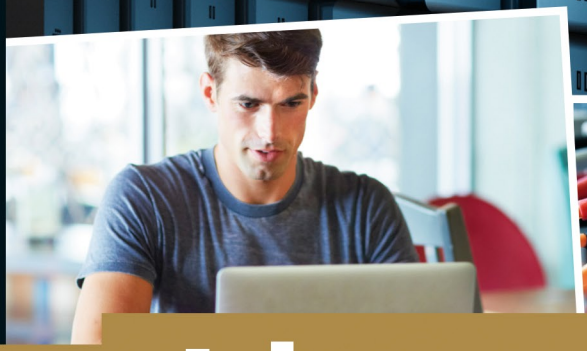
Dirk Deimeke
Stefan Kania
Daniel van Soest
Peer Heinlein
Axel Miesen

- CentOS 7
- Debian GNU/Linux 9
- openSUSE Leap 15
- Ubuntu Server 18.04 LTS

Inkl. Container-Verwaltung



```
### Verkehr aus dem internen
$IPT -A int_to_dmz -m state -
$IPT -A int_to_dmz -m state -
$IPT -A int_to_dmz -m state -
$IPT -A int_to_dmz -m state -
      -d $MAIL -j ACCEPT
$IPT -A int_to_dmz -m state -
                   mz -m state -
                   mz -j ext_to_
```

Linux-Server

Das umfassende Handbuch

- Linux-Server distributionsunabhängig einrichten und administrieren
- Backup, Sicherheit, Samba, Kerberos und LDAP, Web-, Mail- und FTP-Server, Datenbanken, KVM und Docker, Ansible u. v. m.
- Inklusive sofort einsetzbarer Praxislösungen

Mit allen Konfigurationsdateien zum Download

5., aktualisierte und erweiterte Auflage

Rheinwerk Computing

Vorwort

Willkommen zur fünften Auflage von »Linux-Server. Das umfassende Handbuch«! Ja, eine neue Auflage. Es gibt schon wieder so viele Neuerungen, dass es Zeit wurde, das Buch zu überarbeiten. Alle Distributionen, die in der letzten Auflage verwendet wurden, sind mittlerweile überarbeitet worden, und es gibt aktuelle Versionen.

In der letzten Zeit hat sich so einiges geändert. Alle hier verwendeten Distributionen setzen auf *systemd* und *journald*. Bei SUSE ist es sogar so, dass kein anderer Log-Daemon mehr verwendet wird. Bei Samba wurde der Support für die Version 3 komplett eingestellt. Mit Windows 10 ist die erste Version der Microsoft-Betriebssysteme erschienen, die keine *NT-Style*-Domänen mehr unterstützt. Aus diesem Grund haben wir uns entschieden, nur noch die Einrichtung von Active Directory-Domänen mit Samba zu erklären, da wir auch nur noch Windows 10 als Microsoft-Client verwenden.

Beim Thema Datenbanken haben wir uns entschieden, MySQL durch MariaDB zu ersetzen, da MySQL zum Teil gar nicht mehr in den Repositorys der Distributionen vorhanden ist. Fast alle Kapitel wurden von uns komplett überarbeitet, teilweise neu geschrieben, um möglichst aktuell zu bleiben. Hier war es uns wieder besonders wichtig, die neuen Funktionen von Programmen, die uns wichtig erschienen, mit aufzunehmen und eventuell ältere Optionen und Vorgehensweisen herauszunehmen.

Zudem haben wir auch wieder viele Anregungen und Kommentare erhalten, die uns dazu inspiriert haben, eine neue Auflage zu schreiben. Bei den Distributionen hat sich auch etwas geändert. Bei Debian ist die Version Debian Stretch neu hinzugekommen, bei SUSE sind wir auf openSUSE Leap 15 umgestiegen. Von Ubuntu ist eine neue LTS-Version auf dem Markt, die Version 18.04. Bei CentOS werden wir die Version 7.x nutzen.

Wie schon bei der vierten Auflage wollen wir wieder Ihnen, dem Systemadministrator, mit diesem Buch eine Anleitung bieten, wie Sie die verschiedensten Dienste, die Ihnen ein Linux-System bereitstellen kann, schnell und einfach konfigurieren. Ohne große Umwege geht es schnell über die Konfiguration hin zu einem funktionsfähigen Dienst, den Sie dann an Ihre eigenen Bedürfnisse anpassen können. Zudem haben wir alle großen Neuerungen für Sie so zusammengefasst, dass Sie auch neue Techniken nachlesen und umsetzen können.

Wir wollen Ihnen ein Nachschlagewerk an die Hand geben, das Sie mit vielen verschiedenen Techniken und Diensten vertraut macht. In den einzelnen Kapiteln gehen wir auch immer wieder auf Besonderheiten der verschiedenen Distributionen ein. Gerade durch die Vielfalt der Dienste und Techniken können Sie dieses Buch wie ein Schweizer Taschenmesser nutzen: immer griffbereit und immer das richtige Werkzeug. Mit jeder Auflage bekommt dieses

Schweizer Taschenmesser wieder ein paar Werkzeuge mehr, und die bestehenden Werkzeuge wurden an vielen Stellen noch schärfer und präziser gemacht.

Ein neuer Autor ist in die Runde aufgenommen worden: Axel Miesen. Er hat auch gleich zwei neue Kapitel für das Buch geschrieben: das Kapitel zu Docker und zusätzlich noch das Kapitel zu Ansible. So finden Sie jetzt auch hier einen guten Einstieg in die Technik der Container und des Konfigurationsmanagements.

Für wen haben wir das Buch geschrieben?

Dieses Buch richtet sich an alle Linux-Systemadministratoren, die immer wieder vor der Aufgabe stehen, neue Dienste in ihrem Netzwerk zu etablieren, und die am Anfang einen möglichst schnellen und kompakten Einstieg in das Thema wünschen. Grundlegende Linux-Kenntnisse, wie sie zum Beispiel in LPIC-1 verlangt werden, sollten auf jeden Fall schon vorhanden sein, damit Sie die einzelnen Dienste erfolgreich in das eigene Netz integrieren können.

Wie können Sie mit diesem Buch arbeiten?

Wir haben das Buch so geschrieben, dass Sie gezielt mit den Beispielen aus den einzelnen Kapiteln einen neuen Dienst konfigurieren und testen können. An vielen Stellen haben wir aber auch auf andere Dienste, die hier im Buch beschrieben sind, verwiesen, um Ihnen die Möglichkeit zu geben, auch komplexere Aufgaben zu realisieren.

Was dieses Buch nicht ist

Dieses Buch ist kein Lehrbuch, um den Umgang mit Linux von Grund auf zu verstehen, dafür gibt es viele andere Bücher auf dem Markt. Auch war das Buch von Anfang an nicht dazu gedacht, einen oder mehrere einzelne Dienste bis ins Letzte zu konfigurieren. Denken Sie an Ihr Schweizer Taschenmesser: Es kann Ihnen bei vielen Aufgaben helfen, aber für spezielle Aufgaben gibt es spezielle Werkzeuge. Das Gleiche gilt für unser Buch.

Viele Aufgaben können Sie mithilfe unseres Buches erledigen, aber wenn es dann sehr speziell wird, brauchen Sie ein Buch, das genau dieses eine Thema bis in kleinste Detail beschreibt.

Vorwort von Dirk Deimeke

Im April 2008 kam Marcus Fischer, der Autor zahlreicher Ubuntu-Bücher beim Rheinwerk Verlag, mit der Idee auf mich zu, ein Linux-Adminbuch zu schreiben. Da es kein deutsches Werk gibt, das die Lücke zwischen Einsteigerbüchern und Fachbüchern schließt, die sich einem einzelnen Thema widmen, war und bin ich immer noch Feuer und Flamme. In den folgenden fünf Monaten arbeiteten wir zusammen mit Jan Watermann, dem damaligen Lektor, an dem Konzept des Buches. Uns war zu jedem Zeitpunkt klar, dass es ein Buch werden sollte, das viel Bezug zur Praxis hat und einige Probleme behandelt, denen Linux-Systemadministratoren täglich begegnen. Das schreibt sich so leicht in ein oder zwei Sätzen, aber es war ein längerer Dialog, da jeder eine etwas andere Vorstellung davon hatte, wie das Buch aussehen sollte. Der Begriff »Kochbuch« durfte aufgrund von Markenrechten nicht verwendet werden, traf aber das, was wir machen wollten, am besten.

Nachdem Marcus aufgrund seiner Dissertation keine Zeit hatte, an dem Buch zu arbeiten, ging die Suche nach Autoren los, und Mitstreiter wurden gefunden. Aufgrund interner Schwierigkeiten trennte sich die initiale Gruppe jedoch wieder, und es drohte das Aus. In einem zweiten Anlauf fanden sich dann die Autoren zusammen, die die erste Auflage des Buchs geschrieben haben. Stefan Kania ist außer mir aus der ersten Gruppe dabeigeblieben. Zu uns gestoßen sind für die zweite Auflage Stefan Semmelroggen, Daniel van Soest und Charly Kühnast. Mit der dritten Auflage hat sich das Team leider wieder geändert: Stefan Semmelroggen verließ das Team, und Peer Heinlein stieß dazu. In der vierten Auflage verließ uns Charly Kühnast. In der fünften Auflage dürfen wir jetzt Axel Miesen als zusätzlichen Autor begrüßen, der die Kapitel zu Ansible und Docker beigesteuert hat. Gleichzeitig durften wir für diese Auflage mit einem neuen Lektor – Christoph Meister – zusammenarbeiten.

Anfang 2011 erschien die Ursprungsversion des Buches. Aufgrund von Änderungen in den Distributionen und von Anregungen unserer Leser gingen wir mit dem gleichen Team in die zweite Runde für Mitte 2012. Nach den größeren Änderungen der dritten Auflage legten wir mit der vierten Auflage noch eins drauf und haben den Verlag gewechselt – nein, im Ernst, in diese Zeit fiel auch die Umbenennung des Verlags von Galileo Computing in Rheinwerk Verlag. Jetzt, in der fünften Auflage, arbeiten wir schwer daran, das Format zu halten und nicht zu ausschweifend zu werden. Wir kommen leider sehr nah an die technischen Limits, die ein Buch mit rund 1300 Seiten erreicht.

Seit der vierten Auflage bieten wir Unterstützung für die am weitesten verbreiteten Distributionen mit längerer Laufzeit und sind mit openSUSE Leap kompatibel mit dem SUSE Linux Enterprise Server, und mit CentOS sind wir kompatibel mit Red Hat Enterprise Linux. Sie, liebe Leser, haben diese Änderungen angenommen und uns bewiesen, dass wir auf dem richtigen Weg sind.

Neben einem intensiven Review und einem Test, ob die angegebenen URLs noch funktionieren, habe ich noch viele kleinere Änderungen in die fünfte Auflage aufgenommen.

In einem Ihrer Lieblingskapitel, »Der Administrator«, habe ich noch weitere Informationen zur DevOps-Kultur aufgenommen, die im Moment in aller Munde ist.

Natürlich wurden alle Beispiele mit den neuen Versionen der Distributionen getestet und entsprechend angepasst. Wir hoffen, dass wir Ihnen mit diesem Buch weiterhin Unterstützung bei Ihrer täglichen Arbeit geben können!

Danksagung
Allen voran möchte ich meiner Frau danken, ohne die mein Teil an diesem Buch nie möglich gewesen wäre.

Dann möchte ich Sebastian Kestel und Anne Scheibe vom Rheinwerk Verlag für ihre wertvollen Hinweise und ihre Geduld danken. Sie haben uns »Greenhorns« überhaupt erst in die Lage versetzt, ein solches Projekt in Angriff zu nehmen.

Sebastian hat mit Christoph Meister einen würdigen Nachfolger gefunden, der die Arbeit fortgeführt hat. Danke!

Aber auch hinter den Kulissen haben wir bei »den Rheinwerkern« helfende Hände gefunden, allen voran möchte ich unserer Korrektorin Friederike Daenecke danken, die jeden noch so kleinen sprachlichen Fehler gefunden und behoben hat. Danke! Unserem Hersteller Norbert Englert möchte ich danken, weil wir mit seiner Hilfe aus dem Manuskript ein ansehnliches Buch machen konnten.

Mein besonderer Dank gilt aber meinen Mitautoren Daniel, Peer, Axel und Stefan für die tolle Zusammenarbeit.

Dass die Idee, die diesem Buch zugrunde liegt, so erfolgreich ist, damit haben wir nicht gerechnet. Noch viel weniger haben wir damit gerechnet, dass dieses Buch begonnen hat, sich als Standardwerk zu etablieren.

Jetzt halten Sie, liebe Leserin, lieber Leser, bereits die fünfte Auflage in Ihren Händen. Sie ist möglich geworden, weil Sie uns mit Ihren Anregungen und Ihrer konstruktiven Kritik motiviert haben.

Danke!

Vorwort von Stefan Kania

Ich hatte meine erste Berührung mit Computern bei der Bundeswehr. Dort wurde 1982 ein Kurs »BASIC Programmierung unter CP\M« angeboten. Nach mehreren Jahren Abendschule habe ich dann von 1992 bis 1994 am *b.i.b. e.V.* in Paderborn eine Ausbildung zum Informatiker absolviert. Dort kam ich auch zum ersten Mal mit dem Betriebssystem Linux in Kontakt. Damals war es die Version 0.96 PL4, und die Installation erfolgte noch sehr abenteuerlich über 30 Disketten.

Im Anschluss an die Ausbildung folgte noch eine Zusatzqualifikation zum CNE/CNI (Certified Novell Engineer/Certified Novell Instructor). Nachdem ich von 1995 bis 1997 fest angestellt als Trainer und Netzwerkadministrator gearbeit hatte, habe ich mich 1997 selbstständig gemacht und bin seitdem als IT-Berater und IT-Trainer tätig.

In den letzten Jahren habe ich einige Projekte durchgeführt, bei denen es um die Umstellung von Windows NT auf die Kombination »LDAP und Samba« ging. Vor ein paar Jahren wurde

ich dann von einem meiner Lehrer des *b.i.b.* angesprochen, ob ich nicht einmal für die Lehrer einen Kurs zum Thema Samba und LDAP halten könne. Das war etwas, was sich jeder Schüler bestimmt schon mal gewünscht hat: endlich mal die Seiten zu wechseln. Für diesen Kurs ist dann die erste Version meines Samba-LDAP-Workshops entstanden, der in den folgenden Jahren immer mehr gewachsen ist und als Grundlage für die Kapitel über Samba und LDAP hier im Buch diente.

2008 hielt ich auf einem Kongress der Firma Heinlein Support als Referent einen Vortrag zum Thema »Serverkonsolidierung mit Samba«. Im Anschluss an meinen Vortrag sprach mich einer der ursprünglichen Autoren dieses Buches, Dr. Michael Schwartzkopff, darauf an, ob ich nicht an einem Buch für Linux-Administratoren mitwirken wolle. In einem spontanen Anfall von Leichtsinn sagte ich sofort zu. Da wusste ich ja noch nicht, was für eine Arbeit auf mich zukommen würde.

Ich wollte mit diesem Buch anderen die Möglichkeit geben, sich schnell in ein neues Thema einzuarbeiten, und zwar anhand von Beispielen, die auch gut nachvollziehbar sind. Ich habe zu den unterschiedlichsten Themen schon die verschiedensten Bücher und andere Veröffentlichungen gelesen, nur deckte sich das dort Gelesene in den seltensten Fällen mit der Art und Weise, wie ich mir die Herangehensweise an neue Themen wünsche. Ich habe immer erst einmal gerne eine Anleitung, nach der alles läuft, und dann kann ich anfangen, mit der Umgebung zu experimentieren, um immer tiefer in das Thema einzusteigen. Ich hoffe, dass ich es mit meinen Kapiteln geschafft habe, Ihnen eine Anleitung an die Hand zu geben, mit der Sie die ersten Schritte erfolgreich gehen können.

Die zweite Auflage wurde für mich aus dem Grunde spannend, dass ich die Idee hatte, alle Dienste möglichst gegen Kerberos authentifizieren zu lassen. Jetzt, in der fünften Auflage, habe ich die Kapitel zum Thema Samba 4 komplett umgebaut. Alles, was zum Thema Samba 3 gehörte, ist endgültig aus dem Buch verschwunden. Ich habe besonderen Wert auf die Einrichtung der Domaincontroller und der Fileserver gelegt. Gerade die Einrichtung der Freigaben auf einem Server und die dazugehörigen Berechtigungen sind auch in meinen Projekten immer wieder ein großes Thema.

Danksagung
Ohne die anderen Autoren wäre das Buch nicht das, was es jetzt ist. Deshalb gilt mein Dank natürlich meinen vier Mitstreitern bei diesem Buch. Aber auch meiner Lebensgefährtin möchte ich danken für ihr Verständnis für die viele Zeit, die ich wieder für das Schreiben des Buches benötigt habe. Und auch dafür, dass sie immer wieder Kapitel Korrektur gelesen hat.

Vorwort von Peer Heinlein

Als ich 1992 als Jugendlicher eine Computermailbox zu meinem Hobby machte, kam mir nie in den Sinn, dass dies auch fast 25 Jahre später noch mein täglicher Lebensinhalt sein würde.

Was anfangs noch ein MS-DOS-System war, wurde schon wenig später auf das damals noch revolutionäre und brandneue Linux-System umgerüstet. Den Um- und Irrweg über ein Windows-System habe ich darum nie gehen müssen – weder auf Servern noch auf meinen Privatcomputern –, und vermutlich liegt es daran, dass ich bis heute ein halbwegs entspanntes Verhältnis zu meinen Kisten habe. Sie machen schließlich das, was sie machen sollen. Meistens.

Die Vernetzung von Menschen und der freie Fluss der Kommunikation sind seitdem mein Lebensinhalt geworden. Seit rund 25 Jahren bin ich darum als Trainer dabei, anderen Administratoren die Linux-Systemadministration zu vermitteln: technische Fakten, vor allem aber auch »Softskills«, also Kompetenzen rund um das technische Verständnis der Abläufe, die Fähigkeit, sich selbst neue Themen zu erarbeiten, sicher und zielgerichtet Fehler einzukreisen und Fehlverhalten zu debuggen. Die Arbeit mit Menschen ist es, die Spaß macht. Computer selbst sind kein Selbstzweck, sie sind nur Mittel zum Zweck: Arbeitstiere. Aber praktische Arbeitstiere, wenn man sie effizient und sicher einsetzt.

In diesem Werk habe ich vor allem die Verantwortung für die Mailserver-Kapitel übernommen, schließlich habe ich bereits einige Fachbücher rund um Postfix und Dovecot veröffentlicht. In diesem Administrationshandbuch haben wir die Gelegenheit genutzt, statt eines umfassenden vollständigen Nachschlagewerks eine klare, nachvollziehbare Anleitung zum Aufbau eines eigenen Mailsystems auszuarbeiten, ohne allzu viel Grundlagenwissen vorauszusetzen oder den Anspruch zu haben, den perfekten Postmaster auszubilden.

Und getreu dem Motto »Zuerst hatten wir kein Glück, und dann kam auch noch Pech hinzu« haben ich auch das Kapitel »Backup und Recovery« zu verantworten, denn mit »Relax & Recover (ReaR)« kann die von vielen Administratoren so vernachlässigte Disaster Recovery bequem Einzug finden. Also: Vergessen Sie vor lauter Euphorie über Aufbau und Überarbeitung Ihrer Linux-Systeme nicht, rechtzeitig an den Plan B zu denken, falls es mal schiefgeht! Vielen Dank an Schlomo Schapiro für ReaR – und auch für die Fachkontrolle meines Backup-Kapitels.

Danksagung
Ein großes Dankeschön an unseren Nagios-Experten Sven Velt (*https://velt.biz/*), der auch in dieser Auflage wieder tatkräftig hinter den Kulissen am Monitoring-Kapitel mitgeholfen hat und maßgeblich den Aufbau des Naemon-Servers beigesteuert hat. Sven – wir arbeiten nun schon fast 15 Jahre zusammen und es ist immer wieder ein nicht nur fachliches, sondern auch menschliches Vergnügen, mit dir befreundet zu sein! Vielen Dank für so vieles in dieser Zeit!

Vielen Dank an die aktuellen und früheren Autoren Charly, Daniel, Dirk und vor allem auch an Stefan Kania. Auch mit ihm arbeite ich seit fast 15 Jahren zusammen und habe ihn nicht nur als fachlich jederzeit hochkompetenten Spezialisten, sondern auch privat sehr zu schätzen gelernt. Vielen Dank für diesen doch schon immens langen Weg, den wir zusammen gehen.

Der Dank an mein Team hier bei Heinlein Support kann gar nicht groß genug sein, denn Ihr haltet mir in so vielen Bereichen den Rücken frei, damit ausreichend Zeit bleibt, auch Projekte wie dieses Buch voranzutreiben. Vielen Dank für alles, was wir gemeinsam im Team leisten und was auch jeder Einzelne bewegt, vorantreibt, korrigiert und gestaltet.

Zu guter Letzt danke ich meinen Kindern Caro und Antonia sowie meiner Frau Ivonne für alles und uns als Familie dafür, dass alles so ist, wie es ist!

Vorwort von Daniel van Soest

Wie die Jungfrau Maria zum Kind, so bin ich zu diesem Buch gekommen – oder eher dazu, ein Koautor dieses Buches zu werden. Nun halten Sie bereits die fünfte Auflage in den Händen; davon hätte ich vor acht Jahren nicht einmal zu träumen gewagt.

Der Praxisbezug ist mir sehr wichtig, ebenso wie das Aufbauen von Hintergrundwissen. In meiner nun 15-jährigen Berufserfahrung im Kommunalen Rechenzentrum Niederrhein (KRZN) durfte ich viele Hürden überwinden, aber noch mehr Erfolge feiern. Ich habe in diesem Buch stets versucht, nicht nur die Technik zu erläutern, sondern auch einen Großteil meiner Erfahrung mit einfließen zu lassen. Für dieses Buch war einer meiner Leitsätze: »Man kann nur die Technik beherrschen, die man versteht«.

Ich hoffe, diesem Motto gerecht geworden zu sein und mit diesem Buch nicht nur eine Anleitung geschaffen zu haben, sondern Sie darin unterstützen zu können, die Technik zu verstehen, selbst kreative Ideen zu entwickeln und nicht nur stumpf nach Plan zu arbeiten.

Abschließend bleibt mir nur noch eins: Ihnen viel Spaß mit diesem Buch zu wünschen.

Danksagung
Vorab möchte ich mich bei meinen Koautoren Dirk, Stefan, Peer und Axel bedanken. Die Zusammenarbeit war sowohl kreativ als auch produktiv, auch wenn wir die ein oder andere Hürde meistern mussten: Das Ergebnis kann sich sehen lassen. Ebenso möchte ich mich bei Christoph Meister bedanken – ohne dein Lektorat, die Geduld und die guten Lösungsansätze wären wir jetzt nicht da, wo wir sind. Nicht vergessen werden darf auch Norbert Englert: Vielen Dank für die schönen Bilder und noch viel mehr für die Latex- und Satz-Unterstützung. Natürlich darf die Korrektorin nicht vergessen werden – vielen Dank, Frau Daenecke. Ebenso geht mein Dank an meine Band (4Dirty5): Danke, dass ihr mir die Möglichkeit gebt, den Ausgleich zu bekommen, den ich zum Alltag brauche, und dass ihr meine Abwesenheit verkraftet habt.

Zum Abschluss möchte ich mich bei der wichtigsten Person in meinem Leben bedanken, ohne viele Worte zu bemühen: Ich bin dankbar, dich gefunden zu haben. Danke, Nicole!

Vorwort von Axel Miesen

Zur fünften Auflage des Linux-Server-Handbuchs darf ich nun auch etwas beisteuern: Zwei neue Kapitel über die Themen Ansible und Docker. Wie kam es dazu? Schon seit Jahren hatte ich die Idee, irgendwann einmal etwas Gedrucktes zu veröffentlichen. Zumal ich nicht bei Null beginnen müsste, denn durch jahrelange Schulungen im Linux-Umfeld hatte ich genug eigene Unterlagen, die als Grundlage für ein solches Unterfangen dienen könnten. Natürlich scheute ich den Aufwand, aber die Vorstellung, den eigenen Namen auf einem Buchcover zu sehen, war doch sehr reizvoll. Durch meinen Kollegen Oliver Liebel kam der Kontakt mit dem Rheinwerk Verlag zustande, und recht bald kam dann der Vorschlag, dass ich doch etwas zur neuen Auflage des beliebten Linux-Server-Handbuchs beitragen könnte. Ansible und Docker hatte ich in den Jahren zuvor im Projekt- und Schulungsumfeld kennen und schätzen gelernt, und genau diese Themen dürfen im Jahr 2018 in einem umfassenden Linux-Handbuch sicher nicht fehlen – perfekt! Ich hoffe nun, dass Ihnen mit meinen Kapiteln die ersten Schritte im Konfigurationsmanagement und in der schönen neuen Containerwelt leichter fallen, und wünsche Ihnen viel Spaß und Erfolg!

Danksagung
Mein großer Dank gilt zunächst meinen Autorenkollegen, die dieses Buch in vielen Jahren zu einem großen Erfolg geführt haben und mir auch immer mit nützlichen Tipps weitergeholfen haben. Weiterhin danke ich meinen lieben Kollegen Oliver Liebel und Berno Janßen für das Korrekturlesen früher Kapitelversionen mit zahlreichen Verbesserungsvorschlägen. Und nicht zuletzt danke ich meiner Lebensgefährtin Ana und meiner Tochter Lena dafür, dass ich auch oft am Wochenende einmal die Tür zum Arbeitszimmer schließen und am Buch arbeiten durfte, denn Bücher schreibt man in der Freizeit, und das braucht immer sehr viel Verständnis seitens der Familie!

Über dieses Buch

An dieser Stelle möchten wir Ihnen erklären, was wir uns bei der Verwendung der verschiedenen Formatierungsmöglichkeiten gedacht haben. Hier finden Sie auch die Beschreibung zu den im Buch verwendeten Icons und die Begründung, warum wir uns gerade für diejenigen Distributionen entschieden haben, die im Buch verwendet werden.

Formales

Damit Sie den größtmöglichen Nutzen aus diesem Buch ziehen können, verwenden wir einige formale Konventionen, die im Folgenden erläutert werden.

Kommandozeile

Gleich zu Beginn ein Hinweis an den mausverwöhnten Windows-Nutzer: Wir werden im Rahmen dieses Buches hauptsächlich Gebrauch von der Kommandozeile machen, da sich viele Aufgaben unter Linux einfacher und ökonomischer durch einige Tastaturkommandos erledigen lassen. Nur in einem Kapitel stehen die grafischen Werkzeuge mehr im Vordergrund, und zwar im Samba-4-Kapitel. Da müssen Sie sich als Linux-Admin an die Verwendung von grafischen Werkzeugen gewöhnen. Denn wenn Sie eine Active Directory-Domäne verwalten wollen, kommen Sie an den grafischen Werkzeugen nicht vorbei.

Das soll allerdings nicht heißen, dass wir gänzlich auf den Komfort einer grafischen Umgebung verzichten, denn wie bei vielen Dingen im Leben gilt auch hier: Die Mischung macht's. Für viele Bereiche gibt es heute grafische Werkzeuge, gerade webbasierte, die Ihnen als Administrator das Leben leichter machen können. Auch wir nutzen diese Werkzeuge und werden an den entsprechenden Stellen auf sie eingehen.

Befehle eingeben

Für Kommandozeilenbefehle soll folgende Schreibweise verwendet werden: Im fließenden Text werden Konsolenbefehle durch die Verwendung von `Nicht-Proportionalschrift` gekennzeichnet.

Viele der Beispiele zu den Kommandos werden aber auch in Listings dargestellt. Alle Listings werden in `Nicht-Proportionalschrift` wiedergegeben. In den Listings werden Sie von der Befehlszeile bis zum Ergebnis alles nachvollziehen können, wie Sie hier im Beispiel sehen:

```
stefan@adminbuch~$ ps
 PID TTY          TIME CMD
```

```
 4008 pts/2    00:00:00 bash
 4025 pts/2    00:00:00 ps
```

Listing 1 Beispiel für ein Listing

Privilegierte Rechte

Für die Administration von Linux-Systemen werden Sie immer root-Rechte benötigen, um die entsprechenden Konfigurationsdateien bearbeiten oder um Dienste starten oder stoppen zu können.

Ubuntu vertritt im Unterschied zu anderen Linux-Distributionen eine eigene Philosophie: Der Standardbenutzer der ersten Installation kann jeden Administratorbefehl durch Voranstellen des Befehls `sudo` ausführen. Anschließend muss dann das Passwort des Standardbenutzers eingegeben werden:

```
stefan@adminbuch~$ sudo /etc/init.d/networking restart
[sudo] password for <user>: <Hier eigenes Passwort eingeben>
```

Listing 2 Arbeiten als root

Sind mehrere Befehle als Administrator einzugeben, so kann das Voranstellen von `sudo` auch lästig werden. In diesem Fall verschaffen Sie sich mit dem folgenden Befehl vorübergehend eine root-Shell:

```
stefan@adminbuch~$ sudo -s
[sudo] password for <user>: <Hier eigenes Passwort eingeben>
root@adminbuch~#
```

Listing 3 Eine root-Shell öffnen unter Ubuntu

Eingabe langer Befehle

Und noch eine weitere wichtige, eher technische Konvention: Einige der vorgestellten Kommandozeilenbefehle oder Ausgaben von Ergebnissen erstrecken sich über mehrere Buchzeilen. Im Buch kennzeichnet am Ende der entsprechenden Zeilen ein »\«, dass der Befehl oder die Ausgabe in der nächsten Zeile weitergeht.

Screenshots

Wie heißt es doch so schön: Ein Bild sagt mehr als tausend Worte. Wann immer es sinnvoll erscheint, soll daher ein Screenshot zur Erhellung des Sachverhalts beitragen.

Internetverweise

Da wir in diesem Buch sehr viele verschiedene Dienste ansprechen, ist es nicht möglich, alle Funktionen und Fähigkeiten eines Dienstes bis ins kleinste Detail zu beschreiben. Aus diesem Grund haben wir an geeigneten Stellen auf Internetadressen verwiesen.

Verweise auf Internetadressen werden besonders ausgezeichnet, wie zum Beispiel so: *www.debian.org*. Die Listings aus dem Buch können Sie hier herunterladen: *www.rheinwerk-verlag.de/4575*

Icons

Sie werden in den einzelnen Kapiteln am Rand häufig Icons finden, die Sie auf bestimmte Zusammenhänge oder Besonderheiten hinweisen sollen. Die Icons haben die folgenden Bedeutungen:

Hier wird es immer sehr wichtig

Wann immer Sie das nebenstehende Symbol sehen, ist Vorsicht angeraten: Hier weisen wir auf besonders kritische Einstellungen hin oder auf Fehler, die dazu führen können, dass das System nicht mehr stabil läuft. Damit sich die Warnungen deutlich vom restlichen Text abheben, haben wir diese Textbereiche dann noch zusätzlich mit einem grauen Kasten hinterlegt.

Beispiele – etwa für Konfigurationsdateien – haben wir mit diesem Symbol gekennzeichnet. Wir haben an vielen Stellen Beispiele eingefügt, die es Ihnen leichter machen, eine entsprechende Aufgabe umzusetzen.

Alle Textstellen, die wir mit diesem Icon versehen haben, sollten Sie unbedingt lesen! Hier handelt es sich um wichtige Hinweise zu den unterschiedlichen Distributionen, die wir verwenden, oder um wichtige Eigenschaften oder Konfigurationsmöglichkeiten eines Dienstes.

Es gibt keine fehlerfreie Software! Große und kleine Fehler, die bei den einzelnen Diensten bekannt sind, werden durch diesen kleinen »Bug« gekennzeichnet. Die nachweislich erste Erwähnung des Wortes »Bug« stammt übrigens von Grace Hopper, einer Computerpionierin aus den USA: *http://de.wikipedia.org/wiki/Grace_Hopper*

Bei diesem Symbol finden Sie nützliche Tipps und Tricks zu bestimmten Aufgaben.

Linux-Distributionen

Als damals der Gedanke zur ersten Auflage für dieses Buch aufkam, mussten wir uns erst einmal einig werden, welche Distributionen wir denn für das Buch verwenden wollten. Aufgrund der folgenden Kriterien haben wir dann unsere Entscheidung getroffen:

- Wir wollten auf jeden Fall mindestens eine Distribution, die *rpm*-Pakete, und eine, die *deb*-Pakete für die Softwareverwaltung nutzt.
- Da es in diesem Buch um Serverdienste geht, musste die Distribution nicht unbedingt die aktuellsten Pakete haben, wie man es gerne auf einem Desktop hat, sondern uns kam es

in erster Linie auf die Stabilität an. Dennoch haben wir bei manchen Diensten durchaus auf eine bestimmte minimale Versionsnummer geachtet.

- Die Distributionen sollten sehr verbreitet sein und oft in Firmen zum Einsatz kommen.
- Der Supportzeitraum sollte mindestens vier bis fünf Jahre betragen, also ungefähr die Laufzeit, die IT-Systeme in Unternehmen haben.

Aufgrund dieser Kriterien haben wir uns im Laufe der Zeit immer wieder Gedanken gemacht, welche Distribution wir einsetzen, so auch dieses Mal. Dabei ist die Auswahl auf die folgenden Distributionen gefallen:

- **Debian Stretch**
  Debian ist seit Jahren für stabile Versionen und hohe Zuverlässigkeit bekannt. Auch ist die Bereitstellung der Sicherheitsupdates für einen langen Zeitraum gesichert.
- **SUSE Leap**
  Viele Leser haben uns gefragt, warum wir nicht mehr mit SUSE arbeiten, und wir sehen auch, dass die SUSE-Distributionen, auch in Unternehmen, immer öfter eingesetzt werden. Gerade wenn es um Desktop-Systeme in Domänen geht, wird SUSE immer häufiger verwendet. Deshalb haben wir uns auch im Samba-4-Kapitel dafür entschieden, SUSE Leap als grafischen Client einzusetzen.
- **Ubuntu-Server 18.04 LTS**
  Der Ubuntu-Server basiert auf Debian und stellt mit der *LTS-(Long Term Support-)*Version eine gute Alternative zum Debian-Server dar. Der Ubuntu-Server setzt dabei auf neuere Pakete und Kernel als Debian, da bei Ubuntu die Releasezyklen kürzer sind.
- **CentOS 7**
  Dieses ist die zweite Auflage, die auch eine auf Red Hat basierende Distribution enthält. Bisher war es immer schwer, eine gute Distribution zu finden, die freie aktuelle Updates über lange Zeit zur Verfügung stellt. Mit CentOS 7 ist aber jetzt eine Distribution auf dem Markt, die aktuell ist und einen langen Support garantiert. Auch steigt die Verwendung von CentOS in Unternehmen an, sodass wir dieses Mal die Besonderheiten dieser Distribution aufgenommen haben.

Wenn Sie sich jetzt fragen: »Aber meine Lieblingsdistribution erfüllt die Punkte auch, warum ist die nicht dabei?«, können wir an dieser Stelle nur sagen, dass wir natürlich alle Dienste unter allen von uns verwendeten Distributionen konfiguriert und ausgetestet haben. Allein für das Testen mit vier verschiedenen Distributionen benötigt man schon eine geraume Zeit. Deshalb haben wir uns für diese vier Distributionen entschieden.

Jetzt bleibt uns nur noch, Ihnen viel Spaß mit dem Buch zu wünschen und zu hoffen, dass Ihnen unser Buch bei Ihrer täglichen Arbeit eine Hilfe sein wird.

Kapitel 2

Bootvorgang

Der Startvorgang eines Linux-Systems ist die Basis dafür, überhaupt etwas mit dem System anfangen zu können. Wir geben einen Einblick in den Bootloader und die initiale Ramdisk. Wir widmen uns init-Skripten und blicken auf »eventgesteuertes Starten« mittels »systemd«.

2.1 Einführung

Mit dem Bootloader wird das Betriebssystem gestartet. Nachdem das BIOS den mehr oder weniger ausführlichen Systemcheck durchgeführt hat, werden die Bootmedien in der Reihenfolge der Präferenzen abgearbeitet. Wenn es zur Festplatte kommt, werden die ersten 512 Byte der Festplatte ausgewertet; in diesen ist der *Master Boot Record* (MBR) zu finden. Von den 512 Byte sind die ersten 446 für den Bootloader reserviert. In diesem begrenzten Bereich lassen sich keine großen Programme unterbringen, daher wird der Bereich dafür genutzt, Code von anderer Stelle nachzuladen. Der frühere *Linux Loader* (LILO) ist heute kaum noch verbreitet, daher beschränken wir uns im Weiteren auf die Weiterentwicklung des *Grand Unified Bootloader* (GRUB) mit dem Namen *GRUB 2*.

2.2 Der Bootloader GRUB 2

Mit *GRUB 2* wurde GRUB von Grund auf neu entwickelt. Die Entwickler haben sich sehr viel Zeit gelassen und sich in kleinen Sprüngen der Version 2 genähert. GRUB2 ist bei openSUSE und Ubuntu in Version 2.02, in Debian in Version 2.02-beta3 und in CentOS in Version 2.02-beta2 enthalten. Da die Macher des Bootloaders einen sehr konservativen Ansatz bei der Versionierung verfolgen, darf man sich von »beta« nicht schrecken lassen. Die erste Version des Bootloaders hat beispielsweise nie die Version 1 erreicht; die höchste Versionsnummer war 0.97.

2.2.1 Funktionsweise

Der große Unterschied von GRUB 2 im Vergleich zu GRUB ist, dass die ehemaligen Stages 1.5 und 2, vom Laden der Dateisystemtreiber bis zum Anzeigen des Bootmenüs, zu einem einzigen Stage 2 zusammengelegt wurden. Dabei nutzt GRUB 2 einen minimalistischen und

sehr kleinen Kern und viele Module, die je nach Bedarf nachgeladen werden können, um auf die Konfigurationsdatei zugreifen zu können. Auf diese Weise unterstützt GRUB 2 auch das Starten von LVM oder Software-RAIDs mit *md*.

2.2.2 Installation

GRUB 2 wird genauso wie GRUB mit `grub-install` (bei CentOS und openSUSE mit `grub2-install`) installiert, allerdings müssen Sie bei GRUB 2 angeben, wo der Bootloader installiert werden soll. Dabei zeigt GRUB 2 deutlich weniger Ausgaben bei der Installation (siehe Listing 2.1):

```
# Debian und Ubuntu
grub-install /dev/sda
Installing for i386-pc platform.
Installation finished. No error reported.

# CentOS und openSUSE
Installing for i386-pc platform.
Installation finished. No error reported.
```

Listing 2.1 Installation von »GRUB 2«

2.2.3 Konfiguration

Die Konfigurationsdatei von GRUB 2 liegt in */boot/grub/grub.cfg* bzw. */boot/grub2/grub.cfg*. Bitte ändern Sie diese nicht von Hand, sie wird von den Skripten unter */etc/grub.d* erstellt. In diesem Verzeichnis wird den Skripten eine Nummer vorangestellt, um die Reihenfolge festzulegen. Das Verfahren, die Konfiguration aus einzelnen Bausteinen (Skripten) zusammenstellen zu lassen, macht GRUB 2 deutlich flexibler und besser automatisierbar als seinem Vorgänger: So werden installierte Kernel automatisch erkannt und in das Bootmenü aufgenommen.

Die hohe Flexibilität wird allerdings durch eine komplexere Konfiguration erkauft. Ohne gutes Shell-Scripting-Know-how kommt man da nicht viel weiter. Einfachere Konfigurationen wie das Bootmenü sind relativ leicht machbar. Einstellungen, die das komplette Bootverhalten beeinflussen, wie beispielsweise Timeouts oder der Kernel, der standardmäßig gestartet werden sollte, werden in der Datei */etc/default/grub* vorgenommen.

In der von uns beschriebenen Ubuntu-Version 18.04 sind die folgenden Dateien im Verzeichnis */etc/grub.d* zu finden:

- **00_header**
  Mit diesem Skript werden die Standardeinstellungen aus der Datei */etc/default/grub* gesetzt.

- **05_debian_theme**
  Diese Datei sorgt für das Aussehen des Bootmenüs: Hier werden Farben und Hintergrundbild definiert.
- **10_linux**
  Dieses Skript nimmt alle installierten Kernel in das Bootmenü auf.
- **20_linux_xen**
  Hier werden besondere Einstellungen vorgenommen und spezielle Kernel für die Xen-Virtualisierung gesetzt.
- **30_os-prober**
  Dieses Skript sucht nach installierten (anderen) Betriebssystemen und nimmt sie in das Bootmenü auf.
- **30_uefi-firmware**
  Besondere Einstellungen für *UEFI-Systeme* werden mit diesem Skript getroffen.
- **40_custom**
  Diese Datei ist für eigene Booteinträge vorhanden.
- **41_custom**
  Hiermit wird die */boot/grub/custom.cfg* eingebunden, sofern sie existiert.
- **README**
  Diese Datei enthält Hintergrundinformationen für die Skripte in diesem Verzeichnis.

Die Skriptnummern, die mit 00, 10 oder 20 beginnen, sind reserviert. Alle Nummern dazwischen können Sie für eigene Skripte verwenden. Je nachdem, welche Nummer Sie Ihrem Skript geben, wird es früher oder später im Prozess ausgeführt. Apropos »ausgeführt«: Die Skripte unterhalb von */etc/grub.d* müssen alle ausführbar sein.

Wir legen jetzt einen neuen Eintrag im Bootmenü an. Dazu werden am Ende der Datei *40_custom* die Zeilen aus Listing 2.2 neu eingefügt:

```
#!/bin/sh
exec tail -n +3 $0
# This file provides an easy way to add custom menu entries.  Simply type the
# menu entries you want to add after this comment.  Be careful not to change
# the 'exec tail' line above.

menuentry "Ubuntu 18.04.1 LTS, kernel 4.15.0-34-Adminbuch" {
  set root='(hd0,1)'
  linux /vmlinuz-4.15.0-34-generic \
root=/dev/mapper/ubuntu--vg-root ro console=hvc0
  initrd /initrd.img-4.15.0-34-generic
}
```

Listing 2.2 Eigener Eintrag in der Datei »40_custom«

Das Skript sorgt nur dafür, dass die Zeilen ab der dritten Zeile ausgegeben werden. Die eigentliche Konfiguration findet sich in der geschweiften Klammer nach dem `menuentry`, der den Text des Eintrags im Bootmenü enthält.

Wie gewohnt kennzeichnet `set root` die Partition, in der sich das Verzeichnis */boot* befindet. Natürlich bietet GRUB 2 eine Besonderheit: Die Festplattennummerierung beginnt bei 0, und die Nummerierung der Partition beginnt bei 1. So wird aus der Partition */dev/sdb3* unter GRUB `hd1,2` und unter GRUB 2 `hd1,3`.

Nach `linux` (früher `kernel`) folgt der zu startende Betriebssystemkern. Und `initrd` ist so, wie bereits beschrieben, die Initial Ramdisk. Mithilfe von `update-grub` wird ein neuer Bootloader geschrieben, und beim nächsten Start finden wir unseren neuen Eintrag im Bootmenü.

[+] Wie bereits beschrieben, ist GRUB 2 modular aufgebaut und bringt keine Treiber mit, daher muss man eventuell noch Module mit dem Kommando `insmod` hinzuladen, um aus einem einfachen Menüeintrag ein startfähiges System zu machen.

Beispiele dafür sind LVM, besondere Dateisysteme oder auch RAID. Alle verfügbaren Module Ihrer GRUB-2-Installation finden sich im Verzeichnis */boot/grub/i386-pc* oder */boot/grub2/i386-pc* und enden auf *.mod*. In Listing 2.3 finden Sie die Module eines Debian-Stretch-Systems:

```
root@debian:~# ls /boot/grub/i386-pc
915resolution.mod      gcry_whirlpool.mod        password_pbkdf2.mod
acpi.mod               gdb.mod                   pata.mod
adler32.mod            geli.mod                  pbkdf2.mod
affs.mod               gettext.mod               pbkdf2_test.mod
afs.mod                gfxmenu.mod               pci.mod
ahci.mod               gfxterm.mod               pcidump.mod
all_video.mod          gfxterm_background.mod    plan9.mod
aout.mod               gfxterm_menu.mod          play.mod
archelp.mod            gptsync.mod               png.mod
at_keyboard.mod        gzio.mod                  priority_queue.mod
ata.mod                halt.mod                  probe.mod
backtrace.mod          hashsum.mod               procfs.mod
bfs.mod                hdparm.mod                progress.mod
biosdisk.mod           hello.mod                 pxe.mod
bitmap.mod             help.mod                  pxechain.mod
bitmap_scale.mod       hexdump.mod               raid5rec.mod
blocklist.mod          hfs.mod                   raid6rec.mod
boot.img               hfsplus.mod               read.mod
boot.mod               hfspluscomp.mod           reboot.mod
bsd.mod                http.mod                  regexp.mod
btrfs.mod              hwmatch.mod               reiserfs.mod
bufio.mod              iorw.mod                  relocator.mod
```



```
cat.mod                iso9660.mod               romfs.mod
cbfs.mod               jfs.mod                   scsi.mod
cbls.mod               jpeg.mod                  search.mod
cbmemc.mod             keylayouts.mod            search_fs_file.mod
cbtable.mod            keystatus.mod             search_fs_uuid.mod
cbtime.mod             ldm.mod                   search_label.mod
chain.mod              legacy_password_test.mod  sendkey.mod
cmdline_cat_test.mod   legacycfg.mod             serial.mod
cmosdump.mod           linux.mod                 setjmp.mod
cmostest.mod           linux16.mod               setjmp_test.mod
cmp.mod                loadenv.mod               setpci.mod
command.lst            loopback.mod              sfs.mod
configfile.mod         ls.mod                    signature_test.mod
core.img               lsacpi.mod                sleep.mod
cpio.mod               lsapm.mod                 sleep_test.mod
cpio_be.mod            lsmmap.mod                spkmodem.mod
cpuid.mod              lspci.mod                 squash4.mod
crc64.mod              luks.mod                  syslinuxcfg.mod
crypto.lst             lvm.mod                   tar.mod
crypto.mod             lzopio.mod                terminal.lst
cryptodisk.mod         macbless.mod              terminal.mod
cs5536.mod             macho.mod                 terminfo.mod
date.mod               mda_text.mod              test.mod
datehook.mod           mdraid09.mod              test_blockarg.mod
datetime.mod           mdraid09_be.mod           testload.mod
disk.mod               mdraid1x.mod              testspeed.mod
diskfilter.mod         memdisk.mod               tftp.mod
div_test.mod           memrw.mod                 tga.mod
dm_nv.mod              minicmd.mod               time.mod
drivemap.mod           minix.mod                 tr.mod
echo.mod               minix2.mod                trig.mod
efiemu.mod             minix2_be.mod             true.mod
efiemu32.o             minix3.mod                truecrypt.mod
efiemu64.o             minix3_be.mod             udf.mod
ehci.mod               minix_be.mod              ufs1.mod
elf.mod                mmap.mod                  ufs1_be.mod
eval.mod               moddep.lst                ufs2.mod
exfat.mod              modinfo.sh                uhci.mod
exfctest.mod           morse.mod                 usb.mod
ext2.mod               mpi.mod                   usb_keyboard.mod
extcmd.mod             msdospart.mod             usbms.mod
fat.mod                multiboot.mod             usbserial_common.mod
file.mod               multiboot2.mod            usbserial_ftdi.mod
```

```
font.mod              nativedisk.mod        usbserial_pl2303.mod
freedos.mod           net.mod               usbserial_usbdebug.mod
fs.lst                newc.mod              usbtest.mod
fshelp.mod            nilfs2.mod            vbe.mod
functional_test.mod   normal.mod            verify.mod
gcry_arcfour.mod      ntfs.mod              vga.mod
gcry_blowfish.mod     ntfscomp.mod          vga_text.mod
gcry_camellia.mod     ntldr.mod             video.lst
gcry_cast5.mod        odc.mod               video.mod
gcry_crc.mod          offsetio.mod          video_bochs.mod
gcry_des.mod          ohci.mod              video_cirrus.mod
gcry_dsa.mod          part_acorn.mod        video_colors.mod
gcry_idea.mod         part_amiga.mod        video_fb.mod
gcry_md4.mod          part_apple.mod        videoinfo.mod
gcry_md5.mod          part_bsd.mod          videotest.mod
gcry_rfc2268.mod      part_dfly.mod         videotest_checksum.mod
gcry_rijndael.mod     part_dvh.mod          xfs.mod
gcry_rmd160.mod       part_gpt.mod          xnu.mod
gcry_rsa.mod          part_msdos.mod        xnu_uuid.mod
gcry_seed.mod         part_plan.mod         xnu_uuid_test.mod
gcry_serpent.mod      part_sun.mod          xzio.mod
gcry_sha1.mod         part_sunpc.mod        zfs.mod
gcry_sha256.mod       partmap.lst           zfscrypt.mod
gcry_sha512.mod       parttool.lst          zfsinfo.mod
gcry_tiger.mod        parttool.mod
gcry_twofish.mod      password.mod
```

Listing 2.3 GRUB-2-Module eines Debian-Stretch-Systems

Auf dem gleichen System findet sich in der */boot/grub/grub.cfg* ein Beispiel dafür, wie ein Teil dieser Module eingesetzt wird (siehe Listing 2.4):

```
[…]
menuentry 'Debian GNU/Linux, with Linux 3.16.0-4-amd64' --class debian \
--class gnu-linux --class gnu --class os $menuentry_id_option \
'gnulinux-3.16.0-4-amd64-advanced-eac6da17-314e-43c0-956f-379457a505fa' {
    load_video
    insmod gzio
    if [ x$grub_platform = xxen ]; then insmod xzio; insmod lzopio; fi
    insmod part_msdos
    insmod ext2
    set root='hd0,msdos1'
    if [ x$feature_platform_search_hint = xy ]; then
        search --no-floppy --fs-uuid --set=root --hint-bios=hd0,msdos1 \
```



```
        --hint-efi=hd0,msdos1 --hint-baremetal=ahci0,msdos1 \
        eac6da17-314e-43c0-956f-379457a505fa
    else
        search --no-floppy --fs-uuid --set=root eac6da17-314e-43c0-956f-379457a505fa
    fi
    echo    'Loading Linux 3.16.0-4-amd64 ...'
    linux   /boot/vmlinuz-3.16.0-4-amd64 root=UUID=eac6da17-314e-43c0-956f-\
            379457a505fa ro quiet
    echo    'Loading initial ramdisk ...'
    initrd  /boot/initrd.img-3.16.0-4-amd64
}
[…]
```

Listing 2.4 Die Optionen des Standardkernels aus der »/boot/grub/grub.cfg«

Änderungen in der Datei */boot/grub/grub.cfg* werden nicht automatisch übernommen. Mit dem Kommando `update-grub` wird GRUB 2 aktualisiert, wie in Listing 2.5 zu sehen ist:

```
root@debian:~# update-grub
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-3.16.0-4-amd64
Found initrd image: /boot/initrd.img-3.16.0-4-amd64
done
```

Listing 2.5 »update-grub«

Interessant ist, dass die Konfigurationsdatei */etc/default/grub* ein Shell-Skript ist. Allerdings werden dort nur Variablen gesetzt, die nach dem Aufruf von `update-grub` durch */etc/grub.d/00_header* ausgewertet werden. In der folgenden Auflistung finden Sie die wichtigsten Variablen:

- **`GRUB_DEFAULT=0`**
  Hiermit wird der Standardeintrag gesetzt.
- **`GRUB_TIMEOUT=5`**
  Nach Ablauf der durch `TIMEOUT` gesetzten Zeit wird der Standardeintrag gestartet.
- **`GRUB_HIDDEN_TIMEOUT=0`**
  Wenn nur ein Betriebssystem existiert, wird dieser Wert als Wartezeit benutzt. Sobald ein weiterer Eintrag hinzukommt, ist der Wert bedeutungslos.
- **`GRUB_HIDDEN_TIMEOUT_QUIET=true`**
  Mit `true` wird kein Countdown angezeigt, bei `false` wird er entsprechend angezeigt.
- **`GRUB_CMDLINE_LINUX=`**
  Hiermit werden Standardoptionen für jede `linux`-Zeile gesetzt.

Die Variablen werden erst nach einem erneuten Aufruf von `update-grub` gültig.

2.3 Bootloader Recovery

Es passiert selten, aber wenn Sie Ihr System aufgrund einer Fehlkonfiguration des Bootloaders nicht mehr starten können, sollten Sie den Bootloader reparieren. Dazu können Sie den Rechner von einer beliebigen Live-CD[1] oder DVD oder von einem USB-Stick neu starten.

[+] Der einfachste Weg, eine Reparatur durchzuführen, ist, die Live-CD des Systems zu verwenden, mit der Sie den Rechner installiert haben. Beachten Sie jedoch, dass Sie in jedem Fall bei Benutzung einer anderen Rettungs-CD dieselbe Architektur verwenden, die auch Ihr installiertes System aufweist.

Nach dem Start des Rettungssystems wird die Festplatte Ihres defekten Systems eingebunden. Das bedeutet, dass Sie alle Partitionen *mounten*. Im Regelfall werden die Partitionen unter */mnt* eingebunden. Sie können natürlich auch eigene Verzeichnisse verwenden, wenn Sie dabei keines der vom Live-System benutzten Verzeichnisse einsetzen.

Das Kommando `fdisk -l` zeigt Ihnen alle gefundenen Festplatten an. Falls Software-RAIDs oder LVM benutzt werden, müssen diese vor der Benutzung aktiviert werden. Wie das geht, erklären wir in Kapitel 3, »Festplatten und andere Devices«.

```
   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *        2048      499711      248832   83  Linux
/dev/sda2          501758    20969471    10233857    5  Extended
/dev/sda5          501760    20969471    10233856   8e  Linux LVM
```

Listing 2.6 Ausgabe »fdisk -l« auf einem Testsystem

In Listing 2.6 finden Sie den seltenen Fall eines Systems ohne eigene *Swap*-Partition. Vermutlich findet sich in der ersten Partition das *boot*-Verzeichnis, was wir durch Mounten verifizieren können (siehe Listing 2.7):

```
Rescue:~# mount /dev/sda1 /mnt
Rescue:~# ls /mnt
System.map-3.16.0-4-amd64  grub                      vmlinuz-3.16.0-4-amd64
config-3.16.0-4-amd64      initrd.img-3.16.0-4-amd64
Rescue:~# umount /mnt
```

Listing 2.7 Mounten des vermeintlichen »boot«-Filesystems

Die fünfte Partition wird vom *Logical Volume Manager* (LVM) verwaltet.

Mit dem Kommando `lvs` können wir uns die gefundenen *Logical Volumes* anzeigen lassen. Der Parameter `-o +lv_path` sorgt dafür, dass uns auch gleich der Pfad für das Mounten gezeigt wird (siehe Listing 2.8):

1 Zum Beispiel die »System Rescue CD«, *www.sysresccd.org*


```
Rescue:~# lvs -o +lv_path
  LV     VG     Attr     LSize   Pool Origin Data%  Move Log Copy%  Convert \
Path
  root   debian -wi-ao--   9.31g                                             \
/dev/debian/root
  swap_1 debian -wi-ao-- 460.00m                                             \
/dev/debian/swap_1
```

Listing 2.8 Gefundene Partition im LVM

An dieser Stelle haben wir alle Informationen zusammen, um die Dateisysteme benutzen zu können (siehe Listing 2.9):

```
Rescue:~ # mount /dev/debian/root /mnt
Rescue:~ # mount /dev/sda1 /mnt/boot/
```

Listing 2.9 Mounten der Dateisysteme

Um das Linux-System komplett zu machen, müssen wir die dynamischen Pseudo-Dateisysteme (*/dev*, */proc* und */sys*) aus der Live-CD in die Verzeichnisse unterhalb von */mnt* einbinden. Das funktioniert über *Bind-Mounts*. Wenn das nicht passieren würde, erhielten wir nach einem Wechsel der root-Umgebung mittels `chroot` (»change root environment«) keine Informationen über verbundene Geräte und Kernelparameter (siehe Listing 2.10):

```
Rescue:~ # mount --bind /dev  /mnt/dev
Rescue:~ # mount --bind /proc /mnt/proc
Rescue:~ # mount --bind /sys  /mnt/sys
```

Listing 2.10 Bind-Mount der Pseudodateisysteme

Damit sind jetzt alle Vorarbeiten abgeschlossen, um via `chroot` auf das System zu wechseln und den Bootloader zu reparieren (siehe Listing 2.11):

```
Rescue:~ # chroot /mnt
Rescue:/ # grub-install
Rescue:/ # exit
```

Listing 2.11 Neuinstallation des Bootloaders

Sobald Sie fertig sind, müssen alle Dateisysteme ausgehängt werden. Anschließend müssen Sie das System neu starten.

2.4 Der Kernel und die »initrd«

Beim Laden des Kernels gibt es ein klassisches Henne-Ei-Problem: Der Kernel probiert nämlich zunächst, alle notwendigen Module zu laden, die für den Zugriff auf die Hardware not-

wendig sind. Das sind insbesondere die Treiber zum Ansprechen der Festplatte und des Dateisystems. Die dafür notwendigen Module liegen aber auf dem noch nicht lesbaren Dateisystem.

Um dieses Dilemma zu lösen, lädt der Bootloader nicht nur den Kernel direkt in den Speicher, sondern auch die *Initial Ramdisk* (initrd). Die initrd besteht aus einem komprimierten *cpio*-Archiv und enthält ein absolut minimales Linux mit allen für den Start notwendigen Modulen. Der Kernel benutzt die initrd als *root*-Filesystem. Sobald alle nötigen Treiber geladen sind, bindet der Kernel das eigentliche *root*-Filesystem ein und startet den *systemd*-Prozess.

2.4.1 »initrd« erstellen und modifizieren

Bei der Installation eines Systems wird auch eine *Initial Ramdisk* (initrd) erstellt, die Treiber enthält, die für den Start des Rechners benötigt werden, bevor die Dateisysteme verfügbar sind. Diese Ramdisk wird bei jedem Kernelupdate neu erstellt und mit neuen Versionen der Treiber versehen.

Wenn Sie allerdings Hardware benutzen, die Treiber benötigt, die nicht im Kernel vorhanden sind, wie beispielsweise besondere *RAID*-Controller oder Netzwerkkarten, so müssen Sie – wenn Sie Ihr System von den Geräten aus starten wollen – selbst Hand anlegen, wenn das nicht die Installationsroutine des Herstellers für Sie übernimmt. Die verschiedenen Distributionen nutzen unterschiedliche Tools für die Erstellung.

In den folgenden Abschnitten finden Sie die Beschreibungen für die im Buch unterstützten Distributionen, gefolgt von einem Abschnitt über die komplett manuelle Erstellung der Initial Ramdisk.

Debian und Ubuntu

Debian und Ubuntu benutzen `mkinitramfs` und `update-initramfs`. Wenn Sie nicht besondere Gründe haben, sollte Sie immer `update-initramfs` verwenden, da dieses Kommando unter anderem auch `mkinitramfs` auf Basis der bereits bestehenden Konfiguration aufruft.

Die Erstellung der initrd wird über die Konfigurationsdatei */etc/initramfs-tools/initramfs.conf* und weitere Dateien innerhalb des Verzeichnisses */etc/initramfs-tools* gesteuert. Aufgrund der vielen Kommentare in den Dateien werden Sie schnell zum Ziel kommen.

Einen besonderen Blick verdient die wichtigste Variable, `MODULES`. Sie kann verschiedene Werte annehmen, wie folgende Auflistung zeigt:

- **`most`**
  Das ist die Standardeinstellung bei Ubuntu und Debian. Damit werden fast alle Dateisystem- und Hardwaretreiber übernommen. Die daraus resultierende sehr große Initial Ramdisk kann dafür aber auch fast jedes System starten.

- **`dep`**
  Das laufende System wird analysiert, um festzustellen, welche Module wichtig sind. Diese Einstellung verkleinert die Initial Ramdisk auf ein Minimum.
- **`netboot`**
  Wie der Name es beschreibt, werden mit dieser Einstellung nur Treiber verwendet, die für das Starten über das Netzwerk nötig sind.
- **`list`**
  Ausschließlich Module aus */etc/initramfs-tools/modules* werden zum Bau der Initial Ramdisk verwendet. Dies erlaubt die größtmögliche Kontrolle.

Auch ohne weitere Konfiguration werden die Module aus */etc/initramfs-tools/modules* bei den Parametern `most`, `dep` und `netboot` zur Initial Ramdisk hinzugefügt. [+]

Die Konfigurationen in den Dateien unterhalb von */etc/initramfs-tools/conf.d* können die Werte aus */etc/initramfs-tools/initramfs.conf* überschreiben. [!]

Um eine neue Initial Ramdisk zu erstellen bzw. die bestehende aktualisieren zu lassen, können Sie mit `update-initramfs` den Neubau starten. Die unten stehenden Parameter helfen bei der Erstellung:

- **`update-initramfs -u`**
  Hiermit werden alle vorhandenen Initial Ramdisks aktualisiert.
- **`update-initramfs -k KERNEL`**
  Dieser Parameter wird benötigt, wenn nur die Initial Ramdisks einer bestimmten Kernelversion aktualisiert werden sollen.
- **`update-initramfs -c`**
  Dieser Parameter erstellt komplett neue Initial Ramdisks.

Der Name der Initial Ramdisk ergibt sich aus dem Namen des Kernels. Eine vorhandene Ramdisk wird somit bei jedem Aufruf von `update-initramfs` überschrieben.

Wenn Sie dieses Verhalten nicht wünschen, sollten Sie den Parameter `backup_initramfs=yes` in der Datei */etc/initramfs-tools/update-initramfs.conf* setzen oder manuelle Backups erstellen (siehe Listing 2.12):

```
root@debian:~# update-initramfs -v -k 3.16.0-4-amd64 -c
update-initramfs: Generating /boot/initrd.img-3.16.0-4-amd64
Copying module directory kernel/drivers/hid
(excluding hid-*ff.ko hid-a4tech.ko hid-cypress.ko hid-dr.ko hid-elecom.ko \
hid-gyration.ko hid-icade.ko hid-kensington.ko hid-kye.ko hid-lcpower.ko   \
hid-magicmouse.ko hid-multitouch.ko hid-ntrig.ko hid-petalynx.ko           \
hid-picolcd.ko hid-pl.ko hid-ps3remote.ko hid-quanta.ko hid-roccat-ko*.ko  \
hid-roccat-pyra.ko hid-saitek.ko hid-sensor-hub.ko hid-sony.ko             \
hid-speedlink.ko hid-tivo.ko hid-twinhan.ko hid-uclogic.ko hid-wacom.ko    \
```

```
hid-waltop.ko hid-wiimote.ko hid-zydacron.ko)
Adding module /lib/modules/3.16.0-4-amd64/kernel/drivers/hid/hid.ko
[…]
Adding library /lib/x86_64-linux-gnu/librt.so.1
Adding module /lib/modules/3.16.0-4-amd64/kernel/drivers/md/dm-mod.ko
/usr/share/initramfs-tools/scripts/local-premount/ORDER ignored: not executable
/usr/share/initramfs-tools/scripts/init-top/ORDER ignored: not executable
/usr/share/initramfs-tools/scripts/init-bottom/ORDER ignored: not executable
Building cpio /boot/initrd.img-3.16.0-4-amd64.new initramfs
```

Listing 2.12 Neuerstellen einer »initrd«

[!] Wenn der Name der Initial Ramdisk bereits existierte, ist nichts weiter zu tun. Sollten Sie aber einen neuen Namen verwenden, muss im Bootloader der entsprechende Name eingetragen werden, sonst können Sie das System nicht mehr starten.

CentOS und openSUSE

Anders als bei Ubuntu und Debian nutzen CentOS und openSUSE das Skript `mkinitrd`, um eine Initial Ramdisk zu erstellen. Das Skript ermittelt die Treiber, die aufgenommen werden müssen, und nutzt die Informationen aus */etc/sysconfig/kernel*, in der eine Liste von Modulen zu finden ist, die zusätzlich hinzugefügt werden sollen (siehe Listing 2.13):

```
Creating initrd: /boot/initrd-4.1.27-27-default
Executing: /usr/bin/dracut --logfile /var/log/YaST2/mkinitrd.log \
--force /boot/initrd-4.1.27-27-default 4.1.27-27-default
...
*** Including module: bash ***
*** Including module: warpclock ***
*** Including module: i18n ***
*** Including module: ifcfg ***
*** Including module: btrfs ***
*** Including module: kernel-modules ***
Omitting driver i2o_scsi
*** Including module: resume ***
*** Including module: rootfs-block ***
*** Including module: terminfo ***
*** Including module: udev-rules ***
Skipping udev rule: 91-permissions.rules
Skipping udev rule: 80-drivers-modprobe.rules
*** Including module: haveged ***
*** Including module: systemd ***
*** Including module: usrmount ***
*** Including module: base ***
*** Including module: fs-lib ***
```



```
*** Including module: shutdown ***
*** Including module: suse ***
*** Including modules done ***
*** Installing kernel module dependencies and firmware ***
*** Installing kernel module dependencies and firmware done ***
*** Resolving executable dependencies ***
*** Resolving executable dependencies done***
*** Hardlinking files ***
*** Hardlinking files done ***
*** Stripping files ***
*** Stripping files done ***
*** Generating early-microcode cpio image ***
*** Store current command line parameters ***
Stored kernel commandline:
 resume=UUID=54c1a2e0-c4d6-4850-b639-7a5af8ef4339
root=UUID=0f4f79aa-7544-44b0-a8b7-d1f1947cd24f \
rootflags=rw,relatime,space_cache,subvolid=257,subvol=/@ rootfstype=btrfs
*** Creating image file ***
*** Creating image file done ***
Some kernel modules could not be included
This is not necessarily an error:
*** Including module: udev-rules ***
Skipping udev rule: 91-permissions.rules
Skipping udev rule: 80-drivers-modprobe.rules
*** Including module: haveged ***
*** Including module: systemd ***
*** Including module: usrmount ***
*** Including module: base ***
*** Including module: fs-lib ***
*** Including module: shutdown ***
*** Including module: suse ***
*** Including modules done ***
*** Installing kernel module dependencies and firmware ***
*** Installing kernel module dependencies and firmware done ***
*** Resolving executable dependencies ***
*** Resolving executable dependencies done***
*** Hardlinking files ***
*** Hardlinking files done ***
*** Stripping files ***
*** Stripping files done ***
*** Generating early-microcode cpio image ***
*** Store current command line parameters ***
Stored kernel commandline:
```

```
 resume=UUID=54c1a2e0-c4d6-4850-b639-7a5af8ef4339
root=UUID=0f4f79aa-7544-44b0-a8b7-d1f1947cd24f \
rootflags=rw,relatime,space_cache,subvolid=257,subvol=/@ rootfstype=btrfs
*** Creating image file ***
*** Creating image file done ***
Some kernel modules could not be included
This is not necessarily an error:
```

Listing 2.13 Beispiel »mkinitrd« auf openSUSE

Die Installation des Systems setzt automatisch die Variable `INITRD_MODULES`. Wenn diese Liste um eigene Einträge ergänzt wird, muss anschließend `mkinitrd` aufgerufen werden.

Analog zu `update-initramfs` bei Ubuntu und Debian bietet auch `mkinitrd` einige Optionen an, die Ihnen helfen, die Initial Ramdisk anzupassen:

- **`-k KERNEL`**
  Angabe des Kernels, für den die Initial Ramdisk gebaut werden soll. Ohne Angabe des Parameters wird *vmlinuz* benutzt.
- **`-i INITRD`**
  setzt den Namen der Initial Ramdisk. Ohne diese Angabe wird */boot/initrd* genommen.
- **`-m MODULES`**
  nimmt eine Liste von Modulen auf der Kommandozeile, ansonsten wird der Inhalt der Variablen `INITRD_MODULES` aus `/etc/sysconfig/kernel` ausgelesen.
- **`-f FEATURES`**
  setzt Funktionalitäten für den Kernel, abhängig davon werden weitere Module und Skripte eingebunden. Als Beispiel seien hier *Software-RAID* (Parameter `dm`) und *Logical Volume Manager* (Parameter `lvm2`) genannt.

In Listing 2.14 sehen Sie einen Beispielaufruf von `mkinitrd`:

```
opensuse:~ # mkinitrd -k 4.1.27-27-default -i initrdtest -m ext4 \
-f "lvm2 dm block"
```

Listing 2.14 Beispielaufruf von »mkinitrd«

2.4.2 »initrd« manuell modifizieren

Zusätzlich zu den vorgestellten Methoden, die zugegebenermaßen relativ beschränkt sind, lässt sich die Initial Ramdisk (initrd) auch manuell verändern.

Als Basis für Ihre Arbeiten nehmen Sie sich bitte eine vorhandene *initrd* und packen diese aus. Listing 2.15 zeigt Ihnen, dass es sich bei der initrd um ein minimales root-Filesystem handelt:


```
root@debian:~# mkdir /var/tmp/initrd
root@debian:~# cd /var/tmp/initrd/
root@debian:/var/tmp/initrd# gzip -dc /boot/initrd.img-3.16.0-4-amd64 \
| cpio --extract --make-directories
90084 blocks
root@debian:/var/tmp/initrd# ls -l
total 40
drwxr-xr-x 2 root root 4096 Aug  4 15:31 bin
drwxr-xr-x 3 root root 4096 Aug  4 15:31 conf
drwxr-xr-x 5 root root 4096 Aug  4 15:31 etc
-rwxr-xr-x 1 root root 7137 Aug  4 15:31 init
drwxr-xr-x 7 root root 4096 Aug  4 15:31 lib
drwxr-xr-x 2 root root 4096 Aug  4 15:31 lib64
drwxr-xr-x 2 root root 4096 Aug  4 15:31 run
drwxr-xr-x 2 root root 4096 Aug  4 15:31 sbin
drwxr-xr-x 5 root root 4096 Aug  4 15:31 scripts
```

Listing 2.15 »initrd« entpacken

In dem resultierenden Verzeichnis */var/tmp/initrd* können Sie nun Ihre Änderungen einpflegen und danach alles wieder einpacken (siehe Listing 2.16):

```
root@debian:/var/tmp/initrd# find . \
| cpio --create --format=newc \
| gzip > /boot/initrd.adminbuch
90084 blocks
```

Listing 2.16 »initrd« einpacken

In der Datei */boot/initrd.adminbuch* findet sich nun die initrd, die alle Ihre Änderungen enthält.

2.5 »systemd«

Nach dem Bootvorgang, in dem der Kernel das root-Filesystem eingebunden und alle notwendigen Module geladen hat, übernimmt der *systemd*-Daemon den weiteren Ablauf.

Der klassische *init*-Prozess folgt dem in System V[2] vorgestellten Verfahren und wird nach diesem auch *SysVinit* genannt. Er ist verantwortlich für das Starten der Dienste in der richtigen Reihenfolge, das Folgen und auch den Wechsel von Runleveln sowie für das Stoppen von Prozessen. Dieses Verfahren ist sehr robust, aber leider auch sehr statisch.

2 *https://de.wikipedia.org/wiki/System_V*

systemd ist der Nachfolger, den mittlerweile alle Distributionen verwenden. Mit *systemd* gibt es einen Übergang vom statischen Starten von Skripten zum eventbasierten Starten. So können Bedingungen definiert werden, die erfüllt sein müssen, um Dienste starten zu können (beispielsweise wird der Webserver erst dann gestartet, wenn das Netzwerk verfügbar ist, oder ein Virenscanner erst dann, wenn ein USB-Stick eingesteckt wird). Der Start von Diensten mit *systemd* ist im Unterschied zu *SysVinit* hoch parallelisierbar. Als besonderes Feature ist *systemd* auch in der Lage, abgestürzte Dienste neu zu starten.

Es gibt kaum ein Thema in den letzten Jahren, das in der Linux-Community so kontrovers diskutiert wurde wie die Einführung von *systemd*.

systemd schickt sich an, den kompletten altbekannten und bewährten Bootvorgang auf den Kopf zu stellen. Den einen gehen die Änderungen zu weit, die anderen feiern mit *systemd* die Ankunft im neuen Jahrtausend. Tatsache ist, dass mit *systemd* Start-Skripte – genauer gesagt Startkonfigurationen – parallel ausgeführt werden können und nicht wie früher linear. Dazu kommt, dass *systemd* Programme voneinander kapselt und in eigenen *Control Groups* und *Namespaces* startet und so sicherstellt, dass beim Beenden eines Dienstes auch alle Prozesse im gleichen Namespace mit beendet werden und dass es keine verwaisten Prozesse gibt.

Weitergehende Änderungen sind, dass mit *journald* ein eigenes Logging-Framework mitgeliefert wird, das es erlaubt, fälschungssichere Logs zu führen. Dadurch soll das altbekannte *syslog* abgelöst werden. *timers* in *systemd* sind in der Lage, klassische Cron- und Anacron-Jobs abzulösen, *systemd-mounts* könnten die *fstab* überflüssig machen.

Die beiden Hauptkritikpunkte der *systemd*-Gegner sind, dass *systemd* von der UNIX-Philosophie »one task, one tool« abweicht und dass das systemd-Team bei der Weiterentwicklung zum Teil fragwürdige Entscheidungen trifft.

2.5.1 Begrifflichkeiten

systemd wird mit *Units* verwaltet. Units kapseln verschiedene Aspekte eines Systems und können untereinander in Beziehung gesetzt werden. Die Definitionen der Units sind einfache Textdateien, die ein wenig an ini-Dateien aus Windows erinnern. Die einzelnen Unit-Typen sind die folgenden:

- **Service Units**
  werden benutzt, um Dienste und Prozesse zu starten.
- **Socket Units**
  kapseln IPC- oder Netzwerk-Sockets, die vor allem gebraucht werden, um Socket-basierend Dienste zu aktivieren.
- **Target Units**
  können zur Gruppierung von Diensten oder zur Erstellung von Synchronisationspunkten benutzt werden (hiermit lassen sich Runlevel wie im SysVinit emulieren).

- **Device Units**
  sind die Verbindung zu Kernel-Devices und können ebenfalls benutzt werden, um Device-basierende Dienste zu steuern.
- **Mount Units**
  kontrollieren Mountpunkte im System.
- **Automount Units**
  werden für zugriffsbasiertes Einbinden von Dateisystemen benutzt und dienen insbesondere auch der Parallelisierung im Bootprozess.
- **Snapshot Units**
  können den Status einer Anzahl von *systemd*-Units aufzeichnen und diesen Status durch Aktivierung auch wiederherstellen.
- **Timer Units**
  bieten die Möglichkeit, eine zeitbasierte Steuerung anderer Units vorzunehmen.
- **Swap Units**
  verwalten – analog zu Mount Units – Swap-Speicherplatz.
- **Path Units**
  werden benutzt, um andere Dienste bei einer Veränderung von Dateisystemobjekten zu aktivieren.
- **Slice Units**
  gruppieren Units in hierarchischer Form, die Systemprozesse – beispielsweise Service oder Scope Units – verwalten.
- **Scope Units**
  gleichen Service Units, verwalten aber Fremdprozesse, anstatt sie nur zu starten.

Wie Sie allein an den verschiedenartigen Units feststellen können, kann man in *systemd* vielfältige Aspekte eines Systems beeinflussen. Im Folgenden gehen wir auf System Units ein, das sind die Units, mit denen Sie am häufigsten in Kontakt kommen werden.

2.5.2 Kontrollieren von Diensten

Das Hauptkommando, mit dem Sie *systemd* kontrollieren können, heißt `systemctl`. Dieser Befehl wird auch benutzt, um Dienste zu verwalten. Analog zu den früheren init-Skripten gibt es die Kommandos `start`, `stop`, `reload`, `restart` und `status`.

systemctl macht Gebrauch von Farben im Terminal. Stellen Sie daher bitte sicher, dass Ihr Terminal auch Farben darstellen kann. [+]

Service Units in *systemd* enden auf *.service*, diese Endung muss aber nicht explizit angegeben werden. In Listing 2.17 sehen Sie, dass weder das Stopp- noch das Start-Subkommando sehr gesprächig ist, daher sollte das Ergebnis mit einer Statusabfrage überprüft werden.

```
# systemctl stop sshd
# systemctl status sshd
*  sshd.service - OpenSSH server daemon
   Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled; vendor preset: \
                  enabled)
   Active: inactive (dead) since Fre 2018-08-05 13:26:32 CEST; 5s ago
     Docs: man:sshd(8)
           man:sshd_config(5)
  Process: 2054 ExecStart=/usr/sbin/sshd -D $OPTIONS (code=exited, status=0/SUCCESS)
 Main PID: 2054 (code=exited, status=0/SUCCESS)

Aug 05 13:25:49 centos sshd[2054]: Server listening on 0.0.0.0 port 22.
Aug 05 13:25:49 centos sshd[2054]: Server listening on :: port 22.
Aug 05 13:25:49 centos systemd[1]: Started OpenSSH server daemon.
Aug 05 13:25:49 centos systemd[1]: Starting OpenSSH server daemon...
Aug 05 13:26:32 centos sshd[2054]: Received signal 15; terminating.
Aug 05 13:26:32 centos systemd[1]: Stopping OpenSSH server daemon...
Aug 05 13:26:32 centos systemd[1]: Stopped OpenSSH server daemon.

# systemctl start sshd
# systemctl status sshd
* sshd.service - OpenSSH server daemon
   Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled; vendor preset: \
                  enabled)
   Active: active (running) since Fre 2018-08-05 13:26:43 CEST; 5s ago
     Docs: man:sshd(8)
           man:sshd_config(5)
 Main PID: 2072 (sshd)
 CGroup: /system.slice/sshd.service
         `-2072 /usr/sbin/sshd -D

Aug 05 13:26:43 centos sshd[2072]: Server listening on 0.0.0.0 port 22.
Aug 05 13:26:43 centos sshd[2072]: Server listening on :: port 22.
Aug 05 13:26:43 centos systemd[1]: Started OpenSSH server daemon.
Aug 05 13:26:43 centos systemd[1]: Starting OpenSSH server daemon...
```

Listing 2.17 Stoppen des SSH-Servers

Gerade die Statusausgaben sind auf den ersten Blick sehr verwirrend:

- In der ersten Zeile finden Sie den Namen des Dienstes und die Beschreibung.
- Die Zeile, die mit `Loaded:` beginnt, zeigt Ihnen, ob die Unit-Datei geladen ist, und den Speicherort. Mit `enabled` ist gemeint, dass die Unit standardmäßig (beispielsweise beim Start des Systems) ausgeführt wird und wie die Einstellung des Distributors (CentOS, Debian, openSUSE oder Ubuntu) ist. Mehr Informationen dazu finden Sie in Abschnitt 2.5.3, »Aktivieren und Deaktivieren von Diensten«.
- `Active`: kennzeichnet den aktuellen Status und seit wann dieser Status besteht.
- `Docs`: verweist auf Dokumentationen zum Service, hier sind es Manpages, üblich ist aber auch der Hinweis auf eine URL.
- `Process`: zeigt Ihnen, wie der Dienst gestartet wurde, und den letzten Status.
- `Main PID`: enthält die Hauptprozess-ID.
- `CGroup`: stellt die Control Group, in der der Dienst gestartet wurde dar.
- Zum Schluss folgt ein Auszug der letzten Log-Ausgaben. Wie Sie mehr Log-Ausgaben sehen können, erfahren Sie in Abschnitt 2.5.8.

Die Subkommandos `restart`, um den Dienst neu zu starten, und `reload`, um die Konfiguration – in diesem Fall */etc/ssh/sshd_config* – neu einzulesen, vervollständigen die Basiskommandos.

Folgende Befehle wurden in diesem Abschnitt behandelt:

- `systemctl start <SERVICE>`
- `systemctl stop <SERVICE>`
- `systemctl status <SERVICE>`
- `systemctl restart <SERVICE>`
- `systemctl reload <SERVICE>`

2.5.3 Aktivieren und Deaktivieren von Diensten

Units werden mit den Subkommandos `enable` und `disable` aktiviert und deaktiviert:

```
# systemctl disable sshd
Removed symlink /etc/systemd/system/multi-user.target.wants/sshd.service.

# systemctl enable sshd
Created symlink from /etc/systemd/system/multi-user.target.wants/sshd.service \
to /usr/lib/systemd/system/sshd.service.
```

Listing 2.18 Aktivieren und Deaktivieren des SSH-Servers

Die Subkommandos lesen die Servicedefinition und schauen, für welches Target der Dienst aktiviert werden soll. In Listing 2.18 sehen Sie, dass es das *multi-user.target* für den Dienst `sshd` ist. Beim Aktivieren wird nun ein Link in dem Verzeichnis des Targets erstellt und beim Deaktivieren wieder gelöscht.

Folgende Befehle wurden in diesem Abschnitt behandelt:

- `systemctl enable <SERVICE>`
- `systemctl disable <SERVICE>`

2.5.4 Erstellen und Aktivieren eigener Service Units

Wie Sie bereits in Listing 2.17 im Abschnitt 2.5.2 gesehen haben, ist die Konfigurationsdatei */usr/lib/systemd/system/sshd.service* die Datei, in der die Angaben des SSH-Servers gespeichert werden. Den Inhalt sehen Sie im folgenden Listing. Viele Ausgaben des Statuskommandos werden durch Einträge im Servicefile angezeigt, beispielsweise die `Description` oder `Documentation`:

```
# cat /usr/lib/systemd/system/sshd.service
[Unit]
Description=OpenSSH server daemon
Documentation=man:sshd(8) man:sshd_config(5)
After=network.target sshd-keygen.service
Wants=sshd-keygen.service

[Service]
EnvironmentFile=/etc/sysconfig/sshd
ExecStart=/usr/sbin/sshd -D $OPTIONS
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure
RestartSec=42s

[Install]
WantedBy=multi-user.target
```

Listing 2.19 Inhalt von »sshd.service«

Die Ausgabe von `systemctl show sshd` zeigt Ihnen neben den Einträgen aus dem Servicefile auch noch die ganzen Standardoptionen und Statusinformationen an. Die 143 Zeilen Information werden Ihnen an dieser Stelle erspart, da Sie diese jederzeit selbst abrufen können. Die einzelnen Parameter der Konfiguration aus Listing 2.19 haben die folgende Bedeutung:

- **Description**
  enthält eine lesbare Beschreibung des Dienstes.
- **Documentation**
  verweist auf weiterführende Informationen.

- **After**
  wird benutzt, um Abhängigkeiten zu definieren. In diesem Fall soll der Dienst nach den angegebenen anderen Diensten gestartet werden (analog dazu gibt es `Before`).
- **Wants**
  erfordert, dass die angegebenen Dienste vor dem Start erfolgreich gelaufen sind (abgemilderte Form von `Require`).
- **EnvironmentFile**
  gibt eine Datei mit Umgebungsvariablen an, die zur Verfügung stehen sollen.
- **ExecStart**
  enthält das Kommando, das zum Start benutzt wird (analog dazu gibt es `ExecStop`).
- **ExecReload**
  Mit diesem Kommando werden die Konfigurationsdateien neu eingelesen.
- **KillMode**
  zeigt, mit welchem Verfahren der Prozess gekillt werden kann.
- **Restart**
  definiert die Option für den automatischen Neustart des Dienstes.
- **RestartSec**
  enthält die Zeit, nach der neu gestartet werden soll.
- **WantedBy**
  beschreibt das Target (oder den Service), durch das der Dienst automatisch gestartet werden soll.

Zusätzlich zu den Optionen, die Sie in der Definition des SSH-Servers sehen, gibt es noch die folgenden Optionen, denen Sie häufiger begegnen werden:

- **Before**
  wird analog zu `After` benutzt, um Abhängigkeiten zu definieren. In diesem Fall soll der Dienst vor den angegebenen anderen Diensten gestartet werden.
- **Require**
  erfordert, dass die angegebenen Dienste vor dem Start erfolgreich gelaufen sind. Wenn die Dienste beendet werden, soll unser Dienst ebenfalls gestoppt werden.
- **ExecStop**
  enthält das Kommando, das zum Stoppen benutzt wird.
- **Conflicts**
  beendet die angegebenen Dienste, wenn dieser jetzt konfigurierte Dienst gestartet wird.
- **OnFailure**
  enthält eine Liste an Units, die gestartet werden, wenn sich dieser Service fehlerhaft beendet.

- **Type**
  ist für Services entweder `simple` oder `forking`, wobei das Erste für einen Prozess steht, der ständig läuft, und das Zweite für einen Prozess, der einen Kindprozess abspaltet und sich danach beendet. (`oneshot` ist ein Service, der nur läuft und sich danach selbst beendet. Dieser Typ wird manchmal als Ziel für `OnFailure` benutzt.)

[+] Eigene Servicedateien sollten Sie im dafür vorgesehenen Verzeichnis */etc/systemd/system* anlegen und nach dem Anlegen mittels `systemctl daemon-reload` aktivieren.

Weitergehende Informationen finden Sie in der Manpage *systemd.unit*.

Folgender Befehl wurde im aktuellen Abschnitt behandelt:

- `systemctl show <SERVICE>`

2.5.5 Target Units

Die folgenden Targets finden sich auf einem Desktop-System:

```
# systemctl list-units "*.target"

UNIT                   LOAD   ACTIVE SUB    DESCRIPTION
basic.target           loaded active active Basic System
bluetooth.target       loaded active active Bluetooth
cryptsetup.target      loaded active active Encrypted Volumes
getty.target           loaded active active Login Prompts
graphical.target       loaded active active Graphical Interface
local-fs-pre.target    loaded active active Local File Systems (Pre)
local-fs.target        loaded active active Local File Systems
multi-user.target      loaded active active Multi-User System
network-online.target loaded active active Network is Online
network.target         loaded active active Network
nfs-client.target      loaded active active NFS client services
paths.target           loaded active active Paths
remote-fs-pre.target   loaded active active Remote File Systems (Pre)
remote-fs.target       loaded active active Remote File Systems
slices.target          loaded active active Slices
sockets.target         loaded active active Sockets
sound.target           loaded active active Sound Card
swap.target            loaded active active Swap
sysinit.target         loaded active active System Initialization
timers.target          loaded active active Timers
```



```
LOAD   = Reflects whether the unit definition was properly loaded.
ACTIVE = The high-level unit activation state, i.e. generalization of SUB.
SUB    = The low-level unit activation state, values depend on unit type.

20 loaded units listed. Pass --all to see loaded but inactive units, too.
To show all installed unit files use 'systemctl list-unit-files'.
```

Listing 2.20 Targets auf einem Desktop-System

Mittels `systemctl list-dependencies multi-user.target` können Sie sich anzeigen lassen, von welchen Diensten das Multi-User-Target abhängt.

Folgender Befehl wurde in diesem Abschnitt behandelt:

- `systemctl list-units '*.targets'`

2.5.6 »systemd«- und Servicekonfigurationen

Ein installiertes System kommt mit einer großen Anzahl an Diensten daher. Um da nicht den Überblick zu verlieren, bietet *systemd* einige Kommandos, um das laufende System abzufragen. Listing 2.21 zeigt Ihnen einen Auszug der 228 bekannten Units auf einem minimal installierten CentOS-System:

```
UNIT FILE                                  STATE
[…]
crond.service                              enabled
fstrim.service                             static
kdump.service                              enabled
NetworkManager.service                     enabled
postfix.service                            enabled
sshd.service                               enabled
ctrl-alt-del.target                        disabled
graphical.target                           static
hibernate.target                           static
hybrid-sleep.target                        static
network-online.target                      static
runlevel0.target                           disabled
runlevel1.target                           disabled
runlevel2.target                           static
runlevel3.target                           static
runlevel4.target                           static
runlevel5.target                           static
runlevel6.target                           disabled
```

```
[…]
228 unit files listed.
```

Listing 2.21 Auszug der bekannten Units eines Systems

Von diesen 228 bekannten Units wurden aber nur 96 geladen:

```
# systemctl list-units
UNIT                          LOAD   ACTIVE SUB       DESCRIPTION
[…]
-.mount                       loaded active mounted   /
boot.mount                    loaded active mounted   /boot
dbus.service                  loaded active running   D-Bus System Message Bus
getty@tty1.service            loaded active running   Getty on tty1
NetworkManager.service        loaded active running   Network Manager
getty.target                  loaded active active    Login Prompts
local-fs.target               loaded active active    Local File Systems
multi-user.target             loaded active active    Multi-User System
network-online.target         loaded active active    Network is Online
network.target                loaded active active    Network
systemd-tmpfiles-clean.timer  loaded active waiting   Daily Cleanup of \
                                                      Temporary Directories
[…]
LOAD   = Reflects whether the unit definition was properly loaded.
ACTIVE = The high-level unit activation state, i.e. generalization of SUB.
SUB    = The low-level unit activation state, values depend on unit type.

96 loaded units listed. Pass --all to see loaded but inactive units, too.
To show all installed unit files use 'systemctl list-unit-files'.
```

Listing 2.22 Auszug der geladenen Unit-Dateien

Folgender Befehl wurde in diesem Abschnitt behandelt:

- `systemctl list-unitfiles`

2.5.7 Anzeige von Dienstabhängigkeiten

Wenn man sich die Voraussetzungen anschauen möchte, die erforderlich sind, um eine Unit starten zu können, kann man das Subkommando `list-dependencies` benutzen:

```
# systemctl list-dependencies sshd.service
sshd.service
* |-sshd-keygen.service
```



```
* |-system.slice
* `-basic.target
*   |-microcode.service
*   |-rhel-autorelabel-mark.service
*   |-rhel-autorelabel.service
*   |-rhel-configure.service
*   |-rhel-dmesg.service
*   |-rhel-loadmodules.service
*   |-paths.target
*   |-slices.target
*   | |-.slice
*   | `-system.slice
*   |-sockets.target
*   | |-dbus.socket
*   | |-dm-event.socket
*   | |-systemd-initctl.socket
*   | |-systemd-journald.socket
*   | |-systemd-shutdownd.socket
*   | |-systemd-udevd-control.socket
*   | `-systemd-udevd-kernel.socket
*   |-sysinit.target
*   | |-dev-hugepages.mount
*   | |-dev-mqueue.mount
*   | |-kmod-static-nodes.service
*   | |-lvm2-lvmetad.socket
*   | |-lvm2-lvmpolld.socket
*   | |-lvm2-monitor.service
*   | |-plymouth-read-write.service
*   | |-plymouth-start.service
*   | |-proc-sys-fs-binfmt_misc.automount
*   | |-sys-fs-fuse-connections.mount
*   | |-sys-kernel-config.mount
*   | |-sys-kernel-debug.mount
*   | |-systemd-ask-password-console.path
*   | |-systemd-binfmt.service
*   | |-systemd-firstboot.service
*   | |-systemd-hwdb-update.service
*   | |-systemd-journal-catalog-update.service
*   | |-systemd-journal-flush.service
*   | |-systemd-journald.service
*   | |-systemd-machine-id-commit.service
*   | |-systemd-modules-load.service
```

```
*   | |-systemd-random-seed.service
*   | |-systemd-sysctl.service
*   | |-systemd-tmpfiles-setup-dev.service
*   | |-systemd-tmpfiles-setup.service
*   | |-systemd-udev-trigger.service
*   | |-systemd-udevd.service
*   | |-systemd-update-done.service
*   | |-systemd-update-utmp.service
*   | |-systemd-vconsole-setup.service
*   | |-cryptsetup.target
*   | |-local-fs.target
*   | | |-.mount
*   | | |-boot.mount
*   | | |-rhel-import-state.service
*   | | |-rhel-readonly.service
*   | | `-systemd-remount-fs.service
*   | `-swap.target
*   |   `-dev-mapper-centos_centosswap.swap
*   `-timers.target
*     `-systemd-tmpfiles-clean.timer
```

Listing 2.23 Auszug der Abhängigkeiten von »sshd«

Folgender Befehl wurde in diesem Abschnitt behandelt:

- `systemctl list-dependencies <UNIT>`

2.5.8 Logs mit »journald«

Wie bereits zu Beginn von Abschnitt 2.5 beschrieben, bringt *systemd* sein eigenes Logging-Framework namens *journald* mit. Dass die Log-Dateien binär gespeichert werden, um sie länger und fälschungssicher – so zumindest der Anspruch der *systemd*-Entwickler – speichern zu können, ist jedoch ein großer Kritikpunkt der Linux-Community. Allerdings hat *journald* Charme und bringt außer der Umgewöhnung auch einige Vorteile mit, wie beispielsweise dass Fehler in den Log-Dateien in Rot markiert werden und so eher auffallen.

Rufen Sie beispielsweise `journalctl` ohne weitere Parameter auf, bekommen Sie einen interaktiven Auszug aller Log-Dateien, so wie sie früher in */var/log/syslog* oder */var/log/messages* landeten. Hier können Sie auch durch Eingabe eines großen »F« in den Follow-Modus wechseln. Mit dem Parameter `-f` oder `--follow` wird Ihnen das Log analog zu einem `tail -f` angezeigt. Wenn Sie die letzten 20 Log-Einträge anschauen wollen, benutzen Sie `-n 20` oder `--lines=20`. Der Parameter `--reverse` zeigt die Einträge in umgekehrter Reihenfolge an.

Einträge eines bestimmten Zeitraums grenzen Sie durch `--since` und `--until` ein. Dabei wird ein Datum in der Form `"2018-07-30 18:17:16"` ausgewertet. Ohne Datum wird der heutige Tag angenommen, ohne Sekunden wird 0 (null) angenommen, Sonderausdrücke wie `yesterday`, `today`, `tomorrow` oder `now` sind möglich.

Einer der wichtigsten Parameter ist `-u` oder `--unit=`, womit nur die Log-Dateien einer einzelnen Unit oder eines Satzes an Units ausgegeben werden. Wollen Sie beispielsweise die Log-Einträge des SSH-Daemons vom 5. August 2018 zwischen 13:00 Uhr und 14:00 Uhr haben, geben Sie den Befehl aus Listing 2.24 ein:

```
# journalctl --since="2018-08-05 13:00" --until="2018-08-05 14:00" \
--unit=sshd.service

-- Logs begin at Fr 2018-08-05 07:19:24 CEST, end at Fr 2018-08-05 15:56:51 CEST. --
Aug 05 13:07:24 centos sshd[13128]: reverse mapping checking getaddrinfo for \
                                    1-2-3-4.a.b [1.2.3.4] failed - POSSIBLE BREAK-IN\
                                    ATTEMPT!
Aug 05 13:07:24 centos sshd[13130]: reverse mapping checking getaddrinfo for \
                                    1-2-3-4.a.b [1.2.3.4] failed - POSSIBLE BREAK-IN\
                                    ATTEMPT!
Aug 05 13:07:24 centos sshd[13128]: Connection closed by 1.2.3.4 [preauth]
Aug 05 13:07:24 centos sshd[13130]: Connection closed by 1.2.3.4 [preauth]
```

Listing 2.24 Log-Auszug des SSH-Daemons

Die Logs von *journald* werden nach einem Neustart gelöscht. Wenn Sie das nicht wollen, sollten Sie das Verzeichnis */var/log/journal* anlegen und das Signal `SIGUSR1` an den *journald*-Prozess senden. Damit werden die Logs in dem angegebenen Verzeichnis persistiert, sodass sie maximal zehn Prozent der Größe des Dateisystems belegen. Weitere Konfigurationen nehmen Sie in der Datei */etc/systemd/journald.conf* vor.

Folgender Befehl wurde in diesem Abschnitt behandelt:

- `journalctl`

2.5.9 Abschlussbemerkung

Die hier vorgestellten Befehle und Direktiven bilden nur einen Ausschnitt der Möglichkeiten von *systemd* ab. Dieser Ausschnitt ist jedoch eine gute Basis für weitere Schritte. Eigene Skripte und Job-Definitionen können Sie damit bereits jetzt erstellen. Über die Manpages können Sie noch einige andere Kommandos und Subkommandos finden.

Kapitel 6

Paketmanagement

Der Umgang mit Software – Quellcode, Paketen und Paketmanagementlösungen – steht im Vordergrund dieses Kapitels. Erfahren Sie, wie Sie Ihre selbst entwickelte Software in Pakete schnüren, Update-Pakete erstellen und das Paketmanagement Ihrer Distribution richtig einsetzen.

Zu den Aufgaben einer Paketverwaltung (engl. *packet management*) gehören das Installieren, das Aktualisieren und das Deinstallieren von Software. Schon in den Anfangsjahren von Linux wurde solch ein System erforderlich. Durch die wachsende Anzahl von Programmen der *GNU-Sammlung* konnten Linux-Systeme fast alle Aufgaben bewältigen. Lediglich das Installieren der Software sorgte für Unmut, da das Übersetzen des Quellcodes (*Kompilieren*) auf dem eigenen System oft beschwerlich war. Vorausgesetzte Software (*dependencies*[1]) musste von Hand nachinstalliert werden, Updates sorgten für Konflikte mit anderer installierter Software, und die Deinstallation musste ebenfalls von Hand durchgeführt werden.

6.1 Paketverwaltung

Hier setzt die Paketverwaltung an. Sie löst Abhängigkeiten auf und versucht, notwendige Software selbstständig nachzuinstallieren. Sie entfernt Programme sauber aus dem System und prüft bei Updates, ob Überschneidungen existieren und wie diese gelöst werden können. Dafür musste die Software aber in eine entsprechende Form gebracht werden: in Pakete.

Eine Paketverwaltung besteht aus zwei Teilen. Der erste Teil ist für das Laden der Programme aus einem *Repository*[2] sowie für das Auflösen von Abhängigkeiten und Konflikten verantwortlich. Der zweite Teil, zum Beispiel *dpkg (Debian Packet Management*) oder *rpm (RPM Packet Management)*, sorgt für die eigentliche Installation.

In diesem Kapitel erfahren Sie alles zu den zwei größten Paketverwaltungen *dpkg* und *rpm*. Sie erfahren, wie Pakete zu diesen Paketverwaltungen konvertiert werden, welche weiteren Programme es gibt und wie Sie selbst Pakete aktualisieren, erstellen und patchen können.

1 *dependencies*, engl. für *Abhängigkeiten*. Damit meint man in der Paketverwaltung Software, die für den Betrieb anderer Software benötigt wird.
2 *Repository*, engl. für *Lager*, *Depot*.

6.1.1 »rpm« oder »deb«?

Trotz aller Vorteile, die eine Paketverwaltung bietet, besitzen Paketverwaltungen auch Nachteile. Zum einen verfügen Sie zum Großteil lediglich über Binärpakete. Diese Pakete sind für eine Architektur übersetzt und für diese optimiert und auch nur auf dieser lauffähig. Dies führt zu statischen Paketen, die nicht voll an das System, auf dem sie laufen, angepasst sind. Zum anderen kommt es bei Paketverwaltungen auch zu Konflikten.

Wenn in den Paketen *Alpha* und *Beta* teilweise die gleichen Dateien enthalten sind, können Sie nicht beide Pakete gleichzeitig installieren. Falls die Aktualisierung des Pakets *Gamma* auch die Aktualisierung des Pakets *Delta* fordert, das Paket *Epsilon* aber die »alte Version« des Pakets *Delta* benötigt, ist eine Aktualisierung von *Gamma* nicht möglich. Eine gute Paketverwaltung zeichnet sich dadurch aus, dass sie diese Konflikte effektiv löst.

In der heutigen Zeit haben sich drei Konzepte durchgesetzt:

- die *rpm*-Paketverwaltung
- die *dpkg*-Paketverwaltung
- quellenbasierte Distributionen

Die ersten beiden Paketverwaltungen haben die Linux-Welt quasi in zwei Lager gespalten. Die letzte Variante findet eher selten Anwendung, aber Distributionen wie zum Beispiel *Gentoo Linux* verzichten ganz auf Binärpakete und stellen lediglich den Quellcode zur Verfügung, sodass jede Installation voll an das System angepasst wird.

Jede Distribution verfügt über ihre eigene Implementierung einer Paketverwaltung. Das hat zur Folge, dass zum Beispiel zwei auf *rpm* basierende Distributionen nicht zwingend miteinander kompatibel sein müssen.

»rpm«

Die Paketverwaltung *rpm* wurde ursprünglich als *Red Hat Packet Management* bezeichnet, da sie von Red Hat entwickelt wurde. Da *rpm* aber ein Teil der *Linux Standard Base* werden wollte, wurde das Projekt einfach in ein rekursives Akronym umbenannt: *RPM Packet Management*. Das Ziel vom *rpm* bestand darin, Softwarepakete sowohl für Entwickler als auch für den Anwender einfacher zu gestalten. Abhängigkeiten sollten berücksichtigt und automatisch aufgelöst werden, Redundanzen sollten vermieden werden, und das saubere Entfernen von Software sollte möglich gemacht werden. Ebenso sollte das Einspielen von Updates einfacher gestaltet werden, ebenso wie das sichere Verwalten von Konfigurationen.

1999 kam es zur Zweiteilung von *rpm*, da der Hauptentwickler Jeff Johnson Red Hat verließ und einen Fork des Projekts startete. Jetzt gibt es zwei *rpm*-Versionen: zum einen das in SUSE, Red Hat, Fedora und vielen anderen Distributionen verwendete *rpm* von *rpm.org*, zum anderen den Fork von Johnson, *rpm5*, der zum Beispiel in *Alt Linux*, *ArkLinux* und *Unity Linux* Anwendung findet.

Bei *rpm*-Paketen handelt es sich um komprimierte Dateien, denen ein binärer Header vorangestellt ist. Dieser Header ist nicht komprimiert, sodass er leicht durchsucht werden kann.

Die zentrale Datei eines Pakets im *rpm*-Format ist die *SPEC*-Datei. Sie enthält alle Metainformationen des Pakets, also Informationen darüber, was in dem Paket enthalten ist, wohin es installiert wird, wie es installiert wird, sowie Informationen zum Paket selbst.

Diese Datei ist in folgende acht Sektionen unterteilt:

- **Präambel**
  Informationen über das Paket, die mittels `rpm` dem Benutzer angezeigt werden
- **Prep**
  ein Skript, das den Quellcode-Dateibaum erzeugt
- **Build**
  ein Kompilierungsskript
- **Install**
  Installationsanweisung für die Binarys – in den meisten Fällen `make install`
- **Files**
  eine Auflistung aller Dateien, die im Paket enthalten sind
- **Install/Uninstall**
  Skripte für die Installation und Deinstallation. Diese Sektion besteht wiederum aus vier Unterabschnitten: **pre**, **post**, **prerun** und **postrun**. Das sind Befehle, die vor/nach der Installation/Deinstallation ausgeführt werden.
- **Verify**
  Skript zur Prüfung der Installation
- **Clean**
  Befehle, die auf dem Entwicklerrechner ausgeführt werden sollen

Die Dateinamen der *rpm*-Pakete haben folgende Konvention:

```
<Bezeichnung>-<Versionsnummer>-<Revisionsnummer>.<Architektur>.rpm
```

Listing 6.1 Schema der »rpm«-Paketdateinamen

»deb«

Die Paketverwaltung *deb* wurde von Ian Murdock entwickelt. Die Bezeichnung *deb* bezieht sich hierbei auf die ersten drei Buchstaben der Linux-Distribution *Debian*. Die Zielsetzung von *deb* entspricht der von *rpm*, allerdings sind beide nicht miteinander kompatibel. Hierfür gibt es aber Konvertierungstools, die es erlauben, Pakete für die jeweils andere Paketverwaltung nutzbar zu machen.

Ein *deb*-Paket setzt sich aus folgenden Abschnitten zusammen:

- **debian-binary**
  Textdatei mit Angabe der Versionsnummer des verwendeten Paketformats
- **control**
  enthält alle relevanten Informationen des Pakets, die in eigenen Dateien verwaltet werden. Die folgenden Dateien sind immer vorhanden:
  - **control**
    Kurzbeschreibung des Pakets und Auflistung der Abhängigkeiten
  - **md5sums**
    Prüfsummen der enthaltenen Dateien
  - **conffiles**
    Auflistung der Konfigurationsdateien im Paket
  - **preinst, postinst, prerm, postrm**
    Skripte, die vor oder nach der Installation/Deinstallation ausgeführt werden
  - **config**
    ein *debconf*-Skript, das Metainformationen für die *debconf*-Datenbank enthält
  - **shlibs**
    Auflistung der Programmbibliotheken
- **data**
  Archiv der enthaltenen Programmdaten

Auch hier gibt es eine zentrale Datei, die für das Paket verantwortlich ist: *control*. In ihr sind sowohl alle Metainformationen zum Paket enthalten als auch die Installationsanweisungen.

Dateinamen von *deb*-Paketen folgen einer definierten Syntax:

```
<Bezeichnung>_<Versionsnummer>-<Revisionsnummer>_<Architektur>.deb
```

Listing 6.2 Schema der »deb«-Paketdateinamen

6.1.2 »yum«, »yast«, »zypper« oder »apt«?

Wie bereits erläutert wurde, besteht eine Paketverwaltung nicht allein aus einem Programm. Zu ihr gehört immer (mindestens) ein Tool, das sich um das Laden der Software, das Auflösen von Abhängigkeiten und die Beseitigung von Konflikten kümmert. Jede Distribution besitzt eine eigene Umsetzung. So findet *yum* auf Red-Hat-basierten Systemen Anwendung, *yast* und *zypper* auf SUSE-basierten und *apt* auf Debian-basierten. Wenn Distributionen binärkompatibel miteinander sind, können Sie auch Pakete aus der jeweils anderen Distribution installieren.

[»] Ubuntu- und Debian-Pakete sind zum Teil binärkompatibel, sodass Debian-Pakete auch unter einigen Ubuntu-Versionen lauffähig sind. Ebenso gilt dies für Red Hat und SUSE.

Die Bedienung der einzelnen Tools unterscheidet sich erheblich voneinander. Die Debian-basierte Variante *apt* ist eine Suite mit mehreren Anwendungen, die jeweils eine Anwendung für spezielle Aufgaben zur Verfügung stellt. Die Red-Hat-Variante *yum* kommt hingegen mit nur einem Programm aus, und die SUSE-Variante *yast* kann sowohl als Kommandozeilentool als auch im ASCII-Menü gesteuert werden, wobei ab openSUSE 10.2 das zusätzliche Tool *zypper* parallel zu *yast* eingesetzt werden kann.

Neben den von den Distributionen ausgelieferten Programmen können auch unabhängige Programme eingesetzt werden. Dies ist aber in den meisten fällen nicht notwendig. [+]

Tabelle 6.1 zeigt einen Auszug der Befehle, mit denen Sie die Paketverwaltung verschiedener Distributionen über die Kommandozeile bedienen können. Tabelle 6.2 zeigt, wie Sie lokale Pakete verwalten.

Aktion	CentOS	openSUSE Leap	Debian/Ubuntu
Paket installieren	`yum install`	`yast -i` `zypper install`	`apt install`
Paket deinstallieren	`yum remove`	`yast --remove` `zypper remove`	`apt remove`
Paketinformationen abfragen	`yum info`	`zypper info`	`apt show`
Paket zur Datei suchen	`yum provides`	`zypper wp`	`apt-file search`
Alle installierten Pakete anzeigen	`yum list *`	–	–

Tabelle 6.1 Auszug: Kommandozeilenbefehle für Repositorys

Aktion	CentOS	openSUSE Leap	Debian/Ubuntu
Paket installieren	`yum install`	`rpm -i`	`dpkg -i`
Paket deinstallieren	`yum remove`	`rpm -e`	`dpkg -r`
Paketinformationen abfragen	`yum info`	`rpm -qi`	`dpkg -p`
Paket zur Datei suchen	`yum provides`	`rpm -qf`	–
Alle installierten Pakete anzeigen	`yum list *`	`rpm -qa`	`dpkg -l`

Tabelle 6.2 Auszug: Kommandozeilenbefehle für lokale Pakete

6.1.3 Außerirdische an Bord – »alien«

Es gibt Software, die nur in einer Paketform angeboten wird und nicht als Quellcode zur Verfügung steht. Um diese Software auf dem eigenen System betreiben zu können, müssen Sie das Paket umwandeln. Hier eilt Ihnen *alien* zur Hilfe. Das ursprünglich vom Debian-Entwickler Christoph Lameter geschriebene Tool konvertiert Pakete in die Formate *rpm*, *deb*, *slp* (Stampedes), *tgz* (Slackwares) und *pkg* (Solaris).

Die Installation von *alien* setzt viele weitere Pakete voraus, unter anderem auch die Kompilierungsprogramme (*gcc*, *make* etc.). Auf Systemen, die auf Debian basieren, können Sie *alien* aus den Paketquellen installieren. Leider ist *alien* weder in CentOS noch in openSUSE Leap oder Ubuntu enthalten. Laden Sie den aktuellen Quellcode unter *http://packages.debian.org/stable/source/alien* herunter, und führen Sie nach dem Entpacken wie in der enthaltenen Datei *INSTALL* beschrieben die folgenden Befehle aus:

```
daniel@example:/usr/local/alien# perl MakeFiles.pl
[…]
root@example:/usr/local/alien# make
root@example:/usr/local/alien# make install
```

Listing 6.3 »alien« aus dem Quellcode installieren

Oder erzeugen Sie auf einem Debian-basierten System einfach ein *rpm*-Paket mit *alien*. Laden Sie dafür die entsprechende *deb*-Datei mit `apt download alien` herunter. Wenden Sie *alien* nun mit dem Schalter `-r` (oder `-to-rpm`) an:

```
root@example:/opt# alien -r alien_8.95_all.deb
alien-8.95-2.noarch.rpm generated
```

Listing 6.4 Umwandeln von »deb« in »rpm«

alien erzeugt nun die Datei *alien-8.95-2.noarch.rpm*. Wie Sie dem neuen Dateinamen entnehmen können, wurde auch die Versionsnummer angepasst. Falls Sie die eigentliche Versionsnummer des Pakets behalten wollen, verwenden Sie den Parameter `-k` beim Aufruf von *alien*.

[+]

Speicherort installierter Pakete

Installierte Pakete auf Systemen mit *deb*-Paketverwaltung finden Sie im Verzeichnis */var/cache/apt/archives/*. Bei openSUSE Leap finden Sie die installierten Pakete unter */var/cache/zypp/packages/<Repository>:<RepoName>/rpm/* und bei CentOS unterhalb von */var/cache/yum*.

Falls Sie ein Paket auf dem System konvertieren, auf dem die Software installiert werden soll, können Sie diese mit dem Schalter `-i` auch direkt von *alien* installieren lassen.

6.2 Pakete im Eigenbau

Neben der Vielzahl an Software, die über die Repositorys bezogen werden kann, gibt es auch Software, die lediglich als Quellcode zur Verfügung steht. Um solche Programme auf Ihrem System betreiben zu können, müssen Sie sie kompilieren. Hierfür verwenden Sie den Ihnen vermutlich bereits bekannten Dreisprung:

```
daniel@example:/usr/local/packet# ./configure
daniel@example:/usr/local/packet# make
daniel@example:/usr/local/packet# sudo make install
```

Listing 6.5 Kompilierung im Dreisprung

Dies ist natürlich eine adäquate Möglichkeit, Software auf »einem« System zu verwenden. Wenn Sie aber mehrere Systeme der gleichen Architektur mit dem gleichen Betriebssystem verwenden, können Sie den Quellcode auch in ein Paket schnüren, um so die Software einfach zu verteilen, eine Update-Sicherheit zu erzeugen oder auch Patches effektiv zu verteilen. In diesem Abschnitt erfahren Sie alles zu Paketen: wie Sie diese aus einem *tarball*[3] erzeugen, wie Sie Patches einspielen und wie Sie Update-Pakete erstellen.

6.2.1 Vorbereitungen

Damit Sie Pakete erzeugen können, müssen einige Pakete auf Ihrem System vorhanden sein, die zur Erstellung notwendig sind: die sogenannte *Build-Umgebung*. Keine Sorge, Sie müssen die benötigten Pakete nicht einzeln identifizieren und separat installieren. In den Paketquellen sind Metapakete oder Gruppen vorhanden, die alle benötigten Pakete enthalten.

Auf Debian/Ubuntu müssen Sie das Metapaket aus Listing 6.6 installieren:

```
root@ubuntu:~$ apt install build-essential   ### bei Ubuntu zusätzlich: debhelper
```

Listing 6.6 Build-Umgebung installieren unter Debian/Ubuntu

Auf einem CentOS-System müssen Sie die Gruppe aus Listing 6.7 installieren:

```
[root@centos ~]# yum -y groupinstall Development\ Tools
```

Listing 6.7 Build-Umgebung installieren unter CentOS

Auf einem openSUSE-leap-System wiederum können Sie die benötigten Pakete über Muster (engl. *pattern*) installieren lassen:

```
leap:~> sudo zypper install -t pattern devel_C_C++ devel_basis devel_rpm_build
```

Listing 6.8 Build-Umgebung installieren unter openSUSE Leap

3 *tarball (tape archive ball)* – Archivformat, in dem oft Quellcode ausgeliefert wird.

6.2.2 Am Anfang war das Makefile

Software, die nicht in den Repositorys zur Verfügung steht, wird in der Regel von den Entwicklern als *tgz*- oder auch *tar.gz*-Datei zur Verfügung gestellt. Aus dieser Datei entpacken Sie den Quellcode und kompilieren dann das Programm. Um selbst geschriebene Software so zur Verfügung zu stellen, muss diese erst mal in die richtige Form gebracht werden. Dazu dient das *Makefile*, auf das das *configure*-Skript zugreift, das das Programm entsprechend Ihrem System verarbeitet.

Im Grunde genommen sind Makefiles nichts weiter als Textdateien, in denen der Übersetzungsprozess von Programmen formalisiert enthalten ist. Darüber hinaus stellt das Makefile eine Intelligenz zur Verfügung, die bereits vorhandene und kompilierte Abhängigkeiten (oder bei Updates auch eigenen Code) erkennt und diese nicht erneut kompiliert.

Das Erstellen eines solchen Makefiles von Hand kann bei größeren Projekten mit vielen Abhängigkeiten schnell in unüberschaubare Arbeit ausarten. Um effizient Makefiles zu erstellen, gibt es die »*GNU autotools*«, die Ihnen die Arbeit zu einem Großteil abnehmen. Folgende Arbeitsschritte müssen Sie durchlaufen, um ein eigenes *tgz* zu erstellen:

1. Quellcode erstellen
2. *Makefile.am* erstellen
3. *autoscan* – *configure.scan* in *configure.in* umwandeln
4. *aclocal* – Anpassung für die Sprachumgebung
5. *autoconf* – Verarbeitung von Konfigurationsdateien
6. *autoheader* – Verarbeitung von zusätzlichem Quellcode
7. *automake* – Erzeugen des Makefiles

In diesem Abschnitt zeigen wir Ihnen, wie Sie das allseits beliebte Programm »*helloworld*« von seinem C-Code-Dasein befreien und als *tgz*-Datei zur Verfügung stellen.

[+] Wenn noch nicht vorhanden, installieren Sie das Paket *autoconf* aus den Paketquellen (in der Regel ist das Paket aber Bestandteil der Build-Umgebung).

Erstellen Sie zunächst ein Verzeichnis *hello-1.0* in Ihrem Homeverzeichnis. In diesem Verzeichnis erstellen Sie das *helloworld.c*-Programm mit folgendem Quellcode:

```
/* Simple  "Hello World!" program in C */
#include <stdio.h>
main()
{
 printf("Hello world! Once again... \n");
}
```

Listing 6.9 »helloworld.c«

Erstellung als Benutzer

Die Erstellung von Paketen (*tgz*, *rpm* oder *deb*) sollte immer als normaler Benutzer ausgeführt werden. Da *root* Vollzugriff auf alle Dateien hat, könnten während der Installation eines von ihm erstellten Pakets Dateien überschrieben oder gelöscht werden, die auf dem System benötigt werden.

Die Installation des gepackten Programms selbst muss hingegen von *root* durchgeführt werden, da Benutzern die entsprechenden Rechte fehlen.

[!]

Das soeben erstellte Programm dient als zu verpackende Basis. Erstellen Sie nun die Datei *Makefile.am* mit folgendem Inhalt:

```
bin_PROGRAMS = helloworld
helloworld_SOURCES = helloworld.c
```

Listing 6.10 »Makefile.am«

Die Datei *Makefile.am* wird von den *autotools* ausgewertet. In ihr werden der Name des Programms (`bin_PROGRAMS`) und dessen Quellcode (`helloworld_SOURCES`) deklariert. Bei größeren Projekten werden hier alle programmrelevanten Dateien aufgeführt und auch alle Bibliotheken. Führen Sie nun `autoscan` aus. Dieses Programm analysiert die *Makefile.am* und ihren Quellcode. Es erstellt die Datei *configure.scan*, die als Basis zur Erstellung der *configure.ac* dient (siehe Listing 6.11):

```
#                                               -*- Autoconf -*-
# Process this file with autoconf to produce a configure script.

AC_PREREQ([2.69])
AC_INIT([FULL-PACKAGE-NAME], [VERSION], [BUG-REPORT-ADDRESS])
AC_CONFIG_SRCDIR([helloworld.c])
AC_CONFIG_HEADERS([config.h])

# Checks for programs.
AC_PROG_CC

# Checks for libraries.

# Checks for header files.

# Checks for typedefs, structures, and compiler characteristics.

# Checks for library functions.
```

```
AC_CONFIG_FILES([Makefile])
AC_OUTPUT
```

Listing 6.11 »configure.scan«

Benennen Sie die Datei entsprechend um, und passen Sie die Parameter dem *helloworld.c*-Programm an:

```
AC_PREREQ([2.69])
AC_INIT([helloworld], [1.0], [bugs@example.com])
AC_CONFIG_SRCDIR([helloworld.c])
AM_INIT_AUTOMAKE([1.9 foreign])
AC_PROG_CC
AC_CONFIG_FILES([Makefile])
AC_OUTPUT
```

Listing 6.12 »configure.ac«

Achten Sie darauf, die Zeile `AC_CONFIG_HEADERS([config.h])` zu entfernen, da unser Programm keine Header-Datei besitzt. Da Sie weiter mit den *autotools* arbeiten, müssen Sie die vierte Zeile hinzufügen, die von `autoscan` nicht erstellt wurde. Führen Sie anschließend die Befehle `aclocal` und `autoconf` aus. Diese ergänzen und erweitern die Struktur des *tgz*. Führen Sie nun `autoheader` aus, das mit einer Fehlermeldung endet (siehe Listing 6.13):

```
daniel@example:/home/daniel/hello-1.0# autoheader
autoheader: error: AC_CONFIG_HEADERS not found in configure.ac
```

Listing 6.13 Fehlermeldung: »autoheader«

Obwohl der Befehl eine Fehlermeldung ausgibt, nimmt er wichtige Konfigurationen vor. Anschließend beenden Sie die Verarbeitung mit `automake`, das neben dem *configure*-Skript das passende *Makefile* erzeugt:

```
daniel@example:/home/daniel/hello-1.0# automake --add-missing
```

Listing 6.14 Fehlende Dateien ergänzen

Der Parameter `-add-missing` gibt an, dass *automake* fehlende Standarddateien hinzufügen soll. Jetzt stehen Ihnen alle benötigten Dateien zur Verfügung. Packen Sie diese mittels *tar* in ein Archiv, das Sie auf jedem System entpacken und kompilieren können:

```
daniel@example:/home/daniel/# tar -cavhf hello-1.0.tar.gz hello-1.0/
```

Listing 6.15 Ein »tar«-Archiv erstellen

Nach dem bekannten Dreisprung können Sie das Programm einfach über den Befehl `hello-world` aufrufen und erhalten die erwartete Ausgabe: »`Hello World! Once again ...`«. Beachten Sie, dass der Befehl `make install` immer als *root* ausgeführt werden muss!

6.2.3 Vom Fellknäuel zum Paket

Eine noch elegantere Variante, um das Programm *helloworld* zur Verfügung zu stellen, besteht darin, es in ein *rpm*- oder *deb*-Paket zu hüllen. In diesem Abschnitt zeigen wir Ihnen die minimalen Anforderungen, um ein Paket erzeugen zu können. Bitte beachten Sie, dass noch weitere Konfigurationen notwendig sind, wenn Sie Pakete in die Paketverwaltung Ihrer Distribution hochladen wollen. (Lesen Sie dafür die *Guidelines* der Distribution.)

DEB-Pakete erzeugen: »dpkg-buildpackage«

Zum Erzeugen von DEB-Paketen wird eine gewisse Verzeichnisstruktur vorausgesetzt. Erzeugen Sie daher die benötigten Verzeichnisse:

```
daniel@ubuntu:~$ mkdir -p build/hello-1.0/debian
```

Listing 6.16 Build-Umgebung erzeugen

Legen Sie nun die Datei *control* im Verzeichnis *debian* an. Diese enthält wichtige Informationen zum Paket. In Listing 6.17 sehen Sie die Werte, die mindestens gesetzt sein müssen:

```
Source: hello
Build-Depends: debhelper (>= 9)
Maintainer: Daniel van Soest <daniel@example.com>
Standards-Version: 3.9.5

Package: hello
Architecture: any
Description: Easy to use "helloworld" Program written in C.
 Need some sunshine? Just go and run this nifty little program.
```

Listing 6.17 Inhalt von »debian/control«

Bitte passen Sie den *Maintainer*[4] mit gültigen Werten an (in Fettschrift dargestellt). Beachten Sie, dass die erste Zeile der Direktive `Description` die Kurzbeschreibung darstellt. Die folgenden Zeilen beginnen stets mit einem Leerzeichen – all diese Zeilen werden als ausführliche Beschreibung ausgewertet. Eine weitere Datei, die im Verzeichnis *debian* zwingend vorhanden sein muss, ist *changelog*. Erstellen Sie diese mit dem Inhalt aus Listing 6.18:

```
hello (1.0-1) unstable; urgency=low

  * Initial release

 -- Daniel van Soest <daniel@example.com>  Sat, 21 May 2018 07:10:08 +0200
```

Listing 6.18 Inhalt von »debian/changelog«

4 *Maintainer* (engl. »to maintain«, dt. »instandhalten«) – Autor des distributionsspezifischen Pakets.

Auch hier müssen Sie in der letzten Zeile den Maintainer korrigieren und das Datum und die Uhrzeit anpassen. Zusätzlich werden noch weitere Dateien benötigt, die wir in einem Schwung mit den Befehlen aus Listing 6.19 erzeugen:

```
daniel@ubuntu:~/build/hello-1.0$ echo '9' > debian/compat
daniel@ubuntu:~/build/hello-1.0$ echo 'helloworld /usr/bin/' > debian/hello.install
daniel@ubuntu:~/build/hello-1.0$ echo -e '%:\n\tdh $@' > debian/rules
```

Listing 6.19 Weitere Dateien anlegen und befüllen

Zu guter Letzt müssen Sie noch die bereits kompilierte Version des Programms mit `cp /usr/local/bin/helloworld ~/build/hello-1.0` ins Stammverzeichnis der Build-Umgebung kopieren.

Damit sind alle benötigten Dateien vorhanden, und Sie können das DEB-Paket mit dem Befehl aus Listing 6.20 erzeugen:

```
daniel@ubuntu:~/build/hello-1.0$ dpkg-buildpackage -rfakeroot
[…]
daniel@ubuntu:~/build/hello-1.0$ ls -lha ../
insgesamt 28K
drwxrwxr-x 3 daniel daniel 4,0K Mai 21 08:03 .
drwxr-xr-x 8 daniel daniel 4,0K Mai 21 08:02 ..
drwxrwxr-x 3 daniel daniel 4,0K Mai 21 08:03 hello-1.0
-rw-rw-r-- 1 daniel daniel 1,1K Mai 21 08:03 hello_1.0-1_amd64.changes
-rw-r--r-- 1 daniel daniel 2,4K Mai 21 08:03 hello_1.0-1_amd64.deb
-rw-rw-r-- 1 daniel daniel  485 Mai 21 08:03 hello_1.0-1.dsc
-rw-rw-r-- 1 daniel daniel 2,2K Mai 21 08:03 hello_1.0-1.tar.gz
```

Listing 6.20 DEB-Paket erzeugen mit »dpkg-buildpackage«

RPM-Pakete erzeugen: »rpmbuild«

Um ein RPM-Paket aus dem Programm *helloworld* zu erzeugen, müssen Sie zunächst eine Build-Umgebung erstellen. Führen Sie dafür die Befehle aus Listing 6.21 aus:

```
[daniel@centos ~]$ mkdir -p ~/rpmbuild/{BUILD,RPMS,SOURCES,SPECS,SRPMS}
[daniel@centos ~]$ echo '%_topdir %(echo $HOME)/rpmbuild' > ~/.rpmmacros
```

Listing 6.21 Build-Umgebung erstellen: »CentOS«

Mit diesen Befehlen erzeugen Sie alle benötigten Unterverzeichnisse und erstellen eine Makro-Datei, damit das `rpmbuild` auch weiß, was zu tun ist. Bei openSUSE Leap ist diese Datei eigentlich bereits unter */usr/src/packages* vorhanden – Sie können aber auch eine neue erstellen, so wie in Listing 6.21 gezeigt.

Kopieren Sie anschließend die gepackten Quellen in den erzeugten Ordner *SOURCES*. Für das Beispielprogramm *helloworld* genügt der Befehl aus Listing 6.22:


```
[daniel@centos rpmbuild]$ cp ../hello-1.0.tar.gz SOURCES/
```

Listing 6.22 Quellen kopieren nach »SOURCES«

Zum Erzeugen eines RPM-Pakets wird stets eine sogenannte *spec*-Datei benötigt. Diese enthält alle Informationen zum Paket und spezifiziert, wie das Paket erzeugt werden soll.

Öffnen Sie nun die Datei *SPECS/hello.spec* mit einem Editor. Da die Build-Umgebung erzeugt wurde, wird im Editor direkt die passende Template-Datei geöffnet. Diese unterscheidet sich von Distribution zu Distribution. In dieser Datei müssen aber immer mindestens folgende Zeilen angepasst werden:

```
Name:           hello
Version:        1.0
Release:        1
Summary:        Easy to use "helloworld" Program written in C.
License:        GPLv2
Group:          Development/Tools
Source:         %name-%version.tar.gz
[…]
%files
%doc
/usr/bin/helloworld
```

Listing 6.23 Erforderliche Anpassungen in »SPECS/hello.spec«

Falls in Ihrem Paket die Standarddokumentation, wie zum Beispiel *README* oder *COPYING*, vorhanden ist, müssen Sie diese unter `doc` mit Leerzeichen getrennt angeben: `%doc <FILE1> <FILE2> <...>`. Ebenso sollten Sie eine ausführliche Beschreibung unter `%description` angeben. Falls Sie Aktualisierungen herausbringen, sollten die Änderungen unter `%changelog` erläutert werden. Dies alles gehört zu einem guten Stil und ist ein »Muss«.

Wenn Sie alle Angaben eingetragen haben, können Sie nun mit dem Programm `rpmbuild` das Paket so erzeugen, wie in Listing 6.24 dargestellt:

```
[daniel@centos rpmbuild]$ rpmbuild -ba SPECS/hello.spec
[…]
[daniel@centos rpmbuild]$ find ./ -name '*.rpm'
./RPMS/x86_64/hello-1.0-1.el7.centos.x86_64.rpm
./RPMS/x86_64/hello-debuginfo-1.0-1.el7.centos.x86_64.rpm
./SRPMS/hello-1.0-1.el7.centos.src.rpm
```

Listing 6.24 Erstellung des Pakets

Wie Sie in Listing 6.24 sehen, war die Erstellung des Pakets erfolgreich. Dieses befindet sich dann, je nach Architektur, unterhalb von *RPMS/<ARCH>/*.

6.2.4 Patchen mit »patch« und »diff«

Das klassische Patchen von Software, das Sicherheitslücken oder Bugs behebt, findet heutzutage kaum noch Anwendung. In den meisten Fällen werden Updates binnen kürzester Zeit als Paket angeboten und in die Repositorys übernommen.

Dennoch gibt es Patches, die eventuelle Verbesserungen mit sich bringen oder sogar in Paketen enthalten sind. Letzteres tritt oft ein, wenn der Maintainer nicht das ganze Paket neu erstellt, sondern lediglich das Update in Form eines Patches hinzugefügt hat.

Ein Patch besteht aus einer *diff*-Datei. Das Programm *diff* erstellt eine Übersicht über die Unterschiede zweier Dateien. Diese wird in einer speziellen Syntax dargestellt, sodass hinzugefügte, gelöschte oder veränderte Zeilen separat dargestellt werden.

Für ein einfaches Beispiel erstellen Sie zwei Textdateien, wie sie in Tabelle 6.3 gezeigt werden.

datei01.txt	datei02.txt

`Ein einfaches Beispiel,`
`das die Funktionsweise von`
`'diff' darstellen soll.`
	`Ein einfaches BEISPIEl,`
	`das die Funktionsweise`
	`des gut funktionierenden Programms`
	`'diff' darstellen soll.`

Tabelle 6.3 Ausgangsdateien

Erstellen Sie einen *diff* der Dateien über den nachstehenden Befehl:

```
daniel@example:~# diff -Naur datei01.txt datei02.txt > datei01.patch
```

Listing 6.25 »diff«-Erzeugung

Die Schalter `-Naur` weisen *diff* an, die Unterschiede der *datei01.txt* zur *datei02.txt* vollständig zu ermitteln und im C-Format darzustellen. Die Ausgabe wird in die Datei *datei01.patch* geschrieben, die folgenden Inhalt hat:

```
daniel@example:~# cat datei01.patch
--- datei01.txt 2018-08-31 16:52:51.284364815 +0200
+++ datei02.txt 2018-08-31 16:53:04.668178142 +0200
@@ -1,3 +1,4 @@
-Ein einfaches Beispiel,
-das die Funktionsweise von
+Ein einfaches BEISPIEl,
+das die Funktionsweise
+des gut funktionierenden Programms
 'diff' darstellen soll.
```

Listing 6.26 »diff« der »datei01.txt« zu »datei02.txt«

In den ersten beiden Zeilen werden die Ursprungsdateien definiert. Durch die dreifachen Plus- und Minuszeichen wird angegeben, von welcher zu welcher Datei der *diff* erstellt wurde. Die dritte Zeile, die von At-Zeichen (@) umgeben ist, zeigt an, wo sich der entsprechende Block in beiden Dateien befindet und, durch ein Komma getrennt, wie lang er in der jeweiligen Datei ist. Anschließend folgen die Zeilen, die durch Symbole kategorisiert sind. Das Programm *diff* kennt drei Kategorien:

- **Hinzugefügte Zeilen** = voranstehendes Pluszeichen (+)
- **Gelöschte Zeilen** = voranstehendes Minuszeichen (–)
- **Nicht veränderte Zeilen** = ohne voranstehendes Zeichen

In einer *diff*-Datei können auch mehrere Dateien und deren Veränderungen aufgelistet sein. In ihr wird nie der gesamte Inhalt einer Datei aufgelistet, sondern lediglich die Veränderungen und deren direkte Umgebung. Daher kann die dritte Zeile des Beispiels auch mehrfach vorkommen und somit mehrere geänderte Blöcke einer Datei referenzieren.

Diese Dateien bilden die Grundlage eines Patches. Das Programm *patch* wird angewandt, um diese Veränderungen einzuspielen. Der Standardaufruf von *patch* lautet:

```
root@example:~# patch -Np0 -i datei01.patch
```

Listing 6.27 Patch einspielen

Das Programm spielt nun die Änderungen aus der Datei *datei01.patch* ein. Der Schalter `-N` gibt dabei an, dass die Richtung aus der *diff*-Datei verwendet werden soll. Der Schalter `p0` weist *patch* an, die Verzeichnisstruktur ab dem Level »0« anzuwenden, und der Schalter `-i` verlangt die Patch-Datei.

Software-Patches werden meist mit einer Verzeichnisstruktur erzeugt. Erstellen Sie folgende Verzeichnisstruktur:

```
build/
|-- orig
|   `-- datei.txt
`-- orig.patch
    `-- datei.txt
```

Listing 6.28 Patch-Verzeichnisstruktur

Kopieren Sie anschließend die *datei01.txt* nach *build/orig/datei.txt* und die Datei *datei02.txt* nach *build/orig.patch/datei.txt*, und stellen Sie deren ursprünglichen Inhalt wieder her. Erstellen Sie nun den neuen *diff* mit folgendem Befehl:

```
daniel@example:~# diff -Naur build/orig/datei.txt \
build/orig.patch/datei.txt > datei.patch
```

Listing 6.29 Neue Patch-Datei erstellen

Spielen Sie den angelegten Patch ein:

```
daniel@example:~# patch -Np0 -i datei.patch
patching file build/orig/datei.txt
```

Listing 6.30 Patch-Datei einspielen

Die Datei *orig/datei.txt* entspricht jetzt der Datei *orig.patch/datei.txt*. Die Besonderheit des Parameters `p0` wird im nächsten Beispiel deutlich. Hier wird der gleiche Patch erneut eingespielt, diesmal aber von einer tieferen Verzeichnisebene aus:

```
daniel@example:~/build# patch -Np1 -i ../datei.patch
patching file orig/datei.txt
```

Listing 6.31 Patch-Datei aus tieferer Verzeichnisebene einspielen

Da sich die Verzeichnisstruktur fest in der Patch-Datei befindet, muss *patch* über den Parameter `1` angewiesen werden, den Pfad um den ersten Teil der Verzeichnisstruktur zu kürzen, damit die Dateien auch gefunden werden.

[!]

Länge der Pfade

Das Programm *patch* prüft vor jedem Durchlauf, ob bereits ein Patch eingespielt wurde. Dies kann Ihnen zum Verhängnis werden, wenn der Pfad zur »neuen« Datei nicht länger ist als der zur »alten«. Beachten Sie dies nicht, wird der erstellte *diff* nicht von *patch* eingespielt.

6.2.5 Updates sicher konfigurieren

Zu den Nachteilen einer Paketverwaltung gehört, dass sie als *root* ausgeführt wird und somit Vollzugriff auf alle Dateien hat. So kann es vorkommen, dass beim Aktualisieren von Paketen vorhandene Konfigurationsdateien überschrieben werden. Ein weiterer Nachteil besteht darin, dass Abhängigkeiten von den Paketverwaltungen nur so weit verarbeitet werden können, wie diese von den Maintainern gepflegt wurden. Diese Umstände führen teilweise zu Störungen, deren Ursache nicht leicht auffindbar ist. In diesem Abschnitt erfahren Sie, wie Sie den *GAU* vermeiden können.

Der »hold«-Status

Leider kommt es immer wieder vor, dass die Weiterentwicklung von Software länger dauert als geplant. Dies kann zu Konflikten führen.

[»]

Die Entwicklung der Software *Alpha* steht still. Hingegen wird die Software *Beta*, von der *Alpha* abhängt, stetig weiterentwickelt. Wenn nun »alte« Funktionen aus der Software *Beta* nicht weiter in die »neuen« Versionen übernommen werden, kann die Software *Alpha* nicht mehr ordnungsgemäß arbeiten.

Eine Paketverwaltung löst solche Probleme meist auf und unterbindet das weitere Einspielen von Updates für die entsprechende Software. Leider kommt es vor, dass Entwickler keine Versionsangaben für abhängige Software einpflegen. Dann kommt es zum *GAU*. Um diesen zu verhindern, haben Sie die Möglichkeit, einzelne Pakete aus den Updates herauszulösen, sodass die von Ihnen benötigte Software zwar im veralteten, aber dafür lauffähigen Zustand bleibt. Hierfür wurde in den *deb*-Paketverwaltungen der *hold*-Status und unter der *rpm*-Paketverwaltung der *lock*-Status implementiert.

Debian/Ubuntu

Unter *deb*-basierten Distributionen können Sie den Status von Paketen mittels `dpkg` und dem Parameter `--set-selections` ändern. So setzen Sie das Paket *vsftpd* auf `hold`:

```
root@debian:~# echo "vsftpd hold" | dpkg --set-selections
```

Listing 6.32 »deb«-Paket auf »hold« setzen

openSUSE Leap

Den Status eines *rpm*-Pakets können Sie mit `zypper addlock <PAKET>` sperren.

CentOS

Unter CentOS müssen Sie zunächst das Paket *yum-plugin-versionlock* installieren. Anschließend können Sie mit dem Befehl `yum versionlock <PAKET>` ein Paket sperren.

Übersicht der gesperrten Pakete

Eine Übersicht der in *deb*-Systemen auf *hold* gesetzten Pakete erhalten Sie über den folgenden Befehl:

```
root@debian:~# dpkg --get-selections | grep hold
vsftpd                                          hold
```

Listing 6.33 Übersicht gesperrter »deb«-Pakete

Ebenso können Sie sich auf *rpm*-Systemen die als gesperrt markierten Pakete anzeigen lassen. Verwenden Sie dafür `zypper` auf openSUSE Leap und `yum` auf CentOS:

```
### openSUSE Leap
daniel@leap:~> zypper locks
# | Name   | Typ     | Repository
--+--------+---------+-----------
1 | vsftpd | package | (beliebig)

### CentOS
[daniel@centos ~]$ yum versionlock
Geladene Plugins: fastestmirror, versionlock
```

```
0:vsftpd-3.0.2-11.el7_2.*
versionlock list done
```

Listing 6.34 Übersicht gesperrter »rpm«-Pakete

Konfigurationsdateien

Prüfen Sie die aktuellen Paketverwaltungen, ob Veränderungen an vorhandenen Konfigurationsdateien vorgenommen wurden, und legen Sie entsprechend Sicherungskopien an, sodass bei einem Update Ihre »alte« Konfiguration nicht blind überschrieben wird.

»rpm«

Bei der Installation untersucht *rpm*, ob es Änderungen an der Konfigurationsdatei gibt. Je nachdem, wie das Paket erstellt wurde, wird Ihre aktuelle Konfigurationsdatei ersetzt und eine Sicherheitskopie erzeugt (Suffix: *.rpmsave* oder *.rpmorig*), oder Ihre Konfigurationsdatei bleibt bestehen, und die »neue« Konfigurationsdatei wird mit dem Suffix *.rpmnew* angelegt.

[+]

».rpmsave«, ».rpmorig« und ».rpmnew«

Kontrollieren Sie nach »großen« Updates, ob diese Dateien existieren, und passen Sie gegebenenfalls Ihre Konfiguration an!

»deb«

Bei »deb«-basierten Systemen wird ein anderer Ansatz verfolgt. Hier wird ebenfalls geprüft, ob eine lokale Konfigurationsdatei existiert.

Wird mit dem »neuen« Paket eine Konfigurationsdatei ausgeliefert, wird Ihnen die Wahl gelassen:

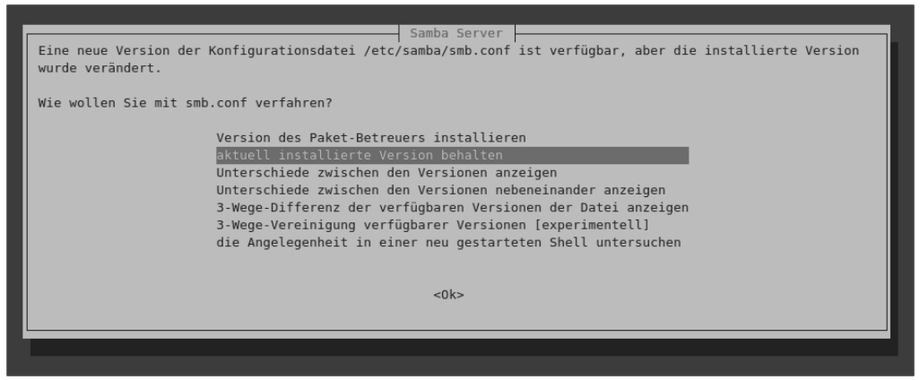


Abbildung 6.1 Auswahl zur Änderung der Konfigurationsdatei »samba«

6.3 Updates nur einmal laden: »Cache«

Bandbreite ist Zeit, und Zeit ist kostbar. Es ist ärgerlich, wenn ein und dasselbe Update von zehn (oder mehr) Ihrer Server Ihre Internetleitung minutenlang bis zum Anschlag füllt. Vor allem dann, wenn es doch ausreichen würde, das Update einmal aus dem Internet herunterzuladen und in Ihrem Netzwerk zu verteilen!

6.3.1 »deb«-basierte Distributionen: »apt-cacher-ng«

Der *apt-cacher-ng* stellt umfangreiche Funktionen zur Verfügung. Mit ihm können Sie einen effizienten Cache betreiben, er lässt multiple Zugriffe zu und bietet Ihnen auch noch Auswertungsmöglichkeiten.

6.3.2 Installation

Installieren Sie das Paket wie gewohnt mit `apt install apt-cacher-ng` aus den Paketquellen. Die benötigten Abhängigkeiten werden automatisch mit installiert. Auf Ubuntu-Systemen muss dafür die *universe*-Paketquelle aktiviert sein.

6.3.3 Konfiguration

Nach der Installation des Pakets *apt-cacher-ng* finden Sie unter */etc/apt-cacher-ng* die gut dokumentierte Konfigurationsdatei *acng.conf* des Dienstes. In der Standardkonfiguration sind bereits einige Parameter gesetzt. Listing 6.35 zeigt den Inhalt der Datei ohne die beschreibenden Kommentarzeilen:

```
CacheDir: /var/cache/apt-cacher-ng
LogDir: /var/log/apt-cacher-ng
SupportDir: /usr/lib/apt-cacher-ng
Remap-debrep: file:deb_mirror*.gz /debian ; file:backends_debian # Debian […]
Remap-uburep: file:ubuntu_mirrors /ubuntu ; file:backends_ubuntu # Ubuntu […]
Remap-cygwin: file:cygwin_mirrors /cygwin # ; file:backends_cygwin # incom[…]
Remap-sfnet:  file:sfnet_mirrors # ; file:backends_sfnet # incomplete, ple[…]
Remap-alxrep: file:archlx_mirrors /archlinux # ; file:backend_archlx # Arc[…]
Remap-fedora: file:fedora_mirrors # Fedora Linux
Remap-epel:   file:epel_mirrors # Fedora EPEL
Remap-slrep:  file:sl_mirrors # Scientific Linux
Remap-gentoo: file:gentoo_mirrors.gz /gentoo ; file:backends_gentoo # Gento[…]
ReportPage: acng-report.html
ExThreshold: 4
LocalDirs: acng-doc /usr/share/doc/apt-cacher-ng
```

Listing 6.35 Auszug der Standardwerte: »acng.conf«

Dabei sind folgende dienstspezifische Parameter konfiguriert:

- **CacheDir**
  Verzeichnis, in dem der Cache abgelegt wird
- **LogDir**
  Verzeichnis, in dem die Log-Dateien abgelegt werden
- **Port**
  Definition des TCP-Ports, auf dem der Dienst lauscht (Standard: 3142)
- **ExTreshold**
  Angabe in Tagen, ab wann ein nicht referenziertes Paket als abgelaufen betrachtet wird

Zusätzlich können Sie auch nachstehende Parameter konfigurieren:

- **VerboseLog**, **Debug**, **ForeGround**
  Parameter zur Fehleranalyse
- **Offlinemode**
  steuert das Verhalten, wenn keine Internetverbindung existiert
- **StupidFs**
  steuert zusätzliche Maßnahmen, wenn der Cache auf einem NTFS- oder FAT-Dateisystem betrieben wird
- **Proxy**
  Darüber kann ein übergeordneter Proxy-Server angegeben werden, falls Ihr Server nicht direkt mit dem Internet kommunizieren kann.

Neben diesen dienstspezifischen Konfigurationen findet das *Remapping* über den Parameter `Remap-` statt. In jeder Zeile wird definiert, für welche Anfragen eine entsprechende Datei (Angabe nach dem Parameter `file`) geladen werden soll. In diesem Zusammenhang spricht man auch von *Backends*.

Wie Sie Listing 6.35 entnehmen können, sind neben den Ubuntu-Quellen auch direkt Debian-, cygwin-, Arch-Linux-, Fedora- und weitere Quellen definiert. Zu der reinen Konfiguration des *Remappings* gehört aber noch eine jeweilige Datei, in der das eigentliche Mapping stattfindet. Im Standard finden Sie bereits für Debian und Ubuntu entsprechende *Backends* unter */etc/apt-cacher-ng*.

Schauen wir uns diese für Ubuntu mal genauer an, wie in Listing 6.36 dargestellt:

```
http://de.archive.ubuntu.com/ubuntu/
```

Listing 6.36 Standardwerte: »backends_ubuntu«

Darüber wird definiert, wie die Anfragen der Clients übersetzt werden sollen. Die Standardkonfiguration ist sofort einsetzbar und muss nicht angepasst werden. Falls Sie Veränderungen vorgenommen haben, müssen Sie diese dem *apt-cacher-ng* über ein Neuladen der

Konfiguration bekannt machen. Das Neuladen können Sie mittels `systemctl reload apt-cacher-ng` durchführen.

6.3.4 Clientkonfiguration

Die Clientkonfiguration geht schnell von der Hand, da nur eine Datei editiert werden muss. Passen Sie bei den Clients die Konfigurationsdatei */etc/apt/apt.conf.d/01proxy* an, ändern Sie diese, oder erstellen Sie sie, wie in Listing 6.37 dargestellt:

```
Acquire::http::Proxy "http://<HOSTNAME oder IP des CACHE>:3142";
```

Listing 6.37 Clientkonfiguration: »01proxy«

Da kein Dienst im Hintergrund läuft, ist keine weitere Aktion notwendig. Alle zukünftigen Downloads werden nun über den *apt-cacher-ng* geladen.

6.3.5 Fütterungszeit – bereits geladene Pakete dem Cache hinzufügen

Falls Sie den *apt-cacher-ng* auf einem System betreiben, das bereits Updates geladen hat, können Sie diese Ihrem frischen Cache hinzufügen. So können Sie bereits die erste Bandbreitenersparnis generieren. Dafür müssen folgende Schritte durchlaufen werden:

1. **Lokale Proxy-Konfiguration**
   Zunächst müssen Sie dem Server mitteilen, dass er ebenfalls über den *apt-cacher-ng* Updates laden soll (siehe Abschnitt 6.3.4, »Clientkonfiguration«).
2. **Paketlisten laden**
   Laden Sie nun die aktuellen Paketlisten herunter. Darüber werden dem *apt-cacher-ng* die bereits vorhandenen Pakete bekannt gemacht:
   ```
   apt update
   ```
3. **Aufräumen der Pakete**
   Nun sollten Sie zunächst die benötigten Pakete von Ihrem Server entfernen. Nutzen Sie dafür folgenden Befehl:
   ```
   apt autoclean
   ```
4. **Verzeichnis »_import« anlegen**
   Erstellen Sie das Verzeichnis *_import* mit dem folgenden Befehl:
   ```
   mkdir -p /var/cache/apt-cacher-ng/_import
   ```
5. **Rechte anpassen**
   Passen Sie die Rechte an dem soeben erstellten Verzeichnis an, damit der *apt-cacher-ng* dieses auch benutzen kann. Setzen Sie dafür den nachstehenden Befehl ab:
   ```
   chown apt-cacher-ng /var/cache/apt-cacher-ng/_import
   ```

6. **Kopieren der Pakete**
   Kopieren Sie nun die lokalen Pakete mit folgendem Befehl in das Importverzeichnis:
   ```
   cp -al /var/cache/apt/archives/* /var/cache/apt-cacher-ng/_import/
   ```
7. **Importvorgang starten**
   Sie können den Importvorgang über den Button START IMPORT auf dem Webinterface von *apt-cacher-ng* (*http://localhost:3142/acng-report.html*) starten (siehe Abbildung 6.2).
8. **Aufräumen**
   Nach dem erfolgreichen Import wird das Verzeichnis *_import* nicht länger benötigt. Entfernen Sie es mit folgendem Befehl:
   ```
   rm -rf /var/cache/apt-cacher-ng/_import
   ```

Abbildung 6.2 Import über das Webinterface

Herzlichen Glückwunsch, Sie haben soeben die erste Bandbreitenersparnis erzeugt! Diesen Vorgang müssen Sie nur einmal umsetzen. Sobald Ihre Server den *apt-cacher-ng* verwenden, füllt sich der Cache automatisch.

6.3.6 Details: »Report-HTML«

Wenn der Dienst gestartet ist, können Sie über die URL *http://<IP>:3142/acng-report.html* Statusinformationen abrufen, wie in Abbildung 6.3 dargestellt. Beachten Sie dabei, dass Sie `<IP>` durch die IP-Adresse Ihres Servers ersetzen. Auf der Webseite können Sie sogar kleine Konfigurationen vornehmen. Einen Blick ist sie allemal wert.

Transfer statistics

Period	Cache efficiency					
	Requests			Data		
	Hits	Misses	Total	Hits	Misses	Total
2016-05-20 14:16 - 2016-05-21 14:16	22 (18.64%)	96 (81.36%)	118	0.00 MiB (0.00%)	26.81 MiB (100.00%)	26.81 MiB

Note: data table is created based on the current log file. Deviation from real request count is possible due to previous log file optimization.

Abbildung 6.3 Report-HTML: »apt-cacher-ng«

6.3.7 »rpm«-basierte Distributionen

Dieser Abschnitt fällt äußerst kurz aus. Leider gibt es für *rpm*-basierte Distributionen keine angepasste Software, die einen Repository-Cache bereitstellt. Selbstverständlich können Sie einen Squid oder Nginx dafür missbrauchen. Diesen fehlen allerdings die notwendigen Fähigkeiten, um mit Paketen anständig umzugehen. Daher empfehlen wir Ihnen, darauf zu verzichten.

6.4 Alles meins: »Mirror«

Im Gegensatz zum Proxy oder Cache wird bei einem Mirror das gesamte Repository gespiegelt. Dadurch können Sie ein vollständiges System installieren, ohne auch nur ein Paket aus dem Internet zu laden. Allerdings benötigt ein vollständiger Spiegel auch sehr viel Plattenplatz. Pro Sektion können schon mal 20 GB zusammenkommen. Achten Sie also darauf, dass Ihr System genügend Platz zur Verfügung hat.

6.4.1 »deb«-basierte Distributionen: »debmirror«

Installieren Sie folgende Pakete:

- *debmirror*
- *ubuntu-keyring* (auf Ubuntu) oder *debian-keyring* (auf Debian)
- *apache2*

Die benötigten Abhängigkeiten werden automatisch mit installiert. Im Beispiel verwenden wir ein Ubuntu-System, passen Sie daher gegebenenfalls die Befehle Ihrer Umgebung an.

6.4.2 Konfiguration

Da *debmirror* kein eigener Dienst ist, sondern nur ein Programm zur Verfügung stellt, müssen Sie die Struktur selbst aufbauen. Dazu gehören folgende Arbeitsschritte:

1. Benutzer und Gruppe anlegen
2. Verzeichnisstruktur anlegen
3. Mirror-Skript erstellen (Ubuntu)
4. Cronjobs einrichten
5. Schlüssel importieren
6. Mirror erstellen
7. Mirror verfügbar machen – Webdienst konfigurieren
8. Clientkonfiguration

6.4.3 Benutzer und Gruppe anlegen

Erstellen Sie zunächst einen eigenen Benutzer, der für den Spiegel zuständig ist. Das hat den Vorteil, dass die Erzeugung des Spiegels nicht mit root-Rechten durchgeführt werden muss.

```
daniel@server:/# sudo groupadd mirror
daniel@server:/# sudo useradd -g mirror -d /var/www/mirror -s /bin/bash -m \
-c "Ubuntu Mirror" mirror
```

Listing 6.38 Benutzer und Gruppe »mirror« anlegen

In Listing 6.38 wird zunächst die Gruppe *mirror* angelegt. Anschließend wird der Benutzer *mirror* mit `useradd` erzeugt, die Parameter haben dabei nachstehende Bedeutung:

- `-g` = weist den Benutzer einer Gruppe zu, hier *mirror*.
- `-d` = Das Homeverzeichnis des Benutzers wird auf die nachstehende Pfadangabe geändert, hier */var/www/mirror*. Da der Benutzer nur als Dienstgeber fungiert, sollte der Spiegel auch unter */var/* liegen und nicht unter */home/*.
- `-s` = setzt die Shell des Benutzers auf die Bash.
- `-m` = Falls das angegebene Homeverzeichnis nicht existiert, wird es mit angelegt.
- `-c` = Der Benutzer wird mit einem Kommentar versehen, hier »Ubuntu Mirror«.
- `mirror` = gibt den Benutzernamen an, der erzeugt wird.

6.4.4 Verzeichnisstruktur anlegen

Nun können wir die Verzeichnisstruktur anlegen. Dazu wechseln wir mittels *su* zum neu angelegten Benutzer und erstellen die benötigten Verzeichnisse unter seinem Namen, wie es Listing 6.39 darstellt:

```
daniel@server:/# sudo su - mirror
mirror@server:/# mkdir -p ~/bin ~/ubuntu ~/ubuntu-security ~/ubuntu-updates \
~/ubuntu-backports
```

Listing 6.39 Verzeichnisse anlegen

Da die Verzeichnisse direkt als Benutzer *mirror* erzeugt wurden, sind die Rechte bereits korrekt gesetzt.

6.4.5 Mirror-Skript erstellen (Ubuntu)

Damit der Spiegel erstellt und regelmäßig aktualisiert wird, empfiehlt sich der Einsatz eines Skripts, das über den *cron* zyklisch gestartet wird. In unserem Beispiel erzeugen wir Skripte für das Spiegeln von Ubuntu.

Dazu erstellen wir ein zentrales Skript, das mit einem Parameter aufgerufen werden muss. Der Parameter gibt das zu spiegelnde Repository an. Erstellen Sie also zunächst die Datei */var/www/mirror/bin/mirror-ubuntu.sh* mit dem Inhalt aus Listing 6.40, und geben Sie dem Skript mit `chmod +x mirror-ubuntu.sh` Ausführungsrechte:

```
 1: #!/bin/bash
 2: #
 3: # Mirror Ubuntu-Repostiorys
 4: #
 5: # VARS
 6: #  $1  sets the repository name
 7:
 8: REPO=$1
 9: if [ -z $REPO ] ; then
10:  logger -t mirror-$REPO[$$] ERROR no repository name given!
11:  exit 1
12: fi
13:
14: logger -t mirror-$REPO[$$] Updating $REPO
15:
16: debmirror \
17:  --passive \
18:  --progress \
19:  --nosource \
20:  --host=de.archive.ubuntu.com \
21:  -e http \
22:  --root=ubuntu \
23:  --dist=bionic,xenial \
24:  --section=multiverse,universe,restricted,main \
25:  --arch=i386,amd64 \
26:  --verbose \
27:  /var/www/mirror/$REPO
28:
29: logger -t mirror-$REPO[$$] Finished Updating Ubuntu
```

Listing 6.40 Skript zum Laden der Spiegel: »mirror-ubuntu.sh«

Da dieses Skript zugegebenermaßen doch etwas umfangreicher ist, schauen wir uns die einzelnen Zeilen nun im Detail an:

- **Zeile 1**
  Der *Shebang* des Skripts gibt an, mit welcher Sprache es interpretiert werden soll.
- **Zeile 2–7**
  Kommentare, die das Skript und dessen Variablen beschreiben

- **Zeile 8**
  weist der Variablen `REPO` den ersten übergebenen Parameter (`$1`) zu.
- **Zeile 9–12**
  prüft, ob ein Repository-Name übergeben wurde. Wenn nicht, wird mit einer Fehlermeldung abgebrochen.
- **Zeile 14**
  protokolliert im *Syslog*, dass das Update begonnen hat.
- **Zeile 16–27**
  führt das Update aus (die Details erläutern wir im Anschluss).
- **Zeile 29**
  protokolliert im *Syslog*, dass das Update beendet wurde.

Der in Zeile 16–27 von Listing 6.40 dargestellte Aufruf von `debmirror` enthält viele Parameter, die wir uns nun genauer ansehen:

- `--passive`
  setzt den Download-Modus von FTP auf passiv.
- `--progress`
  stellt einen Fortschrittsbalken beim Download dar. Dies ist nur relevant, wenn Sie das Skript von Hand auf einer Konsole ausführen.
- `--nosource`
  gibt an, dass keine Quellen geladen werden sollen, sondern nur Binärpakete.
- `--host`
  gibt den Server an, von dem der Spiegel geladen werden soll. Wählen Sie stets einen Server, der eine gute Verfügbarkeit hat und in Ihrer Nähe ist. Im Beispiel wurde *de.archive.ubuntu.com* verwendet, was eine gute Wahl ist, da sich dahinter stets mehrere verfügbare und vollständige Repositorys verbergen.
- `-e`
  gibt das Übertragungsprotokoll an. Im Beispiel haben wir *http* verwendet.
- `--root`
  setzt das Wurzelverzeichnis auf dem Quellserver, in dem die Ubuntu-Archive liegen. Wenn nicht anders angegeben, wird im Standard stets `ubuntu` verwendet.
- `--dist`
  spezifiziert durch Komma getrennt, welche Distributionen geladen werden sollen. Das Beispielskript in Listing 6.40 lädt die LTS-Versionen Bionic und Xenial. Falls Sie ausschließlich die Version 16.04 im Einsatz haben, können Sie den Parameter entsprechend anpassen.
- `--section`
  gibt die Sektionen an, die geladen werden sollen.

- `--arch`
  spezifiziert durch Komma getrennt die Architektur. Falls Sie nur 64-Bit-Systeme betreiben, genügt es auch, nur diese Pakete zu spiegeln.
- `--verbose`
  erweitert die Ausgabe.
- `/var/www/mirror/$REPO`
  gibt den lokalen Speicherort des jeweiligen Spiegels an. Über den Parameter `$REPO` wird also auch das Verzeichnis angegeben. Soll zum Beispiel das *ubuntu-security*-Repository gespiegelt werden, dann finden Sie die Daten unter */var/www/mirror/ubuntu-security*.

Bei wenig Plattenplatz: »precleanup«

Falls Ihr Spiegelsystem nicht genügend Plattenplatz hat, können Sie dem Programm `debmirror` den Parameter `--precleanup` hinzufügen. Dieser sorgt dafür, dass vor dem Download Ihr lokaler Spiegel aufgeräumt wird. Die Entwickler warnen aber davor, dass dies zu einem inkonsistenten Spiegel führen kann. Setzen Sie diesen Parameter also nur im Notfall ein.

6.4.6 Cronjobs einrichten

Damit Sie nicht ständig alle Updates von Hand anstoßen müssen, können Sie mehrere Cronjobs einrichten, die das für Sie erledigen. Vorzugsweise sollte der Spiegel in der Nacht erzeugt werden, da in der Regel dann niemand Ihre Internetleitung benötigt.

Erzeugen Sie also für den Benutzer *mirror* die notwendigen Cronjobs. Dazu benutzen wir das Tool `crontab` so, wie in Listing 6.41 dargestellt:

```
daniel@server:/# sudo crontab -e -u mirror
```

Listing 6.41 Cronjobs des Benutzers »mirror« öffnen

Fügen Sie anschließend die Zeilen aus Listing 6.42 in den sich öffnenden Editor ein:

```
0 1 * * * bash -l -c "/var/www/mirror/bin/mirror-ubuntu.sh ubuntu"
0 2 * * * bash -l -c "/var/www/mirror/bin/mirror-ubuntu.sh ubuntu-security"
0 3 * * * bash -l -c "/var/www/mirror/bin/mirror-ubuntu.sh ubuntu-updates"
0 4 * * * bash -l -c "/var/www/mirror/bin/mirror-ubuntu.sh ubuntu-backports"
```

Listing 6.42 Cronkonfiguration

Damit geben Sie an, dass jede Nacht ab 01:00 Uhr stündlich die Updates laufen sollen. Über die Parameter nach dem Skript `mirror-ubuntu.sh` definieren Sie, welche Repositorys geladen werden sollen. Falls Sie also keine Pakete aus den *Backports* benötigen, können Sie die entsprechende Zeile einfach entfernen.

6.4.7 Schlüssel importieren

Damit der angelegte Benutzer *mirror* den Spiegel erstellen und aktualisieren kann, benötigt er die GPG-Schlüssel der Repositorys. Diese können wir einfach aus dem System importieren, wie in Listing 6.43 dargestellt:

```
mirror@server:~$ gpg --no-default-keyring --keyring trustedkeys.gpg \
--import /usr/share/keyrings/ubuntu-archive-keyring.gpg

gpg: Verzeichnis `/var/www/mirror/.gnupg' erzeugt
gpg: Die "Keybox" `/var/www/mirror/.gnupg/trustedkeys.gpg' wurde erstellt
gpg: key 3B4[…]F32: 3 Beglaubigungen wegen fehlender Schlüssel nicht geprüft
gpg: /var/www/mirror/.gnupg/trustdb.gpg: trust-db erzeugt
gpg: Schlüssel 3B4[…]F32: Öffentlicher Schlüssel "Ubuntu Archive Automatic \
     Signing Key (2012) <ftpmaster@ubuntu.com>" importiert
gpg: key D94[…]092: 3 Beglaubigungen wegen fehlender Schlüssel nicht geprüft
gpg: Schlüssel D94[…]092: Öffentlicher Schlüssel "Ubuntu CD Image Automatic \
     Signing Key (2012) <cdimage@ubuntu.com>" importiert
gpg: Anzahl insgesamt bearbeiteter Schlüssel: 2
gpg:               importiert: 2
gpg: keine ultimativ vertrauenswürdigen Schlüssel gefunden
mirror@server:~$
```

Listing 6.43 GPG-Schlüssel für »mirror« importieren

Stolperfalle: richtiger Benutzer

Achten Sie darauf, dass der Import als Benutzer *mirror* durchgeführt wird! Ansonsten kann das System die GPG-Schlüssel nicht finden und bricht mit einer entsprechenden Fehlermeldung ab.

6.4.8 Mirror erstellen

Um zu prüfen, ob unser Skript auch funktionstüchtig ist, können wir es einfach ausführen. Fehler werden entsprechend in der Bash ausgegeben.

Falls Sie nicht direkt den Download starten wollen, können Sie temporär dem Befehl `debmirror` den zusätzlichen Parameter `--dry-run` mitgeben. Damit läuft das gesamte Update, aber ohne Pakete herunterzuladen.

6.4.9 Mirror verfügbar machen – Webdienst konfigurieren

Damit Ihr Spiegel auch von Ihren Clients benutzt werden kann, müssen Sie diese verfügbar machen. In der Regel wird hierfür als Übertragungsprotokoll HTTP eingesetzt.

Falls Sie bereits einen Apache auf dem System betreiben, müssen Sie lediglich einen weiteren virtuellen Host anlegen und diesen so, wie in Listing 6.44 beschrieben, einrichten:

```
<VirtualHost *:80>
  DocumentRoot /var/www/mirror/
  <Directory "/var/www/mirror/">
    Options Indexes FollowSymLinks
    AllowOverride All
    Require all granted
   </Directory>
</VirtualHost>
```

Listing 6.44 Apache-Konfiguration des Spiegels: »/etc/apache2/sites-available/mirror.conf«

Abschließend müssen Sie den neuen virtuellen Host noch aktivieren und den Apache so durchstarten, wie in Listing 6.45 dargestellt:

```
daniel@server:/# sudo a2ensite mirror
[…]
daniel@server:/# sudo systemctl reload apache2
```

Listing 6.45 Host aktivieren und Apache neu laden

6.4.10 Clientkonfiguration

Damit Ihre Server von nun an über Ihren eigenen Spiegel Updates laden, müssen Sie ihnen dieses mitteilen. Dafür müssen Sie die lokale Datei */etc/apt/sources.list* anpassen.

Speichern Sie die aktuelle Version dieser Datei unter einem anderen Namen, und legen Sie eine neue Version mit dem Inhalt aus Listing 6.46 an:

```
deb http://<IP>/mirror/ubuntu bionic universe multiverse main restricted
deb http://<IP>/mirror/ubuntu-security bionic-security universe main \
 restricted multiverse

deb http://<IP>/mirror/ubuntu-updates bionic-updates main restricted \
 universe multiverse
deb http://<IP>/mirror/ubuntu-backports bionic-backports main restricted \
 universe multiverse
```

Listing 6.46 Clientkonfiguration: »sources.list«

Release beachten!

Beachten Sie dabei, dass Sie das für das System zutreffende Release eintragen. Also `biconic` für Ubuntu 18.04 bzw. `xenial` für Ubuntu 16.04.

Ersetzen Sie `<IP>` durch die IP-Adresse oder den Hostnamen des Systems, auf dem Sie den Spiegel betreiben. Beachten Sie, dass die Zeilen aus Listing 6.46 aufgrund ihrer Länge umbrochen wurden, aber eigentlich in eine Zeile gehören. Anschließend werden alle Updates von Ihrem eigenen Mirror geladen.

6.4.11 rpm-basierte Distributionen

Um einen Spiegel einer *rmp*-basierten Distribution zu erstellen, wird kein gesondertes Softwarepaket benötigt. Die Spiegelung erfolgt einfach mittels *rsync*.

Im Beispiel werden wir einen Mirror für openSUSE einrichten – Abweichungen zu CentOS stellen wir entsprechend dar. Nachstehende Punkte müssen dafür abgearbeitet werden:

1. Benutzer und Gruppe anlegen
2. Verzeichnisstruktur anlegen
3. Mirror-Skript erstellen
4. Cronjobs einrichten
5. Mirror erstellen
6. Mirror verfügbar machen – Webdienst konfigurieren
7. Clientkonfiguration

Diese Schritte werden wir nun nacheinander abarbeiten.

6.4.12 Benutzer und Gruppe anlegen

Zunächst sollte ein Benutzer für den Spiegelvorgang eingerichtet werden:

```
leap:/ # groupadd mirror
leap:/ # useradd -g mirror -d /srv/pub/opensuse -m -c "Mirror User" mirror
leap:/ # chown -R mirror:mirror /srv/pub/opensuse
```

Listing 6.47 Benutzer und Gruppe »mirror« anlegen

Der erste Befehl aus Listing 6.47 legt die Gruppe *mirror* an. Im zweiten Befehl wird ein gleichnamiger Benutzer angelegt. Dieser wird direkt mit dem Parameter `-g` der Gruppe hinzugefügt. Mit dem Parameter `-d` wird das Heimatverzeichnis auf */srv/pub/opensuse* festgelegt. Falls es nicht existiert, wird es durch die Angabe von `-m` direkt mit angelegt. Zu guter Letzt wird über den Parameter `-c` noch ein Kommentar für den Benutzer angegeben.

Abschließend werden dem angelegten Benutzer Rechte am erstellten Verzeichnis zugewiesen. Wenn Sie CentOS einsetzen, sollten Sie als Verzeichnis natürlich besser */srv/pub/CentOS* verwenden.

6.4.13 Verzeichnisstruktur anlegen: »openSUSE Leap«

Der Mirror soll unter */srv/pub/opensuse* abgelegt werden. Dafür müssen die entsprechenden Verzeichnisse erstellt und dem neu angelegten Benutzer zugeordnet werden. Setzen Sie dafür die Befehle aus Listing 6.49 ab:

```
root@server:/# mkdir -p /srv/pub/opensuse/update
```

Listing 6.48 openSUSE Leap: Verzeichnisse anlegen

6.4.14 Verzeichnisstruktur anlegen: »CentOS«

Bei CentOS soll der Mirror unter */srv/pub/CentOS* abgelegt werden. Dafür müssen Sie die Verzeichnisse so erstellen, wie in Listing 6.49 gezeigt:

```
[root@centos /]# mkdir -p /srv/pub/CentOS/7.5.1804
[root@centos /]# cd /srv/pub/CentOS
[root@centos CentOS]# ln -s 7.5.1804 7
[root@centos CentOS]# cd 7.5.1804
[…]# mkdir -p {addons,centosplus,contrib,cr,extras,fasttrack,isos,os,updates}
[…]# for i in $(ls); do mkdir $i/{i386,x86_64}; done
```

Listing 6.49 CentOS: Verzeichnisse anlegen

6.4.15 Mirror-Skript erstellen

Damit Sie nicht ständig die gleichen Befehle absetzen müssen, erstellen wir ein kleines Skript, das uns die Arbeit abnimmt. Erstellen Sie dafür mit root-Rechten die Datei */usr/bin/mirror.sh* mit den Zeilen aus Listing 6.50:

```
#!/bin/bash
SERVER=$1
MIRROR=$2
if [ -z "$1" ] ; then echo "need server"; exit 1; fi
if [ -z "$2" ] ; then echo "need mirror path"; exit 1; fi
mkdir -p $MIRROR 2>&1
echo -e "\nStarting download process..."
rsync -av --delete --progress --delay-updates --timeout=300 ${SERVER} ${MIRROR}
```

Listing 6.50 Mirror-Skript: »/usr/bin/mirror.sh«

Das Skript erwartet zwei Parameter: als Erstes den Server, von dem die Pakete geladen werden, und als Zweites den Pfad zu Ihrem lokalen Spiegel.

Das Skript prüft zunächst, ob die benötigten Parameter übergeben wurden. Anschließend wird, falls noch nicht vorhanden, der lokale Pfad angelegt. Zuletzt startet das Skript den Synchronisierungsprozess mit `rsync`. Dies sehen wir uns nun genauer an:

- `-av`
  aktiviert den *archive*- und *verbose*-Modus. Dabei wird sichergestellt, dass symbolische Links sowie die Attribute und Rechte der Ursprungsdateien erhalten bleiben. Gleichzeitig erhalten Sie eine umfangreichere Ausgabe.
- `--delete`
  Hierüber wird *rsync* angewiesen, unbenötigte Dateien zu entfernen.
- `--progress`
  zeigt einen Fortschrittsbalken des Vorgangs an.
- `--delay-updates`
  stellt sicher, dass die neuen Dateien erst nach dem Ende der Verarbeitung in das Zielverzeichnis kopiert werden – so ist sichergestellt, dass keine Inkonsistenzen während des Herunterladens entstehen.
- `--timeout 300`
  setzt die maximale Zeitspanne, in der auf eine Transaktion gewartet wird, auf 300 Sekunden – also fünf Minuten.

Damit das Skript ausgeführt werden kann, müssen dessen Rechte angepasst werden. Führen Sie dazu abschließend den Befehl aus Listing 6.51 aus:

```
daniel@server:~> sudo chmod +x /usr/bin/mirror.sh
```

Listing 6.51 Rechtekorrektur am Mirror-Skript

6.4.16 Cronjobs einrichten

Zur automatisierten Aktualisierung des Spiegels wird das soeben erstellte Skript nun in Cronjobs gepackt. Öffnen Sie den Cron-Editor für den Benutzer *mirror* mit dem Befehl aus Listing 6.52:

```
daniel@server:~> sudo crontab -u mirror -e
```

Listing 6.52 Cronjobs des Benutzers »mirror« bearbeiten

Fügen Sie nun im sich öffnenden Editor die Zeilen aus Listing 6.53 ein. Beachten Sie, dass die Befehle umbrochen wurden und eigentlich in einer Zeile stehen.

```
0 1 * * *    sleep $(($RANDOM/64)); /usr/bin/mirror.sh \
             "ftp.gwdg.de::pub/opensuse/distribution/leap/15.0/repo/oss/" \
             "/srv/pub/opensuse/leap/15.0/oss/"

0 3 * * *    sleep $(($RANDOM/64)); /usr/bin/mirror.sh \
             "ftp.gwdg.de::pub/opensuse/distribution/leap/15.0/repo/non-oss/" \
             "/srv/pub/opensuse/leap/15.0/non-oss/"
```

Listing 6.53 Cronjobs definieren für openSUSE Leap

Jede Nacht wird um 01:00 Uhr der Spiegel *oss* und um 03:00 Uhr der Spiegel *non-oss* geladen. Damit dieser Vorgang nicht immer strikt zu den angegebenen Uhrzeiten gestartet wird, wurde mithilfe des `sleep`-Kommandos eine Variabilität hinzugefügt. Der Befehl bewirkt, dass immer eine zufällige Anzahl an Sekunden gewartet wird, bevor der eigentliche Job gestartet wird.

Wie bereits erörtert wurde, erwartet das *mirror.sh*-Skript zwei Parameter. Im ersten Parameter steht der Server, von dem der Spiegel geladen wird. Wie Sie Listing 6.53 entnehmen können, wird der *gwdg.de*-Mirror-Server verwendet. Die Gemeinschaftseinrichtung der Universität Göttingen und der Max-Planck-Gesellschaft ist einer der bekanntesten openSUSE-Mirror-Anbieter. Der Server wurde in der Rsync-Syntax angegeben (zwei aufeinanderfolgende Doppelpunkte zur Trennung von Server und Pfad). Im zweiten Parameter wird der lokale Pfad angegeben, den wir zuvor erstellt haben.

Sie können den Serverpfad auch im Browser eingeben, um so einsehen zu können, welche Distributionen dort angeboten werden. Unter *ftp://ftp.gwdg.de/pub/opensuse* wird Ihnen der Inhalt angezeigt.

Anpassungen pro Spiegel

Beachten Sie, dass im Beispiel lediglich ein Spiegel von *openSUSE Leap 15.0* geladen wird. Wollen Sie eine andere oder mehrere Distributionen anbieten, müssen Sie die Pfade anpassen oder mehrere *mirror.sh*-Skriptzeilen erstellen!

Um einen CentOS-Mirror zu erstellen, muss ein Cronjob, wie in Listing 6.54 dargestellt, eingerichtet werden:

```
0 1 * * *    sleep $(($RANDOM/64)); /usr/bin/mirror.sh \
             "ftp.fau.de::centos/7.5.1804/" "/srv/pub/CentOS/7.5.1804/"
```

Listing 6.54 Cronjobs definieren für CentOS

6.4.17 Mirror erstellen

Selbstverständlich müssen Sie jetzt nicht eine Nacht warten, um zu sehen, ob der Mirror erstellt wird. Das Skript können Sie auch von der Konsole aus starten. Damit die Dateiberechtigungen korrekt gesetzt werden, sollten Sie das Skript als Benutzer *mirror* starten. Listing 6.55 zeigt die Ausgabe, die das Skript erzeugt:

```
daniel@leap:~> sudo su mirror /usr/bin/mirror.sh \
"ftp.gwdg.de::pub/opensuse/distribution/leap/15.0/repo/non-oss/" \
"/srv/pub/opensuse/leap/15.0/non-oss/"

Starting download process...
```

```
Welcome to ftp.gwdg.de

receiving file list ... 81 files to consider
./
.current.txt
             14 100%   13.67kB/s    0:00:00 (xfr#1, to-chk=79/81)
CHECKSUMS
            163 100%  159.18kB/s    0:00:00 (xfr#2, to-chk=78/81)
CHECKSUMS.asc
            481 100%  234.86kB/s    0:00:00 (xfr#3, to-chk=77/81)
[…]
```

Listing 6.55 Auszug: Ausgabe des Skripts »mirror.sh« auf openSUSE Leap

6.4.18 Mirror verfügbar machen – Webdienst konfigurieren

Damit Ihre Clients den erstellten Spiegel auch nutzen können, müssen Sie diesen verfügbar machen. Hierfür genügt eine Webserverinstallation – mehr zum Thema erfahren Sie in Kapitel 8, »Webserver«. Im Beispiel verwenden wir einen Apache-Webserver.

Richten Sie für den Mirror einen neuen *VirtualHost* ein. Dafür erstellen Sie die Datei */etc/apache2/vhosts.d/mirror.example.com.conf* mit dem Inhalt aus Listing 6.56:

```
<VirtualHost *:80>
   ServerAdmin admin@example.com
   ServerName mirror.example.com

   DocumentRoot "/srv/pub/opensuse"

   <Directory "/srv/pub/opensuse">
       Options FollowSymLinks Indexes
       IndexOptions FancyIndexing VersionSort NameWidth=* Charset=UTF-8 \
                    TrackModified FoldersFirst XHTML
       AllowOverride None
       Order allow,deny
       Allow from all
   </Directory>
</VirtualHost>
```

Listing 6.56 Konfiguration des VirtualHost: »/etc/apache2/vhosts.d/mirror.example.com.conf«

Selbstverständlich müssen Sie den Dateinamen und die Direktive `ServerName` Ihrer Umgebung anpassen. Nach den Änderungen müssen Sie den Dienst neu starten. Anschließend können Ihre Clients – vorausgesetzt, das *mirror.sh*-Skript ist bereits einmal vollständig gelaufen – Ihren neu geschaffenen Mirror-Server einsetzen.

6.4.19 Clientkonfiguration: »openSUSE Leap«

Wir bleiben im SUSE-Universum und zeigen Ihnen in diesem Abschnitt, wie Sie den Clients in Ihrem Netz den neu geschaffenen Mirror-Server mitteilen. Öffnen Sie dafür den *yast2*, und wählen Sie unter SOFTWARE den Unterpunkt SOFTWARE REPOSITORYS aus. Im sich öffnenden Dialog können Sie über die Schaltfläche ADD oder die Tastenkombination Alt + A ein neues Repository hinzufügen.

Wählen Sie im folgenden Dialog als Medien-Typ »HTTP...« aus, und klicken Sie auf die Schaltfläche NEXT. Nun werden Sie aufgefordert, einen Namen (*Repository Name*) und die URL anzugeben. Geben Sie hier den Namen und den Pfad Ihres Spiegelservers an, zum Beispiel *http://mirror.example.com/opensuse/15.0*.

Anschließend wird Ihr Spiegelserver durchsucht und analysiert. Nach Abschluss der Verarbeitung steht Ihnen der neue Server in der Liste der Software-Repositorys zur Verfügung. Damit der Server benutzt wird, müssen Sie diesen aktivieren und ihm eine niedrigere Priorität zuweisen. Alternativ können Sie auch die bisherigen Server deaktivieren oder entfernen.

6.4.20 Clientkonfiguration: »CentOS«

Damit CentOS-Systeme Ihren Mirror verwenden, müssen Sie die Datei *CentOS-Base.repo* im Verzeichnis */etc/yum.repos.d/* so anpassen, wie in Listing 6.57 dargestellt ist:

```
[base]
name=CentOS-$releasever - Base
baseurl=http://mirror.example.com/CentOS/$releasever/os/$basearch/
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7

#released updates
[updates]
name=CentOS-$releasever - Updates
baseurl=http://mirror.example.com/CentOS/$releasever/updates/$basearch/
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7

#additional packages that may be useful
[extras]
name=CentOS-$releasever - Extras
baseurl=http://mirror.example.com/CentOS/$releasever/extras/$basearch/
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7

#additional packages that extend functionality of existing packages
```

```
[centosplus]
name=CentOS-$releasever - Plus
baseurl=http://mirror.example.com/CentOS/$releasever/centosplus/$basearch/
gpgcheck=1
enabled=0
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7
```

Listing 6.57 Anpassungen in »/etc/yum.repos.d/CentOS-Base.repo«

Beachten Sie, dass Sie zwingend die Direktiven `mirrorlist` entfernen müssen und die URL (fett hervorgehoben) Ihren Gegebenheiten anpassen.

Auf einen Blick

Inhalt