thema zur Hilfe. Wenn Sie wissen möchten, welche Hilfethemen es gibt, die das Schlüsselwort *abc* enthalten, geben Sie : help abc Strg+D ein.

Das Hilfefenster

Der Hilfetext wird in einem eigenen Teilbereich von Vim angezeigt (einem sogenannten Fenster, auch wenn es sich dabei nicht um ein eigenständiges Fenster im Sinne des Linux-Grafiksystems handelt). Dieses Fenster schließen Sie mit : q wieder. Sie können das Hilfefenster aber auch geöffnet lassen und im ursprünglichen Text weiterarbeiten. Dazu wechseln Sie mit Strg+W, W das gerade aktive Fenster. Mehr Informationen zum Umgang mit Vim-Fenstern, -Puffern und zur Bearbeitung mehrerer Dateien folgen in Abschnitt 16.5, »Mehrere Dateien gleichzeitig bearbeiten«.

Navigation in der Hilfe

Im Hilfetext sind Verweise auf andere Hilfethemen hervorgehoben (in der von mir getesteten Version hellblau). Um zu diesem Thema zu springen, bewegen Sie den Cursor auf das Schlüsselwort und führen Strg+] aus. Noch einfacher geht es, wenn die Maus aktiviert ist (siehe Abschnitt 16.7, »Tipps und Tricks«): Dann reicht ein Doppelklick auf das Hilfethema, um dorthin zu springen. Strg+T führt zur ursprünglichen Seite zurück.

16.2 Cursorbewegung

Die Cursortasten funktionieren sowohl im Standardmodus als auch im Einfügemodus. Außerdem können Sie die Cursorposition durch diverse Tastenkombinationen im Standardmodus ändern (siehe Tabelle 16.2). Vi-Freaks bewegen sich damit effizienter durch den Text als mit den Cursortasten.

Eine Eigenheit von Vim besteht darin, dass ← am Beginn einer Zeile den Cursor nicht an das Ende der vorherigen Zeile stellt. Analog funktioniert auch → am Ende einer Zeile nicht wie gewohnt. Um das übliche Verhalten anderer Editoren zu erzielen, führen Sie im Standardmodus : set whichwrap=b,s,<,>,[,] aus bzw. fügen dieses set-Kommando in .vimrc ein.

Cursorpositionen speichern

M buchstabe speichert die aktuelle Cursorposition in einem Positionsmarker. Mit ' buchstabe bewegen Sie den Cursor zurück an die so gespeicherte Position.

Vim merkt sich die Cursorposition, an der eine neue Cursorbewegung beginnt. ', ' führt zurück zu dieser Position. Nochmals ', ' bewegt den Cursor wieder an die letzte Position. ',] bzw. ',] bewegen den Cursor an den Beginn bzw. das Ende des zuletzt veränderten Textabschnitts.

Wo bin ich?

Im Einfügemodus zeigt Vim rechts in der Statuszeile die aktuelle Zeilen- und Spaltennummer sowie eine Prozentzahl an, die angibt, in welchem Abschnitt des Texts Sie sich befinden (z. B. 92 %). Mit Strg+G zeigt Vim in der Statuszeile auch den Namen der Datei, ihren Zustand (z. B. *Verändert*), die gesamte Länge in Zeilen und die relative Position im Text in Prozent an.

Tastenkürzel	Funktion
Cursortasten	Die Cursortasten haben die übliche Bedeutung.
H / L / J / K	bewegt den Cursor nach links/rechts/unten/oben.
⇧+H / ⇧+L	bewegt den Cursor an den Beginn bzw. das Ende der aktuellen Seite.
⇧+M	bewegt den Cursor in die Mitte der aktuellen Seite.
B / W	bewegt den Cursor um ein Wort nach links/rechts.
E	bewegt den Cursor an das Ende des Worts.
G, E	bewegt den Cursor an den Anfang des Worts.
(,)	bewegt den Cursor an den Beginn des aktuellen/nächsten Satzes.
{, }	bewegt den Cursor an den Beginn des aktuellen/nächsten Absatzes.
^, $	bewegt den Cursor an den Beginn bzw. das Ende der Zeile.
⇧+G	bewegt den Cursor an das Ende der Datei.
G, G	bewegt den Cursor an den Beginn der Datei.
n ⇧+G	bewegt den Cursor in die Zeile *n*.
n	bewegt den Cursor in die Spalte *n*.
%	bewegt den Cursor zur korrespondierenden Klammer ()[]{}.

Tabelle 16.2 Tastenkürzel zur Cursorbewegung im Standardmodus

16.3 Text bearbeiten

Textzeichen mehrfach einfügen

Um ein Textzeichen mehrfach einzufügen, geben Sie im Standardmodus die Anzahl, das Kommando A (*append*), das gewünschte Zeichen und schließlich Esc ein. Um also 50-mal das Zeichen = einzugeben, geben Sie 5, 0, A, =, Esc ein. Nach dem Kommando befinden Sie sich wieder im Standardmodus.

Tippfehler

Vim hilft auch bei der Korrektur typischer Tippfehler: ~ ändert die Groß- und Kleinschreibung des aktuellen Buchstabens. X, P vertauscht die folgenden zwei Buchstaben.

Text löschen

Tabelle 16.3 gibt einen Überblick über die wichtigsten Kommandos zum Löschen von Text. Wenn Sie vor dem Löschkommando eine Zahl eingeben, wird das Löschkommando entsprechend oft wiederholt. Wie für alle anderen Vim-Kommandos gilt: . wiederholt das letzte Kommando, *n* . wiederholt es *n*-mal.

Tastenkürzel	Funktion im Einfügemodus
Entf, ←	Diese Tasten haben die übliche Bedeutung.
	Funktion im Standardmodus
X	löscht das Zeichen an der Cursorposition bzw. den markierten Text.
⇧+X	löscht das Zeichen vor dem Cursor.
D, D	löscht die aktuelle Zeile.
D cursorkommando	löscht den Text entsprechend dem Kommando zur Cursorbewegung (siehe Tabelle 16.2). Beispiele: D, $ löscht bis zum Ende der Zeile. D, B löscht das vorige Wort. D, W löscht das nächste Wort.

Tabelle 16.3 Text löschen

Text wird grundsätzlich in ein Kopierregister gelöscht. Der zuletzt gelöschte Text kann von dort mit ⇧+P an der aktuellen Cursorposition bzw. mit P hinter der Cursorposition wieder in den Text eingefügt werden.

Eine eigentümliche Art, Text zu löschen und dann durch neuen Text zu ersetzen, bieten die C-Kommandos (*change*): Beispielsweise löscht C, W das aktuelle Wort und aktiviert den Einfügemodus. Sie geben nun das neue Wort ein und schließen die Eingabe mit Esc ab. Analog funktioniert C auch für andere Cursorkommandos.

Text kopieren

Sie können Text auch in das Kopierregister einfügen, ohne ihn zu löschen. Tabelle 16.4 fasst die entsprechenden Kommandos zusammen. Alle gelten für den Standardmodus.

Text markieren

Einige (Lösch-)Kommandos setzen voraus, dass Sie zuerst einen Textausschnitt markieren. Vim sieht dazu drei verschiedene Markierungsmodi vor, die Sie mit V, ⇧+V bzw. Strg+V am Startpunkt der Markierung aktivieren bzw. ebenso wieder deaktivieren. Während einer dieser Modi aktiv ist, enthält die unterste Vim-Zeile

den Text `-- VISUELL --`. Sie bewegen den Cursor nun zum Endpunkt der Markierung oder erweitern die Markierung durch einige spezielle Markierungskommandos (siehe Tabelle 16.5). Solange der Markierungsmodus aktiv ist, stehen Ihnen diverse Kommandos zur Bearbeitung des markierten Texts zur Auswahl (siehe Tabelle 16.6).

Tastenkürzel	Funktion
Y	kopiert den markierten Text in das Kopierregister.
Y, Y	kopiert die aktuelle Zeile in das Kopierregister.
Y cursorkommando	kopiert den durch die Cursorbewegung erfassten Text. Beispiel: Y, } kopiert den Text bis zum Ende des Absatzes.

Tabelle 16.4 Text in das Kopierregister kopieren

Tastenkürzel	Funktion
V	(de)aktiviert den Zeichenmarkierungsmodus.
⇧+V	(de)aktiviert den Zeilenmarkierungsmodus.
Strg+V	(de)aktiviert den Blockmarkierungsmodus.
A, W	vergrößert die Markierung um ein Wort.
A, S	vergrößert die Markierung um einen Satz.
A, P	vergrößert die Markierung um einen Absatz.
A, B	vergrößert die Markierung um eine ()-Ebene.
A, ⇧+B	vergrößert die Markierung um eine {}-Ebene.
G, V	markiert den zuletzt markierten Text nochmals.
O	wechselt die Cursorposition zwischen Markierungsanfang und -ende.

Tabelle 16.5 Text markieren

Tastenkürzel	Funktion
X	löscht den markierten Text.
Y	kopiert den markierten Text in das Kopierregister.
~	ändert die Groß-/Kleinschreibung.
J	fügt die markierten Zeilen zu einer langen Zeile zusammen.

Tabelle 16.6 Markierten Text bearbeiten

Tastenkürzel	Funktion
G, Q	führt einen Zeilenumbruch durch (für Fließtext).
>, <	rückt den Text um eine Tabulatorposition ein oder aus.
=	rückt den Text dem aktuellen `indent`-Modus entsprechend neu ein.
`!sort`	sortiert die Zeilen mit dem externen Kommando `sort`.

Tabelle 16.6 Markierten Text bearbeiten (Forts.)

Zeilen einrücken

Gerade beim Editieren von Code ist das richtige Einrücken von Zeilen wichtig. Vim hilft dabei auf vielfältige Weise. Die elementarsten Kommandos sind > , > bzw. < , < . Sie rücken die aktuelle Zeile um eine Tabulatorposition ein oder aus. Wenn Sie vorher mehrere Zeilen Text markieren, können Sie die Kommandos auf einen ganzen Block anwenden. Dabei reicht die einfache Eingabe von > bzw. < . : `set shiftwidth=N` verändert die Einrücktiefe (normalerweise 8 Zeichen).

Vim kann auch versuchen, neue Zeilen schon während der Eingabe automatisch einzurücken. Dazu aktivieren Sie einen Einrückmodus, beispielsweise durch : `set cindent`. Im Folgenden sind die Grundfunktionen der wichtigsten Vim-Einrückmodi kurz zusammengefasst:

- **autoindent:** Rückt die folgende Zeile genauso weit ein wie die vorherige.
- **smartindent:** Funktioniert wie `autoindent`, berücksichtigt aber zusätzlich {}-Klammerebenen. Damit Vim die schließenden Klammern richtig erkennt, sollten diese am Beginn einer neuen Zeile angegeben werden. Das Ausmaß der Einrückung je nach Klammernebene steuern Sie durch die Option `shiftwidth`. Zuvor markierter Text kann durch = neu eingerückt werden.
- **cindent:** Funktioniert wie `smartindent`, berücksichtigt aber auch diverse Codestrukturen von C bzw. C++. Der Einrückmechanismus kann durch verschiedene Optionen den persönlichen Vorlieben angepasst werden (siehe dazu : `help C-indenting`).

Fließtext

Vim ist so vorkonfiguriert, dass das Verfassen von Code bzw. das Ändern von Konfigurationsdateien möglichst gut funktioniert. Aus diesem Grund führt Vim keinen automatischen Zeilenumbruch durch (d. h., Sie müssen neue Zeilen selbst mit ↵ beginnen). Sie können Vim aber selbstverständlich auch zum Verfassen gewöhnlichen Texts einsetzen (etwa für E-Mails). Tabelle 16.7 fasst einige spezielle Kommandos und Optionen zusammen, die dabei helfen.

Tastenkürzel	Funktion
⇧+J	verbindet die aktuelle Zeile mit der folgenden.
n ⇧+J	verbindet *n* Zeilen zu einer langen Zeile.
G, Q, A, P	umbricht den aktuellen Absatz neu und stellt den Cursor an den Beginn des nächsten Absatzes.
G, W, A, P	wie oben, aber belässt den Cursor am aktuellen Ort.
: set textwidth=nn	automatischer Zeilenumbruch nach maximal *nn* Zeichen (normalerweise: 0 = deaktiviert)

Tabelle 16.7 Fließtext bearbeiten

Die G-Kommandos berücksichtigen den `autoindent`-Modus sowie die Einstellung von `textwidth`. Wenn `textwidth` 0 enthält, beträgt die maximale Zeilenlänge 79 Zeichen. Konfigurationsmöglichkeiten für eine besonders komfortable Fließtexteingabe bietet die Option `formatoptions` (siehe den dazugehörigen Hilfetext).

Wortergänzungen

Das Eintippen langer Wörter und von Funktions- und Variablennamen ist mühsam und fehleranfällig. Vim hilft Ihnen dabei auf geniale Weise: Sie geben lediglich die ersten Buchstaben ein und drücken dann Strg+P. Wenn das Wort bereits eindeutig bestimmt ist, wird es sofort vervollständigt. Andernfalls können Sie mit Strg+P bzw. den Cursortasten das gewünschte Wort auswählen. Vim berücksichtigt bei der Wortergänzung alle Wörter aller geladenen Dateien, wobei Wörter aus der aktuellen Datei und dabei wiederum Wörter in der Nähe der Cursorposition bevorzugt werden.

16.4 Suchen und Ersetzen

Text suchen

Im Standardmodus bewegt / `suchtext` ↵ den Cursor zum gesuchten Text. N wiederholt die Suche, ⇧+N wiederholt die Suche rückwärts. Um von vornherein rückwärts zu suchen, beginnen Sie die Suche mit ? `suchausdruck`. Tabelle 16.8 erläutert die wichtigsten Sonderzeichen, um nach Mustern zu suchen.

Groß- und Kleinschreibung

Vim unterscheidet bei der Suche standardmäßig zwischen Groß- und Kleinschreibung. Wenn Sie das nicht möchten, leiten Sie das Suchmuster mit `/c` ein (gilt nur für diese Suche) oder führen : `set ignorecase` aus (gilt für alle weiteren Suchen).

Inkrementelle Suche

Mit : `set incsearch` aktivieren Sie die sogenannte inkrementelle Suche: Bereits während der Eingabe des Suchtexts durch / `suchausdruck` bewegt Vim den Cursor zum ersten passenden Ort. ↵ beendet die Suche, Esc bricht sie ab. Nach der Suche bleiben alle Übereinstimmungen im Text markiert, bis Sie eine neue Suche durchführen oder : `nohlsearch` ausführen.

Zeichen	Bedeutung
`.`	ein beliebiges Zeichen
`^ $`	Zeilenanfang/Zeilenende
`\< \>`	Wortanfang/Wortende
`[a-e]`	ein Zeichen zwischen *a* und *e*
`\s`, `\t`	ein Leerzeichen bzw. ein Tabulatorzeichen
`\( \)`	fasst ein Suchmuster als Gruppe zusammen.
`\=`	Der Suchausdruck muss 0- oder einmal auftreten.
`*`	Der Suchausdruck darf beliebig oft (auch 0-mal) auftreten.
`\+`	Der Suchausdruck muss mindestens einmal auftreten.

Tabelle 16.8 Sonderzeichen im Suchausdruck

Suchen und Ersetzen

Um alle Vorkommen des Texts `abc` ohne Rückfrage durch `efg` zu ersetzen, führen Sie im Standardmodus `:` `%s/abc/efg/g` aus. `'.`, `'.` führt anschließend zurück an den Beginn der Suche. Tabelle 16.9 stellt einige Varianten des Suchen-und-Ersetzen-Kommandos vor. Beim Suchen und Ersetzen mit Rückfrage können Sie mit `Y` oder `N` für jeden gefundenen Suchausdruck angeben, ob dieser durch den neuen Text ersetzt werden soll oder nicht. `Q` bricht den Vorgang ab, `A` ersetzt alle weiteren Vorkommen. Im Ersetzen-Ausdruck können Sie sich mit *\n* auf die *n*-te Gruppe im Suchmuster beziehen. Eine Menge weiterer Tipps zum Suchen und Ersetzen sowie zahlreiche Beispiele finden Sie in der Online-Hilfe (`:` `help substitute`).

Tastenkürzel	Funktion
`:` `%s/abc/efg/g`	ersetzt ohne Rückfrage alle Vorkommen von *abc* durch *efg*.
`:` `%s/abc/efg/gc`	ersetzt mit Rückfrage alle Vorkommen von *abc* durch *efg*.
`:` `%s/abc/efg/gci`	ersetzt ohne Berücksichtigung der Groß- und Kleinschreibung.

Tabelle 16.9 Suchen und Ersetzen

16.5 Mehrere Dateien gleichzeitig bearbeiten

Im Standardmodus lädt `:` `e dateiname` eine neue Datei. Die neue Datei ersetzt die momentan bearbeitete Datei, die Sie vorher speichern müssen – andernfalls bricht

Vim den Vorgang ab. Sie können das Laden mit `:e! dateiname` zwar erzwingen, verlieren dann aber alle durchgeführten Änderungen an der zuletzt aktuellen Datei.

Puffer und Fenster

Selbstverständlich können Sie in Vim auch mehrere Dateien gleichzeitig bearbeiten. Vorher sollten Sie allerdings das nicht besonders intuitive Konzept verstehen, wie Vim intern Texte verwaltet und anzeigt. Jeder im Editor dargestellte Text befindet sich intern in einem sogenannten Puffer.

Das gilt sowohl für Dateien als auch für Hilfetexte. Solange es nur einen Puffer gibt, wird dieser auf der gesamten Vim-Arbeitsfläche angezeigt. Um mehrere Puffer gleichzeitig anzuzeigen, wird die Arbeitsfläche in mehrere sogenannte Fenster aufgeteilt, wie dies auch bei der Anzeige von Hilfetexten der Fall ist. Ein derartiges Fenster ist kein eigenständiges Fenster im Sinne des Linux-Grafiksystems, sondern nur ein Teilbereich der Arbeitsfläche.

Die Puffer veränderter Dateien sind immer in einem Fenster sichtbar. Puffer von Dateien, die seit dem letzten Speichern nicht mehr geändert wurden, können dagegen ausgeblendet werden. Die Puffer bleiben dabei im Speicher, gelten nun aber als nicht mehr aktiv. (Vorsicht: Wenn Sie ein Fenster mit einer noch nicht gespeicherten Datei schließen, gehen alle Änderungen verloren! Der inaktive Puffer der Datei bleibt zwar verfügbar, enthält aber die Datei zum Zeitpunkt der letzten Speicherung.)

Vim ist auch in der Lage, eine Datei gleichzeitig in mehreren Fenstern darzustellen, z. B. um unterschiedliche Teile eines sehr langen Texts zu bearbeiten.

Tabbed-Fenster

Vim erleichtert die Bearbeitung mehrerer Dateien mit Dialogblättern im Textmodus (siehe Abbildung 16.2). Wirklich komfortabel funktioniert das, wenn Sie die Maus aktiviert haben (siehe Abschnitt 16.7, »Tipps und Tricks«): Dann können Sie bequem mit der Maus die gerade aktive Datei auswählen bzw. mit dem X-Button rechts oben schließen. Weitere Informationen zu diesen *Tabbed Pages* erhalten Sie mit `:help tabpage`.

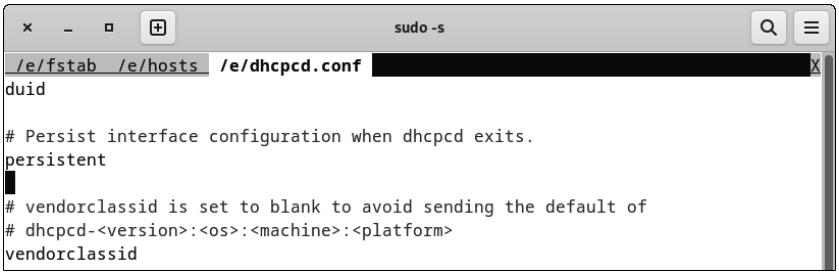

Abbildung 16.2 Drei Dateien in drei Tabbed-Fenstern

Eine neue Datei laden

Je nachdem, ob Sie mit Fenstern oder Tabbed-Fenstern arbeiten möchten, laden Sie neue Dateien mit `:new dateiname` oder `:tabnew dateiname`. Wenn Sie beim Programmstart mehrere Dateien übergeben, also beispielsweise `vim datei1 datei2`

`datei3`, wird lediglich die erste Datei angezeigt; die anderen werden in unsichtbare Puffer geladen. Um jede Datei in einem Fenster bzw. in einem Tabbed-Fenster zu öffnen, müssen Sie zusätzlich die Option `-o` bzw. `-p` übergeben.

Tabelle 16.10 fasst die wichtigsten Kommandos zusammen, um Dateien zu laden und zu speichern, zwischen (Tabbed-)Fenstern zu wechseln etc.

Tastenkürzel	Funktion
`:e dateiname`	lädt eine Datei in den aktuellen Puffer.
`:w`	speichert die aktuelle Datei.
`:wall`	speichert alle offenen Dateien.
`:wq`	speichert und schließt den Puffer.
`:q`	schließt den aktuellen Puffer und beendet Vim, wenn keine weiteren Puffer mehr offen sind.
`:q!`	schließt den Puffer auch mit ungesicherten Änderungen.
`:qall`	schließt alle Puffer und beendet Vim.
`:split`	teilt das Fenster und zeigt in beiden Teilen denselben Text an.
`:new`	erzeugt einen leeren Puffer und zeigt ihn in einem Fenster an.
`:new datei`	lädt eine Datei in einen neuen Puffer.
`:only`	maximiert das aktuelle Fenster und schließt die anderen Puffer.
`:all`	zeigt alle Puffer in entsprechend verkleinerten Fenstern an.
`:buffers`	liefert die Liste aller Puffer.
`:buffer n`	zeigt den Puffer *n* an und löscht den aktuellen Puffer.
`:buffer datei`	zeigt den Puffer mit der Datei im aktuellen Fenster an.
`:tabnew`	erzeugt einen Puffer und zeigt ihn in einem Tabbed-Fenster an.
`:tabnew datei`	lädt eine Datei und zeigt sie in einem Tabbed-Fenster an.
`:tabnext`	wechselt in das nächste Tabbed-Fenster.
`:tabprevious`	wechselt in das vorige Tabbed-Fenster.
`:tabclose`	schließt das aktuelle Tabbed-Fenster.
`:tabonly`	schließt alle anderen Tabbed-Fenster.

Tabelle 16.10 Dateien, Puffer und Fenster

16.6 Interna

Optionen

Auf den vorigen Seiten wurden immer wieder besondere Funktionen von Vim durch Optionen aktiviert oder verändert, und dieser Abschnitt stellt eine Menge weiterer Optionen vor.

Beachten Sie, dass manche Optionen nur lokal für die aktuelle Datei bzw. den aktuellen Puffer gelten. `:set` verändert in diesem Fall nur die lokale Einstellung. Um die Option global zu verändern, verwenden Sie `:setglobal`. Tabelle 16.11 fasst die wichtigsten Kommandos zur Bearbeitung von Optionen zusammen.

Tastenkürzel	Funktion
`:help options`	liefert allgemeine Informationen sowie eine Optionsreferenz.
`:help 'option'`	liefert Informationen zur angegebenen Option.
`:set`	liefert alle Optionen, die nicht im Grundzustand sind.
`:set <booleschеoption>`	aktiviert die boolesche Option.
`:set no<booleschеoption>`	deaktiviert die boolesche Option.
`:set inv<booleschеoption>`	invertiert den aktuellen Zustand der Option.
`:set option?`	zeigt die Einstellung der Option an.
`:set option=wert`	stellt die Option neu ein.
`:set option+=wert`	verändert die Option.
`:set option-=wert`	verändert die Option.
`:set option&`	setzt die Option auf den Grundzustand zurück.

Tabelle 16.11 Umgang mit Optionen

Konfiguration (vimrc)

Sämtliche Optionseinstellungen gehen beim Beenden von Vim verloren. Um Optionen bleibend einzustellen, verändern Sie die Vim-Konfigurationsdatei `.vimrc`. Beachten Sie, dass Kommentare durch das Zeichen " eingeleitet werden!

Die folgenden Zeilen geben ein einfaches Beispiel, wie `.vimrc` aussehen kann. Alle dort verwendeten Optionen wurden bzw. werden in diesem Kapitel vorgestellt. Wenn Sie im Internet nach `vimrc` suchen, finden Sie zahllose weitere Beispiele.

```
" Beispiel für ~/.vimrc
" Cursorposition mit der Maus festlegen
set mouse=a
```

```
" <Cursor links> am Zeilenanfang bewegt den Cursor an das Ende der vorigen Zeile,
" <Cursor rechts> am Zeilenende bewegt den Cursor an den Beginn der nächsten
" Zeile
set whichwrap=b,s,<,>,[,]
" Backups (name~) beim Speichern erzeugen
set backup
" inkrementelle Suche aktivieren
set incsearch
" generell Leerzeichen statt Tabs einfügen
set expandtab
```

Die Konfigurierbarkeit von Vim geht aber noch viel weiter: Sie können Vim-Optionen für unterschiedliche Dateitypen unterschiedlich einstellen, neue Funktionen selbst programmieren etc. Erfinden Sie aber nicht das Rad neu! Unter der folgenden Adresse finden Sie zahllose fertige Lösungen, die Sie mit wenig Aufwand nutzen können. In der Regel reicht es aus, die betreffende Datei in `.vimrc` durch `source dateiname` einzubinden.

https://www.vim.org/scripts

Unabhängig von der Backup-Einstellung aktualisiert Vim während des Schreibens regelmäßig eine sogenannte Swap-Datei. Der Name dieser Datei ergibt sich aus einem vorangestellten Punkt, dem aktuellen Dateinamen sowie der Endung `.swp` (also beispielsweise `.mycode.c.swp`, wenn Sie gerade `mycode.c` bearbeiten). Diese Datei enthält in einem Binärformat alle Änderungen, die Sie seit dem letzten Speichern durchgeführt haben. Sie wird automatisch aktualisiert, wenn Sie mehr als 200 Zeichen neuen Text eingegeben haben oder vier Sekunden lang keine Eingaben durchgeführt haben. Die Swap-Datei wird beim regulären Beenden von Vim gelöscht.

Sollte der Strom ausfallen, Linux oder Vim abstürzen etc., können Sie Ihre ungesicherte Arbeit beim nächsten Vim-Start wiederherstellen. Vim bemerkt beim Öffnen der Datei, dass eine Swap-Datei existiert, und stellt verschiedene Optionen zur Auswahl. Im Regelfall werden Sie sich für [W] (Wiederherstellen) entscheiden und die so gerettete Datei anschließend speichern. Danach sollten Sie Vim verlassen und die Swap-Datei explizit löschen. (Das erfolgt nicht automatisch!)

Zeichensatz

Vim kommt standardmäßig mit den meisten wichtigen Zeichensätzen zurecht, unter anderem mit diversen Latin-Varianten, Unicode (`utf-8`, `utf-16`, `ucs-2`, `ucs-2l`, `ucs-4` etc.) sowie einigen asiatischen 2-Byte-Zeichensätzen (z. B. `euc-kr`, also Koreanisch).

Vim ermittelt beim Start den Standardzeichensatz des Betriebssystems (Option `encoding`). Beim Lesen einer neuen Datei versucht Vim, auch deren Zeichensatz zu erkennen (Option `fileencoding`). Dabei werden der Reihe nach alle in der Option `fileencodings` aufgezählten Zeichensätze ausprobiert, bis ein Zeichensatz zur fehlerfreien Darstellung des gesamten Texts gefunden wird. (Die Grundeinstellung für `fileencodings` lautet oft `utf-8,latin1`.)

Um herauszufinden, welchen Zeichensatz die gerade bearbeitete Datei nutzt, führen Sie : `set fileencoding?` aus. Mit : `set fileencoding=<neuerZeichensatz>` verändern Sie den Zeichensatz. Wenn Sie die Datei nun speichern, wird sie im neuen Zeichensatz gespeichert!

16.7 Tipps und Tricks

:-Kommandos effizient eingeben

Bei der Eingabe von Kommandos, die mit : beginnen, gibt es einige Eingabehilfen: Mit den Cursortasten können Sie durch die zuletzt benutzten Kommandos blättern. (Vim speichert die Kommandos in `.viminfo` und merkt sich die Kommandos somit auch nach dem Programmende.) Weiters können Sie mit ⇆ Schlüsselwörter vervollständigen (z. B. bei der Eingabe von Optionen). Und zu guter Letzt können Sie viele :-Kommandos abkürzen (z. B. : `tabn` statt : `tabnext`).

Zeilennummern anzeigen

: `set number` zeigt neben jeder Zeile die Zeilennummer an. : `set nonumber` deaktiviert diesen Modus wieder.

Backups

Standardmäßig erstellt Vim beim Speichern kein Backup (also keine Kopie der ursprünglichen Datei). Wenn Sie das wünschen, führen Sie im Standardmodus : `set backup` aus. Die Backup-Datei erhält den Namen `alternameˆ`. Um Backups generell zu aktivieren, fügen Sie `set backup` in `.vimrc` ein.

Maus aktivieren

Wenn Sie Vim in einer Textkonsole oder in einem Konsolenfenster verwenden, dann ist die Funktion der Maus auf ihre Grundfunktionen unter X beschränkt: Sie können damit zwar Text kopieren und an der aktuellen Cursorposition einfügen, Sie können aber nicht die aktuelle Cursorposition verändern etc.

Um der Maus in Vim mehr Funktionen zu geben, verwenden Sie entweder die grafische Variante `gvim`, oder Sie führen im Standardmodus : `set mouse=a` aus. Sie können nun den Cursor durch einen Mausklick neu positionieren, das aktive Vim-Fenster auswählen, mit dem Mausrad durch den Text scrollen etc.

Dieser Mausmodus hat allerdings einen Nachteil: Die mittlere Maustaste fügt nun den zuletzt in Vim gelöschten Text ein. Die Maus kann nicht mehr zum Kopieren von Text zwischen Vim und anderen Programmen genutzt werden. Abhilfe ist aber einfach: Die herkömmlichen Mausfunktionen sind weiterhin verfügbar, wenn Sie zusätzlich die ⇧-Taste drücken. Achten Sie aber darauf, dass sich Vim beim Einfügen von Text tatsächlich im Einfügemodus befindet! Andernfalls wird der per Maus eingefügte Text als Kommando interpretiert, und das kann schiefgehen.

Leerzeichen statt Tabulatoren

Damit Vim in Ihren Text grundsätzlich Leerzeichen statt Tabulatorzeichen einfügt, führen Sie : `set expandtab` aus bzw. fügen die Anweisung in `.vimrc` ein. Um in der vorhandenen Datei alle Tabulatorzeichen durch die entsprechende Anzahl von

Leerzeichen zu ersetzen, führen Sie anschließend [:] `retab` aus. Um umgekehrt Leerzeichen durch Tabulatoren zu ersetzen, führen Sie [:] `set unexpandtab` und dann [:] `retab!` aus.

Makros

[.] wiederholt das letzte Kommando – so viel wissen Sie schon. Wenn Sie aber eine ganze Abfolge von Kommandos mehrfach ausführen möchten, definieren Sie ein Makro. Dazu starten Sie im Standardmodus mit [Q] den Makromodus. Das nächste Zeichen gibt den Namen des Makros an (genau genommen den Namen des Registers, in dem das Makro gespeichert wird). Alle weiteren Kommandos werden im Makro gespeichert, bis Sie die Eingabe abermals durch [Q] beenden. Das so aufgezeichnete Makro können Sie nun mit [@] `makroname` ausführen. (Wenn Sie Vim verlassen, gehen alle gespeicherten Makros verloren.)

Ein Beispiel: Die folgende Tastensequenz zeichnet das Makro *a* auf, das am Beginn und am Ende eines Worts das Anführungszeichen " einfügt:

[Q], [A], [I], ["], [Esc], [E], [A], ["], [Esc], [Q]

Wenn der Cursor nun innerhalb eines Worts steht und Sie [@], [A] ausführen, wird dieses Wort in Anführungszeichen gestellt. [@], [@] wiederholt das letzte Registerkommando, ohne dass Sie sich an den Makro- bzw. Registernamen erinnern müssen.

Linux-Kommandos ausführen

Wenn Sie in Vim ein Linux-Kommando ausführen möchten, ohne Vim zu verlassen, führen Sie im Standardmodus [:] `!kommandoname` aus (also beispielsweise `!ls`, um die Liste der Dateien im aktuellen Verzeichnis zu ermitteln). Vim zeigt das Ergebnis des Kommandos an. Mit [↵] gelangen Sie zurück in den Editor. Zur Ausführung mehrerer Kommandos öffnen Sie mit [:] `sh` eine neue Shell. Von dort gelangen Sie mit [Strg]+[D] zurück in den Editor.

Vim im Easy-Modus verwenden

Wenn Sie sich nicht an die verschiedenen Vim-Modi gewöhnen können, auf Vim aber nicht mehr verzichten möchten, starten Sie das Programm am besten mit `vim -y` bzw. führen [:] `set insertmode` aus. Damit verbleibt der Editor immer im Einfügemodus. Sie müssen nun jedes Kommando mit [Strg]+[O] einleiten. Um mehrere Kommandos auf einmal auszuführen, ist es auch möglich, mit [Strg]+[L] für längere Zeit in den Standardmodus zu wechseln und diesen mit [Esc] zu verlassen.

Noch ähnlicher zu anderen Editoren verhält sich Vim, wenn Sie `evim` starten: Damit wird der Editor in der grafischen Benutzeroberfläche `gvim` gestartet. Textmarkierungen können mit [⇧] und den Cursortasten durchgeführt werden. Texte werden mit [Strg]+[C] kopiert, mit [Strg]+[X] gelöscht und mit [Strg]+[V] wieder eingefügt. `man evim` bezeichnet den Easy-Modus als »Vim for gumbies« (was auch immer ein *gumby* ist ...): Sie verlieren damit so viel von den Grundeigenschaften von Vim, dass es besser ist, gleich einen anderen Editor einzusetzen.

Kapitel 25
Kernel und Module

Dieses Kapitel beschäftigt sich mit dem Linux-Kernel und seinen Modulen. Module sind Teile des Kernels, die bei Bedarf geladen werden – etwa wenn eine bestimmte Hardware-Komponente zum ersten Mal angesprochen wird. Vorweg ein Überblick:

- **Kernelmodule:** Abschnitt 25.1 erklärt, warum Kernelmodule automatisch geladen werden und was Sie tun müssen, wenn dieser Automatismus versagt.
- **Device Trees:** Auf Smartphones und in Embedded Devices gelten für die Verwaltung der Kernelmodule ganz andere Voraussetzungen als auf PCs. Dort haben sich Device Trees zur Verwaltung von Kernelmodulen durchgesetzt. Eine Einführung in diese auch auf dem Raspberry Pi übliche Technik gibt Abschnitt 25.2.
- **Kernelmodule selbst kompilieren:** Wenn Sie spezielle Hardware einsetzen, müssen Sie eventuell ein eigenes Kernelmodul kompilieren. Wie das gelingt, verrät Abschnitt 25.3.
- **Den ganzen Kernel kompilieren:** Auch wenn eher selten die Notwendigkeit besteht, den ganzen Kernel neu zu kompilieren, beweist Abschnitt 25.4, dass dies durchaus keine Hexerei ist.
- **Kernel-Update ohne Reboot:** `kexec` ermöglicht es, einen anderen Kernel zu aktivieren, ohne den Rechner neu zu starten (siehe Abschnitt 25.5).
- **Kernel-Live-Patches:** Noch eleganter ist es, Kernel-Updates im laufenden Betrieb durchzuführen. Abschnitt 25.6 stellt Mechanismen zur Aktivierung derartiger Live-Patches vor.
- **/proc- und /sys-Dateisystem:** Abschnitt 25.7 zeigt, wie Sie aus dem `/proc`- bzw. `/sys`-Dateisystem aktuelle Informationen über den Kernel ermitteln.
- **Kerneloptionen und -parameter:** Abschnitt 25.8 und Abschnitt 25.9 erklären, wie Sie während des Rechnerstarts Optionen an den Kernel übergeben bzw. im laufenden Betrieb Kernelparameter ändern.
- **Spectre, Meltdown & Co.:** Abschnitt 25.10 beschreibt schließlich, mit welchen Maßnahmen der Kernel Sie vor Sicherheitslücken in aktuellen CPUs schützt.

Dieses Kapitel richtet sich explizit an fortgeschrittene Linux-Anwender. Einsteiger sind gut beraten, nur den für ihre Distribution vorgesehenen Kernel zu verwenden! Alle Informationen in diesem Kapitel gelten für die Kernelversionen 4.*n*, 5.*n* und 6.*n*.

25.1 Kernelmodule

Der Kernel ist jener Teil von Linux, der für elementare Funktionen wie Speicherverwaltung, Prozessverwaltung, Zugriff auf SSDs und Netzwerkkarten etc. zuständig ist. Der Kernel verfolgt dabei ein modularisiertes Konzept: Anfänglich – also beim Hochfahren des Rechners – wird ein Basiskernel geladen, der nur jene Funktionen enthält, die zum Rechnerstart erforderlich sind.

Wenn im laufenden Betrieb Zusatzfunktionen benötigt werden, z. B. für spezielle Hardware, wird der erforderliche Code als Modul mit dem Kernel verbunden. Werden diese Zusatzfunktionen eine Weile nicht mehr benötigt, kann das Modul wieder aus dem Kernel entfernt werden. Dieses modularisierte Konzept hat viele Vorteile:

- Kernelmodule können nach Bedarf eingebunden werden. Wenn ein bestimmtes Modul auf Ihrem Rechner nicht benötigt wird, kann so Speicher gespart werden, d. h., der Kernel ist nicht größer als unbedingt notwendig und optimal an Ihre Hardware angepasst.
- Bei einer Änderung der Hardware (z. B. nach dem Einbau einer neuen Netzwerkkarte) muss kein neuer Kernel kompiliert, sondern nur das entsprechende Modul geladen werden.
- Bei der Entwicklung eines Kernelmoduls muss nicht ständig der Rechner neu gestartet werden. Es reicht, ein Modul neu zu kompilieren. Anschließend kann es bei laufendem Betrieb getestet werden.

Eine Menge Hintergrundinformationen zum Umgang mit Kernelmodulen finden Sie im Module-HOWTO-Dokument. Es ist zwar 15 Jahre alt, an den Prinzipien der Modulverwaltung hat sich seither aber wenig geändert.

https://www.tldp.org/HOWTO/Module-HOWTO

Module automatisch laden

Dafür, dass Kernelmodule tatsächlich automatisch geladen werden, sobald sie benötigt werden, ist die in den Kernel integrierte Komponente `kmod` verantwortlich. `kmod` wird durch die Dateien `/etc/modprobe.conf` und `/etc/modprobe.d/*.conf` gesteuert. Diese Konfigurationsdateien werden weiter unten genauer beschrieben.

Kernel und Module müssen zusammenpassen

In den Anfangstagen von Linux mussten der Kernel und seine Module exakt zusammenpassen: Es war nicht möglich, ein Modul zu laden, das für eine andere, vielleicht nur geringfügig veränderte Kernelversion kompiliert wurde. Aus diesem Grund gab und gibt es bis heute für jede Kernelversion ein eigenes Modulverzeichnis `/lib/modules/n.n`.

2006 hat man mit *Module Versioning* versucht, zusammen mit einem Modul Zusatzinformationen zu speichern, die Aufschluss darüber geben, ob eine Zusammenarbeit zwischen dem Modul und dem Kernel auch bei unterschiedlicher Versionsnummer

möglich ist. Damit konnten oft auch nicht zur Kernelversion passende Module genutzt werden.

Dieser Mechanismus funktioniert allerdings nur, wenn das Module Versioning beim Kompilieren aktiviert wurde und wenn es zwischen der Kernel- und der Modulversion keine Änderungen an den Schnittstellen gegeben hat. In der Praxis hat sich dieser Mechanismus nicht besonders gut bewährt, weswegen seine Anwendung heute unüblich ist.

Kommandos zur Modulverwaltung

Alle gängigen Distributionen sind so eingerichtet, dass Module bei Bedarf automatisch geladen werden. Ein Beispiel: Sie binden mit `mount` das Dateisystem eines USB-Sticks in den Verzeichnisbaum ein. Daraufhin wird automatisch das `vfat`-Modul aktiviert, das zum Lesen des Dateisystems erforderlich ist.

Im Regelfall erfolgt die Modulverwaltung also automatisch und transparent, ohne dass Sie mit den im Folgenden beschriebenen Kommandos zur manuellen Modulverwaltung eingreifen müssen. Dennoch sollten Sie die Modulkommandos kennen, um Module zur Not auch manuell laden zu können.

Alle Module befinden sich im Verzeichnis `/lib/modules/n.n`. Dabei ist `n.n` die Version des laufenden Kernels. Moduldateien haben die Dateiendung `*.ko` bzw. `.ko.xz`, wenn sie mit `xz` komprimiert sind.

Kernelversion ermitteln

Das Kommando `uname -r` liefert die Versionsnummer des laufenden Kernels:

```
user$  uname -r
6.3.1-arch2-1
user$  ls /lib/modules/6.3.1-arch2-1
extramodules
modules.alias
modules.builtin
modules.builtin.bin
...
```

Moduldatei laden modprobe

`modprobe` lädt das angegebene Modul in den Kernel. Dabei muss nur der Modulname angegeben werden. Zusätzlich können Parameter (Optionen) an das Modul übergeben werden. Falls Sie hexadezimale Werte angeben möchten, müssen Sie `0x` voranstellen, also etwa `option=0xff`.

```
root#  modprobe cifs
```

Im Vergleich zum Low-Level-Kommando `insmod`, auf das ich hier nicht eingehe, agiert `modprobe` relativ intelligent:

- `modprobe` sucht die Moduldatei selbst im verzweigten Modulverzeichnisbaum des gerade aktiven Kernels. Bei Modulen, die schon beim Kompilieren in den Kernel integriert wurden, endet `modprobe` ohne Fehlermeldung.
- `modprobe` lädt gegebenenfalls auch alle Module, die als Voraussetzung für das gewünschte Modul benötigt werden.
- `modprobe` berücksichtigt alle in `/etc/modprobe*` angegebenen Moduloptionen.

`modprobe` setzt eine korrekte Modulkonfiguration durch die Dateien `/etc/modprobe*` und `/lib/modules/n.n/modules.dep` voraus.

Liste der geladenen Module

`lsmod` liefert eine normalerweise recht lange Liste aller momentan geladenen Kernelmodule. Beachten Sie, dass `lsmod` in den Kernel einkompilierte Module nicht anzeigt, sondern nur solche Module, die nachträglich geladen wurden!

```
user$  lsmod | sort
Module                  Size  Used by
8250_dw                24576  0
ac97_bus               16384  1 snd_soc_core
acpi_pad               24576  0
acpi_thermal_rel       16384  1 int3400_thermal
aesni_intel           401408  12
...
```

Sehr hilfreich bei der Zuordnung von Kernelmodulen ist das Kommando `lspci` mit der Option `-k` (*Kernel driver in use*). Es listet alle über den PCI-Bus angeschlossenen Hardware-Komponenten auf und zeigt an, welche Treiber geeignet wären, um das Gerät zu steuern, und welcher Treiber tatsächlich verwendet wird:

```
user$  lspci -k
00:00.0 Host bridge: Intel Corporation 8th Gen Core Processor Host Bridge/DRAM
        Registers (rev 07)
        Subsystem: Lenovo 8th Gen Core Processor Host Bridge/DRAM Registers
        Kernel driver in use: skl_uncore
00:01.0 PCI bridge: Intel Corporation Xeon E3-1200 v5/E3-1500 v5/6th Gen Core
        Processor PCIe Controller (x16) (rev 07)
        Kernel driver in use: pcieport
00:02.0 VGA compatible controller: Intel Corporation UHD Graphics 630 (Mobile)
        Subsystem: Lenovo UHD Graphics 630 (Mobile)
        Kernel driver in use: i915
        Kernel modules: i915
...
```

Modulinformationen

`modinfo` liefert eine Menge Informationen über ein Modul. Das Modul muss sich nicht im Kernel befinden.

```
root#  modinfo cifs
filename:       /lib/modules/6.3.1-arch2-1/kernel/fs/cifs/cifs.ko
```

```
version:        2.42
description:    VFS to access SMB3 servers e.g. Samba, Macs, Azure and Windows
                (and also older servers complying with the SNIA CIFS Specification)
license:        GPL
author:         Steve French
...
parm:           cifs_min_rcv: Network buffers in pool. Default: 4 Range: 1 to 64
parm:           cifs_min_small: Small network buffers in pool. Default: 30 ...
...
```

Module entfernen

`rmmod` entfernt das angegebene Modul wieder aus dem Kernel und gibt den belegten Speicher frei. Das Kommando kann nur erfolgreich ausgeführt werden, wenn das Modul gerade nicht verwendet wird.

```
root#  rmmod cifs
```

Modulkonfiguration

Die Modulverwaltung funktioniert scheinbar wie von Zauberhand:

- Wenn Sie eine zusätzliche Partition in das Dateisystem einbinden und dabei ein bisher nicht genutztes Dateisystemformat zum Einsatz kommt, wird automatisch das Modul für dieses Dateisystem geladen.
- Wenn sich die Partition auf einer SATA-Disk befindet, werden auch die SATA-Module aktiviert, sofern diese nicht ohnedies schon geladen sind.
- Während der Initialisierung von Netzwerkfunktionen wird automatisch der erforderliche Treiber für Ihre Netzwerkkarte geladen etc.

Für das automatische Laden von Kernelmodulen ist die in den Kernel integrierte Komponente `kmod` verantwortlich. Dafür, dass all das funktioniert, sorgen unterschiedliche Konfigurationsmechanismen:

- **Für den Rechnerstart erforderliche Module:** Manche Kernelmodule werden sofort beim Start des Rechners benötigt – etwa Module zum Zugriff auf das Dateisystem. Sofern diese Module nicht integrale Bestandteile des Kernels sind, müssen sie in einer Initrd-Datei durch GRUB beim Rechnerstart an den Kernel übergeben werden (siehe Abschnitt 23.1, »GRUB-Grundlagen«).
- **Module für Grundfunktionen:** Die Module für die Basisverwaltung von Hardware-Komponenten (z. B. für das USB-System) werden von verschiedenen Scripts des Init-Prozesses direkt durch `modprobe`-Anweisungen geladen.
- **Module für Schnittstellen:** Eine Reihe weiterer Module wird dann geladen, wenn eine Schnittstelle zum ersten Mal benutzt wird. Hier tritt allerdings das Problem auf, dass es für manche Schnittstellen je nach der eingesetzten Hardware unterschiedliche Module gibt. Wenn Sie also die Schnittstelle `eth0` für die erste

Netzwerkkarte im Rechner ansprechen, muss das zu dieser Karte passende Modul geladen werden.

Da der Kernel nicht hellsehen kann, benötigt er eine Information darüber, welches Modul das richtige ist. Diese Information befindet sich in `/etc/modprobe.conf` sowie in den Dateien der Verzeichnisse `/etc/modprobe.d` und `/etc/modules-load.d`. Dort befinden sich installations- oder distributionsspezifische Optionen sowie Anweisungen, welche Module *nicht* automatisch zu laden sind (`blacklist`-Datei). Bei systemd-Distributionen werden diese Informationen durch die Service-Datei `systemd-modules-load.service` ausgewertet.

Auch die automatische Device-Verwaltung durch das `udev`-System lädt bei Bedarf die notwendigen Module. Die entsprechenden Regeln finden Sie in `/etc/udev/rules.d`.

- **Module für USB-Geräte etc.:** Derartige Hardware-Komponenten nehmen eine Sonderrolle ein. Die Datei `/lib/modules/n.n/modules.alias` enthält eine Referenz mit Identifikationscodes der Komponenten und entscheidet darüber, welches Modul geladen wird.
- **Modulabhängigkeiten:** Eine Menge Module sind voneinander abhängig. Beispielsweise funktioniert das Modul `nfs` für das NFS-Dateisystem nur, wenn auch die Module `lockd`, `nfs_acl` und `sunrpc` geladen sind. Derartige Modulabhängigkeiten sind zentral in der Datei `/lib/modules/n.n/modules.dep` verzeichnet.

Module beim Rechnerstart laden

Manchmal wollen Sie unabhängig von den hier zusammengefassten Konfigurationswegen erreichen, dass beim Rechnerstart ein bestimmtes Kernelmodul geladen wird – und das, ohne sich auf irgendwelche Automatismen zu verlassen. Die optimale Vorgehensweise hängt von Ihrer Distribution ab.

Besonders einfach ist es bei Debian und Ubuntu: Dort kümmert sich das durch systemd einmalig ausgeführte Kommando `kmod` darum, alle in `/etc/modules` zeilenweise aufgelisteten Module zu laden. Sie müssen also lediglich das gewünschte Modul in einer neuen Zeile in `/etc/modules` angeben.

Bei Fedora und RHEL fügen Sie `modprobe modulname` in das für lokale Anpassungen vorgesehene Init-Script `/etc/rc.d/rc.local` ein. Beachten Sie aber, dass die Module damit je nach Distribution erst zum Ende des Init-Prozesses geladen werden – also zu einem Zeitpunkt, zu dem es für manche System- und Netzwerkprozesse schon zu spät ist.

modprobe-Syntax

Die folgenden Absätze beschreiben die wichtigsten Schlüsselwörter für die Dateien `modprobe.conf`, `/etc/modprobe.d/*` sowie `/lib/modules/n.n/modules.alias`. Weitere Details liefert `man modprobe.conf`.

alias

`alias`-Anweisungen geben an, welche Kernelmodule für welche Devices eingesetzt werden. Dazu ein Beispiel: Für das Device `/dev/eth0` soll das Modul `8139too` verwendet werden:

```
alias eth0 8139too
```

Der Zugriff auf viele Hardware-Komponenten erfolgt durch block- und zeichenorientierte Device-Dateien im `/dev`-Verzeichnis. Aus der Sicht des Kernels werden diese Device-Dateien nicht durch ihren Namen, sondern durch die Major- und Minor-Device-Nummer charakterisiert (siehe auch Abschnitt 11.10, »Device-Dateien«). Zahlreiche `alias`-Anweisungen stellen den Zusammenhang zwischen Device-Nummern und Modulen her.

Analog sieht auch die Definition von Netzwerkprotokollen aus: Um ein bestimmtes Protokoll zu nutzen, sucht der Kernel nach einer Protokollfamilie mit dem Namen `net-pf-N`, wobei `N` die ID-Nummer des Protokolls ist. Das folgende Beispiel bewirkt, dass für die Protokollfamilie 5 das AppleTalk-Modul geladen wird:

```
alias net-pf-5 appletalk
```

Wenn Sie dieses veraltete Protokoll nicht brauchen und womöglich das entsprechende Modul gar nicht installiert ist, dann erspart Ihnen die folgende Anweisung lästige Fehlermeldungen:

```
alias net-pf-5 off
```

options

`options`-Anweisungen geben an, mit welchen Optionen ein bestimmtes Modul geladen werden soll. Die folgende Anweisung bewirkt, dass das Modul `ne` (für NE-2000-kompatible Ethernet-Karten) mit der Option `io=0x300` geladen wird:

```
options ne io=0x300
```

include

`include`-Anweisungen laden weitere Konfigurationsdateien.

install

Mit `install`-Anweisungen geben Sie Kommandos an, die ausgeführt werden, anstatt das betreffende Modul einfach zu laden. Auch hierzu sehen Sie ein Beispiel, das aus Platzgründen auf zwei Zeilen verteilt wurde. Wenn das ALSA-Modul `snd` benötigt wird, sollen die folgenden Kommandos ausgeführt werden:

```
install snd modprobe --ignore-install snd $CMDLINE_OPTS && \
  { modprobe -Qb snd-ioctl32 ; : ; }
```

remove

Mit `remove` geben Sie Kommandos an, die beim Entfernen eines Moduls ausgeführt werden sollen.

blacklist

`blacklist` bewirkt, dass modulinterne Alias-Definitionen nicht berücksichtigt werden. `blacklist`-Anweisungen befinden sich üblicherweise in der Datei `/etc/modprobe.d/blacklist`. Sie enthält Module, die beispielsweise wegen Kompatibilitätsproblemen

oder aufgrund von besseren Alternativen *nicht* geladen werden sollen. Beispielsweise verhindert die folgende Zeile, dass das Modul `usbmouse` geladen wird:

```
blacklist usbmouse
```

25.2 Device Trees

Der Begriff »Device Tree« bezeichnet die hierarchische Darstellung von Hardware-Komponenten. Der Device Tree wird während des Boot-Vorgangs vom Kernel geladen und teilt diesem mit, welche Hardware-Komponenten zur Verfügung stehen und über welche Anschlüsse diese Komponenten genutzt werden.

Device Trees wurden von den Linux-Kernelentwicklern ersonnen, um der Vielfalt von Chips und Geräten (sprich: Smartphones) auf Basis von ARM-CPUs Herr zu werden. Dank Device Trees ist es möglich, dass ein für ARM-Geräte kompilierter Kernel auf unterschiedlichen Geräten mit unterschiedlichen Zusatzkomponenten laufen kann. Bei PCs ist das selbstverständlich; in der ARM-Welt war dies aufgrund der Vielfalt von CPU- und Hardware-Varianten aber bisher unmöglich.

Sofern Sie nicht gerade ein Kernelentwickler für Smartphones sind, kommen Sie am ehesten bei der Arbeit mit Minicomputern wie dem Raspberry Pi mit Device Trees in Kontakt. Device Trees werden beispielsweise in Raspberry Pi OS standardmäßig eingesetzt.

Beim Raspberry Pi beschreibt der Device Tree den Chip BCM2835/-36/-37 mit all seinen vielen Steuerungsmöglichkeiten, Bus-Systemen, GPIOs etc. Die Beschreibung erfolgt in einem Textformat, das dann aus Platz- und Effizienzgründen in ein binäres Format umgewandelt wird.

Die Details dieses Formats sind aus Anwendersicht nicht relevant. Wenn Sie sich dennoch dafür interessieren, finden Sie auf den folgenden Seiten eine umfassende technische Referenz:

https://devicetree.org
https://elinux.org/Device_Tree
https://linux-magazin.de/Ausgaben/2013/06/Kern-Technik

Device-Tree-Konfiguration beim Raspberry Pi

boot-Verzeichnis

Die Device-Tree-Konfiguration des Raspberry Pi ist über mehrere Orte verteilt. Das Verzeichnis `/boot` enthält Device-Tree-Beschreibungen für alle momentan gängigen Raspberry-Pi-Modelle. Die Dateikennung `*.dtb` steht dabei für *Device Tree Blob*, wobei ein *Blob* einfach ein binäres Objekt ist. Die Nummern 2708 bis 2711 sind Broadcom-

interne Nummern zur Beschreibung der Chip-Familie. Die auf dem Raspberry Pi eingesetzten Modelle BCM2835 bis BCM2837 sind konkrete Implementierungen (»Familienmitglieder«, wenn Sie so wollen).

- `bcm2708-rpi-0-w.dtb`: Raspberry Pi Zero W/WH
- `bcm2710-rpi-3-b.dtb/bcm2710-rpi-3-b-plus.dtb`: Raspberry Pi Modell 3B und 3B+
- `bcm2711-rpi-4-b.dtb`: Raspberry Pi Modell 4B
- `bcm2711-rpi-400.dtb`: Raspberry Pi 400

Während des Boot-Prozesses übergibt das in der Boot-Datei `start.elf` enthaltene Programm den für das jeweilige Raspberry-Pi-Modell geeigneten Device Tree an den Kernel.

overlays-Verzeichnis

Das Verzeichnis `/boot/overlays` enthält über 100 Device-Tree-Blob-Overlays (DTBOs), von denen ich hier nur einige wenige exemplarisch nenne:

- `hifiberry-dacplus-overlay.dtbo`: für die HiFiBerry-Erweiterung DAC+
- `lirc-rpi-overlay.dtbo`: für Infrarot-Fernbedienungen
- `i2c-rtc-overlay.dtbo`: für I²C-Komponenten mit Real Time Clock
- `w1-gpio-overlay.dtbo`: für 1-Wire-Temperatursensoren

Overlays sind also Ergänzungen zum Haupt-Device-Tree `bcmxxx`. Diese DTBs werden *nicht* automatisch geladen, sondern nur, wenn die Konfigurationsdatei `config.txt` entsprechende Hinweise enthält. Sie ergänzen den Device Tree des BCM28xx bzw. BCM2711.

config.txt

Die Raspberry-Pi-spezifische Datei `/boot/config.txt` kennt drei Schlüsselwörter zum Umgang mit Device Trees:

- `dtparam=i2c_arm=on/off,i2s=on/off,spi=on/off` aktiviert oder deaktiviert die Bussysteme I²C, I²S und SPI. Die Beschreibung dieser Bussysteme ist im Haupt-Device-Tree bereits enthalten. Standardmäßig sind alle drei Bussysteme inaktiv (entspricht `dtparam=i2c_arm=off,i2s=off,spi=off`). Wenn ein Bussystem oder mehrere genutzt werden sollen, müssen sie explizit aktiviert werden.
- `dtoverlay=name,key1=val1,key2=val2...` bewirkt, dass die genannte Overlay-Datei geladen wird, wobei die übergebenen Parameter berücksichtigt werden.
- `device_tree=`, also ohne die Zuweisung eines konkreten Werts, deaktiviert das gesamte Device-Tree-System.

Die Schlüsselwörter `dtparam` und `dtoverlay` können mehrfach verwendet werden.

Manuelle Konfiguration

Anhand der Device-Tree-Konfiguration entscheidet der Linux-Kernel, welche Module er lädt. Daher ersetzt die Device-Tree-Konfiguration die bei älteren Raspbian-Installationen erforderlichen Einstellungen in `/etc/modules` bzw. in `/etc/modprobe.d/*`.

In einfachen Fällen können Sie `config.txt` mit dem Programm `raspi-config` im Untermenü Advanced Options korrekt einrichten. Das gilt insbesondere, wenn Sie die Bussysteme I^2C und SPI verwenden möchten.

In allen anderen Fällen müssen Sie die entsprechenden Zeilen hingegen selbst in `config.txt` eintragen. Das folgende Listing illustriert die Syntax anhand einiger Beispiele. Beachten Sie, dass mehrere Optionen nur durch Kommata, aber nicht durch Leerzeichen voneinander getrennt werden dürfen (siehe z. B. `dtoverlay=...`).

```
# Datei /boot/config.txt
# Beispiele zur Steuerung des Device-Tree-Systems (weitere Details
# siehe /boot/overlays/README in einer Raspberry-Pi-OS-Installation)

# Audio-System aktivieren
dtparam=audio=on

# SPI-Bus aktivieren
dtparam=spi=on

# I2C-Bus aktivieren
dtparam=i2c_arm=on

# HiFiBerry DAC+ verwenden
dtoverlay=hifiberry-dacplus

# 1-Wire-Temperatursensor mit Standardeinstellungen verwenden
dtoverlay=w1-gpio-pullup

# 1-Wire-Temperatorsensor verwenden, der mit
# GPIO X verbunden ist (per Default GPIO 4),
# und dabei den internen Pull-up-Widerstand aktivieren
dtoverlay=w1-gpio-pullup,gpiopin=X,pullup=y

# Echtzeituhr-Modell ds1307 verwenden
dtoverlay=rtc-i2c,ds1307

# IR-Empfänger verwenden
dtoverlay=lirc-rpi
```

25.3 Kernelmodule selbst kompilieren

Wenn Sie Linux in Kombination mit VirtualBox einsetzen, die proprietären Grafiktreiber von NVIDIA nutzen möchten oder ein anderes hardware-spezifisches Kernelmodul brauchen, das im Kernel Ihrer Distribution fehlt, dann müssen Sie das Modul passend zum laufenden Kernel kompilieren.

Entwicklungswerkzeuge

Zum Kompilieren eines Moduls sind neben dem C-Compiler `gcc` und `make` auch weitere grundlegende Entwicklungswerkzeuge erforderlich. Die meisten Distributionen erleichtern die Sache durch fertige Paketselektionen oder Meta-Pakete, die auf alle relevanten Pakete verweisen (siehe Tabelle 25.1).

Distribution	Kommando
Debian, Ubuntu	`apt install build-essential`
Fedora, RHEL	`dnf groupinstall development-tools`
SUSE	`zypper install -t pattern devel_basis`

Tabelle 25.1 Kommandos zur Installation grundlegender Entwicklungswerkzeuge

Kernel-Include-Dateien

Außerdem brauchen Sie zumindest die Include-Dateien (Header-Dateien) zum aktuellen Kernel. Diese Dateien sind Teil des Kernelcodes. Bei vielen Distributionen (aber nicht bei SUSE) befinden sich die Include-Dateien und der Rest des Codes in zwei getrennten Paketen. Das hat den Vorteil, dass Sie nicht gleich den riesigen Kernelcode installieren müssen, wenn Sie nur die vergleichsweise kleinen Include-Dateien brauchen.

Tabelle 25.2 gibt an, in welchen Paketen sich die Include-Dateien des Kernels bei den gängigen Distributionen befinden und wohin diese Dateien installiert werden. `n.n` ist dabei ein Platzhalter für die installierte Kernelversion. Diese Information ermitteln Sie mit dem Kommando `uname -a`.

Distribution	Paket	Pfad
Debian	`linux-headers-n.n-amd64`	`/usr/include/linux-headers-n.n`
Fedora/RHEL	`kernel-devel-n.n`	`/lib/modules/n.n/build/include`
SUSE	`kernel-devel`	`/usr/src/linux-n.n/include`
Ubuntu	`linux-headers-n.n-generic`	`/usr/include/linux-headers-n.n`

Tabelle 25.2 Pakete mit den Kernel-Header-Dateien

Wenn Sie den Kernel selbst kompilieren (siehe Abschnitt 25.4), landen die zum Kernel passenden Include-Dateien automatisch im Verzeichnis `/lib/modules/n.n/build/include`.

Modul kompilieren

Die meisten Programme, die eigene Kernelmodule benötigen, enthalten ein Installations-Script, das sich um das Kompilieren und Einrichten des Moduls kümmert. Das gilt beispielsweise für VMware, VirtualBox, die Grafiktreiber von NVIDIA etc. Bei manchen Distributionen ist der Prozess sogar dahingehend automatisiert, dass nach

jedem Kernel-Update automatisch das Modul neu kompiliert wird (siehe *DKMS* weiter unten).

Wenn Sie dagegen den Quellcode für eine noch nicht offiziell unterstützte Hardware-Komponente heruntergeladen haben, müssen Sie sich um den Kompilierprozess selbst kümmern. Dazu führen Sie in der Regel die folgenden Kommandos aus. Nur das letzte `make`-Kommando erfordert `root`-Rechte.

```
user$  cd quellcodeverzeichnis
user$  make clean
user$  make
root#  make install
```

Modul-Updates automatisieren

DKMS

DKMS steht für *Dynamic Kernel Module Support* und hilft dabei, nach einem Kernel-Update selbst kompilierte Kernelmodule automatisch zu aktualisieren. DKMS besteht aus einigen Shell-Scripts und wurde von Dell entwickelt. Die entsprechenden `dkms`-Pakete stehen gegenwärtig für die Distributionen Debian, Fedora und Ubuntu zur Verfügung.

Um DKMS zu nutzen, muss der Quellcode des Moduls in einem Verzeichnis der Form `/usr/src/NAME-VERSION` installiert werden. Das Verzeichnis muss die Datei `dkms.conf` enthalten, die DKMS erklärt, wie es mit dem Code umgehen soll. Die folgenden Zeilen stammen vom NVIDIA-Treiber für Ubuntu, wobei ich die Formatierung des Listings geändert habe, um die Lesbarkeit zu verbessern. Tatsächlich sind vor und nach dem Zeichen = keine Leerzeichen erlaubt!

```
# Datei /usr/src/nvidia-525.105.17/dkms.conf
PACKAGE_NAME            = "nvidia#"
PACKAGE_VERSION         = "525.105.17"
CLEAN                   = "make clean"
BUILT_MODULE_NAME[0]    = "nvidia"
DEST_MODULE_LOCATION[0] = "/kernel/drivers/char/drm"
PROCS_NUM               = `nproc`
[ $PROCS_NUM -gt 16 ] && PROCS_NUM=16
MAKE[0]                 = "unset ARCH; [ ! -h /usr/bin/cc ] && export CC=..."
BUILT_MODULE_NAME[1]    = "nvidia-modeset"
DEST_MODULE_LOCATION[1] = "/kernel/drivers/char/drm"
BUILT_MODULE_NAME[2]    = "nvidia-drm"
DEST_MODULE_LOCATION[2] = "/kernel/drivers/char/drm"
...
```

Sind diese Voraussetzungen erfüllt, übergeben Sie das Kernelmodul mit `dkms add` der Kontrolle von DKMS, kompilieren es mit `dkms build` für den aktuellen Kernel und installieren es mit `dkms install`. Die folgenden Beispiele beziehen sich wieder auf den NVIDIA-Kerneltreiber. Die Kommandos werden normalerweise nach dem Update des

Kernels oder eines Treibers automatisch ausgeführt (Kommando `dkms autoinstall`). Nur wenn dieser Automatismus nicht funktioniert, müssen Sie manuell nachhelfen und sich bei eventuellen Fehlern auf die Suche nach deren Ursachen machen. (Mitunter sind die Versionen des Kernels und des Treibers nicht kompatibel zueinander.)

```
root#  dkms add     -m nvidia -v 525.105.17
root#  dkms build   -m nvidia -v 525.105.17
root#  dkms install -m nvidia -v 525.105.17
```

Die obigen Kommandos gelten für den gerade laufenden Kernel. Um Kernelmodule für eine andere Kernelversion zu kompilieren und zu installieren, fügen Sie den obigen Kommandos die Option `-k n.n` hinzu, wobei `n.n` die Versionsnummer eines auf Ihrem Rechner installierten Kernels ist.

`dkms status` bzw. ein Blick in das Verzeichnis `/var/lib/dkms` verrät, welche Kernelmodule sich momentan unter der Kontrolle von DKMS befinden.

DKMS unter Debian und Ubuntu

Bei Debian und Ubuntu gibt es eine ganze Reihe von `xxx-name-dkms`-Paketen, mit denen Kernelmodule für Hardware-Treiber kompiliert werden können:

```
root#  apt-cache --names-only search '.*-dkms' | sort
```

Bei Debian befinden sich die Pakete zum Teil in den Paketquellen *contrib* und *non-free*, die Sie eventuell vorher aktivieren müssen. Die Installation eines Kernelmoduls sieht dann wie folgt aus, wobei Sie einfach `nvidia` durch den Namen des gewünschten Treibers und `amd64` durch Ihre CPU-Architektur ersetzen:

```
root#  apt update
root#  apt install linux-headers-n.n-amd64 nvidia-nnn-dkms
```

Im Rahmen der Installation werden alle abhängigen Pakete installiert sowie das Kernelmodul kompiliert und als DKMS-Modul eingerichtet. Bei zukünftigen Kernel-Updates wird somit automatisch eine neue Version des Moduls erzeugt.

DKMS oder module-assistant?

Vor allem in Debian kamen zum Kompilieren von Kernelmodulen in der Vergangenheit häufig die Werkzeuge aus dem Paket `module-assistant` zum Einsatz. Das gleichnamige Kommando bzw. dessen Kurzform `m-a` existiert weiterhin, muss aber extra installiert werden; nach Möglichkeit sollten Sie DKMS-Pakete vorziehen!

akmods

Die RPMFusion-Paketquelle für Fedora verwendet anstelle von DKMS einen eigenen Mechanismus zum Kompilieren von Kernelmodulen: Das Kommando `akms` wird nach Kernel-Updates aufgerufen. Es kompiliert zu allen RPM-Source-Paketen, die sich in `/usr/src/akmods` befinden, die entsprechenden Kernelmodule und installiert diese.

https://rpmfusion.org/Packaging/KernelModules/Akmods

25.4 Kernel selbst konfigurieren und kompilieren

Der durchschnittliche Linux-Anwender muss seinen Kernel nicht selbst kompilieren. Bei allen aktuellen Distributionen werden ein brauchbarer Standardkernel und eine umfangreiche Sammlung von Modulen mitgeliefert. Dennoch kann es Gründe geben, den Kernel neu zu kompilieren:

- Sie wollen Ihr System besser kennenlernen. Das Motto dieses Buchs ist es ja, Ihnen auch einen Blick hinter die Linux-Kulissen zu ermöglichen.
- Sie brauchen besondere Funktionen, die weder in den mitgelieferten Kernel integriert sind noch als Modul vorliegen.
- Sie möchten eine aktuellere Version des Kernels verwenden als die, die mit Ihrer Distribution mitgeliefert wurde.
- Sie möchten selbst an der Kernelentwicklung teilnehmen und daher mit dem neuesten Entwicklerkernel experimentieren.
- Sie wollen in Ihrem Bekanntenkreis mit Insider-Wissen auftrumpfen: »Ich habe den neuesten Linux-Kernel selbst kompiliert!«

Hürden

Es gibt allerdings gewichtige Gründe, die gegen das Kompilieren eines eigenen Kernels sprechen:

- Viele Distributionen verwenden nicht den Originalkernel, wie er von Linus Torvalds freigegeben wird, sondern eine gepatchte Version mit diversen Zusatzfunktionen, wobei natürlich jede Distribution andere Patches verwendet.

  An sich ist das eine feine Sache für den Anwender: Er bekommt auf diese Weise Zusatzfunktionen, von denen der Distributor glaubt, dass sie schon ausreichend stabil funktionieren. Wenn Sie sich nun aber selbst den Quellcode des Originalkernels herunterladen, fehlen diese Patches. Einzelne Funktionen Ihrer Distribution, die bisher einwandfrei gearbeitet haben, machen plötzlich Probleme oder funktionieren gar nicht mehr.
- Das Kompilieren eines eigenen Kernels ist nicht schwierig. Schwierig ist aber die vorherige Konfiguration des Kompilationsprozesses. Dabei stehen Tausende von Optionen zur Auswahl. Sie können mit diesen Optionen beeinflussen, welche Funktionen direkt in den Kernel integriert werden, welche als Module und welche gar nicht zur Verfügung stehen sollen.

  Wenn Sie sich – mangels Detailwissen – für die falschen Optionen entscheiden, ist das Ergebnis wie oben: Einzelne Funktionen verweigern den Dienst, und es ist relativ schwierig, die Ursache herauszufinden. Gerade für Linux-Einsteiger ist es praktisch unmöglich, die passenden Einstellungen für alle Optionen richtig zu erraten.

Aus diesen Gründen verweigern die meisten Distributoren jeden Support, wenn Sie nicht den mit der Distribution mitgelieferten Kernel verwenden. Lassen Sie sich von diesen Warnungen aber nicht abschrecken, es einmal selbst zu versuchen. Wenn Sie nach der in diesem Abschnitt präsentierten Anleitung vorgehen, können Sie Ihren Rechner anschließend sowohl mit dem alten als auch mit dem neuen Kernel hochfahren – es kann also nichts passieren!

Entwicklungswerkzeuge

Zur Kompilierung des Kernels sind dieselben Entwicklungswerkzeuge wie zum Kompilieren eines einzelnen Moduls erforderlich (siehe Abschnitt 25.3).

Je nach Distribution sind darüber hinaus weitere Pakete notwendig. Unter Debian und Ubuntu müssen Sie beispielsweise in `/etc/apt/sources.list` die `deb-src`-Paketquellen aktivieren und dann die folgenden Kommandos ausführen:

```
root#  apt update
root#  apt install flex bison
root#  apt build-dep linux
```

Unter Fedora sind häufig die folgenden Pakete nicht automatisch installiert:

```
root#  dnf install dwarves ncurses-devel zstd
```

Grundlagen

Kernelversionen

Die Weiterentwicklung des Linux-Kernels erfolgt nun schon seit Jahren im gleichen Rhythmus: Sobald Linus Torvalds befindet, dass die gerade in Entwicklung befindliche Kernelversion ausreichend stabil ist, wird diese offiziell freigegeben. Gleichzeitig beginnt eine rund zweiwöchige Phase, während der diverse Entwickler Linus Torvalds Patches für neue Funktionen senden. Danach dauert es typischerweise fünf bis sieben Wochen, bis alle Probleme und Fehler im neuen Code behoben sind und die nächste Kernelversion fertig wird.

Versionsnummern

Die Versionsnummer des Kernels besteht aus drei Teilen: *Major Release Number*, *Minor Release Number* und *Bugfix Release Number*. Als ich dieses Kapitel Mitte Mai 2023 überarbeitete, war die Kernelversion 6.3 aktuell. Diese hatte Linus Torvalds am 23. April freigegeben. Bis Mitte Mai gab es nur ein kleineres Update zur Behebung von Bugs und Sicherheitsproblemen. Die vollständige Versionsnummer lautete zu diesem Zeitpunkt also 6.3.1.

Long Term Releases

Grundsätzlich wird die Wartung alter Kernelversionen mit der Fertigstellung der jeweils neuesten Version beendet. Ausgenommen von dieser Regel sind ausgewählte Kernelversionen, die durch die Kernelentwicklergemeinde über mehrere Jahre gepflegt werden. Aktuell genießen die folgenden Kernelversionen Langzeitunterstützung: 4.14, 4.19, 5.4, 5.10, 5.15 sowie 6.1.

Für Linux-Distributoren gibt es drei Varianten, mit Kernel-Updates umzugehen:

- Vordergründig am einfachsten wäre es, stets die gerade aktuellste Kernelversion als Update anzubieten. Das kann aber zu Stabilitätsproblemen führen (vor allem dann, wenn weitere Komponenten, z. B. das Grafiksystem, nicht ebenfalls aktualisiert werden), weswegen die meisten Distributoren vor diesem Weg zurückschrecken. Zu den wenigen Ausnahmen zählen Fedora sowie Rolling-Release-Distributionen wie Arch Linux oder openSUSE Tumbleweed.
- Die nächstbeste Variante besteht darin, für die Distribution die gerade aktuellste Long-Term-Linux-Version zu verwenden. Das hat für den Distributor den großen Vorteil, dass er sich um die Pflege des Kernelcodes nicht selbst kümmern muss.
- Am aufwendigsten ist es, wenn ein Distributor nicht auf einen Langzeit-Kernel zurückgreift, sondern den ursprünglich mit der Distribution ausgelieferten Kernel selbst pflegt. Dazu müssen Sicherheits-Updates und fallweise auch Zusatzfunktionen aus aktuellem Kernelcode »rück-portiert« werden. Besonders bemerkenswert ist diesbezüglich Red Hat, das für seine Enterprise-Distributionen mit immensem Arbeitsaufwand ein ganzes Jahrzehnt lang alte Kernelversionen mit Sicherheits-Updates versorgt.

Statistik Der Kernel besteht zurzeit aus ca. 36 Millionen Zeilen Code. Der Großteil davon ist in C geschrieben, ein kleiner Teil in Assembler sowie in der Sprache Rust. Wenn Sie wissen möchten, wer bzw. welche Firmen zur Kernelentwicklung beitragen, verfolgen Sie einfach die Linux-News-Site *lwn.net*. Dort finden Sie zu jedem Kernel-Release eine statistische Aufarbeitung, wer die meisten Änderungen durchgeführt hat:

https://lwn.net/Articles/929582 (für Version 6.3)

Links Tipps zur Kompilierung des Kernels finden Sie auf der folgenden Seite:

https://kernelnewbies.org/FAQ

Wenn Sie sich für technische Interna interessieren, sind die Dokumentationsdateien des Kernelcodes sehr aufschlussreich. Gerade neue Funktionen des Kernels werden zuerst an dieser Stelle beschrieben, noch bevor die entsprechenden `man`-Seiten aktualisiert werden:

https://www.kernel.org/doc/Documentation

Kernelcode installieren

Der Quellcode für den Kernel befindet sich üblicherweise im Verzeichnis `/usr/src/linux`; nur bei Red Hat und Fedora gibt es abweichende Gepflogenheiten, die weiter unten behandelt werden. Falls dieses Verzeichnis leer ist, haben Sie den Kernelcode nicht installiert. Sie können nun wahlweise den Kernelquellcode Ihrer Distribution

installieren oder den gerade aktuellen offiziellen Kernelcode herunterladen. Weniger Probleme bereitet zumeist die erste Variante, insbesondere für Einsteiger.

Ist genug Platz auf der Festplatte?

Der Platzbedarf für den Kernelcode ist beachtlich: Die komprimierten Quellcodepakete sind mehr als 120 MiB groß. Nach dem Entpacken beträgt der Platzbedarf in etwa ein weiteres GiB und nach dem Kompilieren mit den resultierenden Binärdateien über 20 GiB! Zuletzt können Sie mit `make clean` zahllose Objektdateien wieder löschen, aber zwischenzeitlich brauchen Sie genug freien Speicherplatz.

Kernelcode der Distribution installieren

Bei den meisten Distributionen gibt es ein eigenes Paket, das den Kernelquellcode enthält. Tabelle 25.3 gibt für einige gängige Distributionen an, in welchen Paketen sich der Kernelcode befindet. Dabei ist `n.n` ein Platzhalter für die installierte Kernelversion.

Distribution	Paket
Debian, Ubuntu	`linux-source-n.n`
Fedora, Red Hat	`kernel-n.n` (Quellcodepaket!)
SUSE	`kernel-source`

Tabelle 25.3 Pakete mit dem Kernelquellcode

Bei Debian und Ubuntu wird der Kernelcode als `tar`-Archiv in das Verzeichnis `/usr/src` installiert. Sie müssen das Archiv selbst mit `tar xjf linux-n.n.tar.bz2` auspacken. Die Kennung `.bz2` deutet darauf hin, dass der Quellcode mit dem besonders effizienten bzip2-Verfahren komprimiert wurde.

Fedora

Fedora ist unter Kernelentwicklern eine besonders beliebte Distribution. Bevor Sie loslegen, müssen Sie diverse Entwicklerpakete installieren und den aktuellen User-Account für `pesign` freischalten. (`pesign` ist ein Kommando zum Signieren von UEFI-Programmen.)

```
user$  sudo dnf install fedpkg fedora-packager rpmdevtools \
         ncurses-devel pesign openssl
user$  sudo pesign
pesign: Nothing to do.
```

Die Fedora-Entwickler empfehlen, den Quellcode des Kernels sowie aller Fedora-Patches mit `fedpkg clone -a kernel` aus einem Git-Repository herunterzuladen. Details können Sie hier nachlesen:

https://fedoraproject.org/wiki/Building_a_custom_kernel

Offiziellen Kernelcode installieren

Der mit der Distribution mitgelieferte Kernel ist oft schon veraltet. Den aktuellen Kernelcode in Form von komprimierten `tar`-Archiven finden Sie hier:

https://www.kernel.org

Anstatt eine bestimmte Kernelversion herunterzuladen und mit ihr zu arbeiten, kann es sinnvoller sein, einen Klon des offiziellen Git-Repositorys für stabile Kernelversionen zu erzeugen. `git tag` ermittelt in Kombination mit `sort -V` (numerische Sortierordnung) die aktuellsten zehn im Code enthaltenen Versionen. `git checkout` aktiviert die Version, die Sie anschließend tatsächlich nutzen (kompilieren) möchten – hier den Release Candidate 2 der noch in Entwicklung befindlichen Version 6.4:

```
user$  git clone \
         git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git
user$  cd linux-stable
user$  git tag -l | sort -V | tail -n 10
v6.3-rc2
v6.3-rc3
v6.3-rc4
v6.3-rc5
v6.3-rc6
v6.3-rc7
v6.3.1
v6.3.2
v6.4-rc1
v6.4-rc2
user$  git checkout v6.4-rc2
```

Anfänglich wird durch `git clone` zwar deutlich mehr Code heruntergeladen (Download-Umfang gut 4 GiB, Platzbedarf auf der SSD ca. 6 GiB); dafür verlaufen spätere Updates bzw. Versionswechsel aber einfacher und schneller – elementare Grundkenntnisse des `git`-Kommandos einmal vorausgesetzt.

Egal, ob `tar`-Archiv oder `git`: Beachten Sie, dass es sich hier um den originalen Kernelcode handelt, wie ihn Linus Torvalds freigegeben hat – also ohne distributionsspezifische Patches. Dieser Kernel wird oft *Vanilla Kernel* genannt.

Varianten und Entwicklerzweige

Es gibt im Internet unzählige weitere Git-Repositories mit diversen Varianten und Entwicklungszweigen des Kernelcodes. Werfen Sie insbesondere einen Blick in das Blog des Kernelentwicklers Greg Kroah-Hartman (vierter Link)!

https://git.kernel.org
https://github.com/torvalds/linux
https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
http://www.kroah.com/log/blog/2019/06/15/linux-stable-tree-mirror-at-github

Mitgelieferte Kernelkonfigurationsdateien verwenden

Der Kernel besteht aus Tausenden von Einzelfunktionen bzw. Komponenten. Bei nahezu allen Funktionen können Sie vor dem Kompilieren angeben, ob sie direkt in den Kernel integriert werden, als Modul kompiliert werden oder gar nicht verfügbar sein sollen. Dieser Vorgang heißt den »Kernel konfigurieren«.

.config-Datei

Die Kernelkonfiguration wird durch die Datei `.config` im Verzeichnis `/usr/src/linux-n.n` bestimmt. Dabei handelt es sich um eine fast 8000 Zeilen lange Textdatei, die angibt, ob eine Funktion direkt in den Kernel integriert (`name=y`) oder als Modul kompiliert werden soll (`name=m`). Nicht benötigte Funktionen erscheinen in der Konfigurationsdatei nicht bzw. nur in Kommentarzeilen. Die Datei kann auch zusätzliche Einstellungen enthalten (`name=wert`). Die folgenden Zeilen zeigen den Beginn einer `.config`-Datei:

```
CONFIG_64BIT=y
CONFIG_X86_64=y
CONFIG_X86=y
CONFIG_INSTRUCTION_DECODER=y
CONFIG_PERF_EVENTS_INTEL_UNCORE=y
CONFIG_OUTPUT_FORMAT="elf64-x86-64"
```

Wenn Sie bei der manuellen Kernelkonfiguration (siehe den folgenden Abschnitt) keinen Ausgangspunkt haben, müssen Sie sich wirklich um alle Kerneloptionen kümmern. Gerade beim ersten Mal ist es so gut wie sicher, dass Sie irgendetwas übersehen werden. Sie sparen eine Menge Zeit und Mühe, wenn Sie die mit Ihrer Distribution mitgelieferte Kernelkonfigurationsdatei als Ausgangspunkt verwenden:

```
user$  cp old-config /pfad/zum/code/linux-n.n/.config
```

Dieses Verfahren hat leider einen Nachteil: Wenn der ursprüngliche Kernelcode andere Patches enthält als der neu zu kompilierende Code, enthält auch die ursprüngliche Konfigurationsdatei Optionen, die im neuen Code nicht vorgesehen sind. Das kann zu Problemen führen. Wie ich schon erwähnt habe, bauen viele Distributoren diverse Patches in ihren Kernel ein, die im Standardkernel nicht enthalten sind. Deswegen müssen Sie anschließend in das Quellcodeverzeichnis wechseln und dort das folgende Kommando ausführen:

```
user$  cd /pfad/zum/code/linux-n.n
user$  make oldconfig
```

`make oldconfig` wertet die vorhandene `.config`-Datei aus. Fehlen dort Optionen, die der aktuelle Kernelcode vorsieht, dann werden entsprechende Rückfragen angezeigt. In der Regel können Sie einfach mit ↵ antworten; dann gelten für diese Optionen die von den Entwicklern vorgeschlagenen Defaulteinstellungen. Genau das macht – ohne jede Rückfrage – das folgende Kommando:

```
user$  make olddefconfig
```

Aktuelle Konfiguration feststellen

Jetzt bleibt noch die Frage offen, woher Sie die aktuelle Kernelkonfigurationsdatei für das Kommando `cp old-config` nehmen. Bei nahezu allen Distributionen befindet sich im Verzeichnis `/boot` die zum laufenden Kernel passende Konfigurationsdatei, also z. B. `/boot/config-n.n`. Somit wird aus `cp old-config` beispielsweise:

```
user$  cd /pfad/zum/code/linux-n.n
user$  cp /boot/config-6.2.11-300.fc38.x86_64.config
user$  make oldconfig
```

cloneconfig

Der mit SUSE mitgelieferte Kernel verwendet die `cloneconfig`-Option (Gruppe *General setup*). Das bedeutet, dass `/proc/config.gz` den komprimierten Inhalt der `.config`-Datei enthält, mit der der gerade laufende Kernel kompiliert wurde. Mit `make cloneconfig` kopieren Sie die zuletzt verwendete Konfiguration in die Datei `.config`.

Kernel manuell konfigurieren

Monolithischer oder modularisierter Kernel?

Prinzipiell müssen Sie sich zwischen zwei Kerneltypen entscheiden: monolithischen Kerneln oder modularisierten Kerneln. Monolithische Kernel enthalten alle benötigten Treiber direkt im Kernel und unterstützen keine Module. Modularisierte Kernel sind über die integrierten Treiber hinaus in der Lage, im laufenden Betrieb zusätzliche Module aufzunehmen. Ein modularisierter Kernel ist in fast allen Fällen die bessere Entscheidung.

Komponentenauswahl

Bei den meisten Komponenten haben Sie die Wahl zwischen drei Optionen: YES, MODULE und NO. YES bedeutet, dass diese Komponente direkt in den Kernel integriert wird. MODULE bedeutet, dass diese Komponente als Modul kompiliert wird (nur sinnvoll bei einem modularisierten Kernel). NO bedeutet, dass die Komponente überhaupt nicht kompiliert wird. Es gibt auch eine Reihe von Funktionen, die nicht als Module zur Verfügung gestellt werden können – dort reduziert sich die Auswahl auf YES oder NO.

Konfigurationsstrategien

Die übliche Vorgehensweise besteht darin, in den modularisierten Kernel nur relativ wenige elementare Funktionen zu integrieren und alle anderen Funktionen als Module verfügbar zu machen. Der Vorteil: Der Kernel an sich ist relativ klein, Module werden nur nach Bedarf nachgeladen.

Eine alternative Strategie besteht darin, einen monolithischen Kernel möglichst exakt für die eigenen Hard- und Software-Ansprüche zu optimieren. Alle Funktionen, die genutzt werden sollen, integrieren Sie direkt in den Kernel. Bei allen anderen Komponenten entscheiden Sie sich für NO.

Generell wird ein monolithischer Kernel immer etwas größer als ein modularisierter Kernel. Dafür funktioniert er ohne die dynamische Modulverwaltung, und der Rechnerstart gelingt ohne Initrd-Datei. Der Nachteil ist offensichtlich: Wenn Sie eine

bestimmte Funktion später brauchen, müssen Sie den Kernel neu kompilieren. Und nur echte Linux-Profis können abschätzen, welche Funktionen sie nutzen werden.

Werkzeuge zur manuellen Kernelkonfiguration

Um abweichend von der aktuellen Konfiguration einzelne Einstellungen zu verändern, können Sie `.config` manuell editieren. Das ist aber fehleranfällig und erfordert eine gute Kenntnis der Namen der diversen Optionen. Besser ist es daher, mit `make xxxconfig` ein spezielles Konfigurationsprogramm zu starten. Dabei stehen unterschiedliche Varianten zur Verfügung, die Sie mit einem der aufgelisteten `make`-Kommandos starten:

```
user$ cd /usr/src/linux-n.n
user$ make menuconfig          (dialoggeführte Konfiguration im Textmodus)
user$ make nconfig             (dialoggeführte Konfiguration im Textmodus)
user$ make localmodconfig      (automatische Konfiguration für die aktuelle Hardware)
```

make menuconfig und make nconfig

`make menuconfig` und `make nconfig` setzen voraus, dass Sie vorher das Paket `ncurses-devel` bzw. `libncurses5-dev` installiert haben. Die Konfiguration erfolgt ebenfalls im Textmodus. Der große Vorteil im Vergleich zu `make config` besteht darin, dass die Einstellung der unzähligen Optionen durch verschachtelte Dialoge strukturiert ist (siehe Abbildung 25.1). Die Gestaltung der Dialoge und die Tastenkürzel variieren ein wenig, in ihrer Funktionsweise sind beide Varianten aber recht ähnlich.

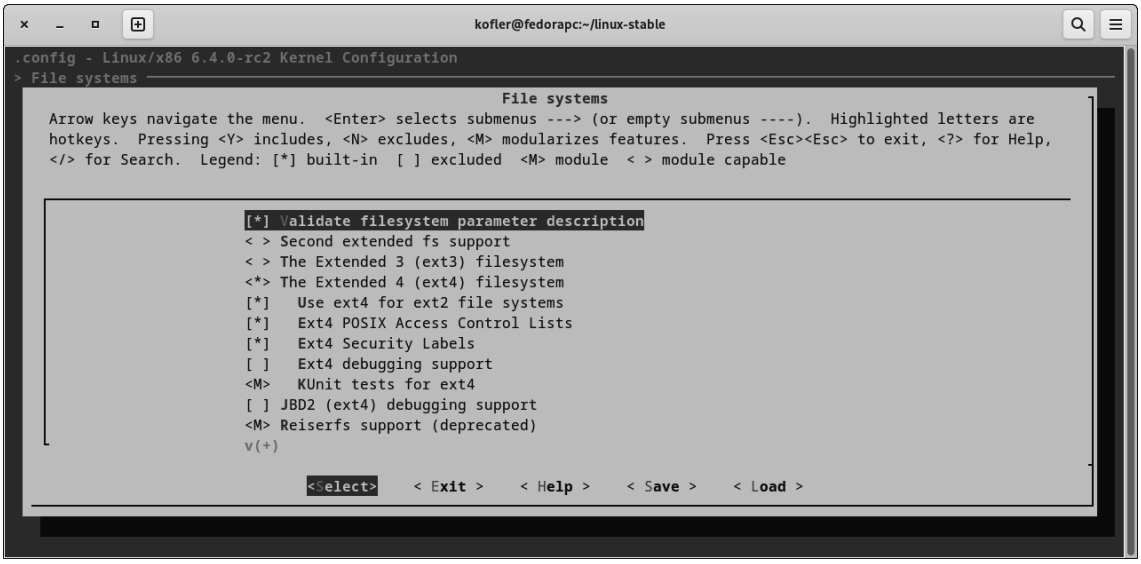

Abbildung 25.1 Kernelkonfiguration mit »make menuconfig«

make localmodconfig

`make localmodconfig` ist eine interessante Kompiliervariante für alle, die es eilig haben. Dabei werden nur die Module kompiliert, die im gerade laufenden Kernel tatsächlich genutzt werden. Das hat Vor- und Nachteile: Der offensichtliche Vorteil besteht darin, dass wirklich nur der Teil des Kernelcodes übersetzt wird, der tatsächlich benötigt wird. Das kann die Übersetzungszeit auf ein Drittel senken!

Allerdings läuft der so kompilierte Kernel auf einem anderen Rechner unter Umständen nicht, wenn für seine Hardware-Komponenten relevante Treiber fehlen. Auch das Nachladen eines Moduls, das zur Kompilierzeit nicht aktiv war, wird scheitern. Der Kernel ist also nur zu Testzwecken geeignet, nicht aber für eine längerfristige Nutzung.

Detailinformationen zu dieser `make`-Variante können Sie hier nachlesen:

https://heise.de/-1402386

Nur wenige Änderungen durchführen

Wenn Sie nur einzelne Optionen an einer vorgegebenen Kernelkonfiguration ändern möchten, ist es am besten, auf `make xxxconfig` ganz zu verzichten. Stattdessen geben Sie die gewünschten Einstellungen in der getrennten Datei `config-local` an. Die hier gewählten Optionen haben Vorrang gegenüber `.config`.

Kernel kompilieren und installieren

Kernel und Module kompilieren

Nachdem Sie mit der Konfiguration des Kernels vermutlich einige Zeit verbracht haben, muss jetzt der Rechner arbeiten. Die folgenden Kommandos beschäftigen einen zeitgemäßen Rechner (SSD, 6 CPU-Cores mit Hyperthreading) circa 20 bis 30 Minuten. Die Option `-j N` gibt an, wie viele Prozesse `make` parallel starten soll.

```
user$  cd /pfad/zum/code/linux-n.n
user$  make -j 12 all                    (alles kompilieren, 12 virtuelle CPU-Cores)
```

Das Ergebnis am Ende dieses Prozesses ist die Datei `bzImage` im Verzeichnis `/pfad/zum/code/linux-n.n/arch/x86_64/boot`. Die Größe der Datei liegt meist zwischen 5 und 10 MiB und hängt davon ab, wie viele Funktionen direkt in den Kernel inkludiert sind und wie viele als Module bzw. überhaupt nicht kompiliert wurden.

Fehler beim Kompilieren

Wenn beim Kompilieren ein Fehler auftritt, sollten Sie naturgemäß versuchen, ihm auf den Grund zu gehen. Wenn das Problem bei einer Funktion auftritt, die für Sie nicht wichtig ist, können Sie die Konfiguration so ändern, dass die betroffene Funktion eben nicht kompiliert wird.

Hartgesottene Linux-Freaks können `make` einfach mit der zusätzlichen Option `-k` aufrufen, also z. B. `make -k all`. Diese Option bewirkt, dass Fehler ignoriert werden. `make` fährt also einfach mit der Kompilation der nächsten Datei fort. Wenn Sie Glück haben, betrifft das Kompilationsproblem ein für Sie unwichtiges Modul, das dann eben nicht zur Verfügung steht.

Module installieren

`make modules_install` kopiert die Moduldateien dorthin, wo die Kommandos zur Modulverwaltung (etwa `insmod`) diese erwarten: in das Verzeichnis `/lib/modules/n.n`. Dabei ist `n.n` die Versionsnummer des soeben kompilierten Kernels.

```
user$  sudo make modules_install          (Module samt Debug-Infos installieren)
```

Standardmäßig enthalten die neu erzeugten Kernelmodule eine Menge Debugging-Informationen. Sie sind deswegen mehr als zehnmal größer als »gewöhnliche« Module und führen in der Folge zu riesigen Initrd-Dateien (bei meinen Tests z. B. 400 MiB anstatt 38 MiB!). Sofern Sie nicht vorhaben, sich auf die Suche nach Kernelfehlern zu begeben, sollten Sie die Debugging-Informationen aus den Modulen entfernen. (In der Kerneldokumentation wird dieser Vorgang »stripping« genannt.) Anstelle des obigen Kommandos führen Sie dazu die folgende Variante aus:

```
user$  sudo make INSTALL_MOD_STRIP=1 modules_install      (nur Module installieren)
```

Kernel installieren

Der frisch erzeugte neue Kernel ist natürlich noch nicht aktiv. Bisher wurden nur eine Menge neuer Dateien erstellt, sonst nichts! Der neue Kernel kann erst beim nächsten Start von Linux aktiviert werden und auch dann nur, wenn Sie den Kernel in das `/boot`-Verzeichnis kopieren und eine dazu passende Initrd-Datei erzeugen. Außerdem ist es üblich, die beim Kompilieren verwendete Datei `.config` unter dem Namen `config-n.n` im `/boot`-Verzeichnis zu speichern. Damit wird dokumentiert, wie Ihr Kernel kompiliert wurde.

Dankenswerterweise kümmert sich `make install` um sämtliche gerade zusammengefassten Punkte:

```
user$  sudo make install                  (Kerneldateien installieren)
```

Falls Ihr System Kernelmodule benötigt, deren Code außerhalb des offiziellen Kernelcodes liegt (typischerweise der NVIDIA-Treiber), müssen auch diese Module neu kompiliert und berücksichtigt werden. Das erfolgt bei vielen Distributionen mittels DKMS. Beachten Sie, dass das Kompilieren des NVIDIA-Treibers bei neuen Kernelversionen oft scheitert bzw. die allerneueste Version des NVIDIA-Treibers voraussetzt!

Zuletzt müssen Sie die GRUB-Konfiguration aktualisieren:

```
root#  update-grub                                  (Debian und Ubuntu)
root#  grub2-mkconfig -o /boot/grub2/grub.cfg       (Fedora, RHEL, SUSE)
```

Neustart und Aufräumarbeiten

Ob alles funktioniert hat, merken Sie beim Neustart:

```
user$  uname -a
Linux fedorapc 6.4.0-rc2 ...
```

Sollte der neue Kernel aus irgendeinem Grund nicht funktionieren, starten Sie den Rechner einfach mit dem bisherigen Kernel und unternehmen einen weiteren Versuch, den Kernel richtig zu konfigurieren und neu zu kompilieren. Läuft der neue Kernel dagegen zufriedenstellend, sollten Sie die nun nicht mehr benötigten Objekt-Dateien des Compilers aufräumen. Sie gewinnen auf diese Weise ca. 15 GiB Platz auf Ihrer SSD zurück!

```
root#  cd /pfad/zum/code/linux-n.n
root#  make clean
```

25.5 Kernelneustart mit kexec

Nach einem Update oder nach dem Kompilieren eines eigenen Kernels ist es üblich, den neuen Kernel durch einen Neustart des Rechners zu aktivieren. Alternativ können Sie den Kernel aber auch über die `kexec`-Funktion neu starten. Dann übergibt der laufende Kernel die Kontrolle an eine andere Kernelversion.

Im Vergleich zu einem echten Reboot ist ein Kernelneustart mit `kexec` um ein paar Sekunden schneller. Insbesondere entfallen die BIOS/EFI-Initialisierung und das Durchlaufen des GRUB-Prozesses. Die Downtime ist entsprechend um ein paar Sekunden kürzer, was vor allem Server-Administratoren freut. Dem steht allerdings der Nachteil entgegen, dass es unter Umständen Probleme bei der Neuinitialisierung diverser Hardware-Komponenten geben kann. Ein richtiger Neustart funktioniert zuverlässiger.

Installation (Debian/Ubuntu)

Bei vielen Linux-Distributionen ist das erforderliche `kexec`-Kommando standardmäßig installiert. Unter Debian und Ubuntu müssen Sie wie folgt nachhelfen:

```
root#  apt install kexec-tools     (Debian/Ubuntu)
```

Nach der Installation erscheint ein Konfigurationsdialog mit der Frage, ob `kexec` Neustarts verwalten soll. Wenn Sie die Frage mit JA beantworten, verwendet Ubuntu in Zukunft `kexec` bei allen Reboots. Nach meinen Erfahrungen funktioniert das durchaus nicht immer zuverlässig! Antworten Sie lieber mit NEIN, und probieren Sie `kexec` zuerst manuell aus. Wenn `kexec` auf Ihrer Hardware zuverlässig funktioniert, können Sie den Mechanismus später mit `dpkg-reconfigure kexec-tools` immer noch aktivieren. Beachten Sie, dass der Reboot-Mechanismus der `kexec-tools` auf einem Init-V-Script basiert. Es wird von systemd nur dank dessen Init-V-Kompatibilität ausgeführt.

Das kexec-Kommando

Der Kernelneustart wird nun in zwei Schritten initiiert. Zuerst übergeben Sie an `kexec` den Ort der neuen Kerneldatei, den Ort der Initrd-Datei sowie die gewünschten Kernelparameter. Dabei können Sie wahlweise eine Zeichenkette in der Form `--commandline="xxx"` übergeben oder einfach die bisher verwendeten Parameter mit `--reuse-cmdline` wiederverwerten. Welche Parameter zuletzt an den Kernel übergeben wurden, können Sie der Datei `/proc/cmdline` oder der GRUB-Konfigurationsdatei (üblicherweise `/boot/grub/grub.cfg`) entnehmen.

```
root#  kexec -l /boot/vmlinuz-6.3.1-arch2-1 \
           --initrd=/boot/initramfs-6.3.1-arch2-1 --reuse-cmdline
```

Im zweiten Schritt aktivieren Sie dann den neuen Kernel – entweder unmittelbar mit `kexec -e` oder nachdem Sie via systemd alle laufenden Dienste heruntergefahren haben. Auf einem Server ist die zweite Variante unbedingt vorzuziehen! Nur sie stellt sicher, dass z. B. ein Datenbankserver alle offenen Transaktionen zu Ende führen und alle Dateien ordnungsgemäß schließen kann.

```
root#  kexec -e                    (unmittelbarer Neustart)
root#  systemctl isolate kexec     (zuerst Dienste herunterfahren, dann Neustart)
```

Meine Erfahrungen mit `kexec` waren durchwachsen. Auf manchen Systemen bzw. mit manchen Distributionen, sowohl mit realer Hardware als auch in virtuellen Maschinen, funktionierte `kexec` einwandfrei. Bei der Mehrheit meiner Tests scheiterte der Kernelstart dagegen vollständig: Der Bildschirm wurde schwarz, das System reagierte erst nach einem Reset wieder.

25.6 Kernel-Live-Patches

Die meisten Updates eines Linux-Systems können im laufenden Betrieb erfolgen. Aktualisierte Netzwerkdienste müssen zwar anschließend neu gestartet werden, aber es besteht keine Notwendigkeit, den ganzen Rechner neu zu starten. Die Ausnahme von dieser Regel ist der Kernel: Damit Sicherheits-Updates im Kernel wirksam werden, müssen Sie einen neuen Kernel und neue Module installieren und anschließend den ganzen Rechner oder via `kexec` den Kernel und alle Dienste neu starten.

Auf Notebooks, die üblicherweise jeden Tag ein- und ausgeschaltet werden, ist das egal. Aber bei Servern, die möglichst ohne Unterbrechung ständig verfügbar sein sollen, ist ein Neustart zumeist unerwünscht. Und selbst Administratoren, die Updates automatisieren, scheuen in der Regel davor zurück, auch die erforderlichen Neustarts zu automatisieren. Zu groß ist die Gefahr, dass etwas schiefgeht und dann (zumeist mitten in der Nacht) keiner Zeit hat, das Problem zu beheben.

Ksplice

Die erste Lösung für dieses Problem bot die Funktion *Ksplice*: Bei vielen Updates ist es möglich, die betreffende Kernelfunktion im laufenden Betrieb zu deaktivieren und durch neuen Code zu ersetzen. Die nicht eben trivialen technischen Hintergründe des Verfahrens sind auf den folgenden Seiten beschrieben:

https://ksplice.oracle.com
https://lwn.net/Articles/340477
https://en.wikipedia.org/wiki/Ksplice

Mitte 2011 übernahm Oracle die Firma Ksplice. Mit der Funktion Ksplice durchgeführte Kernel-Updates waren über mehrere Jahre ein durchaus gewichtiges Unterscheidungsmerkmal zu anderen Enterprise-Linux-Versionen. Oracle bietet Ksplice allerdings nur zahlenden Kunden an. Ksplice steht zwar als Open-Source-Code zur Verfügung, ist aber nicht Bestandteil des offiziellen Linux-Kernels.

kPatch und kGraft

Red Hat und SUSE wollten in dieser Hinsicht natürlich nicht zurückstecken und entwickelten unter den Namen *kPatch* und *kGraft* vergleichbare Update-Mechanismen. Die beiden Mechanismen stehen seit 2014 in den Enterprise-Versionen von Red Hat und SUSE zur Verfügung. Die für kPatch und kGraft erforderlichen Funktionen (gewissermaßen der größte gemeinsame Nenner) wurden von Linus Torvalds 2015 in den offiziellen Linux-Kernel aufgenommen. Ob Ihr Kernel die Funktionen enthält, erkennen Sie am Vorhandensein des Verzeichnisses `/sys/kernel/livepatch`. Allerdings stellen auch Red Hat und SUSE geeignete Live-Kernel-Patches nur zahlenden Kunden zur Verfügung.

https://github.com/dynup/kpatch
https://en.wikipedia.org/wiki/KGraft

Ubuntu

Canonical trat zuletzt in das Live-Patching-Geschäft ein. Seit Ende 2016 bietet Canonical seinen kommerziellen Kunden für kritische Sicherheitsprobleme in Ubuntu-LTS-Versionen Live-Patches an, zuerst im Rahmen der Landscape-Dienste, mittlerweile unter der Bezeichnung »Ubuntu Pro«. Das zugrunde liegende Programm `canonical-livepatch` greift dabei auf die vorhin erwähnte Live-Patch-Infrastruktur im Kernel zurück.

Erfreulicherweise stellt Canonical die Live-Patches in beschränktem Umfang auch nichtkommerziellen Anwendern zur Verfügung: Sofern Sie über ein kostenloses Ubuntu-One-Konto verfügen, erhalten Sie einen Token, mit dem Sie fünf Rechner in das Ubuntu-Pro-Programm aufnehmen können:

```
user$  sudo apt install pro
user$  sudo pro attach <token>
...
This machine is now attached to 'Ubuntu Pro - free personal subscription'
```

```
SERVICE     ENTITLED  STATUS    DESCRIPTION
esm-apps         yes  enabled   Expanded Security Maintenance for Applications
esm-infra        yes  enabled   Expanded Security Maintenance for Infrastructure
fips             yes  disabled  NIST-certified core packages
fips-updates     yes  disabled  NIST-certified core packages with priority ...
livepatch        yes  enabled   Canonical Livepatch service
usg              yes  disabled  Security compliance and audit tools
```

Im Rahmen der Ubuntu-Pro-Aktivierung wird auch der Live-Patch-Dienst eingerichtet (siehe die vorletzte Zeile im obigen Listing). Die erforderliche Software steht ausschließlich als Snap-Paket zur Verfügung. Den Live-Patch-Status können Sie mit `canonical-livepatch status` ermitteln:

```
root#  canonical-livepatch status
last check: 24 minutes ago
kernel: 5.4.0-74.83-generic
server check-in: succeeded
patch state: OK, all applicable livepatch modules inserted
tier: updates (Free usage; This machine beta tests new patches.)
```

Achten Sie auf das Kleingedruckte!

Die Statusmeldung enthält einen Hinweis, der leicht zu übersehen ist: *This machine beta tests new patches*. Konkret bedeutet das, dass Sie als nicht zahlender Ubuntu-Pro-Nutzer ein Beta-Tester für Kernel-Updates sind! Canonical liefert Kernel-Patches zuerst an die nicht zahlenden Kunden aus. Wenn das zu keinen Problemen führt, werden die gleichen Updates später auch den zahlenden Kunden zur Verfügung gestellt.

Grundsätzlich ist diese Vorgehensweise nachvollziehbar. Ich nutze das kostenlose Ubuntu-Pro-Angebot auf mehreren Rechnern/Servern und bin bisher sehr zufrieden damit. Ärgerlich ist aber die instransparente Informationspolitik von Canonical. Obwohl ich eine Weile danach gesucht habe, bin ich im Internet auf kein offizielles Dokument gestoßen, das deutlich auf den Beta-Charakter der Kernel-Updates bei der kostenlosen Nutzung hinweist.

Wenn Sie detaillierte Informationen zu den behobenen Sicherheitslücken wünschen, geben Sie zusätzlich die Option `--verbose` an. (Die folgenden Ergebnisse stammen von einem anderen Rechner, der schon länger nicht mehr neu gestartet wurde.)

```
root#  canonical-livepatch status --verbose
  * cve-2023-0461
    It was discovered that the Upper Level Protocol (ULP) subsystem in the
    Linux kernel did not properly handle sockets entering the LISTEN state
    in certain protocols, leading to a use-after-free vulnerability. A
    local attacker could use this to cause a denial of service (system
    crash) or possibly execute arbitrary code.
```

```
  * cve-2023-1281
    It was discovered that the Traffic-Control Index (TCINDEX)
    implementation in the Linux kernel contained a use-after-free
    vulnerability. A local attacker could use this to cause a denial of
    service (system crash) or possibly execute arbitrary code.
```

`dmesg | grep livepatch` liefert Meldungen über zuletzt durchgeführte Live-Patches:

```
root#  dmesg | grep 'livepatch:'
livepatch: enabling patch 'lkp_Ubuntu_4_15_0_200_211_generic_94'
livepatch: 'lkp_Ubuntu_4_15_0_200_211_generic_94': starting patching transition
livepatch: 'lkp_Ubuntu_4_15_0_200_211_generic_94': patching complete
```

Live-Patches nur für Notfälle

Den Code des laufenden Kernels zu verändern, ist ein diffiziler Vorgang (wenn Sie ein Freund dramatischer Vergleiche sind: wie die Operation am offenen Herzen). Deswegen wird dieser Mechanismus aktuell von allen Distributoren nur für gefährliche Sicherheitslücken im Kernel verwendet.

Bei sonstigen Korrekturen wird wie bisher ein neuer Kernel installiert, auf die Aktivierung der dort durchgeführten Änderungen via Live-Patches aber verzichtet. Somit kann es trotz aktivierter Live-Patches sein, dass Ihre Distribution nach der Installation von (Kernel-)Updates meldet, dass ein Neustart erforderlich ist. Lassen Sie sich davon nicht verunsichern.

25.7 Die Verzeichnisse /proc und /sys

Die Verzeichnisse `/proc` und `/sys` werden während des Systemstarts in das Dateisystem eingebunden. Sie dienen dazu, Informationen über den Kernel, laufende Prozesse, geladene Module und viele andere Parameter auf eine transparente Art und Weise sichtbar zu machen.

Intern sind die Verzeichnisse `/proc` und `/sys` als virtuelle Dateisysteme realisiert. Sie enthalten also keine echten Dateien und beanspruchen daher auch keinen Platz auf der Festplatte. Das gilt auch für die scheinbar sehr große Datei `/proc/kcore`, die den Arbeitsspeicher abbildet.

Die meisten der `/proc`- und `/sys`-Dateien liegen im Textformat vor. Um die Dateien zu lesen, müssen Sie unter Umständen `cat` statt `less` verwenden, weil manche `less`-Versionen mit virtuellen Dateien nicht zurechtkommen.

Das `/proc`-Verzeichnis liefert eine Menge interner Kernelinformationen sowie Daten zu allen gerade laufenden Prozessen (siehe Tabelle 25.4). Unter anderem ist dort

jedem Prozess ein eigenes Unterverzeichnis zugeordnet. Innerhalb des Prozessverzeichnisses befinden sich dann einige Dateien mit diversen Verwaltungsdaten (z. B. die zum Start verwendete Kommandozeile). Diese Daten werden von diversen Kommandos zur Prozessverwaltung (z. B. `top`, `ps` etc.) ausgewertet.

Datei	Bedeutung
`/proc/N/*`	Informationen zum Prozess mit der PID=`N`
`/proc/asound`	ALSA (Advanced Linux Sound Architecture)
`/proc/bus/usb/*`	USB-Informationen
`/proc/bus/pccard/*`	PCMCIA-Informationen
`/proc/bus/pci/*`	PCI-Informationen
`/proc/cmdline`	an den Kernel übergebene Parameter
`/proc/config.gz`	Kernelkonfigurationsdatei (SUSE)
`/proc/cpuinfo`	CPU-Informationen
`/proc/devices`	Nummern von aktiven Devices
`/proc/fb`	Informationen zum Frame-Buffer
`/proc/filesystems`	im Kernel enthaltene Dateisystemtreiber
`/proc/interrupts`	Nutzung der Interrupts
`/proc/lvm/*`	Nutzung des Logical Volume Managers
`/proc/mdstat`	RAID-Zustand
`/proc/modules`	aktive Module
`/proc/mounts`	aktive Dateisysteme
`/proc/net/*`	Netzwerkzustand und -nutzung
`/proc/partitions`	Partitionen der Festplatten
`/proc/scsi/*`	SCSI/SATA-Geräte und -Controller
`/proc/sys/*`	System- und Kernelinformationen
`/proc/uptime`	Zeit in Sekunden seit dem Rechnerstart
`/proc/version`	Kernelversion

Tabelle 25.4 Wichtige /proc-Dateien

Das `/sys`-Verzeichnis enthält teilweise dieselben Informationen wie `/proc`, allerdings sind die Daten systematischer organisiert (siehe Tabelle 25.5). Das Ziel des `/sys`-Ver-

zeichnisses ist es, den Zusammenhang zwischen dem Kernel und der Hardware abzubilden.

Datei	Bedeutung
/sys/block/*	Informationen über alle Block-Devices (Festplatten etc.)
/sys/bus/*	Informationen über alle Bus-Systeme (PCI, SCSI, USB etc.)
/sys/class/*	Informationen über Device-Klassen (Bluetooth, Grafik etc.)
/sys/devices/*	Informationen über angeschlossene Hardware-Komponenten
/sys/firmware/*	Informationen über Hardware-Treiber und -Firmware
/sys/kernel/*	Informationen über den Kernel
/sys/module/*	Informationen über geladene Module
/sys/power/*	Informationen über die Energieverwaltung

Tabelle 25.5 Wichtige /sys-Dateien

25.8 Kernel-Boot-Optionen

Nicht immer, wenn ein Detail im Kernel geändert werden soll, muss der Kernel gleich neu kompiliert werden! Es gibt zwei Möglichkeiten, ohne ein Neukompilieren auf den Kernel Einfluss zu nehmen:

- Zum einen können Sie mit dem Bootloader während des Systemstarts Parameter an den Kernel übergeben. Dieser Mechanismus ist Thema dieses Abschnitts.
- Zum anderen können Sie eine Reihe von Kernelfunktionen dynamisch – also im laufenden Betrieb – verändern. Diese Art des Eingriffs ist insbesondere zur Steuerung von Netzwerkfunktionen gebräuchlich und wird in Abschnitt 25.9, »Kernelparameter verändern«, beschrieben.

GRUB Bei der Konfiguration von GRUB können Sie in der Zeile `linux` bzw. in der Datei `/etc/default/grub` Kernel-Boot-Optionen angeben. Derartige Optionen können Sie auch interaktiv beim Start eines Linux-Installationsprogramms oder beim Start von GRUB über die Tastatur eintippen (siehe Abschnitt 23.2, »GRUB-Bedienung (Anwendersicht)«). Die Syntax für die Angabe von Optionen sieht so aus:

```
linux /boot/vmlinux-n.n optionA=parameter optionB=parameter1,parameter2
```

Die Parameter zu einer Option müssen ohne Leerzeichen angegeben werden. Mehrere Optionen müssen durch Leerzeichen voneinander getrennt werden, nicht durch

Kommata. Hexadezimale Adressen werden in der Form `0x1234` angegeben. Ohne vorangestelltes `0x` wird die Zahl dezimal interpretiert.

Kernel-Boot-Optionen helfen oft dabei, Hardware-Probleme zu umgehen. Wenn der Linux-Kernel beispielsweise aufgrund eines fehlerhaften BIOS nicht erkennt, wie viel RAM Ihr Rechner hat, geben Sie den korrekten Wert mit dem Parameter `mem=` an.

Beachten Sie, dass die beim Linux-Start angegebenen Parameter nur Einfluss auf die in den Kernel integrierten Treiber haben! Parameter für Kernelmodule müssen dagegen in der Datei `/etc/modprobe.conf` bzw. in den Verzeichnissen `/etc/modprobe.d` oder `/etc/modules-load.d` angegeben werden.

Dieser Abschnitt beschreibt nur die wichtigsten Kernel-Boot-Optionen. Weitere Informationen erhalten Sie mit `man bootparam` sowie auf den folgenden Seiten:

https://tldp.org/HOWTO/BootPrompt-HOWTO.html
https://www.kernel.org/doc/Documentation/admin-guide/kernel-parameters.txt

Wichtige Kernel-Boot-Optionen

- `root=/dev/sdb3`: Die `root`-Option gibt an, dass nach dem Laden des Kernels die dritte primäre Partition des zweiten SATA-Laufwerks als Systempartition für das Root-Dateisystem verwendet werden soll. Analog können natürlich auch andere Laufwerke und Partitionen angegeben werden.

  Wenn die Partition mit einem Label bezeichnet ist, kann die Systempartition auch in der Form `root=LABEL=xxx` angegeben werden. Insbesondere Fedora und Red Hat machen von dieser Möglichkeit Gebrauch. Als Name für die Systempartition wird üblicherweise das Zeichen / verwendet. Bei `ext`-Partitionen ermitteln Sie den Partitionsnamen mit `e2label` bzw. verändern ihn mit `tune2fs`.

  Eine weitere Variante ist die Angabe der Systempartition durch `root=UUID=nnn`, wobei `nnn` die UUID der Festplattenpartition ist. Diese Identifikationsnummer ermitteln Sie mit `/lib/udev/vol_id partition`.

- `ro`: Die Option `ro` gibt an, dass das Dateisystem vorerst *read-only* gemountet werden soll. Das ist (in Kombination mit einer der beiden folgenden Optionen) praktisch, wenn ein defektes Dateisystem manuell repariert werden muss.

- `init`: Nach dem Kernelstart wird bei den meisten Distributionen systemd gestartet (siehe Kapitel 24, »Das Init-System«). Wenn Sie dies nicht wollen, können Sie mit der Option `init` ein anderes Programm angeben.

  Mit `init=/bin/sh` erreichen Sie beispielsweise, dass eine Shell gestartet wird. Die Option kann Linux-Profis helfen, ein Linux-System wieder zum Laufen zu bringen, wenn bei der Init-Konfiguration etwas schiefgegangen ist. Beachten Sie, dass das

root-Dateisystem nur read-only zur Verfügung steht. (Das können Sie mit `mount -o remount` ändern, siehe Abschnitt 22.8, »mount und /etc/fstab«.) Beachten Sie außerdem, dass in der Konsole das US-Tastaturlayout gilt und dass die `PATH`-Variable noch leer ist.

- `single` oder `emergency`: Wenn Sie eine dieser Optionen verwenden, startet der Rechner im Single-User-Modus (Init-V) bzw. im Rescue-Modus (systemd). Genau genommen werden diese Optionen nicht vom Kernel ausgewertet, sondern so wie alle unbekannten Optionen an das erste vom Kernel gestartete Programm weitergegeben (siehe Kapitel 24, »Das Init-System«).
- `initrd=name`: `initrd` gibt den Namen der zu ladenden Initial-RAM-Disk-Datei an. Wenn Sie *keine* Initrd-Datei verwenden möchten, geben Sie `initrd=` oder `noinitrd` an.
- `ipv6.disable=1`: Diese Option deaktiviert alle IPv6-Funktionen des Kernels.
- `reserve=0x300,0x20`: Diese Option gibt an, dass die 32 Bytes (hexadezimal 0x20) zwischen 0x300 und 0x31F von keinem Hardware-Treiber angesprochen werden dürfen, um darin nach irgendwelchen Komponenten zu suchen. Die Option ist bei manchen Komponenten notwendig, die auf solche Tests allergisch reagieren. Sie tritt im Regelfall in Kombination mit einer zweiten Option auf, die die exakte Adresse der Komponente angibt, die diesen Speicherbereich für sich beansprucht.
- `pci=bios|nobios|nommconf`: Diese Option steuert, wie die Hardware-Erkennung von PCI-Komponenten erfolgt. Sollten dabei Probleme auftreten, hilft manchmal `pci=bios` oder `pci=nommconf`.
- `quiet`: Diese Option bewirkt, dass während des Kernelstarts keine Meldungen auf dem Bildschirm dargestellt werden.
- `video=1024x768`: Mit dieser Option kann per *Kernel Mode Setting* (KMS) die gewünschte Grafikauflösung eingestellt werden, falls der Kernel nicht selbst die optimale Auflösung wählt, z. B. wenn das Video-Signal über einen KVM-Switch geleitet wird. Wenn Sie auch die Farbtiefe (z. B. 24 Bit) und die Bildfrequenz angeben möchten, sieht die Syntax so aus: `video=1280x800-24@60`

  Die Einstellung der Grafikdaten funktioniert nur bei KMS-kompatiblen Treibern. Die `video`-Einstellung gilt normalerweise für alle angeschlossenen Monitore. Wenn Sie die Auflösung nur für einen einzelnen Monitor ändern möchten, geben Sie den entsprechenden Signalausgang an, z. B. `video=VGA-1:1024x768`.
- `nomodeset`: Diese Option deaktiviert das Kernel Mode Setting (KMS). Sie verhindert, dass bei einem Notebook mit NVIDIA-Grafikkarte, aber ohne die erforderlichen NVIDIA-Treiber der Bildschirm beim Start des Grafiksystems schwarz wird.
- `mitigations=auto|auto,nosmt|off`: Diese Option steuert, welche Schutzmaßnahmen der Kernel gegen CPU-Fehler ergreifen soll (siehe Abschnitt 25.10, »Spectre, Meltdown & Co.«).

- `dis_ucode_ld`: Diese Option verhindert, dass der Kernel Microcode-Updates an die CPU weitergibt (siehe Abschnitt 20.9, »Firmware-, BIOS- und EFI-Updates«). Sie sollte nur verwendet werden, wenn es aufgrund eines derartigen Updates zu Abstürzen oder Instabilitäten kommt.

SMP-Optionen

SMP steht für *Symmetric Multiprocessing* und bezeichnet die Fähigkeit des Kernels, mehrere CPUs bzw. CPU-Cores gleichzeitig zu nutzen. Sollten dabei Probleme auftreten, können die folgenden Optionen hilfreich sein:

- `maxcpus=1`: Wenn Sie bei einem Multiprozessorsystem Boot-Probleme haben, können Sie mit dieser Option die Anzahl der genutzten Prozessoren auf 1 reduzieren. Der Wert 0 entspricht der Option `nosmp`.
- `nosmp`: Diese Option deaktiviert die SMP-Funktionen. Der Kernel nutzt nur eine CPU.
- `noht`: Diese Option deaktiviert die Hyper-Threading-Funktion. Dank dieser Funktion verhalten sich viele CPUs so, als stünden mehrere Cores zur Verfügung. Daraus ergibt sich eine etwas höhere Rechenleistung, wenngleich die Steigerung nicht so hoch ist wie bei echtem SMP.
- `nolapic`: APIC steht für *Advanced Programmable Interrupt Controller* und bezeichnet ein Schema, um Hardware-Interrupts an die CPUs weiterzuleiten. Bei aktuellen Kernelversionen wird APIC immer aktiviert. Wenn Sie Probleme mit APIC vermuten, verhindern Sie durch `nolapic`, dass der Kernel den lokalen APIC aktiviert bzw. nutzt.
- `noapic`: Diese Option reicht etwas weniger weit als `nolapic` und deaktiviert nur den IO-Teil von APIC.
- `lapic`: Diese Option aktiviert APIC explizit. Das ist dann notwendig, wenn APIC durch das BIOS deaktiviert ist, aber dennoch genutzt werden soll.

ACPI-Optionen

Das Energieverwaltungssystem ACPI (*Advanced Configuration and Power Interface*) ist nicht nur für das Ein- und Ausschalten verantwortlich, sondern auch für den sparsamen Umgang mit Energie, für die Verwaltung verschiedener Hibernate-Modi etc. Im Folgenden sind die wichtigsten Optionen zur Steuerung der ACPI-Funktionen des Kernels zusammengefasst:

- `acpi=on/off`: Diese Option (de)aktiviert die ACPI-Funktionen im Kernel.

- `acpi=oldboot`: Damit werden die ACPI-Funktionen nur während des Boot-Vorgangs genutzt. Sobald der Rechner läuft, werden die ACPI-Funktionen aber nicht mehr verwendet.
- `pci=noacpi`: Diese Option deaktiviert die Interrupt-Zuweisungen durch ACPI.
- `noresume`: Diese Option bewirkt, dass vorhandene Hibernate-Daten in der Swap-Partition ignoriert werden. Sie ist also dann sinnvoll, wenn der Rechner nicht mehr richtig aufwacht, z. B., weil die Hibernate-Daten defekt sind.

25.9 Kernelparameter verändern

Viele Parameter des Kernels können im laufenden Betrieb über das `/proc`-Dateisystem verändert werden. Das folgende Beispiel zeigt, wie Sie die Masquerading-Funktion aktivieren, um den Rechner als Internet-Gateway für andere Rechner einzusetzen:

```
root#  echo 1 > /proc/sys/net/ipv4/ip_forward
```

sysctl

Einen eleganteren Weg bietet das Kommando `sysctl`, das mit den meisten aktuellen Distributionen mitgeliefert wird. Das analoge Kommando, um das Masquerading wieder abzuschalten, würde so aussehen:

```
root#  sysctl -w net.ipv4.ip_forward=0
```

`sysctl -a` liefert eine Liste aller Kernelparameter zusammen mit ihren aktuellen Einstellungen. Mit `sysctl -p` können die in einer Datei gespeicherten `sysctl`-Einstellungen aktiviert werden. Als Dateiname wird üblicherweise `/etc/sysctl.conf` verwendet. Die Syntax ist in der Manual-Seite zu `sysctl.conf` beschrieben. Viele Distributionen sehen vor, dass diese Datei während des Init-Prozesses automatisch ausgewertet und ausgeführt wird.

25.10 Spectre, Meltdown & Co.

Im Winter 2018 wurde bekannt, dass die CPUs mehrerer Hersteller (besonders betroffen ist Intel) gravierende Design-Schwächen enthalten, die es einem Prozess erlauben, unerlaubterweise die Daten anderer Prozesse zu lesen. Die ersten derartigen Sicherheitslücken erhielten die klingenden Namen *Spectre* und *Meltdown*. Seither wurden diverse weitere verwandte Probleme entdeckt: *Fallout*, *Foreshadow*, *RIDL*, *ZombieLoad* usw.:

https://en.wikipedia.org/wiki/Transient_execution_CPU_vulnerability