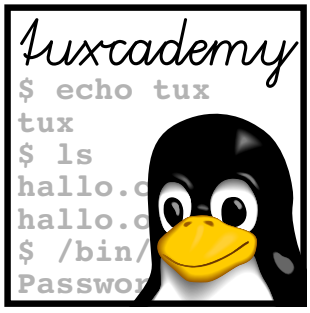
Linux Essentials

Die Einsteiger-Zertifizierung des LPI



tuxcademy – Linux- und Open-Source-Lernunterlagen für alle
www.tuxcademy.org · info@tuxcademy.org

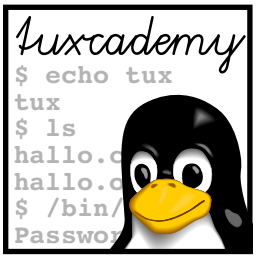
Das tuxcademy-Projekt bietet hochwertige frei verfügbare Schulungsunterlagen zu Linux- und Open-Source-Themen – zum Selbststudium, für Schule, Hochschule, Weiterbildung und Beruf.
Besuchen Sie https://www.tuxcademy.org/! Für Fragen und Anregungen stehen wir Ihnen gerne zur Verfügung.

Linux Essentials Die Einsteiger-Zertifizierung des LPI

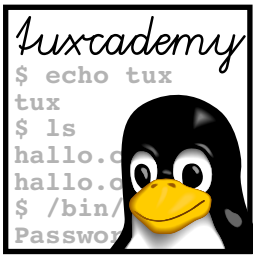
Revision: lxes:fc4046bd9e85427c:2015-08-08
adm1:9f8c99fa2fddd494:2015-08-08 13–14
grd1:a13e1ba7ab759bab:2015-08-04 4–11, B
lxes:7472be66ccb72646:2012-05-21 1–3, 12, 15
lxes:kSMh4rAlHsGj9Dk419037

Autoren: Tobias Elsner, Thomas Erker, Anselm Lingnau
Technische Redaktion: Anselm Lingnau ⟨*anselm.lingnau@linupfront.de*⟩

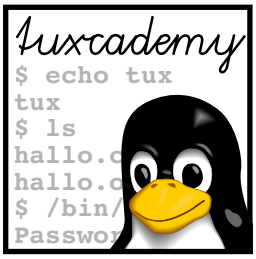
Gesetzt in Palatino, Optima und DejaVu Sans Mono



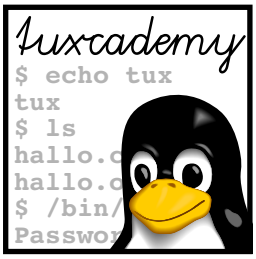
Inhalt



Tabellenverzeichnis



Abbildungsverzeichnis



Vorwort

Linux Essentials ist eine neue Zertifizierung des *Linux Professional Institute* (LPI), die sich besonders an Schulen und Hochschulen richtet, wo Jugendliche und junge Erwachsene an Linux herangeführt werden sollen. Das *Linux-Essentials*-Zertifikat soll das Basiswissen definieren, das erforderlich ist, um einen Linux-Rechner produktiv zu nutzen und über ein korrespondierendes Ausbildungsprogramm Jugendlichen, jungen Erwachsenen und Open-Source-Einsteigern dabei helfen, Linux und Open Source im Kontext der IT-Industrie zu verstehen. Mehr Informationen über die *Linux-Essentials*-Zertifizierung finden Sie im Anhang C.

Mit dieser Schulungsunterlage legt die Linup Front GmbH die erste umfassende Dokumentation zur Vorbereitung auf *Linux Essentials* vor. Die Unterlage präsentiert das Prüfungswissen ausführlich und mit praktischen Beispielen und bietet so Prüfungskandidaten, aber auch allgemeinen Linux-Einsteigern ein solides Fundament für den Einsatz des freien Betriebssystems Linux und den Erwerb vertiefter Kenntnisse in der Linux-Anwendung und -Administration. Neben einer ausführlichen Einführung in den Hintergrund von Linux und freier bzw. Open-Source-Software erklären wir die wichtigsten Konzepte und Werkzeuge von Linux wie die Shell, den Umgang mit Dateien und Skripten und die Struktur des Systems. Einblicke in Administration, Benutzer- und Rechteverwaltung sowie Linux als Client im Netz runden die Darstellung ab.

Aufbauend auf den Inhalten dieser Schulungsunterlage können Absolventen von *Linux Essentials* sich auf weiterführende Zertifizierungen wie das LPIC-Programm des LPI, aber auch herstellerspezifische Zertifikate wie die von Red Hat oder SUSE vorbereiten.

Die Schulungsunterlage ist besonders für den Einsatz in einem *Linux-Essentials*-Vorbereitungskurs an allgemein- oder berufsbildenden Schulen oder Hochschulen gedacht, aber durch ihren ausführlichen Ansatz und zahlreiche Übungen mit Musterlösungen ist sie auch zum Selbststudium geeignet.

Diese Schulungsunterlage soll den Kurs möglichst effektiv unterstützen, indem das Kursmaterial in geschlossener, ausführlicher Form zum Mitlesen, Nach- oder Vorarbeiten präsentiert wird. Das Material ist in Kapitel eingeteilt, die jeweils für sich genommen einen Teilaspekt umfassend beschreiben; am Anfang jedes Kapitels sind dessen Lernziele und Voraussetzungen kurz zusammengefasst, am Ende finden sich eine Zusammenfassung und (wo sinnvoll) Angaben zu weiterführender Literatur oder WWW-Seiten mit mehr Informationen.

Kapitel

Lernziele
Voraussetzungen

Zusätzliches Material oder weitere Hintergrundinformationen sind durch das »Glühbirnen«-Sinnbild am Absatzanfang gekennzeichnet. Zuweilen benutzen diese Absätze Aspekte, die eigentlich erst später in der Schulungsunterlage erklärt werden, und bringen das eigentlich gerade Vorgestellte so in einen breiteren Kontext; solche »Glühbirnen«-Absätze sind möglicherweise erst beim zweiten Durcharbeiten der Schulungsunterlage auf dem Wege der Kursnachbereitung voll verständlich.

Absätze mit dem »Warnschild« weisen auf mögliche Probleme oder »gefährliche Stellen« hin, bei denen besondere Vorsicht angebracht ist. Achten Sie auf die scharfen Kurven!

Übungsaufgaben

Die meisten Kapitel enthalten auch Übungsaufgaben, die mit dem »Bleistift«-Sinnbild am Absatzanfang gekennzeichnet sind. Die Aufgaben sind numeriert und Musterlösungen für die wichtigsten befinden sich hinten in dieser Schulungsunterlage. Bei jeder Aufgabe ist in eckigen Klammern der Schwierigkeitsgrad angegeben. Aufgaben, die mit einem Ausrufungszeichen (»!«) gekennzeichnet sind, sind besonders empfehlenswert.

Auszüge aus Konfigurationsdateien, Kommandobeispiele und Beispiele für die Ausgabe des Rechners erscheinen in `Schreibmaschinenschrift`. Bei mehrzeiligen Dialogen zwischen Benutzer und Rechner werden die Benutzereingaben in **`fetter Schreibmaschinenschrift`** angegeben, um Missverständnisse zu vermeiden. Wenn Teile einer Kommandoausgabe ausgelassen wurden, wird das durch »◁◁◁◁◁« kenntlich gemacht. Manchmal sind aus typografischen Gründen Zeilenumbrüche erforderlich, die in der Vorlage auf dem Rechner nicht stehen; diese werden als »▷ ◁« dargestellt. Bei Syntaxdarstellungen stehen Wörter in spitzen Klammern (»⟨*Wort*⟩«) für »Variable«, die von Fall zu Fall anders eingesetzt werden können; Material in eckigen Klammern (»[`-f` ⟨*Datei*⟩]«) kann entfallen und ein vertikaler Balken trennt Alternativen (»`-a`|`-b`«).

Wichtige Konzepte

Definitionen

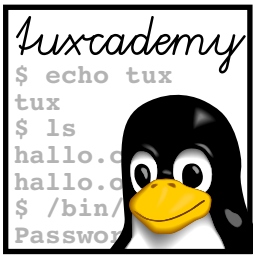
Wichtige Konzepte werden durch »Randnotizen« hervorgehoben; die **Definitionen** wesentlicher Begriffe sind im Text fett gedruckt und erscheinen ebenfalls am Rand.

Verweise auf Literatur und interessante Web-Seiten erscheinen im Text in der Form »[GPL91]« und werden am Ende jedes Kapitels ausführlich angegeben.

Wir sind bemüht, diese Schulungsunterlage möglichst aktuell, vollständig und fehlerfrei zu gestalten. Trotzdem kann es passieren, dass sich Probleme oder Ungenauigkeiten einschleichen. Wenn Sie etwas bemerken, was Sie für verbesserungsfähig halten, dann lassen Sie es uns wissen, etwa indem Sie eine elektronische Nachricht an

```
info@tuxcademy.org
```

schicken. (Zur Vereinfachung geben Sie am besten den Titel der Schulungsunterlage, die auf der Rückseite des Titelblatts enthaltene Revisionsnummer sowie die betreffende(n) Seitenzahl(en) an.) Vielen Dank!



1

Computer, Software und Betriebssysteme

Inhalt

Lernziele

- Grundlegende Computer-Hardware-Kenntnisse erwerben
- Von verschiedenen Betriebssysteme gehört haben und ihre Gemeinsamkeiten und Unterschiede einschätzen können

Vorkenntnisse

- Grundlegende Computer-Kenntnisse sind nützlich

1.1 Was ist eigentlich ein Computer?

Bevor wir uns damit beschäftigen, was ein Computer überhaupt ist, hier ein paar Zitate von Berühmtheiten aus der Szene:

> »Wir dachten zuerst, dass wir mit einem halben Dutzend Computern in Forschungslaboratorien allen Bedarf in [den Vereinigten Staaten] abdecken könnten« *Howard H. Aiken, 1952*

Frühe Computer

Howard Aiken war ein Computerpionier und der Entwickler von IBMs erstem Computer, dem »Harvard Mark I«. Die ersten Computer im modernen Sinne wurden im zweiten Weltkrieg gebaut, um bei der Entschlüsselung von Geheimcodes oder bei komplizierten Berechnungen zu helfen, und es waren große, aufwendige und fehleranfällige Geräte – die elektronischen Bauteile wie Transistoren oder integrierte Schaltkreise, aus denen heutige Computer bestehen, waren allesamt noch nicht erfunden. Was sich in dieser Zeit und den Jahren unmittelbar nach dem Krieg herausstellte, waren allerdings einige Grundannahmen, die erfüllt sein müssen, damit man von einem Gerät als »Computer« reden kann:

- Ein Computer verarbeitet *Daten* gemäß einer Folge von *automatisch* ausgeführten Instruktionen, einem *Programm*.
- Programme müssen *Entscheidungen* und *Wiederholungen* erlauben.
- Es muss möglich sein, das ausgeführte Programm zu ändern oder auszutauschen.

Zum Beispiel enthalten viele technische Geräte – vom Fernseher über die Digitalkamera oder Waschmaschine bis zum Auto – heute programmierte Steuerungen, fast kleine Computer. Trotzdem betrachtet man solche Geräte nicht als »Computer«, weil sie nur feste, nicht änderbare Programme ausführen. Umgekehrt erlaubt ein Taschenrechner zwar die »Verarbeitung von Daten«, aber – jedenfalls solange es kein teurer »programmierbarer Taschenrechner« ist – passiert das nicht automatisch, sondern ein Mensch drückt die Tasten.

In den frühen 1950er Jahren waren Computer hochspezialisierte Geräte, die man – wie Aiken sagte – am ehesten in Forschungsinstituten vermuten würde. In Science-Fiction-Filmen jener Zeit sieht man die Säle, voll mit Reihen von Schränken, in denen sich geheimnisvolle Spulen drehen. In nicht ganz 70 Jahren hat sich dieses Bild grundlegend gewandelt[1].

> »Es gibt keinen Grund für irgendwen, einen Computer daheim zu haben.« *Ken Olsen, 1977*

»Kleine« Computer in den 1970ern

Ken Olsen war der Chef einer anderen Computerfirma, der *Digital Equipment Corporation (DEC)*, die in den 1970er Jahren an der Spitze der Entwicklung »kleiner« Computer stand[2]. Wobei »klein« damals durchaus relativ zu sehen war; es bedeutete zuerst etwas wie »braucht keinen kompletten Computersaal mit Klimaanlage und eigenem Elektrizitätswerk und kostet weniger als eine Million Dollar« (oder so); Fortschritte bei der Hardware führten aber dazu, dass gegen Ende der 1970er Jahre »klein« etwas hieß wie »kann von zwei Personen hochgehoben werden«.

DEC ist für uns Linux-Freunde wichtig, denn auf zwei DEC-Computermodellen, der PDP-8 und der PDP-11, wurde Unix entwickelt – das System, das gut 20 Jahre später Linus Torvalds dazu inspirierte, mit Linux zu beginnen.

Heimcomputer

In den 1970er Jahren kamen auch die ersten »Heimcomputer« auf. Diese waren nicht mit heutigen PCs zu vergleichen – man musste sie selber zusammenlöten

[1] Das klassische »Zitat« in diesem Zusammenhang wird eigentlich Thomas J. Watson, dem Chef von IBM, zugeschrieben, der 1943 etwas gesagt haben soll wie »Es gibt einen Weltmarkt für vielleicht fünf Computer«. Leider ist das nicht belegt. Und wenn er es tatsächlich 1943 gesagt hätte, hätte er zumindest für die nächsten zehn Jahre oder so durchaus recht behalten.

[2] DEC wurde 1998 von Compaq gekauft und Compaq 2002 von Hewlett-Packard.

(heute ein Ding der physikalischen Unmöglichkeit), und sie hatten selten eine vernünftige Tastatur und fast nie einen anständigen Bildschirm. Sie waren eigentlich ein Zeitvertreib für Bastler, so ähnlich wie eine elektrische Eisenbahn, denn wirklich anfangen konnte man mit ihnen ehrlicherweise nicht viel. Trotzdem waren es »Computer« im Sinne unserer Definition von oben, denn sie waren frei programmierbar – auch wenn man die Programme umständlich eintippen oder (mit Glück) von einer Audiocassette lesen konnte. Wirklich für voll genommen wurden diese Apparate trotzdem nicht wirklich, und Ken Olsens Zitat ist deswegen auch oft missverstanden worden: Er hatte absolut nichts gegen kleine Computer (seine Firma verkaufte ja welche). Was er nicht einsah, war die Idee, den kompletten Haushalt von einem Computer steuern zu lassen (Heizung, Licht, Unterhaltung und so weiter) – eine Idee, die damals noch sehr hypothetisch war, heute eher möglich und vielleicht nicht mehr ganz so absurd scheinend.

Erst in den späten 1970er und 1980er Jahren wandelten die »Heimcomputer« sich vom Bausatz zum fertigen Gerät (Namen wie »Apple II« und »Commodore 64« sind zumindest den Älteren unter uns vielleicht noch ein Begriff) und begannen sich auch in den Büros auszubreiten. 1981 wurde der erste IBM PC vorgestellt, und 1984 kam der erste »Macintosh« von Apple auf den Markt. Der Rest ist, wie man sagt, Geschichte – aber man sollte nicht vergessen, dass die Welt der Computer nicht nur aus PCs und Macs besteht. Die riesigen, Säle füllenden Rechner von früher sind immer noch unter uns – auch wenn sie seltener werden und oft in Wirklichkeit aus großen Gruppen von Computern bestehen, die ziemlich eng mit den PCs auf unserem Tisch verwandt sind und zusammenarbeiten. Am Prinzip hat sich aber seit Howard Aikens Zeit nichts geändert: Computer sind immer noch Daten automatisch verarbeitende Geräte mit auswechselbaren Programmen, die Verzweigungen und Schleifen enthalten können. Und das wird auch weiter so bleiben.

IBM PC

Übungen

1.1 [1] Was war der erste Computer, den Sie benutzt haben? Was für einen Prozessor hatte er, wieviel RAM-Speicher und wie groß war die Festplatte (falls es eine gab – falls nicht, wie wurden Daten dauerhaft gespeichert)?

1.2 Bestandteile eines Computers

Werfen wir bei dieser Gelegenheit nochmal einen Blick auf das »Innenleben« eines Computers (genauer gesagt eines »IBM-kompatiblen« PC) und die Komponenten, die wir dort finden:

Prozessor Der Prozessor (neudeutsch »CPU« für *central processing unit*) ist das Herzstück des Computers: Hier findet die automatische programmgesteuerte Datenverarbeitung statt, die den Computer erst zum Computer macht. Heutige Prozessoren enthalten meist mehrere »Kerne«, das heißt, die wesentlichen Bestandteile des Prozessors sind mehrfach vorhanden und können unabhängig voneinander arbeiten, was grundsätzlich die Verarbeitungsgeschwindigkeit und damit die Leistung des Computers erhöht – und besonders schnelle Computer haben außerdem mehr als einen Prozessor. PCs enthalten normalerweise Prozessoren von Intel oder AMD (die sich in Details unterscheiden, aber dieselben Programme ausführen können). Tablets und Smartphones benutzen in der Regel ARM-Prozessoren, die nicht so leistungsfähig sind, aber dafür deutlich stromsparender. Intel- bzw. AMD-Prozessoren können nicht direkt Programmcode ausführen, der für ARM-Prozessoren gemeint sind und umgekehrt.

RAM-Speicher Den Arbeitsspeicher eines Computers nennt man »RAM« (für *random-access memory*, beliebig zugreifbaren Speicher). Hier werden nicht

nur die Daten abgelegt, mit denen der Computer arbeitet, sondern auch der Programmcode des Computers.

Das ist ein genialer Trick, der auf den Computerpionier John von Neumann, einen Zeitgenossen von Howard Aiken, zurückgeht. Im Grunde gibt es dann nämlich gar keinen Unterschied zwischen Programmcode und Daten mehr – das heißt, dass Programme Programmcode genauso manipulieren können wie Adressen oder Kochrezepte. (In der alten Zeit musste man zum »Programmieren« außen am Computer Kabel umstöpseln, oder die Programme waren auf Papierstreifen oder -karten gelocht und konnten nicht ohne Weiteres geändert werden.)

Heutige Computer haben meist einen RAM-Speicher von 1 Gibibyte oder mehr. 1 Gibibyte sind 2^{30}, also 1.073.741.824 Bytes[3] – eigentlich eine unvorstellbar große Zahl. Zum Vergleich: *Harry Potter und die Heiligtümer des Todes* enthält um die 600 Seiten mit jeweils an die 1.700 Buchstaben, Leer- und Satzzeichen[4] – vielleicht eine Million Zeichen. Ein Gigabyte entspricht also über den Daumen gepeilt dem Inhalt von 1.000 *Harry-Potter*-Schinken, bei gut 600 g pro Buch ist das schon ein Lieferwagen voll, und wenn man sich nicht nur für die Abenteuer des Zauberlehrlings interessiert, sind 1.000 Bücher eine ansehnliche Bibliothek.

Grafikkarte Früher war man froh, wenn ein Computer eine elektrische Schreibmaschine ansteuern konnte, um seine Ausgabe zu drucken. Die alten Heimcomputer schloss man an den Fernseher an, mit zum Teil grottenschlechter Bildqualität. Heutzutage dagegen haben selbst einfache »Smartphones« sehr eindrucksvolle Grafik, und gängige PCs enthalten Grafikhardware, für die man in den 1990er Jahren noch den Gegenwert eines teuren Sportwagens oder kleinen Hauses hätte lockermachen müssen[5]. Das Stichwort der heutigen Zeit lautet »3D-Beschleunigung«, was nicht heißt, dass man die Bildschirminhalte wirklich räumlich sehen kann (auch wenn das langsam tatsächlich in Mode kommt), sondern dass bei der *Verarbeitung* der Grafik im Computer außer links, rechts, oben und unten – den auf dem Monitor sichtbaren Richtungen – auch noch vorne und hinten ins Spiel kommen, und das durchaus im wörtlichen Sinne: Für fotorealistische Spiele ist es ja absolut entscheidend, ob ein Monster vor oder hinter einer Mauer steht, also zu sehen ist oder nicht, und der Sinn moderner Grafikkarten ist unter anderem, dem Prozessor des Rechners solche Entscheidungen abzunehmen, damit der sich in der Zeit um andere Dinge kümmern kann. Heutige Grafikkarten haben entsprechend auch eigene Prozessoren, die mitunter viel schneller rechnen können als die CPU des Rechners selbst, aber nicht so allgemein gehalten sind.

Viele Computer haben gar keine separate Grafikkarte, sondern die Grafikhardware ist in den Prozessor integriert. Das macht den Rechner kleiner, billiger, leiser und stromsparender, aber Sie müssen dann auch Abstriche bei der Grafikleistung machen – was möglicherweise kein wirkliches Problem darstellt, wenn Sie nicht die neuesten Spiele spielen wollen.

Hauptplatine Die Hauptplatine (neudeutsch »Motherboard«) ist das (meist) viereckige beschichtete Stück Kunststoff, auf dem die CPU, der RAM-Speicher und die Grafikkarte des Computers befestigt sind – zusammen mit diversen anderen Komponenten, die ein Computer braucht, etwa Anschlüsse für

[3] Der Volksmund spricht oft von »Gigabyte«, aber ein »Gigabyte« sind 10^9 Bytes, also knapp 7 Prozent weniger.

[4] Jedenfalls die englische Ausgabe – die einzige, die der Autor dieser Zeilen zur Hand hatte.

[5] Alles übrigens dank dem nicht nachlassenden Interesse der Menschheit an beeindruckenden Computerspielen. Wer behauptet, dass Videospiele zu nichts nütze seien, sollte sich das mal in Ruhe durch den Kopf gehen lassen.

Festplatten, Drucker, Tastatur, Maus oder Netzwerkkabel und der zur Verwaltung dieser Anschlüsse nötigen Elektronik. Hauptplatinen für Computer gibt es in allen möglichen Größen und Farben[6] – etwa für kleine leise Computer, die als Videorecorder im Wohnzimmer funktionieren oder im Büro nicht durch heulende Lüfter stören sollen, oder für schnelle Server, die viel Platz für RAM-Speicher und mehrere Prozessoren bieten.

Netzteil Um zu funktionieren, braucht ein Computer Strom – wie viel, hängt davon ab, aus welchen Bestandteilen er genau besteht. Das Netzteil dient dazu, den Wechselstrom aus der Steckdose mit seinen 240 Volt in den Gleichstrom mit verschiedenen niedrigen Spannungen umzuwandeln, den die Elektronik im Computer benötigt. Man muss es so wählen, dass es genug Strom für alle Komponenten des Computers liefern kann (schnelle Grafikkarten sind typischerweise die Gierhälse Nr. 1), aber dabei möglichst nicht überdimensioniert ist, damit es bei seiner Arbeit nicht selber unnötig zu viel Strom verschluckt (Stichwort »Wirkungsgrad«).

Der meiste Strom, den das Netzteil in den Computer pumpt, wird übrigens über kurz oder lang zu Wärme, weswegen gute Kühlung sehr wichtig ist. Der Einfachheit halber haben die meisten Computer darum einen oder mehrere Lüfter, die der teuren Elektronik frische Luft zufächeln oder die heiße Luft aus dem Gehäuse pusten sollen. Wenn man sehr sorgfältig ist, kann man Computer bauen, die ohne Lüfter auskommen und darum schön leise sind, aber solche Computer sind dann im Vergleich ziemlich teuer und meistens auch nicht so schnell (denn »schnell« ist bei Prozessoren und Grafikkarten in etwa gleichbedeutend mit »heiß«).

Festplatten Während der RAM-Speicher des Computers für die Daten benutzt wird, die gerade in Bearbeitung sind (Texte, Tabellen, Web-Seiten, Programme, die geschrieben werden, Musik, Videos, …– und natürlich die Programme selbst, mit denen Sie diese Daten bearbeiten), werden Daten, die gerade nicht aktiv bearbeitet werden, auf einer Festplatte gespeichert. Hauptgrund dafür ist, dass Festplatten viel mehr Daten speichern können als der RAM-Speicher gängiger Computer – die Kapazitäten von Festplatten messen wir heute im Tebibyte-Bereich (1 TiB = 2^{40} Byte), sie liegen also um einen Faktor 100–1000 über typischen RAM-Kapazitäten.

Für den Zugewinn an Platz bezahlen wir mit Abstrichen bei der Wiederfinde-Geschwindigkeit – die Zugriffszeiten auf RAM werden in Nanosekunden gemessen, die auf (magnetische) Festplatten dagegen in Millisekunden. Das sind mal schlappe 6 Größenordnungen – der Unterschied zwischen einem Meter und 1.000 Kilometern.

Traditionell bestehen Festplatten aus rotierenden Scheiben, die mit einem magnetisierbaren Material beschichtet sind. Schreib/Leseköpfe können dieses Material gezielt magnetisieren und die so gespeicherten Informationen später dann wieder lesen. Die Scheiben rotieren mit 4.500 bis 15.000 Umdrehungen pro Minute, und der Abstand zwischen Scheibe und Kopf ist äußerst gering (bis zu 3 nm). Festplatten sind also extrem empfindlich gegenüber Erschütterungen und Herunterfallen, denn wenn der Schreib/Lesekopf im Betrieb mit der Scheibe in Kontakt kommt – der gefürchtete *head crash* – führt das zur Zerstörung der Festplatte.

Neumodische Festplatten für mobile Rechner haben Beschleunigungssensoren, die feststellen können, dass der Computer herunterfällt, und die Festplatte schnell stilllegen (oder das zumindest versuchen).

Langsam in Mode kommen SSDs oder *solid-state disks*, die die Daten statt auf magnetischen Scheiben in Flash-Speicher ablegen, einer Art von RAM-

[6] Ehrlich! Auch wenn man seine Hauptplatine nicht nach der Farbe aussuchen sollte.

Speicher, der auch ohne Strom seine Informationen halten kann. SSDs sind schneller als magnetische Festplatten, aber pro Gigabyte Speicherkapazität beträchtlich teurer. Dafür haben sie keine beweglichen Teile, sind stoßunempfindlich und im Vergleich zu herkömmlichen Festplatten energiesparend, was sie für tragbare Computer interessant macht.

SSDs sagt man außerdem nach, dass sie sich »abnutzen«, da die Flash-Speicherplätze, »Zellen« genannt, nur für eine gewisse Anzahl von Schreibvorgängen gut sind. Messungen haben gezeigt, dass das in der Praxis aber keine nennenswerte Behinderung darstellt.

Es gibt verschiedene Methoden, eine Festplatte (magnetisch oder SSD) an einen Computer anzuschließen. Aktuell üblich ist »serielles ATA« (SATA), ältere Computer verwenden »paralleles ATA«, auch »IDE« genannt. Im Serverbereich finden Sie ferner SCSI- oder SAS-(»seriell angeschlossenes SCSI«)Platten. Für externe Platten benutzt man USB oder eSATA (eine SATA-Abart mit robusteren Steckern).

Übrigens: Gerade bei Festplatten kommt der Unterschied zwischen Giga- und Gibibyte (bzw. Tera- und Tebibyte) besonders zum Tragen. Typisches Beispiel: Sie kaufen ein neues »100-GB-Laufwerk«, schließen es an Ihren Computer an und stellen zu Ihrem Entsetzen fest, dass Ihr Computer Ihnen auf der neuen Platte nur 93 »GB« Kapazität anzeigt! Ihre Platte ist aber nicht kaputt (zum Glück) – nur, wenn der Festplattenhersteller von »GB« spricht, meint er (übrigens ganz korrekt) »Gigabyte«, also Milliarden Byte, während Ihr Computer wahrscheinlich (strenggenommen ungenau) in »Gibibyte«, also Einheiten von 2^{30} Byte, rechnet.

Optische Laufwerke Außer Festplatten unterstützen PCs in der Regel optische Laufwerke, die Medien wie CD-ROMs, DVDs oder Blu-ray-Discs lesen und unter Umständen auch schreiben können. (In mobilen Geräten ist für sowas mitunter physikalisch kein Platz, was nicht heißt, dass solche Laufwerke nicht meist als externe Geräte angeschlossen werden könnten.) Optische Medien – der Name kommt daher, dass die Informationen darauf mit einem Laser abgetastet werden – dienen vor allem zur Verteilung von Software und »Inhalten« (Musik und Filme), und ihre Bedeutung schwindet allmählich, weil mehr und mehr Firmen auf das Internet als Distributionsmedium setzen.

Früher hat man optische Medien auch für Sicherheitskopien in Betracht gezogen, aber das ist heute nicht mehr realistisch – auf eine CD-ROM passen höchstens an die 900 MiB Daten, auf eine DVD bis zu maximal um die 9 GiB, für eine Komplettsicherung einer 1-TiB-Festplatte bräuchten Sie also 1000 Medien von CD- oder 100 von DVD-Größe, und die ständige Wechselei wäre auch nervend. (Sogar auf Blu-ray-Discs passen maximal 50 GiB oder so, und Laufwerke, die Blu-ray-Discs *schreiben* können, sind noch ziemlich teuer.)

Anzeige In alten Filmen sieht man sie noch: die grün leuchtenden Computerbildschirme. In der Realität sind sie allerdings bis auf Spezialfälle verschwunden, Farbe ist angesagt, und neue Bildschirme sind auch nicht mehr fett und klobig wie die Röhrenmonitore (CRT, engl. *cathode ray tube*), die früher üblich waren, sondern sind flache, elegante Monitore auf der Basis von Flüssigkristallen (LCD, *liquid crystal display*). LCDs haben nicht nur den Vorteil, weniger Platz auf dem Schreibtisch wegzunehmen, sondern flimmern auch nicht und belasten den Anwender nicht mit möglicherweise schädlicher Strahlung – eine Win-Win-Situation! Ein paar Nachteile gibt es auch, zum Beispiel ändert sich bei LCDs das Aussehen des Bilds, wenn man aus

einem zu schrägen Winkel auf die Anzeige schaut, und billigere Geräte liefern mitunter ein fleckiges Bild, weil die Beleuchtung nicht gleichmäßig ist.

Bei Röhrenmonitoren musste man noch darauf achten, sie nicht lange unbenutzt mit demselben Bild in der Anzeige herumstehen zu lassen, da sich dieses Bild sonst »einbrannte« und permanent schemenhaft im Hintergrund zu sehen war. Man verwendete darum »Bildschirmschoner«, die nach einer gewissen Zeit der Untätigkeit den Bildschirminhalt durch eine mehr oder weniger putzige Animation ersetzten, um das Einbrennen zu verhindern (der Klassiker war ein Aquarium mit Fischen und anderem Meeresgetier). LCDs haben das Einbrenn-Problem nicht mehr, aber die Bildschirmschoner sind für ihren Deko-Wert erhalten geblieben.

LCDs gibt es in allen Größen vom Smartphone-Format bis zu wandfüllenden Großbildschirmen; wichtigstes Kennzeichen ist die Auflösung, die bei PC-Displays heutzutage meist zwischen 1366 × 768 (horizontal × vertikal) und 1920 × 1080 Bildpunkten liegt. (Niedrigere und höhere Auflösungen sind möglich, ergeben aber nicht unbedingt ökonomischen oder visuellen Sinn.) An die meisten Computer könnte man auch mehr als einen Bildschirm anschließen, um die Arbeitsfläche zu vergrößern.

Auflösung

Eingebürgert hat sich auch bei Computermonitoren ein Seitenverhältnis von 16 : 9, was dem hochauflösenden Fernsehen entspricht – eigentlich eine dumme Entwicklung, da die meisten Computer überhaupt nicht zum Fernsehen benutzt werden und eine höhere, aber schmalere Anzeige (etwa im früher üblichen 4 : 3-Format) den viel gängigeren Anwendungen wie Textverarbeitung oder der Arbeit mit Tabellen entgegenkommen würde.

Andere Peripheriegeräte Natürlich können Sie an einen Computer noch viele andere Geräte anschließen außer den bisher hier genannten: Drucker, Scanner, Kameras, Fernsehempfänger, Modems, Roboter-Arme, kleine Raketenwerfer zum Ärgern der Schreibtisch-Nachbarn und so weiter. Die Liste ist schier endlos, und wir können hier auch nicht jede Geräteklasse einzeln diskutieren. Einige Beobachtungen können wir trotzdem machen:

- Ein erfreulicher Trend ist zum Beispiel die Vereinheitlichung der Schnittstellen, mit der die meisten Geräte angeschlossen werden. Während früher fast jede Geräteklasse ihre eigene Schnittstelle hatte (parallele Schnittstellen für Drucker, serielle für Modems, »PS/2«-Schnittstellen für Tastaturen und Mäuse, SCSI für Scanner, …), wird heute für die meisten Geräte USB (*Universal Serial Bus*) benutzt, eine relativ idiotensichere und leidlich schnelle und flexible Methode, die außerdem das An- und Abstöpseln von Geräten im laufenden Betrieb gestattet.
- Ein weiterer Trend ist der hin zu mehr »Intelligenz« in den Peripheriegeräten selbst: Früher waren sogar teure Drucker ziemlich stupide Apparate etwa auf der IQ-Ebene von elektrischen Schreibmaschinen, und als Programmierer musste man sich im Detail darum kümmern, genau die richtigen Steuerungscodes an den Drucker zu schicken, um eine bestimmte Ausgabe zu erzielen. Heutzutage sind Drucker (zumindest gute Drucker) eigentlich auch Computer, die ihre eigenen Programmiersprachen unterstützen und dem Programmierer jede Menge Arbeit abnehmen. Dasselbe gilt entsprechend für viele andere Peripheriegeräte.

  Natürlich gibt es weiterhin strunzdumme Drucker (vor allem im unteren Preissegment), die es dem Computer überlassen, die Druckausgabe aufzubereiten. Allerdings machen auch die es dem *Programmierer* so einfach wie ihre teureren Verwandten.

Übungen

1.2 [2] Schrauben Sie Ihren Computer auf (ggf. unter der Aufsicht Ihres Lehrers, Trainers oder Erziehungsberechtigten – und das Gerät unbedingt vorher ausschalten und vom Stromnetz trennen!) und identifizieren Sie die wichtigsten Komponenten wie CPU, RAM-Speicher, Hauptplatine, Grafikkarte, Netzteil und Festplatte. Welche Komponenten in Ihrem Computer haben wir hier nicht aufgezählt?

1.3 Software

Genauso wichtig wie die »Hardware« eines Computers, also die technischen Komponenten, aus denen er besteht[7], ist die »Software«, also die Programme, die darauf laufen. Ganz grob läßt die Software eines Computers sich in drei Bereiche einteilen:

Firmware

- Die **Firmware** ist auf der Hauptplatine des Computers gespeichert und läßt sich nur mühsam, wenn überhaupt, ändern oder austauschen. Sie dient dazu, den Computer beim Einschalten in einen definierten Zustand zu versetzen und das eigentliche Betriebssystem zu laden. Meist gibt es auch die Möglichkeit, beim Einschalten in einen Einstellungs-Modus zu springen, wo Sie die Uhr stellen und bestimmte Eigenschaften der Hauptplatine ein- oder ausschalten können.

  Bei PCs heißt die Firmware »BIOS« (*Basic Input/Output System*) oder, auf neueren Systemen, »EFI« (*Extensible Firmware Interface*).

  Manche Hauptplatinen haben in ihrer Firmware heute ein kleines Linux-System, das angeblich schneller startet als Windows und mit dem Sie mal eben im Netz surfen oder eine DVD anschauen können sollen, ohne extra Windows booten zu müssen. Ob sich das wirklich rentiert, ist eine andere Frage.

Betriebssystem

- Das **Betriebssystem** sorgt dafür, dass Sie mit dem Computer tatsächlich etwas anfangen können: Es verwaltet die Ressourcen des Computers, also den RAM-Speicher, die Festplatten, die Rechenzeit auf der (oder den) CPU(s), die einzelne Programme nutzen dürfen, und den Zugriff auf andere Peripheriegeräte. Es sorgt dafür, dass Programme gestartet und beendet werden können, dass die verschiedenen Benutzer des Computers einander nicht in die Quere kommen und dass jedes Programm auch mal laufen darf. Außerdem regelt es – auf einer elementaren Ebene – die Teilnahme des Computers an einem lokalen Netz oder dem Internet. Das Betriebssystem stellt meistens auch eine grafische Oberfläche zur Verfügung und bestimmt so maßgeblich, wie der Computer für seine Benutzer »aussieht« und sich »anfühlt«.

  Wenn Sie einen neuen Computer kaufen, wird dieser meist mit einem vorinstallierten Betriebssystem geliefert: PCs mit Microsoft Windows, Macs mit (Mac) OS X, Smartphones oft mit Android (einem Linux-Ableger). Das Betriebssystem ist mit einem Computer aber nicht so eng verbunden wie die Firmware, sondern kann in vielen Fällen durch ein anderes ersetzt werden – auf den meisten PCs und Macs können Sie zum Beispiel Linux installieren.

  Oder Sie installieren Linux *zusätzlich* zum existierenden Betriebssystem – in der Regel auch kein Problem.

Benutzerprogramme

- **Benutzerprogramme** erlauben es Ihnen, mit dem Computer etwas Nützliches zu tun, etwa Texte zu schreiben, Bilder zu malen oder zu bearbeiten, Musik zu komponieren, Spiele zu spielen, im Internet zu surfen oder

neue Programme zu schreiben. Solche Programme nennt man auch **Anwendungsprogramme**[8]. Daneben gibt es meistens **Dienstprogramme**, die das Betriebssystem zur Verfügung stellt, damit Sie – oder ein designierter »Systemadministrator« – Änderungen an der Konfiguration des Computers machen kann und ähnliches. Auf Servern wiederum laufen meistens Programme, die Dienste für andere Computer erbringen, zum Beispiel Web-, Mail- oder Datenbankserver.

Anwendungsprogramme
Dienstprogramme

1.4 Die wichtigsten Betriebssysteme

1.4.1 Windows und OS X

Wenn es um Betriebssysteme für Computer geht, denken die meisten Leute automatisch zuerst an Windows von Microsoft[9]. Das liegt daran, dass die meisten PCs heutzutage mit Windows verkauft werden – eigentlich keine schlechte Sache, weil ihre Besitzer sie dann relativ zügig in Betrieb nehmen können, ohne erst umständlich ein Betriebssystem installieren zu müssen, aber auf der anderen Seite auch ein Problem, weil das anderen Betriebssystemen wie Linux das Leben unnötig schwer macht.

Windows

Es ist tatsächlich gar nicht einfach, einen Computer ohne Windows zu bekommen – etwa weil Sie ihn nur mit Linux benutzen wollen –, es sei denn, Sie bauen ihn selber aus Einzelteilen zusammen. Theoretisch können Sie sich das Geld für ein unbenutztes vorinstalliertes Windows von der Herstellerfirma des Computers zurückerstatten lassen, wir wissen aber von niemandem, der das jemals geschafft hätte.

Das heutige Windows ist ein Nachkomme von »Windows NT«, mit dem Microsoft in den 1990er Jahren versuchte, ein zeitgemäßes Betriebssystem zur Verfügung zu stellen (frühere Versionen wie »Windows 95« waren grafische Aufsätze für das damalige Betriebssystem von Microsoft, MS-DOS, und selbst für die 1990er ziemlich primitiv). Der Anstand verbietet uns hier eine kritische Würdigung von Windows; es soll genügen, dass es in etwa das tut, was man von einem Betriebssystem erwarten würde, eine grafische Oberfläche zur Verfügung stellt und Unterstützung für die meisten Peripheriegeräte mitbringt (oder die Hersteller dieser Geräte bieten Windows-Unterstützung an).

Windows NT

Der »Macintosh« von Apple kam 1984 in den Handel und verwendet seither ein Betriebssystem namens »Mac OS«. Über die Jahre hat Apple verschiedene zum Teil radikale Änderungen an der Plattform (heutige Macs sind technisch im Wesentlichen dasselbe wie Windows-PCs) und am Betriebssystem gemacht. Bis einschließlich zur Version 9 war Mac OS ein ziemlich wackliges Etwas, das zum Beispiel die gleichzeitige Ausführung von mehreren Programmen nur rudimentär unterstützte. Das aktuelle »Mac OS X« – das »X« ist eine römische 10, nicht der Buchstabe »X« – basiert auf einem an BSD-Unix angelehnten Fundament und ist Linux konzeptuell nicht unähnlich.

Mac OS

Seit Februar 2012 heißt das Macintosh-Betriebssystem offiziell nicht mehr »Mac OS X«, sondern nur noch »OS X«. Sollte uns hin und wieder doch noch ein »Mac OS« rausrutschen, wissen Sie, wie das gemeint ist.

Der große Unterschied zwischen Windows und OS X ist, dass OS X nur mit Apple-Computern verkauft wird und auf »normalen« PCs nicht läuft. Das macht

Unterschiede

[7] Definition von Hardware: »Die Teile des Computers, die man treten kann« (Jeff Pesis)

[8] Manchmal hört man auch das Wort »Applikation«, aber eine Applikation ist etwas, was über ein Loch in den Jeans genäht wird. Die neudeutsche Kurzversion »App« ist jedoch wahrscheinlich nicht mehr zu stoppen.

[9] Viele Leute wissen überhaupt gar nicht, dass es etwas Anderes gibt.

es für Apple viel leichter, ein System zur Verfügung zu stellen, das offensichtlich »aus einem Guss« ist. Windows dagegen muss auf allen möglichen PCs laufen und eine viel breitere Palette von Hardwarekomponenten unterstützen, die in völlig unkalkulierbaren Kombinationen auftauchen können. Windows-Benutzer haben daher eher mit Unverträglichkeiten zu kämpfen, die sich möglicherweise nur schwer oder gar nicht beheben lassen. Zum Ausgleich *gibt* es viel mehr Hardwareauswahl für Windows-Rechner, und die Preise sind im Großen und Ganzen weniger exorbitant.

Gemeinsamkeiten

Windows und OS X haben gemeinsam, dass es sich bei ihnen um »proprietäre« Software handelt: Anwender sind gezwungen, das zu akzeptieren, was Microsoft oder Apple ihnen vorsetzen, und sie können keinen Einblick in die tatsächliche Implementierung des Systems nehmen, geschweige denn Änderungen daran machen. Sie sind gebunden an die Upgrade-Zyklen des Systems, und wenn der Hersteller ein Merkmal des Systems entfernt oder durch etwas Anderes ersetzt, müssen die Anwender sich damit arrangieren.

Auch hier gibt es einen Unterschied: Apple ist im Grunde eine Hardware-Firma und beschäftigt sich mit OS X nur, um Leuten einen Anreiz zu geben, Macs zu kaufen (deswegen gibt es das Mac-OS auch nicht für Nicht-Macs). Microsoft hingegen baut selber keine Computer, sondern verdient sein Geld damit, Software wie Windows zu verkaufen, die auf *x*-beliebigen PCs läuft. Ein Betriebssystem wie Linux ist darum für Microsoft eine viel größere Gefahr als für Apple – die meisten Leute, die einen Apple-Computer kaufen, tun das, weil sie einen *Apple*-Computer (das Gesamtpaket) haben wollen, nicht weil es ihnen speziell auf OS X ankommt. Der PC als Plattform wird dagegen von Tablets und anderen neumodischen Computer-Sorten bedrängt, die nicht mit Windows laufen, und das setzt Microsoft unter extremen Druck. Die Firma Apple könnte bequem damit überleben, statt Macs nur iPhones und iPads zu verkaufen – Microsoft ohne Windows wäre dagegen trotz wohlgefülltem Bankkonto relativ bald bankrott.[10]

1.4.2 Linux

Linux ist ein Betriebssystem, das ursprünglich von Linus Torvalds als Neugierprojekt begonnen wurde, aber dann ein Eigenleben annahm – inzwischen arbeiten Hunderte von Entwicklern (nicht nur Studenten und Hobby-Programmierer, sondern auch Profis von Firmen wie IBM, Red Hat oder Oracle) an seiner Weiterentwicklung.

Linux wurde inspiriert von Unix, einem in den 1970er Jahren bei den AT&T Bell Laboratories entwickelten Betriebssystem für »kleine« Computer (siehe oben für die Bedeutung von »klein« in diesem Kontext), das sich schnell zum bevorzugten System für Wissenschaft und Technik entwickelte. Linux verwendet in sehr weiten Teilen dieselben Konzepte und Grundideen wie Unix, und für Unix geschriebene Software ist leicht auf Linux zum Laufen zu bringen, aber Linux selbst enthält keinen Unix-Code, ist also ein unabhängiges Projekt.

Im Gegensatz zu Windows und OS X steht hinter Linux keine einzelne Firma, deren wirtschaftlicher Erfolg vom Erfolg von Linux abhängt. Linux ist »frei verfügbar« und kann von jedem benutzt werden – auch kommerziell –, der die Spielregeln einhält (dazu im nächsten Kapitel mehr). Dies zusammen mit dem Umstand, dass Linux inzwischen nicht nur auf PCs läuft, sondern in im Wesentlichen identischer Form vom Telefon (das populärste Betriebssystem für Smartphones, Android, ist ein Ableger von Linux) bis zum größten Großrechner (die 10

[10] Eigentlich ist der wirkliche Kriegsschauplatz nicht Windows, sondern Office – die meisten Leute kaufen ja nicht Windows, weil sie Windows so innig lieben, sondern weil es das einzige Betriebssystem für beliebige (d. h. billige) PCs ist, wo Office läuft –, aber da gilt dieselbe Überlegung, wenn man statt »Apple« »Google« einsetzt. Tatsächlich sind Office und Windows (für PCs) die einzigen Produkte, mit denen Microsoft nennenswert Geld verdient; alles andere, was Microsoft macht (mit der möglichen Ausnahme der Spielekonsole Xbox), ist ein Verlustgeschäft.

schnellsten Rechner der Welt laufen alle unter Linux) auf allen Arten von Rechnern zu finden ist, macht Linux zum flexibelsten Betriebssystem in der Geschichte des modernen Computers.

Strenggenommen ist »Linux« nur der Betriebssystem-*Kern*, also das Programm, das sich um die Ressourcenverteilung an Anwendungsprogramme und Dienstprogramme kümmert. Da ein Betriebssystem ohne Anwendungsprogramme aber nicht besonders nützlich ist, installiert man meist eine Linux-*Distribution*, also ein Paket aus dem eigentlichen »Linux« nebst einer Auswahl von Anwendungs- und Dienstprogrammen, Dokumentation und anderen Nützlichkeiten. Das Schöne ist, dass die meisten Linux-Distributionen wie Linux selbst »frei verfügbar« und damit kostenlos oder sehr preisgünstig zu haben sind. Auf diese Weise können Sie einen Rechner mit Programmen ausstatten, deren Äquivalente für Windows oder OS X etliche tausend Euro kosten würden, und Sie geraten nicht in Gefahr, wegen Lizenzverstößen belangt zu werden, nur weil Sie Ihre Linux-Distribution auf allen Ihren Computern und denen von Tante Frieda und Ihren Kumpels Susi und Alex installiert haben.

Distributionen


Über Linux und Linux-Distributionen finden Sie mehr Informationen in Kapitel 2.

1.4.3 Mehr Unterschiede und Gemeinsamkeiten

Tatsächlich unterscheiden die drei großen Betriebssysteme – Linux, Windows und OS X – sich in ihrer Anmutung für den Benutzer heute nur noch in Details. Alle drei bieten eine grafische Oberfläche (GUI, *graphical user interface*) an, die es auch dem weitgehend unbedarften Anwender ermöglicht, ohne tiefgehende Computerkenntnisse durch einfache Gesten wie Klicken und Ziehen seine Dateien zu verwalten, Programme zu starten und so weiter. Viele populäre Anwendungsprogramme stehen auch für alle drei Plattformen zur Verfügung, so dass es fast unerheblich wird, welches der Betriebssysteme Sie letzten Endes benutzen, solange Sie sowieso die meiste Zeit im Web-Browser, Office-Paket oder Mail-Programm verbringen. Das ist ein Vorteil, weil so bei Bedarf eine »schrittweise« Migration von einem System zu einem anderen möglich wird.

grafische Oberfläche

Neben der grafischen Oberfläche bieten alle drei Systeme auch eine Möglichkeit, auf einer »Kommandozeile« textuelle Befehle einzugeben, die das System dann ausführt. Bei Windows und OS X nutzen das vorwiegend Systemadministratoren, während »gewöhnliche« Benutzer normalerweise einen weiten Bogen darum machen – eine Frage der Kultur. Bei Linux dagegen ist die Kommandozeile weniger verpönt, was mit der Abstammung des Systems aus der technisch-wissenschaftlichen Unix-Philosophie zu tun haben mag. Tatsache ist, dass viele Aufgaben sich bequemer und effizienter über die Kommandozeile erledigen lassen, vor allem mit den mächtigen Werkzeugen, die Linux (und eigentlich auch OS X) dafür zur Verfügung stellen. Als angehender Linux-Anwender tun Sie gut daran, sich der Kommandozeile zu öffnen und ihre Stärken und Schwächen kennenzulernen, so wie Sie auch die Stärken und Schwächen der grafischen Oberfläche kennenlernen sollten. Eine Kombination aus beiden gibt Ihnen die größte Flexibilität.

Kommandozeile

Übungen


1.3 [1] Wenn Sie Erfahrung mit einem proprietären Betriebssystem wie Windows oder (Mac) OS X haben: Welche Anwendungsprogramme nutzen Sie dort vor allem? Welche davon sind »freie Software«?

1.5 Ausblick

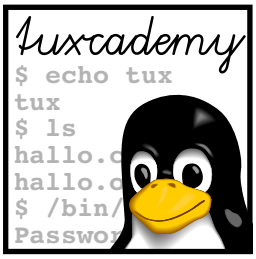
Heutige PCs unter Linux, Windows oder OS X haben unter dem Strich mehr Gemeinsamkeiten als Unterschiede – jedenfalls was die Hardware, die Konzepte und

die Bedienung angeht. Zweifellos können Sie mit jedem der drei Systeme passabel arbeiten, und keines davon ist eindeutig und unbestritten »das Beste«.

Allerdings reden wir hier vor allem über Linux und werden uns im Rest dieser Unterlage bemühen, Ihnen einen möglichst umfassenden Einstieg in dieses System zu geben – Ihnen seine Benutzung erklären, die Stärken herausstellen und, wo nötig, auch auf Schwächen hinweisen. Inzwischen ist Linux eine ernstgemeinte Alternative zu den anderen beiden Systemen und ihnen in diversen Punkten auch überlegen, manchmal sogar weit überlegen. Wir freuen uns, dass Sie bereit sind, sich auf Linux einzulassen, wünschen Ihnen viel Spaß beim Lernen, Üben und Benutzen und – falls Sie die *Linux-Essentials*-Zertifizierung des LPI anstreben – viel Erfolg in der Prüfung!

Zusammenfassung

- Computer sind Geräte, die Daten gemäß einem automatisch ablaufenden, änderbaren Programm verarbeiten, das Entscheidungen und Wiederholungen erlaubt.
- Zu den wichtigsten Komponenten von PCs gehören der Prozessor, der RAM-Speicher, Grafikkarte, Hauptplatine, Festplatten und Ähnliches.
- Die Software auf einem Computer können Sie einteilen in Firmware, Betriebssystem und Benutzerprogramme.
- Das verbreitetste Betriebssystem für PCs ist Windows von Microsoft. Apple-Computer benutzen ein anderes Betriebssystem namens OS X (früher »Mac OS X«).
- Linux ist ein alternatives Betriebssystem für PCs (und viele andere Arten von Computern), das nicht von einer einzigen Firma entwickelt wird, sondern an dem viele Freiwillige mitarbeiten.
- Linux-Distributionen erweitern den Linux-Betriebssystemkern mit Benutzerprogrammen und Dokumentation zu einem tatsächlich benutzbaren System.



2

Linux und freie Software

Inhalt

Lernziele

- Die Grundprinzipien von Linux und freier Software kennen
- Die gängigen FOSS-Lizenzen einordnen können
- Von den wichtigsten freien Anwendungsprogrammen gehört haben
- Von den wichtigsten Linux-Distributionen gehört haben

Vorkenntnisse

- Grundkenntnisse über Computer und Betriebssysteme (Kapitel 1)

2.1 Linux: Eine Erfolgsgeschichte

Im Sommer 1991 studierte Linus Torvalds, damals 21 Jahre alt, Informatik an der Technischen Universität von Helsinki (Finnland)[1]. Er hatte damals einen neuen 386-PC, den er ausprobieren wollte, und amüsierte sich damit, einen Terminal-Emulator zu schreiben, der ohne Betriebssystem auf der rohen Hardware lief und mit dem er auf den Unix-Rechner an der Universität zugreifen konnte. Aus diesem Programm wurde schließlich der erste Linux-Betriebssystemkern.

Unix hatte zu diesem Zeitpunkt schon gut 20 Jahre auf dem Buckel, aber war das Betriebssystem der Wahl an Universitäten und überall da, wo Forschung und Entwicklung betrieben wurde – die technisch-wissenschaftlichen »Workstations« der damaligen Zeit liefen praktisch alle mit verschiedenen Unix-Versionen.

Unix-Urzeit

Unix selbst hatte – fast wie Linux – als »Hobbyprojekt« von Ken Thompson und Dennis Ritchie bei den Bell Laboratories, dem Forschungsinstitut des US-Telekommunikationsgiganten AT&T, angefangen. Es mauserte sich schnell zu einem sehr nützlichen System, und da es zum größten Teil in einer höheren Programmiersprache (C) geschrieben war, konnte es relativ schnell von der ursprünglichen PDP-11 auf andere Rechnerplattformen portiert werden. Außerdem durfte AT&T in den 1970er Jahren keine Software verkaufen, so dass Unix ohne Support zu Selbstkosten »verschenkt« wurde – und da das System klein und übersichtlich war, wurde es zur beliebten Fallstudie in den Betriebssystem-Seminaren der meisten Universitäten.

Ende der 1970er portierten Studenten der University of California in Berkeley Unix auf die VAX, die Nachfolgeplattform der PDP-11, und bauten dabei verschiedene Verbesserungen ein, die als »BSD« (kurz für *Berkeley Software Distribution*) in Umlauf kamen. Diverse Ableger von BSD sind auch heute noch aktuell.

BSD

Minix

Zur Entwicklung der ersten Linux-Versionen benutzte Linus »Minix«, ein Unix-artiges Betriebssystem, das Andrew S. Tanenbaum an der Universität Amsterdam für Unterrichtszwecke geschrieben hatte. Minix war ziemlich simpel gehalten und außerdem nicht frei verfügbar, stellte also kein vollwertiges Betriebssystem dar – Abhilfe war offensichtlich nötig![2]

Am 25. August 1991 kündigte Linus sein Projekt öffentlich an und lud den Rest der Welt zur Mithilfe ein. Zu diesem Zeitpunkt funktionierte das System als alternativer Betriebssystemkern für Minix.

Einen richtigen Namen hatte das System damals noch nicht. Linus nannte es »Freax« (aus »Freak« und »Unix«); er hatte am Anfang kurz über »Linux« als Name nachgedacht, die Idee aber dann als zu selbstverliebt abgelehnt. Als Linus' System auf den FTP-Server der Universität hochgeladen wurde, benannte Linus' Kollege Ari Lemmke, dem der Name »Freax« nicht gefiel, es in eigener Regie in »Linux« um. Linus stimmte der Änderung später zu.

Linux stieß auf beträchtliches Interesse und viele Freiwillige entschlossen sich zur Mitarbeit. Im Dezember 1992 erschien Linux 0.99, die erste Version unter der GPL (Abschnitt 2.2.3) und durchaus ein ausgewachsenes Betriebssystem mit vollständiger (wenn auch simpler) Unix-Funktionalität.

Linux 2.0

Linux 2.0 kam Anfang 1996 heraus und führte einige wichtige Neuerungen ein, etwa die Unterstützung von Mehrprozessor-Rechnern und die Möglichkeit, Module dynamisch zur Laufzeit in den Linux-Kern zu laden – ein bedeutender Schritt hin zu benutzerfreundlichen Linux-Distributionen.

[1] Linus Torvalds ist ethnischer Finne (im September 2010 nahm er die US-amerikanische Staatsbürgerschaft an), aber Angehöriger der schwedischsprachigen Minderheit. Darum hat er einen aussprechbaren Namen.

[2] BSD stand damals noch nicht frei zur Verfügung; Linus sagte einmal, dass er, wenn es BSD schon gegeben hätte, nie mit Linux angefangen hätte.

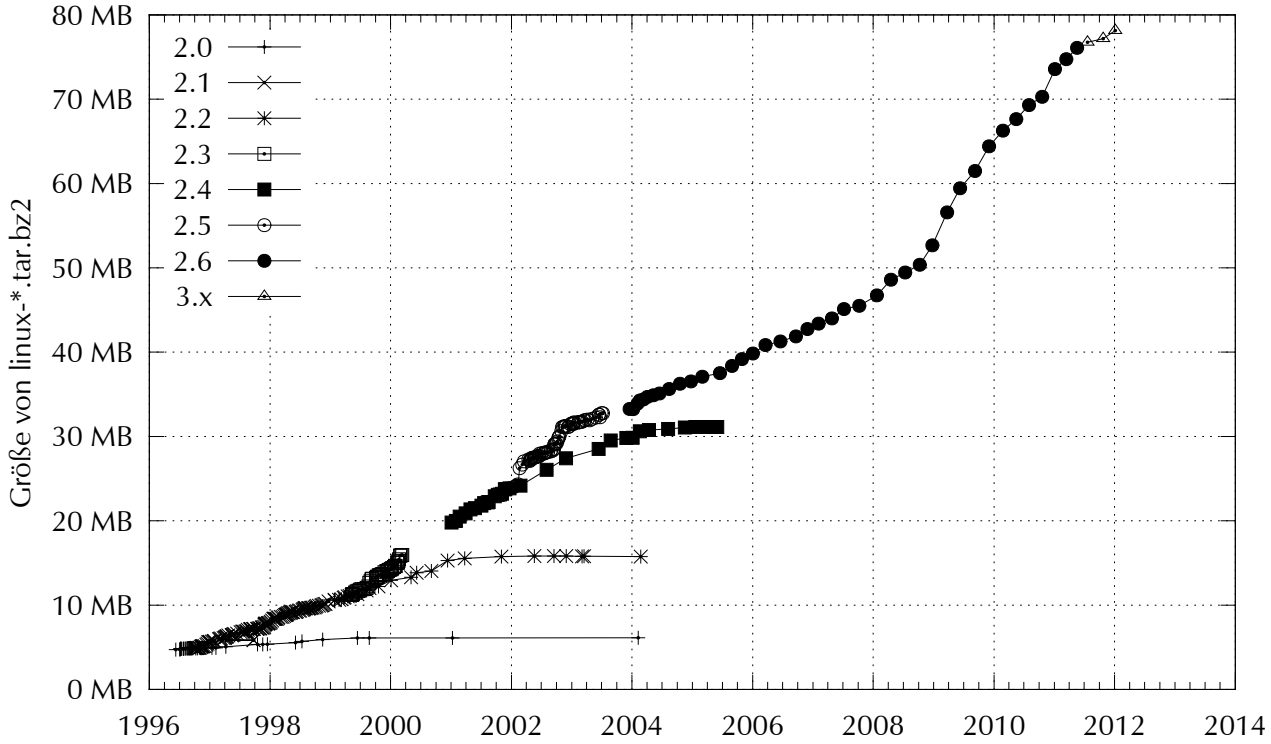


Bild 2.1: Die Weiterentwicklung von Linux, gemessen an der Größe von `linux-*.tar.bz2`. Jede Marke entspricht einer Linux-Version. In den 15 Jahren von Linux 2.0 bis Linux 3.2 hat der Umfang des komprimierten Linux-Quellcodes sich nahezu versechzehnfacht.

Ebenfalls neu in Linux 2.0 war »Tux«, der Pinguin, als offizielles Linux-Maskottchen. Linus Torvalds war in Australien von einem Pinguin angefallen worden, was ihn sehr beeindruckt hatte. Der sitzende Pinguin mit den gelben Füßen wurde von Larry Ewing gezeichnet und der Allgemeinheit zur Verfügung gestellt. Tux

Mit Linux 2.6 wurde Anfang 2004 der Entwicklungsprozess neu organisiert. Galten vorher Versionsnummern mit ungerader zweiter Stelle (etwa »2.3«) als Entwickler-Versionen und solche mit gerader zweiter Stelle (zum Beispiel »2.0«) als stabile Versionen für Endbenutzer, beschlossen die Linux-Entwickler, die Entwickler- und die stabilen Versionen künftig nicht mehr so stark divergieren zu lassen. Ab Linux 2.6 gab es keine separate Entwicklungslinie mehr, sondern Neuerungen fließen jeweils in die nächste Version ein und werden vor deren offizieller Freigabe möglichst ausgiebig getestet. Entwicklungsprozess

Man kann sich das ungefähr so vorstellen: Nach der Freigabe von Linux 2.6.37 sammelt Linus Neuerungen für den nächsten Linux-Kern, baut sie in seine offizielle Version ein und veröffentlicht diese dann als Linux 2.6.38-rc1 (für *release candidate* 1). Diese Version wird von diversen Leuten ausprobiert, und Reparaturen und Verbesserungen fließen in die Version 2.6.38-rc2 ein und so weiter. Irgendwann sieht der Code stabil genug aus, dass er als »Linux 2.6.38« offiziell freigegeben werden kann, und der Prozess wiederholt sich dann mit der Version 2.6.39.

Außer Linus' offizieller Version gibt es Linux-Versionen, die von anderen Entwicklern gepflegt werden. Zum Beispiel gibt es den »Staging Tree«, in dem neue Gerätetreiber »reifen« können, bis sie (nach einigen Runden von Verbesserungen) gut genug dafür sind, Linus für seine Version unterbreitet

zu werden. Einmal freigegebene Linux-Versionen bekommen meist noch eine Weile lang Reparaturen, so dass es auch Versionen wie 2.6.38.1, 2.6.38.2, … geben kann.

Linux 3.0

Im Juli 2011 erklärte Linus die damals gerade in Vorbereitung befindliche Version 2.6.40 summarisch zur Version 3.0 – nur um die Nummerierung zu vereinfachen, umwälzende Neuerungen gab es keine.

Release-Kandidaten heißen heute 3.2-rc1 und so weiter, die nach der Veröffentlichung reparierten Versionen entsprechend 3.1.1, 3.1.2, …

Das Projekt »Linux« ist auch heute nicht abgeschlossen. Linux wird ständig aktualisiert und erweitert, und zwar von Hunderten von Programmierern auf der ganzen Welt, denen inzwischen mehrere Millionen zufriedene private und kommerzielle Anwender gegenüberstehen. Man kann auch nicht sagen, dass das System »nur« von Studenten und anderen Amateuren entwickelt wird – viele Leute, die am Linux-Kern mitarbeiten, haben wichtige Posten in der Computerindustrie und gehören zu den fachlich angesehensten Systementwicklern überhaupt. Inzwischen läßt sich mit Berechtigung behaupten, dass Linux das Betriebssystem mit der breitesten Hardwareunterstützung überhaupt ist, nicht nur bezogen auf die Plattformen, auf denen es läuft (vom Smartphone bis zum Großrechner), sondern auch auf die Treiberunterstützung zum Beispiel auf der Intel-PC-Plattform. Linux dient auch als Test- und Forschungsplattform für neue Betriebssystem-Ideen in Industrie und Hochschule; es ist zweifellos eines der innovativsten derzeit verfügbaren Betriebssysteme.

Virtualisierung

Die Flexibilität von Linux macht es auch zum Betriebssystem der Wahl für Anwendungen wie Virtualisierung und »Cloud Computing«. Virtualisierung erlaubt es, auf einem tatsächlichen (»physikalischen«) Rechner mehrere bis viele »virtuelle« Rechner zu simulieren, die über ihr eigenes Betriebssystem verfügen und für dort laufende Programme so aussehen wie »echte« Rechner. Dies führt zu effizienterer Nutzung von Ressourcen und zu höherer Flexibilität: Die gängigen Infrastrukturen für Virtualisierung gestatten es, virtuelle Maschinen sehr schnell von einem physikalischen Rechner auf einen anderen zu »migrieren«, und als Betreiber einer entsprechenden Infrastruktur können Sie so sehr bequem auf Lastsituationen und Ausfälle reagieren.

Cloud Computing

Cloud Computing ist darauf aufbauend die Idee, Rechenleistung nach Bedarf »auf Abruf« zur Verfügung zu stellen und Firmen damit die Möglichkeit zu geben, auf großangelegte Rechenzentren zu verzichten, die nur gelegentlich bei Anfragespitzen tatsächlich voll ausgelastet werden und ansonsten vor allem Kosten verursachen. Anbieter von Cloud-Computing gestatten ihren Kunden die Nutzung virtueller Maschinen über das Internet, bei Abrechnung auf Basis der tatsächlichen Nutzungsdauer, und das kann gegenüber dem Unterhalt eines »realen« Rechenzentrums zu erheblichen Einsparungen führen, insbesondere wenn man einbezieht, dass Sie als Kunde nicht nur die Anschaffungskosten, sondern auch die Personal-, Material- und Stromkosten für den 24/7-Betrieb des Rechenzentrums vermeiden.

Übungen

2.1 [2] Suchen Sie im Internet nach der berühmt-berüchtigten Diskussion zwischen Andrew S. Tanenbaum und Linus Torvalds, in der Tanenbaum sagt, Linus wäre mit etwas wie Linux bei ihm im Praktikum durchgefallen. Was halten Sie davon?

2.2 [1] Welche Versionsnummer hat der älteste Linux-Kern-Quellcode, den Sie noch finden können?

2.2 Frei oder Open Source?

2.2.1 Urheberrecht und »freie Software«

Im Mittelalter war die Vervielfältigung von Büchern und anderen Schriftstücken eine aufwendige Angelegenheit: Man musste jemanden finden, der des Schreibens mächtig war und die nötige Zeit hatte – was größere Projekte wie das Kopieren von Bibeln zur Domäne von Klöstern machte (die Mönche dort konnten schreiben und hatten jede Menge Zeit). Mit dem Aufkommen des Buchdrucks im 16. Jahrhundert entstand auf einmal ein Problem: Plötzlich wurde das Kopieren ein gutes Stück einfacher und billiger (jedenfalls sofern man eine Druckerpresse sein Eigen nannte), und eifrige Verleger nutzten das weidlich aus, um alles zu vervielfältigen, was sie für verkäuflich hielten. Die Autoren damals hatten eigentlich keine Rechte; sie konnten froh sein, wenn die Verleger ihnen irgendetwas dafür bezahlten, dass sie ihre Werke druckten. Das weitverbreitete Nachdrucken führte auch dazu, dass die Erstdrucker sich betrogen fühlten, wenn andere Drucker ihre Veröffentlichungen ohne Gegenleistung übernahmen. Deswegen wandten viele von ihnen sich an die Obrigkeit mit der Bitte darum, für bestimmte Werke ein Exklusivrecht zur Veröffentlichung zu erhalten. Dies kam der Obrigkeit durchaus zupass, da sie gerne den Überblick über die Druckwerke behalten wollte, die in ihrem Einflussbereich im Umlauf waren. Aus diesen »Privilegien« sowie weiteren Entwicklungen wie der (lobenswerten) Idee, auch den *Autoren* von Werken zum Beispiel das Recht auf ein Honorar einzuräumen, entstand das Konzept des Urheberrechts (bzw. im angelsächsischen Rechtsraum das verwandte Konzept des *copyright*).

Urheberrecht bedeutet nichts Anderes, als dass dem Urheber eines Werks, also dem Autor eines Buchs, Maler eines Bildes, … das Recht zuerkannt wird, als Einziger darüber zu verfügen, was mit dem Werk passiert. Er kann als Autor zum Beispiel einem Verlag das Recht abtreten (bzw., in der Praxis, gegen ein Honorar verkaufen), das Buch zu drucken und in Umlauf zu bringen; der Verlag kann damit Geld verdienen und der Autor muss sich nicht selbst um Vervielfältigung, Werbung, Vertrieb usw. kümmern, so dass beiden gedient ist.

Neben diesen »Verwertungsrechten« gibt es auch »moralische« Rechte, etwa das Recht, als Urheber eines Werks identifiziert zu werden. Diese moralischen Rechte kann ein Urheber nicht abtreten.

Im 20. Jahrhundert wurden die Konzepte des Urheberrechts weltweit (einigermaßen) vereinheitlicht und auch auf andere Werke wie Tonaufnahmen und Filme erweitert.

Mit der Ankunft von Computern und dem Internet gegen Ende des 20. Jahrhunderts wurde die Situation noch einmal gravierend verändert: Während das Urheberrecht früher vor allem dazu diente, Verleger vor anderen Verlegern zu schützen (Privatleute hatten selten die Mittel, Bücher, Schallplatten oder Filme in kommerziell merkbarem Umfang zu vervielfältigen), war es plötzlich jedem, der einen Computer besaß, möglich, digitale Inhalte (etwa Software oder Bücher, Musik oder Filme in digitaler Form) ohne Qualitätsverlust nahezu beliebig oft zu vervielfältigen und weiterzugeben – für Verleger, Musik-, Film- und Softwarefirmen eine Katastrophe, da die bestehenden Geschäftsmodelle, die auf dem Verkauf von physikalischen Artefakten wie Büchern oder CDs beruhten, ins Wanken kamen. Die »Inhalteindustrie« agitiert seitdem für eine Verschärfung der Urheberrechtsgesetze und höhere Strafen für »Raubkopierer« und versucht sich daran, Copyright-Sünder dingfest zu machen (mit wechselndem Erfolg).

Man spricht heute von »geistigem Eigentum« und meint damit nicht nur Urheberrechte, sondern auch Markenzeichen und Patente. Patente sollen die Erfinder von technischen Verfahren dafür belohnen, dass sie ihre Erfindungen dokumentieren und veröffentlichen, indem sie für eine Weile ein Exklusivrecht bekommen, diese Erfindungen auszunutzen (etwa indem sie Anderen gegen Geld gestatten, die Erfindungen anzuwenden). Markenzeichen

stellen sicher, dass niemand unerlaubt die Bekanntheit einer »Marke« ausnutzen darf, um seine eigenen Produkte unter die Leute zu bringen. Zum Beispiel sorgt ein Markenzeichen auf »Coca-Cola« dafür, dass nicht jeder, der eine dunkelbraune Zuckerbrühe brauen kann, diese als »Coca-Cola« in den Handel bringen darf. Die drei Formen von »geistigem Eigentum« sind verwandt, aber verschieden – bei Patenten geht es um Ideen, bei Urheberrechten um die konkrete Umsetzung von Ideen in Werke und bei Markenzeichen um die Verhinderung unsauberer Geschäftspraktiken.

Urheberrecht auf ein Werk erhält man automatisch, indem man das Werk verfasst oder erstellt – jedenfalls sofern das Werk eine gewisse minimale eigene Kreativität beinhaltet. Patente müssen beim Patentamt beantragt werden und werden auf Neuheit geprüft. Markenrechte kann man sich ebenfalls eintragen lassen oder erhält sie dadurch, dass man eine Marke eine Weile lang verwendet und als Anbieter eines Produkts von der Öffentlichkeit damit in Verbindung gebracht wird.

Auch Software für Computer (die man als eine Art von Schriftwerk ansehen kann und in der mitunter erhebliche Kreativität steckt) unterliegt dem Schutz des Urheberrechts. Das heißt, dass es grundsätzlich nicht erlaubt ist, ein Programm oder ein komplettes Softwarepaket einfach so zu kopieren, ohne dass der Urheber (der Programmierer oder dessen Firma) das ausdrücklich gestattet hat.

In der Frühzeit des Computers war es nicht üblich, Software zu verkaufen. Man bekam sie entweder kostenlos zu seinem Computer dazu (der war ja teuer genug – Millionen von Dollar oder Mark) oder man schrieb sie selbst. In den Universitäten der 1960er und der Heimcomputerszene der 1970er Jahre war es völlig normal, Programme zu tauschen und zu kopieren, und ein gewisser Bill Gates war 1976 völlig entsetzt darüber, dass sein BASIC-Interpreter für den MITS Altair 8800 zwar sehr verbreitet war und er jede Menge Lob dafür bekam, aber fast niemand es nötig fand, ihm den Kaufpreis dafür zu bezahlen! Das war von den Benutzern natürlich nicht korrekt, aber schon die *Idee*, dass Software etwas kosten sollte, war damals einfach so absurd, dass die wenigsten überhaupt auf den Gedanken kamen.

In den späten 1970er und 1980er Jahren wurde es mit der zunehmenden Verbreitung von Computern in Büros immer selbstverständlicher, Software nicht zu verschenken, sondern zu verkaufen. Nicht nur das – die Softwarefirmen verkauften nur den ausführbaren Maschinencode, nicht den Quellcode, an dem man hätte sehen können, wie die Software funktioniert, oder gar Veränderungen hätte machen können. Irgendwann war ein Punkt erreicht, wo Richard M. Stallman, damals Forscher am MIT, beschloss, dieser Entwicklung entgegenzuwirken und an einem System zu arbeiten, das die Kultur des Teilens, wie sie in den 1960er und 1970er Jahren üblich war, wieder in den Vordergrund stellen sollte. Dieses »GNU«-System[3] ist zwar immer noch nicht ganz fertig, aber viele Komponenten davon kommen heute in Linux-Systemen zum Einsatz.

freie Software

Auf Richard M. Stallman (oft kurz »RMS« genannt) geht das Konzept der »freien Software« zurück. »Frei« heißt in diesem Zusammenhang nicht »kostenlos«, sondern dass der Benutzer »frei« ist, verschiedene Dinge zu tun, die er mit proprietärer Software nicht könnte oder dürfte[4]. RMS nennt eine Software »frei«, wenn vier Bedingungen, die »vier Freiheiten«, erfüllt sind:

- Man muss das Programm für jeden beliebigen Zweck benutzen dürfen (Freiheit 0)

[3] »GNU« ist eine »rekursive« Abkürzung für *GNU's Not Unix*.

[4] Die *Free Software Foundation*, Stallmans Stiftung zur Förderung freier Software, nennt das *free as in speech, not as in beer*, also »frei wie freie Rede, nicht wie Freibier«.

- Man muss studieren können, wie das Programm funktioniert, und es an seine eigenen Bedürfnisse anpassen können (Freiheit 1)
- Man muss das Programm weitergeben dürfen, um seinem Nächsten zu helfen (Freiheit 2)
- Man muss das Programm verbessern und die Verbesserungen veröffentlichen dürfen, um der Allgemeinheit zu nutzen (Freiheit 3).

Zugang zum Quellcode ist Vorbedingung für die Freiheiten 1 und 3.

Die Idee der freien Software fand grundsätzlich Anklang, aber die Ziele von RMS und der *Free Software Foundation* (FSF) wurden doch oft missverstanden. Insbesondere Firmen störten sich an dem Wort »frei«, das sie trotz entsprechender Klarstellungen zu sehr mit dem Wort »kostenlos« verwechselten. Ende der 1990er Jahre gründeten Eric S. Raymond, Bruce Perens und Tim O'Reilly, die *Open Source Initiative* (OSI), deren Ziel es war, freier Software zu besserem und weniger ideologischem Marketing zu verhelfen. Die FSF war nicht begeistert von dieser »Verwässerung«, und die Kontroverse ist trotz der doch sehr ähnlichen Ziele von FSF und OSI auch heute noch nicht komplett beigelegt (wobei auch die ausgeprägten Egos einiger der maßgeblich beteiligten Personen eine Rolle spielen könnten).

Während das »frei« in »freie Software« die Verwechslung mit »kostenlos« nahelegt, kann man den »offengelegten Quellcode« in »Open Source Software« auch so interpretieren, dass der Quellcode zwar zu besichtigen ist, aber keine Änderung oder Weitergabe zuläßt – beides eigentlich Kernprinzipien auch der OSI. In diesem Sinne ist keiner der beiden Begriffe wirklich zu 100% treffend, so dass in der Szene oft von »FOSS« (kurz für *free and open-source software*) oder gar »FLOSS« (*free, libre and open-source software*, wobei das *libre* den Begriff der »Freiheit« unterstützen soll) die Rede ist.

FOSS
FLOSS

Wie kann man mit freier Software Geld verdienen, wenn jeder die Software ändern und weitergeben darf? Eine sehr berechtigte Frage. Hier sind ein paar Ideen für »Open-Source-Geschäftsmodelle«:

Geschäftsmodelle

- Sie können für die freie Software Zusatzleistungen wie Unterstützung, Dokumentation oder Training anbieten und sich dafür bezahlen lassen (für uns als Linup Front GmbH funktioniert das sehr gut, und auch das LPI kann anscheinend vom Linux-Zertifizierungsgeschäft einigermaßen leben).
- Sie können auf Anfrage kundenspezifische Weiterentwicklungen oder Erweiterungen erstellen und sich Ihre Arbeitszeit vergüten lassen (auch wenn das Ergebnis der Entwicklung dann Bestandteil der allgemein verfügbaren Version wird). Das funktioniert sogar für freie Software, die Sie nicht ursprünglich selber geschrieben haben.

  Im »traditionellen« Modell der proprietären Softwareentwicklung hat zunächst einmal der Hersteller ein Monopol auf Änderungen und Weiterentwicklungen. Als Kunde einer solchen Firma haben Sie ein Problem, wenn der Hersteller das Produkt abkündigt oder selbst verschwindet (etwa insolvent geht oder von seinem größten Konkurrenten aufgekauft wird), weil Sie dann mit Kosten und Mühe eine Software eingeführt haben, die keine Zukunft hat. Bei freier Software dagegen können Sie immer versuchen, jemanden zu finden, der statt der ursprünglichen Herstellerfirma die Unterstützung übernimmt – notfalls tun Sie sich dafür mit anderen Anwendern der Software zusammen, die genausowenig im Regen stehen wollen.

- Wenn Sie ein Softwarepaket anbieten, können Sie eine Version davon als FOSS vertreiben, die die Grundfunktionalität abdeckt, und hoffen, dass genug Leute, die von der FOSS-Version angelockt wurden, die proprietäre »Vollversion« kaufen, um wirklich etwas getan zu kriegen. (Der Jargon dafür ist *open core*.)

Dies ist eine zweischneidige Sache: Zum einen ist es natürlich schön, wenn es mehr freie Software gibt, aber zum anderen läuft es oft darauf hinaus, dass man die proprietäre Version *braucht*, weil wichtige Funktionen in der freien Version nicht zur Verfügung stehen und es zuviel Aufwand wäre, diese in eigener Regie »nachzurüsten«. Die »freie« Version hat dann in erster Linie die Rolle eines Werbeträgers, wenn die Herstellerfirma als modern und »open-source-freundlich« gelten möchte, ohne das aber wirklich zu meinen.

2.2.2 Lizenzen

Wie wird eine Software zur »freien« oder »Open-Source«-Software? Wir hatten gesagt, dass gewisse Rechte – etwa das Recht zur Vervielfältigung oder zur Veränderung eines Werks – dem Urheber des Werks vorbehalten sind, er diese Rechte aber auch an andere weitergegeben kann. Dies geschieht (wenn es geschieht) auf dem Weg einer »Lizenz«, also eines juristischen Textes, der festlegt, welche Rechte der Empfänger der Software durch das Kaufen, Herunterladen, … an der Software erwirbt.

Lizenz

Schon das Urheberrecht als solches erlaubt es dem Käufer (oder rechtmäßigem Herunterlader) einer Software, die Software zum Beispiel auf einem Computer zu installieren, aufzurufen und laufen zu lassen. Das resultiert einfach daraus, dass ihm die Software vom Urheber zugänglich gemacht wurde – schließlich hat es keinen Zweck, jemandem ein Programm zu verkaufen, das er anschließend nicht benutzen darf. (Umgekehrt hat der, der dem Programmautor Geld für das Programm gegeben hat, grundsätzlich sogar ein *Recht* darauf, als Gegenleistung das Programm benutzen zu dürfen.) Andere Aktionen, etwa das unkontrollierte Kopieren und Weitergeben oder das Verändern des Programms, schließt das Urheberrecht dagegen aus, und wenn der Programmautor jemandem das Recht dazu geben möchte, muss er es in die Lizenz schreiben.

EULA

Proprietäre Programme kommen oft mit einem »Endbenutzer-Lizenzabkommen« (*end-user license agreement* oder EULA), das der Käufer akzeptieren muss, bevor er die Software tatsächlich benutzen darf. Der Verkäufer der Software benutzt das EULA, um dem Käufer Dinge zu verbieten, die er gemäß dem Urheberrechtsgesetz eigentlich hätte – etwa die Software »gebraucht« weiterzuverkaufen oder in der Öffentlichkeit schlecht (oder überhaupt) über die Software zu reden. Ein solches EULA ist ein Vertrag, der von beiden Seiten akzeptiert werden muss, und die rechtlichen Hürden dafür liegen (zumindest in Deutschland) ziemlich hoch. Zum Beispiel müssen einem Software-Käufer die EULA-Bedingungen vor dem Kauf bekannt sein oder es muss zumindest möglich sein, die Software unbenutzt zurückzugeben, wenn dem Käufer die Bedingungen nicht zusagen.

Open-Source-Lizenzen

Die Lizenzen für freie und Open-Source-Software dagegen dienen dazu, dem Erwerber der Software Dinge zu erlauben, die er laut Urheberrechtsgesetz eigentlich sonst *nicht* dürfte. Sie versuchen in der Regel nicht, den *Gebrauch* der Software einzuschränken, sondern ordnen vor allem Fragen der Veränderung und Weitergabe der Software. In diesem Sinne stellen sie den Erwerber der Software nicht schlechter, als er bei irgendeinem Sachkauf zu erwarten hätte. Deshalb sind Freie-Software-Lizenzen im Gegensatz zu EULAs normalerweise keine Verträge, die der Empfänger der Software akzeptieren muss, damit sie wirksam werden, sondern einseitige Erklärungen des Softwareautors, und die von ihnen eingeräumten Rechte eine Art »Sonderbonus« zum einfachen Nutzungsrecht.

Inzwischen gibt es einen ganzen Zoo von Lizenzen, die die Anforderungen an freie oder Open-Source-Software erfüllen. Die bekannteste Freie-Software-Lizenz ist die *General Public License* (GPL) des GNU-Projekts von Richard M. Stallman, aber es gibt einige andere. Die OSI „zertifiziert“ Lizenzen, die ihrer Meinung nach

den Open-Source-Gedanken verkörpern, genau wie die FSF Lizenzen gutheißt, die die »vier Freiheiten« sichern. Welche das im Einzelfall sind, erfahren Sie auf den Web-Seiten dieser Organisationen.

Wenn Sie gedenken, ein freies oder Open-Source-Softwareprojekt zu starten, dann können Sie sich natürlich auch eine Lizenz überlegen, die den Anforderungen der FSF oder OSI genügt. Empfehlenswerter ist es jedoch, eine bestehende, bereits anerkannte Lizenz für das Projekt zu übernehmen. Zum einen müssen Sie dann nicht bei FSF oder OSI um die Anerkennung Ihrer Lizenz werben, und zum anderen sind die gängigen Lizenzen bereits juristisch hinreichend abgeklopft worden, um als einigermaßen wasserdicht gelten zu können – als Amateur auf dem Gebiet des Vertrags- oder Lizenzrechts könnten Sie durchaus wichtige Dinge übersehen, die Sie dann später in Schwierigkeiten bringen könnten.

FOSS und Urheberrecht

Es ist eine sehr wichtige Beobachtung, dass es den Proponenten freier oder Open-Source-Software in keiner Weise darum geht, das Urheberrecht für Software komplett abzuschaffen. Tatsächlich kann freie Software, so wie wir sie kennen, überhaupt nur funktionieren, weil es das Urheberrecht *gibt* und die Urheber von Software darum das Recht haben, Veränderung und Weitergabe der Software an Bedingungen zu koppeln wie dass der Empfänger der Software wiederum das Recht zur Veränderung und Weitergabe bekommt. Ohne Urheberrecht könnte jeder sich (wie zu den Zeiten Gutenbergs) beliebig bei der verfügbaren Software bedienen, und zentrale Grundsätze wie die »vier Freiheiten« wären in großer Gefahr, weil es möglich wäre, nur zu hamstern und nie zu teilen.

2.2.3 Die GPL

Der Linux-Kern und weite Teile dessen, was man sonst unter »Linux« versteht, stehen unter der *General Public License* (GPL). Die GPL wurde von RMS für das GNU-Projekt entworfen und soll sicherstellen, dass Software, die einmal unter der GPL steht, auch weiter unter der GPL bleibt (eine solche Lizenz nennt man auch *Copyleft*-Lizenz). Und das funktioniert ungefähr so:

- GPL-Software muss im Quellcode zur Verfügung stehen und darf für beliebige Zwecke benutzt werden.
- Es ist ausdrücklich erlaubt, diesen Quellcode zu ändern und in Originalform oder geänderter Form weiterzugeben, solange der Empfänger dieselben Rechte unter der GPL erhält.
- Es ist ebenfalls ausdrücklich erlaubt, GPL-Software in ausführbarer Form weiterzugeben oder sogar zu verkaufen. In diesem Fall muss aber der Quellcode (mitsamt den GPL-Rechten) mitgeliefert oder für einen bestimmten Zeitraum auf Anfrage zugänglich gemacht werden.

  »Quellcode« heißt in diesem Zusammenhang »alles, was nötig ist, um die Software auf einem Computer zum Laufen zu bringen«. Was das im Einzelfall bedeutet, also ob zum Beispiel auch die kryptografischen Schlüssel dazugehören, die nötig sind, um auf einem entsprechend abgesicherten Rechner einen angepassten Linux-Kernel zu starten, ist Gegenstand hitziger Diskussionen.

  Wenn jemand eine GPL-Software für Geld kauft, bekommt er damit natürlich das Recht, die Software nicht nur auf allen seinen Computern zu installieren, sondern sie auch zu kopieren und weiterzuverkaufen (unter der GPL). Das hat einerseits die Konsequenz, dass es nicht viel Sinn ergibt, GPL-Software „pro Rechner“ zu verkaufen, aber andererseits auch den angenehmen Nebeneffekt, dass es die Preise etwa für Linux-Distributionen halbwegs vernünftig hält.

- Wenn Sie ein neues Programm schreiben, das Teile eines GPL-Programms beinhaltet, müssen Sie das neue Programm als »abgeleitetes Werk« auch unter die GPL stellen.

  
  Auch hier erhitzen sich die Gemüter an einer Abgrenzung, wie viel und was für Teile von einem GPL-Programm übernommen sein müssen, damit ein anderes Programm als »abgeleitetes Werk« gilt. Wenn man der FSF glaubt, führt schon die Benutzung einer dynamisch ladbaren GPL-Bibliothek in einem Programm dazu, dass das Programm unter die GPL fällt, sogar wenn es selber überhaupt keinen Code der GPL-Bibliothek beinhaltet und darum im Sinne des Urheberrechts auch nicht »abgeleitet« sein kann. Wieviel hiervon auf Wunschdenken beruht und wieviel tatsächlich urheberrechtlich haltbar ist, bedarf grundsätzlich einer gerichtlichen Klärung.

Die GPL bezieht sich ausschließlich auf die Veränderung und Weitergabe der Software und nicht auf deren Einsatz.

GPLv3


Im Moment sind zwei Versionen der GPL in Gebrauch. Die neuere Version 3 (auch als »GPLv3« bezeichnet) wurde Ende Juni 2007 veröffentlicht und unterscheidet sich von der älteren Version 2 (auch »GPLv2«) durch Präzisierungen in Bereichen wie Softwarepatenten, der Kompatibilität mit anderen freien Lizenzen und der Einführung von Restriktionen dafür, Änderungen an der theoretisch »freien« Software von Geräten unmöglich zu machen, indem man sie durch spezielle Hardware ausschließt (die »Tivoisierung«, nach einem digitalen Videorekorder auf Linux-Basis, dessen Linux-Kern man nicht verändern oder austauschen kann). Die GPLv3 erlaubt ihren Benutzern auch das Hinzufügen weiterer Klauseln. – Die GPLv3 stieß nicht auf die uneingeschränkte Zustimmung der Szene, so dass viele Projekte, allen voran der Linux-Kernel, mit Absicht bei der einfacheren GPLv2 geblieben sind. Viele Projekte werden auch »unter der GPLv2 oder einer neueren Version der GPL« vertrieben, so dass Sie sich entscheiden können, welcher Version der GPL Sie bei der Weitergabe oder Änderung solcher Software folgen wollen.


Es gilt unter Entwicklern freier Software als guter Stil, Beiträge zu einem Projekt unter dieselbe Lizenz zu stellen, die das Projekt schon benutzt, und die meisten Projekte bestehen tatsächlich darauf, dass das zumindest für solchen Code gilt, der in die »offizielle« Version einfließen soll. Manche Projekte bestehen sogar auf *copyright assignments*, bei denen der Urheber des Codes seine Rechte an das Projekt (oder eine geeignete Organisation) abtritt. Dieser Schritt hat den Vorteil, dass urheberrechtlich nur das Projekt für den Code verantwortlich ist und Lizenzverstöße – die nur der Rechte-Inhaber verfolgen kann – leichter geahndet werden können. Ein entweder gewollter oder ausdrücklich unerwünschter Nebeneffekt ist auch, dass es einfacher möglich ist, die Lizenz für das ganze Projekt zu ändern, denn auch das darf nur der Rechte-Inhaber.


Im Falle des Linux-Betriebssystemkerns, wo ausdrücklich keine *copyright assignments* verlangt werden, ist eine Lizenzänderung praktisch sehr schwierig bis ausgeschlossen, da der Code ein Flickenteppich von Beiträgen von über tausend Autoren ist. Die Angelegenheit wurde im Zuge der Veröffentlichung der GPLv3 erörtert, und man war sich einig, dass es ein riesengroßes Projekt wäre, die urheberrechtliche Provenienz jeder einzelnen Zeile des Linux-Quellcodes zu klären und die Zustimmung der Autoren zu einer Lizenzänderung einzuholen. Manche Linux-Entwickler wären auch vehement dagegen, andere sind nicht mehr zu finden oder sogar verstorben, und der betreffende Code müsste durch etwas Ähnliches mit klarem Copyright ersetzt werden. Zumindest Linus Torvalds ist aber nach wie vor Anhänger der GPLv2, so dass das Problem sich in der Praxis (noch) nicht wirklich stellt.

GPL und Geld

Die GPL sagt nichts über den möglichen Preis des Produkts aus. Es ist absolut legal, dass Sie Kopien von GPL-Programmen verschenken oder auch Geld dafür verlangen, solange Sie auch den Quellcode mitliefern oder auf Anfrage verfügbar machen und der Empfänger der Software auch die GPL-Rechte bekommt. GPL-Software ist damit also nicht unbedingt »Freeware«.

Mehr Informationen erhalten Sie durch Lesen der GPL [GPL91], die übrigens jedem entsprechenden Produkt (auch Linux) beiliegen muss.

Die GPL gilt als konsequenteste der freien Lizenzen in dem Sinne, dass sie – wie erwähnt – sicherzustellen versucht, dass einmal unter der GPL veröffentlichter Code auch frei *bleibt*. Es haben schon öfters Firmen versucht, GPL-Code in ihre eigenen Produkte zu integrieren, die dann nicht unter der GPL freigegeben werden sollten. Allerdings haben diese Firmen bisher immer nach einer nachdrücklichen Ermahnung durch (meist) die FSF als Rechteinhaberin eingelenkt und die Lizenzbestimmungen eingehalten. Zumindest in Deutschland ist die GPL auch schon gerichtlich für gültig erklärt worden – einer der Linux-Kernelprogrammierer konnte vor dem Landgericht Frankfurt ein Urteil gegen die Firma D-Link (einen Hersteller von Netzwerkkomponenten, hier einem NAS-Gerät auf Linux-Basis) erwirken, in dem letztere zu Schadensersatz verklagt wurde, weil sie bei der Verbreitung ihres Geräts den GPL-Auflagen nicht genügte [GPL-Urteil06].

Warum funktioniert die GPL? Manche Firma, der die Anforderungen der GPL lästig waren, hat versucht, sie für ungültig zu erklären oder erklären zu lassen. So wurde sie in den USA schon als »unamerikanisch« oder »verfassungswidrig« apostrophiert; in Deutschland wurde versucht, das Kartellrecht heranzuziehen, da die GPL angeblich illegale Preisvorschriften macht. Die Idee scheint zu sein, dass Software, die unter der GPL steht, für jeden beliebig benutzbar ist, wenn mit der GPL irgendetwas nachweislich nicht stimmt. Alle diese Angriffe verkennen aber eine Tatsache: Ohne die GPL hätte niemand außer dem ursprünglichen Autor überhaupt das Recht, mit dem Code irgendetwas zu tun, da Aktionen wie das Weitergeben oder gar Verkaufen nach dem Urheberrecht nur ihm vorbehalten sind. Fällt die GPL weg, dann stehen alle anderen Interessenten an dem Code also viel schlechter da als vorher.

Ein Prozess, bei dem ein Softwareautor eine Firma verklagt, die seine GPL-Software vertreibt, ohne sich an die GPL zu halten, würde aller Wahrscheinlichkeit nach so aussehen:

Richter Was ist jetzt das Problem?

Softwareautor Herr Vorsitzender, die Beklagte hat meine Software vertrieben, ohne die Lizenz dafür einzuhalten.

Richter (zum Rechtsanwalt der Beklagten) Stimmt das?

An dieser Stelle kann die beklagte Firma »Ja« sagen und der Prozess ist im Wesentlichen vorbei (bis auf das Urteil). Sie kann auch »Nein« sagen, aber sie muss dann begründen, warum das Urheberrecht für sie nicht gilt. Ein unangenehmes Dilemma und der Grund, warum wenige Firmen sich diesen Stress machen und die meisten GPL-Streitigkeiten außergerichtlich geklärt werden.

Verstößt ein Hersteller proprietärer Software gegen die GPL (etwa indem er ein paar hundert Zeilen Quellcode aus einem GPL-Projekt in seine Software integriert), bedeutet das übrigens in keinem Fall, dass seine Software dadurch automatisch komplett unter die GPL fällt und offengelegt werden muss. Es bedeutet zunächst nur, dass er GPL-Code gegen dessen Lizenz vertrieben hat. Lösen kann der Hersteller das Problem auf verschiedene Arten:

- Er kann den GPL-Code entfernen und durch eigenen Code ersetzen. Die GPL wird dann irrelevant für sein Programm.

- Er kann mit dem Urheberrechtsinhaber des GPL-Codes verhandeln und zum Beispiel eine Zahlung von Lizenzgebühren vereinbaren (wenn dieser aufzutreiben ist und mitmacht). Siehe auch den Abschnitt über Mehrfachlizenzierung weiter unten.
- Er kann sein komplettes Programm *freiwillig* unter die GPL stellen und so den Anforderungen der GPL entsprechen (die unwahrscheinlichste Methode).

Unabhängig davon könnte für den vorher stattgefundenen Lizenzverstoß Schadensersatz verhängt werden. Der urheberrechtliche Status der proprietären Software bleibt davon allerdings unberührt.

2.2.4 Andere Lizenzen

Außer der GPL gibt es auch noch andere Lizenzen, die im FOSS-Umfeld üblich sind. Hier ist ein kleiner Überblick:

BSD-Lizenz Die BSD-Lizenz stammt aus dem Umfeld der Unix-Distribution der University of California in Berkeley und ist absichtlich sehr einfach gehalten: Der Empfänger der Software bekommt im Wesentlichen das Recht, mit der Software zu machen, was er will, solange er nicht den Eindruck erweckt, als stünde die Universität (oder im erweiterten Sinne der ursprüngliche Softwareautor) hinter ihm. Eine Haftung für das Programm wird so weit wie möglich ausgeschlossen. Der Lizenztext muss im Quellcode des Programms erhalten bleiben und – beim Vertrieb des Programms oder veränderter Versionen davon in ausführbarer Form – in der Dokumentation wiedergegeben werden.

Früher enthielt die BSD-Lizenz auch die Anforderung, dass in Werbematerial für Software oder Systeme, die BSD-Code enthielten, auf diesen Umstand aufmerksam gemacht werden musste. Diese Klausel wurde jedoch inzwischen entfernt.

Im Gegensatz zur GPL versucht die BSD-Lizenz nicht, den Quellcode der Software in der Öffentlichkeit zu halten. Wer sich BSD-lizenzierte Software besorgt, kann sie grundsätzlich in seine eigenen Programme integrieren und diese Programme nur in ausführbarer Form weitergeben (die GPL würde verlangen, dass auch der Quellcode weitergegeben werden muss).

Kommerzielle Softwarefirmen wie Microsoft oder Apple, die von GPL-Software gemeinhin nicht besonders begeistert sind, haben in der Regel kein Problem mit BSD-lizenzierter Software. Windows NT zum Beispiel verwendete eine ganze Weile lang den TCP/IP-Netzwerkcode von BSD (in angepasster Form), und auch große Teile des Betriebssystemkerns von OS X auf dem Macintosh beruhen auf BSD.

In der FOSS-Szene gehen seit langem die Meinungen darüber auseinander, ob nun die GPL oder die BSD-Lizenz »freier« ist. Einerseits kann man sagen, dass man mit BSD-lizenzierter Software als Empfänger mehr anfangen kann, dass also in absoluten Begriffen die BSD-Lizenz mehr Freiheit einräumt. Dem entgegen steht die Sicht der GPL-Verfechter, die sagen, dass es wichtiger ist, dass Code für alle frei bleibt, als dass er in proprietären Systemen verschwindet, und dass ein Kennzeichen größerer Freiheit der GPL darum darin besteht, dass diejenigen, die sich aus dem Vorrat der GPL-Software bedienen, auch verpflichtet werden, etwas zurückzugeben.

Apache-Lizenz Die Apache-Lizenz ähnelt der BSD-Lizenz darin, dass sie die Nutzung und Übernahme von lizenziertem Code gestattet, ohne (wie die GPL) zu verlangen, dass modifizierter Apache-lizenzierter Code wieder

der Öffentlichkeit zugänglich gemacht wird. Sie ist komplexer als die BSD-Lizenz, aber enthält auch Klauseln über die Nutzung von Patenten und Markenzeichen und andere Details.

Mozilla Public License (MPL) Die Mozilla-Lizenz (die zum Beispiel für den Firefox-Browser gilt) ist eine Mischung aus der BSD-Lizenz und der GPL. Sie ist eine »schwache Copyleft-Lizenz«, da sie einerseits verlangt, dass man Code, den man unter der MPL erhält, unter der MPL weitergeben muss (ähnlich wie bei der GPL), andererseits aber auch gestattet, dass man dem MPL-Code Code unter anderen Lizenzen hinzufügt, der dann nicht unter der MPL weitergegeben werden muss.

Creative Commons

Der Erfolg der FOSS-Gemeinde brachte den Juristen Lawrence (Larry) Lessig dazu, das Konzept auf andere Werke außer Software anzuwenden. Ziel war, den Fundus von Kulturgütern wie Büchern, Bildern, Musik, Filmen, … zu vergrößern, die anderen zur freien Übernahme, Veränderung und Weitergabe zur Verfügung stehen. Da die gängigen FOSS-Lizenzen sehr auf Software abgestimmt sind, wurden hierfür die *Creative-Commons*-Lizenzen entwickelt, die es Urhebern ermöglichen, ihre Werke kontrolliert der Allgemeinheit zu übergeben. Dabei können sie verschiedene Einschränkungen festlegen, wie dass das Werk nur unverändert weitergegeben werden darf, dass Änderungen erlaubt sind, aber (GPL-mäßig) Empfänger der Änderungen wieder Änderungen machen dürfen, oder dass eine kommerzielle Nutzung des Werks ausgeschlossen ist.

Public Domain

»Public Domain« beschreibt Kulturgüter, die *gemeinfrei* sind, also keinen Urheberrrechten mehr unterliegen. Während es in der angelsächsischen Rechtstradition möglich ist, ein Werk (also auch eine Software) ausdrücklich in die *public domain* zu stellen, geht das in Deutschland nicht, da das Konzept im deutschen Urheberrecht gar nicht existiert. Hierzulande werden Werke erst 70 Jahre nach dem Ableben des letzten Urhebers automatisch gemeinfrei. Bis das erste Computerprogramm auf diesem Weg allgemein nutzbar wird, wird es also noch ein bisschen dauern.

Es bestehen allerdings gute Chancen, dass überhaupt keine Werke, die nach ca. 1930 erstellt wurden, jemals gemeinfrei werden. In den USA jedenfalls wird die Copyright-Frist jedesmal vom Kongress verlängert, wenn die Gefahr besteht, dass eine gewisse Comic-Maus in die *public domain* übergeht. Der Rest der Welt zieht dann normalerweise bereitwillig nach. Es ist nicht ganz klar, warum sogar noch die *Urenkel* von Walt Disney dank der Kreativität des großen Zeichners Geld scheffeln müssen wie Dagobert Duck, aber wie so oft hängt das vor allem davon ab, wer die cleversten Lobbyisten hat.

Mehrere Lizenzen

Grundsätzlich kann der Inhaber des Urheberrechts für eine Software diese Software auch gleichzeitig unter mehreren Lizenzen vertreiben – etwa für FOSS-Entwickler unter der GPL und für Firmen, die ihren eigenen Quellcode nicht offenlegen möchten, unter einer proprietären Lizenz. Das lohnt sich natürlich vor allem für Bibliotheken, die andere Entwickler in ihre eigenen Programme integrieren. Wer proprietäre Software entwickeln möchte, kann sich dann von den Anforderungen der GPL »freikaufen«.

Übungen

2.3 [!1] Welche der folgenden Aussagen über die GPL stimmen und welche sind falsch?

1. GPL-Programme dürfen nicht verkauft werden.
2. GPL-Programme dürfen von Firmen nicht umgeschrieben und zur Grundlage eigener Produkte gemacht werden.
3. Der Urheber eines GPL-Programms darf das Programm auch unter einer anderen Lizenz vertreiben.

4. Die GPL gilt nicht, weil man die Lizenz erst zu sehen bekommt, nachdem man das Programm schon hat. Damit Lizenzbestimmungen gültig werden, muss man sie vor dem Erwerb der Software sehen und ihnen zustimmen können.

2.4 [2] Vergleichen Sie die »vier Freiheiten« der FSF mit den *Debian Free Software Guidelines* (`http://www.debian.org/social_contract#guidelines`) des Debian-Projekts (siehe Abschnitt 2.4.4). Welche Definition freier Software gefällt Ihnen besser und warum?

2.3 Wichtige freie Programme

2.3.1 Überblick

Linux ist ein leistungsfähiges und elegantes Betriebssystem, aber das schönste Betriebssystem ist nichts wert ohne Programme, die darauf laufen. In diesem Abschnitt präsentieren wir eine Auswahl der wichtigsten freien bzw. Open-Source-Programme, die auf typischen Linux-PCs zu finden sind.

Dass irgendein Programm nicht enthalten ist, heißt nicht, dass wir das Programm nicht gut finden. Der Platz ist halt begrenzt, und wir versuchen, vor allem diejenige Software abzudecken, die das LPI in den Prüfungsrichtlinien erwähnt hat (*just in case*).

2.3.2 Büro- und Produktivitätsprogramme

Die meisten Computer werden wahrscheinlich für »Büroanwendungen« wie das Schreiben von Briefen und Memos, Diplom- und Doktorarbeiten, die Auswertung von Daten mit Tabellenkalkulations- und Grafikprogrammen und Ähnliches verwendet. Außerdem verbringen viele Anwender große Teile ihrer Computer-Zeit im Internet oder mit dem Lesen und Schreiben von E-Mail. Kein Wunder, dass es auf diesem Gebiet jede Menge gute freie Software gibt.

Die meisten der Programme in diesem Abschnitt stehen nicht nur für Linux zur Verfügung, sondern auch für Windows, OS X oder gar andere Unix-Varianten. Das macht es möglich, Benutzer langsam auf ein FOSS-Umfeld »umzugewöhnen«, indem Sie zum Beispiel auf einem Windows-Rechner zuerst LibreOffice, Firefox und Thunderbird anstelle von Microsoft Office, Internet Explorer und Outlook installieren, bevor Sie das Betriebssystem selbst durch Linux ersetzen. Wenn Sie das geschickt anfangen[5], merken die Benutzer den Unterschied vielleicht gar nicht.

OpenOffice.org ist seit Jahren das Flaggschiff-Produkt der FOSS-Gemeinde, wenn es um große Büropakete geht. Es hat vor vielen Jahren als »StarOffice« angefangen und wurde schließlich von Sun aufgekauft und (in einer leicht abgespeckten Version) als freie Software in Umlauf gebracht. OpenOffice.org enthält alles, was man so in einem Office-Paket erwartet – Textverarbeitung, Tabellenkalkulation, Präsentationsprogramm, Geschäftsgrafik, Datenbank, … –, und kann auch einigermaßen mit den Datenformaten des großen Mitbewerbers von Microsoft umgehen.

LibreOffice Nachdem Sun von Oracle übernommen wurde und die Zukunft von OpenOffice.org zunächst in den Sternen stand, taten sich einige der wesentlichen OpenOffice.org-Entwickler zusammen und veröffentlichten ihre eigene Version von OpenOffice.org unter dem Namen »LibreOffice«. Die beiden Pakete werden im Moment nebeneinander her entwickelt (Oracle hat

[5] Zum Beispiel könnten Sie Ihren Schäflein das aktuelle Ubuntu als »Windows-8-Beta« unterschieben …

OpenOffice.org der Apache Software Foundation übergeben) – sicherlich kein optimaler Zustand, aber es ist unklar, ob und wie es zu einer »Wiedervereinigung« kommen wird.


Die meisten großen Linux-Distributionen liefern inzwischen LibreOffice aus, das aktiver weiterentwickelt und – vor allem – aufgeräumt wird.

Firefox ist inzwischen der beliebteste Web-Browser und wird von der Mozilla Foundation in Umlauf gebracht. Firefox ist sicherer und effizienter als der bisherige »Platzhirsch« (Microsofts Internet Explorer), kann mehr und entspricht eher den Standards für das World Wide Web. Außerdem gibt es einen großen Zoo an Erweiterungen, mit dem Sie Firefox an Ihre eigenen Anforderungen anpassen können.

Chromium ist die FOSS-Variante des Google-Browsers »Chrome«. Chrome hat in der letzten Zeit angefangen, Firefox mächtig Konkurrenz zu machen – auch Chrom{e,ium} ist ein leistungsfähiger, sicherer Browser mit zahlreichen Erweiterungen. Vor allem die Google-Webangebote sind auf den Google-Browser abgestimmt und laufen dort besonders gut.

Thunderbird ist ein Mail-Programm von der Mozilla Foundation. Es hat große Teile seines Unterbaus mit dem Firefox-Browser gemeinsam und bietet – wie Firefox – einen reichen Fundus an Erweiterungen für verschiedene Zwecke.

2.3.3 Grafik- und Multimedia-Werkzeuge

Grafik und Multimedia ist seit jeher die Domäne der Macs (auch wenn die Windows-Seite da auch nette Programme zu bieten hat). Zugegebenermaßen fehlt unter Linux noch ein echtes Äquivalent zu Programmen wie Adobes Photoshop, aber die Software, die es gibt, ist auch nicht zu verachten.

The GIMP ist ein Programm, mit dem Sie Fotos und ähnliche Vorlagen bearbeiten können. Es kann noch nicht ganz mit Photoshop mithalten (zum Beispiel fehlen Funktionen bei der Druckvorstufe), aber ist für viele Anwendungen absolut brauchbar und bietet zum Beispiel bei der Erstellung von Grafiken für das World Wide Web ein paar Eigenschaften, die Photoshop nicht genauso bequem zur Verfügung stellt.

Inkscape Wenn The GIMP das Photoshop von Linux ist, dann ist Inkscape das Pendant zu Illustrator – eine leistungsfähige Software zur Erstellung vektorbasierter Grafiken.

ImageMagick ist ein Softwarepaket, mit dem Sie fast beliebige Grafikformate in fast beliebige andere umwandeln können. Außerdem erlaubt es Ihnen die skriptgesteuerte Manipulation von Grafiken auf nahezu endlose verschiedene Arten. Genial für Webserver und andere Umgebungen, wo mit Grafiken gearbeitet werden muss, ohne dass Maus und Monitor zum Einsatz kommen können.

Audacity dient als Mehrspurtonband, Mischpult und Schneidestation für Audiodaten aller Art und ist auch auf Windows und dem Mac populär.

Cinelerra und andere Programme wie KDenlive oder OpenShot sind »nichtlineare Videoeditoren«, die Videos von digitalen Camcordern, TV-Karten oder Webcams schneiden, vertonen, mit Effekten versehen und in verschiedenen Formaten (von YouTube bis zur DVD) ausgeben können.

Blender ist nicht nur ein leistungsfähiger Videoeditor, sondern erlaubt auch das fotorealistische »Rendern« von dreidimensionalen bewegten Szenen und ist damit das angesagte Werkzeug zum Erstellen von Animationsfilmen in professioneller Qualität.

An dieser Stelle ein Hinweis darauf, dass heutzutage kein Hollywood-Blockbuster mehr entsteht, ohne dass Linux mit im Spiel ist: Die Spezialeffekt-Renderfarmen der großen Studios laufen inzwischen alle unter Linux.

2.3.4 Server-Dienste

Ohne Linux wäre das Internet nicht wiederzuerkennen: Die Hunderttausende Server von Google laufen genauso mit Linux wie die Handelssysteme der meisten großen Börsen der Welt (die Deutsche Börse genau wie die Börsen von London und New York), weil nur mit Linux die dafür nötige Leistung realisierbar ist. Tatsächlich wird die meiste Internet-Software heute zuerst auf Linux entwickelt, und auch an Universitäten findet die Forschung in diesem Bereich natürlich auf der quelloffenen Linux-Plattform statt.

Apache ist mit Abstand der beliebteste Web-Server auf dem Internet – mehr als die Hälfte aller Webseiten laufen auf einem Apache-Server.

Es gibt natürlich auch andere gute Web-Server auf Linux – etwa Nginx oder Lighttpd –, aber Apache bleibt der populärste.

MySQL und PostgreSQL sind frei verfügbare relationale Datenbankserver. MySQL eignet sich vor allem für Webseiten, während PostgreSQL ein innovativer und hochleistungsfähiger Datenbankserver für alle möglichen Anwendungen ist.

Postfix ist ein sicherer und extrem leistungsfähiger Mailserver, der für alle Anforderungen vom »Home-Office« bis zum großen Provider oder DAX-Unternehmen gerüstet ist.

2.3.5 Infrastruktur-Software

Auch innerhalb eines lokalen Netzes macht ein Linux-Server eine gute Figur: Zuverlässig, schnell und wartungsarm kann man ihn installieren und vergessen (bis auf die regelmäßigen Sicherungskopien natürlich!).

Samba macht einen Linux-Rechner zu einem Server für Windows-Clients, der Plattenplatz und Drucker für alle Windows-Rechner im Netz zugänglich macht (Linux-Rechner übrigens auch). Mit dem neuen Samba 4 kann ein Linux-Rechner sogar als Active-Directory-Domänencontroller dienen und macht darum einen Windows-Server überflüssig. Stabilität, Effizienz und eingesparte Lizenzkosten sind sehr überzeugende Argumente.

NFS ist das Unix-basierte Äquivalent zu Samba und erlaubt anderen Linux- und Unix-Rechnern im Netz den Zugriff auf die Platten eines Linux-Servers. Linux unterstützt das moderne NFSv4 mit erhöhter Leistung und Sicherheit.

OpenLDAP dient als Verzeichnisdienst zur Strukturierung mittlerer und großer Netze und erlaubt durch seine leistungsfähigen Features zur Verteilung und Replikation von Daten eine hohe Redundanz und Geschwindigkeit bei der Abfrage und Aktualisierung.

DNS und DHCP gehören zur grundlegenden Netzwerkinfrastruktur. Mit BIND unterstützt Linux den Referenz-DNS-Server, und der ISC-DHCP-Server kann mit BIND kooperieren, um auch in sehr großen Netze Clients mit Netzparametern wie IP-Adressen zu versorgen. Dnsmasq ist ein bequem zu administrierender DNS- und DHCP-Server für kleine Netze.

2.3.6 Programmiersprachen und Entwicklung

Linux war von Anfang an ein System von Entwicklern für Entwickler. Entsprechend stehen Übersetzer und Interpreter für alle wichtigen Programmiersprachen zur Verfügung – die GNU-Compilerfamilie deckt zum Beispiel C, C++, Objective C, Java, Fortran und Ada ab. Natürlich werden die populären Skriptsprachen wie Perl, Python, Tcl/Tk, Ruby, Lua oder PHP unterstützt, und auch »Exoten« wie Lisp, Scheme, Haskell, Prolog und Ocaml sind Bestandteil vieler Linux-Distributionen.

Eine reichhaltige Auswahl von Editoren und Hilfswerkzeugen macht die Softwareentwicklung zum Vergnügen. Der Standard-Editor `vi` steht ebenso zur Verfügung wie professionelle Umgebungen zur Programmierung, etwa GNU Emacs oder Eclipse.

Linux taugt außerdem als Entwicklungssystem für »Embedded-Systeme«, also Rechner, die als Bestandteil von Konsumgütergeräten oder spezialisierte »Appliances« entweder selbst unter Linux laufen oder eigene Betriebssysteme mitbringen. Es ist leicht, unter Linux auf dem PC eine Compilerumgebung zu installieren, die zum Beispiel Maschinencode für ARM-Prozessoren erzeugt. Daneben ist Linux auch gut geeignet zur Softwareentwicklung für Android-Smartphones, die von Google mit professionellen Werkzeugen unterstützt wird.

Übungen

2.5 [!1] Von welchen FOSS-Programmen haben Sie schon gehört? Welche davon haben Sie schon selber benutzt? Finden Sie sie besser oder schlechter als proprietäre Alternativen? Wenn ja, warum? Wenn nein, warum nicht?

2.4 Wichtige Linux-Distributionen

2.4.1 Überblick

Wenn jemand etwas sagt wie »Ich habe Linux auf meinem Rechner«, meint er damit in der Regel nicht (nur) Linux, den Betriebssystemkern, sondern eine komplette Softwareumgebung auf der Basis von Linux. Dazu gehören meist die Shell `bash` und die Kommandozeilenwerkzeuge aus dem GNU-Projekt, der X.org-Grafikserver und eine grafische Arbeitsumgebung wie KDE oder GNOME, Produktivitätsprogramme wie LibreOffice, Firefox oder The GIMP und viele andere nette Software aus dem vorigen Abschnitt. Es ist natürlich grundsätzlich möglich, sich alle diese Komponenten selbst aus dem Internet zusammenzusuchen, aber die meisten Linux-Anwender benutzen eine vorgefertigte Software-Sammlung oder »Linux-Distribution«.

Die ersten Linux-Distributionen kamen Anfang 1992 heraus – allerdings wird keine Distribution aus der damaligen Zeit noch aktiv weiterentwickelt, und sie sind zum größten Teil vergessen. Die älteste Distribution, an der tatsächlich noch gearbeitet wird, ist Slackware, die zuerst im Juli 1993 erschien.

Es gibt eine Vielzahl von Linux-Distributionen mit verschiedenen Zielen und Ansätzen und unterschiedlicher Organisationsstruktur. Einige Distributionen werden von Firmen veröffentlicht und mitunter sogar nur gegen Geld verkauft, während andere Distributionen von Freiwilligen-Teams oder Einzelpersonen zusammengestellt werden. Im Folgenden diskutieren wir kurz die wichtigsten Distributionen für den allgemeinen Gebrauch.

Wenn wir Ihre Lieblings-Distribution hier nicht erwähnen, hat das weniger damit zu tun, dass wir sie nicht ausstehen können, sondern mehr damit, dass unser Platz und unsere Zeit begrenzt sind und wir uns (leider!) auf das Wesentliche konzentrieren müssen. Wenn eine Distribution hier nicht

vorkommt, bedeutet das also nicht, dass sie schlecht oder nutzlos ist, sondern nur, dass sie hier nicht vorkommt.

Die Webseite »DistroWatch« (`http://distrowatch.com/`) listet die wichtigsten Linux-Distributionen auf und dient als Anlaufstelle für Neuigkeiten rund um Distributionen. Im Moment sind dort 317 Distributionen (in Worten: dreihundertsiebzehn!) gelistet, wobei diese Zahl, wenn Sie diese Zeilen lesen, wahrscheinlich schon nicht mehr stimmt.

2.4.2 Red Hat

Red Hat (`http://www.redhat.com/`) wurde 1993 unter dem Namen »ACC Corporation« als Vertriebsfirma für Linux- und Unix-Zubehör gegründet. 1995 kaufte der Firmengründer, Bob Young, das Geschäft von Marc Ewing, der 1994 eine Linux-Distribution unter dem Namen »Red Hat Linux« veröffentlicht hatte, und firmierte zu »Red Hat Software« um. 1999 ging Red Hat an die Börse und ist seitdem wahrscheinlich das größte Unternehmen, das ausschließlich auf Linux und Open-Source-Software setzt. Es ist in den »Standard & Poor's 500« vertreten, einem Aktienindex, der (ähnlich dem DAX in Deutschland) als Indikator für die US-amerikanische Wirtschaft dient.

Aus dem ursprünglichen Privatkundengeschäft hat Red Hat sich zurückgezogen (das letzte »Red Hat Linux« erschien im April 2004) und vertreibt heute unter dem Namen »Red Hat Enterprise Linux« (RHEL) eine Distribution für den professionellen Einsatz in Firmen. RHEL wird pro Server lizenziert, wobei Sie damit nicht für die Software bezahlen – die steht wie üblich unter der GPL oder unter anderen FOSS-Lizenzen –, sondern für den Zugriff auf zeitnahe Aktualisierungen und Unterstützung bei Problemen. RHEL ist vor allem für den Einsatz in Rechenzentren gedacht und erlaubt (mit den entsprechenden Zusatzpaketen) zum Beispiel den Aufbau von fehlertoleranten »Clustern«.

»Fedora« (`http://www.fedoraproject.org/`) ist eine von Red Hat maßgeblich gesteuerte Distribution, die als »Testumgebung« für RHEL dient. Neue Software und Ideen werden zuerst in Fedora umgesetzt, und was sich bewährt, taucht früher oder später in RHEL auf. Im Gegensatz zu RHEL wird Fedora nicht verkauft, sondern nur zum kostenlosen Download angeboten; das Projekt wird von einem Komitee geleitet, dessen Mitglieder teils von der Entwicklergemeinde gewählt und teils von Red Hat bestimmt werden. (Der Vorsitzende des Komitees wird von Red Hat bestimmt und hat ein Vetorecht.) Der Fokus auf aktuelle Software und neue Ideen macht für viele Fedora-Anwender den Reiz der Distribution aus, auch wenn das häufige Aktualisierungen bedeutet. Für Anfänger und den Einsatz auf Servern, die zuverlässig laufen sollen, ist Fedora weniger gut geeignet.

Da Red Hat seine Software konsequent unter FOSS-Lizenzen wie der GPL vertreibt, ist es grundsätzlich möglich, ein System zu betreiben, das dem aktuellen RHEL entspricht, ohne dafür Lizenzgebühren an Red Hat zahlen zu müssen.

CentOS
Scientific Linux

Es gibt Distributionen wie CentOS (`http://www.centos.org/`) oder Scientific Linux (`https://www.scientificlinux.org/`), die im Wesentlichen auf RHEL basieren, aber alle Markenzeichen von Red Hat entfernen. Das heißt, dass man im Wesentlichen identische Software bekommt, allerdings ohne die Unterstützung von Red Hat.

Insbesondere CentOS ist so nahe an RHEL dran, dass Red Hat Ihnen bei Bedarf Unterstützung für Ihre CentOS-Rechner verkauft, ohne dass Sie erst RHEL dort installieren müssen.

2.4.3 SUSE

Gründung

Die Firma SUSE wurde 1992 unter dem Namen »Gesellschaft für Software- und

System-Entwicklung« als Unix-Beratungshaus gegründet und schrieb sich entsprechend zuerst »S.u.S.E.«. Eines ihrer Produkte war eine an den deutschen Markt angepasste Version der Linux-Distribution Slackware (von Patrick Volkerding), die ihrerseits von der ersten kompletten Linux-Distribution, *Softlanding Linux System* oder SLS, abgeleitet war. S.u.S.E. Linux 1.0 erschien 1994 und differenzierte sich langsam von Slackware, indem beispielsweise Eigenschaften von Red Hat Linux wie die RPM-Paketverwaltung oder die `/etc/sysconfig`-Datei übernommen wurden. Die erste S.u.S.E.-Linux-Version, die nicht mehr wie Slackware aussah, war die 4.2 von 1996. SuSE (die Punkte wurden irgendwann weggelassen) eroberte bald die Marktführerschaft im deutschsprachigen Raum und veröffentlichte SuSE Linux als »Kiste« in zwei Geschmacksrichtungen, »Personal« und »Professional«; letztere war merklich teurer und enthielt unter anderem mehr Software aus dem Server-Bereich.

Übernahme durch Novell

Im November 2003 kündigte die US-amerikanische Softwarefirma Novell an, die SuSE für 210 Millionen Dollar übernehmen zu wollen; der Handel wurde dann im Januar 2004 perfekt gemacht. (Bei dieser Gelegenheit wurde auch das »U« groß gemacht.) Im April 2011 wurde Novell mitsamt SUSE von der Firma Attachmate übernommen. Attachmate ist in Bereichen wie Terminalemulation, Systemmonitoring und Anwendungsintegration aktiv und hat sich im Linux- und Open-Source-Bereich bisher nicht besonders hervorgetan. Seitdem wird Novell in zwei Einheiten weitergeführt, eine davon ist die SUSE.

Attachmate

Wie Red Hat bietet SUSE ein »Unternehmens-Linux« an, den *SUSE Linux Enterprise Server* (SLES, `http://www.suse.com/products/server/`), der RHEL darin ähnelt, dass er relativ selten erscheint und einen langen Lebenszyklus von 7–10 Jahren verspricht. Daneben gibt es mit dem *SUSE Linux Enterprise Desktop* (SLED) auch eine Distribution, die für den Einsatz auf Arbeitsplatzrechnern gedacht ist. SLES und SLED unterscheiden sich in der Paketauswahl; der Fokus bei SLES liegt mehr auf Serverdiensten und bei SLED eher auf interaktiver Software.

Auch SUSE hat inzwischen wie Red Hat den Schritt gemacht, die »Privatkunden«-Distribution zu öffnen und als »openSUSE« (`http://www.opensuse.org/`) frei verfügbar zu machen (früher gab es die Distribution immer erst mit einigen Monaten Verzögerung zum Download). Im Gegensatz zu Red Hat bietet SUSE aber nach wie vor eine »Kiste« an, die zusätzlich proprietäre Software enthält. Im Gegensatz zu Fedora ist openSUSE eine ernstgemeinte Plattform, die aber trotzdem einen relativ kurzen Lebenszyklus hat.

YaST

Bezeichnendes Merkmal der SUSE-Distributionen ist der »YaST«, ein umfassendes grafisch orientiertes Systemverwaltungswerkzeug.

2.4.4 Debian

Im Gegensatz zu den beiden großen Linux-Distributionsfirmen Red Hat und Novell/SUSE ist das **Debian-Projekt** (`http://www.debian.org/`) ein Zusammenschluss von Freiwilligen, die es sich zum Ziel gesetzt haben, eine hochwertige Linux-Distribution unter dem Namen *Debian GNU/Linux* frei zur Verfügung zu stellen. Das Debian-Projekt wurde am 16. August 1993 von Ian Murdock angekündigt; der Name ist eine Zusammensetzung seines Vornamens mit dem seiner damaligen Freundin (jetzt Ex-Frau) Debra (und wird darum »Debb-Ian« ausgesprochen). Inzwischen umfasst das Projekt über 1000 Freiwillige.

Debian-Projekt

Grundlage

Grundlage von Debian sind drei Dokumente:

- Die *Debian Free Software Guidelines* (DFSG) definieren, welche Software im Sinne des Projekts als „frei“ gilt. Das ist wichtig, denn nur DFSG-freie Software kann Teil der eigentlichen Debian-GNU/Linux-Distribution sein. Das Projekt vertreibt auch nichtfreie Software, diese ist jedoch auf den Distributionsservern strikt von der DFSG-freien Software getrennt: Letztere steht in

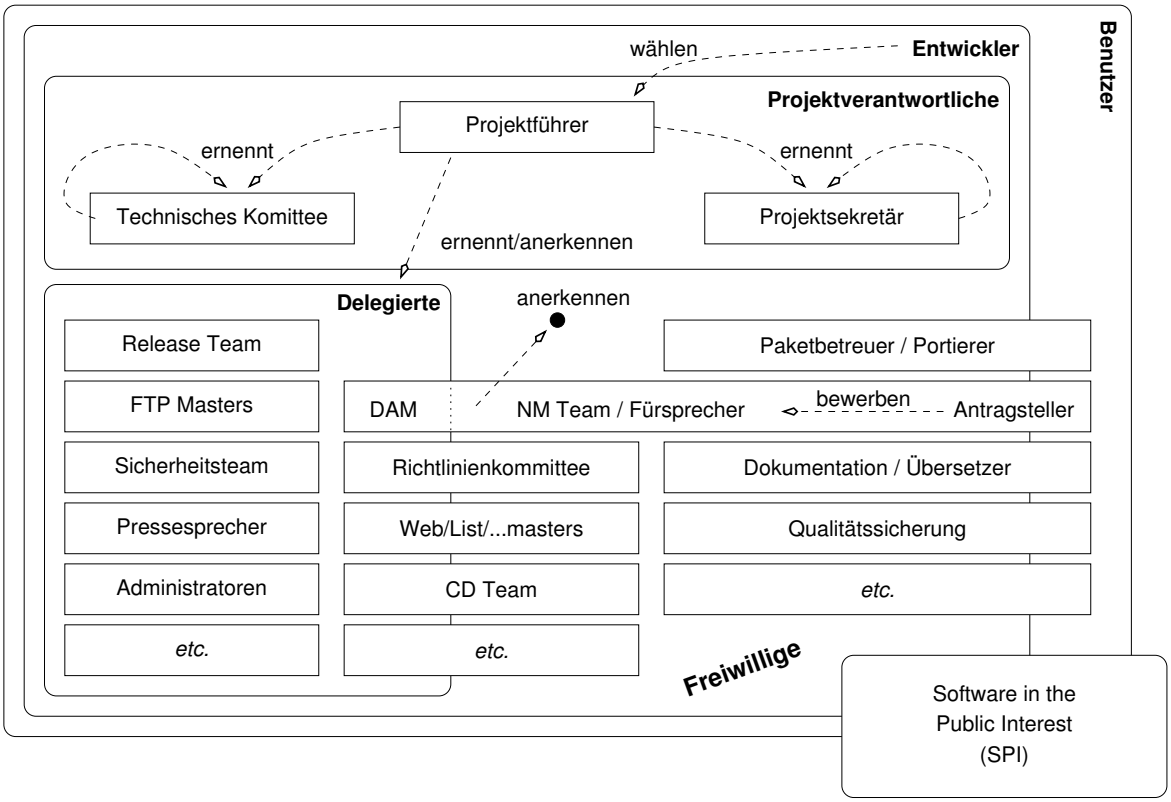


Bild 2.2: Organisationsstruktur des Debian-Projekts. (Grafik von Martin F. Krafft.)

einem Unterverzeichnis `main`, erstere in `non-free`. (Es gibt auch noch ein Mittelding namens `contrib`; dort findet sich Software, die für sich genommen DFSG-frei wäre, aber nicht ohne andere nichtfreie Komponenten funktioniert.)

- Der *Social Contract* (»Gesellschaftsvertrag«) beschreibt die Ziele des Projekts.
- Die *Debian Constitution* (»Verfassung«) beschreibt die Organisation des Projekts (siehe Bild 2.2).

Versionen

Zu jedem Zeitpunkt existieren mindestens drei Versionen von Debian GNU/Linux: In den `unstable`-Zweig werden neue oder korrigierte Versionen von Paketen eingebracht. Tauchen in einem Paket keine gravierenden Fehler auf, wandert es nach einer gewissen Wartezeit in den `testing`-Zweig. In gewissen Abständen wird der Inhalt von `testing` »eingefroren«, gründlich getestet und schließlich als `stable` freigegeben. Ein häufig geäußerter Kritikpunkt an Debian GNU/Linux sind die langen Zeiträume zwischen `stable`-Versionen; dies wird allerdings von vielen auch als Vorteil empfunden. Debian GNU/Linux wird vom Projekt ausschließlich zum Download zur Verfügung gestellt; Datenträger sind von Drittanbietern erhältlich.

Ableger-Projekte

Debian GNU/Linux ist durch seine Organisation, die weitgehende Abwesenheit kommerzieller Interessen und die saubere Trennung von freier und nichtfreier Software eine gute Grundlage für Ableger-Projekte. Einige populäre solche Projekte sind Knoppix (eine „Live-CD“, die es möglich macht, Linux auf einem PC zu testen, ohne es zuerst installieren zu müssen), SkoleLinux (heute »Debian/EDU«, ein speziell auf die Anforderungen von Schulen ausgerichtetes Linux) oder kommerzielle Distributionen wie Xandros. Auch Limux, das Münchner Desktop-Linux, basiert auf Debian GNU/Linux.

2.4.5 Ubuntu

Der wahrscheinlich populärste Debian-Ableger ist Ubuntu (`http://www.ubuntu.com/`), das von der britischen Firma Canonical Ltd. des südafrikanischen Unternehmers Mark Shuttleworth angeboten wird. (»Ubuntu« ist ein Wort aus der Zulu-Sprache und steht in etwa für »Menschlichkeit gegenüber anderen«.) Das Ziel von Ubuntu ist es, auf der Basis von Debian GNU/Linux ein aktuelles, leistungsfähiges und verständliches Linux anzubieten, das in regelmäßigen Abständen erscheint. Dies wird zum Beispiel dadurch erreicht, dass Ubuntu im Gegensatz zu Debian GNU/Linux nur drei statt über zehn Rechnerarchitekturen unterstützt und sich auf eine Teilmenge der in Debian GNU/Linux angebotenen Software beschränkt.

Ziel von Ubuntu

Ubuntu basiert auf dem `unstable`-Zweig von Debian GNU/Linux und verwendet in weiten Teilen dieselben Werkzeuge etwa zur Softwareverteilung, allerdings sind Debian- und Ubuntu-Softwarepakete nicht notwendigerweise miteinander kompatibel. Ubuntu erscheint in einem ziemlich regelmäßigen 6-Monats-Rhythmus, wobei alle zwei Jahre eine »LTS«-, also *long-term-support*-Version, herauskommt, für die Canonical Updates über einen Fünf-Jahres-Zeitraum verspricht.

Einige Ubuntu-Entwickler sind auch im Debian-Projekt aktiv, so dass es einen gewissen Austausch gibt. Andererseits sind nicht alle Debian-Entwickler begeistert von den Abkürzungen, die Ubuntu zuweilen im Namen des Pragmatismus nimmt, wo Debian vielleicht nach tragfähigeren, aber aufwendigeren Lösungen suchen würde. Ubuntu fühlt sich auch nicht im selben Maße der Idee der freien Software verpflichtet; während alle Infrastrukturwerkzeuge von Debian (etwa das Verwaltungssystem für Fehlerberichte) als freie Software zur Verfügung stehen, ist das für die von Ubuntu nicht immer der Fall.

Ubuntu vs. Debian

Ubuntu will nicht nur ein attraktives Desktop-System anbieten, sondern auch im Server-Bereich mit den etablierten Systemen wie RHEL oder SLES konkurrieren, also stabile Distributionen mit langem Lebenszyklus und guter Wartung anbieten. Es ist nicht klar, wie Canonical Ltd. auf lange Sicht Geld zu verdienen gedenkt; einstweilen wird das Projekt vor allem aus Mark Shuttleworths Schatulle unterstützt, die seit dem Verkauf seiner Internet-Zertifizierungsstelle Thawte an Verisign gut gefüllt ist …

Ubuntu vs. SUSE/Red Hat

2.4.6 Andere

Außer den genannten Distributionen gibt es noch viele weitere, etwa Mandriva Linux (`http://www.mandriva.com/en/linux/`) oder Turbolinux (`http://www.turbolinux.com/`) als kleinere Konkurrenten von Red Hat und SUSE, Gentoo Linux (`http://www.gentoo.org/`) als Distribution mit Fokus auf den Quellcode, zahlreiche »Live-Systeme« für verschiedene Zwecke von der Firewall bis hin zur Spiele- oder Multimedia-Plattform sowie sehr kompakte Systeme, die als Router, Firewall oder Rettungssystem einsetzbar sind.

Erwähnenswert vielleicht einfach aufgrund der Anzahl der »installierten Systeme« ist Android, das man mit etwas Wohlwollen durchaus als »Linux-Distribution« ansehen kann. Android besteht aus einem Linux-Betriebssystemkern mit einer Benutzerumgebung von Google auf der Basis von Googles Version von Java (»Dalvik«) anstatt der üblichen auf GNU, X, KDE usw. basierenden Benutzerumgebung, die die Grundlage der meisten »normalen« Distributionen bildet. Ein Android-Smartphone oder -Tablet präsentiert sich dem Benutzer also völlig anders als ein typischer Linux-PC unter openSUSE oder Debian GNU/Linux, ist aber dennoch im Grunde seines Wesens ein Linux-System.

Android

Während die meisten Android-Anwender ihr System vorinstalliert auf dem Telefon oder Tablet kaufen und dann nicht mehr ändern, ist es bei den meisten Android-basierten Geräten durchaus möglich (notfalls mit Tricks), eine alternative Android-»Distribution« zu installieren, von denen es inzwi-

schen auch etliche gibt. Für viele Geräte ist das die einzige Möglichkeit, aktualisierte Android-Versionen zu bekommen, wenn der Geräte-Hersteller oder der Telefondienst-Anbieter es nicht nötig finden, eine offizielle neue Version zu veröffentlichen.

2.4.7 Unterschiede und Gemeinsamkeiten

Gemeinsamkeiten

Obwohl es eine Unzahl an Distributionen gibt, verhalten zumindest die »großen« Distributionen sich in der täglichen Arbeit recht ähnlich. Das liegt zum einen daran, dass sie die gleichen grundlegenden Programme benutzen – beispielsweise ist der Kommandozeileninterpreter fast immer die `bash`. Zum anderen gibt es Standards, die dem Wildwuchs entgegenzuwirken versuchen. Zu nennen sind hier der *Filesystem Hierarchy Standard* (FHS) oder die *Linux Standard Base* (LSB), die versucht, eine einheitliche »Basisversion« von Linux zu definieren und die Versionsstände von Werkzeugen und Bibliotheken festzuschreiben, damit Drittanbieter es leichter finden, ihre Software für eine Vielzahl von Linux-Distributionen zu vertreiben.

Leider war die LSB nicht der durchschlagende Erfolg, mit dem man ursprünglich gerechnet hatte – sie wurde oft als Ansatz missverstanden, die Innovation in Linux zu verlangsamen oder zu stoppen und die Diversität zu verrringern (wobei es in den meisten Distributionen möglich ist, eine LSB-Umgebung parallel zu und unabhängig von der eigentlichen Umgebung für distributionseigene Software zur Verfügung zu stellen), während die Software-Drittanbieter, die eigentlich angesprochen werden sollten, es unter dem Strich vorzogen, ihre Pakete für die wichtigen »Enterprise-Distributionen« wie RHEL und SLES zu »zertifizieren« und nur diesen Plattformen Unterstützung zu gewähren – es ist also absolut nicht ausgeschlossen, dass SAP oder Oracle zum Beispiel auch auf Debian GNU/Linux laufen (oder zum Laufen zu bringen sind), aber die Kosten für große kommerzielle Softwarepakete wie diese sind so, dass die Lizenzgebühren für RHEL oder SLES keinen merkbaren Unterschied im Gesamtbild machen.

Paketformate

Ein merkbarer Punkt, wo die Distributionen sich unterscheiden, ist die Methode, mit der Softwarepakete verwaltet, also eingespielt und entfernt werden, und daraus resultierend das Dateiformat der fertigen Pakete in der Distribution. Hier gibt es zur Zeit zwei konkurrierende Ansätze, und zwar den von Debian GNU/Linux (»`deb`«) und den ursprünglich von Red Hat entwickelten (»`rpm`«). Wie üblich ist keiner der beiden dem anderen eindeutig überlegen, aber jeder von ihnen hat genug Alleinstellungsmerkmale, damit seine Anhänger nicht ohne weiteres bereit sind, umzusatteln. Der `deb`-Ansatz wird von Debian GNU/Linux, Ubuntu und anderen Debian-Ablegern verwendet, während Red Hat, SUSE und diverse andere von diesen abgeleitete Distributionen auf `rpm` setzen.

Abhängigkeiten

Beiden Ansätzen gemein ist eine umfassende Verwaltung von »Abhängigkeiten« zwischen Softwarepaketen, die dabei hilft, die Konsistenz des Systems zu sichern und zum Beispiel das Entfernen von Softwarepaketen verhindert, auf deren Anwesenheit sich andere Pakete im System verlassen (oder umgekehrt deren Installation erzwingt, wenn ein anderes Paket, das gerade installiert wird, sie benötigt und sie nicht schon vorhanden sind).

Während Windows- und OS-X-Anwender es gewöhnt sind, sich Software aus den verschiedensten Quellen zusammenzusuchen[6], versuchen zumindest die »großen« Distributionen wie Debian, openSUSE oder Ubuntu ihren Benutzern eine möglichst umfassende Softwareauswahl direkt über die Paketverwaltungswerkzeuge zur Verfügung zu stellen. Diese erlauben den Zugriff auf »Repositories«, also Paketlagerstätten, der Distributionen und

Repositories

[6] Wobei Apple und auch Microsoft derzeit vehement dabei sind, das von Smartphones bekannte Konzept des zentralen »App Stores« auch in ihren PC-Betriebssystemen zu etablieren.

gestatten zum Beispiel eine (mehr oder weniger komfortable) Suche nach bestimmten Softwarepaketen, die dann bequem – gegebenenfalls mit ihren Abhängigkeiten – über das Netz installiert werden können.

Es ist wichtig, anzumerken, dass es nicht notwendigerweise möglich ist, Pakete zwischen Distributionen auszutauschen, selbst wenn diese dasselbe Paketformat (`deb` oder `rpm`) verwenden – in den Paketen stecken diverse Annahmen darüber, wie eine Distribution aufgebaut ist, die weit über das Paketformat hinausgehen. Völlig ausgeschlossen ist es nicht (Debian GNU/Linux und Ubuntu sind einander zum Beispiel ähnlich genug, dass es unter Umständen durchaus klappen kann, ein für Debian gedachtes Paket auf einem Ubuntu-System zu installieren oder umgekehrt), aber als Faustregel gilt, dass die Gefahr eines Misserfolgs umso größer ist, je tiefer ein Paket im System verankert ist – ein Paket, das nur ein paar ausführbare Programme und ihre Dokumentation mitbringt, wirft weniger Probleme auf als ein Systemdienst, der in die Startsequenz integriert werden muss. Lassen Sie im Zweifelsfall die Finger von Experimenten mit ungewissem Ausgang.

Tabelle 2.1 zeigt eine Übersichtstabelle der wichtigsten Linux-Distributionen. Für mehr Informationen sollten Sie DistroWatch oder die Webseiten der einzelnen Distributionen heranziehen.

Tabelle 2.1: Vergleich der wichtigsten Linux-Distributionen (Stand: Februar 2012)

	RHEL	Fedora	SLES	openSUSE	Debian	Ubuntu
Hersteller	Red Hat	Red Hat + Comm.	SUSE	SUSE + Comm.	Debian-Projekt	Canonical + Comm.
Zielgruppe	Untern.	Geeks	Untern.	Privat	Unt/Priv	Unt/Priv
Kosten?	ja	nein	Support	nein	nein	nein
Erstveröfftl.	2003	2003	2000	2006	1993	2004
Rhythmus	3–4 J.	ca. 6 Mon.	3–4 J.	8 Mon.	ca. 2 J.	6 Mon.
Lebensdauer	10 J.	ca. 1 J.	7 J.	18 Mon.	3–4 J.	5 J. (LTS)
Plattformen	6	2	5	2	10	3
Pakete (ca.)	3000	26.000	?	14.650	29.050	37.000
Paketformat	`rpm`	`rpm`	`rpm`	`rpm`	`deb`	`deb`
Live-Medium?	?	ja	nein	ja	ja	ja

Zusammenfassung

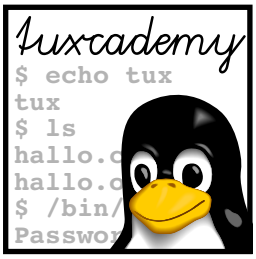
- Die erste Version von Linux wurde von Linus Torvalds entwickelt und als »freie Software« im Internet zur Verfügung gestellt. Heute wirken Hunderte von Entwicklern weltweit an der Aktualisierung und Erweiterung des Systems mit.
- Freie Software erlaubt Anwendern den Einsatz zu beliebigen Zwecken, die Einsichtnahme in und die Veränderung des Codes sowie die Weitergabe originalgetreuer oder modifizierter Versionen.
- Lizenzen für freie Software geben dem Empfänger der Software Rechte, die er sonst nicht hätte, während Lizenzverträge für proprietäre Software versuchen, den Empfänger zum Verzicht auf Rechte zu bewegen, die er sonst hätte.
- Die GPL ist eine der populärsten Lizenzen für freie Software.
- Andere gängige Lizenzen für freie Software sind die BSD-Lizenz, die Apache- oder die Mozilla Public License. Creative-Commons-Lizenzen sind gedacht für andere Kulturgüter außer Software.
- Es gibt eine große Auswahl freier und Open-Source-Programme für die unterschiedlichsten Anwendungen.
- Es gibt sehr viele verschiedene Linux-Distributionen. Zu den bekanntesten gehören Red Hat Enterprise Linux und Fedora, SUSE Linux Enterprise Server und openSUSE, Debian GNU/Linux und Ubuntu.

Literaturverzeichnis

GPL-Urteil06 Landgericht Frankfurt am Main. »Urteil 2-6 0 224/06«, Juli 2006. http://www.jbb.de/urteil_lg_frankfurt_gpl.pdf

GPL91 Free Software Foundation, Inc. »GNU General Public License, Version 2«, Juni 1991. http://www.gnu.org/licenses/gpl.html

TD01 Linus Torvalds, David Diamond (Hg.) *Just for Fun: Wie ein Freak die Computerwelt revolutionierte*. Hanser Fachbuch, 2001. ISBN 3-446-21684-7.



3

Erste Schritte mit Linux

Inhalt

Lernziele

- Einfache Funktionen von Linux ausprobieren
- Dateien mit einem Texteditor anlegen und ändern können

Vorkenntnisse

- Grundlegende Computerkenntnisse mit anderen Betriebssystemen sind nützlich

3.1 Anmelden und Abmelden

Zugangsberechtigung

Das Linux-System unterscheidet verschiedene Benutzer. Konsequenterweise können Sie deswegen nach Einschalten des Rechners mitunter nicht sofort loslegen. Zuerst müssen Sie dem Rechner mitteilen, wer Sie sind – Sie müssen sich anmelden (oder, neudeutsch, „einloggen“). Mit dieser Kenntnis kann das System entscheiden, was Sie dürfen (oder nicht dürfen). Natürlich müssen Sie eine Zugangsberechtigung zum System haben – der Systemverwalter muss Sie als Benutzer eingetragen und Ihnen einen Benutzernamen (zum Beispiel `hugo`) und ein Kennwort (zum Beispiel `geheim`) zugeordnet haben. Das Kennwort soll sicherstellen, dass nur Sie diese Zugangsberechtigung nutzen können; Sie müssen es geheimhalten und sollten es niemandem sonst zugänglich machen. Wer Ihren Benutzernamen und Ihr Kennwort kennt, kann sich gegenüber dem System als Sie ausgeben, alle Ihre Dateien lesen (oder löschen), in Ihrem Namen elektronische Post verschicken und alle möglichen anderen Arten von Schindluder treiben.

Modernere Linux-Distributionen möchten es Ihnen leicht machen und erlauben es, auf einem Rechner, den sowieso nur Sie benutzen, den Anmeldevorgang zu überspringen. Wenn Sie so ein System verwenden, dann müssen Sie sich nicht explizit anmelden, sondern der Rechner startet direkt in Ihre Benutzersitzung. Sie sollten das natürlich nur ausnutzen, wenn Sie nicht damit rechnen müssen, dass Dritte Zugang zu Ihrem Rechner haben; verkneifen Sie sich das vor allem auf Notebooks und anderen mobilen Systemen, die gerne mal verloren gehen oder gestohlen werden.

Anmelden in einer grafischen Umgebung Linux-Arbeitsplatzrechner bieten Ihnen heutzutage, wie sich das gehört, eine grafische Umgebung an, und auch der Anmeldevorgang spielt sich normalerweise auf dem Grafikbildschirm ab. Ihr Rechner präsentiert Ihnen einen Dialog, wo Sie Ihren Benutzernamen und Ihr Kennwort eingeben können.

Wundern Sie sich nicht, wenn Sie beim Eingeben Ihres Kennworts nur Sternchen oder Punkte sehen. Das bedeutet nicht, dass Ihr Rechner Ihre Eingabe missversteht, sondern dass er es Leuten schwerer machen möchte, die Ihnen beim Tippen über die Schulter schauen, um Ihr Kennwort zu erhaschen.

Nach dem Anmelden baut Ihr Rechner eine grafische Sitzung für Sie auf, in der Sie über Menüs und Icons (Sinnbilder auf dem Bildschirmhintergrund) bequemen Zugang zu Ihren Anwendungsprogrammen bekommen. Die meisten grafischen Umgebungen für Linux unterstützen eine „Sitzungsverwaltung“, die dafür sorgt, dass beim Anmelden so weit wie möglich der Zustand wieder hergestellt wird, den Ihre Sitzung hatte, als Sie sich zuletzt abgemeldet haben. Auf diese Weise müssen Sie sich nicht merken, welche Programme Sie laufen hatten, wo deren Fenster auf dem Bildschirm platziert waren, und welche Dateien Sie gerade bearbeitet haben.

Abmelden in einer grafischen Umgebung Wenn Sie mit der Arbeit fertig sind oder den Rechner für einen anderen Benutzer frei machen wollen, müssen Sie sich abmelden. Das ist auch wichtig, damit die Sitzungsverwaltung Ihre aktuelle Sitzung für das nächste Mal sichern kann. Wie das Abmelden im Detail funktioniert, hängt von Ihrer grafischen Umgebung ab, aber in der Regel gibt es irgendwo einen Menüeintrag, der alles für Sie erledigt. Konsultieren Sie im Zweifel die Dokumentation oder fragen Sie Ihren Systemadministrator (oder sich gut auskennenden Kumpel).

Anmelden auf der Textkonsole Im Gegensatz zu Arbeitsplatzrechnern haben Serversysteme oft nur eine Textkonsole oder stehen in zugigen, lauten Rechnerräumen, wo man sich nicht länger aufhalten möchte als nötig, so dass man sich

lieber über das Netz anmeldet, um auf dem Rechner zu arbeiten. In beiden Fällen bekommen Sie keinen grafischen Anmeldebildschirm, sondern der Rechner fragt Sie direkt nach Ihrem Benutzernamen und Kennwort. Zum Beispiel könnten Sie einfach etwas sehen wie

```
rechner login: _
```

(wenn wir mal annehmen, dass der betreffende Rechner „`rechner`“ heißt). Hier müssen Sie Ihren Benutzernamen eingeben und mit der Eingabetaste abschließen. Daraufhin erscheint in ähnlicher Form die Frage nach dem Kennwort:

```
Password: _
```

Hier müssen Sie Ihr Kennwort eingeben. (Hier erscheinen normalerweise nicht einmal Sternchen, sondern einfach gar nichts.) Wenn Sie sowohl Benutzername als auch Kennwort korrekt angegeben haben, sollte das System die Anmeldung akzeptieren. Der Kommandozeileninterpreter (die *Shell*) wird gestartet, und Sie können über die Tastatur Kommandos eingeben und Programme aufrufen. Nach der Anmeldung befinden Sie sich automatisch in Ihrem „Heimatverzeichnis“, wo Sie Ihre Dateien finden können.

Wenn Sie zum Beispiel die „Secure Shell“ verwenden, um sich auf einem anderen Rechner über das Netz anzumelden, entfällt in der Regel die Frage nach Ihrem Benutzernamen, da das System, wenn Sie nicht ausdrücklich etwas anderes angeben, davon ausgeht, dass Sie auf dem entfernten Rechner denselben Benutzernamen haben wie auf dem Rechner, von dem aus Sie die Sitzung aufbauen. Die Details würden hier aber zu weit führen; die Secure Shell wird in der Linup-Front-Schulungsunterlage *Linux-Administration II* ausführlich besprochen.

Abmelden auf der Textkonsole Auf der Textkonsole können Sie sich zum Beispiel mit dem Befehl `logout` abmelden:

```
$ logout
```

Auf einer Textkonsole zeigt das System nach dem Abmelden wieder die Startmeldung und eine Anmeldeaufforderung für den nächsten Benutzer. Bei einer Sitzung mit der „Secure Shell“ über das Netz bekommen Sie einfach wieder die Eingabeaufforderung Ihres lokalen Rechners.

Übungen

3.1 [!1] Versuchen Sie sich beim System anzumelden. Melden Sie sich anschließend wieder ab. (Einen Benutzernamen und ein Kennwort verrät Ihnen die Dokumentation Ihres Systems oder Ihr Trainer/Lehrer.)

3.2 [!2] Was passiert, wenn Sie (a) einen nicht existierenden Benutzernamen, (b) ein falsches Kennwort angeben? Fällt Ihnen etwas auf? Welchen Grund könnte es dafür geben, dass das System sich so verhält, wie es sich verhält?

3.2 Arbeitsumgebung und Browser

3.2.1 Grafische Arbeitsumgebungen

Wenn Sie sich in einer grafischen Umgebung angemeldet haben, präsentiert Ihr Linux-Rechner Ihnen eine Oberfläche, die nicht sehr von dem abweicht, was Sie bei anderen modernen Computern antreffen würden.

Leider ist es uns nicht möglich, hier spezifisch zu werden, weil »Linux« an dieser Stelle nicht gleich »Linux« ist. Im Gegensatz zu Systeme wie Windows oder dem Macintosh-Betriebssystem, wo eine bestimmte grafische Oberfläche fest zum System gehört, gibt es bei Linux hier Auswahl – bei der Installation des Systems können Sie sich bei den meisten großen Distributionen eine von mehreren Grafikumgebungen aussuchen:

- KDE und GNOME sind »Arbeitsumgebungen« (»Desktops«), die versuchen, eine möglichst umfassende Auswahl von Anwendungen zur Verfügung zu stellen, die ähnlich aussehen und sich ähnlich benehmen. Ziel von KDE und GNOME ist es, eine Benutzererfahrung zu bieten, die vergleichbar mit der proprietärer Systeme ist oder diese noch übertrifft. Deswegen versucht man sich an innovativen Eigenschaften wie der »semantischen Suche« von KDE, die Dateien und Dokumente im Hintergrund indiziert und zum Beispiel den bequemen Zugriff auf »alle Fotos, die ich letzten Monat in Spanien aufgenommen habe« zulassen soll, egal wo diese tatsächlich auf der Platte zu finden sind[1]. Grob gesagt setzt KDE auf umfassende Einstellungs- und Konfigurationsmöglichkeiten für engagierte Benutzer, während GNOME tendenziell dazu neigt, seinen Benutzern im Interesse der Übersichtlichkeit und Bedienbarkeit feste Vorgaben zu machen, die nicht oder nur aufwendig zu ändern sind.
- LXDE und XFCE sind »leichtgewichtige« Arbeitsumgebungen. Sie ähneln in ihrem grundsätzlichen Ansatz KDE und GNOME, aber sind eher auf sparsamen Umgang mit den Systemressourcen getrimmt und verzichten daher auf verschiedene aufwendige Dienste wie etwa die semantische Suche.
- Wenn Sie keine komplette Arbeitsumgebung möchten, können Sie auch einen beliebigen »Fenstermanager« (*window manager*) installieren und Anwendungsprogramme aus verschiedenen Quellen benutzen. Dann sind natürlich gewisse Abstriche bei der Einheitlichkeit und dem Zusammenwirken der Programme nötig, die daraus resultieren, dass es historisch kaum Vorgaben für das Aussehen und Benehmen von grafischen Programmen unter Unix und Linux gab. Früher – vor KDE & Co., die in diesem Bereich Standards setzten – war das der Normalfall, aber heute baut die Mehrheit der Linux-Anwender auf eine der vorgefertigten Umgebungen.

Selbst wenn zwei Distributionen dieselbe Grafikumgebung – etwa KDE – verwenden, ist noch nicht gesagt, dass die Erscheinung des Bildschirms bei beiden identisch ist. Die Grafikumgebungen erlauben normalerweise eine sehr umfangreiche Anpassung des optischen Erscheinungsbilds über »Themen«, und Distributionen nutzen das gerne aus, um sich einen individuellen Anstrich zu geben. Denken Sie an Autos; so gut wie alle Personenwagen haben vier Räder und eine Windschutzscheibe, aber Sie würden trotzdem nie einen BMW mit einem Citroën oder Ferrari verwechseln.

Steuerleiste

Typischerweise finden Sie am oberen oder unteren Bildschirmrand eine Leiste, die Ihnen über Menüeinträge Zugriff auf die wichtigsten Anwendungsprogramme gibt oder es Ihnen erlaubt, sich abzumelden oder den Rechner kontrolliert auszuschalten. KDE setzt hier auf ein an Windows angelehntes »Panel«, bei dem ein »Startknopf« (der allerdings nicht so heißt) ein Menü mit Programmen aufklappt, während im Rest der Leiste Icons für die aktuell laufenden Programme zu sehen sind, neben Nützlichkeiten wie einer Uhr, dem Netzwerkstatus und so weiter. GNOME verwendet keinen »Startknopf«, sondern verlegt die Menüleiste an den oberen Bildschirmrand; die wichtigsten Programme sind über Aufklappmenüs im linken Teil zugänglich, während der rechte Teil der Leiste für Systemstatus-Icons und Ähnliches reserviert ist.

[1] Microsoft hat uns das schon ein paarmal versprochen, aber immer wieder abgekündigt.

Die Grafikumgebungen bieten in der Regel auch einen »Dateimanager«, mit dem Sie auf Verzeichnisse (»Ordner«) auf der Platte zugreifen und die darin enthaltenen Dateien und Unterverzeichnisse manipulieren können. Die Vorgehensweisen dabei unterscheiden sich ebenfalls kaum von denen auf anderen grafischen Systemen: Sie können Dateien kopieren oder verschieben, indem Sie sie von einem Verzeichnis-Fenster in ein anderes ziehen, und wenn Sie mit der rechten Maustaste auf das Icon einer Datei klicken, öffnet sich ein »Kontextmenü« mit weiteren Aktionen, die Sie auf die Datei anwenden können. Experimentieren Sie.

Dateimanager

Häufig gebrauchte Dateien oder Programme können Sie meist entweder auf dem Bildschirmhintergrund ablegen oder in einen bestimmten Bereich (ein »Dock«) kopieren, damit sie schnell zugänglich sind.

Dock

Eine nützliche Eigenschaft der meisten Grafikumgebungen unter Linux, die bei OS X und Windows nicht oder zumindest nicht standardmäßig vorhanden ist, sind »virtuelle Arbeitsflächen« (*virtual desktops*). Diese vervielfachen Ihren verfügbaren Platz auf dem Bildschirm, indem Sie bequem zwischen mehreren simulierten »Bildschirmen« hin und her schalten können – jeder mit seiner eigenen Auswahl an Programmfenstern. Damit können Sie zum Beispiel alles für die Arbeit an einem Programm oder Dokument auf einer Arbeitsfläche platzieren, eine andere für Ihr E-Mail-Programm und eine dritte für Ihren Web-Browser reservieren und bei Bedarf zum Beispiel schnell eine Mail lesen oder schreiben, ohne dass Sie die Fensteranordnung in Ihrem »Programmierbildschirm« ändern müssen.

virtuelle Arbeitsflächen

3.2.2 Browser

Eines der wichtigsten Programme auf heutigen Computern ist der Web-Browser. Glücklicherweise sind die beliebtesten Browser Open-Source-Programme, und Firefox oder Google Chrome stehen Ihnen für Linux genauso zur Verfügung wie für Windows oder OS X. (Ihre Distribution bietet Ihnen wahrscheinlich nicht Google Chrome an, sondern die echte Open-Source-Variante »Chromium«, aber das ist kein großer Unterschied.) Suchen Sie im Menü für Anwendungsprogramme nach einem Eintrag wie »Internet«, wo Sie (unter anderem) einen Browser finden sollten.

Aus markenrechtlichen Gründen heißt Firefox bei Debian GNU/Linux und diversen abgeleiteten Distributionen nicht »Firefox«, sondern »Iceweasel« (jaja, geniales Wortspiel …). Das liegt daran, dass die Mozilla Foundation, der »Hersteller« von Firefox, die Verbreitung von vorübersetzten Versionen des Browsers unter dem Namen »Firefox« nur erlaubt, wenn der Programmcode mit der »offiziellen« Version übereinstimmt. Da das Debian-Projekt sich einerseits vorbehält, zum Beispiel sicherheitsrelevante Probleme in eigener Regie zu reparieren, und andererseits das Urheber- und Markenrecht sehr ernst nimmt, war ein anderer Name nötig. (Andere Distributionen bleiben entweder bei der offiziellen Version oder nehmen es mit dem Namen nicht so genau.)

3.2.3 Terminals und Shells

Selbst in einer grafischen Oberfläche ist es unter Linux oft nützlich, auf ein »Terminalfenster« zurückgreifen zu können, wo Sie in einer »Shell« textuelle Kommandos eingeben können (der Rest dieser Unterlage redet vor allem über Kommandos in der Shell, Sie werden das also brauchen).

Zum Glück ist ein Terminalfenster bei den gängigen Linux-Arbeitsumgebungen nur ein paar Mausklicks entfernt. Bei KDE auf Debian GNU/Linux finden Sie zum Beispiel im »Startmenü« unter »System« den Eintrag »Konsole (Terminal)«, über den Sie ein komfortables Programm öffnen können, in dem eine Shell läuft, die Textkommandos annimmt und ausführt. Für andere Arbeitsumgebungen und Distributionen gilt Ähnliches.

Übungen

3.3 [!2] Welche Grafikumgebung ist (wenn überhaupt) auf Ihrem Rechner installiert? Schauen Sie sich um. Öffnen Sie einen Dateimanager und prüfen Sie, was passiert, wenn Sie mit der rechten Maustaste auf ein Icon für eine Datei oder ein Verzeichnis klicken. Was passiert, wenn Sie mit der rechten Maustaste auf den freien Fensterhintergrund (zwischen den Icons) klicken? Wie können Sie eine Datei von einem Verzeichnis in ein anderes verschieben? Wie können Sie eine neue Datei oder ein neues Verzeichnis anlegen? Wie können Sie eine Datei umbenennen?

3.4 [2] Welcher Web-Browser ist auf Ihrem Rechner installiert? Gibt es vielleicht sogar mehrere? Rufen Sie den oder die Browser testhalber auf und vergewissern Sie sich, dass sie funktionieren.

3.5 [!2] Öffnen Sie ein Terminalfenster und schließen Sie es wieder. Unterstützt Ihr Terminalfenster-Programm mehrere Sitzungen im selben Fenster (etwa über »Unterfenster« mit Karteireitern)?

3.3 Textdateien anlegen und ändern

Egal ob Sie Skripte oder Programme schreiben, als Systemadministrator Konfigurationsdateien bearbeiten oder einfach nur einen Einkaufszettel verfassen wollen: Linux läuft zu großer Form auf, wenn es um das Bearbeiten von Textdateien geht. Eine Ihrer ersten »Amtshandlungen« als neuer Linux-Anwender sollte also sein, zu lernen, wie Sie Textdateien anlegen und ändern können. Das Werkzeug der Wahl hierfür ist ein Texteditor.

Texteditor

Texteditoren für Linux gibt es in allen Größen, Formen und Farben. Wir machen es uns an dieser Stelle einfach und erklären Ihnen die wichtigsten Eigenschaften von »GNU Nano«, einem simplen, einsteigergeeigneten Texteditor, der in einer Terminalsitzung läuft.

Natürlich gibt es in den gängigen Grafikumgebungen auch grafikorientierte Texteditoren mit Menüs, Werkzeugleisten und allem, was man so brauchen könnte – vergleichbar mit Programmen wie »Notepad« unter Windows (oder besser). Suchen Sie zum Beispiel bei KDE nach »Kate« oder bei GNOME nach »gedit«. Wir betrachten diese Texteditoren hier nicht im Detail, aus zwei Gründen:

- Sie erklären sich im Wesentlichen selber, und wir wollen Ihre Intelligenz nicht mehr als nötig beleidigen.
- Nicht immer sind Sie in einer Situation, wo Ihnen eine Grafikumgebung zur Verfügung steht. Spätestens wenn Sie mit der »Secure Shell« auf einem entfernten Rechner arbeiten oder als Systemadministrator irgendwo im Keller an der Maschinenkonsole stehen (!), stehen die Chancen gut, dass Sie nur eine Textoberfläche zur Verfügung haben.

Im Übrigen müssen Sie sich ja nicht jetzt im Moment ein für allemal für *einen* Editor entscheiden. Niemand hält Sie davon ab, an Ihrem grafischen Arbeitsplatz-PC einen grafischen Editor zu verwenden und etwas wie Nano nur zu benutzen, wenn Sie nicht anders können.

Linux-Fanatiker der alten Schule werden über etwas wie Nano verächtlich die Nase rümpfen: Das Werkzeug für den wahren Linux-Profi ist natürlich der `vi` (ausgesprochen »wie Ei«), der als Unix-Urgestein aus einer Zeit übriggeblieben ist, als grün schimmernde Textterminals das Höchste der Gefühle waren und man sich nicht mal darauf verlassen konnte, dass es Pfeiltasten auf der Tastatur gab (!). Wenn Sie eine Karriere als Systemadministrator

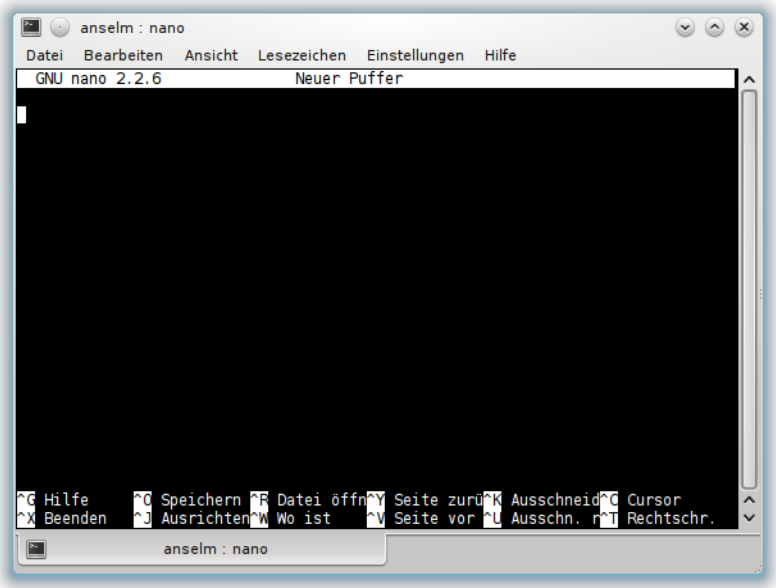


Bild 3.1: Der Editor GNU Nano

anstreben, dann sollten Sie sich früher oder später mit dem `vi` befassen (zumindest auf einer elementaren Ebene), da der `vi` der einzige benutzenswerte Editor ist, der auf praktisch jeder Linux- oder Unix-Version in weitgehend identischer Form vorliegt. Aber dieser Zeitpunkt ist nicht jetzt.

GNU Nano ist ein »Klon« eines simplen Editors namens `pico`, der Teil des E-Mail-Pakets PINE war. (PINE war keine freie Software im Sinne der gängigen Definitionen, so dass das GNU-Projekt den Editor von Grund auf nachprogrammiert hat. Inzwischen steht der Nachfolger von PINE unter dem Namen »alpine« als freie Software zur Verfügung und enthält auch eine freie Version von `pico`.) Die meisten Distributionen sollten entweder GNU Nano oder `pico` anbieten; wir beschränken uns im Rest dieses Abschnitts der Einfachheit halber auf GNU Nano. Praktisch alles, was wir hier sagen, gilt auch für `pico`. `pico`

GNU Nano hat gegenüber dem ursprünglichen `pico` einige Erweiterungen (was nicht überraschend sein sollte – schon dem Namen nach ist er ja um drei Größenordnungen besser), aber die meisten davon betreffen uns nicht wirklich. Eigentlich ist auf den ersten Blick nur eine augenfällig: GNU Nano ist »internationalisiert«, sollte Sie auf einem ansonsten auf »deutsch« eingestellten System also mit deutschen Meldungen und Erklärungen beglücken.

GNU Nano starten Sie am bequemsten in einem Terminalfenster (Abschnitt 3.2.3)GNU Nano starten mit einem Kommando wie

```
$ nano meinedatei
```

(das »$ « ist hier nur eine stilisierte Abkürzung für die Eingabeaufforderung – die bei Ihnen vielleicht etwas barocker aussieht –, und Sie müssen das nicht eintippen. Vergessen Sie aber nicht, das Kommando mit ↩ zu beenden!) Anschließend sollten Sie etwas sehen, was Bild 3.1 ähnelt – also ein weitestgehend leeres Fenster mit einer hervorgehobenen Zeile oben und zwei »Hilfezeilen« unten, die wichtige Kommandos mit kurzen Erklärungen auflisten. Die Zeile direkt über den Hilfezeilen ist die »Statuszeile«, wo Meldungen von Nano erscheinen und wo Sie zum Beispiel Dateinamen zum Speichern eingeben können.

Wenn Sie mehr nutzbaren Platz auf dem Bildschirm möchten, können Sie die Hilfezeilen mit Alt+x ausblenden (halten Sie die Alt-Taste – auf der Tastatur links von der Leertaste – fest, während Sie x drücken). Ein weiteres Alt+x blendet sie wieder ein. (Wenn Sie noch mehr nutzbaren Platz möchten, können Sie mit Alt+o auch noch die Leerzeile direkt unterhalb der obersten Bildschirmzeile unterdrücken.)

Text eingeben und ändern Um neuen Text einzugeben, können Sie im Nano-Fenster einfach lostippen. Sollten Sie sich mal verschrieben haben, löscht die »Backspace«-Taste ⇐ das Zeichen links von der Schreibmarke (neudeutsch »Cursor«). Verwenden Sie die Pfeiltasten, um im Text herumzufahren und zum Beispiel weiter vorne etwas zu ändern. Wenn Sie etwas Neues tippen, wird es genau da eingesetzt, wo der Cursor steht. Die Entf-Taste löscht das Zeichen unter dem Cursor und läßt den Rest der Zeile (falls da noch was kommt) um eine Position nach links rücken. Alles eigentlich ziemlich offensichtlich.

Nano und die Maus

Manche Nano-Versionen unterstützen sogar die Maus, so dass Sie – vorausgesetzt, Sie haben Nano auf einem grafischen Bildschirm laufen oder Ihre Textumgebung kann mit einer Maus umgehen – auf eine Stelle im Text klicken können, um den Cursor dort zu platzieren. Es kann sein, dass Sie die Maus-Unterstützung separat aktivieren müssen, indem Sie Alt+m drücken.

Text speichern Wenn Sie mit dem Eingeben oder Bearbeiten Ihres Textes fertig sind, können Sie ihn speichern, indem Sie Strg+o drücken (halten Sie Strg fest und drücken Sie auf o). Nano fragt Sie in der Statuszeile nach einem Namen für die Datei, den Sie eingeben und mit ↩ beenden können. (Über Dateinamen erfahren Sie spätestens in Kapitel 6 mehr.) Nano speichert den Text dann in der benannten Datei.

Nano beenden Nano verlassen können Sie mit Strg+x. Wenn Ihr Text ungespeicherte Veränderungen enthält, fragt Nano Sie noch, ob der Text gespeichert werden soll; antworten Sie mit j, wenn Sie das möchten (Nano wird Sie gegebenenfalls nach einem Dateinamen zum Speichern fragen), oder mit n, um Nano sofort zu verlassen (wobei Ihre ungespeicherten Änderungen dann flöten gehen).

Dateien einlesen Eine andere (schon existierende) Datei können Sie mit Strg+r in Ihren aktuellen Text einlesen – sie wird dann an der Cursorposition eingefügt. Nano fragt Sie nach dem Namen der gewünschten Datei, den Sie entweder direkt eintippen können, oder Sie klappen mit Strg+t den »Dateibrowser« auf, der Ihnen eine interaktive Auswahl unter den vorhandenen Dateien erlaubt. (Das funktioniert übrigens auch beim Speichern mit Strg+o.)

Ausschneiden und Einfügen Mit dem Kommando Strg+k können Sie die Zeile, in der der Cursor steht, ausschneiden und in einem Puffer ablegen (*Achtung:* Es wird immer die komplette Zeile ausgeschnitten, egal wo in der Zeile der Cursor tatsächlich steht!). Mit Strg+u können Sie den Inhalt des Puffers dann wieder einfügen – entweder an derselben Stelle, wenn Sie versehentlich auf Strg+k gekommen sind oder die Zeile nur kopieren und nicht verschieben wollen, oder auch anderswo im Text.

Eingefügt wird immer an der Cursorposition; wenn der Cursor also mitten in einer Zeile steht, wenn Sie auf Strg+u drücken, dann wird die Zeile aus dem Puffer zum rechten Teil jener Zeile, und was in der ursprünglichen Zeile rechts vom Cursor stand, wird zu einer neuen Zeile.

Sie können auch mehrere aufeinanderfolgende Zeilen in den Puffer tun, indem Sie mehrmals hintereinander auf Strg+k drücken. Diese Zeilen werden dann auch *en bloc* wieder eingefügt.

Wenn Sie nur einen Teil einer Zeile ausschneiden möchten, positionieren Sie den Cursor an die entsprechende Stelle und drücken Alt+a (eigentlich ist hierfür die Kombination Strg+^ vorgesehen, aber da sträuben sich deutsche Tastaturen, die nach einem »^« ein Zeichen erwarten, das mit einem Zirkonflex versehen werden soll). Anschließend können Sie mit dem Cursor ans Ende des auszuschneidenden Bereichs navigieren – Nano zeigt Ihnen hilfreicherweise an, welchen Teil des Texts Sie markiert haben – und den Bereich mit Strg+k ausschneiden und im Puffer ablegen. Beachten Sie dabei, dass das Zeichen unter dem Cursor dann gerade *nicht mehr* ausgeschnitten wird! Danach können Sie den Pufferinhalt wie gehabt mit Strg+u wieder irgendwo einfügen.

Text suchen Wenn Sie Strg+w eingeben, fragt Nano Sie in der Statuszeile nach einem Textstück. Anschließend springt der Cursor an das von seiner aktuellen Position aus gesehen nächste Auftreten dieses Textstücks in Ihrem Text. Das macht es bequem, gezielt Stellen im Text anzusteuern.

Onlinehilfe Mit Strg+g können Sie die Nano-interne Hilfe aufrufen, die Ihnen die Grundlagen des Editors und diverse Tastenbefehle erklärt (es gibt einige mehr, als wir hier erläutert haben). Mit Strg+x verlassen Sie die Hilfe wieder.

Damit kennen Sie die wichtigsten Eigenschaften von GNU Nano. Übung macht den Meister – probieren Sie also ungeniert herum, Sie können nichts kaputt machen.

Nochmal zurück zum Thema `vi` (Sie erinnern sich – der Editor der Linux-Gurus). Wenn Sie Lust auf ein Abenteuer haben, dann stellen Sie sicher, dass auf Ihrem System der Editor `vim` installiert ist (heutzutage die angesagte `vi`-Implementierung; den originalen `vi` aus BSD benutzt unter Linux praktisch niemand), rufen Sie das Kommando `vimtutor` auf und verbringen Sie eine spannende und instruktive halbe Stunde mit der interaktiven `vi`-Einführung. (Es kann sein, dass Sie bei Ihrer Distribution `vimtutor` als separates Paket installieren müssen. Fragen Sie notfalls Ihren Systemadministrator oder sonst jemanden, der sich auskennt.)

Übungen

3.6 [!2] Starten Sie GNU Nano und geben Sie einen einfachen Text ein – etwas wie

```
Rosen sind rot,
Veilchen sind blau,
Linux ist super,
das weiß ich genau.
```

Speichern Sie diesen Text in einer Datei unter dem Namen `rosen.txt`.

3.7 [2] Schneiden Sie im Text aus der vorigen Aufgabe die Zeile

```
Linux ist super,
```

aus und kopieren Sie sie dreimal zurück, so dass der Text jetzt aussieht wie

```
Rosen sind rot,
Veilchen sind blau,
Linux ist super,
```

```
Linux ist super,
Linux ist super,
das weiß ich genau.
```

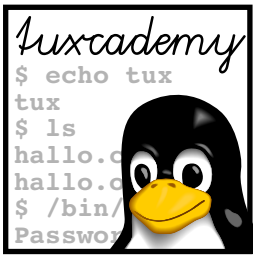
Positionieren Sie anschließend den Cursor auf dem »i« von »ist« in der ersten der Zeilen, markieren Sie diese Position, navigieren Sie zum »i« von »ist« in der dritten Zeile und entfernen Sie den markierten Bereich.

Kommandos in diesem Kapitel

logout	Beendet eine Sitzung („Abmelden“)	bash(1)	53
pico	Sehr einfacher Texteditor aus dem PINE/Alpine-Paket	pico(1)	57
vi	Bildschirmorientierter Texteditor	vi(1)	56

Zusammenfassung

- Vor dem Benutzen eines Linux-Rechners müssen Sie sich (meistens) mit Benutzername und Kennwort anmelden und hinterher wieder abmelden.
- Linux bietet verschiedene Grafikumgebungen an, die zum größten Teil ähnlich und ziemlich intuitiv funktionieren.
- In einem Terminalfenster können Sie in einer Grafikumgebung textuelle Shell-Kommandos eingeben.
- GNU Nano ist ein einfacher Texteditor.



4

Keine Angst vor der Shell

Inhalt

Lernziele

- Die Vorteile einer textorientierten Kommandooberfläche bewerten können
- Mit der Bourne-Again-Shell (Bash) arbeiten
- Struktur von Linux-Kommandos verstehen

Vorkenntnisse

- Grundlegende Kenntnisse im Umgang mit Computern sind hilfreich

4.1 Warum?

Mehr als andere moderne Betriebssysteme basiert Linux (wie Unix) auf der Idee, textuelle Kommandos über die Tastatur einzugeben. Manch einem mag das vorsintflutlich vorkommen, vor allem wenn man Systeme wie Windows gewöhnt ist, die dem Publikum seit 15 Jahren oder so einzureden versuchen, dass grafische Benutzungsoberflächen das A und O darstellen. Für viele, die von Windows zu Linux kommen, ist der Fokus auf die Kommandooberfläche zunächst ein »Kulturschock« vergleichbar dem, den ein Mensch des 21. Jahrhunderts erlitte, wenn er plötzlich an den Hof von Kaiser Barbarossa versetzt würde: Kein Internet, schlechte Tischmanieren und furchtbare Zahnärzte!

Allerdings muss man das heute nicht mehr so krass sehen. Zum einen gibt es für Linux grafische Oberflächen, die dem, was Windows oder MacOS X dem Benutzer bieten, in wenig bis nichts nachstehen oder sie in manchen Punkten an Bequemlichkeit und Leistung sogar noch übertreffen. Zum anderen schließen die grafischen Oberflächen und die textorientierte Kommandooberfläche sich ja nicht aus, sondern ergänzen sich hervorragend (gemäß der Philosophie »Das richtige Werkzeug für jede Aufgabe«).

Unter dem Strich bedeutet das nur, dass Sie als angehender Linux-Anwender gut daran tun, sich *auch* mit der textorientierten Kommandooberfläche, bekannt als »Shell«, vertraut zu machen. Natürlich will Sie niemand davon abhalten, eine grafische Oberfläche für alles zu benutzen, was Ihnen einfällt. Die Shell ist aber eine Abkürzung zu zahlreichen extrem mächtigen Operationen, die sich grafisch einfach nicht gut ausdrücken lassen. Die Shell abzulehnen ist äquivalent dazu, sich in der Fahrschule gegen die Verwendung aller Gänge außer des ersten zu sträuben: Sicher, auch im ersten Gang kommt man letztendlich ans Ziel, aber nur vergleichsweise langsam und mit heulendem Motor. Warum also nicht lernen, wie Sie mit Linux so richtig auf die Tube drücken können? Und wenn Sie gut aufpassen, haben wir noch den einen oder anderen Trick für Sie auf Lager.

4.2 Was ist die Shell?

Für den Anwender ist eine direkte Kommunikation mit dem Linux-Betriebssystemkern nicht möglich. Das geht nur über Programme, die ihn über »Systemaufrufe« ansprechen. Irgendwie müssen Sie aber solche Programme starten können. Diese Aufgabe übernimmt die Shell, ein besonderes Anwendungsprogramm, das (meistens) Kommandos von der Tastatur liest und diese – zum Beispiel – als Programmaufrufe interpretiert und ausführt. Die Shell stellt damit eine Art »Oberfläche« für den Computer dar, die das eigentliche Betriebssystem wie eine Nußschale (engl. *shell* – daher der Name) umschließt, das dadurch nicht direkt sichtbar ist. Natürlich ist die Shell nur ein Programm unter vielen, die auf das Betriebssystem zugreifen.

Auch die grafischen Oberflächen heutiger Systeme wie KDE kann man als »Shells« ansehen. Anstatt dass Sie textuelle Kommandos über die Tastatur eingeben, geben Sie eben grafische Kommandos mit der Maus ein – aber so wie die Textkommandos einer bestimmten »Grammatik« folgen, tun die Mauskommandos das auch. Zum Beispiel wählen Sie Objekte durch Anklicken aus und legen dann fest, was mit ihnen gemacht werden soll: Öffnen, Kopieren, Löschen, …

Schon das allererste Unix – Ende der 1960er Jahre – hatte eine Shell. Die älteste Shell, die heute noch außerhalb von Museen zu finden ist, wurde Mitte der 70er Jahre für »Unix Version 7« von Stephen L. Bourne entwickelt. Diese nach ihm benannte Bourne-Shell enthält alle grundlegenden Funktionen und erfreute sich einer weiten Verbreitung, ist heute aber nur mehr sehr selten in ihrer ursprünglichen Form anzutreffen. Daneben zählen die C-Shell, an der Berkeley-Universi-

Bourne-Shell

C-Shell

tät für BSD entwickelt und (extrem vage) an die Programmiersprache C angelehnt, sowie die zur Bourne-Shell weitgehend kompatible, aber mit einem größeren Funktionsumfang ausgestattete Korn-Shell (von David Korn, ebenfalls bei AT&T) zu den klassischen Unix-Shells. Korn-Shell

Standard für Linux-Systeme ist die Bourne-Again-Shell, kurz `bash`. Diese wurde im Rahmen des GNU-Projektes der *Free Software Foundation* von Brian Fox und Chet Ramey entwickelt und vereinigt Funktionen der Korn- und der C-Shell. Bourne-Again-Shell

Über die genannten Shells hinaus gibt es noch zahlreiche andere. Eine Shell unter Unix ist ein Anwendungsprogramm wie jedes andere auch, und Sie brauchen keine besonderen Privilegien, um eine schreiben zu können – Sie müssen sich nur an die »Spielregeln« halten, die bestimmen, wie eine Shell mit anderen Programmen kommuniziert. Shells: normale Programme

Shells können interaktiv aufgerufen werden und akzeptieren dann Kommandos vom Benutzer (typischerweise auf einem irgendwie gearteten »Terminal«). Die allermeisten Shells können ihre Kommandos aber auch aus Dateien lesen, die vorgekochte Befehlsfolgen enthalten. Solche Dateien heißen Shellskripte. Shellskripte

Die Shell übernimmt dabei im Einzelnen folgende Aufgaben:

1. Ein Kommando von der Tastatur (oder aus einer Datei) einlesen.
2. Das eingegebene Kommando überprüfen.
3. Das Kommando direkt ausführen oder das korrespondierende Programm starten.
4. Das Resultat auf dem Bildschirm (oder anderswohin) ausgeben.
5. Weiter bei 1.

Neben dieser Standardfunktion umfasst eine Shell meist noch weitere Elemente wie z. B. eine Programmiersprache. Mit Schleifen, Bedingungen und Variablen sind damit komplexe Befehlsstrukturen realisierbar (normalerweise in Shellskripten, seltener im interaktiven Gebrauch). Weiterhin kann die Shell durch eine ausgefeilte Verwaltung früherer Kommandos dem Anwender das Leben erleichtern. Programmiersprache

Eine Shell können Sie mit dem Kommando `exit` wieder verlassen. Das gilt auch für die Shell, die Sie gleich nach dem Anmelden bekommen haben. Shell verlassen

Obwohl es, wie erwähnt, diverse Shells gibt, konzentrieren wir uns hier auf die Bash als Standard-Shell bei den meisten Linux-Distributionen. Auch die Prüfungen des LPI beziehen sich ausschließlich auf die Bash.

Übungen

4.1 [2] Melden Sie sich ab und wieder an und prüfen Sie die Ausgabe des Kommandos »`echo $0`« in der »Loginshell«. Starten Sie mit dem Kommando »`bash`« eine neue Shell und geben Sie wiederum das Kommando »`echo $0`«. Vergleichen Sie die Ausgabe der beiden Kommandos. Fällt Ihnen etwas auf?

4.3 Kommandos

4.3.1 Wozu Kommandos?

Die Arbeitsweise eines Rechners, ganz egal, welches Betriebssystem sich darauf befindet, lässt sich allgemein in drei Schritten beschreiben:

1. Der Rechner wartet auf eine Eingabe durch den Benutzer.
2. Der Benutzer legt ein Kommando fest und gibt dieses per Tastatur oder per Maus ein.

3. Der Rechner führt die erhaltene Anweisung aus.

In einem Linux-System zeigt die Shell eine Eingabeaufforderung (engl. *prompt*) auf dem Textbildschirm oder in einem grafischen Terminalprogramm wie `xterm` oder `konsole`. an. Das bedeutet, dass sie dazu bereit ist, einen Befehl entgegenzunehmen. Diese Eingabeaufforderung setzt sich oft aus Benutzer- und Rechnernamen, dem aktuellen Verzeichnis und einem abschließenden Zeichen zusammen.

```
hugo@red:/home > _
```

In diesem Beispiel befindet sich also der Benutzer `hugo` auf dem Rechner `red` im Verzeichnis `/home`. (Aus Platzgründen und um den Text nicht zu sehr auf eine spezielle Linux-Distribution auszurichten, verwenden wir in diesen Unterlagen die neutrale, traditionelle Eingabeaufforderung »`$` «.)

4.3.2 Wie sind Kommandos aufgebaut?

Ein Kommando in der Shell ist grundsätzlich eine Folge von Zeichen, die durch die Eingabetaste (»Return«) abgeschlossen und danach von der Shell ausgewertet wird. Diese Kommandos sind zumeist vage der englischen Sprache entlehnt und ergeben in ihrer Gesamtheit eine eigene »Kommandosprache«. Kommandos in dieser Sprache müssen gewissen Regeln, einer Syntax, gehorchen, damit sie von der Shell verstanden werden können.

Syntax

Um eine Eingabezeile zu interpretieren, versucht die Shell zunächst, die Eingabezeile in einzelne Wörter aufzulösen. Als Trennelement dient dabei, genau wie im richtigen Leben, das Leerzeichen. Bei dem ersten Wort in der Zeile handelt es sich normalerweise um das eigentliche Kommando. Alle weiteren Wörter in der Kommandozeile sind Parameter, die das Kommando genauer spezifizieren.

Wörter

Erstes Wort: Kommando

Parameter

Für DOS- und Windows-Anwender ergibt sich hier ein möglicher Stolperstein, da die Shell zwischen Groß- und Kleinschreibung unterscheidet. Linux-Kommandos werden in der Regel komplett klein geschrieben (Ausnahmen bestätigen die Regel) und nicht anders verstanden. Siehe hierzu auch Abschnitt 4.3.4.

Groß- und Kleinschreibung

Beim Trennen von Wörtern in einem Kommando ist ein einzelnes Leerzeichen so gut wie viele – die Shell macht da keinen Unterschied. Genaugenommen besteht die Shell nicht mal auf Leerzeichen; Tabulatorzeichen sind auch erlaubt, was allerdings vor allem beim Lesen von Kommandos aus Dateien von Bedeutung ist, denn die Bash läßt Sie nicht ohne weiteres Tabulatorzeichen eintippen.

Sie können sogar den Zeilentrenner (↩) verwenden, um ein langes Kommando auf mehrere Eingabezeilen zu verteilen, aber Sie müssen direkt davor ein »\« setzen, damit die Shell das Kommando nicht vorzeitig für vollständig hält.

Die an ein Kommando übergebenen Parameter können grob in zwei Klassen eingeteilt werden:

- Parameter, die mit einem Minuszeichen »`-`« beginnen, werden Optionen genannt. Diese können angegeben werden, aber müssen meistens nicht – die Details hängen vom jeweiligen Kommando ab. Bildlich gesprochen handelt es sich hierbei um »Schalter«, mit denen Sie Aspekte des jeweiligen Kommandos ein- oder ausschalten können. Möchten Sie mehrere Optionen übergeben, können diese (meistens) hinter einem einzigen Minuszeichen zusammengefasst werden, d. h. die Parameterfolge »`-a -l -F`« entspricht »`-alF`«. Viele Programme haben mehr Optionen, als sich leicht merkbar auf Buchstaben abbilden lassen, oder unterstützen aus Gründen der besseren Lesbarkeit »lange Optionen« (oft zusätzlich zu »normalen« Optionen, die dasselbe tun). Lange Optionen werden meist mit zwei Minuszeichen eingeleitet und können nicht zusammengefasst werden: »`bla --fasel --blubb`«.

Optionen

lange Optionen

- Parameter ohne einleitendes Minuszeichen nennen wir »Argumente«. Dies sind oft die Namen der Dateien, die mit dem Kommando bearbeitet werden sollen. Argumente

Die allgemeine Befehlsstruktur lässt sich also folgendermaßen darstellen: Befehlsstruktur

- Kommando – »Was wird gemacht?«
- Optionen – »Wie wird es gemacht?«
- Argumente – »Womit wird es gemacht?«

Üblicherweise stehen hinter einem Kommando erst die Optionen und dann kommen die Argumente. Allerdings bestehen nicht alle Kommandos darauf; bei manchen können Sie Argumente und Optionen beliebig mischen, und sie benehmen sich so, als stünden alle Optionen direkt hinter dem Kommando. Bei anderen dagegen werden Optionen erst in dem Moment beachtet, wo das Kommando beim Abarbeiten der Kommandozeile auf sie stößt.


Die Kommandostruktur heutiger Unix-Systeme (einschließlich Linux) ist über fast 40 Jahre historisch gewachsen und weist daher diverse Inkonsistenzen und kleine Überraschungen auf. Wir finden auch, dass man da mal gründlich aufräumen müßte, aber Shellskripte im Werte von 30 Jahren lassen sich leider nicht völlig ignorieren … Seien Sie also gefasst darauf, sich hin und wieder mit gewissen Merkwürdigkeiten anfreunden zu müssen.

4.3.3 Arten von Kommandos

In Shells gibt es prinzipiell zwei Arten von Kommandos:

Interne Kommandos Diese Befehle werden von der Shell selbst zur Verfügung gestellt. Zu dieser Gruppe gehören bei der Bash etwa 30 Kommandos, die den Vorteil haben, dass sie besonders schnell ausgeführt werden können. Einige Kommandos (etwa `exit` oder `cd`) können nur als interne Kommandos realisiert werden, weil sie den Zustand der Shell selbst beeinflussen.

Externe Kommandos Diese Kommandos führt die Shell nicht selber aus, sondern startet dafür ausführbare Dateien, die im Dateisystem üblicherweise in Verzeichnissen wie `/bin` oder `/usr/bin` abgelegt sind. Sie können als Benutzer Ihre eigenen Programme der Shell bekannt machen, so dass diese wie alle anderen externen Kommandos ausgeführt werden können.

Um die Art eines Kommandos heraus zu finden, können Sie das Kommando `type` benutzen. Übergeben Sie hier den Befehlsnamen als Argument, wird die Art des Kommandos oder der entsprechende Verzeichnispfad ausgegeben, etwa Extern oder intern?

```
$ type echo
echo is a shell builtin
$ type date
date is /bin/date
```

(`echo` ist ein nettes Kommando, das einfach nur seine Parameter ausgibt:

```
$ echo Ha, wackrer Tell! Das gleicht dem Waidgesellen!
Ha, wackrer Tell! Das gleicht dem Waidgesellen!
```

`date` liefert das aktuelle Datum und die Uhrzeit, gegebenenfalls angepasst an die aktuelle Zeitzone und Spracheinstellung:

```
$ date
Mo 12. Mär 09:57:34 CET 2012
```

Mehr über `echo` und `date` erfahren Sie in Kapitel 9.)

Hilfe

Hilfe für die internen Kommandos der Bash erhalten Sie übrigens über das Kommando `help`:

```
$ help type
type: type [-afptP] name [name ...]
    For each NAME, indicate how it would be interpreted if used as a
    command name.

    If the -t option is used, `type' outputs a single word which is one of
    `alias', `keyword', `function', `builtin', `file' or `', if NAME is an
◁◁◁◁◁
```

Übungen

4.2 [2] Welche der folgenden Kommandos sind bei der Bash extern und welche intern realisiert: `alias`, `echo`, `rm`, `test`?

4.3.4 Noch mehr Spielregeln

Wie schon weiter oben erwähnt, unterscheidet die Shell bei der Kommandoeingabe Groß- und Kleinschreibung. Das gilt nicht nur für die Kommandos selbst, sondern konsequenterweise auch für Optionen und Parameter (typischerweise Dateinamen).

Leerzeichen

Außerdem sollten Sie beachten, dass die Shell bestimmte Zeichen in der Eingabe besonders interpretiert. An erster Stelle ist hier das schon erwähnte Leerzeichen zu nennen, das als Trennzeichen für Wörter auf der Kommandozeile dient. Weitere Zeichen mit Sonderbedeutung sind

```
$&;(){}[]*?!<>«
```

Maskierung

Wenn Sie eines dieser Zeichen verwenden wollen, ohne dass es von der Shell in seiner besonderen Bedeutung interpretiert werden soll, müssen Sie es **maskieren**. Hierfür können Sie den Rückstrich »\« zum Maskieren eines einzelnen Sonderzeichens oder einfache oder doppelte Anführungszeichen (`'…'`, `"…"`) zum Maskieren mehrerer Sonderzeichen verwenden. Zum Beispiel:

```
$ touch 'Neue Datei'
```

Durch die Anführungszeichen bezieht dieses Kommando sich auf eine Datei namens `Neue Datei`. Ohne Anführungszeichen wären zwei Dateien namens `Neue` und `Datei` gemeint.

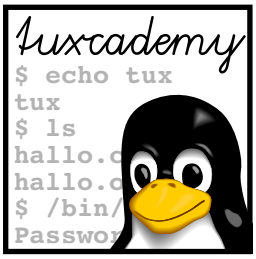
Die anderen Sonderzeichen zu erklären würde an dieser Stelle zu weit führen. Die meisten tauchen anderswo in dieser Schulungsunterlage auf – oder beziehen Sie sich auf die Bash-Dokumentation.

Kommandos in diesem Kapitel

bash	Die „Bourne-Again-Shell“, ein interaktiver Kommandointerpreter	bash(1)	63
date	Gibt Datum und Uhrzeit aus	date(1)	65
echo	Gibt alle seine Parameter durch Leerzeichen getrennt auf der Standardausgabe aus	bash(1), echo(1)	65
help	Zeigt Hilfe für bash-Kommandos	bash(1)	65
konsole	Ein „Terminalemulator“ für KDE	KDE: help:/konsole	63
type	Bestimmt die Art eines Kommandos (intern, extern, Alias)	bash(1)	65
xterm	Ein „Terminalemulator“ für das X-Window-System	xterm(1)	63

Zusammenfassung

- Die Shell liest Benutzerkommandos und führt sie aus.
- Die meisten Shells haben die Eigenschaften von Programmiersprachen und erlauben das Erstellen von Shellskripten, also vorgekochten Kommandofolgen, die in Dateien abgelegt werden.
- Kommandos haben Optionen und Argumente. Die Optionen geben an, *wie* das Kommando wirkt und die Argumente *worauf*.
- Shells unterscheiden zwischen *internen* Kommandos, die in der Shell selbst implementiert sind, und *externen* Kommandos, für die andere Programme als Hintergrundprozesse gestartet werden.



5

Hilfe

Inhalt

Lernziele

- Mit Handbuch- und Infoseiten umgehen können
- HOWTOs kennen und finden können
- Die wichtigsten anderen Informationsquellen kennen

Vorkenntnisse

- Linux-Überblick
- Kenntnisse über Linux auf der Kommandozeile (etwa aus den vorangegangenen Kapiteln)

5.1 Hilfe zur Selbsthilfe

Linux ist ein leistungsfähiges und vielschichtiges System, und leistungsfähige und vielschichtige Systeme sind in der Regel komplex. Ein wichtiges Hilfsmittel zur Beherrschung dieser Komplexität ist Dokumentation, und viele (leider nicht alle) Aspekte von Linux sind sehr ausführlich dokumentiert. Dieses Kapitel beschreibt einige Methoden zum Zugriff auf die Dokumentation.

»Hilfe« unter Linux bedeutet in vielen Fällen »Selbsthilfe«. Die Kultur der freien Software beinhaltet auch, dass man die Zeit und das Wohlwollen anderer Leute, die als Freiwillige in der Szene unterwegs sind, nicht unnötig mit Dingen strapaziert, die offensichtlich in den ersten paar Absätzen der Dokumentation erklärt sind. Als Linux-Anwender tun Sie gut daran, zumindest einen Überblick über die vorhandene Dokumentation und die empfohlenen Wege dafür zu haben, wo Sie notfalls Hilfe herbekommen können. Wenn Sie Ihre Hausaufgaben machen, werden Sie normalerweise erfahren, dass Ihnen aus der Klemme geholfen wird, aber die Toleranz gegenüber Leuten, die sich selber auf die faule Haut legen und erwarten, dass andere sich in ihrer Freizeit für sie in Knoten binden, ist nicht notwendigerweise besonders ausgeprägt.

Wenn Sie gerne möchten, dass Ihnen auch für die nicht so gründlich selber recherchierten Fragen und Probleme rund um die Uhr, sieben Tage die Woche, ein offenes Ohr zur Verfügung steht, müssen Sie auf eines der zahlreichen »kommerziellen« Support-Angebote zurückgreifen. Diese stehen für alle namhaften Distributionen zur Verfügung, teils vom Anbieter der Distribution selbst und teils von Drittfirmen. Vergleichen Sie die verschiedenen Dienstleister und suchen Sie sich einen davon aus, dessen Dienstgüteversprechen und Preis Ihnen zusagen.

5.2 Der `help`-Befehl und die `--help`-Option

Interne `bash`-Kommandos

Die in der Shell integrierten Kommandos werden durch den Befehl `help` mit dem entsprechenden Befehlsnamen als Argument genauer beschrieben:

```
$ help exit
exit: exit [n]
    Exit the shell with a status of N.
    If N is omitted, the exit status
    is that of the last command executed.
$ _
```

Ausführliche Erklärungen finden Sie in den Handbuchseiten und der Info-Dokumentation der Shell. Auf diese Informationsquellen gehen wir später in diesem Kapitel ein.

Option `--help` bei externen Kommandos

Viele externe Kommandos (Programme) unterstützen statt dessen die Option `--help`. Daraufhin erscheint bei den meisten Befehlen eine kurze Angabe, die Parameter und Syntax des Kommandos angibt.

Nicht jedes Kommando reagiert auf `--help`; oft heißt die Option `-h` oder `-?`, oder die Hilfe wird ausgegeben, wenn Sie irgendeine ungültige Option oder ansonsten illegale Kommandozeile angeben. Leider gibt es da keine einheitliche Konvention.

Tabelle 5.1: Gliederung der Handbuchseiten

Abschnitt	Inhalt
NAME	Kommandoname mit knapper Funktionsbeschreibung
SYNOPSIS	Beschreibung der Befehlssyntax
DESCRIPTION	Ausführliche Beschreibung der Kommandowirkung
OPTIONS	Die möglichen Optionen
ARGUMENTS	Die möglichen Argumente
FILES	Benötigte bzw. bearbeitete Dateien
EXAMPLES	Beispiele zur Anwendung
SEE ALSO	Querverweise auf verwandte Themen
DIAGNOSTICS	Fehlermeldungen des Kommandos
COPYRIGHT	Autor(en) des Kommandos
BUGS	Bekannte Fehler des Kommandos

5.3 Die Handbuchseiten

5.3.1 Überblick

Zu fast jedem kommandozeilenorientierten Programm gibt es eine »Handbuchseite« (engl. *manual page* oder kurz *manpage*), genau wie für viele Konfigurationsdateien, Systemaufrufe und so weiter. Diese Texte werden in der Regel bei der Installation einer Software mit installiert und können mit dem Kommando »`man` ⟨*Name*⟩« eingesehen werden. ⟨*Name*⟩ ist dabei der Kommando- oder Dateiname, den Sie erklärt haben möchten. »`man bash`« zum Beispiel liefert unter anderem eine Auflistung der oben erwähnten internen Kommandos der Shell.

Kommando `man`

Die Handbuchseiten haben für den Anwender allerdings einige Nachteile: Zum einen liegen viele davon nur in englischer Sprache vor. Lediglich einige Distributionen enthalten deutsche Übersetzungen, die allerdings oftmals recht knapp gehalten sind. Zum anderen sind die Texte oft sehr komplex. Jedes einzelne Wort kann bedeutsam sein, was dem Einsteiger den Zugang natürlich erschwert. Ferner ist gerade bei langen Texten die Aufteilung recht unübersichtlich. Dennoch ist der Wert dieser Dokumentation nicht zu unterschätzen. Statt den Anwender mit einer Unmenge Papier zu überhäufen, ist die Hilfe immer vor Ort verfügbar.

Viele Linux-Distributionen verfolgen die Philosophie, dass es zu jedem auf der Kommandozeile aufrufbaren Programm auch eine Handbuchseite geben muss. Dies gilt leider nicht im selben Umfang für Programme, die zu den grafischen Arbeitsumgebungen KDE und GNOME gehören, von denen viele nicht nur keine Handbuchseite haben, sondern die auch innerhalb der grafischen Umgebung überaus schlecht dokumentiert sind. Der Umstand, dass viele dieser Programme von Freiwilligen erstellt wurden, bietet hierfür nur eine schwache Entschuldigung.

5.3.2 Struktur

Der Aufbau der Handbuchseiten folgt lose der in Tabelle 5.1 angegebenen Gliederung. Allerdings enthält nicht jede Handbuchseite alle Punkte; vor allem die EXAMPLES kommen oft zu kurz.

Gliederung von Handbuchseiten

Die Überschrift BUGS wird gerne missverstanden: Echte *Fehler* in der Implementierung gehören natürlich repariert; was hier dokumentiert wird, sind in der Regel Einschränkungen, die aus dem *Ansatz* des Kommandos folgen, nicht mit vertretbarem Aufwand zu beheben sind und über die Sie als Anwender Bescheid wissen sollten. Beispielsweise wird in der Dokumentation zum Kommando `grep` darauf hingewiesen, dass bestimmte Konstrukte im

Tabelle 5.2: Themenbereiche der Handbuchseiten

Nr.	Themenbereich
1	Benutzerkommandos
2	Systemaufrufe
3	Funktionen der Programmiersprache C
4	Gerätedateien und Treiber
5	Konfigurationsdateien und Dateiformate
6	Spiele
7	Diverses (z. B. `groff`-Makros, ASCII-Tabelle, …)
8	Kommandos zur Systemadministration
9	Kernel-Funktionen
n	»Neue« Kommandos

zu suchenden regulären Ausdruck dazu führen können, dass ein `grep`-Prozess sehr viel Speicher braucht. Dies ist eine Konsequenz daraus, wie `grep` das Suchen implementiert, und kein trivialer, leicht zu behebender Fehler.

Handbuchseiten werden in einem speziellen Format geschrieben, das mit dem Programm `groff` für die Anzeige in einem Textterminal oder den Ausdruck aufbereitet werden kann. Die Quelltexte für die Handbuchseiten liegen im Verzeichnis `/usr/share/man` in Unterverzeichnissen der Form `man`*n*, wobei *n* eine der Kapitelnummern aus Tabelle 5.2 ist.

Handbuchseiten in anderen Verzeichnissen können Sie integrieren, indem Sie die Umgebungsvariable `MANPATH` setzen, die die von `man` durchsuchten Verzeichnisse und deren Reihenfolge benennt. Das Kommando `manpath` gibt Tipps für die `MANPATH`-Einstellung.

5.3.3 Kapitel

Kapitel

Jede Handbuchseite gehört zu einem »Kapitel« im konzeptuellen Gesamthandbuch (Tabelle 5.2). Wichtig sind vor allem die Kapitel 1, 5 und 8. Sie könnnen im `man`-Kommando eine Kapitelnummer angeben, um die Suche einzuschränken. So zeigt zum Beispiel »`man 1 crontab`« die Handbuchseite zum `crontab`-Kommando und »`man 5 crontab`« die Handbuchseite, die das Format von `crontab`-Dateien erklärt. Wenn man auf Handbuchseiten verweist, wird gerne das Kapitel in Klammern angehängt; wir unterscheiden also zwischen `crontab`(1), der Anleitung für das `crontab`-Kommando, und `crontab`(5), der Beschreibung des Dateiformats.

`man -a`

Mit dem Parameter `-a` zeigt `man` die zum Suchbegriff gehörenden Handbuchseiten aller Kapitel nacheinander an, ohne diesen Schalter wird nur der erste gefundene Text, also meist der im Kapitel 1, dargestellt.

5.3.4 Handbuchseiten anzeigen

Anzeigeprogramm

Als Anzeigeprogramm für das Kommando `man` ist in der Regel `less` voreingestellt, das noch ausführlich besprochen wird. Wichtig ist zu diesem Zeitpunkt nur, dass Sie sich mit den Cursortasten ↑ und ↓ durch den Handbuchtext bewegen können. Innerhalb des Textes lässt sich nach Drücken der Taste / ein Stichwort suchen. Nachdem das Wort eingetippt und abschließend die Eingabetaste betätigt wurde, springt der Cursor zum gesuchten Wort – sofern es denn im aktuellen Text vorkommt. Wenn Ihr Wissensdurst gestillt ist, können Sie die Anzeige durch Drücken der Taste q beenden und zur Shell zurückkehren.

Mit dem KDE-Webbrowser Konqueror ist es bequem möglich, ansprechend formatierte Handbuchseiten angezeigt zu bekommen. Geben Sie in der Adresszeile des Browsers den URL »`man:/`⟨*Name*⟩« oder einfach nur »`#`⟨*Name*⟩« an. Dasselbe funktioniert auch in der KDE-Befehlszeile.

Bevor Sie sich planlos durch die unzähligen Dokumentationen arbeiten, ist es oft sinnvoll, allgemeine Informationen zu einem Stichwort mittels `apropos` abzurufen. Dieses Kommando funktioniert genauso wie »`man -k`«. Beide suchen in den »NAME«-Abschnitten aller Handbuchseiten nach dem entsprechenden Stichwort. Als Ausgabe erhalten Sie eine Liste mit allen Handbuchseiten, die einen Eintrag zu dem eingegebenen Thema enthalten. Stichwortsuche

Verwandt ist das Kommando `whatis` (engl. »was ist«). Auch dieses sucht in allen Handbuchseiten, allerdings nicht nach einem Stichwort, sondern nach den eingegebenen Namen. Auf diese Weise erscheint eine kurze Beschreibung des mit dem Namen verbundenen Kommandos, Systemaufrufs o. ä. – eben der zweite Teil des »NAME«-Abschnitts der gefundenen Handbuchseite(n). `whatis` ist äquivalent zu »`man -f`«. `whatis` Namenssuche

Übungen

5.1 [!1] Schauen Sie sich die Handbuchseite zum Programm `ls` an. Benutzen Sie dafür das textbasierte Programm `man` und – falls vorhanden – den Konqueror-Browser.

5.2 [2] Welche Handbuchseiten auf Ihrem System beschäftigen sich (jedenfalls ihrem NAME-Abschnitt nach) mit Prozessen?

5.3 [5] (Für Fortgeschrittene.) Verwenden Sie einen Editor, um eine Handbuchseite für ein hypothetisches Kommando zu schreiben. Lesen Sie dazu vorher die Handbuchseite `man`(7). Prüfen Sie das Aussehen der Handbuchseite auf dem Bildschirm (mit »`groff -Tascii -man` ⟨*Datei*⟩ `| less`«) und in der Druckansicht (mit etwas wie »`groff -Tps -man` ⟨*Datei*⟩ `| gv -`«).

5.4 Die Info-Seiten

Für einige oft komplexere Kommandos existieren ausschließlich oder zusätzlich zur Handbuchseite so genannte Info-Seiten. Sie sind meist ausführlicher und basieren auf den Prinzipien von Hypertext, ähnlich zum World-Wide Web. Hypertext

Die Idee der Info-Seiten stammt aus dem GNU-Projekt, man findet sie daher vor allem bei Software, die von der FSF veröffentlicht wird oder anderweitig zum GNU-Projekt gehört. Ursprünglich sollte es im »GNU-System« *nur* Info-Dokumentation geben; da GNU aber auch zahlreiche Software integriert,

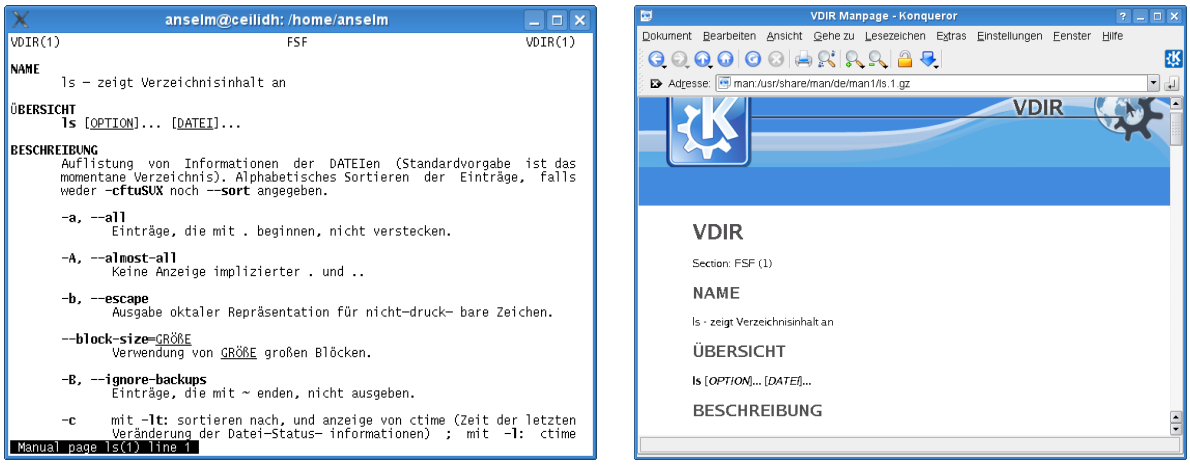


Bild 5.1: Eine Handbuchseite im Textterminal (links) und im Konqueror (rechts)

die nicht unter der Ägide der FSF erstellt wurde, und die GNU-Werkzeuge auch auf Systemen eingesetzt werden, die einen konservativeren Ansatz pflegen, ist die FSF in vielen Fällen weich geworden.

Info-Seiten werden analog zu Handbuchseiten mit dem Befehl »`info` ⟨*Kommando*⟩« aufgerufen (das Paket mit dem `info`-Programm muss möglicherweise zusätzlich installiert werden). Daneben kann man die Infoseiten auch mit dem Editor `emacs` betrachten oder im KDE-Browser Konqueror über URLs der Form »`info:/`⟨*Kommando*⟩« aufrufen.

Ein Vorteil von Info-Seiten ist, dass man sie (ähnlich wie bei den Handbuchseiten) in einem Quellformat schreibt, das bequem automatisch für die Bildschirmanzeige und das Ausdrucken von Handbüchern im PostScript- oder PDF-Format aufbereitet werden kann. Statt `groff` wird hier zur Aufbereitung das Programm TEX verwendet.

Übungen

5.4 [!1] Schauen Sie sich die Info-Seite zum Programm `ls` an. Probieren Sie das textbasierte `info` und, falls vorhanden, den Konqueror-Browser aus.

5.5 [2] Info-Dateien realisieren eine primitive (?) Form von Hypertext, ähnlich den HTML-Seiten im World-Wide Web. Warum werden die Info-Dateien nicht gleich in HTML geschrieben?

5.5 Die HOWTOs

Die Handbuch- und Info-Seiten haben das Problem, dass man im Grunde schon wissen muss, wie das Kommando heißt, das man benutzen möchte. Auch die Suche mit `apropos` ist oft nicht mehr als ein Glücksspiel. Außerdem läßt sich nicht jedes Problem auf ein spezielles Kommando reduzieren. Es besteht also Bedarf für »problemorientierte« statt »kommandoorientierte« Dokumentation. Dafür gibt es die **HOWTOs**.

Problemorientierte Dokumentation

HOWTOs

Die HOWTOs sind umfassendere Dokumente, die nicht nur einzelne Kommandos behandeln, sondern komplette Problemlösungen. So gibt es beispielsweise ein »DSL HOWTO«, das detailliert beschreibt, wie man einen Linuxrechner per DSL ans Internet anbindet, oder ein »Astronomy HOWTO«, das Astronomiesoftware für Linux diskutiert. Viele HOWTOs gibt es auch auf Deutsch; bei Übersetzungen kann es aber sein, dass die deutsche Version hinter dem Original herhinkt. (Einige HOWTOs sind von Anfang an auf Deutsch geschrieben.)

HOWTOs als Paket

Die meisten Linux-Distributionen ermöglichen es, die HOWTOs (oder eine signifikante Teilmenge davon) als Paket zu installieren. Sie befinden sich dann in einem distributionsabhängigen Verzeichnis – bei den SUSE-Distributionen `/usr/share/doc/howto`, bei Debian GNU/Linux `/usr/share/doc/HOWTO` –, typischerweise entweder als einfache Textdokumente oder im HTML-Format. Aktuelle Versionen der HOWTOs und Fassungen in anderen Formaten wie PostScript oder PDF sind im WWW von den Seiten des ***Linux Documentation Project*** (`http://www.tldp.org`) zu beziehen, wo auch andere Linux-Dokumentation zu finden ist.

HOWTOs im Netz

Linux Documentation Project

5.6 Weitere Informationsquellen

Zu (fast) jedem installierten Softwarepaket finden Sie auf Ihrem Rechner unter (typischerweise) `/usr/share/doc` oder `/usr/share/doc/packages` weitere Dokumentation und Beispieldateien. Außerdem gibt es für Programme unter der grafischen Oberfläche (z. B. KDE oder GNOME) entsprechende Hilfemenüs. Viele Distributionen bieten auch spezielle Hilfe-Center an, die bequemen Zugang auf die Dokumentation der verschiedenen Pakete gestatten.

weitere Dokumentation

Unabhängig vom lokalen Rechner gibt es im Internet Unmengen an Dokumentation, unter anderem im WWW und in USENET-Archiven. WWW USENET

Einige interessante Seiten für Linux sind die folgenden:

http://www.tldp.org/ Die Webseiten des »Linux Documentation Project«, das u. a. Handbuchseiten und HOWTOs betreut.

http://www.linux.org/ Ein allgemeines »Portal« für Linux-Interessenten. (Englisch)

http://www.linuxwiki.de/ Eine »Freiform-Text-Informationsdatenbank für alles, was mit GNU/Linux zusammenhängt.«

http://lwn.net/ *Linux Weekly News* – die vermutlich beste Web-Präsenz für Linux-Neuigkeiten aller Art. Neben einer täglichen Übersicht über die neuesten Entwicklungen, Produkte, Sicherheitslücken, Äußerungen pro und contra Linux in der Presse u. ä. erscheint jeden Donnerstag eine umfassende Online-Zeitschrift mit gründlich recherchierten Hintergrundberichten rund um die Vorkommnisse der Woche davor. Die täglichen Neuigkeiten sind frei zugänglich, während die wöchentliche Ausgabe kostenpflichtig abonniert werden muss (verschiedene Preisstufen ab US-$ 5 pro Monat). Eine Woche nach Erscheinen kann man auch auf die wöchentlichen Ausgaben kostenfrei zugreifen. (Englisch)

http://freecode.com/ Hier erscheinen Ankündigungen neuer (vornehmlich freier) Softwarepakete, die meist auch unter Linux zur Verfügung stehen. Daneben gibt es eine Datenbank, in der man nach interessanten Projekten oder Softwarepaketen recherchieren kann. (Englisch)

http://www.linux-knowledge-portal.de/ Eine Seite, die »Schlagzeilen« anderer interessanter Linux-Seiten zusammenträgt (darunter auch *Linux Weekly News* oder *Freshmeat*). (Deutsch/Englisch)

Wenn im WWW oder in USENET-Archiven nichts zu finden ist, gibt es die Möglichkeit, in Mailinglisten oder USENET-Newsgruppen Fragen zu stellen. Dabei sollten Sie jedoch beachten, dass viele Benutzer solcher Foren verärgert reagieren, wenn Sie Fragen posten, deren Antworten auch offensichtlich im Handbuch oder in **FAQ**-Sammlungen (engl. *frequently answered questions*) zu finden sind. FAQ Versuchen Sie, eine Problembeschreibung gründlich vorzubereiten und zum Beispiel mit relevanten Auszügen aus Protokolldateien zu untermauern, ohne die eine »Ferndiagnose« eines komplexen Problems nicht möglich ist (und die nicht komplexen Probleme können Sie ja sicher selber lösen …).

Zugriff auf ein *Newsarchiv* finden Sie u. a. bei `http://groups.google.de/` (ehemals Deja-News)

Interessante *Newsgroups* für Linux finden Sie in englischer Sprache in der `comp.os.linux.*`- und auf Deutsch in der `de.comp.os.unix.linux.*`-Hierarchie. Für viele Linux-Themen sind auch die entsprechenden Unix-Gruppen passend; eine Frage über die Shell steht besser in `de.comp.os.unix.shell` als in `de.comp.os.unix.linux.misc`, weil Shells zumeist keine Linux-spezifischen Programme sind.

Linux-orientierte Mailinglisten gibt es unter anderem bei `majordomo@vger.kernel.org`. Dabei handelt es sich um eine E-Mail-Adresse, an die Sie eine Nachricht mit dem Text »`subscribe LISTE`« schicken müssen, um eine Liste namens LISTE zu abonnieren. Eine kommentierte Aufstellung aller angebotenen Listen finden Sie unter `http://vger.kernel.org/vger-lists.html`. Mailinglisten

Eine probate Strategie zum Umgang mit scheinbar unerklärlichen Problemen besteht darin, die betreffende Fehlermeldung bei Google (oder einer anderen Suchmaschine Ihres Vertrauens) einzugeben. Wenn Sie nicht gleich ein hilfreiches Ergebnis erzielen, dann lassen Sie bei der Anfrage Teile weg, Suchmaschinen

die von Ihrer speziellen Situation abhängen (etwa Dateinamen, die es nur auf Ihrem System gibt). Der Vorteil ist, dass Google nicht nur die gängigen Webseiten indiziert, sondern auch viele Archive von Mailinglisten, und die Chancen gut stehen, dass Sie auf einen Dialog stoßen, wo jemand anderes ein sehr ähnliches Problem hatte wie Sie.

Freie Dokumentation

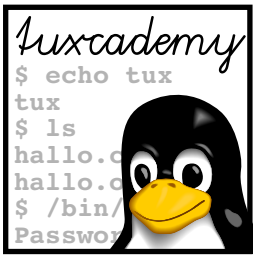
Der große Vorteil an Open-Source-Software ist übrigens nicht nur die riesige Menge an Dokumentation, sondern auch, dass die meisten Dokumente ähnlich wenig Restriktionen unterliegen wie die Software selbst. Dadurch ist eine umfassendere Zusammenarbeit von Softwareentwicklern und Dokumentationsautoren möglich, und auch die Übersetzung von Dokumentation in andere Sprachen ist einfacher. In der Tat gibt es genug Gelegenheit für Nichtprogrammierer, freien Entwicklungsprojekten zuzuarbeiten, indem sie zum Beispiel dabei helfen, gute Dokumentation zu erstellen. Die Freie-Software-Szene sollte versuchen, Dokumentationsautoren dieselbe Achtung entgegenzubringen wie Programmierern – ein Umdenkprozess, der zwar eingesetzt hat, aber noch bei weitem nicht abgeschlossen ist.

Kommandos in diesem Kapitel

apropos	Zeigt alle Handbuchseiten mit dem angegebenen Stichwort im „NAME“-Abschnitt	apropos(1) 72
groff	Programm zur druckreifen Aufbereitung von Texten	groff(1) 72
help	Zeigt Hilfe für bash-Kommandos	bash(1) 70
info	Zeigt GNU-Info-Seiten auf einem Textterminal an	info(1) 74
less	Zeigt Texte (etwa Handbuchseiten) seitenweise an	less(1) 72
man	Zeigt Handbuchseiten des Systems an	man(1) 70
manpath	Bestimmt den Suchpfad für Handbuchseiten	manpath(1) 72
whatis	Sucht Handbuchseiten mit dem gegebenen Stichwort in der Beschreibung	whatis(1) 73

Zusammenfassung

- Interne Kommandos der Bash erklärt »`help` ⟨*Kommando*⟩«. Externe Kommandos unterstützen oft eine Option `--help`.
- Für die meisten Programme gibt es mit `man` abrufbare Handbuchseiten. `apropos` sucht in allen Handbuchseiten nach Stichwörtern, `whatis` nach den Namen von Kommandos.
- Info-Seiten sind für manche Programme eine Alternative zu Handbuchseiten.
- HOWTOs stellen eine problemorientierte Form der Dokumentation dar.
- Es gibt eine Vielzahl interessanter Linux-Ressourcen im World-Wide Web und USENET.



6

Dateien: Aufzucht und Pflege

Inhalt

Lernziele

- Die Linux-Konventionen über Datei- und Verzeichnisnamen kennen
- Die wichtigsten Kommandos zum Umgang mit Dateien und Verzeichnissen beherrschen
- Mit Suchmustern in der Shell umgehen können

Vorkenntnisse

- Arbeit mit der Shell
- Umgang mit einem Texteditor (siehe Kapitel 3)

6.1 Datei- und Pfadnamen

6.1.1 Dateinamen

Dateien

Eine der wesentlichen Dienste eines Betriebssystems wie Linux besteht darin, Daten auf dauerhaften Speichermedien wie Festplatten oder USB-Sticks speichern und später wiederfinden zu können. Um das für Menschen erträglich zu gestalten, werden zusammengehörende Daten normalerweise zu »Dateien« zusammengefasst, die auf dem Speichermedium unter einem Namen gespeichert werden.

Auch wenn Ihnen das trivial vorkommen mag: Selbstverständlich ist das nicht. Betriebssysteme in früheren Zeiten machten es mitunter nötig, dass man Gräuel wie die passenden Spurnummern auf Platten kennen musste, wenn man seine Daten wieder finden wollte.

Bevor wir Ihnen erklären können, wie Sie mit Dateien umgehen, müssen wir Ihnen also erklären, wie Linux Dateien *benennt*.

Erlaubte Zeichen

In Linux-Dateinamen sind grundsätzlich alle Zeichen zugelassen, die der Computer darstellen kann (und noch ein paar mehr). Da einige dieser Zeichen aber eine besondere Bedeutung haben, raten wir von deren Verwendung in Dateinamen aber ab. Komplett verboten innerhalb von Dateinamen sind nur zwei Zeichen, der Schrägstrich und das Nullbyte (das Zeichen mit dem ASCII-Wert 0). Andere Zeichen wie Leerzeichen, Umlaute oder Dollarzeichen können verwendet werden, müssen dann aber auf der Kommandozeile meist durch einen Rückstrich oder Anführungszeichen maskiert werden, um Fehlinterpretationen durch die Shell zu vermeiden.

Groß- und Kleinschreibung

Eine Stolperfalle für Umsteiger ist, dass Linux in Dateinamen zwischen Groß- und Kleinschreibung unterscheidet. Im Gegensatz zu Windows, wo Groß- und Kleinbuchstaben in Dateinamen zwar dargestellt, aber identisch behandelt werden, interpretiert Linux `x-files` und `X-Files` tatsächlich als zwei verschiedene Dateinamen.

Dateinamen dürfen unter Linux »ziemlich lang« sein – eine feste allgemeinverbindliche Grenze gibt es nicht, da das Maximum vom »Dateisystem« abhängt, also der Methode, wie genau die Bytes auf der Platte arrangiert sind (davon gibt es unter Linux mehrere). Eine typische Obergrenze sind aber 255 Zeichen – und da so ein Name auf einem normalen Textterminal gut drei Zeilen einnehmen würde, sollte das Ihren Stil nicht übermäßig einschränken.

Endungen

Eine weitere Abweichung zu DOS- bzw. Windows-Rechnern besteht darin, dass bei Linux Dateien nicht durch entsprechende Endungen charakterisiert werden müssen. Der Punkt ist in einem Dateinamen also ein ganz gewöhnliches Zeichen. Ein Text kann daher als `blabla.txt` abgespeichert werden, aber einfach `blabla` wäre grundsätzlich ebenso zulässig. Was nicht heißt, dass Sie keine »Dateiendungen« benutzen sollten – immerhin erleichtern solche Kennzeichnungen die Identifizierung des Dateiinhaltes.

Manche Programme bestehen darauf, dass ihre Eingabedateien bestimmte Endungen haben. Der C-Übersetzer `gcc` zum Beispiel betrachtet Dateien mit Namen, die auf »`.c`« enden, als C-Quelltext, solche mit Namen auf »`.s`« als Assemblerquelltext und solche mit Namen auf »`.o`« als vorübersetzte Objektdateien.

Sonderzeichen

Umlaute und andere Sonderzeichen können Sie in Dateinamen grundsätzlich ungeniert verwenden. Sollen aber Dateien gleichermaßen auf anderen Systemen benutzt werden, ist es besser, auf Sonderzeichen in Dateinamen zu verzichten, da nicht garantiert ist, dass sie auf anderen Systemen als dieselben Zeichen erscheinen.

Was mit Sonderzeichen passiert, hängt auch von der Ortseinstellung ab, da es keinen allgemein üblichen Standard dafür gibt, Zeichen darzustellen, die den Zeichenvorrat des ASCII (128 Zeichen, die im wesentlichen die englische Sprache, Ziffern und die gängigsten Sonderzeichen abdecken) überschreiten. In weitem Gebrauch sind zum Beispiel die Zeichentabellen ISO 8859-1 und ISO 8859-15 (vulgo ISO-Latin-1 und ISO-Latin-9 … fragen Sie nicht) sowie ISO 10646, gerne salopp und nicht ganz korrekt als »Unicode« bezeichnet und in der Regel als »UTF-8« codiert. Dateinamen mit Sonderzeichen, die Sie anlegen, während Zeichentabelle *X* aktiv ist, können völlig anders aussehen, wenn Sie sich das Verzeichnis gemäß Zeichentabelle *Y* anschauen. Das ganze Thema ist nichts, worüber man beim Essen nachdenken möchte.

Ortseinstellung

Sollten Sie jemals in die Verlegenheit kommen, einen Berg Dateien vor sich zu haben, deren Namen allesamt gemäß einer verkehrten Zeichentabelle codiert sind, kann Ihnen vielleicht das Programm `convmv` helfen, das Dateinamen zwischen verschiedenen Zeichentabellen konvertieren kann. (Sie werden es vermutlich selber installieren müssen, da es sich normalerweise nicht im Standardumfang gängiger Distributionen befindet.) Allerdings sollten Sie sich damit erst befassen, wenn Sie den Rest dieses Kapitels durchgearbeitet haben, denn wir haben Ihnen noch nicht einmal das reguläre `mv` erklärt …

`convmv`

Ohne Bedenken lassen sich alle Zeichen aus

Portable Dateinamen

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
0123456789+-._
```

in Dateinamen benutzen. Allerdings sollten Sie die folgenden Tipps berücksichtigen:

- Um den Datenaustausch mit älteren Unix-Systemen zu ermöglichen, sollte die Länge eines Dateinamens gegebenenfalls maximal 14 Zeichen betragen.
- Dateinamen sollten stets mit einem der genannten Buchstaben oder einer Ziffer beginnen, die vier anderen Zeichen sind nur innerhalb eines Dateinamens problemlos verwendbar.

Diese Konventionen lassen sich am leichtesten mit ein paar Beispielen verstehen. Zulässige Dateinamen wären etwa:

```
X-files
bla.txt.bak
17.Juni
7_of_9
```

Schwierigkeiten wären dagegen möglich (wenn nicht gar wahrscheinlich oder sogar sicher) mit:

```
-20°C                    Beginnt mit »-«, Sonderzeichen
.profile                 Wird versteckt
3/4-Takt                 Enthält verbotenes Zeichen
Müll                     Enthält Umlaut
```

Als weitere Besonderheit werden Dateinamen, die mit einem Punkt (».«) anfangen, an einigen Stellen (etwa beim Auflisten von Verzeichnisinhalten) übersprungen – Dateien mit solchen Namen gelten als »versteckt«. Diese Eigenschaft verwendet man unter anderem gerne für Dateien, die Voreinstellungen für Programme enthalten und die in Dateinamenslisten nicht von wichtigeren Dateien ablenken sollen.

Versteckte Dateien


Für DOS- und Windows-Experten: Diese Systeme erlauben »versteckte« Dateien über ein unabhängig vom Namen zu setzendes »Dateiattribut«. Unter Linux bzw. Unix gibt es so etwas nicht.

6.1.2 Verzeichnisse

Da auf demselben Linux-System potentiell viele Benutzer arbeiten können, wäre es ein Problem, wenn es jeden Dateinamen nur einmal geben dürfte. Dem Benutzer Hugo wäre wohl nur schwer klar zu machen, dass er keine Datei namens `brief.txt` anlegen kann, weil die Benutzerin Susi schon eine Datei hat, die so heißt. Außerdem muss irgendwie (bequem) geregelt werden können, dass Hugo nicht alle Dateien von Susi lesen kann und umgekehrt.

Aus diesem Grund unterstützt Linux das Konzept hierarchischer »Verzeichnisse«, die zum Gruppieren von Dateien dienen. Dateinamen müssen nicht über das ganze System hinweg eindeutig sein, sondern nur im selben Verzeichnis. Das bedeutet insbesondere, dass Hugo und Susi vom System verschiedene Verzeichnisse zugeordnet bekommen können und darin dann jeweils ihre Dateien so nennen dürfen, wie sie mögen, ohne aufeinander Rücksicht nehmen zu müssen. Ferner kann man Hugo den Zugriff auf Susis *Verzeichnis* verbieten (und umgekehrt) und muss sich dann nicht mehr um die einzelnen Dateien kümmern, die dort jeweils stehen.

Verzeichnisse sind bei Linux auch nur Dateien, wenn auch Dateien, die Sie nicht mit denselben Mitteln bearbeiten können wie »gewöhnliche« Dateien. Das bedeutet aber, dass für die Namen von Verzeichnissen dieselben Regeln gelten wie für die Namen von Dateien (siehe voriger Abschnitt). Sie müssen nur noch lernen, dass der Schrägstrich (»/«) dazu dient, Dateinamen von Verzeichnisnamen und Verzeichnisnamen voneinander zu trennen. `hugo/brief.txt` wäre also die Datei `brief.txt` im Verzeichnis `hugo`.

Schrägstrich

Verzeichnisse können andere Verzeichnisse enthalten (das ist das »hierarchisch« von weiter vorne), wodurch sich eine baumartige Struktur ergibt (naheliegenderweise gerne »Verzeichnisbaum« genannt). Ein Linux-System hat ein spezielles Verzeichnis, das die Wurzel des Baums bildet und deswegen »Wurzelverzeichnis« oder neudeutsch *root directory* genannt wird. Es hat nach Konvention den Namen »/« (Schrägstrich).

Verzeichnisbaum


Trotz seiner englischen Bezeichnung hat das Wurzelverzeichnis nichts mit dem Systemverwalter `root` zu tun. Die beiden heißen einfach nur so ähnlich.


Der Schrägstrich spielt hier eine Doppelrolle – er dient sowohl als Name des Wurzelverzeichnisses als auch als Trennzeichen zwischen anderen Verzeichnissen. Hierzu gleich mehr.

Die Grundinstallation gängiger Linux-Distributionen enthält normalerweise Zehntausende von Dateien in einer größtenteils durch gewisse Konventionen vorstrukturierten Verzeichnishierarchie. Über diese Verzeichnishierarchie erfahren Sie mehr in Kapitel 10.

6.1.3 Absolute und relative Pfadnamen

Jede Datei in einem Linux-System wird durch einen Namen beschrieben, der sich ergibt, indem man ausgehend vom Wurzelverzeichnis jedes Verzeichnis nennt bis zu dem, in dem die Datei steht, gefolgt vom Namen der Datei selbst. Beispielsweise benennt `/home/hugo/brief.txt` die Datei `brief.txt`, die im Verzeichnis `hugo` steht, das wiederum im Verzeichnis `home` zu finden ist. Das Verzeichnis `home` steht direkt im Wurzelverzeichnis. Solche Namen, die vom Wurzelverzeichnis ausgehen, heißen »absolute Pfadnamen« (engl. *absolute path names*) – wir sprechen von »Pfadnamen«, weil der Name einen »Pfad« durch den Verzeichnisbaum beschreibt, in dem die Namen von Verzeichnissen und Dateien vorkommen können (es handelt sich also um einen Sammelbegriff).

absolute Pfadnamen

Jeder Prozess in einem Linux-System hat ein »aktuelles Verzeichnis« (engl. *current directory*, auch *working directory*, »Arbeitsverzeichnis«). Dateinamen werden in diesem Verzeichnis gesucht; `brief.txt` ist also eine bequeme Abkürzung für »die Datei `brief.txt` im aktuellen Verzeichnis«, und `susi/brief.txt`« steht für »die Datei `brief.txt` im Verzeichnis `susi`, das im aktuellen Verzeichnis steht«. Solche Namen, die vom aktuellen Verzeichnis ausgehen, nennen wir »relative Pfadnamen« (engl. *relative path names*). aktuelles Verzeichnis relative Pfadnamen

Sie können ganz einfach absolute von relativen Pfadnamen unterscheiden: Ein Pfadname, der mit »/« anfängt, ist absolut; alle anderen Pfadnamen sind relativ.

Das aktuelle Verzeichnis »vererbt« sich vom Elterprozess auf den Kindprozess. Wenn Sie also aus einer Shell heraus eine neue Shell (oder ein anderes Programm) starten, dann hat diese neue Shell dasselbe aktuelle Verzeichnis wie die Shell, aus der Sie sie gestartet haben. Sie können innerhalb der neuen Shell mit `cd` in ein anderes Verzeichnis wechseln, aber das aktuelle Verzeichnis der alten Shell ändert sich dadurch nicht – wenn Sie die neue Shell verlassen, dann sind Sie wieder im (unveränderten) aktuellen Verzeichnis der alten Shell.

In relativen (oder auch absoluten) Pfadnamen gibt es zwei bequeme Abkürzungen: Der Name »`..`« steht für das dem jeweiligen Verzeichnis im Verzeichnisbaum *übergeordnete* Verzeichnis – im Falle von `/home/hugo` also beispielsweise `/home`. Das erlaubt Ihnen oftmals, bequemer als über absolute Pfadnamen auf Dateien Bezug zu nehmen, die aus der Sicht des aktuellen Verzeichnisses in einem »Seitenast« des Verzeichnisbaums zu finden sind. Nehmen wir an, `/home/hugo` hätte die Unterverzeichnisse `briefe` und `romane`. Mit `briefe` als aktuellem Verzeichnis könnten Sie sich über den relativen Pfadnamen `../romane/werther.txt` auf die Datei `werther.txt` im Verzeichnis `romane` beziehen, ohne den umständlichen absoluten Namen `/home/hugo/romane/werther.txt` angeben zu müssen. Abkürzungen

Die zweite Abkürzung ergibt keinen ganz so offensichtlichen Sinn: Der Name »`.`« in einem Verzeichnis steht immer für das Verzeichnis selbst. Es leuchtet nicht unmittelbar ein, wofür man eine Methode braucht, auf ein Verzeichnis zu verweisen, in dem man sich bereits befindet, aber es gibt Situationen, wo das sehr nützlich ist. Zum Beispiel wissen Sie vielleicht schon (oder können in Kapitel 9 nachschauen), dass die Shell die Programmdateien für externe Kommandos in den Verzeichnissen sucht, die in der Umgebungsvariablen `PATH` aufgelistet sind. Wenn Sie als Softwareentwickler ein Programm, nennen wir es mal `prog`, aufrufen wollen, das (a) in einer Datei im aktuellen Verzeichnis steht und (b) dieses aktuelle Verzeichnis *nicht* in `PATH` zu finden ist (aus Sicherheitsgründen immer eine gute Idee), dann können Sie mit

```
$ ./prog
```

die Shell trotzdem dazu bringen, Ihre Datei als Programm zu starten, ohne dass Sie einen absoluten Pfadnamen angeben müssen.

Als Linux-Benutzer haben Sie ein »Heimatverzeichnis«, in dem Sie direkt nach dem Anmelden am System landen. Welches das ist, bestimmt der Systemadministrator beim Anlegen Ihres Benutzerkontos, aber es heißt normalerweise so wie Ihr Benutzername und ist unterhalb von `/home` zu finden – etwas wie `/home/hugo` für den Benutzer `hugo`.

6.2 Kommandos für Verzeichnisse

6.2.1 Das aktuelle Verzeichnis: `cd` & Co.

In der Shell können Sie mit dem Kommando `cd` das aktuelle Verzeichnis wechseln: Geben Sie einfach das gewünschte Verzeichnis als Parameter an. Verzeichnis wechseln

Tabelle 6.1: Einige Dateitypenkennzeichnungen in `ls`

Dateityp	Farbe	Zeichen (`ls -F`)	Kennung (`ls -l`)
gewöhnliche Datei	schwarz	keins	-
ausführbare Datei	grün	*	-
Verzeichnis	blau	/	d
Link	cyan	@	l

```
$ cd briefe                    Wechseln ins Verzeichnis briefe
$ cd ..                        Wechseln ins übergeordnete Verzeichnis
```

Wenn Sie keinen Parameter angeben, landen Sie in Ihrem Heimatverzeichnis:

```
$ cd
$ pwd
/home/hugo
```

Aktuellen Verzeichnisnamen ausgeben

Mit dem Kommando `pwd` (engl. *print working directory*) können Sie, wie im Beispiel zu sehen, den absoluten Pfadnamen des aktuellen Verzeichnisses ausgeben.

Eingabeaufforderung

Möglicherweise sehen Sie das aktuelle Verzeichnis auch an Ihrer Eingabeaufforderung; je nach der Voreinstellung Ihres Systems könnte da etwas erscheinen wie

```
hugo@red:~/briefe> _
```

Dabei ist `~/briefe` eine Abkürzung für `/home/hugo/briefe`; die Tilde (»~«) steht für das Heimatverzeichnis des aktuellen Benutzers.

Das Kommando »`cd -`« wechselt in das Verzeichnis, das vor dem letzten `cd`-Kommando aktuell war. Damit können Sie bequem zwischen zwei Verzeichnissen hin und her wechseln.

Übungen

6.1 [2] Ist `cd` ein externes Kommando oder als internes Kommando in die Shell eingebaut? Warum?

6.2 [3] Lesen Sie in der Handbuchseite der `bash` über die Kommandos `pushd`, `popd` und `dirs` nach. Überzeugen Sie sich, dass diese Kommandos funktionieren.

6.2.2 Dateien und Verzeichnisse auflisten – `ls`

Zur Orientierung im Verzeichnisbaum ist es wichtig, herausfinden zu können, welche Dateien und Verzeichnisse in einem Verzeichnis stehen. Hierzu dient das Kommando `ls` (*list*, »auflisten«).

Tabellenformat

Ohne Übergabe von Optionen werden diese Informationen als mehrspaltige, nach Dateinamen sortierte Tabelle ausgegeben. Da Farbbildschirme heutzutage keine Besonderheit mehr darstellen, hat es sich eingebürgert, Dateien verschiedenen Typs in verschiedenen Farben darzustellen. (Über Dateitypen haben wir noch nicht geredet; dieses Thema kommt im Kapitel 10 zur Sprache.)

Erfreulicherweise sind die meisten Distributionen sich über die Farben inzwischen weitgehend einig. Tabelle 6.1 nennt die verbreitetste Zuordnung.

Tabelle 6.2: Einige Optionen für `ls`

Option	Wirkung
`-a` oder `--all`	zeigt auch versteckte Dateien an
`-i` oder `--inode`	gibt die eindeutige Dateinummer (Inode-Nr.) aus
`-l` oder `--format=long`	liefert zusätzliche Informationen
`--color=no`	verzichtet auf die Farbcodierung
`-p` oder `-F`	markiert Dateityp durch angehängte Sonderzeichen
`-r` oder `--reverse`	kehrt Sortierreihenfolge um
`-R` oder `--recursive`	durchsucht auch Unterverzeichnisse (DOS: `DIR/S`)
`-S` oder `--sort=size`	sortiert Dateien nach Größe (größte zuerst)
`-t` oder `--sort=time`	sortiert Dateien nach Zeit (neueste zuerst)
`-X` oder `--sort=extension`	sortiert Dateien nach Dateityp

Auf Monochrom-Monitoren – auch die gibt es noch – bietet sich zur Unterscheidung die Angabe der Optionen `-F` oder `-p` an. Hierdurch werden zur Charakterisierung Sonderzeichen an die Dateinamen angehängt. Eine Auswahl der Zeichen zeigt Tabelle 6.1.

Versteckte Dateien, deren Name mit einem Punkt beginnt, können Sie mit dem Schalter `-a` (engl. *all*, »alle«) anzeigen. Sehr nützlich ist weiterhin der Parameter `-l` (das ist ein kleines »L« und steht für engl. *long*, »lang«). Damit werden nicht nur die Dateinamen, sondern auch noch diverse Zusatzinformationen ausgegeben.

Versteckte Dateien

Zusatzinformationen

In manchen Linux-Distributionen sind für gängige Kombination dieser hilfreichen Optionen Abkürzungen voreingestellt; in den SUSE-Distributionen steht etwa ein einfaches `l` für das Kommando »`ls -alF`«. Auch »`ll`« und »`la`« sind Abkürzungen für `ls`-Varianten.

Abkürzungen

Hier sehen Sie `ls` ohne und mit `-l`:

```
$ ls
datei.txt
datei2.dat
$ ls -l
-rw-r--r--  1  hugo users 4711  Oct 4 11:11 datei.txt
-rw-r--r--  1  hugo users  333  Oct 2 13:21 datei2.dat
```

Im ersten Fall werden alle sichtbaren Dateien des Verzeichnisses tabellarisch aufgelistet, im zweiten Fall kommen die Zusatzinformationen dazu.

Die einzelnen Teile der langen Darstellung haben folgende Bedeutung: Die erste Spalte gibt den Dateityp (siehe Kapitel 10) an; normale Dateien haben ein »-«, Verzeichnisse ein »`d`« und so weiter (»Kennbuchstabe« in Tabelle 6.1). Die nächsten neun Zeichen informieren über die Zugriffsrechte. Danach folgen ein Referenzzähler, der Eigentümer der Datei, hier `hugo`, und die Gruppenzugehörigkeit der Datei zur Gruppe `users`. Nach der Dateigröße in Bytes sind Datum und Uhrzeit der letzten Änderung zu sehen. Abgeschlossen wird die Zeile durch den Dateinamen.

Format der langen Darstellung

Je nachdem, welche Sprache Sie eingestellt haben, kann insbesondere die Datums- und Uhrzeitangabe ganz anders aussehen als in unserem Beispiel (das wir mit der Minimal-Sprachumgebung »`C`« erzeugt haben). Das ist im interaktiven Gebrauch normalerweise kein Problem, kann aber zu Ärger führen, wenn Sie in einem Shellskript versuchen, die Ausgabe von »`ls -l`« auseinander zu pflücken. (Ohne hier der Schulungsunterlage *Linux für Fortgeschrittene* vorgreifen zu wollen, empfehlen wir, in Shellskripten die Sprachumgebung auf einen definierten Wert zu setzen.)

Wenn Sie die Zusatzinformationen für ein Verzeichnis (etwa `/tmp`) sehen wollen, hilft »`ls -l /tmp`« Ihnen nicht weiter, denn `ls` listet dann die Daten für alle Dateien in `/tmp` auf. Mit der Option `-d` können Sie das unterdrücken und bekommen die Informationen über `/tmp` selbst.

Neben den besprochenen erlaubt `ls` noch eine Vielzahl weiterer Optionen, von denen die Wichtigsten in Tabelle 6.2 dargestellt sind.

In den LPI-Prüfungen *Linux Essentials* und LPI-101 erwartet niemand von Ihnen, dass Sie alle 57 Geschmacksrichtungen von `ls`-Optionen rauf und runter beten können. Das wichtigste gute halbe Dutzend – den Inhalt von Tabelle 6.2 oder so – sollten Sie aber schon auswendig parat haben.

Übungen

6.3 [1] Welche Dateien stehen im Verzeichnis `/boot`? Hat das Verzeichnis Unterverzeichnisse und, wenn ja, welche?

6.4 [2] Erklären Sie den Unterschied zwischen `ls` mit einem Dateinamen als Argument und `ls` mit einem Verzeichnisnamen als Argument.

6.5 [2] Wie können Sie `ls` dazu bringen, bei einem Verzeichnisnamen als Argument Informationen über das benannte *Verzeichnis* selbst anstatt über die darin enthaltenen *Dateien* zu liefern?

6.2.3 Verzeichnisse anlegen und löschen: `mkdir` und `rmdir`

Damit Sie Ihre eigenen Dateien möglichst überschaubar halten können, ist es sinnvoll, selbst Verzeichnisse anzulegen. In diesen »Ordnern« können Sie Ihre Dateien dann zum Beispiel nach Themengebieten sortiert aufbewahren. Zur weiteren Strukturierung können Sie natürlich auch in solchen Ordnern wiederum neue Unterverzeichnisse anlegen – hier sind Ihrem Gliederungseifer kaum Grenzen gesetzt.

Verzeichnisse anlegen

Zum Anlegen neuer Verzeichnisse steht das Kommando `mkdir` (engl. *make directory*, »erstelle Verzeichnis«) zur Verfügung. `mkdir` erwartet zwingend einen oder mehrere Verzeichnisnamen als Argument, andernfalls erhalten Sie statt eines neuen Verzeichnisses lediglich eine Fehlermeldung. Um verschachtelte Verzeichnisse in einem Schritt neu anzulegen, können Sie die Option `-p` angeben, ansonsten wird angenommen, dass alle Verzeichnisse bis auf das letzte im Namen schon existieren. Zum Beispiel:

```
$ mkdir bilder/urlaub
mkdir: cannot create directory `bilder/urlaub': No such file or directory
$ mkdir -p bilder/urlaub
$ cd bilder
$ ls -F
urlaub/
```

Verzeichnisse löschen

Es kann vorkommen, dass ein Verzeichnis nicht mehr benötigt wird. Zur besseren Übersicht können Sie es dann mit dem Kommando `rmdir` (engl. *remove directory*, »entferne Verzeichnis«) wieder entfernen.

In Analogie zum Kommando `mkdir` müssen Sie auch hier den Name mindestens eines zu löschenden Verzeichnisses angeben. Außerdem müssen die Verzeichnisse leer sein, dürfen also keinerlei Einträge (Dateien oder Unterverzeichnisse) mehr enthalten. Auch hier wird nur das letzte Verzeichnis in jedem übergebenen Namen entfernt:

```
$ rmdir bilder/urlaub
$ ls -F
⏪⏪⏪⏪
```

```
bilder/
◁◁◁◁◁
```

Durch Angabe der Option `-p` können Sie von rechts her alle leeren Unterverzeichnisse, die im Namen mit aufgeführt wurden, in einem Schritt mitbeseitigen.

```
$ mkdir -p bilder/urlaub/sommer
$ rmdir bilder/urlaub/sommer
$ ls -F bilder
bilder/urlaub/
$ rmdir -p bilder/urlaub
$ ls -F bilder
ls: bilder: No such file or directory
```

Übungen

6.6 [!2] Legen Sie in Ihrem Heimatverzeichnis ein Verzeichnis `grd1-test` mit den Unterverzeichnissen `dir1`, `dir2` und `dir3` an. Wechseln Sie ins Verzeichnis `grd1-test/dir1` und legen Sie (etwa mit einem Texteditor) eine Datei namens `hallo` mit dem Inhalt »`hallo`« an. Legen Sie in `grd1-test/dir2` eine Datei namens `huhu` mit dem Inhalt »`huhu`« an. Vergewissern Sie sich, dass diese Dateien angelegt wurden. Löschen Sie das Unterverzeichnis `dir3` mit `rmdir`. Versuchen Sie anschließend, das Unterverzeichnis `dir2` mit `rmdir` zu löschen. Was passiert und warum?

6.3 Suchmuster für Dateien

6.3.1 Einfache Suchmuster

Oft kommt es vor, dass Sie ein Kommando auf mehrere Dateien gleichzeitig ausführen wollen. Wenn Sie zum Beispiel alle Dateien, deren Namen mit »`p`« anfangen und mit »`.c`« enden, vom Verzeichnis `prog1` ins Verzeichnis `prog2` kopieren wollten, wäre es höchst ärgerlich, jede Datei explizit benennen zu müssen – jedenfalls wenn es sich um mehr als zwei oder drei Dateien handelt! Viel bequemer ist es, die Suchmuster der Shell einzusetzen.

Suchmuster

Wenn Sie auf der Kommandozeile der Shell einen Parameter angeben, der einen Stern enthält – etwa wie

Stern

```
prog1/p*.c
```

–, dann ersetzt die Shell diesen Parameter im tatsächlichen Programmaufruf durch eine sortierte Liste aller Dateinamen, die auf den Parameter »passen«. »Passen« dürfen Sie dabei so verstehen, dass im tatsächlichen Dateinamen statt des Sterns eine beliebig lange Folge von beliebigen Zeichen stehen darf. In Frage kommen zum Beispiel

```
prog1/p1.c
prog1/paulchen.c
prog1/pop-rock.c
prog1/p.c
```

(beachten Sie vor allem den letzten Namen im Beispiel – »beliebig lang« heißt auch »Länge Null«!). Das einzige Zeichen, auf das der Stern *nicht* passt, ist – na, fällt's Ihnen ein? – der Schrägstrich; es ist normalerweise besser, ein Suchmuster wie den Stern auf das aktuelle Verzeichnis zu beschränken.

Testen können Sie diese Suchmuster bequem mit `echo`. Ein

```
$ echo prog1/p*.c
```

gibt Ihnen unverbindlich und ohne weitere Konsequenzen jedweder Art die passenden Dateinamen aus.

Wenn Sie wirklich ein Kommando auf alle Dateien im *Verzeichnisbaum* ab einem bestimmten Verzeichnis ausführen wollen: Auch dafür gibt es Mittel und Wege. Wir reden darüber in Abschnitt 6.4.4.

Alle Dateien

Das Suchmuster »*« beschreibt »alle Dateien im aktuellen Verzeichnis« – mit Ausnahme der versteckten Dateien, deren Name mit einem Punkt anfängt. Um möglicherweise unangenehme Überraschungen auszuschließen, werden die versteckten Dateien nämlich von Suchmustern konsequent ignoriert, solange Sie nicht explizit mit etwas wie ».*« veranlassen, dass sie einbezogen werden.

Sie kennen den Stern vielleicht von der Kommandooberfläche von Betriebssystemen wie DOS oder Windows[1] und sind von dort gewöhnt, ein Muster wie »*.*« anzugeben, um alle Dateien in einem Verzeichnis zu beschreiben. Unter Linux ist das nicht richtig – das Muster »*.*« passt auf »alle Dateien mit einem Punkt im Namen«, aber der Punkt ist ja nicht vorgeschrieben. Das Linux-Äquvalent ist, wie erwähnt, »*«.

Fragezeichen

Ein Fragezeichen steht als Suchmuster für genau ein beliebiges Zeichen (wieder ohne den Schrägstrich). Ein Muster wie

```
p?.c
```

passt also auf die Namen

```
p1.c
pa.c
p-.c
p..c
```

(unter anderem). Beachten Sie, dass es aber ein Zeichen sein muss – die Option »gar nichts« besteht hier nicht.

Sie sollten sich einen ganz wichtigen Umstand sehr gründlich merken: *Die Expansion von Suchmustern ist eine Leistung der Shell!* Die aufgerufenen Kommandos wissen normalerweise nichts über Suchmuster und interessieren sich auch nicht die Bohne dafür. Alles, was sie zu sehen bekommen, sind Listen von Pfadnamen, ohne dass kommuniziert wird, wo diese Pfadnamen herkommen – also ob sie direkt eingetippt wurden oder über Suchmuster entstanden sind.

Es sagt übrigens niemand, dass die Ergebnisse der Suchmuster immer als Pfadnamen interpretiert werden. Wenn Sie zum Beispiel in einem Verzeichnis eine Datei namens »`-l`« haben, dann wird ein »`ls *`« in diesem Verzeichnis ein interessantes und vielleicht überraschendes Ergebnis liefern (siehe Übung 6.9).

Was passiert, wenn die Shell keine Datei findet, die auf das Suchmuster passt? In diesem Fall bekommt das betreffende Kommando das Suchmuster direkt als solches übergeben; was es damit anfängt, ist seine eigene Sache. Typischerweise werden solche Suchmuster als Dateinamen interpretiert, die betreffende »Datei« wird aber nicht gefunden und es gibt eine Fehlermeldung. Es gibt aber auch Kommandos, die mit direkt übergebenen Suchmustern Vernünftiges anfangen können – hier ist die Herausforderung dann eher, dafür zu sorgen, dass die Shell, die das Kommando aufruft, sich nicht mit ihrer eigenen Expansion vordrängelt. (Stichwort: Anführungszeichen)

[1] Für CP/M sind Sie wahrscheinlich zu jung.

6.3.2 Zeichenklassen

Eine etwas genauere Einschränkung der passenden Zeichen in einem Suchmuster bieten »Zeichenklassen«: In einem Suchmuster der Form

```
prog[123].c
```

passen die eckigen Klammern auf genau die Zeichen, die darin aufgezählt werden (keine anderen). Das Muster im Beispiel passt also auf

```
prog1.c
prog2.c
prog3.c
```

aber nicht

`prog.c`	*Ein Zeichen muss es schon sein*
`prog4.c`	*Die 4 ist nicht in der Aufzählung*
`proga.c`	*Das a auch nicht*
`prog12.c`	*Genau ein Zeichen, bitte!*

Bereiche

Als Schreibvereinfachung können Sie Bereiche angeben wie in

```
prog[1-9].c
[A-Z]brakadabra.txt
```

Die eckigen Klammern in der ersten Zeile passen auf alle Ziffern, die in der zweiten Zeile auf alle Großbuchstaben.

Denken Sie daran, dass in den gängigen Zeichencode-Tabellen die Buchstaben nicht lückenlos hintereinander liegen: Ein Muster wie

```
prog[A-z].c
```

passt nicht nur auf `progQ.c` und `progx.c`, sondern zum Beispiel auch auf `prog_.c`. (Schauen Sie in einer ASCII-Tabelle nach, etwa mit »`man ascii`«.) Wenn Sie nur »Groß- und Kleinbuchstaben« haben wollen, müssen Sie

```
prog[A-Za-z].c
```

schreiben.

Selbst von einer Konstruktion wie

```
prog[A-Za-z].c
```

werden Umlaute nicht erfasst, obwohl die verdächtig aussehen wie Buchstaben.

Komplement

Als weitere Schreibvereinfachung können Sie Zeichenklassen angeben, die als »alle Zeichen außer diesen« interpretiert werden: Etwas wie

```
prog[!A-Za-z].c
```

passt auf alle Namen, bei denen das Zeichen zwischen »`g`« und »`.`« *kein* Buchstabe ist. Ausgenommen ist wie üblich der Schrägstrich.

6.3.3 Geschweifte Klammern

Gerne in einem Atemzug mit Shell-Suchmustern erwähnt, auch wenn sie eigentlich eine nur entfernte Verwandte ist, wird die Expansion geschweifter Klammern in der Form

```
{rot,gelb,blau}.txt
```

– die Shell ersetzt dies durch

```
rot.txt gelb.txt blau.txt
```

Allgemein gilt, dass ein Wort auf der Kommandozeile, das einige durch Kommas getrennte Textstücke innerhalb von geschweiften Klammern enthält, durch so viele Wörter ersetzt wird, wie Textstücke zwischen den Klammern stehen, wobei in jedem dieser Wörter der komplette Klammerausdruck durch eins der Textstücke ersetzt wird. *Dieser Ersetzungsvorgang findet einzig und allein auf textueller Basis statt und ist völlig unabhängig von der Existenz oder Nichtexistenz irgendwelcher Dateien oder Verzeichnisse* – dies im Gegensatz zu Suchmustern, die immer nur solche Namen produzieren, die tatsächlich als Pfadnamen im System vorkommen.

kartesisches Produkt

Sie können auch mehr als einen Klammerausdruck in einem Wort haben, dann erhalten Sie als Ergebnis das kartesische Produkt, einfacher gesagt alle möglichen Kombinationen:

```
{a,b,c}{1,2,3}.dat
```

ergibt

```
a1.dat a2.dat a3.dat b1.dat b2.dat b3.dat c1.dat c2.dat c3.dat
```

Nützlich ist das zum Beispiel, um systematisch neue Verzeichnisse anzulegen; die normalen Suchmuster helfen da nicht, da sie ja nur bereits Existierendes finden können:

```
$ mkdir -p umsatz/200{8,9}/q{1,2,3,4}
```

Übungen

6.7 [!1] Im aktuellen Verzeichnis stehen die Dateien

```
prog.c   prog1.c  prog2.c  progabc.c  prog
p.txt    p1.txt   p21.txt  p22.txt    p22.dat
```

Auf welche dieser Dateinamen passen die Suchmuster (a) `prog*.c`, (b) `prog?.c`, (c) `p?*.txt`, (d) `p[12]*`, (e) `p*`, (f) `*.*`?

6.8 [!2] Was ist der Unterschied zwischen »`ls`« und »`ls *`«? (*Tipp:* Probieren Sie beides in einem Verzeichnis aus, das Unterverzeichnisse enthält.)

6.9 [2] Erklären Sie, warum das folgende Kommando zur angezeigten Ausgabe führt:

```
$ ls
-l  datei1  datei2  datei3
$ ls *
-rw-r--r-- 1 hugo users 0 Dec 19 11:24 datei1
-rw-r--r-- 1 hugo users 0 Dec 19 11:24 datei2
-rw-r--r-- 1 hugo users 0 Dec 19 11:24 datei3
```

6.10 [2] Warum ist es sinnvoll, dass »*« nicht auf Dateinamen mit Punkt am Anfang passt?

Tabelle 6.3: Optionen für `cp`

Option		Wirkung
`-b`	(*backup*)	Legt von existierenden Zieldateien Sicherungskopien an, indem an den Namen eine Tilde angehängt wird
`-f`	(*force*)	Überschreibt bereits existierende Zieldateien ohne Rückfrage
`-i`	(*interactive*)	fragt nach, ob bereits bestehende Dateien überschrieben werden sollen
`-p`	(*preserve*)	Behält möglichst alle Dateiattribute der Quelldatei bei
`-R`	(*recursive*)	Kopiert Verzeichnisse mit allen Unterverzeichnissen und allen darin enthaltenen Dateien
`-u`	(*update*)	Kopiert nur, wenn die Quelldatei neuer als die Zieldatei ist (oder noch keine Zieldatei existiert)
`-v`	(*verbose*)	Zeigt alle Aktivitäten auf dem Bildschirm

6.4 Umgang mit Dateien

6.4.1 Kopieren, Verschieben und Löschen – `cp` und Verwandte

Dateien kopieren

Mit dem Kommando `cp` (engl. *copy*, »kopieren«) können Sie beliebige Dateien kopieren. Dabei gibt es zwei Vorgehensweisen:

1 : 1-Kopie

Nennen Sie `cp` die Namen von Quell- und Zieldatei als Argumente, dann wird unter dem Zieldateinamen eine 1 : 1-Kopie des Inhalts der Quelldatei abgelegt. Standardmäßig fragt `cp` nicht nach, ob es eine bereits existierende Zieldatei überschreiben soll, sondern tut dies einfach – hier ist also Vorsicht (oder die Option `-i`) angebracht.

Zielverzeichnis

Statt eines Zieldateinamens können Sie auch ein Zielverzeichnis angeben. Die Quelldatei wird dann unter ihrem alten Namen in das Verzeichnis hineinkopiert.

```
$ cp liste liste2
$ cp /etc/passwd .
$ ls -l
-rw-r--r--  1  hugo  users   2500  Oct 4 11:11 liste
-rw-r--r--  1  hugo  users   2500  Oct 4 11:25 liste2
-rw-r--r--  1  hugo  users   8765  Oct 4 11:26 passwd
```

Im Beispiel haben wir mit `cp` zunächst eine exakte Kopie der Datei `liste` unter dem Namen `liste2` erzeugt. Anschließend haben wir noch die Datei `/etc/passwd` ins aktuelle Verzeichnis (repräsentiert durch den Punkt als Zielverzeichnis) kopiert. Die wichtigsten der zahlreichen `cp`-Optionen sind in Tabelle 6.3 aufgelistet.

Liste von Quelldateien

Statt einer einzigen Quelldatei sind auch eine längere Liste von Quelldateien oder die Verwendung von Shell-Suchmustern zulässig. Allerdings lassen sich auf diese Art nicht mehrere Dateien in einem Schritt kopieren und umbenennen, sondern nur in ein anderes Verzeichnis kopieren. Während in der DOS- und Windows-Welt mit »`COPY *.TXT *.BAK`« von jeder Datei mit der Endung `TXT` eine Sicherungskopie mit der Endung `BAK` angelegt wird, versagt unter Linux der Befehl »`cp *.txt *.bak`« normalerweise unter Ausgabe einer Fehlermeldung.

Um dies zu verstehen, müssen Sie sich vergegenwärtigen, wie die Shell diesen Befehl verarbeitet. Sie versucht zunächst, alle Suchmuster durch die passenden Dateinamen zu ersetzen, also zum Beispiel `*.txt` durch `brief1.txt` und `brief2.txt`. Was mit `*.bak` passiert, hängt davon ab, auf wieviele Namen `*.txt` gepasst hat und ob es im aktuellen Verzeichnis Namen gibt, die auf `*.bak` passen – allerdings wird fast nie das passieren, was ein DOS-Anwender erwarten würde! Normalerweise wird es wohl darauf hinauslaufen, dass die Shell dem `cp`-Kommando das unersetzte Suchmuster `*.bak` als letztes einer Reihe von Argumenten übergibt, und das geht aus der Sicht von `cp`

schief, da es sich dabei nicht um einen gültigen Verzeichnisnamen handelt (handeln dürfte).

Datei verschieben/umbenennen

Während das Kommando `cp` eine exakte Kopie einer Datei anlegt, also die Datei physikalisch auf dem Datenträger verdoppelt oder auf einem anderen Datenträger identisch anlegt, ohne das Original zu verändern, dient der Befehl `mv` (engl. *move*, »bewegen«) dazu, eine Datei entweder an einen anderen Ort zu verschieben oder deren Namen zu verändern. Dieser Vorgang verändert lediglich Verzeichnisse, es sei denn, die Datei wird auf ein anderes Dateisystem verlagert – etwa von einer Partition auf der Platte auf einen USB-Stick. Dabei wird dann tatsächlich ein physikalisches Verschieben (Kopieren an den neuen Platz und anschließendes Löschen am alten) notwendig.

Syntax und Regeln von `mv` entsprechen denen von `cp` – auch hier können Sie statt einer einzigen eine ganze Liste von Quelldateien angeben, woraufhin das Kommando als letzte Angabe ein Zielverzeichnis erwartet. Einziger Unterschied: Neben gewöhnlichen Dateien können Sie auch Verzeichnisse umbenennen.

Die Optionen `-b`, `-f`, `-i`, `-u` und `-v` entsprechen für `mv` in ihrer Funktion den bei `cp` beschriebenen Parametern.

```
$ mv passwd liste2
$ ls -l
-rw-r--r--  1  hugo  users   2500  Oct 4 11:11 liste
-rw-r--r--  1  hugo  users   8765  Oct 4 11:26 liste2
```

Im Beispiel ist die ursprüngliche Datei `liste2` durch die umbenannte Datei `passwd` ersetzt worden. Ebenso wie `cp` fragt auch `mv` nicht nach, wenn die angegebene Zieldatei bereits existiert, sondern überschreibt diese gnadenlos.

Löschen von Dateien

Der Befehl zum Löschen von Dateien lautet `rm` (engl. *remove*, »entfernen«). Um eine Datei löschen zu dürfen, müssen Sie im entsprechenden Verzeichnis Schreibrechte besitzen. Daher sind Sie im eigenen Heimatverzeichnis der uneingeschränkte Herr im Haus, dort dürfen Sie, gegebenenfalls nach einer entsprechenden Rückfrage, auch fremde Dateien löschen (falls es welche gibt).

Das Schreibrecht auf die *Datei* selbst ist dagegen zum Löschen völlig irrelevant, genau wie die Frage, welchem Benutzer oder welcher Gruppe die Datei gehört.

Löschen ist endgültig!

`rm` geht bei seiner Arbeit genauso konsequent vor wie `cp` oder `mv` – die angegebenen Dateien werden ohne Rückfrage oder -meldung unwiederbringlich aus dem Dateisystem getilgt. Besonders bei der Verwendung von Platzhaltern sollten Sie darum nur mit großer Vorsicht vorgehen. Im Unterschied zur DOS-Welt ist der Punkt innerhalb von Linux-Dateinamen ein Zeichen ohne besondere Bedeutung. Aus diesem Grund löscht das Kommando »`rm *`« unerbittlich alle nicht versteckten Dateien im aktuellen Verzeichnis. Zwar bleiben dabei wenigstens die Unterverzeichnisse unbehelligt; mit »`rm -r *`« müssen aber auch diese daran glauben.

Als Systemadministrator können Sie mit unüberlegten Kommandos wie »`rm -rf /`« im Wurzelverzeichnis das ganze System demolieren; hier ist allergrößte Aufmerksamkeit angebracht! Leicht tippt man einmal »`rm -rf bla *`« statt »`rm -rf bla*`«.

Wo `rm` ohne Rücksicht alle Dateien löscht, die als Parameter angegeben wurden, geht »`rm -i`« etwas behutsamer vor:

```
$ rm -i lis*
rm: remove 'liste'? n
rm: remove 'liste2'? y
$ ls -l
-rw-r--r--  1  hugo  users   2500  Oct 4 11:11 liste
```

Wie das Beispiel zeigt, wird für jede zum Suchmuster passende Datei einzeln nachgefragt, ob sie gelöscht werden soll (»y« für engl. *yes* = »ja«) oder nicht (»n« für engl. *no* = »nein«).

In der deutschen Sprachumgebung können Sie auch »j« für »ja« verwenden. »y« funktioniert trotzdem.

Arbeitsumgebungen wie KDE unterstützen meistens die Verwendung eines »Papierkorbs«, in dem über den Dateimanager gelöschte Dateien erst einmal landen und aus dem Sie sie bei Bedarf wieder hervorholen können. Es gibt äquivalente Ansätze auch für die Kommandozeile, wobei Sie auch da auf eine Umdefinition von `rm` verzichten sollten.

Neben den bereits beschriebenen Optionen `-i` und `-r` sind ferner die bei `cp` beschriebenen Optionen `-v` und `-f` mit vergleichbarer Wirkung für `rm` zulässig.

Übungen

6.11 [!2] Legen Sie in Ihrem Heimatverzeichnis eine Kopie der Datei `/etc/services` unter dem Namen `myservices` an. Benennen Sie sie um in `srv.dat` und kopieren Sie sie unter demselben Namen ins Verzeichnis `/tmp`. Löschen Sie anschließend beide Kopien der Datei.

6.12 [1] Warum hat `mv` keine `-R`-Option wie `cp`?

6.13 [!2] Angenommen, in einem Ihrer Verzeichnisse steht eine Datei namens »`-file`« (mit einem Minuszeichen am Anfang des Namens). Wie würden Sie diese Datei entfernen?

6.14 [2] Wenn Sie ein Verzeichnis haben, in dem Sie nicht versehentlich einem »`rm *`« zum Opfer fallen wollen, können Sie dort eine Datei namens »`-i`« anlegen, etwa mit

```
$ > -i
```

(wird in Kapitel 8 genauer erklärt). Was passiert, wenn Sie jetzt das Kommando »`rm *`« ausführen, und warum?

6.4.2 Dateien verknüpfen – `ln` und `ln -s`

In Linux können Sie Verweise auf Dateien, sogenannte *links*, anlegen und so einzelnen Dateien mehrere Namen geben. Aber wofür ist das gut? Die Anwendungsfälle reichen von Schreibvereinfachungen bei Datei- und Verzeichnisnamen über ein »Sicherheitsnetz« gegen ungewollte Löschoperationen oder Bequemlichkeiten beim Programmieren bis zum Platzsparen bei großen Dateihierarchien, die in mehreren Versionen mit nur kleinen Änderungen vorliegen sollen.

Mit dem Befehl `ln` (engl. *link*, »verknüpfen«) können Sie einer Datei neben dem ursprünglichen Namen (erstes Argument) einen weiteren (zweites Argument) zuweisen:

```
$ ln liste liste2
$ ls -l
-rw-r--r--  2  hugo  users   2500  Oct 4 11:11 liste
-rw-r--r--  2  hugo  users   2500  Oct 4 11:11 liste2
```

Eine Datei mit mehreren Namen

Das Verzeichnis enthält nun scheinbar eine neue Datei `liste2`. In Wahrheit handelt es sich hier jedoch nur um zwei Verweise auf ein und dieselbe Datei. Einen Hinweis auf diesen Sachverhalt liefert der **Referenzzähler**, der in der zweiten Spalte der Ausgabe von »`ls -l`« zu sehen ist. Dessen Wert 2 gibt an, dass diese Datei zwei Namen hat. Ob es sich bei den beiden Dateinamen nun wirklich um Verweise auf

Referenzzähler

Inode-Nummern

dieselbe Datei handelt, ist nur mit Hilfe von »`ls -i`« eindeutig zu entscheiden. In diesem Fall muss die in der ersten Spalte angezeigte Dateinummer für beide Dateien identisch sein. Dateinummern, auch **Inode-Nummern** genannt, identifizieren Dateien eindeutig auf ihrem Dateisystem:

```
$ ls -i
876543 liste  876543 liste2
```

»Inode« ist kurz für *indirection node*, also etwa »Indirektionsknoten« oder »Weiterleitungsknoten«. In den Inodes sind alle Informationen gespeichert, die das System über eine Datei hat, bis auf den Namen. Jede Datei hat genau ein Inode.

Wenn Sie den Inhalt einer der beiden Dateien verändern, ändert sich der Inhalt der anderen ebenfalls, da in Wirklichkeit ja nur eine einzige Datei mit der eindeutigen Nummer 876543 existiert. Wir haben der Datei mit der Nummer 876543 lediglich einen weiteren Namen gegeben.

Verzeichnisse sind nichts anderes als Tabellen, in denen Dateinamen Inode-Nummern zugeordnet werden. Offensichtlich kann es in einer Tabelle mehrere Einträge mit verschiedenen Namen und derselben Inode-Nummer geben. Ein Verzeichniseintrag mit einem Namen und einer Inode-Nummer heißt »Link«.

Sie sollten sich klar machen, dass es bei einer Datei mit zwei Links nicht möglich ist, festzustellen, welcher Name »das Original« ist, also der erste Parameter im `ln`-Kommando. Beide Namen sind aus der Sicht des Systems absolut gleichwertig und ununterscheidbar.

Links auf Verzeichnisse sind unter Linux übrigens nicht erlaubt. Ausgenommen davon sind »`.`« und »`..`«, die vom System für jedes Verzeichnis verwaltet werden. Da auch Verzeichnisse Dateien sind und demnach Inode-Nummern haben, können Sie so verfolgen, wie das Dateisystem intern zusammenhängt. (Siehe hierzu auch Übung 6.20)

Wenn Sie eine der Dateien mit `rm` »löschen«, reduziert das zunächst nur die Anzahl der Namen für Datei Nummer 876543 (der Referenzzähler wird entsprechend angepasst). Erst wenn der Referenzzähler durch das Entfernen eines Namens den Wert 0 annimmt, wird die Datei tatsächlich gelöscht.

```
$ rm liste
$ ls -li
876543 -rw-r--r--  1  hugo  users   2500  Oct 4 11:11 liste2
```

Da Inode-Nummern nur auf demselben physikalischen Dateisystem (Partition, USB-Stick, …) eindeutig sind, sind solche Verknüpfungen nur in demselben Dateisystem möglich, wo auch die Datei liegt.

Ganz stimmt es nicht mit dem Löschen der Dateidaten: Wenn der letzte Dateiname entfernt wird, ist die Datei nicht mehr zugänglich, aber wenn ein Prozess noch mit ihr arbeitet, darf er sie weiterbenutzen, bis er sie explizit schließt oder sich beendet. In Unix-Programmen ist das ein gängiges Idiom für den Umgang mit temporären Dateien, die beim Programmende verschwinden sollen: Sie erzeugen sie mit Schreib- und Lesezugriff und »löschen« sie dann gleich wieder, ohne sie jedoch gleich zu schließen. Anschließend können Sie Daten hinein schreiben und später an den Dateianfang zurückspringen, um sie wieder zu lesen.

`ln` können Sie außer mit zwei Argumenten auch mit einem oder vielen aufrufen. Im ersten Fall wird im aktuellen Verzeichnis ein Link mit demselben Namen angelegt wie die Ursprungsdatei (die sinnvollerweise nicht im aktuellen Verzeichnis liegen sollte), im zweiten Fall werden alle benannten Dateien unter ihrem Namen in das durch das letzte Argument gegebene Verzeichnis »gelinkt« (denken Sie an `mv`).

Mit dem Kommando »`cp -l`« können Sie eine »Link-Farm« anlegen. Das heißt, die angegebenen Dateien werden nicht (wie sonst üblich) an den Zielort kopiert, sondern es werden Links auf die Originale angelegt: Link-Farm

```
$ mkdir prog-1.0.1                                    Neues Verzeichnis
$ cp -l prog-1.0/* prog-1.0.1
```

Der Vorteil dieses Ansatzes ist, dass die Dateien nach wie vor nur einmal auf der Platte stehen und damit auch nur einmal Platz verbrauchen. Bei den heutigen Preisen für Plattenplatz ist das vielleicht nicht zwingend nötig – aber eine gängige Anwendung dieser Idee besteht zum Beispiel darin, regelmäßige Sicherheitskopien von großen Dateihierarchien anzulegen, die auf dem Archivmedium (Platte oder entfernter Rechner) dann als separate, datierte Dateibäume erscheinen sollen. Erfahrungsgemäß ändern sich die meisten Dateien nur selten, und wenn man diese Dateien dann nur einmal abspeichern muss statt immer und immer wieder, dann addiert sich das mit der Zeit schon auf. Außerdem muss man die Dateien dann nicht immer wieder in die Sicherheitskopie schreiben, was mitunter massiv Zeit spart.

Backup-Pakete, die diese Idee aufgreifen, sind zum Beispiel Rsnapshot (`http://www.rsnapshot.org/`) oder Dirvish (`http://www.dirvish.org/`).

Dieser Ansatz ist mitunter mit Vorsicht zu genießen: Zwar können identische Dateien über Links »dedupliziert« werden, Verzeichnisse aber nicht. Das heißt, für jeden datierten Verzeichnisbaum auf dem Archivmedium müssen sämtliche Verzeichnisse neu angelegt werden, selbst wenn in den Verzeichnissen wiederum nur Links auf existierende Dateien stehen. Das kann zu sehr komplexen Verzeichnisstrukturen führen und im Extremfall dazu, dass eine Konsistenzprüfung des Archivmediums fehlschlägt, weil der Rechner nicht genug Hauptspeicher hat, um die Verzeichnishierarchie zu prüfen.

Sie müssen natürlich auch aufpassen, wenn Sie etwa – wie im Beispiel angedeutet – eine »Kopie« eines Programmquellcodes als Link-Farm machen (was sich zum Beispiel für den Linux-Quellcode durchaus plattenplatzmäßig rentieren könnte): Bevor Sie eine Datei in Ihrer neu angelegten Version ändern können, müssen Sie dafür sorgen, dass es sich tatsächlich um eine eigenständige Datei handelt und nicht nur um ein Link auf das Original (das Sie ja höchstwahrscheinlich nicht ändern wollen). Das heißt, Sie müssen das Link auf die Datei entweder manuell durch eine tatsächliche Kopie ersetzen oder einen Editor benutzen, der geänderte Versionen automatisch als eigenständige Datei schreibt[2].

Das ist noch nicht alles: In Linux-Systemen gibt es zwei Arten von Links. Das oben stehende Beispiel entspricht dem Standard des Kommandos `ln` und wird als *hard link* (engl. »feste Verknüpfung«) bezeichnet. Zur Identifikation der Datei wird dabei die Dateinummer gespeichert. **Symbolische Links** (oder in Anlehnung an die *hard links* von eben auch *soft links*, »weiche Verknüpfungen«) sind selbst eigentlich Dateien, in denen aber nur der Name der »Zieldatei« des Links steht; gleichzeitig wird vermerkt, dass es sich um ein symbolisches Link handelt und Symbolische Links

[2] Wenn Sie den Vim (alias `vi`) verwenden, können Sie in die Datei `.vimrc` in Ihrem Heimatverzeichnis das Kommando »`set backupcopy=auto,breakhardlink`« schreiben.

bei Zugriffen auf das Link in Wirklichkeit die Zieldatei gemeint ist. Anders als bei harten Links »weiß« die Zieldatei nicht, dass ein symbolisches Link auf sie existiert. Das Anlegen oder Löschen eines symbolischen Links hat keine Auswirkungen auf die Zieldatei; wird dagegen die Zieldatei entfernt, weist der symbolische Link ins Leere (Zugriffe führen zu einer Fehlermeldung).

Verweise auf Verzeichnisse

Im Gegensatz zu harten Links lassen symbolische Links auch Verweise auf Verzeichnisse zu sowie auf Dateien, die sich im Verzeichnisbaum auf anderen physikalischen Dateisystemen befinden. In der Praxis werden symbolische Links oft bevorzugt, da sich die Verknüpfungen anhand des Pfadnamens leichter nachvollziehen lassen.

Symbolische Links werden gerne verwendet, wenn sich Datei- oder Verzeichnisnamen ändern, aber eine gewisse Rückwärtskompatibilität erwünscht ist. Beispielsweise hat man sich geeinigt, die Postfächer von Systembenutzern (wo deren ungelesene E-Mail steht) im Verzeichnis `/var/mail` abzulegen. Traditionell hieß das Verzeichnis `/var/spool/mail`, und viele Programme haben diesen Namen fest einprogrammiert. Um eine Umstellung auf `/var/mail` zu erleichtern, kann eine Distribution also ein symbolisches Link unter dem Namen `/var/spool/mail` einrichten, das auf `/var/mail` verweist. (Mit harten Links wäre das nicht möglich, weil harte Links auf Verzeichnisse nicht erlaubt sind.)

Um ein symbolisches Link zu erzeugen, müssen Sie dem Kommando `ln` die Option `-s` übergeben:

```
$ ln -s /var/log kurz
$ ls -l
-rw-r--r--  1  hugo  users   2500  Oct 4 11:11 liste2
lrwxrwxrwx  1  hugo  users     14  Oct 4 11:40 kurz -> /var/log
$ cd kurz
$ pwd -P
/var/log
```

Neben der Option `-s` zur Erstellung von »soft links« unterstützt das Kommando `ln` unter anderem die bereits bei `cp` besprochenen Optionen `-b`, `-f`, `-i` und `-v`.

Um nicht mehr benötigte symbolische Links wieder zu entfernen, können Sie sie einfach wie gewöhnliche Dateien mit dem Kommando `rm` löschen. *Diese* Operation wirkt auf das Link und nicht das Ziel des Links:

```
$ cd
$ rm kurz
$ ls
liste2
```

Wie Sie weiter oben gesehen haben, zeigt »`ls -l`« für symbolische Links auch die Datei an, auf die das Link zeigt. Mit den Optionen `-L` und `-H` können Sie `ls` dazu bringen, symbolische Links direkt aufzulösen:

```
$ mkdir dir
$ echo XXXXXXXXX >dir/datei
$ ln -s datei dir/symlink
$ ls -l dir
insgesamt 4
-rw-r--r-- 1 hugo users 11 Jan 21 12:29 datei
lrwxrwxrwx 1 hugo users  5 Jan 21 12:29 symlink -> datei
$ ls -lL dir
-rw-r--r-- 1 hugo users 11 Jan 21 12:29 datei
-rw-r--r-- 1 hugo users 11 Jan 21 12:29 symlink
$ ls -lH dir
```

```
-rw-r--r-- 1 hugo users 11 Jan 21 12:29 datei
lrwxrwxrwx 1 hugo users  5 Jan 21 12:29 symlink -> datei
$ ls -l dir/symlink
lrwxrwxrwx 1 hugo users 5 Jan 21 12:29 dir/symlink -> datei
$ ls -lH dir/symlink
-rw-r--r-- 1 hugo users 11 Jan 21 12:29 dir/symlink
```

Der Unterschied zwischen `-L` und `-H` ist, dass die Option `-L` symbolische Links *immer* auflöst und die Informationen über die eigentliche Datei anzeigt (der angezeigte Name ist trotzdem immer der des Links). Die Option `-H` tut das, wie die letzten drei Kommandos im Beispiel illustrieren, nur für direkt auf der Kommandozeile angegebene Links.

`cp` und symbolische Links

In Analogie zu »`cp -l`« erstellt »`cp -s`« Link-Farmen auf der Basis von symbolischen Links. Die sind allerdings nicht ganz so nützlich wie die oben gezeigten mit harten Links. »`cp -a`« kopiert Dateihierarchien so, wie sie sind, unter Beibehaltung symbolischer Links als solche; »`cp -L`« sorgt beim Kopieren dafür, dass symbolische Links durch die Dateien ersetzt werden, auf die sie zeigen, und »`cp -P`« schließt das aus.

Übungen

6.15 [!2] Erzeugen Sie eine Datei mit beliebigem Inhalt in Ihrem Heimatverzeichnis (etwa mit »`echo Hallo >~/hallo`« oder einem Texteditor). Legen Sie unter dem Namen `link` ein hartes Link auf die Datei an. Überzeugen Sie sich, dass nun zwei Namen für die Datei existieren. Versuchen Sie den Dateiinhalt mit einem Editor zu ändern. Was passiert?

6.16 [!2] Legen Sie unter dem Namen `~/symlink` ein symbolisches Link auf die Datei aus der vorigen Aufgabe an. Prüfen Sie, ob der Zugriff funktioniert. Was passiert, wenn Sie die Zieldatei des Links löschen?

6.17 [!2] Auf welches Verzeichnis zeigt das `..`-Link im Verzeichnis »`/`«?

6.18 [3] Betrachten Sie das folgende Kommando und seine Ausgabe:

```
$ ls -ai /
     2 .      330211 etc         1 proc   4303 var
     2 ..          2 home    65153 root
  4833 bin    244322 lib    313777 sbin
228033 boot   460935 mnt    244321 tmp
330625 dev    460940 opt    390938 usr
```

Offensichtlich haben die Verzeichnisse `/` und `/home` dieselbe Inodenummer. Da es sich dabei offensichtlich nicht wirklich um dasselbe Verzeichnis handeln kann – können Sie dieses Phänomen erklären?

6.19 [2] Betrachten Sie die Inode-Nummern der Links »`.`« und »`..`« im Wurzelverzeichnis (`/`) und in einigen beliebigen anderen Verzeichnissen. Was fällt Ihnen auf?

6.20 [3] Wir haben gesagt, dass harte Links auf Verzeichnisse nicht erlaubt sind. Welchen Grund könnte es dafür geben?

6.21 [3] Woran erkennt man in der Ausgabe von »`ls -l ~`«, dass ein *Unterverzeichnis* von `~` keine weiteren Unterverzeichnisse hat?

6.22 [2] Wie verhalten sich »`ls -lH`« und »`ls -lL`«, wenn ein symbolisches Link auf ein anderes symbolisches Link zeigt?

Tabelle 6.4: Tastaturbefehle für `more`

Taste	Wirkung
↩	zeigt nächste Zeile an
␣	zeigt nächste Seite an
b	zeigt vorherige Seite an
h	gibt einen Hilfstext aus
q	beendet `more`
/ ⟨*Wort*⟩ ↩	sucht nach ⟨*Wort*⟩
! ⟨*Kommando*⟩ ↩	führt ⟨*Kommando*⟩ in Sub-Shell aus
v	ruft Editor (`vi`) auf
Strg + l	zeichnet Bildschirm neu

6.23 [3] Wie lang darf eine »Kette« von symbolischen Links maximal sein? (Mit anderen Worten, wenn Sie mit einem symbolischen Link auf eine Datei anfangen, wie oft können Sie ein symbolisches Link anlegen, das auf das jeweils vorige symbolische Link verweist?)

6.24 [4] (Nachdenk- und Rechercheaufgabe:) Was braucht mehr Platz auf der Platte, ein hartes oder ein symbolisches Link? Warum?

6.4.3 Dateiinhalte anzeigen – `more` und `less`

Darstellung von Textdateien

Eine komfortable Darstellung von Textdateien am Bildschirm ist mittels `more` (engl. für »mehr«) möglich. Mit diesem Kommando können Sie längere Texte seitenweise betrachten. Die Anzeige wird dabei nach einer Bildschirmseite automatisch angehalten und in der letzten Zeile »`--More--`«, gegebenenfalls ergänzt durch eine Prozentangabe, ausgegeben. Der Prozentwert zeigt, welcher Anteil der Datei bereits dargestellt wurde. Auf Tastendruck wird die Ausgabe des restlichen Textes fortgesetzt. Welche Tasten eine Funktion besitzen, zeigt Tabelle 6.4.

Optionen

`more` erlaubt natürlich auch einige Optionen. Durch `-s` (engl. *squeeze*, »quetschen«) werden mehrere aufeinanderfolgende Leerzeilen zu einer einzigen zusammengefasst, Seitenvorschübe (repräsentiert durch das Symbol »^L«) werden bei Übergabe von `-l` ignoriert. Die Zeilenzahl des Bildschirms kann mit `-n` ⟨*Zahl*⟩ auf ⟨*Zahl*⟩ Zeilen eingestellt werden, andernfalls liest `more` den aktuellen Wert aus der Variablen `TERM`.

Einschränkungen

Die Textdarstellung von `more` ist noch immer gravierenden Einschränkungen unterworfen. Etwa fehlt eine allgemein mögliche Rückwartsbewegung in der Ausgabe. Aus diesem Grund ist heute meist die verbesserte Version `less` (engl. für »weniger«[3]) im Einsatz. Nun können Sie sich mit den Cursortasten wie gewohnt vertikal durch den Text bewegen. Ferner wurden die Suchroutinen erweitert und ermöglichen die Suche sowohl textauf- als auch -abwärts. Tabelle 6.5 bietet einen Überblick über die wichtigsten Tastaturbefehle.

`less`

Wie bereits in Kapitel 5 erwähnt, ist `less` üblicherweise als Anzeigeprogramm für `man` voreingestellt. Alle diese Tastaturkommandos greifen daher auch bei der Anzeige der Online-Hilfe via `man`.

6.4.4 Dateien suchen – `find`

Welcher Anwender kennt das nicht: »Da gab's doch mal eine Datei `sowieso`, aber wo war die gleich?« Natürlich können Sie alle Verzeichnisse mühsam von Hand nach der gesuchten Datei durchkämmen. Aber Linux wäre nicht Linux, wenn es Ihnen nicht hilfreich zur Seite stünde.

Das Kommando `find` sucht den Verzeichnisbaum rekursiv nach Dateien ab, die den angegebenen Kriterien entsprechen. »Rekursiv« bedeutet, dass es auch

[3] Ein schwaches Wortspiel – denken Sie an »weniger ist mehr«.

Tabelle 6.5: Tastaturbefehle für `less`

Taste	Wirkung
↓ oder e oder j oder ↩	zeigt nächste Zeile an
f oder Leertaste	zeigt nächste Seite an
↑ oder y oder k	zeigt vorherige Zeile an
b	zeigt vorherige Seite an
Pos1 oder g	an den Textanfang springen
Ende oder ⇑+g	ans Textende springen
p ⟨*Zahl*⟩ ↩	springt an Position ⟨*Zahl*⟩ (in %) des Textes
h	gibt einen Hilfetext aus
q	beendet `less`
/ ⟨*Wort*⟩ ↩	sucht abwärts nach ⟨*Wort*⟩
n	setzt Suche abwärts fort
? ⟨*Wort*⟩ ↩	sucht aufwärts nach ⟨*Wort*⟩
⇑+n	setzt Suche aufwärts fort
! ⟨*Kommando*⟩ ↩	führt ⟨*Kommando*⟩ in Sub-Shell aus
v	ruft Editor (`vi`) auf
r oder Strg+l	zeichnet Bildschirm neu

alle im Startverzeichnis enthaltenen Unterverzeichnisse, deren Unterverzeichnisse usw. mit erfasst. Als Suchergebnis liefert `find` die Pfadnamen der gefundenen Dateien, die dann an andere Programme weitergeleitet werden können. Zur Verdeutlichung der Befehlsstruktur ein Beispiel:

```
$ find . -user hugo -print
./liste
```

Hier wird also das aktuelle Verzeichnis inklusive aller Unterverzeichnisse nach Dateien durchsucht, die dem Benutzer `hugo` gehören. Die Anweisung `-print` dient dazu, das Suchergebnis, im Beispiel nur eine einzige Datei, auf dem Bildschirm auszugeben. Zur Arbeitserleichterung wird daher die Angabe `-print` automatisch ergänzt, wenn Sie nicht ausdrücklich gesagt haben, wie mit den gefundenen Dateinamen zu verfahren ist.

Wie anhand des Beispiels zu sehen ist, benötigt `find` einige Angaben, um seine Aufgabe ordnungsgemäß erledigen zu können.

Startverzeichnis Die Auswahl des Startverzeichnisses sollte mit Bedacht erfolgen. Wenn Sie das Wurzelverzeichnis wählen, wird die gesuchte Datei – sofern sie denn existiert – mit Sicherheit gefunden werden, der Suchvorgang kann jedoch viel Zeit in Anspruch nehmen. Natürlich dürfen Sie nur in denjenigen Verzeichnissen suchen, auf die Sie angemessene Zugriffsrechte haben.

Die Angabe eines absoluten Pfadnamens für das Startverzeichnis liefert als Suchergebnis absolute Pfadnamen, ein relativ angegebenes Startverzeichnis ergibt entsprechend relative Resultate.

Ausgabe absolut oder relativ?

Statt eines einzelnen Startverzeichnisses können Sie auch eine Liste von Verzeichnisnamen angeben, die dann der Reihe nach durchsucht werden.

Verzeichnisliste

Auswahlkriterien Mit diesen Angaben können Sie die Merkmale der zu findenden Dateien genauer festlegen. Tabelle 6.6 enthält eine Auswahl zulässiger Angaben, weitere stehen in der Dokumentation.

Tabelle 6.6: Testkriterien von `find`

Test	Beschreibung
`-name`	Sucht nach passenden Dateinamen. Hierbei sind alle Sonderzeichen nutzbar, die von der Shell zur Verfügung gestellt werden. Mit `-iname` werden dabei Groß- und Kleinbuchstaben gleich behandelt.
`-user`	Sucht nach Dateien, die dem angegebenen Benutzer gehören. Dabei ist statt des Anwendernamens auch die Angabe der eindeutigen Benutzernummer, der UID, möglich.
`-group`	Sucht nach Dateien, die zu der angegebenen Gruppe gehören. Wie bei `-user` ist hier statt des Gruppennamens auch die Angabe der eindeutigen Gruppennummer, der GID, zulässig.
`-type`	Ermöglicht die Suche nach verschiedenen Dateitypen (siehe Abschnitt 10.2). Dabei werden unterschieden: `b` blockorientierte Gerätedatei `c` zeichenorientierte Gerätedatei `d` Verzeichnis `f` normale Datei `l` Symbolisches Link `p` FIFO (*named pipe*) `s` Unix-Domain-Socket
`-size`	Sucht nach Dateien bestimmter Größe. Zahlenwerte werden dabei als 512-Byte-Blöcke interpretiert, durch ein nachgestelltes `c` sind Größenangaben in Byte, durch `k` in Kibibyte erlaubt. Vorangestellte Plus- oder Minuszeichen entsprechen Unter- und Obergrenzen, womit ein Größenbereich abgedeckt werden kann. Das Kriterium `-size +10k` trifft z. B. für alle Dateien zu, die größer als 10 KiB sind.
`-atime`	(engl. *access*, »Zugriff«) sucht Dateien nach dem Zeitpunkt des letzten Zugriffs. Hier sowie für die beiden nächsten Auswahlkriterien wird der Zeitraum in Tagen angegeben; …`min` statt …`time` ermöglicht eine minutengenaue Suche.
`-mtime`	(engl. *modification*, »Veränderung«) wählt passende Dateien über den Zeitpunkt der letzten Veränderung aus.
`-ctime`	(engl. *change*, »Änderung«) sucht Dateien anhand der letzten Änderung der Inodes (durch Zugriff auf den Inhalt, Rechteänderung, Umbenennen etc.)
`-perm`	Findet nur Dateien, deren Zugriffsrechte genau mit den angegebenen übereinstimmen. Die Festlegung erfolgt mittels einer Oktalzahl, die beim Befehl `chmod` beschrieben wird. Möchte man nur nach einem bestimmten Recht suchen, muss der Oktalzahl ein Minuszeichen vorangestellt werden, z. B. berücksichtigt `-perm -20` alle Dateien, die Gruppenschreibrechte besitzen, unabhängig von deren übrigen Zugriffsrechten.
`-links`	Sucht nach Dateien, deren Referenzzähler den angegebenen Zahlenwert hat.
`-inum`	Findet Verweise auf die Datei mit der angegebenen Inode-Nummer.

mehrere Auswahlkriterien

Wenn Sie mehrere Auswahlkriterien gleichzeitig angeben, werden diese automatisch Und-verknüpft, es müssen also alle erfüllt werden. Daneben versteht `find` noch andere logische Operationen (siehe Tabelle 6.7).

Kompliziertere logische Verknüpfungen von Auswahlkriterien können (bzw. sollten) in runde Klammern eingeschlossen werden, um fehlerhafte Auswertungen zu vermeiden. Die Klammern müssen Sie natürlich vor der Shell verstecken:

```
$ find . \( -type d -o -name "A*" \) -print
```

```
./.
./..
./bilder
./Abfall
$ _
```

Das Beispiel listet alle Dateien auf dem Bildschirm auf, die entweder Verzeichnisse repräsentieren oder deren Name mit einem »A« beginnt oder beides.

Aktionen Die Ausgabe der Suchergebnisse auf dem Bildschirm geschieht, wie eingangs erwähnt, mit der Kommandooption `-print`. Daneben existieren mit `-exec` (engl. *execute*, »ausführen«) und `-ok` noch zwei weitere Optionen, die es ermöglichen, Folgekommandos mit den gefundenen Dateien auszuführen. Hierbei unterscheidet sich `-ok` nur dadurch von `-exec` dass vor der Ausführung des Folgekommandos der Benutzer um Zustimmung gebeten wird; `-exec` setzt diese stillschweigend voraus. Im weiteren Text steht daher `-exec` stellvertretend für beide Varianten.

Kommando ausführen

Zur Anwendung von `-exec` gibt es einige allgemeingültige Regeln:

- Das Kommando hinter `-exec` ist mit einem Strichpunkt (»;«) abzuschließen. Da dieser in üblichen Shells eine Sonderbedeutung hat, muss er maskiert werden (etwa als »\;« oder mit Anführungszeichen), damit `find` ihn überhaupt zu sehen bekommt.
- Zwei geschweifte Klammern (»{}«) werden in dem Kommando durch den gefundenen Dateinamen ersetzt. Am besten setzen Sie die geschweiften Klammern in Anführungszeichen, um Probleme mit Leerzeichen in Dateinamen zu vermeiden.

Zum Beispiel:

```
$ find . -user hugo -exec ls -l '{}' \;
-rw-r--r--  1  hugo  users   4711  Oct 4 11:11 datei.txt
$ _
```

Das obenstehende Beispiel sucht nach allen Dateien innerhalb des aktuellen Verzeichnisses, die dem Anwender `hugo` gehören und führt für diese das Kommando »`ls -l`« aus. Mehr Sinn ergibt da schon folgendes:

```
$ find . -atime +13 -exec rm -i '{}' \;
```

Hiermit werden alle Dateien im aktuellen Verzeichnis und darunter interaktiv gelöscht, auf die seit zwei Wochen oder länger nicht mehr zugegriffen wurde.

Mitunter – etwa im letzten Beispiel – ist es sehr ineffizient, für jeden einzelnen Dateinamen extra einen neuen Prozess mit dem `-exec`-Kommando zu starten. In diesem Fall hilft oft das Kommando `xargs`, das so viele Dateinamen wie möglich sammelt, bevor es damit tatsächlich ein Kommando ausführt:

Tabelle 6.7: Logische Operatoren für `find`

Zeichen	Operator	Bedeutung
`!`	Nicht	die folgende Bedingung darf nicht zutreffen
`-a`	*and* (Und)	die Bedingungen links und rechts vom `-a` müssen zutreffen
`-o`	*or* (Oder)	von den Bedingungen links und rechts vom `-o` muss mindestens eine zutreffen

```
$ find . -atime +13 | xargs -r rm -i
```

`xargs` liest seine Standardeingabe bis zu einem bestimmten (konfigurierbaren) Maximum von Zeichen oder Zeilen und verwendet das Gelesene als Argumente für das angegebene Kommando (hier `rm`). Als Trennzeichen für die Argumente gelten dabei Leerzeichen (die mit Anführungszeichen oder »\« maskiert werden können) oder Zeilentrenner. Das Kommando wird nur so oft wie nötig aufgerufen, um die Eingabe zu verbrauchen. – Die Option »`-r`« von `xargs` sorgt dafür, dass das Kommando `rm` nur ausgeführt wird, wenn `find` wirklich einen Dateinamen schickt – ansonsten würde es zumindest einmal gestartet.

Die `find`/`xargs`-Kombination kommt bei eigenartigen Dateinamen in Schwierigkeiten, etwa solchen, die Leerzeichen oder gar Zeilentrenner enthalten, welche dann als Trennzeichen fehlinterpretiert werden. Die todsichere Abhilfe dagegen besteht darin, die `find`-Option »`-print0`« zu benutzen, die wie `-print` die Namen der gefundenen Dateien ausgibt, diese aber nicht durch Zeilentrenner, sondern durch Nullbytes voneinander trennt. Da das Nullbyte in Dateinamen nicht auftauchen kann, ist keine Verwechslung mehr möglich. `xargs` muss mit der Option »`-0`« aufgerufen werden, um diese Form der Eingabe zu verstehen:

```
$ find . -atime +13 -print0 | xargs -0r rm -i
```

Übungen

6.25 [!2] Finden Sie alle Dateien in Ihrem System, die größer als 1 MiB sind, und lassen Sie deren Namen ausgeben.

6.26 [2] Wie können Sie `find` benutzen, um eine Datei zu löschen, die einen merkwürdigen Namen hat (etwa mit unsichtbaren Kontrollzeichen oder mit Umlauten, die von älteren Shells nicht verstanden werden)?

6.27 [3] (Beim zweiten Durcharbeiten.) Wie würden Sie beim Abmelden dafür sorgen, dass etwaige Dateien in `/tmp`, die Ihnen gehören, automatisch gelöscht werden?

6.4.5 Dateien schnell finden – `locate` und `slocate`

Das Kommando `find` erlaubt die Suche nach Dateien gemäß einer Vielzahl von Kriterien, muss aber die komplette Dateihierarchie unterhalb des Startverzeichnisses ablaufen. Je nach deren Umfang kann es also durchaus eine Weile dauern, bis die Suche abgeschlossen ist. Für einen typischen Anwendungsfall – die Suche nach Dateien mit einem bestimmten Namen – gibt es darum ein beschleunigtes Verfahren.

Das Kommando `locate` gibt alle Dateien aus, deren Namen auf ein bestimmtes Shell-Suchmuster passen. Im einfachsten Fall ist das eine simple Zeichenkette:

```
$ locate brief.txt
/home/hugo/Briefe/brief.txt
/home/hugo/Briefe/omabrief.txt
/home/hugo/Briefe/omabrief.txt~
◁◁◁◁◁
```

Obwohl es sich bei `locate` um einen ziemlich grundlegenden Dienst handelt (wie die Tatsache unterstreicht, dass das Programm zum LPIC-1-Stoff gehört), gehört es nicht bei allen Linux-Systemen zur Standardinstallation.


Wenn Sie zum Beispiel eine SUSE-Distribution verwenden, müssen Sie das Paket `findutils-locate` explizit nachinstallieren, bevor Sie `locate` benutzen können.

Die Sonderzeichen »*«, »?« und »[…]« funktionieren bei `locate` wie bei Suchmustern in der Shell. Während eine Anfrage *ohne* Suchmuster-Sonderzeichen jedoch alle Namen liefert, in denen der Suchbegriff irgendwo auftaucht, gibt eine Anfrage *mit* Suchmuster-Sonderzeichen nur diejenigen Namen aus, die *komplett* – von Anfang bis Ende – vom Suchbegriff beschrieben werden. Aus diesem Grund beginnen Suchmuster-Anfragen mit `locate` meistens mit »*«:

```
$ locate "*/brief.t*"
/home/hugo/Briefe/brief.txt
/home/hugo/Briefe/brief.tab
◁◁◁◁◁
```


Am besten stellen Sie `locate`-Anfragen mit Sonderzeichen in Anführungszeichen, damit die Shell sie nicht zu expandieren versucht.

Der Schrägstrich (»/«) erfährt keine Sonderbehandlung:

```
$ locate Briefe/oma
/home/hugo/Briefe/omabrief.txt
/home/hugo/Briefe/omabrief.txt~
```

`locate` ist so schnell, weil es nicht das Dateisystem absucht, sondern in einer »Datenbank« von Dateinamen nachschaut, die irgendwann vorher mit dem Programm `updatedb` aufgebaut wurde. Das bedeutet natürlich, dass es Dateien nicht erwischt, die seit der letzten Aktualisierung der Datenbank dazu gekommen sind, und umgekehrt die Namen von Dateien liefern kann, die seitdem gelöscht wurden.


Mit der Option »`-e`« können Sie `locate` dazu bringen, nur tatsächlich existierende Dateinamen zu liefern, aber den Geschwindigkeitsvorteil des Programms machen Sie damit zunichte.

Das Programm `updatedb` baut die Datenbank für `locate` auf. Da dieser Vorgang einige Zeit dauern kann, sorgt Ihr Systemverwalter meist dafür, dass das passiert, wenn das System sonst nicht viel zu tun hat, auf ständig eingeschalteten Serversystemen zum Beispiel nachts.


Der dafür nötige Systemdienst `cron` wird in der Unterlage *Linux für Fortgeschrittene* ausführlich besprochen. Für jetzt mag genügen, dass die meisten Linux-Distributionen einen Mechanismus mitliefern, der dafür sorgt, dass `updatedb` regelmäßig aufgerufen wird.

Als Systemverwalter können Sie konfigurieren, welche Verzeichnisse `updatedb` beim Aufstellen der Datenbank beachtet. Wie das im Detail passiert, ist distributionsabhängig: `updatedb` selbst liest keine Konfigurationsdatei, sondern übernimmt seine Konfigurationseinstellungen über Kommandozeilenargumente oder (zum Teil) Umgebungsvariable. Allerdings rufen die meisten Distributionen `updatedb` über ein Shellskript auf, das vorher eine Datei wie `/etc/updatedb.conf` oder `/etc/sysconfig/locate` einliest, in der zum Beispiel geeignete Umgebungsvariable gesetzt werden können.


Dieses Shellskript finden Sie zum Beispiel in `/etc/cron.daily` (Details sind distributionsabhängig).

Sie können `updatedb` zum Beispiel mitteilen, welche Verzeichnisse es durchsuchen und welche es auslassen soll; das Programm erlaubt auch die Definiton von »Netz-Dateisystemen«, die von verschiedenen Rechnern verwendet werden und in deren Wurzelverzeichnissen Datenbanken geführt werden sollen, damit nicht jeder Rechner dafür seine eigene Datenbank aufbauen muss.

Eine wichtige Konfigurationseinstellung ist die des Benutzers, mit dessen Identität `updatedb` läuft. Dafür gibt es im wesentlichen zwei Möglichkeiten:

- `updatedb` läuft mit den Rechten von `root` und kann so jede Datei in die Datenbank aufnehmen. Das heißt aber auch, dass Benutzer Dateinamen in Verzeichnissen ausspähen können, für die sie eigentlich gar keine Zugriffsrechte haben, zum Beispiel die Heimatverzeichnisse anderer Benutzer.
- `updatedb` läuft mit eingeschränkten Rechten, etwa denen des Benutzers `nobody`. In diesem Fall landen nur diejenigen Dateinamen in der Datenbank, die in Verzeichnissen stehen, deren Inhalt `nobody` auflisten darf.

Das Programm `slocate` – eine Alternative zum gewöhnlichen `locate` – umgeht dieses Problem, indem es außer dem Namen einer Datei auch noch Eigentümer, Gruppenzugehörigkeit und Zugriffsrechte in der Datenbank speichert und einen Dateinamen nur dann ausgibt, wenn der Benutzer, der `slocate` aufgerufen hat, tatsächlich Zugriff auf die betreffende Datei hat. Auch `slocate` verfügt über ein `updatedb`-Programm, das allerdings nur ein anderer Name für `slocate` selber ist.

In vielen Fällen ist `slocate` so installiert, dass es auch unter dem Namen `locate` aufgerufen werden kann.

Übungen

6.28 [!1] `README` ist ein sehr populärer Dateiname. Geben Sie die absoluten Pfadnamen aller Dateien auf Ihrem System an, die `README` heißen.

6.29 [2] Legen Sie eine neue Datei in Ihrem Heimatverzeichnis an und überzeugen Sie sich durch einen `locate`-Aufruf, dass diese Datei nicht gefunden wird (wählen Sie gegebenenfalls einen hinreichend ausgefallenen Dateinamen). Rufen Sie dann (mit Administratorrechten) das Programm `updatedb` auf. Wird Ihre neue Datei danach mit `locate` gefunden? Löschen Sie die Datei wieder und wiederholen Sie die vorigen Schritte.

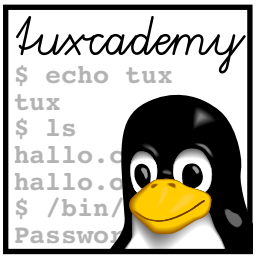
6.30 [1] Überzeugen Sie sich, dass `slocate` funktioniert, indem Sie als gewöhnlicher Benutzer nach Dateien wie `/etc/shadow` suchen.

Kommandos in diesem Kapitel

cd	Wechselt das aktuelle Arbeitsverzeichnis der Shell	bash(1)	81
convmv	Konvertiert Dateinamen zwischen Zeichenkodierungen	convmv(1)	79
cp	Kopiert Dateien	cp(1)	88
find	Sucht nach Dateien, die bestimmte Kriterien erfüllen	find(1), Info: find	96
less	Zeigt Texte (etwa Handbuchseiten) seitenweise an	less(1)	96
ln	Stellt („harte“ oder symbolische) Links her	ln(1)	91
locate	Sucht Dateien über ihren Namen in einer Dateinamensdatenbank	locate(1)	100
ls	Listet Dateien oder den Inhalt von Verzeichnissen auf	ls(1)	82
mkdir	Legt neue Verzeichnisse an	mkdir(1)	84
more	Zeigt Textdaten seitenweise an	more(1)	96
mv	Verschiebt Dateien in andere Verzeichnisse oder benennt sie um	mv(1)	90
pwd	Gibt den Namen des aktuellen Arbeitsverzeichnisses aus	pwd(1), bash(1)	82
rm	Löscht Dateien oder Verzeichnisse	rm(1)	90
rmdir	Entfernt (leere) Verzeichnisse	rmdir(1)	84
slocate	Sucht Dateien über ihren Namen in einer Datenbank und beachtet dabei deren Zugriffsrechte	slocate(1)	102
updatedb	Erstellt die Dateinamensdatenbank für locate	updatedb(1)	101
xargs	Konstruiert Kommandozeilen aus seiner Standardeingabe	xargs(1), Info: find	99

Zusammenfassung

- In Dateinamen dürfen fast alle möglichen Zeichen auftauchen. Im Interesse der Portabilität sollte man sich aber auf Buchstaben, Ziffern und einige Sonderzeichen beschränken.
- Linux unterscheidet Groß- und Kleinschreibung in Dateinamen.
- Absolute Pfade beginnen immer mit einem Schrägstrich und benennen alle Verzeichnisse vom Wurzelverzeichnis des Verzeichnisbaums bis zum betreffenden Verzeichnis bzw. der Datei. Relative Pfade beziehen sich auf das »aktuelle Verzeichnis«.
- Mit cd können Sie das aktuelle Verzeichnis der Shell ändern, mit pwd können Sie seinen Namen anzeigen.
- ls gibt Informationen über Dateien und Verzeichnisse aus.
- Mit mkdir und rmdir können Sie Verzeichnisse anlegen oder entfernen.
- Die Kommandos cp, mv und rm kopieren, verschieben und löschen Dateien und Verzeichnisse.
- Mit ln können Sie »harte« und »symbolische« Links anlegen.
- more und less dienen zum seitenweisen Anzeigen von Dateien auf dem Terminal.
- find sucht nach Dateien oder Verzeichnissen, die bestimmte Kriterien erfüllen.



7

Reguläre Ausdrücke

Inhalt

Lernziele

- Einfache und erweiterte reguläre Ausdrücke verstehen und formulieren können
- Das Kommando `grep` und seine Varianten `fgrep` und `egrep` kennen

Vorkenntnisse

- Grundlegende Kenntnisse über Linux und Linux-Kommandos (etwa aus den vorhergehenden Kapiteln).
- Umgang mit Dateien und Verzeichnissen (siehe Kapitel 6)
- Umgang mit einem Texteditor (siehe Kapitel 3)

7.1 Reguläre Ausdrücke: Die Grundlagen

Viele Linux-Kommandos dienen im weitesten Sinne zur Textbearbeitung – Muster der Form »tue *xyz* für alle Zeilen, die wie folgt aussehen« tauchen immer wieder auf. Ein sehr mächtiges Mittel zur Beschreibung von Textstücken, meist Zeilen in Dateien, sind »reguläre Ausdrücke«[1]. Auf den ersten Blick ähneln reguläre Ausdrücke den Suchmustern der Shell (Abschnitt 6.3), aber sie funktionieren anders und bieten mehr Möglichkeiten.

Reguläre Ausdrücke werden gerne »rekursiv« aus Bausteinen aufgebaut, die selbst als reguläre Ausdrücke gelten. Die einfachsten regulären Ausdrücke sind

Zeichen

Buchstaben, Ziffern und viele andere Zeichen des üblichen Zeichenvorrats, die für sich selber stehen. »`a`« wäre also ein regulärer Ausdruck, der auf das Zeichen »`a`« passt; der reguläre Ausdruck »`abc`« passt auf die Zeichenkette »`abc`«. Ähnlich

Zeichenklassen

wie bei Shell-Suchmustern gibt es die Möglichkeit, Zeichenklassen zu definieren; der reguläre Ausdruck »`[a-e]`« passt also auf genau eines der Zeichen »`a`« bis »`e`«, und »`a[xy]b`« passt entweder auf »`axb`« oder »`ayb`«. Wie bei der Shell können Bereiche gebildet und zusammengefasst werden – »`[A-Za-z]`« passt auf alle Groß- und

Komplement

Kleinbuchstaben (ohne Umlaute) –, nur die Komplementbildung funktioniert etwas anders: »`[^abc]`« passt auf alle Zeichen *außer* »`a`«, »`b`« und »`c`«. (Bei der Shell hieß das »`[!abc]`«.) Der Punkt »`.`« entspricht dem Fragezeichen in Shellsuchmustern, steht also für ein einziges beliebiges Zeichen – ausgenommen davon ist nur der Zeilentrenner »`\n`«: »`a.c`« passt also auf »`abc`«, »`a/c`« und so weiter, aber nicht auf die mehrzeilige Konstruktion

```
a
c
```

Der Grund dafür besteht darin, dass die meisten Programme zeilenorientiert vorgehen und »zeilenübergreifende« Konstruktionen schwieriger zu verarbeiten wären. (Was nicht heißen soll, dass es nicht manchmal nett wäre, das zu können.)

Während Shellsuchmuster immer vom Anfang eines Pfadnamens aus passen müssen, reicht es bei Programmen, die Zeilen aufgrund von regulären Ausdrücken auswählen, meist aus, wenn der reguläre Ausdruck irgendwo in einer Zeile

Anker

passt. Allerdings können Sie diese Freizügigkeit einschränken: Ein regulärer Aus-

Zeilenanfang

druck, der mit dem Zirkumflex (»`^`«) anfängt, passt nur am Zeilenanfang, und ein regulärer Ausdruck, der mit dem Dollarzeichen (»`$`«) aufhört, entsprechend nur

Zeilenende

am Zeilenende. Der Zeilentrenner am Ende jeder Zeile wird ignoriert, so dass Sie »`xyz$`« schreiben können, um alle Zeilen auszuwählen, die auf »`xyz`« enden, und nicht »`xyz\n$`« angeben müssen.

Strenggenommen passen »`^`« und »`$`« auf gedachte »unsichtbare« Zeichen am Zeilenanfang bzw. unmittelbar vor dem Zeilentrenner am Zeilenende.

Schließlich können Sie mit dem Stern (»`*`«) angeben, dass der davorstehen-

Wiederholung

de Ausdruck beliebig oft wiederholt werden kann (auch gar nicht). Der Stern selbst steht nicht für irgendwelche Zeichen in der Eingabe, sondern modifiziert nur das Davorstehende – das Shellsuchmuster »`a*.txt`« entspricht also dem regulären Ausdruck »`^a.*\.txt$`« (denken Sie an die »Verankerung« des Ausdrucks am Anfang und Ende der Eingabe und daran, dass ein einzelner Punkt auf alle Zeichen passt).

Vorrang

Wiederholung hat Vorrang vor Aneinanderreihung; »`ab*`« ist ein einzelnes »`a`« gefolgt von beliebig vielen »`b`« (auch keinem), nicht eine beliebig häufige Wiederholung von »`ab`«.

[1] Der Begriff kommt eigentlich aus der theoretischen Informatik und beschreibt ein Verfahren zur Charakterisierung von Mengen von Zeichenketten, die aus Verkettung von konstanten Elementen, Alternativen von konstanten Elementen und deren möglicherweise unbegrenzter Wiederholung besteht. Routinen zur Erkennung regulärer Ausdrücke sind elementare Bestandteile vieler Programme, etwa Compilern für Programmiersprachen. Reguläre Ausdrücke tauchten in der Geschichte von Unix schon sehr früh auf; die meisten frühen Unix-Entwickler hatten einen Hintergrund in theoretischer Informatik, so dass die Idee ihnen nahelag.

7.2 Reguläre Ausdrücke: Extras

Die Beschreibung aus dem vorigen Abschnitt gilt für praktisch alle Linux-Programme, die reguläre Ausdrücke verarbeiten. Diverse Programme unterstützen aber auch verschiedene Erweiterungen, die entweder Schreibvereinfachungen bieten oder tatsächlich zusätzliche Dinge erlauben. Die »Krone der Schöpfung« markieren hier die modernen Skriptsprachen wie Tcl, Perl oder Python, deren Implementierungen inzwischen weit über das hinausgehen, was reguläre Ausdrücke im Sinne der theoretischen Informatik können.

Erweiterungen

Einige gängige Erweiterungen sind:

Wortklammern Die Sequenz »\<« passt am Anfang eines Worts (soll heißen, an einer Stelle, wo ein Nichtbuchstabe auf einen Buchstaben stößt). Analog passt »\>« am Ende eines Worts (da, wo ein Buchstabe von einem Nichtbuchstaben gefolgt wird).

Gruppierung Runde Klammern (»(…)«) erlauben es, Aneinanderreihungen von regulären Ausdrücken zu wiederholen: »a(bc)*« passt auf ein »a« gefolgt von beliebig vielen Wiederholungen von »bc«, im Gegensatz zu »abc*«, das auf »ab« gefolgt von beliebig vielen Wiederholungen von »c« passt.

Alternative Mit dem vertikalen Balken (»|«) können Sie eine Auswahl zwischen mehreren regulären Ausdrücken treffen: Der reguläre Ausdruck »Affen(brot|schwanz|kletter)baum« passt genau auf eine der Zeichenketten »Affenbrotbaum«[2], »Affenschwanzbaum«[3] oder »Affenkletterbaum«.

Optionales Das Fragezeichen (»?«) macht den vorstehenden regulären Ausdruck optional, das heißt, er tritt entweder einmal auf oder gar nicht. »Boot(smann)?« passt auf »Boot« oder »Bootsmann«.

Mindestens einmal Vorhandenes Das Pluszeichen (»+«) entspricht dem Wiederholungsoperator »*«, bis darauf, dass der vorstehende Ausdruck mindestens einmal auftreten muss, also nicht ganz entfallen darf.

Bestimmte Anzahl von Wiederholungen Sie können in geschweiften Klammern eine Mindest- und eine Maximalanzahl von Wiederholungen angegeben werden: »ab{2,4}« passt auf »abb«, »abbb« und »abbbb«, aber nicht auf »ab« oder »abbbbb«. Sie können sowohl die Mindest- als auch die Maximalanzahl weglassen; fehlt die Mindestanzahl, wird 0, fehlt die Maximalanzahl, so wird »Unendlich« angenommen.

Rückbezug Mit einem Ausdruck der Form »*n*« können Sie eine Wiederholung desjenigen Teils in der Eingabe verlangen, der auf den Klammerausdruck Nr. *n* im regulären Ausdruck gepasst hat. »(ab)\1« zum Beispiel passt auf »abab«, und wenn bei der Bearbeitung von »(ab*a)x\1« die Klammer auf abba gepasst hat, dann passt der gesamte Ausdruck auf abbaxabba (und nichts anderes). Weitere Details finden Sie in der Dokumentation zu GNU grep.

»Bescheidene« Vorgehensweise Die Operatoren »*«, »+« und »?« sind normalerweise »gierig«, das heißt, sie versuchen auf so viel der Eingabe zu passen wie möglich: »^a.*a« bezogen auf die Eingabe »abacada« passt auf »abacada«, nicht »aba« oder »abaca«. Es gibt aber entsprechende »bescheidene« Versionen »*?«, »+?« und »??«, die auf so wenig der Eingabe passen wie möglich. In unserem Beispiel würde »^a.*?a« auf »aba« passen. Auch die geschweiften Klammern haben möglicherweise eine bescheidene Version.

Nicht jedes Programm unterstützt jede Erweiterung. Tabelle 7.1 zeigt eine Übersicht der wichtigsten Programme. Insbesondere Emacs, Perl und Tcl unterstützen noch jede Menge hier nicht diskutierte Erweiterungen.

[2] *Adansonia digitata*

[3] Volkstümlicher Name für die Araukarie (*Araucaria araucana*)

Tabelle 7.1: Unterstützung von regulären Ausdrücken

Erweiterung	GNU `grep`	GNU `egrep`	trad. `egrep`	`vim`	`emacs`	Perl	Tcl
Wortklammern	•	•	•	•1	•1	•4	•4
Gruppierung	•1	•	•	•1	•1	•	•
Alternative	•1	•	•	•2	•1	•	•
Optionales	•1	•	•	•3	•	•	•
mind. einmal	•1	•	•	•1	•	•	•
Anzahl	•1	•	◦	•1	•1	•	•
Rückbezug	◦	•	•	◦	•	•	•
Bescheiden	◦	◦	◦	•4	•	•	•

•: wird unterstützt; ◦: wird nicht unterstützt
Anmerkungen: 1. Wird durch einen vorgesetzten Rückstrich (»\«) angesprochen, also z. B. »`ab\+`« statt »`ab+`«. 2. Braucht keine Klammern; Alternativen beziehen sich immer auf den kompletten Ausdruck. 3. Benutzt »`\=`« statt »`?`«. 4. Ganz andere Syntax (siehe Dokumentation).

Tabelle 7.2: Optionen für `grep` (Auswahl)

Option		Wirkung
`-c`	(*count*)	Liefert nur die Anzahl der passenden Zeilen
`-i`	(*ignore*)	Klein- und Großbuchstaben werden nicht unterschieden
`-l`	(*list*)	Statt kompletter Zeilen werden nur Dateinamen ausgegeben
`-n`	(*number*)	Die Zeilennummer der gefundenen Zeilen in der Eingabe wird mit ausgegeben
`-r`	(*recursive*)	Auch Dateien in Unterverzeichnissen usw. mit durchsuchen
`-v`	(*invert*)	Nur Zeilen, die das Muster *nicht* enthalten, werden ausgegeben

7.3 Dateien nach Textteilen durchsuchen – `grep`

Vielleicht eines der wichtigsten Linux-Programme, die reguläre Ausdrücke benutzen, ist `grep`. Es durchsucht eine oder mehrere Dateien nach Zeilen, auf die ein angegebener regulärer Ausdruck passt. Passende Zeilen werden ausgegeben, nicht passende verworfen.

`grep`-Versionen

Von `grep` existieren zwei Ableger: Traditionell erlaubt das abgespeckte `fgrep` (engl. *fixed*, »fest vorgegeben«) keine regulären Ausdrücke, sondern nur feste Zeichenketten, ist dafür aber sehr flott. Zusätzliche Möglichkeiten bietet `egrep` (engl. *extended*, »erweitert«), arbeitet deshalb jedoch recht langsam und benötigt viel Speicherkapazität.

Früher haben diese Anmerkungen tatsächlich im Großen und Ganzen gestimmt. Insbesondere verwendeten `grep` und `egrep` völlig verschiedene Verfahren zur Auswertung der regulären Ausdrücke, die je nach deren Gestalt und Umfang sowie der Größe der Eingabe zu ganz unterschiedlichen Laufzeitergebnissen führen konnten. Bei der unter Linux üblichen `grep`-Implementierung aus dem GNU-Projekt sind alle drei Varianten in Wirklichkeit dasselbe Programm und unterscheiden sich vor allem in der Syntax dessen, wonach gesucht werden soll.

Syntax

Die Syntax von `grep` erfordert mindestens die Angabe des regulären Ausdrucks, nach dem gesucht werden soll. Darauf folgt der Name der Textdatei (oder die Namen der Textdateien), die nach diesem Ausdruck durchsucht werden soll(en). Wenn kein Dateiname angegeben wurde, bezieht sich `grep` auf die Standard-Eingabe (siehe Kapitel 8).

regulärer Ausdruck

Der reguläre Ausdruck, nach dem die Eingabe durchsucht wird, darf neben den grundlegenden Bausteinen aus Abschnitt 7.1 auch die meisten Erweiterungen aus Abschnitt 7.2 enthalten. Die Operatoren »`\+`«, »`\?`« und »`\{`« müssen Sie bei

grep jedoch mit dem Rückstrich versehen. (Bei egrep können Sie darauf verzichten.) »Bescheidene« Operatoren gibt es leider nicht.

Damit die Sonderzeichen nicht bereits von der Shell interpretiert, sondern korrekt an grep übergeben werden, sollten Sie den regulären Ausdruck in Anführungszeichen setzen, jedenfalls wenn er komplizierter ist als eine einfache Zeichenkette und insbesondere wenn er einem Shellsuchmuster ähnelt.

Außer dem regulären Ausdruck und allfälligen Dateinamen können wie gewohnt auch verschiedene Optionen auf der Kommandozeile angegeben werden (s. Tabelle 7.2).

Regulärer Ausdruck in Datei

Mit der Option -f (engl. *file*, »Datei«) können reguläre Ausdrücke aus einer Datei gelesen werden. Wenn diese Musterdatei mehrere Zeilen enthält, wird jeweils der Inhalt einer kompletten Zeile als einzelner Ausdruck angesehen. Dies bringt bei häufig benutzten Suchoperationen eine deutliche Arbeitserleichterung mit sich.

Wie erwähnt erlaubt fgrep keine regulären Ausdrücke, sondern nur konstante Zeichenketten als Suchobjekte. egrep hingegen macht die meisten Erweiterungen für reguläre Ausdrücke bequemer verfügbar (Tabelle 7.1).

Beispiele

Zum Abschluss noch ein paar Beispiele zur Anwendung von grep. Die Datei frosch.txt enthält das Märchen vom Froschkönig (siehe Kapitel B). Alle Zeilen, die die Zeichenkette Frosch enthalten, finden Sie leicht wie folgt:

```
$ grep Frosch frosch.txt
Der Froschkönig oder der eiserne Heinrich
Sie sah sich um, woher die Stimme käme, da erblickte sie einen Frosch,
»Sei still und weine nicht«, antwortete der Frosch, »ich kann wohl Rat
»Was du haben willst, lieber Frosch«, sagte sie, »meine Kleider, meine
◁◁◁◁◁
```

Um die Zeilen zu finden, die genau das Wort »Frosch« enthalten (und nicht irgendwelche Zusammensetzungen wie »Froschkönig«), brauchen Sie die Wortklammer-Erweiterung:

```
$ grep '\<Frosch\>' frosch.txt
Sie sah sich um, woher die Stimme käme, da erblickte sie einen Frosch,
»Sei still und weine nicht«, antwortete der Frosch, »ich kann wohl Rat
»Was du haben willst, lieber Frosch«, sagte sie, »meine Kleider, meine
Der Frosch antwortete: »Deine Kleider, deine Perlen und Edelsteine und
◁◁◁◁◁
```

Es ist auch einfach, alle Zeilen zu finden, die mit »Frosch« *anfangen*:

```
$ grep ^Frosch frosch.txt
Frosch mag schwätzen, was er will, der sitzt doch im Wasser bei
Frosch.«
Frosch verwandelt worden war, daß er drei eiserne Bänder um sein Herz
```

Ein anderes Beispiel: In /usr/share/dict/words steht eine Liste englischer Wörter (gerne als »Wörterbuch« bezeichnet)[4]. Wir interessieren uns für alle Wörter mit drei oder mehr »a«:

```
$ grep -n 'a.*a.*a' /usr/share/dict/words
8:aardvark
21:abaca
22:abacate
```

[4] Je nach Distribution kann der Inhalt des Wörterbuchs mehr oder weniger umfangreich ausfallen.

```
◁◁◁◁◁                                                    … 7030 andere Wörter …
234831:zygomaticoauricularis
234832:zygomaticofacial
234834:zygomaticomaxillary
```

(in der Reihenfolge: das Erdferkel (*Orycteropus afer*), eine faserspendende Bananenpflanze (*Musa textilis*), der brasilianische Name für die Avocado (*Persea sp.*), ein Muskel im Gesicht und zwei Adjektive aus demselben – medizinischen – Umfeld.)

Bei komplizierteren regulären Ausdrücken kann es schnell unübersichtlich werden, warum `grep` eine Zeile ausgibt und eine andere nicht. Etwas Abhilfe kann hier die Option `--color` schaffen, die die Treffer in einer Zeile farblich hervorhebt:

```
$ grep --color root /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

Durch `export GREP_OPTIONS='--color=auto'` (in `~/.profile` o. Ä.) wird die Option dauerhaft aktiviert; der Zusatz `auto` bewirkt dabei, dass die Farbe unterdrückt wird, sobald die Ausgabe in eine Pipeline oder Datei umgeleitet wird.

Übungen

7.1 [2] Sind die Operatoren `?` und `+` in regulären Ausdrücken wirklich nötig?

7.2 [!1] Finden Sie in `frosch.txt` alle Zeilen, in denen das Wort »Tochter« oder »Königstochter« vorkommt.

7.3 [!2] In der Datei `/etc/passwd` stehen die Benutzer des Rechners (meistens jedenfalls). Jede Zeile der Datei besteht aus einer Reihe von durch Doppelpunkten getrennten Feldern. Das letzte Feld jeder Zeile gibt die Login-Shell eines Benutzers an. Geben Sie eine `grep`-Kommandozeile an, mit der Sie alle Benutzer finden können, die die Bash als Login-Shell verwenden.

7.4 [3] Suchen Sie in `/usr/share/dict/words` nach allen Wörtern, die die genau die fünf Vokale »`a`«, »`e`«, »`i`«, »`o`« und »`u`« in dieser Reihenfolge enthalten (möglicherweise mit Konsonanten davor, dazwischen und dahinter).

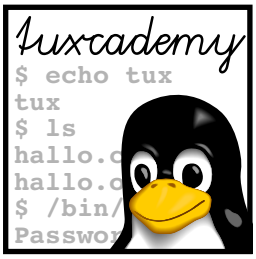
7.5 [4] Geben Sie ein Kommando an, das im »Froschkönig« alle Zeilen sucht und ausgibt, in denen irgendein mindestens vierbuchstabiges Wort zweimal auftritt.

Kommandos in diesem Kapitel

`egrep`	Sucht in Dateien nach Zeilen mit bestimmtem Inhalt, erweiterte reguläre Ausdrücke erlaubt	`grep`(1)	108
`fgrep`	Sucht in Dateien nach Zeilen bestimmten Inhalts, keine regulären Ausdrücke erlaubt	`fgrep`(1)	108
`grep`	Sucht in Dateien nach Zeilen mit bestimmtem Inhalt	`grep`(1)	107

Zusammenfassung

- Reguläre Ausdrücke sind eine mächtige Methode zur Beschreibung von ganzen Gruppen von Zeichenketten.
- `grep` und seine Verwandten durchsuchen Dateiinhalte nach Zeilen, die auf reguläre Ausdrücke passen.



8

Standardkanäle und Filterkommandos

Inhalt

Lernziele

- Die Ein- und Ausgabeumlenkung der Shell beherrschen
- Die wichtigsten Filterkommandos kennen

Vorkenntnisse

- Arbeit mit der Shell
- Umgang mit einem Texteditor (Kapitel 3)
- Umgang mit Dateien und Verzeichnissen (Kapitel 6)

Tabelle 8.1: Standardkanäle unter Linux

Kanal	Bezeichnung	Kürzel	Gerät	Zweck
0	Standard-Eingabe	`stdin`	Tastatur	Programme erhalten Eingaben
1	Standard-Ausgabe	`stdout`	Bildschirm	Programme senden Ausgaben
2	Standard-Fehlerausgabe	`stderr`	Bildschirm	Programme senden Fehlermeldungen

8.1 Ein-/Ausgabeumlenkung und Kommandopipelines

8.1.1 Die Standardkanäle

Viele Linux-Kommandos – beispielsweise `grep` & Co. aus Kapitel 7 – sind so gebaut, dass sie Eingabedaten lesen, diese irgendwie manipulieren und das Ergebnis dieser Manipulationen wieder ausgeben. Wenn Sie zum Beispiel einfach

```
$ grep xyz
```

eingeben, dann können Sie anschließend Textzeilen auf der Tastatur tippen, und `grep` wird nur diejenigen durchlassen, die die Zeichenfolge »xyz« enthalten:

```
$ grep xyz
abc def
xyz 123
xyz 123
aaa bbb
YYYxyzZZZ
YYYxyzZZZ
Strg + d
```

(Die Tastenkombination am Schluss läßt `grep` wissen, dass die Eingabe zu Ende ist.)

Standard-Eingabe
Standard-Ausgabe
Standard-Fehlerausgabe

Man sagt, `grep` liest Daten von der »Standard-Eingabe« – hier der Tastatur – und schreibt sie auf die »Standard-Ausgabe« – hier den Konsolenbildschirm oder, wahrscheinlicher, ein Terminalprogramm in einer grafischen Arbeitsumgebung. Als dritten dieser »Standardkanäle« gibt es noch die »Standard-Fehlerausgabe«; während auf die Standard-Ausgabe die »Nutzdaten« geschrieben werden, die `grep` produziert, landen auf dieser allfällige Fehlermeldungen (etwa weil eine Eingabedatei nicht existiert oder der reguläre Ausdruck einen Syntaxfehler hat).

In diesem Kapitel werden Sie lernen, wie Sie die Standard-Ausgabe eines Programms zum Beispiel in eine Datei umlenken oder die Standard-Eingabe einer Datei entnehmen können. Noch wichtiger werden Sie lernen, wie Sie die Ausgabe eines Programms direkt (ohne Umweg über eine Datei) als Eingabe an ein anderes Programm verfüttern können. Dies öffnet Ihnen die Tür dazu, die für sich genommen alle recht simplen Linux-Kommandos im Baukastenprinzip zu Anwendungen zu verketten, die sehr komplexe Dinge tun können.

Ganz erschöpfend werden wir dieses Thema in diesem Kapitel nicht behandeln können. Freuen Sie sich auf die Linup-Front-Schulungsunterlage *Linux für Fortgeschrittene*, wo die Erstellung von Shellskripten mit den Kommandos des Linux-Baukastens eine sehr wichtige Rolle spielt! Aber hier lernen Sie die sehr wichtigen Grundlagen dafür, schon auf der Kommandozeile Linux-Kommandos geschickt zu kombinieren.

Standardkanäle

Die **Standardkanäle** werden in Bild 8.1 noch einmal zusammengefasst. Sie werden im Jargon meist nur mit den englischen Kürzeln benannt – `stdin`, `stdout` und `stderr` für die Standard-Eingabe, Standard-Ausgabe und Standard-Fehlerausgabe.

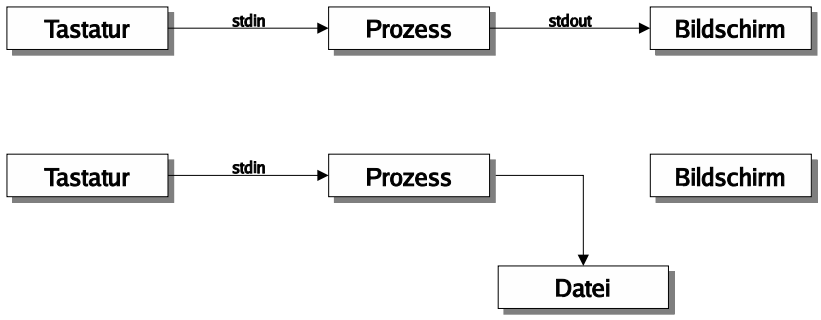


Bild 8.1: Standardkanäle unter Linux

Diesen Kanälen sind respektive auch die Nummern 0, 1 und 2 zugeordnet, was wir gleich noch brauchen werden.

Standardkanäle umleiten

Die Shell kann für einzelne Kommandos diese Standardkanäle auf andere Ziele umleiten, ohne dass die betroffenen Programme davon etwas mitbekommen. Diese benutzen immer die Standardkanäle, lediglich gelangen etwa die Ausgabedaten gegebenenfalls nicht mehr auf den Bildschirm bzw. ins Terminal-Fenster, sondern in eine beliebige andere Datei. Diese kann ein anderes Gerät sein, etwa der Drucker – es ist aber auch möglich, zum Beispiel eine Textdatei anzugeben, in der die Ausgabedaten abgelegt werden. Diese muss nicht einmal vorher existieren, sondern wird bei Bedarf neu erzeugt.

Auf die gleiche Art und Weise können Sie auch den Standard-Eingabe-Kanal umleiten. Ein Programm erhält seine Informationen dann nicht von der Tastatur, sondern entnimmt sie der angegebenen Datei, die wiederum für ein Gerät stehen oder eine Datei im eigentlichen Sinne sein kann.

Tastatur und Bildschirm des »Terminals«, an dem Sie arbeiten (egal ob die Textkonsole eines Linux-Rechners, ein »echtes« seriell angeschlossenes Terminal, ein Terminalfenster in einer grafischen Umgebung oder eine Sitzung über das Netz etwa mit der Secure Shell), können Sie über die »Datei« `/dev/tty` ansprechen – wenn Sie Daten lesen wollen, meint dies die Tastatur, bei der Ausgabe den Bildschirm (umgekehrt wäre ziemlich albern). Der Aufruf

```
$ grep xyz /dev/tty
```

wäre zum Beispiel äquivalent zu unserem Beispiel weiter oben in diesem Abschnitt. Mehr über solche »besonderen Dateien« erzählt Kapitel 10.)

8.1.2 Standardkanäle umleiten

Standard-Ausgabe umleiten

Den Standard-Ausgabe-Kanal können Sie mit dem Operator »>«, also dem »Größer-Als«-Zeichen, umleiten. So wird im folgenden Beispiel die Ausgabe von »`ls -laF`« in eine Datei namens `inhalt` umgelenkt; auf dem Bildschirm erscheint dabei lediglich

```
$ ls -laF >inhalt
$ _
```

Falls die Datei `inhalt` noch nicht existiert, wird sie neu angelegt. Sollte hingegen bereits eine Datei dieses Namens vorhanden sein, wird deren Inhalt überschrieben. Das ganze arrangiert die Shell, noch bevor das gewünschte Programm überhaupt gestartet wird – die Ausgabedatei wird also auch dann angelegt, wenn der eigentliche Programmaufruf falsch eingetippt wurde oder das Programm überhaupt keine Ausgabe liefert (die Datei `inhalt` ist dann anschließend leer).

Existierende Dateien schützen

Wenn Sie verhindern wollen, dass die Shell-Ausgabeumlenkung schon existierende Dateien leert, können Sie in der Bash das Kommando »`set -o noclobber`« geben. In diesem Fall bleibt eine schon existierende Datei, die Ziel einer Ausgabeumlenkung ist, unverändert. Statt dessen erscheint eine Fehlermeldung.

Die Textdatei `inhalt` können Sie nun wie üblich anschauen, z. B. mit `less`:

```
$ less inhalt
total 7
drwxr-xr-x  12 hugo   users     1024 Aug 26 18:55 ./
drwxr-xr-x   5 root   root      1024 Aug 13 12:52 ../
drwxr-xr-x   3 hugo   users     1024 Aug 20 12:30 fotos/
-rw-r--r--   1 hugo   users        0 Sep  6 13:50 inhalt
-rw-r--r--   1 hugo   users    15811 Aug 13 12:33 pingu.gif
-rw-r--r--   1 hugo   users    14373 Aug 13 12:33 hobby.txt
-rw-r--r--   2 hugo   users     3316 Aug 20 15:14 chemie.txt
```

Wenn Sie den Inhalt von `inhalt` genau betrachten, sehen Sie einen Verzeichniseintrag für `inhalt` mit der Dateigröße 0. Das liegt an der Arbeitsweise der Shell: Bei der Bearbeitung der Kommandozeile wird zunächst die Ausgabeumlenkung erkannt und eine neue Datei `inhalt` angelegt bzw. deren Inhalt gelöscht. Danach führt die Shell das Kommando, hier `ls`, aus, wobei sie die Standardausgabe von `ls` mit der Datei `inhalt` statt dem Bildschirm verbindet.

Die Datei hat in der `ls`-Ausgabe die Länge 0, weil das `ls`-Kommando die Dateiinformationen für `inhalt` abgerufen hat, bevor tatsächlich etwas in die Datei geschrieben wurde – obwohl vor dem betreffenden Eintrag eigentlich drei andere Zeilen stehen! Das liegt daran, dass `ls` erst sämtliche Verzeichniseinträge liest, sie alphabetisch sortiert und erst dann die Ausgabe zu schreiben beginnt. `ls` sieht also die von der Shell neu angelegte oder gerade geleerte Datei `inhalt` ohne Inhalt.

Standardausgabe an Datei anhängen

Wenn Sie die Ausgabe eines Programms ans Ende einer bestehenden Datei anhängen wollen, ohne dass deren bisheriger Inhalt ersetzt wird, können Sie den Operator `>>` benutzen. Wenn diese Datei noch nicht existiert, wird sie auch hier neu angelegt:

```
$ date >> inhalt
$ less inhalt
total 7
drwxr-xr-x  12 hugo   users     1024 Aug 26 18:55 ./
drwxr-xr-x   5 root   root      1024 Aug 13 12:52 ../
drwxr-xr-x   3 hugo   users     1024 Aug 20 12:30 fotos/
-rw-r--r--   1 hugo   users        0 Sep  6 13:50 inhalt
-rw-r--r--   1 hugo   users    15811 Aug 13 12:33 pingu.gif
-rw-r--r--   1 hugo   users    14373 Aug 13 12:33 hobby.txt
-rw-r--r--   2 hugo   users     3316 Aug 20 15:14 chemie.txt
Wed Oct 22 12:31:29 CEST 2003
```

Im Beispiel wurde das aktuelle Datum ans Ende der Datei `inhalt` angefügt.

Kommandosubstitution

Eine andere Möglichkeit zur Umleitung der Standardausgabe eines Programms bieten die »verkehrten« Anführungszeichen `` `…` ``. Man spricht auch von **Kommandosubstitution**: Die Standardausgabe eines Kommandos, das in verkehrten Anführungszeichen steht, wird anstelle des Kommandoaufrufs in die Befehlszeile eingebaut; ausgeführt wird dann das, was sich durch diese Ersetzung ergibt. Zum Beispiel:

```
$ cat termine                                    Unser kleiner Terminkalender
22.12. Geschenke besorgen
```

```
23.12. Christbaum besorgen
24.12. Heiligabend
$ date +%d.%m.                                    Welches Datum haben wir?
23.12.
$ grep ^`date +%d.%m.` termine                    Was liegt an?
23.12. Christbaum besorgen
```

Eine unter Umständen bequemere, aber nur von modernen Shells wie der Bash unterstützte Syntax für »`date`« ist »$(date)«. Damit ist es einfacher möglich, solche Aufrufe zu verschachteln.

Mit <, dem »Kleiner-Als«-Zeichen, können Sie den Standard-Eingabe-Kanal umleiten. Nun wird statt der Tastatureingabe der Inhalt der angegebenen Datei ausgewertet:

Standard-Eingabe umleiten

```
$ wc -w <frosch.txt
1255
```

Im Beispiel zählt das Filterkommando wc die in der Datei frosch.txt vorkommenden Wörter.

Einen <<-Umleitungsoperator zur Verkettung mehrerer Eingabedateien gibt es nicht; um den Inhalt mehrerer Dateien als Eingabe an ein Kommando zu leiten, müssen Sie cat benutzen:

```
$ cat datei1 datei2 datei3 | wc -w
```

(Zum »|«-Operator steht mehr im nächsten Abschnitt.) Die meisten Programme akzeptieren allerdings einen oder mehrere Dateinamen als Argumente auf der Kommandozeile.

Selbstverständlich ist auch die kombinierte Umleitung von Ein- und Ausgabekanal zulässig. Die Ausgabe des Wortzähl-Beispiels wird hier in eine Datei namens wortzahl geschrieben:

kombinierte Umleitung

```
$ wc -w <frosch.txt >wortzahl
$ cat wortzahl
1255
```

Neben den Standard-Eingabe- und -Ausgabe-Kanälen existiert noch der Standard-Fehler-Kanal. Sollte ein Programm während seiner Arbeit auf Probleme stoßen, werden die zugehörigen Fehlermeldungen auf diesem Kanal ausgegeben. So bekommen Sie sie zu sehen, auch wenn die Standardausgabe in eine Datei umgeleitet wird. Wenn Sie auch die Standardfehlerausgabe in eine Datei umleiten wollen, müssen Sie beim Umleitungsoperator die Kanalnummer angeben – bei stdin (0<) und stdout (1>) ist diese optional, für stderr (2>) jedoch zwingend erforderlich.

Standard-Fehler-Kanal

Mit dem Operator >& können Sie einen Kanal in einen anderen umleiten:

```
make >make.log 2>&1
```

leitet die Standard-Ausgabe *und* die Standard-Fehlerausgabe des Kommandos make in die Datei make.log.

Passen Sie auf: Hier kommt es auf die Reihenfolge an! Die beiden Kommandos

```
make >make.log 2>&1
make 2>&1 >make.log
```

führen zu sehr unterschiedlichen Resultaten. Im zweiten Fall wird erst die Standard-Fehlerausgabe dorthin geleitet, wohin die Standard-Ausgabe geht (`/dev/tty`, wo sie sowieso schon hingeht) und anschließend die Standard-Ausgabe in die Datei `make.log` geschickt, was an dem Ziel der Standard-Fehlerausgabe aber nichts mehr ändert.

Übungen

8.1 [2] Mit der Option `-U` können Sie `ls` dazu bringen, die Verzeichniseinträge unsortiert auszugeben. Trotzdem hat nach dem Kommando »`ls -laU >inhalt`« der Eintrag für `inhalt` in der Ausgabedatei immer noch die Länge Null. Woran könnte das liegen?

8.2 [!2] Vergleichen Sie die Ausgabe der Kommandos »`ls /tmp`« und »`ls /tmp >ls-tmp.txt`« (wobei wir mit »Ausgabe« im zweiten Fall den Inhalt der Datei `ls-tmp.txt` meinen). Fällt Ihnen etwas auf? Falls ja, wie könnte man das Phänomen erklären?

8.3 [!2] Warum ist es nicht möglich, eine Datei in einem Schritt durch eine neue Version zu ersetzen, etwa mit »`grep xyz datei >datei`«?

8.4 [!1] Und was ist das Problem mit »`cat bla >>bla`« (eine nichtleere Datei `bla` vorausgesetzt)?

8.5 [2] Wie würden Sie in der Shell eine Meldung so ausgeben, dass sie auf der Standard-Fehlerausgabe landet?

8.1.3 Kommando-Pipelines

Oft dient die Ausgabeumleitung nur dazu, das Ergebnis eines Programms abzuspeichern, um es dann mit einem anderen Kommando weiterzubearbeiten. Diese Art von Zwischenspeicherung ist jedoch nicht nur recht mühsam, sondern Sie müssen auch noch daran denken, die nicht mehr benötigten Dateien wieder zu löschen. Linux bietet daher eine direkte Verknüpfung von Kommandos durch **Pipes** an: Die Ausgabe eines Programms wird automatisch zur Eingabe des nächsten Programms.

Pipes

direkte Verknüpfung mehrerer Befehle

Pipeline

Die direkte Verknüpfung mehrerer Befehle zu einer solchen Kommando-***Pipeline*** (»Rohrleitung«) erfolgt mit dem Zeichen `|`. Statt also wie im ersten Beispiel dieses Kapitels zunächst den Verzeichnisinhalt mit dem Kommando »`ls -laF`« in eine Datei `inhalt` umzulenken und diese dann mit `less` anzusehen, können Sie diesen Vorgang auch in einem Schritt ohne Zwischenspeicherung in einer Datei ausführen:

```
$ ls -laF | less
total 7
drwxr-xr-x  12 hugo   users     1024 Aug 26 18:55 ./
drwxr-xr-x   5 root   root      1024 Aug 13 12:52 ../
drwxr-xr-x   3 hugo   users     1024 Aug 20 12:30 fotos/
-rw-r--r--   1 hugo   users      449 Sep  6 13:50 inhalt
-rw-r--r--   1 hugo   users    15811 Aug 13 12:33 pingu.gif
-rw-r--r--   1 hugo   users    14373 Aug 13 12:33 hobby.txt
-rw-r--r--   2 hugo   users     3316 Aug 20 15:14 chemie.txt
```

Derartige Befehlsfolgen können praktisch beliebig lang werden, auch ist eine abschließende Ausgabeumleitung in eine Datei möglich:

```
$ cut -d: -f1 /etc/passwd | sort | pr -2 >userlst
```

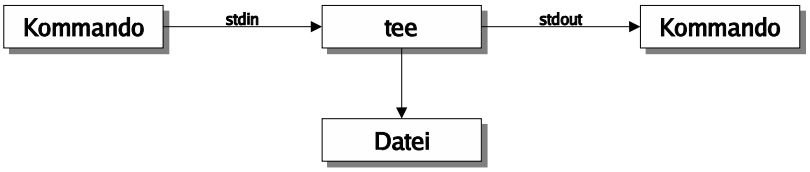


Bild 8.2: Das Kommando `tee`

Diese Kommando-Pipeline entnimmt zunächst der Systemdatei `/etc/passwd` alle Benutzernamen (die in der ersten durch »:« getrennten Spalte stehen), sortiert diese alphabetisch und schreibt sie dann zweispaltig in die Datei `userlst`. Die hier benutzten Kommandos werden übrigens im Rest dieses Kapitels genauer beschrieben.

Manchmal ist es sinnvoll, den Datenstrom innerhalb einer Kommando-Pipeline an einer bestimmten Stelle abzuspeichern, etwa weil das dortige Zwischenergebnis auch für andere Arbeiten von Interesse ist. Der Befehl `tee` führt eine Verzweigung des Datenstroms herbei, indem dieser verdoppelt und je ein Strom in eine Datei sowie an das nächste Kommando geleitet wird. Der Name dieses Kommandos (lautmalerisch für den Buchstaben »T«) lässt sich leicht aus Bild 8.2 herleiten – oder Sie denken an ein »T-Stück« in der »Pipeline« …

Zwischenergebnis abspeichern

Die Anweisung `tee` ohne Parameter legt die gewünschte Datei neu an oder überschreibt eine vorhandene; mit `-a` (engl. *append*, »anhängen«) lässt sich die Ausgabe an das Ende einer bestehenden Datei anfügen.

```
$ ls -laF | tee liste | less
total 7
drwxr-xr-x  12 hugo   users     1024 Aug 26 18:55 ./
drwxr-xr-x   5 root   root      1024 Aug 13 12:52 ../
drwxr-xr-x   3 hugo   users     1024 Aug 20 12:30 fotos/
-rw-r--r--   1 hugo   users      449 Sep  6 13:50 inhalt
-rw-r--r--   1 hugo   users    15811 Aug 13 12:33 pingu.gif
-rw-r--r--   1 hugo   users    14373 Aug 13 12:33 hobby.txt
-rw-r--r--   2 hugo   users     3316 Aug 20 15:14 chemie.txt
```

In diesem Beispiel wird der Inhalt des aktuellen Verzeichnisses sowohl in die Datei `liste` geschrieben als auch auf dem Bildschirm ausgegeben. (Die Datei `liste` ist noch nicht in der Ausgabe von `ls` zu sehen, da sie erst später von `tee` angelegt wird – ein Unterschied zur Ausgabeumlenkung der Shell.)

Übungen

8.6 [!2] Wie würden Sie dasselbe Zwischenergebnis gleichzeitig in mehrere Dateien schreiben?

8.2 Filterkommandos

Eine der Grundlagen von Unix – und damit Linux – ist das »Werkzeugkastenprinzip«. Das System verfügt über eine große Menge von Systemprogrammen,

Werkzeugkastenprinzip

Tabelle 8.2: Optionen für `cat` (Auswahl)

Option	Wirkung
`-b`	(engl. *number non-blank lines*) Numeriert alle nichtleeren Zeilen der Ausgabe, beginnend mit 1.
`-E`	(engl. *end-of-line*) Zeigt am Ende jeder Zeile der Ausgabe ein `$` an (gut zum Aufspüren von ansonsten nicht sichtbaren Leerzeichen).
`-n`	(engl. *number*) Numeriert alle Zeilen der Ausgabe, beginnend mit 1.
`-s`	(engl. *squeeze*) Ersetzt Folgen von Leerzeilen durch je eine.
`-T`	(engl. *tabs*) Stellt Tabulatorzeichen als »`^I`« dar.
`-v`	(engl. *visible*) Macht Steuerzeichen c als »`^`c« und Zeichen α mit Zeichencodes größer als 127 als »`M-`α« sichtbar.
`-A`	(engl. *show all*) Äquivalent zu `-vET`.

die jeweils eine (konzeptuell) simple Aufgabe erledigen. Diese Programme können dann von anderen Programmen als »Bausteine« verwendet werden und ersparen es den Autoren jener Programme, die entsprechende Funktionalität selbst zu entwickeln. So hat z. B. nicht jedes Programm eine eigene Sortierfunktion, sondern viele Programme greifen auf das Kommando `sort` zurück. Dieser modulare Aufbau hat mehrere Vorteile:

- Eine Vereinfachung für die Programmierer, die nicht ständig neue Sortierroutinen schreiben oder zumindest in ihre Programme einbauen müssen.
- Bei einer Fehlerkorrektur oder Optimierung von `sort` profitieren alle Programme, die darauf zugreifen, ebenfalls, und das ohne explizite Anpassung (meistens jedenfalls).

Filterkommandos

Programme, die ihre Eingabe von der Standard-Eingabe lesen und ihre Ausgabe auf die Standard-Ausgabe schreiben, heißen **Filterkommandos** oder kurz »Filter«. Ohne Eingabeumleitung lesen Filter also von der Tastatur. Zum Beenden der Standard-Eingabe eines solchen Kommandos müssen Sie die Tastenkombination Strg+d eingeben, die der Terminaltreiber als »Datei-Ende« interpretiert.

Das gilt wohlgemerkt *nur* für die Eingabe von Inhalten über die Tastatur. Textdateien auf der Platte dürfen natürlich das Strg+d-Zeichen (ASCII 4) enthalten, ohne dass das System glaubt, die Datei wäre an dieser Stelle zu Ende. Dies im Gegensatz zu einem gewissen anderen sehr populären Betriebssystem, das traditionell etwas eigenwillige Vorstellungen von der Bedeutung des Zeichens Strg+z (ASCII 26) hat, selbst wenn es in einer Textdatei steht …

Auch viele »gewöhnliche« Kommandos, zum Beispiel das schon gezeigte `grep`, verhalten sich wie Filter, wenn Sie keine Dateinamen zur Bearbeitung angeben.

Eine Auswahl der wichtigsten dieser Befehle besprechen wir im Rest des Kapitels. Einige haben sich dabei eingeschlichen, die keine waschechten Filter-Kommandos sind, aber für alle gilt, dass sie wichtige Bausteine von Pipelines bilden.

8.3 Dateien lesen und ausgeben

8.3.1 Textdateien ausgeben und aneinanderhängen – `cat`

Dateien verketten

Der Befehl `cat` (engl. *concatenate*, »verketten«) dient eigentlich dazu, mehrere auf der Kommandozeile benannte Dateien zu einer einzigen zusammenzufügen. Übergeben Sie jedoch nur einen Dateinamen als Argument, wird der Inhalt der

betreffenden Datei auf der Standardausgabe ausgegeben. Wenn Sie überhaupt keinen Dateinamen übergeben, liest `cat` seine Standardeingabe – dies scheint nutzlos, aber `cat` bietet über Optionen die Möglichkeit, die gelesene Eingabe mit Zeilennummern zu verbrämen, Zeilenenden und Sonderzeichen sichtbar zu machen oder Folgen von Leerzeilen zu einer zu komprimieren (Tabelle 8.2).

Es versteht sich, dass mit `cat` nur Textdateien eine vernünftige Bildschirmausgabe liefern. Wenn Sie das Kommando auf andere Dateitypen wie etwa die Binärdatei `/bin/cat` anwenden, ist es zumindest auf einem Textterminal sehr wahrscheinlich, dass nach Ende der Ausgabe die Eingabeaufforderung aus unleserlichen Zeichenkombinationen besteht. In diesem Fall können Sie durch (blinde) Eingabe von `reset` den Bildschirmzeichensatz wiederherstellen. Beim Umlenken der `cat`-Ausgabe in eine Datei ist das natürlich kein Problem.

Textdateien

Der *useless use of `cat` award* (Preis für den überflüssigen Gebrauch von `cat`) wird Leuten verliehen, die `cat` benutzen, wo es überhaupt nicht nötig ist. In den meisten Fällen akzeptieren Kommandos Dateinamen und lesen nicht nur ihre Standardeingabe, so dass `cat` nicht erforderlich ist, nur um ihnen eine einzige Datei auf der Standardeingabe zu verfüttern. Ein Aufruf wie »`cat data.txt | grep bla`« ist unnötig, wenn man genausogut »`grep bla data.txt`« schreiben kann. Selbst wenn `grep` nur seine Standardeingabe lesen könnte, wäre »`grep bla <data.txt`« kürzer und würde keinen zusätzlichen `cat`-Prozess involvieren.

Übungen

8.7 [2] Wie können Sie prüfen, ob in einem Verzeichnis Dateien mit »merkwürdigen« Namen enthalten sind, etwa solche mit Leerzeichen am Schluss oder mit unsichtbaren Steuerzeichen in der Mitte?

8.3.2 Anfang und Ende von Dateien – `head` und `tail`

Mitunter interessiert Sie nur ein Teil einer Datei: Die ersten paar Zeilen, um zu sehen, ob es die richtige Datei ist, oder vor allem bei einer Protokolldatei die letzten Einträge. Die Kommandos `head` und `tail` liefern genau das – standardmäßig respektive die ersten oder die letzten 10 Zeilen jeder Datei, deren Name sie als Argument übergeben bekommen (ersatzweise wie üblich die ersten oder letzten 10 Zeilen der Standard-Eingabe). Die Option `-n` erlaubt es, eine andere Anzahl von Zeilen auszugeben: »`head -n 20`« liefert die ersten 20 Zeilen der Standard-Eingabe, »`tail -n 5 daten.txt`« die letzten 5 Zeilen der Datei `daten.txt`.

Traditionell können Sie die Anzahl *n* der gewünschten Zeilen auch direkt in der Form »*-n*« angeben. Offiziell ist das nicht mehr erlaubt, aber die Linux-Versionen von `head` und `tail` unterstützen es noch.

Mit der Option `-c` können Sie angeben, dass nicht Zeilen, sondern Bytes gezählt werden sollen: »`head -c 20`« zeigt die ersten 20 Bytes der Standard-Eingabe, egal wie viele Zeilen das sind. Wenn Sie an die Zahl ein »`b`«, »`k`« oder »`m`« (»Blöcke«, »Kibibyte«, »Mebibyte«) anhängen, wird sie mit 512, 1024 bzw. 1048576 multipliziert.

`head` erlaubt ferner den Einsatz eines Minuszeichens: »`head -c -20`« zeigt die ganze Standard-Eingabe *bis auf* die letzten 20 Bytes.

Aus Fairnessgründen kann `tail` auch etwas, was `head` nicht kann: Wenn die Zeilenzahl mit »+« anfängt, zeigt es alles *ab* der betreffenden Zeile:

```
$ tail -n +3 datei                                   Alles ab Zeile 3
```

Das Programm `tail` unterstützt außerdem die wichtige Option `-f`. Sie führt dazu, dass `tail` nach der Ausgabe des aktuellen Dateiendes wartet und später hinzukommende Daten am Dateiende auch noch ausgibt. Dies ist sehr nützlich zur Beobachtung von Protokolldateien. Wenn Sie `tail -f` mehrere Dateinamen übergeben, schreibt es vor jeder neuen Ausgabe eine Kopfzeile, die angibt, in welcher Datei jetzt wieder neue Daten angekommen sind.

Übungen

8.8 [!2] Wie würden Sie gezielt nur die 13. Zeile der Eingabe ausgeben?

8.9 [3] Probieren Sie »`tail -f`« aus: Legen Sie eine Datei an und rufen Sie »`tail -f`« für diese Datei auf. Hängen Sie dann in einem anderen Fenster bzw. auf einer anderen virtuellen Konsole z. B. mit »`echo >>…`« etwas an die Datei an und beobachten Sie die `tail`-Ausgabe. Wie sieht das ganze aus, wenn `tail` mehrere Dateien gleichzeitig beobachtet?

8.10 [3] Was passiert mit »`tail -f`«, wenn die beobachtete Datei kürzer wird?

8.11 [3] Erklären Sie die Ausgabe der folgenden Kommandos

```
$ echo Hallo >/tmp/hallo
$ echo "Huhu Welt" >/tmp/hallo
```

wenn Sie nach dem ersten `echo` in einem anderen Fenster das Kommando

```
$ tail -f /tmp/hallo
```

gestartet haben.

8.4 Datenverwaltung

8.4.1 Sortierte Dateien – `sort` und `uniq`

Standardeinstellung

Mit dem Kommando `sort` können Sie die Zeilen von Textdateien nach vorgegebenen Kriterien sortieren. Die Standardeinstellung ist eine aufsteigende Sortierung, also von A nach Z, anhand der ASCII-Werte[1] der ersten Zeichen in einer Zeile. Dies ist auch der Grund dafür, dass deutsche Sonderzeichen fehlerhaft einsortiert werden. Beispielsweise beträgt der ASCII-Code von »Ä« 143, dieser Buchstabe wird also weit hinter »Z« mit dem ASCII-Code 91 eingeordnet. Auch der Kleinbuchstabe »a« gilt als »größer« als der Großbuchstabe »Z«.

Selbstverständlich kann `sort` sich auch nach deutschen Gepflogenheiten richten. Setzen Sie dazu eine der Umgebungsvariablen `LANG`, `LC_ALL` oder `LC_COLLATE` auf einen Wert wie »`de`«, »`de_DE`« oder »`de_DE.UTF-8`« (der genaue Wert hängt von Ihrer Distribution und Systemumgebung ab). Wenn Sie diese Einstellung nur für das `sort`-Kommando haben wollen, dann ist eine Angabe der Form

```
$ … | LC_COLLATE=de_DE.UTF-8 sort
```

möglich. Der Wert von `LC_ALL` hat Vorrang vor dem Wert von `LC_COLLATE` und dieser wiederum Vorrang vor dem Wert von `LANG`. Als Nebeneffekt sorgt die deutsche Sortierreihenfolge übrigens dafür, dass Groß- und Kleinschreibung ignoriert werden.

[1] Bekanntlich geht der ASCII nur bis 127. Wirklich gemeint ist hier der ASCII zusammen mit der gerade aktuellen Erweiterung für die Zeichen mit den Codes ab 128, also zum Beispiel ISO-8859-1, auch bekannt als ISO-Latin-1.

Wenn Sie nichts anderes angeben, wird »lexikographisch« unter Betrachtung der kompletten Zeile sortiert, das heißt, wenn die ersten Zeichen zweier Zeilen gleich sind, entscheidet das erste verschiedene Zeichen weiter hinten in der Zeile über deren relative Anordnung. Natürlich kann `sort` Zeilen nicht nur nach der ganzen Zeile, sondern auch »gezielt« nach den »Spalten« oder Feldern einer (zumindest konzeptuellen) Tabelle sortieren. Die Felder werden beginnend mit 1 numeriert; mit der Option »`-k 2`« würde das erste Feld ignoriert und das zweite Feld jeder Zeile zum Sortieren herangezogen. Sind die Werte zweier Zeilen im zweiten Feld gleich, wird der Rest der Zeile angeschaut, wenn Sie nicht mit etwas wie »`-k 2,3`« das letzte anzuschauende Feld angeben. »`-k 2,2`« sortiert *nur* nach dem Wert der zweiten Spalte. Es ist übrigens auch erlaubt, im selben Kommando mehrere `-k`-Optionen anzugeben.

Sortieren in Feldern

`sort` unterstützt außerdem noch eine antiquierte Form der Positionsangabe: Hier werden die Felder beginnend mit 0 numeriert, und das Startfeld wird mit »+*m*« und das Stoppfeld mit »-*n*« angegeben. Um die Differenzen zur modernen Form komplett zu machen, ist die Angabe des Stoppfelds auch noch »exklusive« – benannt wird das erste Feld, nach dem *nicht mehr* sortiert werden soll. Die Beispiele von oben wären also respektive »`+1`«, »`+1 -3`« und »`+1 -2`«.

Als Trennmarkierung zwischen den verschiedenen Feldern dient das Leerzeichen. Folgen mehrere Leerzeichen aufeinander, wird nur das erste als Trennzeichen interpretiert, die restlichen werden dem Inhalt des folgenden Feldes zugeordnet. Dazu ein kleines Beispiel, namentlich die Meldeliste für den alljährlichen Marathonlauf des TSV Lahmhausen. Ganz zu Anfang stellen wir sicher, dass wir die Standardsprachumgebung des Systems (»`POSIX`«) verwenden, indem wir die entsprechenden Umgebungsvariablen zurücksetzen. (Die vierte Spalte ist übrigens die Startnummer.)

Trennmarkierung

```
$ unset LANG LC_ALL LC_COLLATE
$ cat teilnehmer.dat
Schulz        Hugo      SV Schnaufenberg     123 Herren
Schleicher    Detlef    TSV Lahmhausen       13  Herren
Flöttmann     Fritz     Sportfreunde Renntal 217 Herren
Springinsfeld Karlheinz TV Jahnstein         154 Herren
von Traben    Gesine    TV Jahnstein         26  Damen
Rasbichel     Ulla      TSV Lahmhausen       117 Damen
Schwitz       Sieglinde Sportfreunde Renntal 93  Damen
Rasbichel     Katja     TSV Lahmhausen       119 Damen
Langbein      Leni      SV Schnaufenberg     55  Damen
Zielinger     Hannes    TV Jahnstein         45  Herren
Fluschinsky   Käthe     Sportfreunde Renntal 57  Damen
```

Versuchen wir uns zunächst an einer alphabetisch nach dem Nachnamen sortierten Liste. Das ist prinzipiell einfach, weil die Nachnamen ganz vorne in der Zeile stehen:

```
$ sort teilnehmer.dat
Fluschinsky   Käthe     Sportfreunde Renntal 57  Damen
Flöttmann     Fritz     Sportfreunde Renntal 217 Herren
Langbein      Leni      SV Schnaufenberg     55  Damen
Rasbichel     Katja     TSV Lahmhausen       119 Damen
Rasbichel     Ulla      TSV Lahmhausen       117 Damen
Schleicher    Detlef    TSV Lahmhausen       13  Herren
Schulz        Hugo      SV Schnaufenberg     123 Herren
Schwitz       Sieglinde Sportfreunde Renntal 93  Damen
Springinsfeld Karlheinz TV Jahnstein         154 Herren
Zielinger     Hannes    TV Jahnstein         45  Herren
von Traben    Gesine    TV Jahnstein         26  Damen
```

Sie sehen bestimmt die zwei kleinen Probleme in dieser Liste: Einerseits sollte »Flöttmann« vor »Fluschinsky« einsortiert werden, andererseits »von Traben« vor »Zielinger«. Beide verschwinden, wenn wir darauf achten, die deutschen Sortierregeln einzuhalten:

```
$ LC_COLLATE=de_DE sort teilnehmer.dat
Flöttmann     Fritz     Sportfreunde Renntal 217 Herren
Fluschinsky   Käthe     Sportfreunde Renntal 57  Damen
Langbein      Leni      SV Schnaufenberg     55  Damen
Rasbichel     Katja     TSV Lahmhausen       119 Damen
Rasbichel     Ulla      TSV Lahmhausen       117 Damen
Schleicher    Detlef    TSV Lahmhausen       13  Herren
Schulz        Hugo      SV Schnaufenberg     123 Herren
Schwitz       Sieglinde Sportfreunde Renntal 93  Damen
Springinsfeld Karlheinz TV Jahnstein         154 Herren
von Traben    Gesine    TV Jahnstein         26  Damen
Zielinger     Hannes    TV Jahnstein         45  Herren
```

Als nächstes sortieren wir nach dem Vornamen:

```
$ sort -k 2,2 teilnehmer.dat
Schulz        Hugo      SV Schnaufenberg     123 Herren
Schwitz       Sieglinde Sportfreunde Renntal 93  Damen
Langbein      Leni      SV Schnaufenberg     55  Damen
Flöttmann     Fritz     Sportfreunde Renntal 217 Herren
Zielinger     Hannes    TV Jahnstein         45  Herren
Rasbichel     Katja     TSV Lahmhausen       119 Damen
Rasbichel     Ulla      TSV Lahmhausen       117 Damen
Schleicher    Detlef    TSV Lahmhausen       13  Herren
Fluschinsky   Käthe     Sportfreunde Renntal 57  Damen
Springinsfeld Karlheinz TV Jahnstein         154 Herren
von Traben    Gesine    TV Jahnstein         26  Damen
```

Hier kommt die oben erwähnte Eigenschaft von `sort` zum Tragen, das erste einer Folge von Leerzeichen als Trenner zu interpretieren und die folgenden dem Anfang des nächsten Feldes zuzuschlagen. Wie Sie sehen, sind zwar die Vornamen alphabetisch sortiert, aber immer nur innerhalb der jeweils gleich langen Nachnamen. Dies können Sie durch die Option `-b` beheben, die Folgen von Leerzeichen so behandelt wie ein einziges:

```
$ sort -b -k 2,2 teilnehmer.dat
Schleicher    Detlef    TSV Lahmhausen       13  Herren
Flöttmann     Fritz     Sportfreunde Renntal 217 Herren
Zielinger     Hannes    TV Jahnstein         45  Herren
Schulz        Hugo      SV Schnaufenberg     123 Herren
Springinsfeld Karlheinz TV Jahnstein         154 Herren
Rasbichel     Katja     TSV Lahmhausen       119 Damen
Fluschinsky   Käthe     Sportfreunde Renntal 57  Damen
Langbein      Leni      SV Schnaufenberg     55  Damen
Schwitz       Sieglinde Sportfreunde Renntal 93  Damen
von Traben    Gesine    TV Jahnstein         26  Damen
Rasbichel     Ulla      TSV Lahmhausen       117 Damen
```

Die korrekte Sortierung von »Karlheinz«, »Katja« und »Käthe« erreichen Sie natürlich durch die Verwendung der deutschen Sprachumgebung, wobei Sie feststellen werden, dass diese auch die `-b`-Option impliziert. Die sortierte Liste enthält dann immer noch einen Schönheitsfehler; siehe hierzu Übung 8.14.

genauere Feldbestimmung

Das zu sortierende Feld können Sie noch genauer bestimmen, wie das folgende Beispiel zeigt:

Tabelle 8.3: Optionen für `sort` (Auswahl)

Option		Wirkung
`-b`	(*blank*)	ignoriert führende Leerzeichen im Feldinhalt
`-d`	(*dictionary*)	sortiert nach Wörterbuch-Kriterien, d. h. nur Buchstaben, Ziffern und Leerzeichen werden berücksichtigt
`-f`	(*fold*)	keine Unterscheidung von Groß- und Kleinbuchstaben
`-i`	(*ignore*)	nicht druckbare Zeichen werden ignoriert
`-k` ⟨*Feld*⟩`[,`⟨*Feld'*⟩`]`	(*key*)	Sortiere gemäß ⟨*Feld*⟩ (bis einschließlich ⟨*Feld'*⟩)
`-n`	(*numeric*)	betrachtet Feldinhalt als Zahl und sortiert nach dem numerischen Wert, führende Leerzeichen werden ignoriert
`-o datei`	(*output*)	schreibt Arbeitsergebnis in eine Datei, deren Name hier mit der Ursprungsdatei übereinstimmen darf!
`-r`	(*reverse*)	sortiert absteigend, also von Z nach A
`-t`⟨*Zeichen*⟩	(*terminate*)	das ⟨*Zeichen*⟩ dient als Feldtrennzeichen
`-u`	(*unique*)	gibt nur die erste einer Folge von identischen Zeilen aus

```
$ sort -br -k 2.2 teilnehmer.dat
Fluschinsky   Käthe     Sportfreunde Renntal 57  Damen
Schulz        Hugo      SV Schnaufenberg     123 Herren
Flöttmann     Fritz     Sportfreunde Renntal 217 Herren
von Traben    Gesine    TV Jahnstein         26  Damen
Rasbichel     Ulla      TSV Lahmhausen       117 Damen
Schwitz       Sieglinde Sportfreunde Renntal 93  Damen
Schleicher    Detlef    TSV Lahmhausen       13  Herren
Langbein      Leni      SV Schnaufenberg     55  Damen
Rasbichel     Katja     TSV Lahmhausen       119 Damen
Springinsfeld Karlheinz TV Jahnstein         154 Herren
Zielinger     Hannes    TV Jahnstein         45  Herren
```

Hier wird die Datei `teilnehmer.dat` absteigend (`-r`) nach dem zweiten Zeichen der zweiten Tabellenspalte, also dem zweiten Buchstaben des Vornamens, sortiert (sehr sinnvoll!). Auch in diesem Fall ist es erforderlich, führende Leerzeichen mit `-b` zu ignorieren. (Der Schönheitsfehler aus Übung 8.14 manifestiert sich auch hier noch.)

Mit der Option `-t` (engl. *terminate*, »begrenzen«) können Sie statt des Leerzeichens beliebige andere Trennzeichen festlegen. Dies ist fundamental eine gute Idee, weil die zu sortierenden Felder dann Leerzeichen enthalten dürfen. Hier ist eine bequemer zu verwendende (wenn auch schwerer zu lesende) Fassung unserer Beispieldatei:

andere Trennzeichen

```
Schulz:Hugo:SV Schnaufenberg:123:Herren
Schleicher:Detlef:TSV Lahmhausen:13:Herren
Flöttmann:Fritz:Sportfreunde Renntal:217:Herren
Springinsfeld:Karlheinz:TV Jahnstein:154:Herren
von Traben:Gesine:TV Jahnstein:26:Damen
Rasbichel:Ulla:TSV Lahmhausen:117:Damen
Schwitz:Sieglinde:Sportfreunde Renntal:93:Damen
Rasbichel:Katja:TSV Lahmhausen:119:Damen
Langbein:Leni:SV Schnaufenberg:55:Damen
Zielinger:Hannes:TV Jahnstein:45:Herren
Fluschinsky:Käthe:Sportfreunde Renntal:57:Damen
```

Die Sortierung nach dem Vornamen liefert nun mit »`LC_COLLATE=de_DE sort -t: -k2,2`« korrekte Ergebnisse. Auch wird es leichter, zum Beispiel nach der Startnum-

mer (nun Feld 4, unabhängig von der Anzahl der Leerzeichen im Vereinsnamen) zu sortieren:

```
$ sort -t: -k4 teilnehmer0.dat
Rasbichel:Ulla:TSV Lahmhausen:117:Damen
Rasbichel:Katja:TSV Lahmhausen:119:Damen
Schulz:Hugo:SV Schnaufenberg:123:Herren
Schleicher:Detlef:TSV Lahmhausen:13:Herren
Springinsfeld:Karlheinz:TV Jahnstein:154:Herren
Flöttmann:Fritz:Sportfreunde Renntal:217:Herren
von Traben:Gesine:TV Jahnstein:26:Damen
Zielinger:Hannes:TV Jahnstein:45:Herren
Langbein:Leni:SV Schnaufenberg:55:Damen
Fluschinsky:Käthe:Sportfreunde Renntal:57:Damen
Schwitz:Sieglinde:Sportfreunde Renntal:93:Damen
```

Natürlich ist auch die Sortierung nach der Startnummer, wenn Sie nichts anderes sagen, lexikographisch – »117« und »123« stehen vor »13« und das wiederum vor »154«. Dies läßt sich ändern, indem Sie die Option `-n` angeben und dadurch einen numerischen Vergleich erzwingen:

numerischer Vergleich

```
$ sort -t: -k4 -n teilnehmer0.dat
Schleicher:Detlef:TSV Lahmhausen:13:Herren
von Traben:Gesine:TV Jahnstein:26:Damen
Zielinger:Hannes:TV Jahnstein:45:Herren
Langbein:Leni:SV Schnaufenberg:55:Damen
Fluschinsky:Käthe:Sportfreunde Renntal:57:Damen
Schwitz:Sieglinde:Sportfreunde Renntal:93:Damen
Rasbichel:Ulla:TSV Lahmhausen:117:Damen
Rasbichel:Katja:TSV Lahmhausen:119:Damen
Schulz:Hugo:SV Schnaufenberg:123:Herren
Springinsfeld:Karlheinz:TV Jahnstein:154:Herren
Flöttmann:Fritz:Sportfreunde Renntal:217:Herren
```

Diese und die wichtigsten anderen Optionen für `sort` finden sich in Tabelle 8.3; es empfiehlt sich in jedem Fall, die Dokumentation des Programms genau zu studieren. `sort` ist ein vielseitiges und leistungsfähiges Programm, mit dem Sie sich jede Menge Arbeit sparen können.

Kommando `uniq`

Das Kommando `uniq` hat die wichtige Funktion, von unmittelbar aufeinanderfolgenden »gleichen« Zeilen in der Eingabe nur eine durchzulassen. Wann zwei Zeilen als »gleich« gelten, lässt sich – wie üblich – durch Optionen einstellen. `uniq` unterscheidet sich von den meisten der bisher gesehenen Programme dadurch, dass es keine beliebige Anzahl von benannten Eingabedateien akzeptiert, sondern höchstens eine; ein etwaiger zweiter Dateiname wird als Name für die Ausgabedatei angesehen (ersatzweise die Standard-Ausgabe). Wird keine Datei im `uniq`-Aufruf benannt, liest `uniq`, so wie es sich gehört, seine Standard-Eingabe.

`uniq` funktioniert am besten, wenn die Eingabezeilen sortiert sind, so dass *alle* gleichen Zeilen hintereinander stehen. Ist das nicht der Fall, so ist eben nicht gesagt, dass jede Zeile in der Ausgabe nur einmal vorkommt:

```
$ cat uniq-test
Hipp
Hopp
Hopp
Hipp
Hipp
Hopp
$ uniq uniq-test
Hipp
```

```
Hopp
Hipp
Hopp
```

Vergleichen Sie das mit der Ausgabe von »`sort -u`«:

```
$ sort -u uniq-test
Hipp
Hopp
```

Übungen

8.12 [!2] Sortieren Sie die Teilnehmerliste in `teilnehmer0.dat` (der Datei mit Doppelpunkten als Feldtrenner) nach den Vereinsnamen und innerhalb der Vereine nach den Nach- und Vornamen der Spieler (in dieser Reihenfolge).

8.13 [3] Wie können Sie die Teilnehmerliste aufsteigend nach den Vereinsnamen und innerhalb der Vereine absteigend nach der Startnummer sortieren? (*Tipp:* Dokumentation lesen!)

8.14 [!2] Was ist der »Schönheitsfehler«, von dem in den Beispielen die Rede ist, und warum tritt er auf?

8.15 [2] Ein Verzeichnis enthält Dateien mit den folgenden Namen:

```
01-2002.txt  01-2003.txt  02-2002.txt  02-2003.txt
03-2002.txt  03-2003.txt  04-2002.txt  04-2003.txt
<⋘⋘⋘⋘⋘
11-2002.txt  11-2003.txt  12-2002.txt  12-2003.txt
```

Geben Sie ein `sort`-Kommando an, mit dem Sie die Ausgabe von `ls` in die »chronologisch richtige« Reihenfolge

```
01-2002.txt
02-2002.txt
<⋘⋘⋘⋘⋘
12-2002.txt
01-2003.txt
<⋘⋘⋘⋘⋘
12-2003.txt
```

bringen können.

8.4.2 Spalten und Felder – `cut`, `paste` & Co.

Während Sie mit dem Kommando `grep` Zeilen einer Textdatei durchsuchen und ausschneiden können, arbeitet sich `cut` (engl. für »schneiden«) gewissermaßen vertikal durch einen Text. Dies kann auf zwei Arten erfolgen:

Spalten ausschneiden

Eine Möglichkeit ist die absolute Bearbeitung von Spalten. Diese Spalten entsprechen einzelnen Zeichen einer Zeile. Um solche Spalten auszuschneiden, muss nach der Option `-c` (engl. *column*, »Spalte«) die Spaltennummer angegeben werden. Sollen mehrere Spalten in einem Schritt ausgeschnitten werden, können diese als kommaseparierte Liste festgelegt werden. Auch die Angabe von Spaltenbereichen ist zulässig.

absolute Spalten

```
$ cut -c 15,1-5 teilnehmer.dat
SchulH
SchleD
```

```
FlöttF
SprinK
von TG
◁◁◁◁◁
```

Im Beispiel werden der Anfangsbuchstabe des Vornamens sowie die ersten fünf Zeichen des Nachnamens ausgeschnitten. Es illustriert auch gleich die bemerkenswerte Tatsache, dass die Ausgabe immer in der Reihenfolge erfolgt, wie die ausgeschnittenen Spalten in der Eingabe stehen. Auch wenn die ausgewählten Spaltenbereiche sich überlappen, wird jedes Zeichen der Eingabe höchstens einmal ausgegeben:

```
$ cut -c 1-5,2-6,3-7 teilnehmer.dat
Schulz
Schleic
Flöttma
Springi
von Tra
◁◁◁◁◁
```

Felder ausschneiden

Die zweite Möglichkeit ist, relativ in Feldern auszuschneiden. Diese Felder werden durch Trennzeichen abgegrenzt. Möchten Sie feldweise ausschneiden, benötigt `cut` die Option `-f` (engl. *field*, »Feld«) und die gewünschte Feldnummer. Für diese gelten die gleichen Regeln wie für die Spaltennummern. Übrigens schließen sich die Optionen `-c` und `-f` gegenseitig aus.

Trenner

Als Trenner ist das Tabulatorzeichen voreingestellt, andere Trenner lassen sich mit der Option `-d` (engl. *delimiter*, »Trennzeichen«) vorgeben:

```
$ cut -d: -f 1,4 teilnehmer0.dat
Schulz:123
Schleicher:13
Flöttmann:217
Springinsfeld:154
von Traben:26
Rasbichel:117
◁◁◁◁◁
```

Auf diese Weise werden der Meldeliste die Nachnamen der Teilnehmer (Spalte 1) sowie die Benutzernummern (Spalte 4) entnommen, Trennzeichen ist der Doppelpunkt. Aus Gründen der Übersichtlichkeit ist hier nur eine verkürzte Ausgabe abgebildet.

Sie können übrigens mit der Option `--output-delimiter` ein anderes Trennzeichen für die Ausgabe festlegen als das, welches für die Eingabe verwendet wird:

```
$ cut -d: --output-delimiter=': ' -f 1,4 teilnehmer0.dat
Schulz: 123
Schleicher: 13
Flöttmann: 217
Springinsfeld: 154
von Traben: 26
Rasbichel: 117
◁◁◁◁◁
```

Wenn Sie tatsächlich Spalten oder Felder umsortieren wollen, ist schwereres Geschütz angesagt, etwa die Programme `awk` oder `perl`. Notfalls geht es auch mit dem gleich vorgestellten Kommando `paste`, das ist aber etwas mühselig.

Unterdrückung von Zeilen ohne Felder

Bei der feldweisen Bearbeitung von Textdateien ist `-s` (engl. *separator*, »Trenner«) eine sinnvolle Option. Findet »`cut -f`« Zeilen, die kein Trennzeichen enthalten, werden diese üblicherweise komplett ausgegeben; `-s` verhindert diese Ausgabe.

Dateien zeilenweise zusammenfügen

Das Kommando `paste` (engl. für »zusammenkleben«) fügt die angegebenen Dateien zeilenweise zusammen, es wird daher oft in Verbindung mit `cut` benutzt. Wie Sie sicher sofort bemerkt haben, ist `paste` eigentlich kein Filterkommando. Geben Sie jedoch für einen der Dateinamen ein Minuszeichen an, so registriert `paste`, dass dieser Text aus der Standard-Eingabe gelesen werden soll. Die Ausgabe erfolgt stets auf der Standard-Ausgabe.

Dateien parallel durchlaufen

Wie erwähnt arbeitet `paste` zeilenweise. Bei der Angabe von zwei Dateinamen werden die erste Zeile aus der ersten Datei und die erste aus der zweiten Datei, durch ein Tabulatorzeichen getrennt, zur ersten Zeile der Ausgabe verbunden, entsprechend wird mit allen weiteren Zeilen verfahren. Wenn Sie statt des Tabulatorzeichens ein anderes Trennzeichen verwenden wollen, kann dies mit der Option `-d` festgelegt werden.

Trennzeichen

Zum Beispiel können wir eine Version der Marathon-Meldeliste herstellen, bei der die Startnummer vorne steht:

```
$ cut -d: -f4 teilnehmer0.dat >startnr.dat
$ cut -d: -f1-3,5 teilnehmer0.dat \
>   | paste -d: startnr.dat - >tn-startnr.dat
$ cat tn-startnr.dat
123:Schulz:Hugo:SV Schnaufenberg:Herren
13:Schleicher:Detlef:TSV Lahmhausen:Herren
217:Flöttmann:Fritz:Sportfreunde Renntal:Herren
154:Springinsfeld:Karlheinz:TV Jahnstein:Herren
26:von Traben:Gesine:TV Jahnstein:Damen
117:Rasbichel:Ulla:TSV Lahmhausen:Damen
93:Schwitz:Sieglinde:Sportfreunde Renntal:Damen
119:Rasbichel:Katja:TSV Lahmhausen:Damen
55:Langbein:Leni:SV Schnaufenberg:Damen
45:Zielinger:Hannes:TV Jahnstein:Herren
57:Fluschinsky:Käthe:Sportfreunde Renntal:Damen
```

Diese Datei kann jetzt bequem mit »`sort -n tn-startnr.dat`« nach dem numerischen Wert der Startnummer sortiert werden.

Dateien nacheinander durchlaufen

Durch `-s` (engl. *serial*, »nacheinander«) werden die angegebenen Dateien nacheinander durchlaufen. Zunächst werden alle Zeilen der ersten Datei mit Trennzeichen zu einer Zeile zusammengefasst, anschließend alle Zeilen aus der zweiten Datei in der zweiten Zeile usw.

```
$ cat liste1
Hund
Katze
Maus
$ cat liste2
Ei
Blut
Kakao
$ paste -s liste*
Hund    Katze   Maus
Ei      Blut    Kakao
```

Alle Dateien, deren Name dem Suchmuster `liste*` entspricht, hier also lediglich `liste1` und `liste2`, werden von `paste` zusammengesetzt. Die Angabe von `-s` bewirkt, dass jede Zeile dieser Dateien eine Spalte der Ausgabe ergibt.

Übungen

8.16 [!2] Generieren Sie eine neue Version der Datei `teilnehmer.dat` (der mit der festen Spaltenbreite), in der die Startnummer und die Vereinszugehörigkeit nicht auftauchen.

8.17 [!2] Generieren Sie eine neue Version der Datei `teilnehmer0.dat` (der mit den durch Doppelpunkt getrennten Feldern), in der die Startnummer und die Vereinszugehörigkeit nicht auftauchen.

8.18 [3] Erzeugen Sie eine Version der Datei `teilnehmer0.dat`, bei der die Felder nicht durch Doppelpunkte, sondern durch die Zeichenkette »`,␣`« (Komma gefolgt von einem Leerzeichen) getrennt sind.

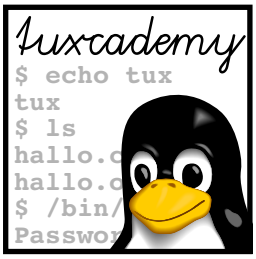
8.19 [3] Wie viele verschiedene Gruppen werden von Benutzern auf Ihrem System als primäre Gruppen benutzt? (Die primäre Gruppe eines Benutzers ist das vierte Feld in der Datei `/etc/passwd`.)

Kommandos in diesem Kapitel

cat	Hängt Dateien aneinander	cat(1)	118
cut	Extrahiert Felder oder Spalten aus seiner Eingabe	cut(1)	125
head	Zeigt den Anfang einer Datei an	head(1)	119
paste	Fügt verschiedene Eingabedateien zeilenweise aneinander	paste(1)	127
reset	Setzt den Bildschirmzeichensatz auf einen „vernünftigen“ Wert	tset(1)	119
sort	Sortiert die Zeilen seiner Eingabe	sort(1)	120
tail	Zeigt das Ende einer Datei an	tail(1)	119
tee	Kopiert die Standardeingabe in die Standardausgabe und außerdem in Dateien	tee(1)	117
uniq	Ersetzt Folgen von gleichen Zeilen in der Eingabe durch die erste solche	uniq(1)	124

Zusammenfassung

- Jedes Linux-Programm unterstützt die Standard-Ein- und -Ausgabe-Kanäle `stdin`, `stdout` und `stderr`.
- Die Standard-Ausgabe und Standard-Fehlerausgabe können mit den Operatoren > und >>, die Standard-Eingabe mit dem Operator < umgeleitet werden.
- Über Pipelines lassen sich die Standard-Aus- und -Eingabe von Programmen direkt (ohne Zwischendateien) miteinander verbinden.
- Mit dem Kommando `tee` können Zwischenergebnisse einer Pipeline in Dateien gespeichert werden.
- Filterkommandos (oder »Filter«) lesen ihre Standardeingabe, manipulieren sie und schreiben die Ergebnisse auf die Standardausgabe.
- `sort` ist ein vielseitiges Sortierprogramm.
- Das Kommando `cut` schneidet bestimmte Spaltenbereiche oder Felder jeder Zeile der Eingabe aus.
- Mit `paste` können die Zeilen von Dateien aneinandergehängt werden.



9

Mehr über die Shell

Inhalt

Lernziele

- Shell- und Umgebungsvariable kennenlernen

Vorkenntnisse

- Shell-Grundkenntnisse (Kapitel 4)
- Dateiverwaltung und einfache Filterkommandos (Kapitel 6, Kapitel 8)
- Umgang mit einem Texteditor (Kapitel 3)

9.1 `sleep`, `echo` und `date`

Bevor wir uns mit der eigentlichen Shell beschäftigen, müssen wir Ihnen noch ein bisschen mehr Material für Experimente geben. Dazu erklären wir Ihnen hier noch einige ganz einfache Kommandos:

sleep Dieses Kommando tut für die als Parameter angegebene Anzahl von Sekunden gar nichts. Sie können es benutzen, wenn Sie wollen, dass Ihre Shell eine »Kunstpause« einlegt:

```
$ sleep 10
                                        Ca. 10 Sekunden lang passiert nichts
$ _
```

Argumente ausgeben

echo Das Kommando `echo` gibt seine Argumente aus (sonst nichts), und zwar durch Leerzeichen getrennt. Es ist aber doch interessant und nützlich, da die Shell vorher Variablenbezüge (siehe Abschnitt 9.2) und ähnliches ersetzt:

```
$ w=Welt
$ echo Hallo $w
Hallo Welt
$ echo Hallo ${w}enbummler
Hallo Weltenbummler
```

(Das zweite `echo` illustriert, was Sie machen können, wenn direkt etwas an den Ausgabewert einer Variable angehängt werden soll.)

Wird `echo` mit der Option `-n` aufgerufen, so schreibt es am Ende seiner Ausgabe keinen Zeilentrenner:

```
$ echo -n Hallo
Hallo$
```

Datum und Uhrzeit anzeigen

date Das Kommando `date` (engl. »Datum«) zeigt das aktuelle Datum sowie die Uhrzeit an. Sie können in weiten Grenzen bestimmen, wie diese Ausgabe aussehen soll – rufen Sie »`date --help`« auf oder lesen Sie mit »`man date`« die Online-Dokumentation.

(Beim zweiten Durcharbeiten dieses Kapitels:) Insbesondere betätigt `date` sich als Weltzeituhr, wenn Sie vorher die Umgebungsvariable `TZ` auf den Namen einer Zeitzone oder wichtigen Stadt (typischerweise Hauptstadt) setzen:

```
$ date
Thu Oct  5 14:26:07 CEST 2006
$ export TZ=Asia/Tokyo
$ date
Tue Oct  5 21:26:19 JST 2006
$ unset TZ
```

Die gültigen Zeitzonen- und Städtenamen können Sie herausfinden, indem Sie in `/usr/share/zoneinfo` stöbern.

Systemzeit stellen

Während jeder Anwender die Systemzeit abfragen darf, ist es nur dem Systemadministrator `root` erlaubt, mit dem Befehl `date` und einem Argument in der Form `MMTThhmm` die Systemzeit zu verändern. Im Argument stehen dabei `MM` für Monat, `TT` für Tag, `hh` für Stunde und `mm` für Minute. Optional dazu können noch jeweils zwei

Stellen für die Jahreszahl (plus möglicherweise zwei für das Jahrhundert) sowie die Sekunden (mit einem Punkt davor) angegeben werden, was aber nur in den seltensten Fällen notwendig sein dürfte.

```
$ date
Thu Oct  5 14:28:13 CEST 2006
$ date 08181715
date: cannot set date: Operation not permitted
Fri Aug 18 17:15:00 CEST 2006
```

Mit dem `date`-Kommando wird nur die interne Zeit des Linux-Systems geändert. Diese Zeit wird nicht notwendigerweise in die CMOS-Uhr auf der Hauptplatine des Rechners übertragen, so dass es notwendig sein kann, diese mit einem speziellen Kommando zu ändern. Viele Distributionen machen das automatisch, wenn das System heruntergefahren wird.

Übungen

9.1 [!3] Angenommen, jetzt ist der 22. Oktober 2003, 12:34 Uhr und 56 Sekunden. Studieren Sie die Dokumentation von `date` und geben Sie die Formatierungsanweisungen an, mit denen Sie die folgenden Ausgaben erreichen können:

1. `22-10-2003`
2. `03-294 (KW43)` (Zweistellige Jahreszahl, fortlaufende Nummer des Tags im Jahr, Kalenderwoche)
3. `12h34m56s`

9.2 [!2] Wie spät ist es gerade in Los Angeles?

9.2 Shell-Variable und die Umgebung

Die Bash hat – wie die meisten gängigen Shells – Eigenschaften, die man sonst in Programmiersprachen findet. Zum Beispiel ist es möglich, Text oder numerische Werte in Variablen abzulegen und später wieder hervorzuholen. Variable steuern auch verschiedene Aspekte der Funktionsweise der Shell selbst.

In der Shell wird eine Variable durch ein Kommando wie »`bla=fasel`« gesetzt (dieses Kommando setzt die Variable `bla` auf den textuellen Wert `fasel`). Achten Sie darauf, dass vor und hinter dem Gleichheitszeichen *keine* Leerzeichen stehen! Verwenden können Sie den Wert der Variablen, indem Sie den Variablennamen mit einem vorgesetzten Dollarzeichen benutzen:

Variable setzen

```
$ bla=fasel
$ echo bla
bla
$ echo $bla
fasel
```

(achten Sie auf den Unterschied).

Wir unterscheiden **Umgebungsvariable** und **Shellvariable**. Shellvariable sind nur in der betreffenden Shell sichtbar. Im Gegensatz dazu werden die Umgebungsvariablen beim Starten eines externen Kommandos an den Kindprozess weitergegeben und können auch dort benutzt werden. (Der Kindprozess muss nicht unbedingt eine Shell sein; jeder Linux-Prozess hat Umgebungsvariable.) Alle Umgebungsvariablen einer Shell sind gleichzeitig auch Shellvariable, aber umgekehrt ist es nicht so.

Umgebungsvariable
Shellvariable

Mit dem Kommando `export` können Sie eine existierende Shellvariable zur Umgebungsvariable erklären:

`export`

Tabelle 9.1: Wichtige Variable der Shell

Variable	Bedeutung
PWD	Name des aktuellen Verzeichnisses
PS1	enthält die Zeichenkette, die am Beginn jeder Eingabezeile angezeigt wird (den Prompt)
UID	enthält die Benutzerkennung
HOME	Pfad des Heimatverzeichnisses des Benutzers
PATH	Liste von Verzeichnissen, die von der Shell automatisch als Suchpfade für Befehle verwendet werden
LOGNAME	enthält den Benutzernamen

```
$ bla=fasel                                    bla ist jetzt Shellvariable
$ export bla                                   bla ist jetzt Umgebungsvariable
```

Oder Sie definieren eine neue Variable gleich als Shell- und Umgebungsvariable:

```
$ export bla=fasel
```

`export` funktioniert auch für mehrere Variable auf einmal:

```
$ export bla blubb
$ export bla=fasel blubb=bli
```

Variable auflisten

Die Umgebungsvariablen können Sie sich mit dem Befehl `export` (ohne Parameter) anzeigen lassen. Auch der Befehl `env` (ebenfalls ohne Parameter) zeigt die aktuelle Umgebung an. Alle Shellvariablen (inklusive diejenigen, die auch in der Umgebung sind) können Sie sich mit dem Befehl `set` anzeigen lassen. Die gebräuchlichsten Variablen und ihre Zuordnung sind in Tabelle 9.1 aufgelistet.

Der Befehl `set` tut noch viele andere fremdartige und wundervolle Dinge. Er wird Ihnen in der Linup-Front-Schulungsunterlage *Linux für Fortgeschrittene* wieder begegnen, wo es um Shellprogrammierung geht.

Auch `env` ist eigentlich dafür gedacht, die Prozessumgebung zu manipulieren, und nicht nur, sie anzuzeigen. Betrachten Sie das folgende Beispiel:

```
$ env bla=fasel bash                           Starte Kind-Shell mit bla
$ echo $bla
fasel
$ exit                                         Zurück in die Elter-Shell
$ echo $bla
                                               Nicht definiert
$ _
```

Zumindest in der Bash (und Verwandten) brauchen Sie `env` nicht wirklich, um Kommandos mit einer erweiterten Umgebung zu starten – ein einfaches

```
$ bla=fasel bash
```

hat dieselbe Wirkung. Allerdings erlaubt `env` Ihnen auch das temporäre Entfernen von Variablen aus der Umgebung (wie?).

Variable löschen

Wenn Sie von einer Shellvariablen genug haben, können Sie sie mit dem Befehl `unset` löschen. Damit verschwindet sie auch aus der Prozessumgebung. Wenn Sie eine Variable aus der Umgebung entfernen wollen, sie aber als Shellvariable beibehalten möchten, verwenden Sie »`export -n`«:

```
$ export bla=fasel                    bla ist Umgebungsvariable
$ export -n bla                       bla ist (nur) Shellvariable
$ unset bla                           bla ist ganz weg
```

Übungen

9.3 [!2] Überzeugen Sie sich, dass die Übergabe (oder Nichtübergabe) von Umgebungs- und Shellvariablen an Kindprozesse funktioniert wie behauptet, indem Sie die folgende Kommandosequenz durchspielen:

```
$ bla=fasel                           bla ist Shellvariable
$ bash                                Neue Shell (Kindprozess)
$ echo $bla
                                      bla ist nicht definiert
$ exit                                Zurück in die Elter-Shell
$ export bla                          bla ist Umgebungsvariable
$ bash                                Neue Shell (Kindprozess)
$ echo $bla
fasel                                 Umgebungsvariable wurde vererbt
$ exit                                Zurück in die Elter-Shell
```

9.4 [!2] Was passiert, wenn Sie eine Umgebungsvariable im Kindprozess ändern? Betrachten Sie die folgende Kommandosequenz:

```
$ export bla=fasel                    bla ist Umgebungsvariable
$ bash                                Neue Shell (Kindprozess)
$ echo $bla
fasel                                 Umgebungsvariable wurde vererbt
$ bla=blubb                           Neuer Wert
$ exit                                Zurück in die Elter-Shell
$ echo $bla                           Was kommt hier heraus??
```

9.3 Arten von Kommandos – die zweite

Steuerung der Shell

Eine Anwendung von Shellvariablen ist, wie erwähnt, die Steuerung der Shell selbst. Hierzu noch ein Beispiel: Wie in Kapitel 4 beschrieben, unterscheidet die Shell interne und externe Kommandos. Externe Kommandos entsprechen ausführbaren Programmen, die die Shell in den Verzeichnissen sucht, die in der Umgebungsvariablen `PATH` stehen. Hier ist ein typischer Wert für `PATH`:

```
$ echo $PATH
/home/hugo/bin:/usr/local/bin:/usr/bin:/bin:/usr/games
```

Die einzelnen Verzeichnisse werden in der Liste durch Doppelpunkte getrennt, die Liste im Beispiel besteht also aus fünf Verzeichnissen. Wenn Sie ein Kommando wie

```
$ ls
```

eingeben, weiß die Shell, dass das kein internes Kommando ist (sie kennt ihre internen Kommandos) und fängt darum an, die Verzeichnisse in `PATH` abzusuchen, beginnend am linken Ende. Konkret prüft sie, ob die folgenden Dateien existieren:

```
/home/hugo/bin/ls                                   Nein …
/usr/local/bin/ls                                   Immer noch nicht …
/usr/bin/ls                                         Nach wie vor nicht …
/bin/ls                                             Hah, Treffer!
                                    Das Verzeichnis /usr/games wird nicht mehr angeschaut.
```

Das heißt, zur Ausführung des Kommandos `ls` wird die Datei `/bin/ls` herangezogen.

Diese Suche ist natürlich ein relativ aufwendiger Prozess, und darum baut die Shell für die Zukunft vor: Wenn sie einmal die Datei `/bin/ls` als Implementierung des Kommandos `ls` ausgemacht hat, dann merkt sie sich diese Zuordnung bis auf weiteres. Diesen Vorgang nennt der Fachmann *hashing*, und dass er passiert ist, können Sie sehen, wenn Sie `type` auf unser Kommando `ls` anwenden:

```
$ type ls
ls is hashed (/bin/ls)
```

Das Kommando »`hash`« sagt Ihnen, welche Kommandos Ihre Bash bereits »gehasht« hat und wie oft sie seitdem aufgerufen wurden. Mit »`hash -r`« können Sie das komplette Hashing-Gedächtnis der Shell löschen. Es gibt noch ein paar andere Optionen, die Sie in der Bash-Dokumentation nachschlagen oder per »`help hash`« herausfinden können.

Die Variable `PATH` muss prinzipiell gesehen überhaupt keine Umgebungsvariable sein – für die aktuelle Shell funktioniert sie auch als Shellvariable (siehe Übung 9.5). Allerdings ist es bequem, sie als Umgebungsvariable zu definieren, damit auch die Kindprozesse der Shell (oft wieder Shells) den gewünschten Wert verwenden.

Wenn Sie wissen wollen, genau welches Programm die Shell für ein externes Kommando heranzieht, können Sie dafür das Kommando `which` verwenden:

```
$ which grep
/bin/grep
```

`which` verwendet dasselbe Verfahren wie die Shell – es beginnt beim ersten Verzeichnis in `PATH` und prüft jeweils, ob es in dem betreffenden Verzeichnis eine ausführbare Datei gibt, die so heißt wie das gesuchte Kommando.

`which` weiß nichts über die internen Kommandos der Shell; selbst wenn etwas wie »`which test`« also »`/usr/bin/test`« liefert, heißt das noch lange nicht, dass dieses Programm tatsächlich ausgeführt wird, denn interne Kommandos haben Vorrang gegenüber externen. Wenn Sie wissen wollen, was wirklich passiert, müssen Sie das Shell-Kommando »`type`« benutzen (Abschnitt 4.3.3).

Das Kommando `whereis` liefert nicht nur ausführbare Programme, sondern auch Dokumentationsdateien (Manpages), Quellcodedateien und andere interessante Dateien, die etwas mit den angegebenen Kommandos zu tun haben. Zum Beispiel:

```
$ whereis passwd
passwd: /usr/bin/passwd /etc/passwd /etc/passwd.org /usr/share/passwd▷
 ◁ /usr/share/man/man1/passwd.1.gz /usr/share/man/man1/passwd.1ssl.gz▷
 ◁ /usr/share/man/man5/passwd.5.gz
```

Dafür wird ein im Programm hartcodiertes Verfahren verwendet, das (ansatzweise) in `whereis`(1) erklärt wird.

Übungen

9.5 [2] Überzeugen Sie sich davon, dass die Kommandosuche der Shell auch funktioniert, wenn `PATH` keine Umgebungsvariable, sondern »nur« eine Shellvariable ist. Was passiert, wenn Sie `PATH` komplett entfernen?

9.6 [!1] Wie heißen auf Ihrem System die ausführbaren Programme, die zur Bearbeitung der folgenden Kommandos herangezogen werden: `fgrep`, `sort`, `mount`, `xterm`

9.7 [!1] Wie heißen auf Ihrem System die Dateien, die die Dokumentation für das Kommando »`crontab`« enthalten?

9.4 Die Shell als komfortables Werkzeug

Da für viele Linux-Anwender die Shell das am meisten genutzte Werkzeug ist, haben deren Entwickler sich große Mühe gegeben, ihre Bedienung komfortabel zu machen. Hier zeigen wir Ihnen noch ein paar nützliche Kleinigkeiten.

Kommandoeditor Sie können Kommandozeilen wie mit einem einfachen Texteditor bearbeiten. Der Cursor kann also in der Zeile hin- und herbewegt sowie Zeichen beliebig gelöscht oder hinzugefügt werden, bis die Eingabe durch Betätigen der Eingabetaste beendet wird. Das Verhalten dieses Editors kann übrigens mit »`set -o vi`« bzw. mit »`set -o emacs`« an das Verhalten der beiden bekanntesten Editoren unter Linux (Kapitel 3) angepasst werden.

Kommandoabbruch Bei den zahlreichen Linux-Kommandos kann es durchaus vorkommen, dass Sie mal einen Namen verwechseln oder einen falschen Parameter übergeben. Deshalb können Sie ein Kommando abbrechen, während es läuft. Hierzu müssen Sie nur die Tasten Strg+c gleichzeitig drücken.

Die »Vorgeschichte« Die Shell merkt sich Ihre letzten soundsovielen Kommandos als Vorgeschichte (engl. *history*), und Sie können sich mit den Cursortasten ↑ und ↓ in der Liste bewegen. Wenn Sie ein früheres Kommando finden, das Ihnen gefällt, können Sie es mit ↩ einfach, so wie es ist, erneut ausführen oder es zunächst (wie weiter oben angedeutet) abändern. Mit Strg+r können Sie die Liste »inkrementell« durchsuchen – tippen Sie einfach eine Zeichenfolge ein, und die Shell zeigt Ihnen das zuletzt ausgeführte Kommando, das die Zeichenfolge enthält. Je länger Ihre Zeichenfolge ist, desto präziser wird die Suche.

Die Vorgeschichte wird beim ordnungsgemäßen Verlassen des Systems in der versteckten Datei `~/.bash_history` gespeichert und steht nach dem nächsten Anmelden wieder zur Verfügung. (Sie können einen anderen Dateinamen verwenden, indem Sie die Variable `HISTFILE` auf den gewünschten Namen setzen.)

Eine Konsequenz der Tatsache, dass die Vorgeschichte in einer »gewöhnlichen« Datei abgespeichert wird, ist, dass Sie sie mit einem Texteditor ändern können. (Kapitel 3 erklärt Ihnen, wie das geht.) Sollten Sie also versehentlich Ihr Kennwort auf der Kommandozeile eintippen, können (und sollten!) Sie es mit der Hand aus der Vorgeschichte entfernen – vor allem, wenn Ihr System zu den eher freizügigen gehört, wo Heimatverzeichnisse standardmäßig für alle anderen Benutzer lesbar sind.

Tabelle 9.2: Tastaturkürzel innerhalb der Bash

Tastaturkürzel	Funktion
↑ bzw. ↓	durch frühere Kommandos blättern
Strg + r	Frühere Kommandos durchsuchen
← bzw. →	Cursor in Kommandozeile bewegen
Pos1 oder Strg + a	Cursor an Zeilenanfang setzen
Ende oder Strg + e	Cursor an Zeilenende setzen
⇐ bzw. Entf	Zeichen vor bzw. nach Cursor löschen
Strg + t	die beiden Zeichen vor/unter Cursor tauschen
Strg + l	Bildschirm löschen
Strg + c	Kommando abbrechen
Strg + d	Eingabe beenden (in der Login-Shell: Abmelden)

Komplettierung von Kommando- bzw. Dateinamen

Autovervollständigung Eine massive Bequemlichkeit ist die Fähigkeit der Bash zur automatischen Komplettierung von Kommando- bzw. Dateinamen. Wenn Sie die Tab-Taste drücken, vervollständigt die Shell eine unvollständige Eingabe, sofern die Fortsetzung eindeutig identifiziert werden kann. Dazu werden für das erste Wort des Kommandos alle ausführbaren Programme und weiter hinten die im aktuellen bzw. angegebenen Verzeichnis befindlichen Dateien herangezogen. Existieren hierbei mehrere Dateien, deren Bezeichnungen gleich beginnen, vervollständigt die Shell die Eingabe so weit wie möglich und gibt durch ein akustisches Signal zu erkennen, dass der Datei- bzw. Befehlsname noch immer unvollendet sein kann. Ein erneutes Drücken von Tab listet dann die verbleibenden Möglichkeiten auf.

Befehlsliste

Wenn Sie einen einzelnen (oder ein paar) Buchstaben eingeben und dann zweimal die Tab-Taste drücken, gibt die Shell eine Liste aller verfügbaren Befehle mit dem gewünschten Anfangsbuchstaben aus. Dies ist zum Beispiel sehr hilfreich, um sich die Schreibweise selten gebrauchter Befehle ins Gedächtnis zurück zu rufen. Der gleiche Effekt ist auch mit der Tastenkombination Esc ? zu erzielen. Soll ein Dateiname vervollständigt werden, kann dies mit Esc / erfolgen.

Es ist möglich, die Vervollständigung der Shell an bestimmte Programme anzupassen. Zum Beispiel könnte sie auf der Kommandozeile eines FTP-Programms statt Dateinamen die Namen bereits besuchter Rechner anbieten. Näheres steht in der Bash-Dokumentation.

Tabelle 9.2 gibt eine Übersicht über die in der Bash möglichen Tastaturkürzel.

Mehrere Kommandos auf einer Zeile Sie können durchaus mehrere Kommandos auf derselben Eingabezeile angeben. Sie müssen sie dazu nur mit dem Semikolon trennen:

```
$ echo Heute ist; date
Heute ist
Fr 5. Dez 12:12:47 CET 2008
```

In diesem Fall wird das zweite Kommando ausgeführt, sobald das erste fertig ist.

Rückgabewert

Bedingte Ausführung Manchmal nützlich ist es, die Ausführung des zweiten Kommandos davon abhängig zu machen, ob das erste korrekt ausgeführt wurde oder nicht. Jeder Unix-Prozess liefert einen **Rückgabewert**, der angibt, ob er korrekt ausgeführt wurde oder ob irgendwelche Fehler aufgetreten sind. Im ersteren Fall ist der Rückgabewert 0, im letzteren von 0 verschieden.


Sie können den Rückgabewert eines Kindprozesses Ihrer Shell herausfinden, indem Sie die Variable `$?` anschauen:

```
$ bash                                          Eine Kind-Shell starten …
$ exit 33                                       … und gleich wieder beenden
exit
$ echo $?
33                                              Der Wert aus unserem exit oben
$ _
```

Aber für das Folgende ist das eigentlich egal.

Mit `&&` als »Trennzeichen« zwischen zwei Kommandos (da, wo sonst ein Semikolon stünde) wird das zweite Kommando nur dann ausgeführt, wenn das erste erfolgreich beendet wurde. Um Ihnen das zu demonstrieren, benutzen wir die `-c`-Option der Bash, mit der Sie der Kind-Shell ein Kommando auf der Kommandozeile übergeben können (beeindruckend, was?):

```
$ bash -c "exit 0" && echo "Erfolgreich"
Erfolgreich
$ bash -c "exit 33" && echo "Erfolgreich"
                                                Nichts - 33 ist kein Erfolg!
```

Umgekehrt wird mit `||` als »Trennzeichen« das zweite Kommando nur dann ausgeführt, wenn das erste *nicht* erfolgreich beendet wurde:

```
$ bash -c "exit 0" || echo "Nicht erfolgreich"
$ bash -c "exit 33" || echo "Nicht erfolgreich"
Nicht erfolgreich
```

Übungen


9.8 [3] Was ist das Problem mit dem Kommando »`echo "Hallo!"`«? (*Tipp:* Experimentieren Sie mit Kommandos der Form »`!-2`« oder »`!ls`«.)

9.5 Kommandos aus einer Datei

Sie können Shell-Kommandos in einer Datei ablegen und *en bloc* ausführen. (Wie Sie bequem eine Datei anlegen können, lernen Sie in Kapitel 3.) Dazu müssen Sie nur die Shell aufrufen und den Namen der Datei als Parameter übergeben:

```
$ bash meine-kommandos
```

Eine solche Datei bezeichnet man auch als **Shellskript**, und die Shell hat umfangreiche Möglichkeiten zur Programmierung, die wir hier nur grob umreißen können. auf die wir an dieser Stelle nicht näher eingehen können. (Die Linup-Front-Schulungsunterlage *Linux für Fortgeschrittene* erklärt die Programmierung von Shellskripten sehr ausführlich.)

Shellskript


Sie können sich das vorgesetzte `bash` sparen, indem Sie als erste Zeile der Skriptdatei die magische Anrufung

```
#!/bin/bash
```

einfügen und die Skriptdatei »ausführbar« machen:

```
$ chmod +x meine-kommandos
```

(Mehr über `chmod` und Zugriffsrechte finden Sie in Kapitel 14.) Anschließend genügt der Aufruf

```
$ ./meine-kommandos
```

Sub-Shell

Wenn Sie ein Shellskript wie oben gezeigt aufrufen – ob mit vorgesetztem `bash` oder als ausführbare Datei –, aufrufen, wird es in einer Sub-Shell ausgeführt, also einer Shell, die ein Kindprozess der aktuellen Shell ist. Das bedeutet, dass Änderungen zum Beispiel an Shell- oder Umgebungsvariablen die aktuelle Shell nicht beeinflussen. Nehmen wir einmal an, die Datei `zuweisung` enthält die Zeile

```
bla=fasel
```

Betrachten Sie die folgende Kommandosequenz:

```
$ bla=blubb
$ bash zuweisung                                   Enthält bla=fasel
$ echo $bla
blubb                          Keine Änderung; Zuweisung war nur in Sub-Shell
```

In der Regel wird das als Vorteil empfunden, aber hin und wieder wäre es schon wünschenswert, Kommandos aus einer Datei auf die *aktuelle* Shell wirken zu lassen. Auch das funktioniert: Das Kommando `source` liest die Zeilen einer Datei so ein, als ob Sie sie direkt in die aktuelle Shell tippen würden – alle Änderungen von Variablen (unter anderem) wirken also auf Ihre aktuelle Shell:

```
$ bla=blubb
$ source zuweisung                                 Enthält bla=fasel
$ echo $bla
fasel                                          Variable wurde geändert!
```

Ein anderer Name für das Kommando `source` ist übrigens ».« (Sie haben richtig gelesen – Punkt!) Statt

```
$ source zuweisung
```

funktioniert also auch

```
$ . zuweisung
```

Wie die Programmdateien für externe Kommandos werden auch die Dateien, die mit `source` bzw. `.` gelesen werden sollen, in den Verzeichnissen gesucht, die die Variable `PATH` angibt.

9.6 Die Shell als Programmiersprache

Shellkommandos aus einer Datei ausführen zu können ist zweifellos eine gute Sache. Noch besser ist es aber, diese Shellkommandos so zu strukturieren, dass sie nicht jedesmal dasselbe tun müssen, sondern zum Beispiel Parameter von der Kommandozeile lesen können. Die Vorteile liegen auf der Hand: Bei oft gebrauchten Vorgängen wird dröge Tipperei eingespart, und bei selten gebrauchten Vorgängen vermeiden Sie Fehler, die sich einschleichen können, weil Sie irgendeinen wichtigen Schritt versehentlich auslassen. Der Platz reicht hier nicht für eine komplette Erklärung der Shell als Programmiersprache, aber für ein paar kurze Beispiele ist zum Glück Raum.

Parameter von der Kommandozeile Die Parameter von der Kommandozeile eines Shellskript-Aufrufs stellt die Shell in den Variablen `$1`, `$2`, …zur Verfügung. Betrachten Sie das folgende Beispiel: Einzelne Parameter

```
$ cat hallo
#!/bin/bash
echo Hallo $1, was machst Du $2?
$ ./hallo Hugo heute
Hallo Hugo, was machst Du heute?
$ ./hallo Susi morgen
Hallo Susi, was machst Du morgen?
```

Die Variable `$*` enthält alle Parameter auf einmal, und in `$#` steht die Anzahl der Parameter: Alle Parameter

```
$ cat parameter
#!/bin/bash
echo $# Parameter: $*
$ ./parameter
0 Parameter:
$ ./parameter Hund
1 Parameter: Hund
$ ./parameter Hund Katze Maus Baum
4 Parameter: Hund Katze Maus Baum
```

Schleifen Mit dem Kommando `for` können Sie Schleifen konstruieren, die über eine Liste von (durch Freiplatz getrennten) Wörtern laufen:

```
$ for i in 1 2 3
> do
>   echo Und $i!
> done
Und 1!
Und 2!
Und 3!
```

Hierbei nimmt die Variable `i` nacheinander jeden der aufgelisteten Werte an. Jedesmal werden die Kommandos zwischen `do` und `done` ausgeführt.

Das Ganze macht mehr Spaß, wenn die Wörter aus einer Variablen kommen:

```
$ liste='4 5 6'
$ for i in $liste
> do
>   echo Und $i!
> done
Und 4!
Und 5!
Und 6!
```

Wenn Sie das »`in` …« weglassen, läuft die Schleife über die Parameter von der Kommandozeile: Schleife über Parameter

```
$ cat sort-wc
#!/bin/bash
# Sortiere Dateien nach ihrer Zeilenzahl
for f
do
    echo `wc -l <"$f» Zeilen in $f
```

```
done | sort -n
$ ./sort-wc /etc/passwd /etc/fstab /etc/motd
```

(Das Kommando »`wc -l`« zählt die Zeilen seiner Standardeingabe oder der übergebenen Datei(en).) Beachten Sie, dass Sie die Standardausgabe der *Schleife* mit einer Pipe nach `sort` leiten können!

Fallunterscheidungen Sie können die weiter vorne gezeigten Operatoren `&&` und `||` benutzen, um bestimmte Kommandos nur unter gewissen Umständen auszuführen. Das Skript

```
#!/bin/bash
# grepcp REGEX
rm -rf backup; mkdir backup
for f in *.txt
do
    grep $1 "$f" && cp "$f" backup
done
```

zum Beispiel kopiert nur diejenigen Dateien ins Verzeichnis `backup`, deren Name auf `.txt` endet (dafür sorgt die `for`-Schleife) und die mindestens eine Zeile haben, auf die der reguläre Ausdruck passt, der als Parameter übergeben wurde.

`test` Nützlich für Fallunterscheidungen ist das Kommando `test`, das eine große Auswahl von Bedingungen überprüfen kann. Es liefert den Rückgabewert 0 (Erfolg), wenn die Bedingung zutrifft, sonst einen von Null verschiedenen Rückgabewert (Misserfolg). Betrachten Sie zum Beispiel

```
#!/bin/bash
# filetest NAME1 NAME2 ...
for name
do
   test -d "$name" && echo $name: Verzeichnis
   test -f "$name" && echo $name: Datei
   test -L "$name" && echo $name: Symbolisches Link
done
```

Dieses Skript betrachtet eine Reihe von übergebenen Dateinamen und gibt für jeden aus, ob er für ein Verzeichnis, eine (normale) Datei oder ein symbolisches Link steht.

Das Kommando `test` existiert sowohl als freistehendes Programm in `/bin/test` als auch als eingebautes Kommando in der Bash und anderen Shells. Die verschiedenen Versionen können (vor allem bei exotischeren Tests) subtil voneinander abweichen. Lesen Sie gegebenenfalls in der Dokumentation nach.

`if` Mit dem `if`-Kommando können Sie (bequem und leserlich) mehr als ein Kommando von einer Fallunterscheidung abhängig machen (Statt »`test` …« können Sie auch »`[` …`]`« schreiben):

```
#!/bin/bash
# filetest2 NAME1 NAME2 ...
for name
do
    if [ -L "$name" ]
    then
        echo $name: Symbolisches Link
    elif [ -d "$name" ]
        echo $name: Verzeichnis
```

```
    elif [ -f "$name" ]
        echo $name: Datei
    else
        echo $name: Keine Ahnung
    fi
done
```

Wenn das Kommando nach dem `if` »Erfolg« meldet (Rückgabewert 0), werden die Kommandos nach `then` ausgeführt, bis zu einem `elif`, `else` oder `fi`. Liefert es hingegen »Misserfolg«, wird als nächstes testhalber das Kommando nach dem nächsten `elif` ausgeführt und dessen Rückgabewert betrachtet. Die Shell macht entsprechend weiter, bis das passende `fi` erreicht ist, wobei die Kommandos hinter dem `else` ausgeführt werden, wenn keines der `if`-Kommandos Erfolg vermelden konnte. Die `elif`- und `else`-Zweige dürfen wegfallen, wenn sie nicht gebraucht werden.

Mehr Schleifen Bei der `for`-Schleife liegt die Anzahl der Schleifendurchläufe von Anfang an fest (die Anzahl der Wörter in der Liste). Oft bekommt man es aber mit Situationen zu tun, wo nicht *a priori* klar ist, wie oft eine Schleife durchlaufen werden soll. Hierfür bietet die Shell die `while`-Schleife an, die (ähnlich wie `if`) ein Kommando ausführt, dessen Erfolg oder Misserfolg darüber entscheidet, wie mit der Schleife verfahren wird: Bei Erfolg werden die »abhängigen« Kommandos ausgeführt, bei Misserfolg wird nach der Schleife im Skript fortgefahren. `while`

Das folgende Skript liest eine auf der Kommandozeile übergebene Datei der Form

```
Liebe Tante Frieda:frieda@example.net:den tollen Kaffeewärmer
Lieber Onkel Hans:hans@example.com:den schönen Fußball
◁◁◁◁◁
```

und konstruiert aus jeder Zeile eine Dankes-E-Mail (Linux ist halt schon sehr nützlich fürs wirkliche Leben):

```
#!/bin/bash
# birthday FILE
IFS=:
while read anrede adresse geschenk
do
    (echo $anrede!
     echo ""
     echo "Vielen Dank für $geschenk!"
     echo "Ich habe mich sehr darüber gefreut."
     echo ""
     echo "Viele Grüße,"
     echo "Dein Hugo") | mail -s "Vielen Dank!" $adresse
done <$1
```

Das Kommando `read` liest dabei die Eingabedatei Zeile für Zeile und teilt jede Zeile an den Doppelpunkten (Variable `IFS`) in die drei Felder `anrede`, `adresse` und `geschenk` auf, die im Inneren der Schleife dann als Variable zur Verfügung stehen. Die Eingabeumleitung für die Schleife steht etwas konterintuitiv ganz am Ende. `read`

Bitte testen Sie dieses Skript nur mit unverfänglichen E-Mail-Adressen!

Übungen

9.9 [1] Was ist der Unterschied (bei der Schleifenausführung) zwischen

```
for f; do …; done
```

und

```
for f in $*; do …; done
```

? (Probieren Sie es notfalls aus.)

9.10 [2] Warum benutzen wir im Skript `sort-wc` das Kommando

```
wc -l <$f
```

und nicht

```
wc -l $f
```

9.11 [2] Ändern Sie das `grepcp`-Skript so, dass es auch die Liste der zu betrachtenden Dateien von der Kommandozeile übernimmt. (*Tipp:* Das Shell-Kommando `shift` entfernt den ersten Kommandozeilenparameter aus `$` und läßt alle anderen um eine Position aufrücken. Nach einem `shift` ist das vormalige `$2` jetzt `$1`, `$3` ist `$2` und so weiter.)

9.12 [2] Warum liefert das Skript `filetest` für ein symbolisches Link die Ausgabe

```
$ ./filetest foo
foo: Datei
foo: Symbolisches Link
```

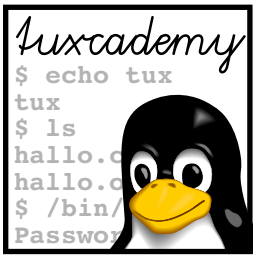
(statt nur einfach »`foo: Symbolisches Link`«)?

Kommandos in diesem Kapitel

.	Liest eine Datei mit Shell-Kommandos so ein, als ob sie auf der Kommandozeile eingegeben worden wäre	bash(1)	138
date	Gibt Datum und Uhrzeit aus	date(1)	130
env	Gibt die Prozessumgebung aus oder startet Programme mit veränderter Umgebung	env(1)	132
export	Definiert und verwaltet Umgebungsvariable	bash(1)	131
hash	Zeigt und verwaltet „gesehene“ Kommandos in der bash	bash(1)	134
set	Verwaltet Shellvariable	bash(1)	132
source	Liest eine Datei mit Shell-Kommandos so ein, als ob sie auf der Kommandozeile eingegeben worden wäre	bash(1)	138
test	Wertet logische Ausdrücke auf der Kommandozeile aus	test(1), bash(1)	140
unset	Löscht Shell- oder Umgebungsvariable	bash(1)	132
whereis	Sucht ausführbare Programme, Handbuchseiten und Quellcode zu gegebenen Kommandos	whereis(1)	134
which	Sucht Programme in PATH	which(1)	134

Zusammenfassung

- Das Kommando `sleep` wartet für die als Parameter angegebene Anzahl von Sekunden.
- Das Kommando `echo` gibt seine Argumente aus.
- Mit `date` lassen sich Datum und Uhrzeit ermitteln.
- Verschiedene Eigenschaften der Bash unterstützen das interaktive Arbeiten, etwa Kommando- und Dateinamenvervollständigung, Editieren der Kommandozeile, Aliasnamen und Variable.



10

Das Dateisystem

Inhalt

Lernziele

- Die Begriffe »Datei« und »Dateisystem« verstehen
- Die verschiedenen Dateitypen kennen
- Sich im Verzeichnisbaum eines Linux-Systems zurechtfinden
- Wissen, wie externe Dateisysteme in den Verzeichnisbaum eingebunden werden

Vorkenntnisse

- Linux-Grundkenntnisse (etwa aus den vorherigen Kapiteln)
- Umgang mit Dateien und Verzeichnissen (Kapitel 6)

10.1 Begriffe

Datei Der Begriff **Datei** steht ganz allgemein für eine abgeschlossene Ansammlung von Daten. Für die Art der enthaltenen Daten gibt es keine Einschränkungen; eine Datei kann ein Text sein, der nur wenige Buchstaben lang ist, aber auch ein viele Megabyte großes Archiv, das das gesamte Lebenswerk eines Anwenders umfaßt. Dateien müssen natürlich nicht unbedingt gewöhnlichen Text enthalten. Bilder, Klänge, ausführbare Programme und vieles andere mehr werden ebenfalls in Form von Dateien auf dem Datenträger abgelegt. Um herauszufinden, welche Art
file von Inhalt eine Datei hat, können Sie den Befehl `file` verwenden:

```
$ file /bin/ls /usr/bin/groups /etc/passwd
/bin/ls:         ELF 32-bit LSB executable, Intel 80386,▷
 ◁ version 1 (SYSV), for GNU/Linux 2.4.1,▷
 ◁ dynamically linked (uses shared libs), for GNU/Linux 2.4.1, stripped
/usr/bin/groups: Bourne shell script text executable
/etc/passwd:     ASCII text
```

`file` errät den Typ einer Datei auf der Basis von Regeln, die im Verzeichnis `/usr/share/file` stehen. `/usr/share/file/magic` enthält die Regeln im Klartext. Sie können eigene Regeln definieren, wenn Sie sie in `/etc/magic` ablegen. Näheres verrät `magic`(5).

Zum ordnungsgemäßen Betrieb benötigt ein Linux-System einige tausend verschiedene Dateien. Hinzu kommen noch die von den verschiedenen Benutzern angelegten »eigenen« Dateien.

Dateisystem Ein **Dateisystem** legt fest, nach welcher Methode die Daten auf den Datenträgern angeordnet und verwaltet werden. Auf der Platte liegen ja letzten Endes nur Bytes, die das System irgendwie wiederfinden können muss – und das möglichst effizient, flexibel und auch für sehr große Dateien. Die Details der Dateiverwaltung können unterschiedlich ausgelegt sein (Linux kennt zahlreiche verschiedene Dateisysteme, etwa `ext2`, `ext3`, `ext4`, ReiserFS, XFS, JFS, btrfs, …), aber die logische Sicht auf die Dateien, die die Benutzer zu sehen bekommen, ist im Großen und Ganzen dieselbe: eine baumartige Hierarchie von Datei- und Verzeichnisnamen mit Dateien unterschiedlicher Typen. (Siehe hierzu auch Kapitel 6.)

Der Begriff »Dateisystem« wird in der Linux-Szene in mehreren Bedeutungen verwendet. »Dateisystem« ist außer der in diesem Abschnitt eingeführten Bedeutung »Methode, Bytes auf einem Medium zu arrangieren« für manche Leute auch das, was wir als »Verzeichnisbaum« bezeichnen, außerdem nennt man ein konkretes Medium (Festplatte, USB-Stick, …) mitsamt den darauf befindlichen Daten »Dateisystem« – etwa in dem Sinn, dass man sagt, dass harte Links (Abschnitt 6.4.2) nicht »über Dateisystemgrenzen hinweg«, also nicht zwischen zwei verschiedenen Partitionen auf der Festplatte oder zwischen der Platte und einem USB-Stick, funktionieren.

10.2 Dateitypen

In Linux-Systemen gilt der Grundsatz: »Alles ist eine Datei«. Dies mag zunächst verwirrend scheinen, ist aber sehr nützlich. Prinzipiell können sechs Dateitypen unterschieden werden:

Normale Dateien (engl. *plain files*) Zu dieser Gruppe gehören Texte, Grafiken, Audiodaten etc., aber auch ausführbare Programme. Normale Dateien können mit den üblichen Werkzeugen (Editoren, `cat`, Shell-Ausgabeumlenkung, …) erzeugt werden.

Tabelle 10.1: Linux-Dateitypen

Typ	`ls -l`	`ls -F`	Anlegen mit …
Normale Datei	`-`	`name`	Diverse Programme
Verzeichnis	`d`	`name/`	`mkdir`
Symbolisches Link	`l`	`name@`	`ln -s`
Gerätedatei	`b` oder `c`	`name`	`mknod`
FIFO (*named pipe*)	`p`	`name|`	`mkfifo`
Unix-Domain-Socket	`s`	`name=`	kein Kommando

Verzeichnisse (engl. *directories)* Auch »Ordner« genannt; sie dienen, wie bereits beschrieben, zur Strukturierung des Speicherplatzes. Ein Verzeichnis ist im Prinzip eine Tabelle mit der Zuordnung von Dateinamen zu Inode-Nummern. Verzeichnisse werden mit `mkdir` angelegt.

Symbolische Links Enthalten eine Pfadangabe, die bei Verwendung des Links auf eine andere Datei verweist (ähnlich zu »Verknüpfungen« unter Windows). Siehe auch Abschnitt 6.4.2. Symbolische Links werden mit `ln -s` angelegt.

Gerätedateien (engl. *devices*) Diese Dateien entsprechen Schnittstellen zu beliebigen Geräten wie etwa Laufwerken. So repräsentiert etwa die Datei `/dev/fd0` das erste Diskettenlaufwerk. Jeder Schreib- oder Lesezugriff auf eine solche Datei wird an das zugehörige Gerät weitergeleitet. Gerätedateien werden mit dem Kommando `mknod` angelegt; dies ist normalerweise Territorium des Systemadministrators und wird in dieser Unterlage darum nicht weiter erklärt.

FIFOs Oft auch *named pipes* genannt. Sie erlauben ähnlich wie die Pipes der Shell (Kapitel 8) die direkte Kommunikation zwischen Programmen ohne Verwendung von Zwischendateien: Ein Prozess öffnet den FIFO zum Schreiben und ein anderer zum Lesen. Im Gegensatz zu den Pipes, die die Shell für ihre Pipelines benutzt und die sich zwar aus der Sicht von Programmen wie Dateien benehmen, aber »anonym« sind – sie existieren nicht im Dateisystem, sondern nur zwischen Prozessen, die in einem Verwandtschaftsverhältnis stehen –, haben FIFOs Dateinamen und können darum von beliebigen Programmen wie Dateien geöffnet werden. Außerdem können für FIFOs Zugriffsrechte gesetzt werden (für Pipes nicht). FIFOs werden mit dem Kommando `mkfifo` angelegt.

Unix-Domain-Sockets Ähnlich wie FIFOs sind Unix-Domain-Sockets ein Mittel zur Interprozesskommunikation. Sie verwenden im Wesentlichen dieselbe Programmierschnittstelle wie »echte« Netzwerkkommunikation über TCP/IP, aber funktionieren nur, wenn die Kommunikationspartner auf demselben Rechner laufen. Dafür sind Unix-Domain-Sockets beträchtlich effizienter als TCP/IP. Im Gegensatz zu FIFOs erlauben Unix-Domain-Sockets bidirektionale Kommunikation – beide beteiligten Programme können sowohl Daten lesen als auch schreiben. Unix-Domain-Sockets werden zum Beispiel vom Grafiksystem X11 verwendet, wenn X-Server und -Clients auf demselben Rechner laufen. – Zum Anlegen von Unix-Domain-Sockets gibt es kein spezielles Programm.

Übungen

10.1 [3] Suchen Sie in Ihrem System nach Beispielen für die verschiedenen Dateitypen. (Tabelle 10.1 zeigt Ihnen, woran Sie die betreffenden Dateien erkennen können.)

```
$ cd /
$ ls -l
insgesamt 125
drwxr-xr-x    2 root  root    4096 Dez 20 12:37 bin
drwxr-xr-x    2 root  root    4096 Jan 27 13:19 boot
lrwxrwxrwx    1 root  root      17 Dez 20 12:51 cdrecorder▷
                                                  ◁ -> /media/cdrecorder
lrwxrwxrwx    1 root  root      12 Dez 20 12:51 cdrom -> /media/cdrom
drwxr-xr-x   27 root  root   49152 Mär  4 07:49 dev
drwxr-xr-x   40 root  root    4096 Mär  4 09:16 etc
lrwxrwxrwx    1 root  root      13 Dez 20 12:51 floppy -> /media/floppy
drwxr-xr-x    6 root  root    4096 Dez 20 16:28 home
drwxr-xr-x    6 root  root    4096 Dez 20 12:36 lib
drwxr-xr-x    6 root  root    4096 Feb  2 12:43 media
drwxr-xr-x    2 root  root    4096 Mär 21  2002 mnt
drwxr-xr-x   14 root  root    4096 Mär  3 12:54 opt
dr-xr-xr-x   95 root  root       0 Mär  4 08:49 proc
drwx------   11 root  root    4096 Mär  3 16:09 root
drwxr-xr-x    4 root  root    4096 Dez 20 13:09 sbin
drwxr-xr-x    6 root  root    4096 Dez 20 12:36 srv
drwxrwxrwt   23 root  root    4096 Mär  4 10:45 tmp
drwxr-xr-x   13 root  root    4096 Dez 20 12:55 usr
drwxr-xr-x   17 root  root    4096 Dez 20 13:02 var
```

Bild 10.1: Inhalt des Wurzelverzeichnisses (SUSE)

10.3 Der Linux-Verzeichnisbaum

Ein Linux-System besteht oft aus Hunderttausenden von Dateien. Um einen Überblick zu behalten, gibt es gewisse Konventionen für die Verzeichnisstruktur und die Dateien, die ein Linux-System ausmachen, den *Filesystem Hierarchy Standard*, kurz »**FHS**«. Die meisten Distributionen halten sich an diesen Standard, allerdings sind kleinere Abweichungen möglich. Der FHS beschreibt alle Verzeichnisse der ersten Hierarchie-Ebene, außerdem definiert er eine zweite Ebene unterhalb von `/usr`.

FHS

Wurzelverzeichnis

Der Verzeichnisbaum beginnt mit dem **Wurzelverzeichnis** »/«. (Zur Unterscheidung: es gibt auch ein Verzeichnis `/root` – das ist das Heimatverzeichnis des Benutzers `root`.) Das Wurzelverzeichnis enthält entweder nur Unterverzeichnisse oder aber zusätzlich, wenn kein spezielles Verzeichnis `/boot` existiert, den Betriebssystemkern.

Mit dem Befehl »`ls -la /`« können Sie sich im Wurzelverzeichnis `/` die Unterverzeichnisse auflisten lassen. Das Ergebnis sieht beispielsweise so aus wie in Bild 10.1. Die einzelnen Verzeichnisse folgen dem FHS und haben daher in allen Distributionen weitestgehend den gleichen Inhalt. Im folgenden werden die einzelnen Verzeichnisse genauer betrachtet.

Über den FHS besteht weithin Einigkeit, aber er ist genausowenig »verbindlich« wie irgend etwas bei Linux verbindlich ist. Zum einen gibt es sicherlich Linux-Systeme (etwa das auf Ihrer FRITZ!Box oder in Ihrem digitalen Videorecorder), die sowieso im wesentlichen nur der Hersteller anfasst und wo es nichts bringt, den FHS bis ins kleinste Detail einzuhalten. Zum Anderen können Sie auf Ihrem System natürlich machen, was Sie wollen, aber müssen gegebenenfalls die Konsequenzen tragen – Ihr Distributor sichert Ihnen zu, dass er sich an seinen Teil des FHS hält, aber erwartet auf der anderen Seite, dass Sie sich nicht beschweren, wenn Sie nicht 100% nach den Regeln spielen und dann Probleme auftauchen. Wenn Sie zum Beispiel ein Programm in `/usr/bin` installieren und die betreffende Datei beim

nächsten System-Upgrade überschrieben wird, sind Sie selber schuld, weil Sie laut FHS Ihre eigenen Programme nicht nach `/usr/bin` schreiben sollen (`/usr/local/bin` wäre richtig).

Der Betriebssystemkern – `/boot` Im Verzeichnis `/boot` liegt das Betriebssystem im engeren Sinne: `vmlinuz` ist der Kernel von Linux. Im Verzeichnis `/boot` finden sich außerdem Dateien, die für den Bootlader (meist GRUB) von Bedeutung sind.

Bei manchen Systemen befindet sich das Verzeichnis `/boot` auf einer separaten Partition. Das kann nötig sein, wenn das eigentliche Dateisystem verschlüsselt oder anderweitig für den Bootlader schwer zugänglich ist, etwa weil besondere Treiber für den Zugriff auf ein Hardware-RAID-System gebraucht werden.

Allgemeine Systemprogramme – `/bin` Unter `/bin` befinden sich die wichtigsten ausführbaren Programme (meist Systemprogramme), die unbedingt zum Starten des Systems notwendig sind. Dazu gehören z. B. `mount` und `mkdir`. Viele dieser Programme sind so elementar, dass sie nicht nur zum Starten, sondern auch während des Systembetriebs ständig gebraucht werden – etwa `ls` oder `grep`. In `/bin` stehen außerdem Programme, die nötig sind, um ein System wieder flott zu machen, bei dem nur das Dateisystem mit dem Wurzelverzeichnis zur Verfügung steht. Weitere Programme, die beim Start oder zur Reparatur nicht unbedingt gebraucht werden, finden sich auch unter `/usr/bin`.

Spezielle Systemprogramme – `/sbin` Ähnlich wie `/bin` enthält auch `/sbin` Programme, die zum Systemstart oder für Reparaturen nötig sind. Allerdings handelt es sich hierbei zum größten Teil um Systemkonfigurationswerkzeuge, die eigentlich nur `root` ausführen kann. »Normale« Benutzer können mit manchen dieser Programme Informationen abfragen, aber nichts ändern. Analog zu `/bin` gibt es auch ein Verzeichnis `/usr/sbin`, wo weitere systemnahe Programme zu finden sind.

Systembibliotheken – `/lib` Hier finden sich die *shared libraries* der Programme in `/bin` und `/sbin` in Form von Dateien und (symbolischen) Links. Shared Libraries sind Programmstücke, die von verschiedenen Programmen gebraucht werden. Solche gemeinsam verwendeten Bibliotheken sparen eine Menge Systemressourcen, da die meisten Prozesse teilweise gleiche Bestandteile haben und diese Bestandteile dann nur einmal geladen werden müssen; ferner ist es einfacher, Fehler in solchen Bibliotheken zu korrigieren, wenn es sie nur einmal im System gibt und alle Programme den betreffenden Programmcode aus einer zentralen Datei holen. Unter `/lib/modules` liegen übrigens auch die **Kernelmodule**, also Systemkern-Programmcode, der nicht notwendigerweise benötigt wird – Gerätetreiber, Dateisysteme, Netzwerkprotokolle und ähnliches. Diese Module können vom Systemkern bei Bedarf nachgeladen und prinzipiell nach Gebrauch auch wieder entfernt werden.

Kernelmodule

Gerätedateien – `/dev` In diesem Verzeichnis und seinen Unterverzeichnissen findet sich eine Unmenge von Einträgen für die Gerätedateien. **Gerätedateien** bilden die Schnittstelle von der Shell (oder allgemein dem Teil des Systems, den Benutzer auf der Kommandozeile oder als Programmierer zu sehen bekommen) zu den Gerätetreibern im Systemkern. Sie haben keinen »Inhalt« wie andere Dateien, sondern verweisen über »Gerätenummern« auf einen Treiber im Systemkern.

Gerätedateien

Früher war es üblich, dass Linux-Distributoren für jedes nur denkbare Gerät einen Dateieintrag in `/dev` machten. So hatte auch ein auf einem Notebook installiertes Linux-System die erforderlichen Gerätedateien für zehn Festplatten mit je 63 Partitionen, acht ISDN-Adapter, sechzehn serielle und vier parallele Schnittstellen und so weiter. Heutzutage geht der Trend weg von den übervollen `/dev`-Verzeichnissen mit einem Eintrag für jedes vorstellbare Gerät und hin zu enger an den laufenden Kernel gekoppelten Systemen, wo nur Einträge für tatsächlich existierende Geräte erscheinen. Das Stichwort

in diesem Zusammenhang heißt udev (kurz für *userspace* /dev) und wird in *Linux-Administration I* genauer besprochen.

zeichenorientierte Geräte
blockorientierte Geräte

Linux unterscheidet zwischen **zeichenorientierten Geräten** (engl. *character devices*) und **blockorientierten Geräten** (engl. *block devices*). Ein zeichenorientiertes Gerät ist beispielsweise ein Terminal, eine Maus oder ein Modem – ein Gerät, das einzelne Zeichen liefert oder verarbeitet. Ein blockorientiertes Gerät ist ein Gerät, das Daten blockweise behandelt – hierzu gehören zum Beispiel Festplatten oder Disketten, auf denen Sie Bytes nicht einzeln lesen oder schreiben können, sondern nur in Gruppen à 512 (oder so). Je nach ihrer Geschmacksrichtung sind die Gerätedateien in der Ausgabe von »ls -l« mit einen »c« oder einem »b« gekennzeichnet:

```
crw-rw-rw-  1  root  root  10,  4 Oct 16 11:11 amigamouse
brw-rw----  1  root  disk   8,  1 Oct 16 11:11 sda1
brw-rw----  1  root  disk   8,  2 Oct 16 11:11 sda2
crw-rw-rw-  1  root  root   1,  3 Oct 16 11:11 null
```

Treibernummer
Gerätenummer

Anstelle der Speichergröße stehen hier zwei Zahlen: Die erste ist die **Treibernummer** (engl. *major device number*). Sie kennzeichnet den Gerätetyp und legt fest, welcher Treiber im Kernel für die Verwaltung zuständig ist. So haben zum Beispiel alle SCSI-Festplatten traditionell die Treibernummer 8. Die zweite Zahl ist die **Gerätenummer** (engl. *minor device number*). Sie dient dem Treiber zur Unterscheidung verschiedener ähnlicher oder verwandter Geräte oder auch zur Kennzeichnung verschiedener Partitionen einer Platte.

Pseudogeräte

Erwähnenswert sind noch ein paar **Pseudogeräte**. Das *null device*, /dev/null, ist quasi ein Mülleimer für Ausgaben eines Programmes, die nicht gebraucht werden, aber irgendwohin geleitet werden müssen. Bei einem Aufruf wie

```
$ programm >/dev/null
```

wird die Standardausgabe, die sonst auf dem Terminal erscheinen würde, verworfen. Wird /dev/null gelesen, reagiert es wie eine leere Datei und liefert sofort das Dateiende. Für /dev/null müssen alle Benutzer Schreib- und Leserechte haben.

Die »Geräte« /dev/random bzw. /dev/urandom liefern zufällige Bytes in »kryptographischer« Qualität, die erzeugt werden, indem »Rauschen« im System gesammelt wird – etwa die Zeitabstände zwischen dem Eintreffen unvorhersehbarer Ereignisse wie Tastendrücke. Die Daten aus /dev/random eignen sich zur Erzeugung von Schlüsseln für gängige Verschlüsselungsverfahren. Die Datei /dev/zero liefert Nullbytes in beliebiger Menge; Sie können diese unter anderem zum Erzeugen und zum Überschreiben von Dateien mit dem Befehl dd verwenden.

Konfigurationsdateien – /etc Das Verzeichnis /etc ist sehr wichtig, denn hier befinden sich die Konfigurationsdateien für die allermeisten Programme. In /etc/inittab und /etc/init.d beispielsweise stehen die meisten der systemspezifischen Daten, die zum Starten von Systemdiensten erforderlich sind. Die wichtigsten Dateien werden hier etwas detaillierter erklärt. Mit wenigen Ausnahmen hat nur der Benutzer root Schreibrechte, aber jeder Benutzer Leserechte.

/etc/fstab Hier sind alle einhängbaren Dateisysteme mit ihren Eigenschaften (Typ, Zugriffsart, *mount point*) aufgelistet.

/etc/hosts Diese Datei ist eine der Konfigurationsdateien des TCP/IP-Netzwerks. Hier werden die Namen der Netzwerkrechner ihren IP-Adressen zugeordnet. In kleinen Netzwerken und bei Einzelrechnern kann diese Datei einen Name-Server ersetzen.

/etc/inittab Die Datei /etc/inittab ist die Konfigurationsdatei für das init-Programm und damit für den Systemstart.

/etc/init.d In diesem Verzeichnis liegen die »Init-Skripte« für die verschiedenen Systemdienste. Mit ihnen werden beim Systemstart und beim Herunterfahren die Dienste gestartet bzw. gestoppt.


Bei den Red-Hat-Distributionen heißt dieses Verzeichnis `/etc/rc.d/init.d`.

/etc/issue In der Datei `/etc/issue` steht der Begrüßungstext, der vor der Aufforderung zum Anmelden ausgegeben wird. Nach der Installation eines neuen Systems wird in diesem Text meistens der Name des Herstellers präsentiert.

/etc/motd Hier steht die »Nachricht des Tages« (engl. *message of the day*), die nach einer erfolgreichen Anmeldung automatisch auf dem Bildschirm erscheint, noch bevor die Shell die erste Eingabeaufforderung ausgibt. Diese Datei kann der Systemadministrator verwenden, um aktuelle Informationen und Neuigkeiten weiterzugeben[1].

/etc/mtab Dies ist eine Liste aller eingehängten Dateisysteme inklusive ihrer *mount points*. `/etc/mtab` unterscheidet sich von `/etc/fstab` darin, dass in `/etc/mtab` alle aktuell eingehängten Dateisysteme aufgezählt sind, während in `/etc/fstab` nur Voreinstellungen und Optionen dafür stehen, welche Dateisysteme wie eingehängt werden *können* – typischerweise beim Systemstart, aber auch später. Sie können natürlich über die Kommandozeile beliebige Dateisysteme einhängen können, wo Sie wollen, und das wird hier auch protokolliert.


Eigentlich gehört es sich nicht, diese Sorte Information in `/etc` abzulegen, wo die Dateien prinzipiell statisch sein sollen. Hier hat offensichtlich die Tradition die Oberhand gewonnen.

/etc/passwd In `/etc/passwd` findet sich eine Liste aller dem System bekannten Benutzer zusammen mit diversen anderen benutzerspezifischen Informationen. In modernen Systemen befinden sich übrigens trotz des Namens dieser Datei die Kennwörter nicht hier, sondern in der Datei `/etc/shadow`. Jene Datei ist für normale Benutzer nicht lesbar.

Zubehör – /opt Dieses Verzeichnis ist eigentlich dafür gedacht, dass Drittanbieter fertige Softwarepakete anbieten können, die sich installieren lassen sollen, ohne mit den Dateien einer Linux-Distribution oder den lokal angelegten Dateien zu kollidieren. Solche Softwarepakete belegen ein Unterverzeichnis `/opt/`⟨*Paketname*⟩. Von Rechts wegen sollte dieses Verzeichnis unmittelbar nach der Installation einer Distribution auf einer neuen Platte also völlig leer sein.

»Unveränderliche Dateien« – /usr In `/usr` finden sich in diversen Unterverzeichnissen Programme und Dateien, die nicht für den Systemstart oder zur Systemreparatur notwendig oder anderweitig unverzichtbar sind. Die wichtigsten Verzeichnisse sind:

/usr/bin Systemprogramme, die nicht für den Systemstart gebraucht werden und/oder anderweitig nicht so wichtig sind.

/usr/sbin Weitere Systemprogramme für `root`.

/usr/lib Weitere, nicht von Programmen in `/bin` oder `/sbin` benötigte Bibliotheken.

/usr/local Verzeichnis für Dateien, die der lokale Systemadministrator installiert hat. Entspricht von der Idee her dem `/opt`-Verzeichnis – die Distribution darf hier nichts ablegen.

[1] Man sagt, dass die einzige Gemeinsamkeit aller Unix-Systeme der Welt die *message of the day* ist, die darauf hinweist, dass die Platten zu 98% voll sind und die Benutzer überflüssige Dateien entsorgen mögen.

/usr/share Daten, die von der Rechnerarchitektur unabhängig sind. Im Prinzip könnte ein Linux-Netz, das z. B. aus Intel-, SPARC- und PowerPC-Rechnern besteht, sich eine gemeinsame Kopie von `/usr/share` auf einem zentralen Rechner teilen. Heute ist Plattenplatz aber so billig, dass keine Distribution sich die Mühe macht, das tatsächlich zu implementieren.

/usr/share/doc Dokumentation – zum Beispiel HOWTOs

/usr/share/info Info-Seiten

/usr/share/man Handbuchseiten (in Unterverzeichnissen)

/usr/src Quellcode für den Kernel und weitere Programme (sofern vorhanden)

Der Name `/usr` wird häufig als »*Unix system resources*« interpretiert, was aber historisch nicht stimmt: Ursprünglich stammt dieses Verzeichnis aus der Zeit, als in einem Rechner eine kleine schnelle und eine große langsame Festplatte zur Verfügung stand. Auf die kleine Festplatte kamen alle häufig verwendeten Programme und Dateien, auf die große langsame (unter `/usr` eingehängt) große Programme und Dateien, die nicht recht auf die kleine Platte passten, oder solche, die nicht so oft benötigt wurden. Heute können Sie die Trennung anders ausnutzen: Wenn Sie es geschickt anstellen, können Sie `/usr` auf eine eigene Partition legen und diese schreibgeschützt in den Verzeichnisbaum einbinden. Es ist grundsätzlich sogar möglich, `/usr` von einem zentralen Server zu importieren, und so auf Arbeitsplatzrechnern Plattenplatz zu sparen und die Wartung zu vereinfachen (nur der zentrale Server muss gegebenenfalls aktualisiert werden), auch wenn die gesunkenen Plattenpreise das heute nicht mehr nötig machen. Die gängigen Linux-Distributionen unterstützen es sowieso nicht.

`/usr` schreibgeschützt

Das Fenster zum Kernel – `/proc` Das ist mit das interessanteste Verzeichnis und auch eines der wichtigsten. `/proc` ist eigentlich ein Pseudo-Dateisystem. Es belegt keinen Platz auf der Festplatte, sondern die Verzeichnisse und Dateien werden vom Systemkern erzeugt, wenn sich jemand für ihren Inhalt interessiert. Hier finden Sie sämtliche Informationen zu den laufenden Prozessen und außerdem weitere Informationen, die der Kernel über die Hardware des Rechners besitzt. In einigen Dateien finden Sie zum Beispiel eine komplette Hardwareanalyse. Die wichtigsten Dateien sind kurz aufgeführt:

Pseudo-Dateisystem

/proc/cpuinfo Hier sind Informationen über den Typ und die Taktfrequenz der CPU enthalten.

/proc/devices Hier findet sich eine vollständige Liste der Geräte, die vom Kernel unterstützt werden mit deren Gerätenummern. Beim Erstellen der Gerätedateien wird auf diese Datei zugegriffen.

/proc/dma Eine Liste der belegten DMA-Kanäle. Ist auf heutigen PCI-basierten Systemen nicht mehr fürchterlich interessant oder wichtig.

/proc/interrupts Eine Liste aller belegten Hardwareinterrupts. Interruptnummer, Anzahl der bisher ausgelösten Interrupts und die Bezeichnung der möglichen auslösenden Geräte sind angegeben. (Ein Interrupt taucht in dieser Liste nur auf, wenn er wirklich von einem Treiber im Kernel beansprucht wird.)

/proc/ioports Ähnlich wie `/proc/interrupts`, aber für I/O-Ports.

/proc/kcore Diese Datei fällt ins Auge wegen ihrer Größe. Sie ist der Zugang zum gesamten Arbeitsspeicher des Rechners, quasi ein Abbild des RAM, und wird zum Debugging des Systemkerns gebraucht. Diese Datei kann nur mit `root`-Privilegien gelesen werden. Am besten lassen Sie die Finger davon!

/proc/loadavg Diese Datei gibt drei Zahlen aus, die ein Maß für die Auslastung der CPU innerhalb der letzten 1, 5 und 15 Minuten sind. Diese Zahlen werden normalerweise vom Programm `uptime` ausgegeben.

/proc/meminfo Zeigt die Speicherauslastung und die Auslastung des Swap-Bereichs an. Diese Datei wird vom Kommando `free` benutzt.

/proc/mounts Wieder eine Liste aller aktuell gemounteten Dateisysteme, weitestgehend identisch mit `/etc/mtab`.

/proc/scsi In diesem Verzeichnis findet man eine Datei mit dem Namen `scsi`, in der die erkannten Geräte aufgelistet sind. Dann gibt es ein Unterverzeichnis für jeden Typ von SCSI-Hostadapter im System, in dem in einer Datei `0` (bzw. `1`, `2` usw. bei mehreren gleichen Adaptern) Informationen über den Adapter gespeichert sind.

/proc/version Enthält die Versionsnummer und das Übersetzungsdatum des laufenden Kernels.

Früher, bevor es `/proc` gab, mussten Programme wie das Prozessstatus-Anzeigeprogramm `ps`, die Systeminformationen aus dem Kern liefern, einiges über die internen Datenstrukturen des Linux-Kerns wissen und brauchten auch entsprechende Zugriffsrechte, um die betreffenden Daten aus dem laufenden Kern zu lesen. Da die Datenstrukturen sich im Zuge der Linux-Weiterentwicklung öfters änderten, war es oft nötig, zu einer neuen Kern-Version auch neue Versionen dieser Dienstprogramme zu installieren, die an die Änderungen angepasst waren. Das `/proc`-Dateisystem liefert eine Abstraktionsschicht zwischen diesen internen Datenstrukturen und den Dienstprogrammen: Heuer müssen Sie nur noch sicherstellen, dass bei einer Änderung einer internen Datenstruktur das Format der Dateien in `/proc` gleich bleibt – und `ps` & Co. funktionieren weiter wie gehabt.

Hardwaresteuerung – /sys Dieses Verzeichnis finden Sie im Linux-Kernel ab der Version 2.6. Es wird ähnlich wie `/proc` vom Kernel selbst nach Bedarf zur Verfügung gestellt und erlaubt in einer umfangreichen Hierarchie von Unterverzeichnissen eine konsistente Sicht auf die verfügbare Hardware sowie diverse Steuerungseingriffe.

Tendenziell sollen alle Einträge von `/proc`, die nichts mit einzelnen Prozessen zu tun haben, allmählich nach `/sys` wandern. Allerdings wissen die Götter, wann dieses strategische Ziel erreicht sein wird.

Veränderliche Dateien – /var Hier befinden sich veränderliche Dateien, verteilt auf verschiedene Verzeichnisse. Beim Aufruf verschiedener Programme generiert der Benutzer, meist ohne sich dessen im Detail bewusst zu sein, Daten. Das geschieht etwa beim Aufruf von `man`, bei dem komprimierte Hilfedaten entpackt werden (und die entpackten Versionen eine Weile aufgehoben werden, nur falls sie gleich wieder gebraucht werden). Ähnliches erfolgt, wenn gedruckt werden soll: Der Druckauftrag muss zwischengespeichert werden, zum Beispiel unter `/var/spool/cups`. Unter `/var/log` werden An- und Abmeldevorgänge sowie weitere Systemereignisse protokolliert (die Log-Dateien), unter `/var/spool/cron` stehen Informationen für das regelmäßige zeitgesteuerte Starten von Kommandos, und ungelesene elektronische Post der Benutzer steht in `/var/mail`.

Log-Dateien

Nur damit Sie es mal gehört haben (es könnte in der Prüfung vorkommen): Das Systemprotokoll wird auf Linux-Rechnern normalerweise über den »Syslog«-Dienst geregelt. Ein Programm namens `syslogd` nimmt Nachrichten von anderen Programmen entgegen und sortiert diese anhand ihrer Herkunft und Priorität (von »Debugginghilfe« über »Fehler« bis hin zu »Katastrophe, System schmiert gerade ab«) in Dateien in `/var/log` ein, wo

Sie sie dann finden können. Außer in Dateien kann der Syslog-Dienst seine Nachrichten auch anderswohin schicken, etwa auf die Konsole oder über das Netz an einen anderen Rechner, der als zentrale Management-Station arbeitet und alle Protokollnachrichten aus Ihrem Rechenzentrum konsolidiert.

Außer dem `syslogd` gibt es bei manchen Linux-Distributionen auch noch einen `klogd`-Dienst. Seine Aufgabe ist es, Nachrichten vom Betriebssystemkern entgegenzunehmen und an den `syslogd` weiterzureichen. Andere Distributionen kommen ohne einen separaten `klogd` aus, weil deren `syslogd` diese Aufgabe selbst übernehmen kann.

Der Linux-Betriebssystemkern spuckt schon jede Menge Nachrichten aus, bevor das System überhaupt so weit ist, dass `syslogd` (und gegebenenfalls `klogd`) läuft und diese Nachrichten tatsächlich entgegennehmen kann. Da die Nachrichten trotzdem wichtig sein können, speichert der Linux-Kern sie intern ab, und Sie können mit dem Kommando `dmesg` darauf zugreifen.

Vergängliche Daten – `/tmp` Viele Dienstprogramme brauchen temporären Speicherplatz, zum Beispiel manche Editoren oder `sort`. In `/tmp` können beliebige Programme temporäre Dateien ablegen. Viele Distributionen löschen beim Booten zumindest wahlweise alle Dateien unterhalb dieses Verzeichnisses; Sie sollten dort also nichts unterbringen, was Ihnen dauerhaft wichtig ist.

Nach Konvention wird `/tmp` beim Booten geleert und `/var/tmp` nicht. Ob Ihre Distribution das auch so macht, sollten Sie bei Gelegenheit prüfen.

Serverdateien – `/srv` Hier finden Sie Dateien, die von verschiedenen Dienstprogrammen angeboten werden, etwa:

```
drwxr-xr-x    2 root     root         4096 Sep 13 01:14 ftp
drwxr-xr-x    5 root     root         4096 Sep  9 23:00 www
```

Dieses Verzeichnis ist eine relativ neue Erfindung, und es ist durchaus möglich, dass es auf Ihrem System noch nicht existiert. Leider gibt es keinen guten anderen Platz für Web-Seiten, ein FTP-Angebot oder ähnliches, über den man sich einig werden konnte (letzten Endes ja der Grund für die Einführung von `/srv`), so dass auf einem System ohne `/srv` diese Daten irgendwo ganz anders liegen können, etwa in Unterverzeichnissen von `/usr/local` oder `/var`.

Zugriff auf CD-ROMs und Disketten – `/media` Dieses Verzeichnis wird oftmals automatisch angelegt; es enthält weitere, leere Verzeichnisse, etwa `/media/cdrom` und `/media/floppy`, die als Mountpunkt für CD-ROMs und Floppies dienen. Je nach Ihrer Ausstattung mit Wechselmedien sollten Sie sich keinen Zwang antun, neue Verzeichnisse wie `/media/dvd` anzulegen, wenn diese als Mountpoints sinnvoll und nicht von Ihrem Distributionshersteller vorgesehen sind.

Zugriff auf andere Datenträger – `/mnt` Dieses ebenfalls leere Verzeichnis dient zum kurzfristigen Einbinden weiterer Dateisysteme. Bei manchen Distributionen, etwa denen von Red Hat, können sich hier (und nicht in `/media`) Verzeichnisse als Mountpunkte für CD-ROM, Floppy, … befinden.

Heimatverzeichnisse für Benutzer – `/home` Unter diesem Eintrag befinden sich die Heimatverzeichnisse aller Benutzer mit Ausnahme von `root`, der ein eigenes Verzeichnis hat.

Wenn Sie mehr als ein paar hundert Benutzer haben, ist es aus Datenschutz- und Effizienzgründen sinnvoll, die Heimatverzeichnisse nicht alle als unmittelbare Kinder von `/home` zu führen. Sie können in so einem Fall zum

Tabelle 10.2: Zuordnung einiger Verzeichnisse zum FHS-Schema

	statisch	dynamisch
lokal	`/etc`, `/bin`, `/sbin`, `/lib`	`/dev`, `/var/log`
entfernt	`/usr`, `/opt`	`/home`, `/var/mail`

Beispiel die primäre Gruppe der Benutzer als weiteres Unterscheidungskriterium zu Rate ziehen:

```
/home/support/hugo
/home/entwickl/emil
◁◁◁◁◁
```

Heimatverzeichnis des Administrators – `/root` Hierbei handelt es sich um ein ganz normales Heimatverzeichnis ähnlich denen der übrigen Benutzer. Der entscheidende Unterschied ist, dass es nicht im Ordner `/home`, sondern im Wurzelverzeichnis (`/`) liegt.

Der Grund dafür ist, dass das Verzeichnis `/home` oftmals auf einem Dateisystem auf einer separaten Partition oder Festplatte liegt, aber es soll sichergestellt sein, dass `root` auch in seiner gewohnten Umgebung arbeiten kann, wenn dieses separate Dateisystem einmal nicht angesprochen werden kann.

Das virtuelle Fundbüro – `lost+found` (Nur bei `ext`-Dateisystemen; nicht vom FHS vorgeschrieben.) Hier werden Dateien gespeichert, die ein Dateisystemcheck nach einem Systemabsturz auf der Platte findet und die zwar vernünftig aussehen, aber in keinem Verzeichnis zu stehen scheinen. Das Prüfprogramm legt in `lost+found` auf demselben Dateisystem Links auf solche Dateien an, damit der Systemverwalter sich anschauen kann, wo die Datei in Wirklichkeit hingehören könnte; `lost+found` wird schon »auf Vorrat« bereitgestellt, damit das Prüfprogramm es an einer festen Stelle finden kann (es hat nach Konvention auf den `ext`-Dateisystemen immer die Inode-Nummer 11).

Eine andere Motivation für die Verzeichnisanordnung ist wie folgt: Der FHS unterteilt Dateien und Verzeichnisse grob nach zwei Kriterien – Müssen sie lokal verfügbar sein oder können sie auch über das Netz von einem anderen Rechner bezogen werden, und sind ihre Inhalte statisch (Veränderung nur durch Intervention des Administrators) oder ändern sie sich im laufenden Betrieb? (Tabelle 10.2)

Die Idee hinter dieser Einteilung ist es, die Pflege des Systems zu vereinfachen: Verzeichnisse können leicht auf Datei-Server ausgelagert und damit zentral administriert werden. Verzeichnisse, die keine dynamischen Daten enthalten, können nur zum Lesen eingebunden werden und sind damit absturzresistenter.

Übungen

10.2 [1] Wie viele Programme enthält Ihr System an den »gängigen« Plätzen?

10.3 [2] Wird `grep` mit mehr als einem Dateinamen als Parameter aufgerufen, gibt es vor jeder passenden Zeile den Namen der betreffenden Datei aus. Dies ist möglicherweise ein Problem, wenn man `grep` mit einem Shell-Dateisuchmuster (etwa »`*.txt`«) aufruft, da das genaue Format der `grep`-Ausgabe so nicht vorhersehbar ist und Programme weiter hinten in einer Pipeline deswegen durcheinander kommen können. Wie können Sie die Ausgabe des Dateinamens erzwingen, selbst wenn das Suchmuster nur zu einem

einzigen Dateinamen expandiert? (*Tipp:* Eine dafür nützliche »Datei« steht im Verzeichnis `/dev`.)

10.4 [3] Das Kommando »`cp bla.txt /dev/null`« tut im Wesentlichen nichts, aber das Kommando »`mv bla.txt /dev/null`« – entsprechende Zugriffsrechte vorausgesetzt – ersetzt `/dev/null` durch `bla.txt`. Warum?

10.5 [2] Welche Softwarepakete stehen auf Ihrem System unter `/opt`? Welche davon stammen aus der Distribution und welche von Drittanbietern? Sollte eine Distribution eine »Schnupperversion« eines Drittanbieter-Programms unter `/opt` installieren oder anderswo? Was halten Sie davon?

10.6 [1] Warum ist es unklug, Sicherheitskopien des Verzeichnisses `/proc` anzulegen?

10.4 Verzeichnisbaum und Dateisysteme

Der Verzeichnisbaum eines Linux-Systems erstreckt sich normalerweise über mehr als eine Partition auf der Festplatte, und auch Wechselmedien wie CD-ROMs, USB-Sticks und ähnliches sowie tragbare MP3-Player, Digitalkameras und ähnliche für einen Computer wie Speichermedien aussehende Geräte müssen berücksichtigt werden. Wenn Sie Microsoft Windows ein bisschen kennen, dann wissen Sie vielleicht, dass dieses Problem dort so gelöst wird, dass die verschiedenen »Laufwerke« über Buchstaben identifiziert werden – bei Linux dagegen werden alle verfügbaren Plattenpartitionen und Medien in den Verzeichnisbaum eingegliedert, der bei »/« anfängt.

Partitionierung

Grundsätzlich spricht nichts Gravierendes dagegen, ein Linux-System komplett auf einer einzigen Plattenpartition zu installieren. Es ist allerdings gängig, zumindest das Verzeichnis `/home` – in dem die Heimatverzeichnisse der Benutzer liegen – auf eine eigene Partition zu tun. Dies hat den Vorteil, dass Sie das eigentliche Betriebssystem, die Linux-Distribution, komplett neu installieren können, ohne um die Sicherheit Ihrer eigenen Daten fürchten zu müssen (Sie müssen nur im richtigen Moment aufpassen, nämlich wenn Sie in der Installationsroutine die Zielpartition(en) für die Installation auswählen). Auch die Erstellung von Sicherheitskopien wird so vereinfacht.

Serversysteme

Auf größeren Serversystemen ist es auch Usus, anderen Verzeichnissen, typischerweise `/tmp`, `/var/tmp` oder `/var/spool`, eigene Partitionen zuzuteilen. Das Ziel ist dabei, dass Benutzer nicht den Systembetrieb dadurch stören können sollen, dass sie wichtige Partitionen komplett mit Daten füllen. Wenn zum Beispiel `/var` voll ist, können keine Protokolldaten mehr geschrieben werden, also möchte man vermeiden, dass Benutzer das Dateisystem mit Unmengen ungelesener Mail, ungedruckten Druckjobs oder riesigen Dateien in `/var/tmp` blockieren. Allerdings wird das System durch so viele Partitionen auch unübersichtlich und unflexibel.

Mehr über die Partitionierung und Strategien dafür finden Sie in der Linup-Front-Schulungsunterlage *Linux-Administration I*.

`/etc/fstab`

Die Datei `/etc/fstab` beschreibt die Zusammensetzung des Systems aus verschiedenen Partitionen. Beim Systemstart wird dafür gesorgt, dass die unterschiedlichen Dateisysteme an den richtigen Stellen »eingebunden« – der Linux-Insider sagt »gemountet«, vom englischen *to mount* – werden, worum Sie als einfacher Benutzer sich nicht kümmern müssen. Was Sie aber möglicherweise interessiert, ist, wie Sie an den Inhalt Ihrer CD-ROMs und USB-Sticks kommen, und auch diese müssen Sie einhängen. Wir tun also gut daran, uns kurz mit dem Thema zu beschäftigen, auch wenn es eigentlich Administratoren-Territorium ist.

Zum Einhängen eines Mediums benötigen Sie ausser dem Namen der Gerätedatei des Mediums (in der Regel ein blockorientiertes Gerät wie `/dev/sda1`) ein Verzeichnis irgendwo im Verzeichnisbaum, wo der Inhalt des Mediums erscheinen soll – den sogenannten *mount point*. Dabei kann es sich um jedes beliebige Verzeichnis handeln.

Das Verzeichnis muss dabei nicht einmal leer sein, allerdings können Sie auf den ursprünglichen Verzeichnisinhalt nicht mehr zugreifen, wenn Sie einen Datenträger »darübergemountet« haben. (Der Inhalt erscheint wieder, wenn Sie den Datenträger wieder aushängen.)

Grundsätzlich könnte jemand ein Wechselmedium über ein wichtiges Systemverzeichnis wie `/etc` mounten (idealerweise mit einer Datei namens `passwd`, die einen `root`-Eintrag ohne Kennwort enthält). Aus diesem Grund ist das Einhängen von Dateisystemen an beliebigen Stellen im Dateisystem mit Recht dem Systemverwalter vorbehalten, der keinen Bedarf für solche Sperenzchen haben dürfte; er ist ja schon `root`.

Die »Gerätedatei für das Medium« haben wir weiter oben `/dev/sda1` genannt. Dies ist eigentlich die erste Partition auf der ersten SCSI-Festplatte im System – der Name kann völlig anders lauten, je nachdem was Sie mit welcher Sorte Medium vor haben. Es ist aber ein naheliegender Name für USB-Sticks, die vom System aus technischen Gründen so behandelt werden, als wären sie SCSI-Geräte.

Mit diesen Informationen – Gerätename und *mount point* – kann ein Systemadministrator das Medium etwa wie folgt einbinden:

```
# mount /dev/sda1 /media/usb
```

Eine Datei namens `datei` auf dem Medium würde dann als `/media/usb/datei` im Verzeichnisbaum in Erscheinung treten. Mit einem Kommando wie

```
# umount /media/usb                                    Achtung: kein »n«
```

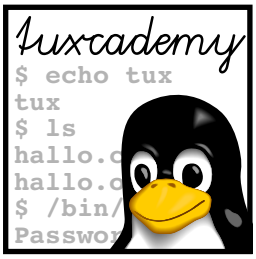
kann der Administrator diese Einbindung auch wieder lösen.

Kommandos in diesem Kapitel

dmesg	Gibt den Inhalt des Kernel-Nachrichtenpuffers aus	dmesg(8)	152
file	Rät den Typ einer Datei anhand des Inhalts	file(1)	144
free	Zeigt die Speicherauslastung und die Auslastung des Swap-Bereichs an	free(1)	151
klogd	Akzeptiert Protokollnachrichten des Systemkerns	klogd(8)	152
mkfifo	Legt FIFOs (benannte Pipes) an	mkfifo(1)	145
mknod	Legt Gerätedateien an	mknod(1)	145
syslogd	Bearbeitet Systemprotokoll-Meldungen	syslogd(8)	152
uptime	Gibt die Zeit seit dem letzten Systemstart sowie die CPU-Auslastung aus	uptime(1)	150

Zusammenfassung

- Dateien sind abgeschlossene Ansammlungen von Daten, die unter einem Namen gespeichert sind. Linux verwendet die Abstraktion »Datei« auch für Geräte und andere Objekte.
- Die Methode der Anordnung von Daten und Verwaltungsinformationen auf einem Speichermedium nennt man Dateisystem. Derselbe Begriff bezeichnet die gesamte baumartige Hierarchie von Verzeichnissen und Dateien im System oder ein konkretes Medium mit den darauf befindlichen Daten.
- Linux-Dateisysteme enthalten normale Dateien, Verzeichnisse, symbolische Links, Gerätedateien (zwei Sorten), FIFOs und Unix-Domain-Sockets.
- Der *Filesystem Hierarchy Standard* (FHS) beschreibt die Bedeutung der wichtigsten Verzeichnisse in einem Linux-System und wird von den meisten Distributionen eingehalten.



11

Dateien archivieren und komprimieren

Inhalt

Lernziele

- Die Begriffe »Archivierung« und »Komprimierung« verstehen
- Mit `tar` umgehen können
- Dateien mit `gzip` und `bzip2` komprimieren und entkomprimieren können
- Dateien mit `zip` und `unzip` verarbeiten können

Vorkenntnisse

- Arbeit mit der Shell (Kapitel 4)
- Umgang mit Dateien und Verzeichnissen (Kapitel 6)
- Umgang mit Filtern (Kapitel 8)

11.1 Archivierung und Komprimierung

»Archivierung« ist der Prozess des Zusammenfassens vieler Dateien zu einer einzigen. Die klassische Anwendung ist das Ablegen eines Verzeichnisbaums auf Magnetband – das Magnetbandlaufwerk erscheint unter Linux als Gerätedatei, auf die die Ausgabe eines Archivierprogramms geschrieben werden kann. Entsprechend können Sie mit einem »Entarchivierungsprogramm« von der Gerätedatei des Bandlaufwerks lesen und diese Daten dann wieder als Verzeichnisbaum abspeichern. Da die meisten entsprechenden Programme sowohl Dateien zusammenfassen als auch Archive wieder aufdröseln können, fassen wir beides unter dem Begriff der Archivierung zusammen.

»Komprimierung« ist das Umschreiben von Daten in eine gegenüber dem Original platzsparende Fassung. Wir interessieren uns hier nur für »verlustfreie« Komprimierung, bei der es möglich ist, aus den komprimierten Daten das Original in identischer Form wieder herzustellen.

Die Alternative besteht darin, einen höheren Komprimierungsgrad zu erreichen, indem man darauf verzichtet, das Original absolut identisch wiedergewinnen zu können. Diesen »verlustbehafteten« Ansatz verfolgen Komprimierungsverfahren wie JPEG für Fotos und »MPEG-1 Audio Layer 3« (besser bekannt als »MP3«) für Audiodaten. Die Kunst besteht hier darin, unnötige Details zu verwerfen; bei MP3 wird zum Beispiel auf der Basis eines »psychoakustischen Modells« des menschlichen Gehörs auf Teile des Signals verzichtet, die der Hörer sowieso nicht wahrnimmt, und der Rest möglichst effizient kodiert. JPEG arbeitet ähnlich.

Lauflängenkodierung

Als einfache Illustration könnten Sie eine Zeichenkette der Form

```
ABBBBAACCCCCAAAABAAAAAC
```

kompakter als

```
A*4BAA*5C*4AB*5AC
```

darstellen. Hierbei steht »`*4B`« für eine Folge von vier »`B`«. Dieses simple Verfahren heißt »Lauflängenkodierung« (engl. *run-length encoding*) und findet sich heute noch zum Beispiel (mit Änderungen) in Faxgeräten. Die »echten« Komprimierprogramme wie `gzip` oder `bzip2` verwenden aufwendigere Methoden.

Während in der Windows-Welt Programme gebräuchlich sind, die Archivierung und Komprimierung kombinieren (PKZIP, WinZIP & Co.), werden die beiden Schritte bei Linux und Unix üblicherweise getrennt. Eine gängige Vorgehensweise besteht darin, eine Menge von Dateien zunächst mit `tar` zu archivieren und die Ausgabe von `tar` dann zum Beispiel mit `gzip` zu komprimieren – PKZIP & Co. komprimieren jede Datei einzeln und fassen die komprimierten Dateien dann zu einer großen zusammen.

Der Vorteil dieses Ansatzes gegenüber dem von PKZIP und seinen Verwandten besteht darin, dass eine Komprimierung über mehrere der Originaldateien hinweg möglich ist und so höhere Komprimierungsraten erzielt werden können. Genau dasselbe ist aber auch ein Nachteil: Wenn das komprimierte Archiv beschädigt wird (etwa durch ein schadhaftes Medium oder gekippte Bits bei der Übertragung), kann das komplette Archiv ab dieser Stelle unbenutzbar werden.

Natürlich hält Sie auch unter Linux niemand davon ab, Ihre Dateien *erst* zu komprimieren und sie dann zu archivieren. Leider ist das nicht so bequem wie die umgekehrte Vorgehensweise.

Selbstverständlich existieren auch für Linux Implementierungen der gängigen Komprimierungs- und Archivprogramme der Windows-Welt, etwa `zip` und `rar`.

Übungen

11.1 [1] Warum verwendet das Beispiel für die Lauflängenkodierung `AA` statt `*2A`?

11.2 [2] Wie würden Sie mit dem skizzierten Verfahren zur Lauflängenkodierung die Zeichenkette »`A*2B****A`« darstellen?

11.2 Dateien archivieren mit `tar`

Der Name `tar` leitet sich von *tape archive* (Bandarchiv) ab. Einzelne Dateien werden hintereinander in eine Archivdatei gepackt und mit Zusatzinformationen (etwa Datum, Zugriffsrechte, Eigentümer, …) versehen. Obwohl `tar` ursprünglich für den Einsatz mit Bandlaufwerken konzipiert wurde, können `tar`-Archive direkt auf die unterschiedlichsten Medien geschrieben werden. `tar`-Dateien sind unter anderem der Standard für die Verbreitung von Linux-Programmquellcodes und anderer freier Software.

Das unter Linux eingesetzte GNU-`tar` ist gegenüber den `tar`-Implementierungen anderer Unix-Varianten stark erweitert. Mit GNU-`tar` können beispielsweise *multi-volume archives* erstellt werden, die sich über mehrere Datenträger erstrecken. Prinzipiell sind somit sogar Sicherungskopien auf Disketten möglich, was sich allerdings nur bei kleineren Archiven lohnt.

multi-volume archives

Eine kleine Anmerkung am Rande: Mit dem Kommando `split` ist es ebenfalls möglich, große Dateien wie etwa Archive in »handliche« Teile zu zerschneiden, diese auf Disketten zu kopieren oder per Mail zu verschicken und am Zielort mit `cat` wieder zusammenzufügen.

Die Vorteile von `tar` sind: Die Anwendung ist einfach, es ist zuverlässig und läuft stabil, es ist universell einsetzbar auf allen Unix- und Linux-Systemen. Nachteilig ist: Fehlerhafte Stellen des Datenträgers können zu Problemen führen, und Gerätedateien können nicht mit jeder Version von `tar` archiviert werden (was nur dann von Bedeutung ist, wenn Sie eine Komplettsicherung Ihres ganzen Systems machen wollen).

In `tar`-Archive lassen sich Dateien und ganze Verzeichnisbäume einpacken. Wenn in einem Netzwerk Windows-Datenträger in den Verzeichnisbaum eingehängt sind, können sogar diese Inhalte mit `tar` gesichert werden. Die mit `tar` angelegten Archive sind normalerweise nicht komprimiert, lassen sich aber zusätzlich durch gängige Kompressionsprogramme (heute in der Regel `gzip` oder `bzip2`) verdichten. Im Falle von Sicherheitskopien ist dies jedoch keine gute Idee, da Bitfehler in den komprimierten Archiven normalerweise zum Verlust des Rests des Archivs führen.

Die typischen Endungen für `tar`-Archive sind `.tar`, `.tar.bz2` oder `.tar.gz`, je nachdem, ob sie gar nicht, mit `bzip2` oder mit `gzip` komprimiert sind. Auch die Endung `.tgz` ist üblich, um gezippte Dateien im `tar`-Format auch in einem DOS-Dateisystem speichern zu können. Die Syntax von `tar` ist

```
tar ⟨Optionen⟩ ⟨Datei⟩|⟨Verzeichnis⟩ …
```

und die wichtigsten Optionen sind:

`tar`-Optionen

`-c` erzeugt (engl. *create*) ein neues Archiv

`-f` ⟨*Datei*⟩ erzeugt oder liest das Archiv von ⟨*Datei*⟩, wobei dies eine Datei (engl. *file*) oder ein Gerät sein kann

`-M` bearbeitet ein `tar`-Archiv, das sich über mehrere Datenträger erstreckt (*multi-volume archive*)

-r hängt Dateien an das Archiv an (nicht für Magnetbänder)

-t zeigt den Inhalt (engl. *table of contents*) des Archivs

-u ersetzt Dateien, die neuer als eine bereits archivierte Version sind. Wenn eine Datei noch nicht archiviert ist, so wird sie eingefügt (nicht für Magnetbänder)

-v ausführlicher Modus (engl. *verbose*, geschwätzig); zeigt auf dem Bildschirm an, was gerade geschieht

-x Auslesen (engl. *extract*) der gesicherten Dateien

-z komprimiert oder dekomprimiert das Archiv mit `gzip`

-Z komprimiert oder dekomprimiert das Archiv mit `compress` (unter Linux normalerweise nicht vorhanden)

-j komprimiert oder dekomprimiert das Archiv mit `bzip2`

Optionssyntax

Die Optionssyntax von `tar` ist etwas ungewöhnlich, da es (wie anderswo) möglich ist, mehrere Optionen hinter einem Minuszeichen zu bündeln, und zwar (ausgefallenerweise) auch solche wie `-f`, die einen Parameter nach sich ziehen. Die Parameter müssen hinter dem »Bündel« angegeben werden und werden der Reihe nach den entsprechenden Optionen im Bündel zugeordnet.

Sie können das Minuszeichen vor dem ersten »Optionsbündel« auch weglassen – Sie werden oft Kommandos sehen wie

```
tar cvf …
```

Wir empfehlen Ihnen das jedoch nicht.

Das folgende Beispiel archiviert alle Dateien im aktuellen Verzeichnis, deren Namen mit `data` beginnen, in die Archivdatei `data.tar` im Heimatverzeichnis des Benutzers:

```
$ tar -cvf ~/data.tar data*
data1
data10
data2
data3
◁◁◁◁◁
```

Das `-c` steht dafür, dass das Archiv neu erzeugt werden soll, »`-f ~/data.tar`« für den Namen, unter dem das Archiv angelegt werden soll. Die Option `-v` ändert am Ergebnis nichts. Sie bewirkt nur, dass der Anwender auf dem Bildschirm eine Übersicht über den Ablauf erhält. (Sollte eine der zu archivierenden Dateien ein Verzeichnis sein, wird außer dem Verzeichnis selbst auch der komplette Verzeichnisinhalt erfasst.)

Verzeichnisse

Mit `tar` können auch ganze Verzeichnisse archiviert werden. Es ist besser, dies vom übergeordneten Verzeichnis aus durchzuführen. Dadurch werden die zu archivierenden Dateien gleich in einem Verzeichnis abgelegt und auch wieder als Verzeichnis extrahiert. Das folgende Beispiel erklärt das genauer.

```
# cd /
# tar -cvf /tmp/home.tar /home
```

Der Systemadministrator `root` legt ein Archiv des Verzeichnisses `/home` (also alle Benutzerdaten) unter dem Namen `home.tar` an. Dieses wird im Verzeichnis `/tmp` abgespeichert.

Wenn Dateien oder Verzeichnisse mit absoluten Pfadnamen angegeben werden, sichert tar diese automatisch als relative Pfadnamen (mit anderen Worten, der »/« am Anfang wird entfernt). Dies beugt Problemen beim Auspacken auf anderen Rechnern vor (siehe Übung 11.6).

Das »Inhaltsverzeichnis« eines Archivs können Sie mit der Option -t anzeigen:

```
$ tar -tf data.tar
data1
data10
data2
◁◁◁◁◁
```

Mit der Option -v ist tar etwas mitteilsamer:

```
$ tar -tvf data.tar
-rw-r--r-- hugo/hugo     7 2009-01-27 12:04 data1
-rw-r--r-- hugo/hugo     8 2009-01-27 12:04 data10
-rw-r--r-- hugo/hugo     7 2009-01-27 12:04 data2
◁◁◁◁◁
```

Auspacken können Sie die Daten mit der Option -x:

```
$ tar -xf data.tar
```

Hierbei produziert tar überhaupt keine Ausgabe auf dem Terminal – dazu müssen Sie wieder -v angeben:

```
$ tar -xvf data.tar
data1
data10
data2
◁◁◁◁◁
```

Wenn das Archiv eine Verzeichnishierarchie enthält, wird diese Hierarchie im aktuellen Verzeichnis originalgetreu wieder aufgebaut. (Sie erinnern sich, tar macht automatisch aus allen absoluten Pfadnamen relative.) Sie können das Archiv relativ zu jedem beliebigen Verzeichnis auspacken – es behält immer seine Struktur.

Sie können auch beim Auspacken Datei- oder Verzeichnisnamen angeben. In diesem Fall werden nur die betreffenden Dateien oder Verzeichnisse ausgepackt. Allerdings müssen Sie darauf achten, die Schreibweise der betreffenden Namen im Archiv genau zu treffen:

```
$ tar -cf data.tar ./data
$ tar -tvf data.tar
drwxr-xr-x hugo/hugo     0 2009-01-27 12:04 ./data/
-rw-r--r-- hugo/hugo     7 2009-01-27 12:04 ./data/data2
◁◁◁◁◁
$ mkdir data-neu
$ cd data-neu
$ tar -xvf ../data.tar data/data2                                    ./ fehlt
tar: data/data2: Not found in archive
tar: Error exit delayed from previous errors
```

Übungen

11.3 [!2] Legen Sie das Inhaltsverzeichnis Ihres Heimatverzeichnisses in einer Datei `inhalt` ab. Erzeugen Sie aus dieser Datei ein `tar`-Archiv. Betrachten Sie die Ursprungsdatei und das Archiv. Was bemerken Sie?

11.4 [2] Legen Sie drei bis vier leere Dateien an und fügen Sie diese dem gerade erzeugten Archiv hinzu.

11.5 [2] Löschen Sie die Originaldateien und entpacken Sie danach den Inhalt des `tar`-Archivs.

11.6 [2] Warum entfernt GNU `tar` prophylaktisch das `/` am Anfang des Pfadnamens, wenn der Name einer zu archivierenden Datei oder eines Verzeichnisses als absoluter Pfadname angegeben wurde? (*Tipp:* Betrachten Sie das Kommando

```
# tar -cvf /tmp/etc-backup.tar /etc
```

und überlegen Sie, was passiert, wenn `etc-backup.tar` (a) absolute Pfadnamen enthält und (b) auf einen anderen Rechner übertragen und dort ausgepackt wird.)

11.3 Dateien komprimieren mit `gzip`

Das gängigste Komprimierungsprogramm für Linux ist `gzip` von Jean-loup Gailly und Mark Adler. Es dient dazu, einzelne Dateien zu komprimieren (die, wie weiter vorne diskutiert, Archive sein können, die wiederum viele Dateien enthalten).

Das Programm `gzip` (kurz für »GNU `zip`«) wurde 1992 veröffentlicht, um Problemen mit dem Programm `compress` aus dem Weg zu gehen, das bei proprietären Unix-Versionen Standard war. `compress` beruht auf dem Lempel-Ziv-Welch-Algorithmus (LZW), der unter der Nummer 4,558,302 in den USA patentiert war. Das Patent gehörte der Firma Sperry (später Unisys) und lief am 20. Juni 2003 ab. `gzip` verwendet dagegen das DEFLATE-Verfahren von Phil Katz [RFC1951], das auf einem nicht patentierten Vorläufer von LZW namens LZ77 nebst dem Huffman-Kodierungsverfahren aufbaut und von Patentansprüchen frei ist. Außerdem funktioniert es besser als LZW.

Grundidee

Die Grundidee hinter Verfahren wie LZ77 ist, dass man versucht, Muster in der Eingabe zu erkennen und statt dem kompletten Muster nur noch den Namen des Musters vermerkt. Die komprimierte Datei besteht dann also im Wesentlichen aus einem Verzeichnis der erkannten Muster und einer Liste der erkannten Muster in ihrer tatsächlichen Abfolge; in dem Moment, wo möglichst lange Muster möglichst oft auftreten, hat man den größten Erfolg beim Komprimieren. Entscheidend für die Laufzeit und den Speicherverbrauch beim Komprimieren ist, wieviel Mühe man sich bei der Suche nach Mustern gibt und wieviel Platz man sich für das Musterverzeichnis gönnt – ein großes Musterverzeichnis verbessert potentiell die Komprimierung, aber muss natürlich auch effizient durchsucht werden können.

`gzip` kann mit `compress` komprimierte Dateien *entkomprimieren*, da das Unisys-Patent nur das Komprimieren abdeckte. Solche Dateien erkennen Sie an der Endung »`.Z`« ihrer Dateinamen.

`gzip` ist nicht zu verwechseln mit PKZIP und ähnlichen Programmen für Windows mit »ZIP« im Namen. Diese Programme können Dateien komprimieren und anschließend gleich archivieren; `gzip` kümmert sich nur um die Komprimierung und überläßt das Archivieren Programmen wie `tar` oder `cpio`. – `gzip` kann ZIP-Archive auspacken, solange das Archiv genau eine Datei enthält und diese mit dem DEFLATE-Verfahren komprimiert ist.

`gzip` bearbeitet und ersetzt einzelne Dateien, wobei an den Dateinamen jeweils die Endung `.gz` angehängt wird. Diese Substitution erfolgt unabhängig davon, ob die resultierende Datei tatsächlich kleiner als die ursprüngliche ist. Sollen mehrere Dateien komprimiert in einem einzigen Archiv gespeichert werden, müssen `tar` und `gzip` kombiniert werden.

Die wichtigsten Optionen von `gzip` sind:

`-c` schreibt die komprimierte Datei auf die Standardausgabe, anstatt die Datei zu ersetzen; die Originaldatei bleibt erhalten

`-d` dekomprimiert die Datei (alternativ: `gunzip` arbeitet wie `gzip -d`)

`-l` zeigt (engl. *list*) wichtige Verwaltungsinformationen der komprimierten Datei, wie Dateiname, ursprüngliche und gepackte Größe an

`-r` packt auch Dateien in darunterliegenden Verzeichnissen (engl. *recursive*)

`-S` ⟨*Suffix*⟩ verwendet anstelle von `.gz` die angegebene Endung

`-v` gibt den Namen und den Kompressionsfaktor für jede Datei aus

`-1 … -9` gibt einen Kompressionsfaktor an; `-1` (oder `--fast`) arbeitet am schnellsten, komprimiert aber nicht so gründlich, während `-9` (oder `--best`) die beste Komprimierung um den Preis höherer Laufzeit liefert; voreingestellt ist `-6`

Der folgende Befehl komprimiert die Datei `brief.tex`, speichert die komprimierte Datei unter `brief.tex.gz` und löscht die Originaldatei:

```
$ gzip brief.tex
```

Entpackt wird mit:

```
$ gzip -d brief.tex
```

oder

```
$ gunzip brief.tex
```

Hier wird die komprimierte Datei unter `brief.tex.t` statt unter `brief.tex.gz` gespeichert (`-S .t`) und der erreichte Kompressionsfaktor ausgegeben (`-v`):

```
$ gzip -vS .t brief.tex
```

Entsprechend muss die Option `-S` auch wieder beim Entpacken angegeben werden, da `gzip -d` eine Datei mit der Endung `.gz` erwartet:

```
$ gzip -dS .t brief.tex
```

Sollen alle Dateien mit der Endung `.tex` in einer Datei `tex-all.tar.gz` komprimiert werden, dann sieht das so aus:

```
# tar -cvzf tex-all.tar.gz *.tex
```

Erinnern Sie sich daran, dass `tar` die Originaldateien nicht löscht! Entpackt wird mit:

```
# tar -xvzf tex-all.tar.gz
```

Übungen

11.7 [2] Komprimieren Sie das `tar`-Archiv aus Übung 11.5 mit maximaler Verkleinerung.

11.8 [!3] Sehen Sie sich den Inhalt des komprimierten Archivs an. Stellen Sie das ursprüngliche `tar`-Archiv wieder her.

11.9 [!2] Wie gehen Sie vor, wenn Sie den gesamten Inhalt Ihres Heimatverzeichnisses in eine mit `gzip` komprimierte Datei packen möchten?

11.4 Dateien komprimieren mit `bzip2`

`bzip2` von Julian Seward ist ein Komprimierungsprogramm, das weitgehend kompatibel zu `gzip` ist. Es verwendet jedoch ein anderes Verfahren, das zu einer höheren Komprimierung führt, aber mehr Zeit und Speicherplatz zum Komprimieren benötigt (beim Entkomprimieren ist der Unterschied nicht so groß).

Wenn Sie es dringend wissen müssen: `bzip2` verwendet eine »Burrows-Wheeler-Transformation«, um häufig auftretende Teilzeichenketten in der Eingabe zu Folgen einzelner Zeichen zu machen. Das Zwischenergebnis wird nochmals anhand der »lokalen Häufigkeit« der einzelnen Zeichen umsortiert und das Resultat dieser Sortierung nach einer Lauflängenkodierung schließlich mit dem Huffman-Verfahren kodiert. Der Huffman-Code wird dann noch besonders platzsparend in eine Datei geschrieben.

Was ist mit `bzip`? `bzip` war ein Vorläufer von `bzip2`, der nach der Blocktransformation statt der Huffman-Kodierung eine arithmetische Kodierung einsetzte. Da es rund um arithmetische Kodierung aber jede Menge Softwarepatente gibt, nahm man von diesem Verfahren wieder Abstand.

Wie `gzip` akzeptiert `bzip2` einen oder mehrere Dateinamen als Parameter zur Komprimierung. Die Dateien werden jeweils durch die komprimierte Version ersetzt, deren Name auf `.bz2` endet.

Die Optionen `-c` und `-d` entsprechen den gleichnamigen Optionen von `gzip`. Anders benehmen sich allerdings die »Qualitätsoptionen« `-1` bis `-9`: Sie bestimmen die Blockgröße, mit der `bzip2` bei der Komprimierung arbeitet. Der Standardwert ist `-9`, während `-1` keinen signifikanten Geschwindigkeitsvorteil bietet.

`-9` verwendet eine Blockgröße von 900 KiB. Dies entspricht einem Speicherplatzverbrauch von etwa 3,7 MiB zum Entkomprimieren (7,6 MiB zum Komprimieren), was auf heutigen Rechnern kein Hinderungsgrund mehr sein dürfte. Eine weitere Erhöhung der Blockgröße bringt keinen merklichen Vorteil mehr. – Es ist hervorhebenswert, dass die Wahl der Blockgröße bei der *Komprimierung* über den Speicherplatzbedarf bei der *Entkomprimierung* entscheidet, was Sie bedenken sollten, wenn Sie auf Ihrem Multi-Gibibyte-PC `.bz2`-Dateien für Rechner mit extrem wenig Speicher (Toaster, Set-Top-Boxen, …) erstellen. `bzip2`(1) erklärt dies detaillierter.

In Analogie zu `gzip` und `gunzip` dient `bunzip2` zum Entkomprimieren von mit `bzip2` komprimierten Dateien. (Eigentlich ist das nur ein anderer Name für das `bzip2`-Programm; Sie können auch »`bzip2 -d`« verwenden, um Dateien zu entkomprimieren.)

11.5 Dateien komprimieren mit `xz`

Für alle, denen `gzip` und `bzip2` noch nicht genug Auswahlmöglichkeiten einräumen, gibt es neuerdings auch noch das Programm `xz`.

xz taucht erstmals in der Version 4.0 der LPI-Prüfung 101 auf. Warum es dringend nötig ist, noch ein Programm in der Prüfung zu haben, das sich nicht wirklich substanziell von den anderen unterscheidet, weiß wohl bloß das LPI.

Eigenschaften

xz bietet noch bessere Komprimierung als bzip2, allerdings bezahlen Sie dafür mit wiederum längeren Komprimierungszeiten (die Entkomprimierung ist vergleichbar schnell). Damit bietet xz sich vor allem für die Software-Verteilung an, wo Quellcodes oder Binärpakete typischerweise einmal zusammengepackt, aber oft übers Internet übertragen und wieder ausgepackt werden.

Seit 2014 steht der Linux-Quellcode in einer xz-komprimierten Version zur Verfügung. Auch viele Distributionen verwenden xz bei der Erstellung von Softwarepaketen, zum Beispiel Debian, Fedora, openSUSE oder Arch Linux.

LZMA-Verfahren

xz verwendet das LZMA-Verfahren von Igor Pavlov, dem Autor des 7-Zip-Komprimierungsprogramms. »LZMA« steht für »Lempel-Ziv-Markov«. Wie gzip beruht LZMA ursprünglich auf dem LZ77-Verfahren, verwendet also ein Musterverzeichnis; der Unterschied zwischen LZ77 und seinen anderen Nachfahren wie DEFLATE und LZMA ist, dass die Ausgabe des verzeichnisorientierten Algorithmus bei LZMA noch einmal mit Hilfe eines komplexen probabilistischen Modells kodiert wird (das »M« wie »Markov« in LZMA).

LZMA2

Tatsächlich benutzt xz das »LZMA2«-Verfahren, das heißt aber nichts anderes als dass für Stücke der Eingabe entschieden wird, ob die LZMA-komprimierte Fassung oder die ursprüngliche unkomprimierte Fassung kürzer ist, und das jeweils Kürzere tatsächlich in die Ausgabe geschrieben wird. Manche Datenformate (etwa JFIF, vulgo »JPEG«, oder MP3) sind nämlich schon so gut komprimiert, dass die Ausgabe fast aussieht wie zufällig ausgewürfelte Bits und mit einem allgemein anwendbaren Programm wie xz dort nicht mehr viel zu holen ist; bevor xz also das Ganze verschlimmbessert, läßt es die Daten lieber (aus seiner Sicht) »unkomprimiert«.

Bedienung

Von der Bedienung her unterscheidet xz sich in seinen Grundzügen nicht von gzip und bzip2. Genau wie bei den anderen Programmen gibt es das Kommando xz zum Komprimieren und das Kommando unxz zum Entkomprimieren[1], aber »xz -d« entkomprimiert auch.

Effizienzparameter

Mehr als bei gzip und bzip2 ist der »Effizienzparameter« -1 …-9 entscheidend für die Leistung des Programms. Wie bei den beiden anderen Programmen bedeuten größere Zahlen (jedenfalls prinzipiell) bessere Komprimierung. Die Standardvoreinstellung -6 ist für die meisten Anwendungen brauchbar und benötigt 94 MiB virtuellen Speicher zum Komprimieren. Die Größe des Musterverzeichnisses beträgt dabei 8 MiB. Zum Vergleich: Mit der Option -9 wird ungefähr 675 MiB virtueller Speicher zum Komprimieren gebraucht, und das Musterverzeichnis ist 64 MiB groß. (Eine genaue Tabelle steht in xz(1).)

Es bringt nichts, ein Musterverzeichnis zu verwenden, das größer ist als die zu komprimierende Datei. Die Optionen ab -7 (mit einem Musterverzeichnis von 16 MiB) lohnen sich also nur, wenn die Eingabe ziemlich groß ist.

Zum Entkomprimieren braucht xz Speicher im Umfang von etwa einem Zehntel des Speichers, den es zum Komprimieren benötigt. Das heißt, mit der Option -9 beim Komprimieren sind zum Entkomprimieren rund 65 MiB Hauptspeicher fällig. Auf älteren Systemen oder Raspberry Pis kann das schon störend werden.

Sie sollten jedenfalls nicht wie bei gzip und bzip2 blindlings -9 für alles benutzen.

[1] Von der eigentlich naheliegenden Idee, das Entkomprimierungsprogramm xunz zu nennen, hat der Autor anscheinend Abstand genommen; ist vielleicht auch besser so.


Die Option `-e` (wie »extrem«) macht die Komprimierung langsamer und vielleicht noch ein bisschen besser – oder auch nicht. Sie müssen selber ausprobieren, ob das für Ihre Anwendung etwas bringt.

Weitere Optionen

Die Option `-q` unterdrückt routinemäßige Meldungen und Warnungen. (`-qq` unterdrückt sogar Fehlermeldungen; Sie müssen sich also den Rückgabewert des Programms anschauen, um festzustellen, ob alles geklappt hat.) Mit `-v` bekommen Sie eine Fortschrittsausgabe.


Sie können `tar` auch dazu bringen, seine Ausgabe direkt mit `xz` zu komprimieren. Die dafür nötige Option ist `-J`.

Übungen


11.10 [3] Besorgen Sie sich eine geeignete Datei (etwa ein größeres `tar`-Archiv mit Quellcode, vielleicht für den Linux-Kernel) und komprimieren Sie sie jeweils mit `gzip`, `bzip2` und `xz`. Messen Sie dabei die Ausführungszeit durch ein vorgesetztes `time`. Wie verhalten die Laufzeiten und die Komprimierungsraten sich zueinander?


11.11 [2] Experimentieren Sie mit der Datei aus der vorigen Aufgabe und vergleichen Sie die Laufzeit und die Größe des Resultats für `xz` mit verschiedenen Effizienz-Optionen. Was ist der Unterschied zwischen `-1` und `-6` und zwischen `-6` und `-9`?

11.6 Dateien archivieren und komprimieren mit `zip` und `unzip`

Für den Datenaustausch mit Windows-Rechnern oder im Internet ist es mitunter nützlich, das verbreitete ZIP-Dateiformat zu verwenden (wobei die gängigen Dateiarchivprogramme unter Windows heuer durchaus auch mit `.tar.gz` zurechtkommen). Unter Linux gibt es dafür die Programme `zip` (zum Erstellen von Archiven) und `unzip` (zum Auspacken von Archiven).


Je nach Distribution kann es sein, dass Sie diese Programme unabhängig voneinander installieren müssen. Bei Debian GNU/Linux zum Beispiel gibt es zwei separate Pakete `zip` und `unzip`.

`zip`

Das Programm `zip` kombiniert Archivieren und Komprimieren, so wie Sie das vielleicht von Programmen wie PKZIP kennen. Im einfachsten Fall fasst es die auf der Kommandozeile aufgezählten Dateien zusammen:

```
$ zip test.zip file1 file2
  adding: file1 (deflated 66%)
  adding: file2 (deflated 62%)
$ _
```

(Dabei ist `test.zip` der Name des resultierenden Archivs.)

Mit der Option `-r` können Sie `zip` sagen, dass es rekursiv in Verzeichnisse absteigen soll:

```
$ zip -r test.zip ziptest
  adding: ziptest/ (stored 0%)
  adding: ziptest/testfile (deflated 62%)
  adding: ziptest/file2 (deflated 62%)
  adding: ziptest/file1 (deflated 66%)
```

Und mit der Option -@ liest zip die Namen der zu archivierenden Dateien von der Kommandozeile:

```
$ find ziptest | zip -@ test
  adding: ziptest/ (stored 0%)
  adding: ziptest/testfile (deflated 62%)
  adding: ziptest/file2 (deflated 62%)
  adding: ziptest/file1 (deflated 66%)
```

(Die .zip-Endung des Archivs dürfen Sie weglassen.)

zip kennt zwei Methoden, um Dateien in ein Archiv zu tun: stored bedeutet, dass die Datei ohne Komprimierung abgespeichert wurde, während deflated für Komprimierung steht (und die Prozentzahl gibt an, *um wieviel* die Datei kleiner gemacht wurde – »deflated 62%« heißt also, dass die Datei im Archiv nur 38% der Originalgröße einnimmt). zip wählt automatisch die sinnvollere Methode, wenn Sie nicht mit der Option -0 die Komprimierung komplett abschalten.

Wenn Sie zip mit einem existierenden ZIP-Archiv als erstem Parameter aufrufen und nichts Anderes sagen, werden die zu archivierenden Dateien zusätzlich zum bisherigen Inhalt dem Archiv *hinzugefügt* (namensgleiche schon im Archiv vorhandene Dateien werden dabei überschrieben). zip benimmt sich damit anders als das weiter oben besprochene Programm tar. (Nur dass Sie es wissen.) Wenn Sie ein »sauberes« Archiv wollen, müssen Sie die Datei vorher löschen.

Außer dumpfem Hinzufügen kennt zip noch einige andere Betriebsarten: Die Option -u (*update*) aktualisiert das Archiv, indem im Archiv existierende Dateien nur von auf der Kommandozeile angegebenen Dateien überschrieben werden, wenn letztere neuer sind als erstere (auf der Kommandozeile benannte Dateien, die noch nicht im Archiv stehen, werden auf jeden Fall aufgenommen). Mit -f (*freshen*) können Sie Dateien im Archiv durch neuere Versionen von der Kommandozeile überschreiben, aber es werden keine Dateien ins Archiv geschrieben, die noch nicht vorher drinstanden. -d (*delete*) interpretiert die auf der Kommandozeile angegebenen Dateinamen als Namen von Dateien *im Archiv* und löscht diese.

Neuere Versionen von zip unterstützen außerdem den -FS-Modus (*filesystem sync*). Dieser »synchronisiert« ein Archiv mit dem Dateisystem, indem er im Wesentlichen dasselbe wie -u macht, aber außerdem Dateien aus dem Archiv löscht, die nicht auf der Kommandozeile benannt wurden (oder, im Falle von -r, in einem durchsuchten Verzeichnis stehen). Der Vorteil dieser Methode gegenüber dem kompletten Neuaufbau des Archivs ist, dass die schon im Archiv existierenden und unveränderten Dateien nicht neu komprimiert werden müssen.

zip kennt jede Menge Optionen, die Sie sich mit »zip -h« oder (ausführlicher) mit »zip -h2« anschauen können. Auch die Manpage zip(1) hat einiges zu sagen.

Wieder auspacken können Sie ein ZIP-Archiv mit unzip (es darf sich dabei auch durchaus um ein ZIP-Archiv von einem Windows-Rechner handeln). Sinnvollerweise werfen Sie zuerst mit der Option -v einen Blick in das Archiv, um zu schauen, was drinsteht – das erspart Ihnen möglicherweise unangenehme Überraschungen mit Unterverzeichnissen (oder der Abwesenheit selbiger).

unzip

```
$ unzip -v test                                          Endung .zip darf wegfallen
Archive:  test.zip
 Length   Method    Size  Cmpr    Date    Time   CRC-32   Name
--------  ------  ------- ---- ---------- ----- --------  ----
       0  Stored        0   0% 2012-02-29 09:29 00000000  ziptest/
```

```
   16163  Defl:N     6191  62% 2012-02-29 09:46 0d9df6ad  ziptest/testfile
   18092  Defl:N     6811  62% 2012-02-29 09:01 4e46f4a1  ziptest/file2
   35147  Defl:N    12119  66% 2012-02-29 09:01 6677f57c  ziptest/file1
--------          -------  ---                            -------
 69402            25121  64%                            4 files
```

Zum tatsächlichen Auspacken genügt es, wenn Sie `unzip` mit dem Namen des Archivs als einzigem Parameter aufrufen:

```
$ mv ziptest ziptest.orig
$ unzip test
Archive:  test.zip
   creating: ziptest/
  inflating: ziptest/testfile
  inflating: ziptest/file2
  inflating: ziptest/file1
```

Mit `-d` können Sie das Archiv in einem anderen Verzeichnis als dem aktuellen auspacken. Dieses Verzeichnis wird, falls nötig, zuerst angelegt:

```
$ unzip -d dir test
Archive:  test.zip
   creating: dir/ziptest/
  inflating: dir/ziptest/testfile
  inflating: dir/ziptest/file2
  inflating: dir/ziptest/file1
```

Wenn Sie bestimmte Dateien auf der Kommandozeile benennen, werden nur diese Dateien ausgepackt:

```
$ rm -rf ziptest
$ unzip test ziptest/file1
Archive:  test.zip
  inflating: ziptest/file1
```

(Das Verzeichnis `ziptest` wird in diesem Beispiel trotzdem angelegt.)

Alternativ können Sie mit der Option `-x` gezielt bestimmte Dateien vom Auspacken ausschließen:

```
$ rm -rf ziptest
$ unzip test -x ziptest/file1
Archive:  test.zip
   creating: ziptest/
  inflating: ziptest/testfile
  inflating: ziptest/file2
```

Sie können auch Shell-Suchmuster verwenden, um bestimmte Dateien auszupacken oder auszuschließen:

```
$ rm -rf ziptest
$ unzip test "ziptest/f*"
Archive:  test.zip
  inflating: ziptest/file2
  inflating: ziptest/file1
$ rm -rf ziptest
$ unzip test -x "*/t*"
Archive:  test.zip
```

```
  creating: ziptest/
 inflating: ziptest/file2
 inflating: ziptest/file1
```

(Achten Sie auf die Anführungszeichen, die die Suchmuster vor der tatsächlichen Shell verstecken, damit unzip sie zu sehen bekommt.) Im Gegensatz zur Shell beziehen sich die Suchmuster auf den *kompletten* Dateinamen (inklusive allfälliger »/«).

Wie nicht anders zu erwarten, können Sie auch bei unzip eine Vielzahl weiterer Optionen angeben. Lassen Sie sich mit »unzip -h« oder »unzip -hh« die programminterne Hilfe anzeigen oder lesen Sie unzip(1).

Übungen

11.12 [!2] Legen Sie in Ihrem Heimatverzeichnis einige Dateien an und verpacken Sie sie in einem zip-Archiv. Betrachten Sie den Inhalt des Archivs mit »unzip -v«. Entpacken Sie das Archiv wieder im Verzeichnis /tmp.

11.13 [!1] Was passiert, wenn eine Datei, die Sie gerade mit unzip auspacken wollen, schon im Dateisystem existiert?

11.14 [2] Ein ZIP-Archiv files.zip enthalte zwei Unterverzeichnisse a und b, in denen eine Mischung von Dateien mit verschiedenen Endungen (etwa .c, .txt und .dat) stehen. Geben Sie ein unzip-Kommando an, mit dem Sie in einem Schritt den kompletten Inhalt von a bis auf die .txt-Dateien extrahieren können.

Kommandos in diesem Kapitel

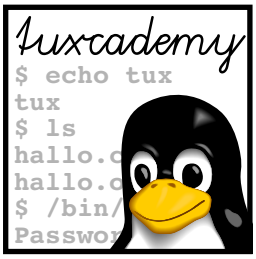
bunzip2	Entkomprimierungsprogramm für .bz2-Dateien	bzip2(1)	164
bzip2	Komprimierungsprogramm	bzip2(1)	164
gunzip	Entkomprimierungsprogramm für .gz-Dateien	gzip(1)	163
split	Teilt Dateien in Stücke bis zu einer gegebenen Größe auf	split(1)	159
tar	Dateiarchivierungsprogramm	tar(1)	159
unzip	Entkomprimierungsprogramm für (Windows-)ZIP-Archive	unzip(1)	167
zip	Archivierungs- und Komprimierungsprogramm à la PKZIP	zip(1)	166

Zusammenfassung

- Bei der »Archivierung« werden viele Dateien zu einer großen zusammengefasst. »Komprimierung« bringt eine Datei umkehrbar in eine kompaktere Form.
- tar ist das gängigste Archivprogramm unter Linux.
- gzip ist ein Programm, das beliebige Dateien komprimiert und dekomprimiert. Es kann zusammen mit tar benutzt werden.
- bzip2 ist ein weiteres Kompressionsprogramm. Es kann höhere Kompressionsraten erzeugen als gzip, aber braucht auch mehr Zeit und Speicherplatz.
- Ein weiteres Komprimierungsprogramm, das noch besser komprimiert und dafür noch länger braucht, ist xz.
- Die Programme zip und unzip erlauben die Verarbeitung von ZIP-Archiven, wie sie zum Beispiel von PKZIP unter Windows angelegt und gelesen werden können.

Literaturverzeichnis

RFC1951 P. Deutsch. »DEFLATE Compressed Data Format Specification version 1.3«, Mai 1996. http://www.ietf.org/rfc/rfc1951.txt



12

Einstieg in die Systemadministration

Inhalt

Lernziele

- Einen Überblick über die Rolle des Systemadministrators haben
- Betriebssystemkern und Prozesse verstehen
- Konzepte der Software-Paketverwaltung kennen

Vorkenntnisse

- Einfache Shell-Benutzung (Kapitel 4)
- Kenntnisse über die Linux-Dateisystemstruktur (Kapitel 10)

12.1 Systemadministration: Grundlagen

Was tut ein Systemadministrator? Den Rechner konfigurieren, Software installieren und gelegentlich entfernen, Geräte anschließen und benutzbar machen, Sicherheitskopien anlegen und bei Bedarf wieder einspielen, Benutzerkonten hinzufügen und entfernen, Benutzern mit Problemen helfen, … die Liste ist eindrucksvoll lang. In der alten Heimcomputer-Zeit war der Benutzer eines Rechners gleichzeitig der Administrator, und diese Idee hat sich in Systemen wie Windows noch lange gehalten (selbst als Windows ein Benutzerkonzept hatte, war es gängig, als »Administrator« zu arbeiten, da diverse wichtige Programme die entsprechenden Rechte einfach voraussetzten). Unix – das System, das Linux inspiriert hat – war dagegen von Anfang an dafür ausgelegt, mehrere Benutzer zu unterstützen, und die Trennung zwischen einem »Administrator« mit besonderen Privilegien und »gewöhnlichen« Benutzern ist darum viel tiefer im System verankert als bei Betriebssystemen aus der Heimcomputer-Tradition.

Die *Linux-Essentials*-Prüfung des LPI stellt Systemadministration nicht in den Vordergrund, aber zumindest ein gewisses Grundlagenwissen sollten Sie schon haben – vielleicht nicht um selbst Systemadministrator zu *sein*, sondern um besser zu verstehen, was Ihr Systemadministrator für Sie tut, oder selber einmal Systemadministrator zu *werden*. Wissen, das für Systemadministratoren wichtig ist, prüft das LPI für weiterführende Zertifikate wie LPIC-1 und vor allem LPIC-2 und LPIC-3.

In diesem Kapitel erwähnen wir einige Themen, die weniger mit der unmittelbaren Benutzung eines Linux-Rechners zu tun haben, sondern zum Beispiel damit, wie Sie sich ein Bild von dem machen können, was auf dem Rechner so alles läuft (Stichwort: »Warum ist mein Computer so langsam??«), oder wie die Software auf dem Rechner verwaltet wird. Es geht uns dabei eher um den groben Überblick als um Details.

`root` Der Administrator auf einem Linux-System hat ein besonderes Benutzerkonto, `root`, zur Verfügung. Dieses Konto ist von den sonst üblichen Rechteprüfungen (Kapitel 14) ausgenommen und darf darum zum Beispiel auf alle Dateien im System zugreifen. Das ist etwa nötig, um neue Software zu installieren – systemweit installierte Programmdateien dürfen »normale« Benutzer zwar lesen und starten, aber nicht ändern, damit Manipulationen zuungunsten anderer Anwender ausgeschlossen sind. Auch beispielsweise zur Erstellung von Sicherheitskopien muss der Administrator die Dateien *aller* Benutzer lesen können (und zum Wiedereinspielen sogar schreiben).

Es ist klar, dass mit der Möglichkeit, alle Dateien im System zu schreiben, auch die Gelegenheit besteht, das System schwer zu beschädigen. Wenn Sie als `root` angemeldet sind, hindert Linux Sie nicht daran, mit einem Kommando wie

```
# rm -rf /
```

das komplette Dateisystem zu löschen (und es gibt jede Menge subtilere Methoden, Schaden anzurichten). Sie sollten also die `root`-Privilegien nur dann in Anspruch nehmen, wenn Sie sie tatsächlich brauchen. Als `root` zum Beispiel im Web zu surfen oder E-Mail zu lesen ist ABSOLUT TABU.

Wenn Sie als `root` alle Dateien im System lesen können, könnten Sie natürlich der Versuchung anheimfallen, auf regelmäßiger Basis zum Beispiel die E-Mail Ihres Chefs oder Ihres Ehegesponses zu inspizieren. TUN SIE SOWAS NICHT. Es gehört sich nicht (jedenfalls im Falle Ihres Ehegesponses) und/oder ist verboten (im Falle Ihres Chefs) und kann Ihnen jede Menge Schwierigkeiten einbringen, die Ihnen den Spaß an Linux und Systemadministration, ganz zu schweigen vom Haus- oder Betriebsfrieden, dauerhaft

verderben können. Es spricht nichts dagegen, im Einzelfall und in Absprache mit den Betroffenen zur Fehlersuche oder -behebung einen kurzen Blick in ein Postfach zu tun – aber lassen Sie es nicht zur Gewohnheit werden.

Denken Sie im Zweifel immer an Peter Parker, alias Spider-Man: »Mit großer Macht kommt große Verantwortung.«

Sie sollten es tunlichst vermeiden, sich direkt (am Ende gar auf der grafischen Oberfläche) als `root` anzumelden. Verwenden Sie statt dessen das Programm `su`, um in einer Terminalsitzung, die Sie als normaler Benutzer gestartet haben, eine Shell zu bekommen, die als `root` läuft:

```
$ /bin/su -
Passwort: geheim                                    Kennwort für root
# _
```

Wenn Sie die `root`-Shell beenden (mit `exit` oder Strg+d), landen Sie wieder in der Shell, in der Sie ursprünglich `su` aufgerufen haben.

Manche Distributionen versuchen, auf ein separates `root`-Konto zu verzichten. Ubuntu zum Beispiel erlaubt dem bei der Installation angelegten Benutzer, einzelne Kommandos mit `root`-Rechten auszuführen, indem man ein `sudo` davorstellt, wie in

`sudo`

```
$ sudo less /var/log/syslog                         Systemprotokoll anschauen
```

(dieses Privileg kann der Benutzer bei Bedarf auch anderen Benutzern einräumen). Für größere Bauarbeiten ist mit »`sudo -i`« eine Shell zu bekommen, die mit Administratorrechten ausgestattet ist.

Die meisten Linux-Distributionen signalisieren, dass eine Shell mit den Rechten von `root` ausgeführt wird, indem sie eine Eingabeaufforderung ausgeben, die auf »`#`« endet. Steht dort etwas Anderes – typischerweise »`$`« oder »`>`«, handelt es sich um eine gewöhnliche Shell.

Übungen

12.1 [2] Probieren Sie `su` aus. Warum rufen wir das Programm im Beispiel über den vollen Pfadnamen auf?

12.2 [2] Für `su` müssen Sie das `root`-Kennwort wissen. `sudo` fragt Sie normalerweise nach Ihrem eigenen Kennwort. Was ist besser?

12.2 Die Systemkonfiguration

Während andere Systeme die Details der Systemkonfiguration in Datenbanken vergraben, die nur über spezielle Software zu ändern und anfällig gegen »Bitfäule« sind (Stichwort Windows-Registry), stehen systemweite Konfigurationsinformationen bei Linux in Textdateien im Verzeichnis `/etc` (ein paar Beispiele stehen in Abschnitt 10.3). Dort kann der Systemverwalter sie mit dem Texteditor seiner Wahl ändern und erweitern. Ein neuer Benutzer kann zum Beispiel angelegt werden, indem die relevanten Daten wie Benutzername, numerische Benutzerkennung oder Heimatverzeichnis in die Datei `/etc/passwd` eingetragen werden. Eine neue Festplatte läßt sich konfigurieren, indem Sie eine Zeile an `/etc/fstab` anhängen, die den Namen der Gerätedatei und den Namen des Verzeichnisses enthält, wo die Platte im System erscheinen soll.

Ein Linux-System ist ein komplexes Gebilde aus Softwarekomponenten unterschiedlicher Herkunft (einige davon in ihrem Ursprung älter als Linux selbst). Aus dieser historisch gewachsenen Struktur folgt, dass die verschiedenen Konfigurationsdateien in `/etc` sehr uneinheitlich aufgebaut sind – manche sind zeilenweise strukturiert, andere enthalten Abschnitte, die von geschweiften Klammern begrenzt werden, wieder andere sind in XML gehalten oder gar ausführbare Shellskripte. Für Administratoren, die mit allen diesen Formaten umgehen müssen, ist das zweifellos ein Ärgernis, aber es ist auch nicht einfach zu ändern, da man alle möglichen Softwarepakete anpassen müßte.

Einige weithin anerkannte Konventionen gibt es doch: Zum Beispiel erlauben die meisten Konfigurationsdateien Kommentare in Zeilen, die mit »#« anfangen.

Während die Idee, die Konfiguration in einzelnen Textdateien zu verwalten, einem auf den ersten Blick vorsintflutlich vorkommen mag, hat sie doch einige handfeste Vorteile:

- Normalerweise ist es nicht möglich, durch Fehler in der Konfiguration eines einzelnen Programmpakets oder Diensts den Rest des Systems nachhaltig zu beschädigen. (Es gibt natürlich ein paar Konfigurationsdateien, die so zentral für die Funktion des Systems sind, dass Irrtümer dort zum Beispiel einen Neustart verhindern können. Aber das ist eindeutig eine kleine Minderheit.)
- Die meisten Konfigurationsdateien erlauben Kommentare. Das macht es möglich, die Details einzelner Konfigurationseinstellungen direkt vor Ort zu dokumentieren und erleichtert so die Zusammenarbeit im Team oder vermeidet Unfälle aufgrund der eigenen Vergesslichkeit. Es ist auf jeden Fall besser, als sich irgendwie erinnern zu müssen, dass im Menü *X* ein Eintrag *Y* existiert, über den man einen Dialog öffnen kann, wo im Reiter *Z* ein Kästchen steht, das unbedingt abgehakt werden muss, weil sonst gar nichts mehr geht. (Zettel, auf denen sowas steht, haben die unangenehme Tendenz, sich aus dem Staub zu machen, wenn man sie am dringendsten braucht.)
- Textdateien können Sie in ein Revisionskontrollsystem wie Git oder Mercurial »einchecken« und damit nicht nur große dateiübergreifende Änderungen dokumentieren, sondern bei Bedarf auch kontrolliert wieder rückgängig machen. Die komplette Konfiguration eines Rechners kann damit auch bequem auf einem zentralen Server gesichert werden, so dass sie sofort wieder zur Verfügung steht, wenn der Rechner aus irgendwelchen Gründen – etwa nach einem katastrophalen Hardwareschaden – neu installiert werden muss. In Rechenzentren ist das ein gravierender Vorteil, vor allem wenn ein detailliertes »Auditing« aller Konfigurationsänderungen gewünscht wird.
- Textdateien erlauben die bequeme Konfiguration ganzer Rechnernetze, indem Sie von einem zentralen Server aus Konfigurationsdateien auf die zu administrierenden Rechner verteilen. Einschlägige Systeme wie »Puppet« oder »Salt« gestatten die Verwendung von »Schablonen« (*templates*) für Konfigurationsdateien, wo bei der Verteilung die für den Zielrechner passenden Details eingesetzt werden – damit wird eine manuelle Konfiguration einzelner Rechner (das »Turnschuhnetz«) völlig überflüssig. Auch das erleichtert die Verwaltung großer Systeme, ist aber auch für kleine Installationen eine definitive Erleichterung.

Übungen

12.3 [3] Stöbern Sie ein bisschen in `/etc` herum. Viele der Dateien dort haben Handbucheinträge – versuchen Sie mal etwas wie »`man fstab`«. Gibt es Dateien in `/etc`, die Sie als normaler Benutzer nicht lesen können, und warum?

12.3 Prozesse

Ein Programm, das gerade ausgeführt wird, nennt man »Prozess«. Neben dem eigentlichen Programmcode (in der Maschinensprache des jeweiligen Prozessors) gehören zu einem Prozess auch Arbeitsspeicher für die Daten und Verwaltungsinformationen wie die gerade aktuell offenen Dateien, eine Prozessumgebung (für die Umgebungsvariablen), ein aktuelles Verzeichnis und eine Prozessnummer oder »PID«, die den Prozess eindeutig im System identifiziert. Der Betriebssystemkern kümmert sich um das Anlegen von Prozessen, um die Zuteilung von Rechenzeit und Arbeitsspeicher und um das Aufräumen hinter Prozessen, die sich beendet haben. Prozesse können den Betriebssystemkern aufrufen, um auf Dateien, Geräte oder das Netzwerk zuzugreifen.

PID

Neue Prozesse entstehen, indem existierende Prozesse sich – ganz wie Bakterien oder andere niedere Lebewesen – in zwei fast identische Kopien aufspalten (»fast identisch«, weil einer der Prozesse als »Elter« und einer als »Kind« gilt). Außerdem kann ein Prozess dafür sorgen, dass er ein anderes Programm ausführt: Wenn Sie in der Shell zum Beispiel das Kommando `ls` aufrufen, erzeugt die Shell einen Kindprozess, in dem zunächst auch der Programmcode der Shell ausgeführt wird. Dieser Code kümmert sich (unter anderem) darum, eine etwaige Ein- und/oder Ausgabeumlenkung in Kraft zu setzen, und ersetzt sich dann selbst durch die Programmdatei `/bin/ls`. Mit dem Ende des `ls`-Programms endet auch der Kindprozess, und die Shell fragt Sie nach dem nächsten Kommando.

Der erste Prozess mit der PID 1 wird beim Systemstart vom Betriebssystemkern angelegt. Nach Konvention ist das ein Programm namens `/sbin/init`, und man spricht deshalb auch vom »Init-Prozess«. Der Init-Prozess ist verantwortlich dafür, das System kontrolliert hochzufahren und zum Beispiel weitere Prozesse für Hintergrunddienste zu starten.

ps Informationen über die Prozesse im System können Sie mit dem Kommando »`ps`« erhalten. Im einfachsten Fall zeigt `ps` Ihnen alle Prozesse, die auf Ihrem aktuellen Terminal (oder, heutzutage, im aktuellen Terminal-Fenster auf Ihrem Grafikbildschirm) laufen:

```
$ ps
  PID TTY   STAT TIME COMMAND
  997 pts/8 S    0:00 -bash
 1005 pts/8 R    0:00 ps
$ _
```

Die Spalten `PID` und `COMMAND` sprechen dabei für sich. `TTY` gibt den Namen des Terminals an (»`pts`/irgendwas« steht meist für ein Terminalfenster), `TIME` die Rechenzeit, die die Prozesse jeweils schon verbraucht haben, und `STAT` den »Prozesszustand«.

Ein Prozess ist bei Linux immer in einem von einer Reihe von Zuständen, namentlich Prozesszustand

Lauffähig (*runnable*, `R`) Der Prozess kann Rechenzeit zugeteilt bekommen.

Schlafend (*sleeping*, `S`) Der Prozess wartet auf ein Ereignis, typischerweise Ein- oder Ausgabe – einen Tastendruck oder Daten von der Platte.

Im Tiefschlaf (*uninterruptible sleep*, `D`) Der Prozess wartet auf ein Ereignis und kann dabei nicht unterbrochen werden. Prozesse sollten nicht zu lange in diesem Zustand verbleiben, weil Sie sie sonst nur durch einen Systemneustart loswerden. Wenn das passiert, deutet das auf einen Fehler hin.

Gestoppt (*stopped*, T) Der Prozess wurde von seinem Eigentümer oder einem Administrator zeitweilig angehalten, kann aber später weiterlaufen.

Zombie (Z) Der Prozess hat sich eigentlich schon beendet, aber sein Rückgabewert wurde vom Elterprozess noch nicht abgeholt, so dass der Prozess nicht richtig »sterben« kann, sondern als Untoter im System verbleibt. Zombies sind im Grunde kein Problem, da sie außer einem Platz in der Prozesstabelle keine Ressourcen belegen. Sollte Ihr System von einer Horde Zombies befallen sein, ist das ein Indiz für einen Fehler in dem Programm, das die Prozesse ursprünglich erzeugt hat – wenn Sie jenes Programm beenden, verschwinden auch die Zombies.

Mit Parametern können Sie steuern, welche Informationen `ps` Ihnen liefert. Zum Beispiel können Sie eine Prozessnummer angeben, um sich über einen speziellen Prozess zu informieren:

```
$ ps 1
  PID TTY      STAT   TIME COMMAND
    1 ?        Ss     0:00 init [2]
```

Mit der Option »`l`« bekommen Sie ausführlichere Informationen über einen Prozess:

```
$ ps l $$
F   UID  PID PPID PRI NI   VSZ  RSS WCHAN  STAT TTY    TIME COMMAND
0  1000 3542 3491  20  0 21152 2288 -      Ss   pts/8  0:00 /bin/bash
```

(»`$$`« steht hier für den »aktuellen Prozess«, die Shell).

`UID` ist die numerische Benutzerkennung des Eigentümers des Prozesses (siehe Kapitel 13), `PPID` die Prozessnummer des »Elters« des Prozesses. `PRI` ist die Priorität des Prozesses – je größer die Zahl, desto niedriger die Priorität (!) –, `VSZ` die Größe des Prozesses im Arbeitsspeicher (in KiB) und `RSS` die aktuelle Größe des Prozesses im RAM (ebenfalls in KiB).

`VSZ` und `RSS` sind nicht dasselbe, weil Teile des Prozesses auf Platte ausgelagert worden sein könnten. Der auf einem Linux-System verfügbare Arbeitsspeicher läßt sich ja vergrößern, indem man Auslagerungsspeicher (*swap space*) auf einer Plattenpartition oder Datei vorsieht.

Das Kommando `ps` unterstützt eine Vielzahl von Optionen, die die Auswahl von Prozessen und die Art und den Umfang der Informationen steuern, die für jeden Prozess ausgegeben werden. Lesen Sie `ps`(1).

`ps` und ähnliche Programme beziehen ihre Informationen aus dem `proc`-Dateisystem, das normalerweise unter `/proc` zur Verfügung steht und vom Betriebsssystemkern zur Verfügung gestellt wird. In den darin enthaltenen »Dateien« stehen aktuelle Daten über Prozesse und andere Systemeigenschaften. (Siehe hierzu auch Abschnitt 10.3.)

free Mit dem Kommando »`free`« erhalten Sie Auskunft über den Speicher im System:

```
$ free
             total       used       free     shared    buffers     cached
Mem:       3921956    1932696    1989260          0      84964     694640
-/+ buffers/cache:    1153092    2768864
Swap:      8388604          0    8388604
```

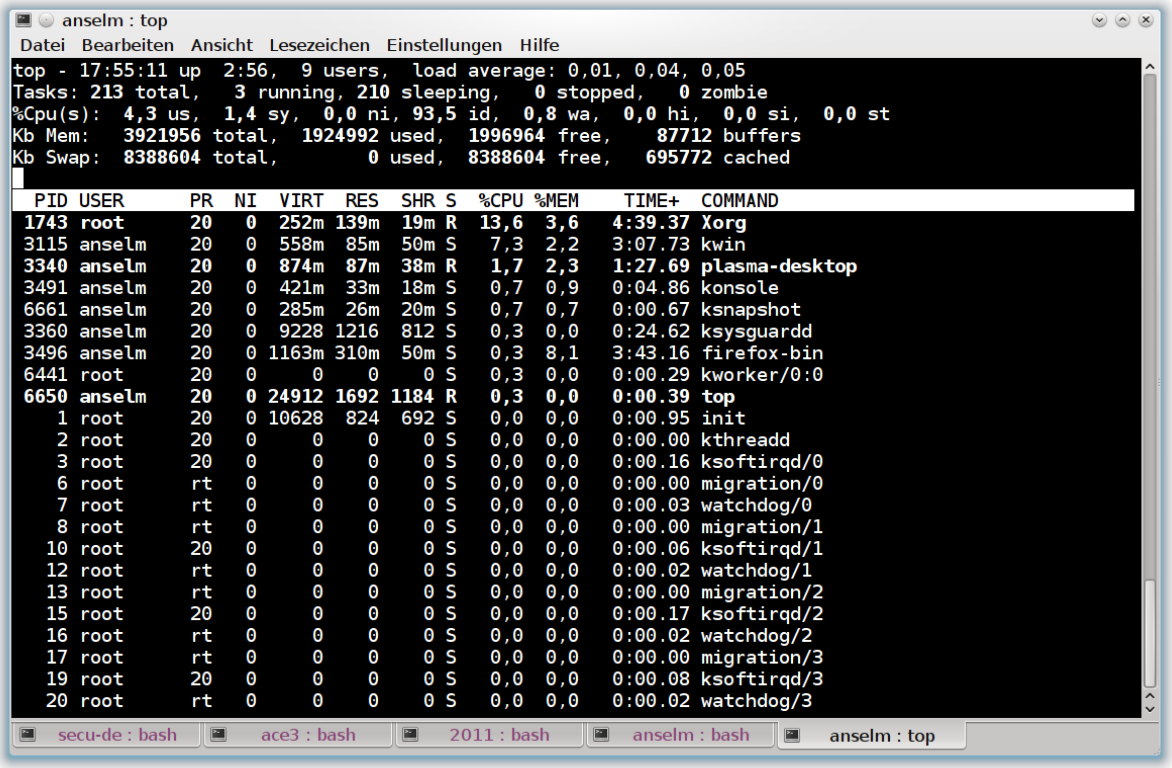


Bild 12.1: Das Programm `top`

Die »`Mem:`«-Zeile verrät Ihnen, dass dieser Rechner an die 4 GiB RAM hat (unter »`total`«; für den Betriebssystemkern geht etwas Speicher ab, der hier nicht auftaucht), der knapp zur Hälfte belegt ist (betrachten Sie »`used`« und »`free`«). Gut 700 MiB verwendet das Betriebssystem zum Speichern von Festplatten-Daten (die Spalten »`buffers`« und »`cached`«), und in der zweiten Zeile sehen Sie, wie sich das auf den freien und belegten Speicher auswirkt. Die dritte Zeile (»`Swap:`«) beschreibt die Auslastung des Auslagerungsspeichers (auf diesem Rechner 8 GiB).

Die »`shared`«-Spalte ist auf modernen Linux-Systemen immer `0` und kann ignoriert werden.

Auch `free` unterstützt eine Reihe von Optionen, etwa um das Ausgabeformat freundlicher zu machen:

```
$ free --human                                          »-h« wäre dasselbe
             total       used       free     shared    buffers     cached
Mem:          3,7G       1,9G       1,8G        0B        84M       678M
-/+ buffers/cache:       1,2G       2,5G
Swap:         8,0G         0B       8,0G
```

Mit »`M`« und »`G`« meint `free` übrigens computerfreundliche Mebi- und Gibibyte. Die Option `--si` schaltet auf Zehnerpotenzen (Mega- und Gigabyte) um.

top Das Kommando »`top`« schließlich ist eine Art Kombination aus `ps` und `free` mit fortlaufender Aktualisierung. Es zeigt eine bildschirmfüllende Darstellung von System- und Prozesskenndaten an; Bild 12.1 zeigt ein Beispiel:

- Im oberen Teil der Ausgabe sehen Sie in der ersten Zeile neben der aktuellen Uhrzeit die *uptime*, also die seit dem Systemstart vergangene Zeit (hier nicht ganz drei Stunden) und die Anzahl der angemeldeten Benutzer (die Neun hier ist nicht ganz wörtlich zu nehmen; jede Sitzung in einem Terminalfenster gilt als »Benutzer«). Rechts auf der ersten Zeile stehen drei Zahlen, die sogenannten *load averages*, die die Systemlast charakterisieren.

  Die *load averages* geben die Anzahl der lauffähigen Prozesse (Zustand `R`) an, respektive gemittelt über die letzte Minute, die letzten fünf Minuten und die letzten fünfzehn Minuten. Sie sollten diese Werte nicht überbewerten (!); man sieht an ihnen eigentlich nicht viel. Ist der Wert für die letzte Minute hoch und für die letzten 15 Minuten niedrig, dann hat Ihr System plötzlich mehr zu tun bekommen; ist der Wert für die letzte Minute niedrig und für die letzten 15 Minuten hoch, dann hatte Ihr System eben noch viel zu tun, aber das ist vorbei.

  Sind die *load averages* dauerhaft niedriger als die Anzahl der Prozessorkerne in Ihrem System, heißt das, dass Sie unnötig Geld für einen teuren Prozessor ausgegeben haben. Auf einem Acht-Kern-System zum Beispiel sind Werte um Acht (die traditionell einem Systemadministrator kalte Schauer über den Rücken jagen würden) absolut unbedenklich; Werte, die über lange Zeiträume viel kleiner sind als Acht, sind traurig.

- Die zweite Zeile gibt die Anzahl der Prozesse an und wie diese sich auf die Prozesszustände verteilen.

- Die dritte Zeile beschreibt in Prozent, womit die CPUs sich beschäftigen: »`us`« ist die Ausführung von Code außerhalb und »`sy`« die von Code innerhalb des Betriebssystemkerns. »`ni`« ist Code außerhalb des Betriebssystemkerns, der vom Benutzer absichtlich in der Priorität gesenkt wurde, und »`id`« ist Untätigkeit. »`wa`« ist Warten auf Ein/Ausgabe, und die anderen drei Spalten sind nicht so interessant.

- Die beiden folgenden Zeilen entsprechen im Wesentlichen der Ausgabe von `free`.

- Der untere Teil des Bildschirms ist eine Prozessliste ähnlich der von »`ps l`«. Sie wird (wie der obere Teil) in Abständen von einigen Sekunden aktualisiert und ist standardmäßig nach dem Anteil an CPU-Zeit sortiert, den die Prozesse in der Liste jeweils verbrauchen (der Prozess, mit dem das System die meiste Zeit verbringt, führt die Liste an).

  Wenn Sie die Taste m drücken, wird die Liste nach dem Speicherverbrauch sortiert – der fetteste Prozess steht vorne. Mit p kommen Sie zur CPU-Zeit-Liste zurück.

Mit h können Sie in `top` eine Hilfeseite aufrufen. Die Handbuchseite unter `top`(1) erklärt die Ausgabe und die möglichen Tastenkombinationen im Detail und zeigt auch, wie Sie den Inhalt der Prozessliste an Ihre Anforderungen anpassen können.

Übungen

12.4 [1] Mit der `ps`-Option `ax` können Sie sich alle Prozesse im System anzeigen lassen. Betrachten Sie die Liste. Welche Prozesse können Sie einordnen?

12.5 [2] Starten Sie in einer Shell-Sitzung einen länger laufenden Prozess (etwas wie »`sleep 120`« sollte reichen). Rufen Sie in einer anderen Sitzung »`ps ax`« auf und suchen Sie den Prozess in der Ausgabe. (*Tipp:* `grep` ist Ihr Freund.)

12.6 [!2] Benutzen Sie `top`, um nachzuschauen, welche Prozesse gerade die meiste Rechenzeit verbrauchen. Welche Prozesse verbrauchen den meisten Speicher?

12.4 Paketverwaltung

Heutige Linux-Distributionen bestehen in der Regel aus einer Vielzahl (typischerweise Tausenden) von »Paketen«, die jeweils alles enthalten, was für eine gewisse (Teil-)Funktionalität nötig ist: Ausführbare Programme, Bibliotheken, Dokumentation, … Bei der Inbetriebnahme eines Linux-Rechners können Sie als Administrator festlegen, welche Pakete auf dem Rechner installiert sein sollen, und natürlich können Sie auch später beliebige Pakete aus Ihrer Distribution nachinstallieren oder nicht benötigte entfernen.

Wie die Paketaufteilung genau aussieht, hängt von der jeweiligen Distribution ab. Bei Bibliotheken ist es zum Beispiel üblich, zwischen einem »Laufzeitpaket« und einem »Entwicklungspaket« zu differenzieren. Das Laufzeitpaket enthält diejenigen Dateien, die installiert sein müssen, damit andere Programme die Bibliothek benutzen können (etwa die tatsächliche dynamisch ladbare Bibliothek in einer `.so`-Datei, die in `/usr/lib` installiert wird). Das Entwicklungspaket müssen Sie nur installieren, wenn Sie vorhaben, neue oder existierende Programme zu *übersetzen*, die die Bibliothek benutzen – darin finden Sie zum Beispiel die Informationen, die der C-Compiler über die Bibliothek braucht (»Includedateien«), eine statisch nutzbare Bibliothek oder die Dokumentation des Bibliotheksinhalts. Die Dokumentation kann, wenn sie umfangreich ist, in ein weiteres Paket ausgelagert werden.

Hier ist zum Beispiel die Paketaufteilung für die `rsvg`-Bibliothek (sie kümmert sich um Grafiken im SVG-Format), à la Debian GNU/Linux 6.0 (»Squeeze«):

`librsvg2-2`	*Die eigentliche Bibliothek*
`librsvg2-dev`	*Entwicklungspaket*
`librsvg2-bin`	*Kommandozeilenprogramme*
`librsvg2-dbg`	*Debugging-Informationen*
`librsvg2-doc`	*Dokumentation*
`librsvg2-common`	*Mehr Kommandozeilenprogramme*
`python-rsvg`	*Anbindung an die Sprache Python*
`libimage-librsvg-perl`	*Anbindung an die Sprache Perl*

Auf jedem Linux-Rechner[1] gibt es eine »Paketdatenbank«, die angibt, welche Pakete der Rechner kennt und welche davon aktuell installiert sind. Sie können die Paketdatenbank in periodischen Abständen mit den »Repositories«, also den Servern für Pakete, Ihrer Distribution abgleichen und so herausfinden, welche der auf dem Rechner installierten Pakete möglicherweise veraltet sind, weil die Distribution neuere Versionen davon anbietet. Das Paketverwaltungssystem erlaubt Ihnen dann meist eine selektive Aktualisierung der betroffenen Pakete.

Paketdatenbank

Wie gut das im Detail klappt, hängt auch (wieder mal) von Ihrer Distribution ab. Die Sache kann nämlich komplizierter sein, als sie zuerst scheint: Die neue Version eines Pakets könnte zum Beispiel verlangen, dass auch eine Bibliothek (die in ihrem eigenen Paket steht) in einer neueren Version

[1] Jedenfalls jedem, der eine der wesentlichen Distributionen verwendet – es gibt ein paar Distributionen, die meinen, ohne ein Paketverwaltungssystem auszukommen, aber die sind mehr was für Fans.

installiert ist, und das kann zu Problemen führen, wenn ein anderes installiertes Programm zwingend die *alte* Version der Bibliothek benötigt. Manchmal kann es also sein, dass ein Paket gar nicht aktualisiert werden kann, ohne dass sich anderswo im System etwas gravierend ändern müßte. Gute Paketverwaltungssysteme erkennen solche Situationen und warnen Sie als Administrator und/oder geben Ihnen Eingriffsmöglichkeiten.

Wie schon in Abschnitt 2.4.7 angedeutet gibt es bei den wesentlichen Linux-Distributionen zwei verbreitete Paketverwaltungssysteme, die jeweils mit eigenen Werkzeugen und einem eigenen Format für die Paketdateien ankommen – die Paketverwaltung von Debian GNU/Linux und seinen Ablegern sowie die RPM-Paketverwaltung von Red Hat, SUSE & Co. Beide lösen im Grunde dieselbe Sorte Problem, unterscheiden sich aber im Detail, etwa in den Kommandos, die zur Paketverwaltung benötigt werden. Auf einem RPM-basierten System wie RHEL, Fedora oder openSUSE können Sie sich die Liste aller installierten Pakete zum Beispiel mit dem Kommando

```
$ rpm --query --all                                    »-qa« würde reichen
```

anzeigen lassen, während auf einem Debian-basierten System hierfür das Kommando

```
$ dpkg --list                                          »-l« würde es auch tun
```

nötig ist.

Die Paketdatenbanken selbst finden sich normalerweise unter `/var/lib`; auf Debian-artigen Systemen in `/var/lib/dpkg` (in `/var/cache/apt` werden die Inhaltsverzeichnisse der Repositories und allfällige von dort heruntergeladene Pakete zwischengelagert) und auf RPM-Systemen in `/var/lib/rpm`.

Programme wie `dpkg` und `rpm` bilden heute in der Regel das »Fundament« der Paketverwaltung. Als Administrator verwenden Sie lieber komfortablere Werkzeuge, die auf den Basisprogrammen aufbauen und zum Beispiel den bequemen Zugriff auf die Paket-Repositories zulassen und Abhängigkeiten zwischen Paketen automatisch auflösen. Verbreitet sind in der Debian-Welt zum Beispiel »Aptitude« und »Synaptic«, während auf der RPM-Seite Red Hat auf ein Programm namens YUM und SUSE auf eines namens »Zypper« setzt (wobei die SUSE Paketverwaltung auch in das allgemeine Administrationswerkzeug YaST integriert hat).

Einige dieser Werkzeuge sind sogar unabhängig vom unterliegenden Paketverwaltungssystem. »PackageKit« zum Beispiel kann nicht nur wahlweise Debian- oder RPM-Paketverwaltung verwenden, sondern sogar unter kontrollierten Umständen gewöhnlichen Benutzern ohne Administratorprivilegien die Installation oder Aktualisierung von Paketen erlauben.

Übungen

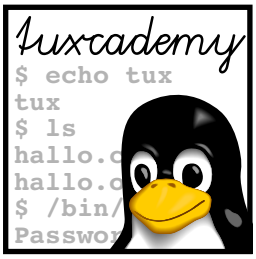
12.7 [2] Wieviele Pakete sind auf Ihrem System installiert? Verwenden Sie den im Text gezeigten `rpm`- bzw. `dpkg`-Aufruf und zählen Sie die Zeilen der Ausgabe. (*Vorsicht:* »`dpkg --list`« zeigt Ihnen auch Pakete, die mal installiert waren, aber entfernt oder von neueren Versionen überlagert wurden. Zählen Sie also nur diejenigen Zeilen in der Ausgabe, die mit »`ii`« anfangen.)

Kommandos in diesem Kapitel

dpkg Verwaltungswerkzeug für Debian-GNU/Linux-Pakete dpkg(8) 180
free Zeigt die Speicherauslastung und die Auslastung des Swap-Bereichs an free(1) 176
ps Gibt Prozess-Statusinformationen aus ps(1) 175
rpm Dient zur Paketverwaltung in vielen Linux-Systemen (Red Hat, SUSE, …) rpm(8) 180
su Startet eine Shell unter der Identität eines anderen Benutzers su(1) 173
sudo Erlaubt normalen Benutzern das Aufrufen bestimmter Kommandos mit Administratorprivilegien sudo(8) 173
top Bildschirmorientiertes Programm zur Beobachtung und Verwaltung von Prozessen top(1) 177

Zusammenfassung

- Linux trennt zwischen »normalen« Benutzern und dem Systemadministrator `root`. `root` unterliegt nicht den gewöhnlichen Rechteprüfungen.
- Als normaler Benutzer können Sie sich über `su` oder `sudo` zeitweilig Administratorprivilegien besorgen.
- Die Systemkonfiguration eines Linux-Rechners steht in Textdateien im Verzeichnis `/etc`.
- Prozesse sind Programme, die gerade ausgeführt werden.
- Kommandos wie `ps` und `top` geben Einblick in den aktuellen Systemzustand.
- Die wichtigen Linux-Distributionen verwenden entweder das Paketverwaltungssystem von Debian GNU/Linux oder das ursprünglich von Red Hat entwickelte RPM-System.
- Aufbauend auf Basiswerkzeugen bieten die meisten Distributionen komfortable Software zum Verwalten, Installieren und Entfernen von Softwarepaketen unter Berücksichtigung von Abhängigkeiten.



13

Benutzerverwaltung

Inhalt

Lernziele

- Das Benutzer- und Gruppenkonzept von Linux verstehen
- Die Struktur und Speicherung von Benutzer- und Gruppendaten bei Linux kennen
- Die Kommandos zur Verwaltung von Benutzer- und Gruppendaten anwenden können

Vorkenntnisse

- Kenntnisse über den Umgang mit Konfigurationsdateien

13.1 Grundlagen

13.1.1 Wozu Benutzer?

Früher waren Computer gross und teuer, aber heute sind Büroarbeitsplätze ohne eigenen PC (»persönlichen Computer«) kaum noch vorstellbar, und auch in den meisten häuslichen Arbeitszimmern ist ein Computer anzutreffen. Und während es in der Familie noch genügen mag, wenn Vater, Mutter und die Kinder ihre Dateien nach Verabredung jeweils in verschiedenen Verzeichnissen ablegen, ist das in Unternehmen oder Hochschulen definitiv nicht mehr ausreichend – spätestens wenn Plattenplatz oder andere Dienste auf zentralen Servern zur Verfügung gestellt werden, auf die viele Anwender zugreifen können, muss das Computersystem in der Lage sein, zwischen verschiedenen Benutzern unterscheiden und diesen unterschiedliche Rechte zuordnen zu können. Schließlich geht Frau Schulz aus der Entwicklungsabteilung die Gehaltsliste für alle Angestellten in der Regel genausowenig etwas an wie Herrn Schmidt aus der Personalabteilung die detaillierten Pläne für die neuen Produkte. Und auch am heimischen Herd ist ein bisschen Privatsphäre möglicherweise auch erwünscht – die Weihnachtsgeschenkliste oder Tochters Tagebuch (ehedem mit Schloss versehen) sollen neugierigen Augen vielleicht nicht ganz ohne weiteres zugänglich sein.

Den Umstand, dass Tochters Tagebuch vielleicht sowieso auf Facebook & Co. der ganzen Welt zum Lesen zur Verfügung steht, lassen wir hier mal außer Acht; und selbst wenn das so ist, dann soll die ganze Welt ja wohl trotzdem nicht in Tochters Tagebuch *schreiben* dürfen. (Aus diesem Grund unterstützt auch Facebook die Idee verschiedener Benutzer.)

Der zweite Grund dafür, verschiedene Benutzer zu unterscheiden, folgt aus der Tatsache, dass diverse Aspekte des Systems nicht ohne besondere Privilegien anschau- oder gar änderbar sein sollen. Linux führt aus diesem Grund eine separate Benutzeridentität (`root`) für den Systemadministrator, die es möglich macht, Informationen wie etwa die Benutzerkennwörter vor »gewöhnlichen« Benutzern geheim zu halten. Die Plage älterer Windows-Systeme – Programme, die Sie per E-Mail oder durch ungeschicktes Surfen erhalten und die dann im System alle Arten von Schindluder treiben – kann Sie unter Linux nicht ereilen, da alles, was Sie als gewöhnlicher Benutzer starten können, nicht in der Position ist, systemweit Schindluder zu treiben.

Ganz korrekt ist das leider nicht: Hin und wieder werden Fehler in Linux bekannt, über die ein »normaler Benutzer« theoretisch Dinge tun kann, die sonst dem Administrator vorbehalten sind. Diese Sorte Fehler ist extrem ärgerlich und wird in der Regel sehr zeitnah nach dem Finden behoben, aber es kann natürlich sein, dass so ein Fehler einige Zeit lang unerkannt im System geschlummert hat. Sie sollten daher bei Linux (wie bei allen Betriebssystemen) anstreben, von kritischen Systembestandteilen wie dem Systemkern immer die neueste Version laufen zu lassen, die Ihr Distributor unterstützt.

Auch die Tatsache, dass Linux die Systemkonfiguration vor unbefugtem Zugriff durch normale Benutzer schützt, sollte Sie nicht dazu verleiten, Ihr Gehirn auszuschalten. Wir geben Ihnen hier einige Tipps (etwa dass Sie sich nicht als `root` auf der grafischen Oberfläche anmelden sollen), aber Sie sollten weiter mitdenken. Mails, die Sie auffordern, Adresse *X* anzusurfen und dort Ihre Bank-PIN und drei Transaktionsnummern einzutippen, können Sie auch unter Linux erhalten, und Sie sollten sie genauso ignorieren wie anderswo auch.

Benutzerkonten

Linux unterscheidet verschiedene Benutzer über unterschiedliche **Benutzerkonten** (engl. *accounts*). Typischerweise werden bei den gängigen Distributionen während der Installation zwei Benutzerkonten eingerichtet, nämlich `root` für Administrationsaufgaben und ein weiteres Konto für einen »normalen« Benutzer.

Zusätzliche Konten können Sie als Administrator dann später selbst einrichten, oder sie ergeben sich, wenn der Rechner als Client in einem größeren Netz installiert ist, aus einer anderswo gespeicherten Benutzerkonten-Datenbank.

Linux unterscheidet *Benutzerkonten*, nicht Benutzer. Es hindert Sie zum Beispiel niemand daran, ein separates Benutzerkonto zum E-Mail-Lesen und Surfen im Internet zu verwenden, wenn Sie zu 100% sicher gehen wollen, dass Sachen, die Sie sich aus dem Netz herunterladen, keinen Zugriff auf Ihre wichtigen Daten haben (was ja trotz der Benutzer-Administrator-Trennung sonst passieren könnte). Mit einem bisschen Trickreichtum können Sie sogar einen Browser und ein E-Mail-Programm, die unter Ihrem Surf-Konto laufen, zwischen Ihren »normalen« Programmen anzeigen lassen[1].

Unter Linux ist jedem Benutzerkonto eine eindeutige numerische Kennung zugeordnet, die sogenannte *User ID* oder kurz **UID**. Zu einem Benutzerkonto gehört außerdem noch ein textueller **Benutzername** (etwa `root` oder `hugo`), der für Menschen leichter zu merken ist. An den meisten Stellen, wo es darauf ankommt – etwa beim Anmelden oder bei der Ausgabe einer Liste von Dateien mit ihren Eigentümern – verwendet Linux, wo möglich, den textuellen Namen.

UID

Benutzername

Der Linux-Kern weiß nichts über die textuellen Benutzernamen; in den Prozessdaten und in den Eigentümerangaben im Dateisystem wird immer nur die UID verwendet. Das kann zu Problemen führen, wenn ein Benutzer gelöscht wird, der noch Dateien im System hat, und anschließend die UID einem anderen Benutzer zugewiesen wird. Jener Benutzer »erbt« die Dateien des vorigen UID-Inhabers.

Grundsätzlich spricht nichts Technisches dagegen, dass mehreren Benutzernamen dieselbe (numerische) UID zugeordnet ist. Diese Benutzer haben gleichberechtigten Zugriff auf alle dieser UID gehörenden Dateien, aber jeder kann sein eigenes Kennwort haben. Sie sollten das aber nicht oder nur mit großer Vorsicht ausnutzen.

13.1.2 Benutzer und Gruppen

Um mit einem Linux-Rechner zu arbeiten, müssen Sie sich erst anmelden (neudeutsch »einloggen«), damit das System Sie als Sie erkennt und Ihnen die richtigen Zugriffsrechte zuordnen kann (hierzu später mehr). Alle Aktionen, die Sie während einer Arbeitssitzung (vom Anmelden bis zum Abmelden) ausführen, werden Ihrem Benutzerkonto zugeordnet. Jeder Benutzer hat außerdem ein **Heimatverzeichnis** (engl. *home directory*), in dem er seine »eigenen Dateien« ablegen kann und auf das andere Benutzer oft keinen Lese- und mit sehr großer Sicherheit keinen Schreibzugriff haben. (Nur der Systemadministrator, `root`, darf alle Dateien lesen und schreiben.)

Heimatverzeichnis

Je nachdem, welche Linux-Distribution Sie benutzen (Stichwort: Ubuntu), kann es sein, dass Sie sich nicht explizit beim System anmelden müssen. Dann »weiß« der Rechner allerdings, dass normalerweise Sie kommen, und nimmt einfach an, dass Sie auch wirklich Sie sind. Sie tauschen hier Sicherheit gegen Bequemlichkeit; dieser konkrete Tauschhandel ist vermutlich nur sinnvoll, wenn Sie mit einiger Gewissheit davon ausgehen können, dass niemand außer Ihnen Ihren Rechner einschaltet – und dürfte damit *eigentlich* auf den Computer in Ihrem Single-Haushalt ohne Putzfrau beschränkt sein. Wir haben es Ihnen gesagt.

Mehrere Benutzer, die bestimmte Systemressourcen oder Daten gemeinsam nutzen, können eine **Gruppe** bilden. Linux identifiziert die Gruppenmitglieder

Gruppe

[1] Was dann natürlich wieder etwas gefährlich ist, da Programme, die auf demselben Bildschirm laufen, miteinander kommunizieren können.

entweder durch feste, namentliche Zuordnung oder durch eine vorübergehende Anmeldung ähnlich der Anmeldeprozedur für Benutzer. Gruppen haben keine automatisch vorhandenen »Heimatverzeichnisse«, aber Sie können als Administrator natürlich beliebige Verzeichnisse einrichten, die für bestimmte Gruppen gedacht sind und entsprechende Rechte haben.

Auch Gruppen werden betriebssystemintern durch numerische Kennungen (engl. *group IDs*, kurz GIDs) identifiziert.

Gruppennamen verhalten sich zu GIDs wie Benutzernamen zu UIDs: Der Linux-Kernel kennt nur erstere und legt auch nur erstere in den Prozessdaten und im Dateisystem ab.

Jeder Benutzer gehört zu einer *primären Gruppe* und möglicherweise mehreren *sekundären* oder *zusätzlichen Gruppen*. In einem Unternehmen wäre es beispielsweise möglich, projektspezifische Gruppen einzuführen und jeweils die Projektmitarbeiter in die betreffende Gruppe aufzunehmen, damit sie Zugriff auf gemeinsame Daten in einem Verzeichnis bekommen, das nur für Gruppenmitglieder zugänglich ist.

Für die Zwecke der Rechtevergabe sind alle Gruppen gleichwertig – jeder Benutzer bekommt immer alle Rechte, die sich aus allen Gruppen ergeben, in denen er Mitglied ist. Der einzige Unterschied zwischen der primären Gruppe und den sekundären Gruppen ist, dass Dateien, die ein Benutzer neu anlegt, in der Regel[2] seiner primären Gruppe zugeordnet werden.

Bis einschließlich zum Linux-Kernel 2.4 konnte ein Benutzer maximal 32 zusätzliche Gruppen haben; seit Linux-Kernel 2.6 ist die Anzahl der zusätzlichen Gruppen nicht mehr beschränkt.

Die UID eines Benutzerkontos, die primäre und die sekundären Gruppen und die dazugehörigen GIDs verrät Ihnen das Programm `id`:

```
$ id
uid=1000(hugo) gid=1000(hugo) groups=24(cdrom),29(audio),44(video),▷
 ◁ 1000(hugo)
$ id root
uid=0(root) gid=0(root) groups=0(root)
```

Mit den Optionen `-u`, `-g` und `-G` läßt `id` sich überreden, nur die UID des Kontos, die GID der primären Gruppe oder die GIDs der sekundären Gruppe auszugeben. (Diese Optionen lassen sich nicht kombinieren). Mit der zusätzlichen Option `-n` gibt es Namen statt Zahlen:

```
$ id -G
1000 24 29 44
$ id -Gn
hugo cdrom audio video
```

Das Kommando `groups` liefert dasselbe Resultat wie das Kommando »`id -Gn`«.

`last` Mit dem Kommando `last` können Sie herausfinden, wer sich wann auf Ihrem Rechner angemeldet hat (und im Falle von Anmeldungen über das Netzwerk, von wo aus):

```
$ last
hugo     pts/1       pchugo.example.c Wed Feb 29 10:51   still logged in
oberboss pts/0       pc01.vorstand.ex Wed Feb 29 08:44   still logged in
```

[2] Die Ausnahme besteht darin, dass der Eigentümer eines Verzeichnisses verfügen kann, dass neue Dateien und Verzeichnisse in diesem Verzeichnis der Gruppe zugeordnet werden, der auch das Verzeichnis selbst zugeordnet ist. Aber das nur der Vollständigkeit halber.

```
hugo     pts/2        pchugo.example.c Wed Feb 29 01:17 - 08:44  (07:27)
susi     pts/0        :0               Tue Feb 28 17:28 - 18:11  (00:43)
<<<<<
reboot   system boot 3.2.0-1-amd64    Fri Feb  3 17:43 - 13:25 (4+19:42)
<<<<<
```

Die dritte Spalte gibt bei Sitzungen über das Netz den Rechnernamen des `ssh`-Clients an. »`:0`« steht für den grafischen Bildschirm (genaugenommen den ersten X-Server – es könnte mehrere geben).

Beachten Sie auch den `reboot`-Eintrag, der angibt, dass der Rechner neu gestartet wurde. In der dritten Spalte steht hier die Versionsnummer des Linux-Betriebssystemkerns, so wie »`uname -r`« ihn liefert.

Mit einem Benutzernamen als Parameter liefert `last` Informationen über einen bestimmten Benutzer:

```
$ last
hugo     pts/1        pchugo.example.c Wed Feb 29 10:51   still logged in
hugo     pts/2        pchugo.example.c Wed Feb 29 01:17 - 08:44  (07:27)
<<<<<
```

Es könnte Sie (mit Recht!) stören, dass diese Sorte doch etwas sensitive Information anscheinend beliebigen Systembenutzern ohne Weiteres zugänglich gemacht wird. Wenn Sie als Administrator die Privatsphäre Ihrer Benutzer etwas besser schützen möchten, als Ihre Linux-Distribution das von sich aus macht, können Sie mit dem Kommando

```
# chmod o-r /var/log/wtmp
```

das allgemeine Leserecht für die Datei entfernen, in der `last` die verräterischen Daten findet. Benutzer, die keine Administratorprivilegien haben, sehen dann nur noch etwas wie

```
$ last
last: /var/log/wtmp: Permission denied
```

13.1.3 »Natürliche Personen« und Pseudobenutzer

Außer für »natürliche Personen« – die menschlichen Benutzer des Systems – wird das Benutzer- und Gruppenkonzept auch verwendet, um die Zugriffsrechte auf gewisse Bereiche des Systems zu strukturieren. Das bedeutet, dass es neben den persönlichen Konten der »echten« Benutzer wie Ihnen weitere Konten auf dem System gibt, die nicht tatsächlichen menschlichen Benutzern zugeordnet, sondern systemintern für administrative Funktionen zuständig sind. Hier werden funktionale Rollen definiert, denen eigene Konten und Gruppen zugeordnet werden.

Pseudobenutzer

Nach der Installation von Linux finden Sie in den Dateien `/etc/passwd` und `/etc/group` eine ganze Reihe solcher Pseudobenutzer. Die wichtigste Rolle hat hier der uns schon bekannte Benutzer `root` mit der gleichnamigen Gruppe. UID und GID von `root` sind 0 (Null).

Die Sonderrechte von `root` sind an die UID 0 gekoppelt; die GID 0 hat keine besondere Bedeutung über die normalen Zugriffsrechte hinaus.

Weitere Pseudobenutzer können für bestimmte Programmsysteme (beispielsweise `news` für News mit INN und `postfix` für Mail mit Postfix) oder für bestimmte Komponenten oder Gerätegruppen (beispielsweise Drucker, Band- und Diskettenlaufwerke) existieren. Diese Konten erreichen Sie gegebenenfalls wie andere auch über dem Befehl `su`. Diese Pseudobenutzer sind als Eigentümer von Dateien

Pseudobenutzer für Rechtevergabe

und Verzeichnissen hilfreich, um die mit dem Eigentum an Systemdaten verbundenen Zugriffsrechte flexibel an die speziellen Anforderungen anzupassen, ohne dazu das `root`-Konto benutzen zu müssen. Dasselbe geht auch für Gruppen; beispielsweise haben Mitglieder der Gruppe `disk` Zugriffsrecht auf die Platten auf Blockebene.

Übungen

13.1 [1] Wodurch unterscheidet der Betriebssystemkern verschiedene Benutzer und Gruppen?

13.2 [2] Was passiert, wenn eine UID zweimal mit unterschiedlichem Namen vergeben wird? Ist das erlaubt?

13.3 [1] Was versteht man unter Pseudobenutzern? Nennen Sie Beispiele!

13.4 [2] (Beim zweiten Durcharbeiten.) Ist es akzeptabel, einen Benutzer in die Gruppe `disk` aufzunehmen, dem Sie nicht das `root`-Kennwort anvertrauen würden? Warum (nicht)?

13.2 Benutzer- und Gruppendaten

13.2.1 Die Datei `/etc/passwd`

zentrale Benutzerdatenbank

Die zentrale Benutzerdatenbank ist die Datei `/etc/passwd`. Jeder Benutzer auf dem System hat einen Eintrag in dieser Datei – eine Zeile, in der Attribute wie Linux-Benutzername, »richtiger« Name usw. festgehalten werden. Bereits mit der Erstinstallation eines Linux-Systems sind in der Datei die meisten Pseudobenutzer eingetragen.

Die Einträge in `/etc/passwd` haben folgendes Format:

⟨*Benutzername*⟩:⟨*Kennwort*⟩:⟨*UID*⟩:⟨*GID*⟩:⟨*GECOS*⟩:⟨*Heimatverzeichnis*⟩:⟨*Shell*⟩
```
hugo:x:1000:1000:Hugo Schulz:/home/hugo:/bin/sh
```

⟨*Benutzername*⟩ Dieser Name sollte aus Kleinbuchstaben und Ziffern bestehen; das erste Zeichen sollte ein Buchstabe sein. Unix-Systeme unterscheiden oft nur die ersten 8 Zeichen – Linux hat diese Einschränkung nicht, aber in heterogenen Netzen sollten Sie darauf Rücksicht nehmen.

Widerstehen Sie der Versuchung, Umlaute, Satzzeichen und ähnliches in Benutzernamen aufzunehmen, selbst falls das System sie durchläßt – nicht alle Werkzeuge zum Anlegen neuer Benutzer sind pingelig, und Sie könnten `/etc/passwd` ja auch direkt ändern. Was auf den ersten Blick prächtig zu funktionieren scheint, kann später anderswo zu Problemen führen.

Ebenfalls Abstand nehmen sollten Sie von Benutzernamen, die nur aus Großbuchstaben oder nur aus Ziffern bestehen. Erstere machen möglicherweise Probleme beim Anmelden (siehe Übung 13.6), letztere können zu Verwirrung führen, vor allem wenn der numerische Benutzername nicht mit der numerischen UID des Kontos übereinstimmt. Programme wie »`ls -l`« zeigen nämlich die UID an, wenn es für die betreffende UID keinen Eintrag in `/etc/passwd` gibt, und es ist nicht unbedingt einfach, UIDs in der `ls`-Ausgabe von rein numerischen Benutzernamen zu unterscheiden.

⟨*Kennwort*⟩ Traditionell steht hier das verschlüsselte Kennwort des Benutzers. Unter Linux sind heute »Schattenkennwörter« (***shadow passwords***) üblich; statt das Kennwort in der allgemein lesbaren `/etc/passwd`-Datei abzulegen, steht es in der Datei `/etc/shadow` gespeichert, auf die nur der Administrator und einige privilegierte Prozesse Zugriff haben. In `/etc/passwd` macht ein »x« auf diesen Umstand aufmerksam. Jedem Benutzer steht das Kommando `passwd` zur Verfügung, um sein Kennwort selbst zu verändern. *shadow passwords*

Linux erlaubt verschiedene Verfahren zur Verschlüsselung von Kennwörtern. Klassisch ist das von Unix übernommene und von DES abgeleitete `crypt`-Verfahren; solche Kennwörter erkennen Sie daran, dass sie in verschlüsselter Form genau 13 Zeichen lang sind. Ebenfalls verbreitet sind die sogenannten MD5-Kennwörter; ihre verschlüsselte Form ist länger und beginnt immer mit der Zeichenfolge `$1$`. Manche Linux-Versionen unterstützen weitere Verfahren.

⟨*UID*⟩ Die numerische Benutzerkennung – eine Zahl zwischen 0 und $2^{32}-1$. Nach Konvention sind UIDs zwischen 0 und 99 (einschließlich) für das System reserviert, UIDs zwischen 100 und 499 können an Softwarepakete ausgegeben werden, falls diese Pseudobenutzer benötigen. UIDs für »echte« Benutzer haben bei den meisten Distributionen Werte ab 1000.

Eben weil die Benutzer im System nicht durch die Namen, sondern durch die UID unterschieden werden, behandelt der Kernel intern zwei Konten als völlig identisch, wenn sie unterschiedliche Benutzernamen aber dieselbe UID haben – jedenfalls was die Zugriffsrechte angeht. Bei den Kommandos, die einen Benutzernamen anzeigen (etwa »`ls -l`« oder `id`), wird in solchen Fällen immer der Benutzername verwendet, der beim Anmelden angegeben wurde.

⟨*GID*⟩ Die GID der **primären Gruppe** des Benutzers nach dem Anmelden. primäre Gruppe

Bei den Novell/SUSE- und manchen anderen Distributionen wird eine bestimmte Gruppe, hier beispielsweise `users`, als gemeinsame Standardgruppe für alle Benutzer eingetragen. Diese Methode ist einfach zu verstehen und hat Tradition.

Bei vielen Distributionen, etwa denen von Red Hat oder Debian GNU/Linux, wird für jeden neuen Benutzer automatisch eine eigene Gruppe angelegt, die die gleiche GID hat wie die UID des Benutzerkontos. Die Idee dahinter ist, eine differenziertere Rechtevergabe zu ermöglichen als bei dem Ansatz, alle Benutzer in dieselbe Gruppe `users` zu tun. Denken Sie an die folgende Situation: Emil (Benutzername `emil`) ist der persönliche Assistent der Vorstandsvorsitzenden Susi (Benutzername `susi`). In dieser Funktion muss er hin und wieder auf Dateien zugreifen, die in Susis Heimatverzeichnis gespeichert sind, die aber alle anderen Benutzer nichts angehen. Der von Red Hat, Debian & Co. verfolgte Ansatz »Eine Gruppe pro Benutzer« macht es einfach, den Benutzer `emil` in die *Gruppe* `susi` zu tun und dafür zu sorgen, dass Susis Dateien für alle Gruppenmitglieder lesbar sind (der Standardfall), aber nicht für den »Rest der Welt«. Im Ansatz »Eine Gruppe für alle« wäre es nötig, eine ganz neue Gruppe einzuführen und die Konten `emil` und `susi` entsprechend umzukonfigurieren.

Jeder Benutzer muss durch die Zuordnung in der Datei `/etc/passwd` Mitglied mindestens einer Benutzergruppe sein.

Die sekundären Gruppen (soweit vorhanden) des Benutzers werden durch entsprechende Einträge in der Datei `/etc/group` festgelegt.

⟨*GECOS*⟩ Dies ist das Kommentarfeld, auch *GECOS-Feld* genannt.

GECOS steht für *General Electric Comprehensive Operating System* und hat nichts mit Linux zu tun, abgesehen davon, dass man in diesem Feld in der Frühzeit von Unix Informationen eingefügt hat, die für die Kompatibilität mit einigen Jobversanddiensten für GECOS-Rechner notwendig waren.

Das Feld enthält diverse Informationen über den Benutzer, vor allem seinen »richtigen« Namen und optionale Informationen wie die Zimmer- oder Telefonnummer. Diese Information wird von Programmen wie `mail` und `finger` benutzt. Oft wird der volle Name von News- und Mail-Programmen bei der Zusammenstellung der Absenderadresse verwendet.

Theoretisch gibt es ein Programm namens `chfn`, mit dem Sie als Benutzer den Inhalt Ihres GECOS-Feldes ändern können. Ob das im Einzelfall klappt, ist eine andere Frage, da man zumindest in Firmen Leuten nicht notwendigerweise erlauben will, ihren Namen beliebig zu modifizieren.

⟨*Heimatverzeichnis*⟩ Das hier benannte Verzeichnis ist der persönliche Bereich des Benutzers, in dem er seine eigenen Dateien aufbewahren kann. Ein neu erstelltes Heimatverzeichnis ist selten leer, denn üblicherweise erhält ein neuer Benutzer vom Administrator einige Profildateien als Erstausstattung. Wenn ein Benutzer sich anmeldet, benutzt seine Shell das Heimatverzeichnis als aktuelles Verzeichnis, das heißt, der Benutzer befindet sich unmittelbar nach der Anmeldung zunächst dort.

⟨*Login-Shell*⟩ Der Name des Programms, das von `login` nach erfolgreicher Anmeldung gestartet werden soll – das ist in der Regel eine Shell. Das siebte Feld reicht bis zum Zeilenende.

Der Benutzer kann mit dem Programm `chsh` diesen Eintrag selbst ändern. Die erlaubten Programme (Shells) sind in der Datei `/etc/shells` aufgelistet. Wenn ein Benutzer keine interaktive Shell haben soll, kann auch ein beliebiges anderes Programm mit allen Argumenten in dieses Feld eingetragen werden (ein gängiger Kandidat ist `/bin/true`). Das Feld kann auch leer bleiben. Dann wird automatisch die Standardshell `/bin/sh` gestartet.

Wenn Sie sich unter einer grafischen Oberfläche anmelden, dann werden normalerweise alle möglichen Programme für Sie gestartet, aber nicht notwendigerweise eine interaktive Shell. Der Shell-Eintrag in `/etc/passwd` kommt aber zum Beispiel zum Tragen, wenn Sie ein Terminal-Emulationsprogramm wie `xterm` oder `konsole` aufrufen, denn diese Programme orientieren sich normalerweise an diesem, um Ihre bevorzugte Shell zu identifizieren.

Einige der hier gezeigten Felder können leer bleiben. Absolut notwendig sind nur Benutzername, UID, GID und Heimatverzeichnis. Für die meisten Benutzerkonten werden alle diese Felder ausgefüllt sein, aber Pseudobenutzer benutzen eventuell nur einen Teil der Felder.

Heimatverzeichnisse

Die Heimatverzeichnisse stehen üblicherweise unter `/home` und heißen so wie der Benutzername des Besitzers. In der Regel ist das eine ganz nützliche Übereinkunft, die dazu beiträgt, dass das Heimatverzeichnis eines bestimmten Benutzers leicht zu finden ist. Theoretisch kann ein Heimatverzeichnis aber an beliebiger Stelle im System stehen, und der Name ist auch beliebig.

Bei großen Systemen ist es gängig, zwischen `/home` und dem Benutzernamen-Verzeichnis noch eine oder mehrere Zwischenebenen einzuführen, etwa

`/home/pers/hugo`	*Hugo aus der Personalabteilung*
`/home/entw/susi`	*Susi aus der Entwicklungsabteilung*
`/home/vorst/heinz`	*Heinz aus dem Vorstand*

Dafür gibt es mehrere Gründe. Zum einen ist es so leichter möglich, die Heimatverzeichnisse einer Abteilung auf einem Server der Abteilung zu halten, sie aber gegebenenfalls auf anderen Client-Rechnern zugänglich zu machen. Zum anderen waren Unix-Dateisysteme (und manche Linux-Dateisysteme) langsam im Umgang mit Verzeichnissen, die sehr viele Dateien enthalten, was sich bei einem `/home` mit mehreren tausend Einträgen in unschöner Weise bemerkbar gemacht hätte. Mit heute aktuellen Linux-Dateisystemen (`ext3` mit `dir_index` und ähnlichem) ist letzteres jedoch kein Problem mehr.

Beachten Sie, dass Sie als Administrator die Datei `/etc/passwd` nicht unbedingt direkt von Hand editieren müssen. Es gibt eine Reihe von Programmen, die Ihnen bei der Einrichtung und Pflege der Benutzerkonten helfen.

Werkzeuge

Prinzipiell ist es auch möglich, die Benutzerdatenbank anderswo zu lagern als in `/etc/passwd`. Auf Systemen mit sehr vielen Benutzern (Tausenden) ist eine Speicherung etwa in einer relationalen Datenbank vorzuziehen, während sich in heterogenen Netzen eine gemeinsame Benutzerverwaltung für unterschiedliche Plattformen etwa auf der Basis eines LDAP-Verzeichnisses anbietet. Die Details würden allerdings den Rahmen dieses Kurses sprengen.

13.2.2 Die Datei `/etc/shadow`

Aus Sicherheitsgründen werden bei fast allen aktuellen Linux-Distributionen die Benutzerkennwörter in verschlüsselter Form in der Datei `/etc/shadow` gespeichert (engl. *shadow passwords*, »Schattenkennwörter«). Die Datei ist für normale Benutzer nicht lesbar; nur `root` darf sie schreiben, während außer ihm auch die Mitglieder der Gruppe `shadow` die Datei lesen dürfen. Wenn Sie sich die Datei als normaler Benutzer anzeigen lassen wollen, erhalten Sie eine Fehlermeldung.

Die Verwendung von `/etc/shadow` ist nicht Pflicht, aber sehr dringend empfohlen. Allerdings kann es Systemkonfigurationen geben, bei denen die durch Schattenkennwörter erreichte Sicherheit wieder zunichte gemacht wird, etwa wenn Benutzerdaten über NIS an andere Rechner exportiert werden (vor allem in heterogenen Unix-Umgebungen).

Für jeden Benutzer ist in dieser Datei wieder genau eine Zeile eingetragen, das Format ist

Format

⟨*Benutzername*⟩:⟨*Kennwort*⟩:⟨*Änderung*⟩:⟨*Min*⟩:⟨*Max*⟩▷
◁:⟨*Warnung*⟩:⟨*Frist*⟩:⟨*Sperre*⟩:⟨*Reserviert*⟩

Zum Beispiel:

```
root:gaY2L19jxzHj5:10816:0:10000::::
daemon:*:8902:0:10000::::
hugo:GodY6c5pZk1xs:10816:0:10000::::
```

Im folgenden die Bedeutung der einzelnen Felder:

⟨*Benutzername*⟩ Entspricht einem Eintrag in der Datei `/etc/passwd`. Dieses Feld »verbindet« die beiden Dateien.

⟨*Kennwort*⟩ Das verschlüsselte Kennwort des Benutzers. Ein leerer Eintrag bedeutet in der Regel, dass der Benutzer sich ohne Kennwort anmelden kann. Steht hier ein Stern oder ein Ausrufungszeichen, kann der betreffende Benutzer sich nicht anmelden. Es ist auch üblich, Benutzerkonten zu sperren, ohne sie komplett zu löschen, indem man einen Stern oder ein Ausrufungszeichen an den Anfang des zugehörigen Kennworts setzt.

⟨*Änderung*⟩ Das Datum der letzten Änderung des Kennworts, in Tagen seit dem 1. Januar 1970.

⟨*Min*⟩ Minimale Anzahl von Tagen, die seit der letzten Kennwortänderung vergangen sein müssen, damit das Kennwort wieder geändert werden kann.

⟨*Max*⟩ Maximale Anzahl von Tagen, die ein Kennwort ohne Änderung gültig bleibt. Nach Ablauf dieser Frist muss der Benutzer sein Kennwort ändern.

⟨*Warnung*⟩ Die Anzahl von Tagen vor dem Ablauf der ⟨*Max*⟩-Frist, an denen der Benutzer eine Warnung erhält, dass er sein Kennwort bald ändern muss, weil die maximale Anzahl abläuft. Die Meldung erscheint in der Regel beim Anmelden.

⟨*Frist*⟩ Die Anzahl von Tagen ausgehend vom Ablauf der ⟨*Max*⟩-Frist, nach der das Konto automatisch gesperrt wird, wenn der Benutzer nicht vorher sein Kennwort ändert. (In der Zeit zwischen dem Ende der ⟨*Max*⟩-Frist und dem Ende dieser Frist kann der Benutzer sich anmelden, muss aber sofort sein Kennwort ändern.)

⟨*Sperre*⟩ Das Datum, an dem das Konto definitiv gesperrt wird, wieder in Tagen seit dem 1. Januar 1970.

Verschlüsselung von Kennwörtern

Kurz noch ein paar Anmerkungen zum Thema »Verschlüsselung von Kennwörtern«. Man könnte auf den Gedanken kommen, dass die Kennwörter, wenn sie verschlüsselt sind, auch wieder *entschlüsselt* werden können. Einem cleveren Cracker, dem die Datei `/etc/shadow` in die Hände fällt, würden so sämtliche Benutzerkonten des Systems offen stehen. Allerdings ist das in Wirklichkeit nicht so, denn die »Verschlüsselung« der Kennwörter ist eine Einbahnstraße: Es ist nicht möglich, aus der »verschlüsselten« Darstellung eines Linux-Kennworts die unverschlüsselte zurückzugewinnen, da das verwendete Verfahren das wirkungsvoll verhindert. Die einzige Möglichkeit, die »Verschlüsselung« zu »knacken«, besteht darin, potenzielle Kennwörter zur Probe zu verschlüsseln und zu schauen, ob dasselbe herauskommt wie das, was in `/etc/shadow` steht.

Nehmen wir mal an, Sie wählen die Zeichen Ihres Kennworts aus den 95 sichtbaren Zeichen des ASCII (es wird zwischen Groß- und Kleinschreibung unterschieden). Das bedeutet, es gibt 95 verschiedene einstellige Kennwörter, $95^2 = 9025$ zweistellige und so weiter. Bei acht Stellen sind Sie schon bei 6,6 Billiarden ($6{,}6 \cdot 10^{15}$) Möglichkeiten. Angenommen, Sie könnten 10 Millionen Kennwörter in der Sekunde zur Probe verschlüsseln (für das traditionelle Verfahren auf heutiger Hardware absolut nicht unrealistisch). Dann müßten Sie knapp 21 Jahre einkalkulieren, um alle Möglichkeiten durchzuprobieren. Wenn Sie in der glücklichen Lage sind, eine moderne Grafikkarte zu besitzen, ist da durchaus noch ein Faktor 50–100 drin, so dass daraus gut zwei Monate werden. Und dann gibt es ja noch nette Dienste wie Amazons EC2, die Ihnen (oder irgendwelchen Crackern) fast beliebige Rechenleistung auf Abruf zur Verfügung stellen, oder das freundliche Russen-Botnet … Fühlen Sie sich also nicht zu sicher.

Es gibt noch ein paar andere Probleme: Das traditionelle Verfahren (meist »crypt« oder »DES« genannt – letzteres, weil es ähnlich zu, aber nicht iden-

tisch mit, dem gleichnamigen Verschlüsselungsverfahren ist[3]) sollten Sie nicht mehr verwenden, wenn Sie es vermeiden können. Es hat nämlich die unangenehme Eigenschaft, nur die ersten acht Zeichen jedes Kennworts zu bearbeiten, und der clevere Cracker kann inzwischen genug Plattenplatz kaufen, um jedenfalls die gängigsten 50 Millionen (oder so) Kennwörter »auf Vorrat« zu verschlüsseln. Zum »Knacken« muss er so nur noch in seinem Vorrat nach der verschlüsselten Form suchen, was sehr schnell geht, und kann dann einfach den Klartext ablesen.

Um das Ganze noch etwas aufwendiger zu machen, wird beim Verschlüsseln eines neu eingegebenen Kennworts traditionell noch ein zufälliges Element addiert (das sogenannte *salt*), das dafür sorgt, dass eine von 4096 Möglichkeiten für das verschlüsselte Kennwort gewählt wird. Der Hauptzweck des *salt* besteht darin, »Zufallstreffer« zu vermeiden, die sich ergeben, wenn Benutzer *X* aus irgendwelchen Gründen einen Blick auf den Inhalt von `/etc/shadow` wirft und feststellt, dass sein verschlüsseltes Kennwort genauso aussieht wie das von Benutzer *Y* (so dass er sich mit seinem *Klartextkennwort* auch als Benutzer *Y* anmelden kann). Als angenehmer Nebeneffekt wird der Plattenplatz für das Cracker-Wörterbuch aus dem vorigen Absatz um den Faktor 4096 in die Höhe getrieben.

Die gängige Methode zur Kennwortverschlüsselung basiert heute auf dem MD5-Algorithmus, erlaubt beliebig lange Kennwörter und verwendet ein 48-Bit-*salt* statt den traditionellen 12 Bit. Netterweise ist das Verfahren auch wesentlich langsamer zu berechnen als »crypt«, was für den üblichen Zweck – die Prüfung beim Anmelden – ohne Bedeutung ist (man kann immer noch ein paar hundert Kennwörter pro Sekunde verschlüsseln), aber die cleveren Cracker schon behindert. (Sie sollten sich übrigens nicht davon irre machen lassen, dass Kryptografen das MD5-Verfahren als solches heutzutage wegen seiner Unsicherheit verpönen. Für die Anwendung zur Kennwortverschlüsselung ist das ziemlich irrelevant.)

Von den verschiedenen Parametern zur Kennwortverwaltung sollten Sie sich nicht zuviel versprechen. Sie werden zwar von der Login-Prozedur auf der Textkonsole befolgt, aber ob sie an anderen Stellen im System (etwa beim grafischen Anmeldebildschirm) genauso beachtet werden, ist systemabhängig. Ebenso bringt es in der Regel nichts, den Anwendern in kurzen Abständen neue Kennwörter aufzuzwingen – meist ergibt sich dann eine Folge der Form `susi1`, `susi2`, `susi3`, … oder sie alternieren zwischen zwei Kennwörtern. Eine *Mindestfrist* gar, die verstreichen muss, bevor ein Benutzer sein Kennwort ändern kann, ist geradezu gefährlich, weil sie einem Cracker möglicherweise ein »Zeitfenster« für unerlaubte Zugriffe einräumt, selbst wenn der Benutzer weiß, dass sein Kennwort kompromittiert wurde.

Das Problem, mit dem Sie als Administrator zu kämpfen haben, ist in der Regel nicht, dass Leute versuchen, die Kennwörter auf Ihrem System mit »roher Gewalt« zu knacken. Viel erfolgversprechender ist in der Regel sogenanntes *social engineering*. Um Ihr Kennwort zu erraten, beginnt der clevere Cracker natürlich nicht mit `a`, `b`, und so weiter, sondern mit den Vornamen Ihres Ehegesponses, Ihrer Kinder, Ihrem Autokennzeichen, dem Geburtsdatum Ihres Hundes et cetera. (Wir wollen Ihnen natürlich in keiner Weise unterstellen, dass *Sie* so ein dummes Kennwort verwenden. Neinnein, *Sie* doch ganz bestimmt nicht! Bei Ihrem Chef sind wir uns da allerdings schon nicht mehr ganz so sicher …) Und dann gibt es

[3] Wenn Sie es genau wissen müssen: Das Klartext-Kennwort fungiert als Schlüssel (!) zur Verschlüsselung einer konstanten Zeichenkette (typischerweise einem Vektor von Nullbytes). Ein DES-Schlüssel hat 56 Bit, das sind gerade 8 Zeichen zu je 7 Bit (denn das höchstwertige Bit im Zeichen wird ignoriert). Dieser Prozess wird insgesamt fünfundzwanzigmal wiederholt, wobei immer die Ausgabe als neue Eingabe dient. Genau genommen ist das verwendete Verfahren auch nicht wirklich DES, sondern an ein paar Stellen abgewandelt, damit es weniger leicht möglich ist, aus kommerziell erhältlichen DES-Verschlüsselungs-Chips einen Spezialcomputer zum Knacken von Kennwörtern zu bauen.

da natürlich noch das altgediente Mittel des Telefonanrufs: »Hallo, hier ist die IT-Abteilung. Wir müssen unsere Systemsicherheitsmechanismen testen und brauchen dazu ganz dringend Ihren Benutzernamen und Ihr Kennwort.«

Es gibt diverse Mittel und Wege, Linux-Kennwörter sicherer zu machen. Neben dem oben genannten verbesserten Verfahren, das die meisten Linux-Systeme heute standardmäßig anwenden, gehören dazu Ansätze wie (zu) simple Kennwörter schon bei der Vergabe anzumeckern oder proaktiv Software laufen zu lassen, die schwache verschlüsselte Kennwörter zu identifizieren versucht, so wie das clevere Cracker auch machen würden (*Vorsicht:* Machen Sie sowas in der Firma nur mit der schriftlichen (!) Rückendeckung Ihres Chefs!). Andere Methoden vermeiden Kennwörter komplett zugunsten von ständig wechselnden magischen Zahlen (Stichwort: SecurID) oder Smartcards. All das würde in dieser Schulungsunterlage zu weit führen, und wir verweisen Sie darum auf die Unterlage *Linux-Sicherheit*.

13.2.3 Die Datei `/etc/group`

Gruppendatenbank

Gruppeninformationen hinterlegt Linux standardmäßig in der Datei `/etc/group`. Diese Datei enthält für jede Gruppe auf dem System einen einzeiligen Eintrag, der ähnlich wie die Einträge in `/etc/passwd` aus Feldern besteht, die durch einen Doppelpunkt »:« voneinander getrennt sind. `/etc/group` enthält genauer gesagt vier Felder pro Zeile:

⟨*Gruppenname*⟩:⟨*Kennwort*⟩:⟨*GID*⟩:⟨*Mitglieder*⟩

Deren Bedeutung ergibt sich wie folgt:

⟨*Gruppenname*⟩ Der textuelle Name der Gruppe, für die Verwendung in Verzeichnislisten usw.

⟨*Kennwort*⟩ Ein optionales Kennwort für diese Gruppe. Damit können auch Benutzer, die nicht per `/etc/shadow` oder `/etc/group` Mitglied der Gruppe sind, mit dem Befehl `newgrp` diese Gruppenzugehörigkeit annehmen. Ein »*« als ungültiges Zeichen verhindert einen Gruppenwechsel von normalen Benutzer in die betreffende Gruppe. Ein »x« verweist auf die separate Kennwortdatei `/etc/gshadow`.

⟨*GID*⟩ Die numerische Gruppenkennung für diese Gruppe

⟨*Mitglieder*⟩ Eine durch Kommas getrennte Liste mit Benutzernamen. Die Liste enthält alle Benutzer, die diese Gruppe als sekundäre Gruppe haben, die also zu dieser Gruppe gehören, aber im GID-Feld der Datei `/etc/passwd` einen anderen Wert stehen haben. (Benutzer mit dieser Gruppe als primärer Gruppe dürfen hier auch stehen, aber das ist unnötig.)

Eine `/etc/group`-Datei könnte zum Beispiel so aussehen:

```
root:x:0:root
bin:x:1:root,daemon
users:x:100:
projekt1:x:101:hugo,susi
projekt2:x:102:emil
```

administrative Gruppen

Die Einträge für die Gruppen `root` und `bin` sind Einträge für administrative Gruppen, ähnlich den imaginären Benutzerkonten auf dem System. Gruppen wie diesen sind viele Dateien auf dem System zugeordnet. Die anderen Gruppen enthalten Benutzerkonten.

GID-Werte

Ähnlich wie bei den UIDs werden auch die GIDs von einer bestimmten Zahl an, typischerweise 100, hochgezählt. Für einen gültigen Eintrag müssen mindestens das erste und dritte Feld (Gruppenname und GID) vorhanden sein. Durch so einen

Eintrag wird einer Gruppennummer, die in der Kennwortdatei einem Benutzer zugeordnet wurde, ein Gruppenname gegeben.

Die Felder für Kennwort und/oder Benutzerliste müssen nur für solche Gruppen ausgefüllt werden, die Benutzern als sekundäre Gruppe zugeordnet sind. Die in der Mitgliederliste eingetragenen Benutzer werden nicht nach einem Kennwort gefragt, wenn sie mit dem Kommando `newgrp` die aktuelle GID wechseln wollen. Wenn im zweiten Feld des Gruppeneintrags ein verschlüsseltes Kennwort eingetragen ist, können Anwender ohne Eintrag in der Mitgliederliste sich mit dem Kennwort legitimieren, um die Gruppenzugehörigkeit anzunehmen.

Mitgliederliste

Gruppenkennwort

In der Praxis werden Gruppenkennwörter so gut wie nie verwendet, da der Verwaltungsaufwand den daraus zu ziehenden Nutzen kaum rechtfertigt. Es ist einerseits bequemer, den jeweiligen Benutzern die betreffende Gruppe direkt als sekundäre Gruppe zuzuordnen (seit dem Linux-Kernel 2.6 gibt es ja keine Beschränkung für die Anzahl der sekundären Gruppen mehr), und andererseits folgt aus einem *einzelnen* Kennwort, das *alle* Gruppenmitglieder kennen müssen, nicht wirklich viel Sicherheit.

Wenn Sie sichergehen wollen, dann sorgen Sie dafür, dass in allen Gruppenkennwort-Feldern ein Stern (»*«) steht.

13.2.4 Die Datei `/etc/gshadow`

Wie bei der Kennwortdatei gibt es auch für die Gruppendatei eine Erweiterung durch das Schattenkennwortsystem. Die Gruppenkennwörter, die sonst in der Datei `/etc/group` analog zu `/etc/passwd` verschlüsselt, aber für alle Benutzer lesbar abgelegt sind, werden dann in der separaten Datei `/etc/gshadow` gespeichert. Dort werden auch zusätzliche Informationen zur Gruppe festgehalten, beispielsweise die Namen der zum Ein- und Austragen von Mitgliedern autorisierten Gruppenverwalter.

13.2.5 Das Kommando `getent`

Natürlich können Sie die Dateien `/etc/passwd`, `/etc/shadow` und `/etc/group` wie alle anderen Textdateien auch mit Programmen wie `cat`, `less` oder `grep` lesen und verarbeiten (OK, OK, für `/etc/shadow` müssen Sie `root` sein). Dabei gibt es aber ein paar praktische Probleme:

- Eventuell bekommen Sie nicht die ganze Wahrheit zu sehen: Es könnte ja sein, dass die Benutzerdatenbank (oder Teile davon) auf einem LDAP-Server, in einer SQL-Datenbank oder einem Windows-Domänencontroller steht und Sie in `/etc/passwd` gar nichts Interessantes finden.
- Wenn Sie gezielt nach einem Benutzereintrag suchen wollen, ist das mit `grep` schon etwas umständlich zu tippen, wenn Sie »falsche Positive« ausschließen wollen.

Das Kommando `getent` macht es möglich, die verschiedenen Datenbanken für Benutzer- und Gruppeninformationen gezielt abzufragen. Mit

```
$ getent passwd
```

bekommen Sie etwas angezeigt, das aussieht wie `/etc/passwd`, aber aus allen Quellen für Benutzerdaten zusammengesetzt ist, die auf dem Rechner aktuell konfiguriert sind. Mit

```
$ getent passwd hugo
```

bekommen Sie den Benutzereintrag für den Benutzer `hugo`, egal wo er tatsächlich gespeichert ist. Statt `passwd` können Sie auch `shadow`, `group` oder `gshadow` angeben, um die betreffende Datenbank zu konsultieren. (Natürlich können Sie auch mit `getent` auf `shadow` und `gshadow` nur als `root` zugreifen.)

Der Begriff »Datenbank« ist hier zu verstehen als »Gesamtheit aller Quellen, aus denen die C-Bibliothek sich Informationen zu diesem Thema (etwa Benutzer) zusammensucht«. Wenn Sie wissen wollen, wo genau diese Informationen herkommen (können), dann lesen Sie `nsswitch.conf`(5) und studieren Sie die Datei `/etc/nsswitch.conf` auf Ihrem System.

Sie dürfen auch mehrere Benutzer- oder Gruppennamen angeben. In diesem Fall werden die Informationen für alle genannten Benutzer oder Gruppen ausgegeben:

```
$ getent passwd hugo susi fritz
```

Übungen

13.5 [1] Welchen Wert finden Sie in der zweiten Spalte der Datei `/etc/passwd`? Warum finden Sie dort diesen Wert?

13.6 [2] Schalten Sie auf eine Textkonsole um (etwa mit Alt+F1) und versuchen Sie sich anzumelden, indem Sie Ihren Benutzernamen in reiner Großschreibung eingeben. Was passiert?

13.7 [2] Wie können Sie prüfen, dass für jeden Eintrag in der `passwd`-Datenbank auch ein Eintrag in der `shadow`-Datenbank vorhanden ist? (`pwconv` betrachtet nur die *Dateien* `/etc/passwd` und `/etc/shadow` und schreibt außerdem `/etc/shadow` neu, was wir hier nicht wollen.)

13.3 Benutzerkonten und Gruppeninformationen verwalten

Nachdem die Installation einer neuen Linux-Distribution abgeschlossen ist, gibt es zunächst nur das `root`-Konto für den Administrator und die Pseudobenutzer. Alle weiteren Benutzerkonten müssen erst eingerichtet werden (wobei die meisten Distributionen heutzutage den Installierer mit sanfter Gewalt dazu nötigen, zumindest *einen* »gewöhnlichen« Benutzer anzulaegen).

Als Administrator ist es Ihre Aufgabe, die Konten für alle Benutzer (real und imaginär) auf Ihrem System anzulegen und zu verwalten. Um dies zu erleichtern, bringt Linux einige Werkzeuge zur Benutzerverwaltung mit. Damit ist das zwar in den meisten Fällen eine problemlose, einfach zu erledigende Angelegenheit, aber es ist wichtig, dass Sie die Zusammenhänge verstehen.

Werkzeuge zur Benutzerverwaltung

13.3.1 Benutzerkonten einrichten

Der Vorgang bei der Einrichtung eines neuen Benutzerkontos ist im Prinzip immer gleich und erfolgt in mehreren Schritten:

1. Sie müssen Einträge in der Kennwortdatei `/etc/passwd` und gegebenenfalls in `/etc/shadow` anlegen.
2. Gegebenenfalls ist ein Eintrag (oder mehrere) in der Gruppendatei `/etc/group` nötig.
3. Sie müssen das Heimatverzeichnis erzeugen, eine Grundausstattung an Dateien hineinkopieren und alles dem neuen Benutzer übereignen.
4. Wenn nötig, müssen Sie den neuen Benutzer noch in weitere Listen eintragen, zum Beispiel für Plattenkontingente, Zugriffsberechtigung auf Datenbanken und spezielle Applikationen.

Alle Dateien, die beim Einrichten eines neuen Kontos bearbeitet werden, sind normale Textdateien. Sie können jeden Schritt ohne weiteres von Hand beziehungsweise mit Hilfe eines Texteditors durchführen. Da dies jedoch eine genauso dröge wie aufwendige Tätigkeit ist, tun Sie besser daran, sich vom System helfen zu lassen. Linux hält hierfür das Programm `useradd` bereit.

`useradd`

Im einfachsten Fall übergeben Sie dem Programm `useradd` lediglich den Namen des neuen Benutzers. Optional können Sie auch noch diverse andere Benutzerparameter setzen; für nicht angegebene Parameter (typischerweise zum Beispiel die UID) werden automatisch »vernünftige« Standardwerte gewählt. Auf Wunsch kann auch das Heimatverzeichnis des Benutzer erzeugt und mit einer Grundausstattung an Dateien versehen werden, die das Programm dem Verzeichnis `/etc/skel` entnimmt. Die Syntax von `useradd` ist:

```
useradd [⟨Optionen⟩] ⟨Benutzername⟩
```

Folgende Optionen stehen dabei unter anderem zur Verfügung:

`-c` ⟨*Kommentar*⟩ Eintrag in in das GECOS-Feld

`-d` ⟨*Heimatverzeichnis*⟩ Fehlt diese Angabe, wird `/home/`⟨*Benutzername*⟩ angenommen

`-e` ⟨*Datum*⟩ Datum, an dem der Zugang automatisch gesperrt wird (Format: »JJJJ-MM-TT«)

`-g` ⟨*Gruppe*⟩ Primäre Gruppe des neuen Benutzers, als Name oder GID. Die Gruppe muss existieren.

`-G` ⟨*Gruppe*⟩**[,**⟨*Gruppe*⟩**]...** Weitere Gruppen, als Namen oder GIDs. Die Gruppen müssen existieren.

`-s` ⟨*Shell*⟩ Login-Shell des Benutzers

`-u` ⟨*UID*⟩ Numerische Benutzerkennung des neuen Benutzers. Die UID darf noch nicht anderweitig vergeben sein, es sei denn, die Option »`-o`« wurde angegeben.

`-m` Legt das Heimatverzeichnis an und kopiert die Datei-Grundausstattung hinein. Diese Grundausstattung kommt aus `/etc/skel`, es sei denn, mit »`-k` ⟨*Verzeichnis*⟩« wurde ein anderes Verzeichnis benannt.

Mit dem Kommando

```
# useradd -c "Hugo Schulz" -m -d /home/hugo -g entw \
>    -k /etc/skel.entw hugo
```

zum Beispiel wird für den Benutzer Hugo Schulz ein Benutzerkonto namens `hugo` angelegt und der Gruppe `entw` zugeordnet. Sein Heimatverzeichnis wird als `/home/hugo` angelegt und die Dateien aus `/etc/skel.entw` als Grundausstattung dort hineinkopiert.

Mit der Option `-D` (bei den SUSE-Distributionen `--show-defaults`) können Sie Vorgabewerte für einige Aspekte neuer Benutzerkonten festlegen. Ohne Zusatzoptionen werden die Werte nur angezeigt:

```
# useradd -D
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/sh
SKEL=/etc/skel
CREATE_MAIL_SPOOL=no
```

Ändern können Sie diese Werte respektive mit den Optionen `-g`, `-b`, `-f`, `-e` und `-s`:

```
# useradd -D -s /usr/bin/zsh                    zsh als Standard-Shell
```

Die letzten beiden Werte in der Liste lassen sich nicht ändern.

`useradd` ist ein relativ niedrig ansetzendes Werkzeug. Im »wirklichen Leben« werden Sie als erfahrener Administrator neue Benutzerkonten wahrscheinlich nicht mit `useradd`, sondern über ein Shellskript anlegen, das Ihre örtlichen Richtlinien mit berücksichtigt (damit Sie nicht immer selber daran denken müssen). Dieses Shellskript müssen Sie leider selbst erstellen – jedenfalls solange Sie nicht Debian GNU/Linux oder eine davon abgeleitete Distribution benutzen (siehe unten).

Vorsicht: Zwar bringt jede ernstzunehmende Linux-Distribution ein Programm namens `useradd` mit, die Implementierungen unterscheiden sich jedoch im Detail.

Die Red-Hat-Distributionen enthalten eine ziemlich »gewöhnliche« Version von `useradd` ohne besondere Extras, die die oben besprochenen Eigenschaften mitbringt.

Das `useradd` der SUSE-Distributionen ist darauf eingerichtet, Benutzer wahlweise in einem LDAP-Verzeichnis statt in `/etc/passwd` anzulegen. (Aus diesem Grund wird die Option `-D` anderweitig verwendet, so dass sie nicht wie bei den anderen Distributionen zum Abfragen oder Setzen von Standardwerten zur Verfügung steht.) Die Details hierzu würden hier etwas zu weit führen.

Bei Debian GNU/Linux und Ubuntu existiert `useradd` zwar, die offizielle Methode zum Anlegen neuer Benutzerkonten ist jedoch ein Programm namens `adduser` (zum Glück gar nicht verwirrend). Der Vorteil von `adduser` besteht darin, dass es die Spielregeln von Debian GNU/Linux einhält und es außerdem ermöglicht, beim Anlegen von Benutzerkonten auch noch beliebige andere Aktionen für das neue Konto auszuführen. Zum Beispiel könnte automatisch ein Verzeichnis im Dokumentbaum eines Web-Servers angelegt werden, in dem der neue Benutzer (und nur dieser) Dateien veröffentlichen kann, oder der Benutzer könnte automatisch zum Zugriff auf einen Datenbankserver autorisiert werden. Die Details stehen in `adduser`(8) und `adduser.conf`(5).

Kennwort

Nach dem Anlegen mit `useradd` ist das neue Konto noch nicht zugänglich; der Systemverwalter muss erst noch ein Kennwort eintragen. Das erklären wir als Nächstes.

13.3.2 Das Kommando `passwd`

Der Befehl `passwd` dient der Vergabe von Benutzerkennwörtern. Wenn Sie als `root` angemeldet sind, dann fragt

```
# passwd hugo
```

nach einem neuen Kennwort für den Benutzer `hugo` (Sie müssen es zweimal angeben, da keine Kontrollausgabe erfolgt).

Das `passwd`-Kommando steht auch normalen Benutzern zur Verfügung, damit sie ihr eigenes Kennwort ändern können (das Ändern der Kennwörter anderer Benutzer ist `root` vorbehalten):

```
$ passwd
Ändern des Passworts für hugo
(aktuelles) UNIX-Passwort: geheim123
Geben Sie ein neues UNIX-Passwort ein: 321mieheg
Geben Sie das neue UNIX-Passwort erneut ein: 321mieheg
passwd: Passwort erfolgreich geändert
```

Normale Benutzer müssen ihr eigenes Kennwort erst einmal richtig angeben, bevor sie ein neues setzen dürfen. Das soll Spaßvögeln das Leben schwer machen, die an Ihrem Computer herumspielen, wenn Sie mal ganz dringend raus mussten und die Bildschirmsperre nicht eingeschaltet haben.

`passwd` dient nebenbei auch noch zur Verwaltung verschiedener Einstellungen in `/etc/shadow`. Sie können sich zum Beispiel den »Kennwortstatus« eines Benutzers anschauen, indem Sie den Befehl `passwd` mit der Option `-S` aufrufen:

```
# passwd -S franz
franz LK 10/15/99 0 99999 7 0
```

Das erste Feld ist dabei (wieder mal) der Benutzername, dann folgt der Kennwortstatus: »`PS`« oder »`P`« heißen hier, dass ein Kennwort gesetzt ist; »`LK`« oder »`L`« stehen für ein gesperrtes Konto, und »`NP`« bezeichnet ein Konto ganz ohne Kennwort. Die übrigen Felder sind respektive das Datum der letzten Kennwortänderung, die Mindest- und Höchstfrist in Tagen für das Ändern des Kennworts, die Warnfrist vor dem Ablauf und die »Gnadenfrist« für die komplette Sperrung des Benutzerkontos nach dem Ablauf des Kennworts. (Siehe hierzu auch Abschnitt 13.2.2.)

Ändern können Sie einige dieser Einstellungen über Optionen von `passwd`. Hier sind ein paar Beispiele:

```
# passwd -l hugo                    Zugang sperren
# passwd -u hugo                    Zugang freigeben
# passwd -n 7 hugo                  Kennwortänderung höchstens alle 7 Tage
# passwd -x 30 hugo                 Kennwortänderung spätestens alle 30 Tage
# passwd -w 3 hugo                  3 Tage Warnfrist vor Kennwortablauf
```

Das Sperren und Freigeben per `-l` und `-u` funktioniert, indem vor das verschlüsselte Kennwort in `/etc/shadow` ein »`!`« gesetzt wird. Da bei der Kennwortverschlüsselung kein »`!`« entsteht, ist es unmöglich, beim Anmelden etwas einzugeben, was auf das »verschlüsselte Kennwort« in der Benutzerdatenbank passt – mithin ist der Zugang über die normale Anmeldeprozedur gesperrt. Sobald das »`!`« entfernt wird, gilt das ursprüngliche Kennwort wieder. (Genial, was?) Allerdings sollten Sie beachten, dass Benutzer sich möglicherweise auch noch auf andere Arten Zugang zum System verschaffen können, die ohne das verschlüsselte Kennwort in der Benutzerdatenbank auskommen – etwa über die Secure Shell mit Public-Key-Authentisierung.

Um die übrigen Einstellungen in `/etc/shadow` zu ändern, müssen Sie das Kommando `chage` heranziehen:

```
# chage -E 2009-12-01 hugo          Sperrung ab 1.12.2009
# chage -E -1 hugo                  Verfallsdatum aufheben
# chage -I 7 hugo                   Gnadenfrist 1 Woche nach Kennwortablauf
# chage -m 7 hugo                   Entspricht passwd -n (Grr.)
# chage -M 7 hugo                   Entspricht passwd -x (Grr, grr.)
# chage -W 3 hugo                   Entspricht passwd -w (Grr, grr, grr.)
```

(`chage` kann alle Parameter ändern, die `passwd` ändern kann, und dann noch ein paar.)

Wenn Sie sich die Optionen nicht merken können, dann rufen Sie `chage` einfach nur mit dem Namen eines Benutzerkontos auf. Das Programm präsentiert Ihnen dann nacheinander die aktuellen Werte zum Bestätigen oder Ändern.

Kennwort im Klartext

Ein bestehendes Kennwort im Klartext auslesen können Sie selbst als Administrator nicht mehr. Auch in der Datei `/etc/shadow` nachzusehen hilft in diesem Fall nichts, denn hier werden alle Kennwörter bereits verschlüsselt abgelegt. Sollte ein Benutzer sein Kennwort vergessen haben, reicht es in der Regel, dessen Kennwort mit `passwd` zu ändern.

Sollten Sie das `root`-Kennwort vergessen haben und gerade nicht als `root` angemeldet sein, bleibt Ihnen noch die Möglichkeit, Linux in eine Shell zu booten oder von einer Rettungsdiskette oder -CD zu booten. Anschließend können Sie mit einem Editor das ⟨*Kennwort*⟩-Feld des `root`-Eintrags in `/etc/passwd` löschen.

Übungen

13.8 [3] Ändern Sie das Kennwort von Benutzer `hugo`. Wie ändert sich die Datei `/etc/shadow`? Fragen Sie den Status zu diesem Kennwort ab.

13.9 [!2] Der Benutzer `trottel` hat sein Kennwort vergessen. Wie können Sie ihm helfen?

13.10 [!3] Stellen Sie die Bedingungen für das Kennwort von Benutzer `hugo` so ein, dass er sein Kennwort frühestens nach einer Woche und spätestens nach zwei Wochen ändern muss. Eine Warnung soll der Benutzer zwei Tage vor Ablauf dieser Zweiwochenfrist erhalten. Kontrollieren Sie anschließend die Einstellungen!

13.3.3 Benutzerkonten löschen

Um ein Benutzerkonto zu löschen, müssen Sie den Eintrag des Benutzers aus `/etc/passwd` und `/etc/shadow` entfernen, alle Verweise auf diesen Benutzer in `/etc/group` löschen und das Heimatverzeichnis sowie alle Dateien entfernen, die der Benutzer erstellt hat oder deren Eigentümer er ist. Wenn der Benutzer z. B. eine Mailbox für eingehende Nachrichten in `/var/mail` hat, sollte auch diese entfernt werden.

`userdel`

Auch für diese Aktionen existiert ein passendes Kommando. Der Befehl `userdel` löscht ein Benutzerkonto komplett. Die Syntax:

```
userdel [-r] ⟨Benutzername⟩
```

Die Option `-r` sorgt dafür, dass das Heimatverzeichnis des Benutzers mit Inhalt sowie seine Mailbox in `/var/mail` entfernt wird, andere Dateien des Benutzers – etwa `crontab`-Dateien usw. – müssen von Hand gelöscht werden. Eine schnelle Methode, die Dateien eines bestimmten Benutzers zu finden und zu löschen, ist der Befehl

```
find / -uid ⟨UID⟩ -delete
```

Ohne die Option `-r` werden nur die Benutzerdaten aus der Benutzerdatenbank gelöscht; das Heimatverzeichnis bleibt stehen.

13.3.4 Benutzerkonten und Gruppenzuordnung ändern

Eine Änderung der Benutzerkonten und -gruppen geschieht traditionell durch das Editieren der Dateien `/etc/passwd` und `/etc/group`. Viele Systeme enthalten auch Befehle wie `usermod` und `groupmod` für denselben Zweck, die Sie im Zweifelsfall vorziehen sollten, weil sie sicherer funktionieren und – meistens – auch bequemer einzusetzen sind.

Das Programm `usermod` akzeptiert im Wesentlichen dieselben Optionen wie `useradd`, aber ändert existierende Benutzerkonten, statt neue anzulegen. Beispielsweise können Sie mit

`usermod -g ⟨Gruppe⟩ ⟨Benutzername⟩`

die primäre Gruppe eines Benutzers ändern.

usermod

Achtung! Wenn Sie die UID eines bestehenden Benutzerkontos ändern möchten, können Sie natürlich das ⟨*UID*⟩-Feld in `/etc/passwd` direkt editieren. Allerdings sollten Sie gleichzeitig mit `chown` die Dateiberechtigungen der Dateien dieses Benutzers auf die neue UID übertragen: »`chown -R tux /home/tux`« vergibt die Eignerrechte an allen Dateien in dem Heimatverzeichnis, das von `tux` benutzt wurde, wieder an `tux`, nachdem Sie die UID für dieses Konto geändert haben. Wenn »`ls -l`« eine numerische UID statt eines alphanumerischen Namens ausgibt, weist dies darauf hin, dass mit der UID dieser Dateien kein Benutzername verbunden ist. Mit `chown` können Sie das in Ordnung bringen.

UID ändern

13.3.5 Die Benutzerdatenbank direkt ändern — `vipw`

Das Kommando `vipw` ruft einen Editor (`vi` oder einen anderen) auf, um die Datei `/etc/passwd` zu ändern. Gleichzeitig wird die jeweilige Datei gesperrt, so dass während dieser Änderung niemand z. B. mit `passwd` ebenfalls eine Änderung vornehmen kann (die dann verloren gehen würde). Mit der Option `-s` wird `/etc/shadow` bearbeitet.

Welcher Editor konkret aufgerufen wird, bestimmt der Wert der Umgebungsvariablen `VISUAL`, ersatzweise der Wert der Umgebungsvariablen `EDITOR`; existiert keine der beiden, wird der `vi` gestartet.

Übungen

13.11 [!2] Legen Sie den Benutzer `test` an. Wechseln Sie auf das Benutzerkonto `test` und legen Sie mit `touch` einige Dateien an, einige davon in einem anderen Ordner als dem Heimatverzeichnis (etwa `/tmp`). Wechseln Sie wieder zurück zu `root` und ändern Sie die UID von `test`. Was sehen Sie, wenn Sie mit `ls` die Dateien von Benutzer `test` auflisten?

13.12 [!2] Legen Sie einen Benutzer `test1` über das entsprechende grafische Tool an, einen anderen, `test2`, mit Hilfe des Kommandos `useradd` und einen weiteren, `test3`, von Hand. Betrachten Sie die Konfigurationsdateien. Können Sie problemlos unter allen drei Konten arbeiten? Legen Sie unter jedem Konto eine Datei an.

13.13 [!2] Löschen Sie das Konto von Benutzer `test2` und stellen Sie sicher, dass es auf dem System keine Dateien mehr gibt, die dem Benutzer gehören!

13.14 [2] Ändern Sie die UID des Benutzers `test1`. Was müssen Sie außerdem tun?

13.15 [2] Ändern Sie das Heimatverzeichnis für Benutzer `test1` um von `/home/test1` in `/home/user/test1`.

13.3.6 Anlegen, Ändern und Löschen von Gruppen

Genau wie Benutzerkonten können Sie Gruppen auf mehrere Arten anlegen. Die Methode »von Hand« gestaltet sich hier wesentlich weniger aufwendig als das manuelle Erstellen neuer Benutzerkonten: Da Gruppen kein Heimatverzeichnis

besitzen, genügt es normalerweise, die Datei `/etc/group` mit einem beliebigen Texteditor zu manipulieren und eine entsprechende neue Zeile einzufügen (siehe unten für `vigr`). Bei Verwendung von Gruppenkennwörtern ist auch noch ein Eintrag in `/etc/gshadow` vorzunehmen.

Übrigens spricht nichts dagegen, daß auch Benutzergruppen ein eigenes Verzeichnis erstellt bekommen. Dort können die Gruppenmitglieder dann die Früchte der gemeinsamen Arbeit ablegen. Die Vorgehensweise ist dabei dem Anlegen von Benutzerheimatverzeichnissen ähnlich, allerdings braucht keine Grundausstattung an Konfigurationsdateien kopiert werden.

Zur Gruppenverwaltung gibt es analog zu `useradd`, `usermod` und `userdel` die Programme `groupadd`, `groupmod` und `groupdel`, auf die Sie zurückgreifen sollten, statt `/etc/group` und `/etc/gshadow` direkt zu editieren. Mit `groupadd` können Sie einfach per Kommandozeilenparameter bzw. Voreinstellungen neue Gruppen im System erzeugen:

groupadd

```
groupadd [-g ⟨GID⟩] ⟨Gruppenname⟩
```

Die Option `-g` ermöglicht die Vorgabe einer bestimmten Gruppennummer. Wie erwähnt handelt es sich dabei um eine positive ganze Zahl. Die Werte bis 99 sind meist Systemgruppen vorbehalten. Ohne Angabe wird die nächste freie GID verwendet.

groupmod

Bereits bestehende Gruppen können Sie mit `groupmod` bearbeiten, ohne direkt in `/etc/group` schreiben zu müssen:

```
groupmod [-g ⟨GID⟩] [-n ⟨Name⟩] ⟨Gruppenname⟩
```

Die Option »`-g` ⟨*GID*⟩« ändert die GID der angegebenen Gruppe. Nicht aufgelöste Gruppenzugehörigkeiten von Dateien müssen danach manuell angepasst werden. Die Option »`-n` ⟨*Name*⟩« weist der angegebenen Gruppe einen neuen Gruppennamen zu. Die GID bleibt dabei unverändert, auch manuelle Anpassungen sind nicht erforderlich.

groupdel

Auch zum Entfernen der Gruppeneinträge existiert ein Hilfsprogramm. Dieses heißt konsequenterweise `groupdel`:

```
groupdel ⟨Gruppenname⟩
```

Ebenso wie beim manuellen Entfernen von Gruppeneinträgen empfiehlt es sich auch hier, anschließend alle Dateien des Verzeichnisbaums zu überprüfen und verwaiste Gruppenzugehörigkeiten per `chgrp` anzupassen. Primäre Gruppen von einzelnen Benutzern dürfen nicht entfernt werden. Entweder muss der betroffene Benutzer vor dem Entfernen der Gruppe ebenfalls ausgetragen oder einer anderen primären Gruppe zugeteilt werden.

gpasswd

Das `gpasswd`-Kommando dient in erster Linie zur Manipulation von Gruppenkennwörtern analog zum `passwd`-Kommando. Allerdings kann der Systemadministrator die Verwaltung der Mitgliederliste einer Gruppe an einen oder mehrere Gruppenadministratoren delegieren. Gruppenadministratoren verwenden dafür ebenfalls das `gpasswd`-Kommando:

Gruppenadministrator

```
gpasswd -a ⟨Benutzer⟩ ⟨Gruppe⟩
```

fügt den ⟨*Benutzer*⟩ der ⟨*Gruppe*⟩ hinzu, und

```
gpasswd -d ⟨Benutzer⟩ ⟨Gruppe⟩
```

entfernt ihn wieder daraus. Mit

```
gpasswd -A ⟨Benutzer⟩,… ⟨Gruppe⟩
```

kann der Systemadministrator Benutzer benennen, die als Gruppenadministratoren fungieren sollen.


In den SUSE-Distributionen ist `gpasswd` seit einer Weile nicht mehr enthalten. Statt dessen gibt es modifizierte Versionen der Programme zur Benutzer- und Gruppenverwaltung, die mit einem LDAP-Verzeichnis umgehen können.

Direkt ändern können Sie die Gruppendatenbank als Systemverwalter mit dem Kommando `vigr`. Es funktioniert analog zu `vipw`, ruft also einen Editor auf, mit dem Sie »exklusiven« Zugriff auf `/etc/group` haben. Entsprechend bekommen Sie mit »`vigr -s`« Zugriff auf `/etc/gshadow`. `vigr`

Übungen


13.16 [2] Wozu werden Gruppen gebraucht? Geben Sie mögliche Beispiele an!


13.17 [1] Können Sie ein Verzeichnis anlegen, auf das alle Mitglieder einer Gruppe Zugriff haben?

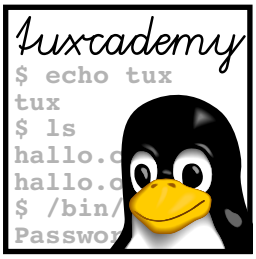

13.18 [!2] Erstellen Sie eine zusätzliche Gruppe `test`. Mitglied dieser Gruppe soll nur `test1` sein. Setzen Sie ein Gruppenkennwort. Melden Sie sich als `test1` und `test2` an und wechseln Sie jeweils in die neue Gruppe!

Kommandos in diesem Kapitel

adduser	Komfortables Kommando zum Anlegen neuer Benutzerkonten (Debian)	adduser(8)	198
chage	Setzt Kennwort-Attribute wie Ablaufdatum und Änderungsfristen	chage(1)	199
chfn	Erlaubt Benutzern das Ändern des GECOS-Felds in der Benutzerdatenbank	chfn(1)	190
getent	Ruft Einträge aus administrativen Datenbanken ab	getent(1)	195
gpasswd	Erlaubt Verwaltung von Gruppenmitgliederlisten durch Gruppenadministratoren und das Setzen des Gruppenkennworts	gpasswd(1)	202
groupadd	Fügt Gruppen in die Gruppendatenbank ein	groupadd(8)	202
groupdel	Löscht Gruppen aus der Gruppendatenbank	groupdel(8)	202
groupmod	Ändert Gruppeneinträge in der Gruppendatenbank	groupmod(8)	202
groups	Gibt die Gruppen aus, in denen ein Benutzer Mitglied ist	groups(1)	186
id	Gibt UID und GIDs eines Benutzers aus	id(1)	186
last	Zeige zuletzt angemeldete Benutzer an	last(1)	186
useradd	Fügt neue Benutzerkonten hinzu	useradd(8)	197
userdel	Entfernt Benutzerkonten	userdel(8)	200
usermod	Modifiziert die Benutzerdatenbank	usermod(8)	200
vigr	Erlaubt das exklusive Ändern von `/etc/group` bzw. `/etc/gshadow`	vipw(8)	203

Zusammenfassung

- Der Zugriff auf das System wird über Benutzerkonten geregelt.
- Jedes Benutzerkonto hat eine numerische UID und (mindestens) einen zugeordneten textuellen Benutzernamen.
- Benutzer können in Gruppen zusammengefasst werden. Gruppen haben Namen und numerische GIDs.
- »Pseudobenutzer« und »-gruppen« dienen zur weiteren Strukturierung der Zugriffsrechte.
- Die zentrale Benutzerdatenbank steht (normalerweise) in der Datei `/etc/passwd`.
- Die verschlüsselten Kennwörter der Benutzer stehen – zusammen mit anderen Kennwortparametern – in der Datei `/etc/shadow`, die nicht mehr für alle Benutzer lesbar ist.
- Gruppeninformationen stehen in den Dateien `/etc/group` und `/etc/gshadow`.
- Kennwörter werden mit dem Programm `passwd` verwaltet.
- Zur Konfiguration der Kennwortparameter in `/etc/shadow` dient das Programm `chage`.
- Benutzerinformationen ändert man mit dem Programm `vipw` oder – besser – mit den dafür vorgesehenen Werkzeugen `useradd`, `usermod` und `userdel`.
- Gruppeninformationen kann man über die Programme `groupadd`, `groupmod`, `groupdel` und `gpasswd` manipulieren.



14

Zugriffsrechte

Inhalt

Lernziele

- Das Linux-Rechtekonzept beherrschen
- Zugriffsrechte für Dateien und Verzeichnisse vergeben können
- Die Begriffe SUID, SGID und *sticky bit* kennen

Vorkenntnisse

- Kenntnisse über das Linux-Benutzer- und Gruppenkonzept (siehe Kapitel 13)
- Kenntnisse über das Linux-Dateikonzept

14.1 Das Linux-Rechtekonzept

Rechtekonzept

Überall, wo mehrere Benutzer auf einem System gleichzeitig arbeiten, muss auch ein Rechtekonzept für Prozesse, Dateien und Verzeichnisse existieren, damit Benutzer *A* nicht ohne weiteres auf Dateien von Benutzer *B* zugreifen kann. Linux implementiert dazu das Standardkonzept aller Unix-Betriebssysteme.

getrennte Rechte

In diesem Konzept sind jede Datei und jedes Verzeichnis genau einem Benutzer (Besitzer) und einer Gruppe zugeordnet. Jede Datei hat getrennte Rechte für den Besitzer, die Mitglieder der Gruppe, der die Datei zugeordnet ist (kurz »die Gruppe« genannt), und alle anderen Benutzer (der »Rest der Welt«). Für diese drei Mengen von Benutzern können jeweils separat Schreib-, Lese- und Ausführungsrechte vergeben werden. Der Eigentümer darf die Zugriffsrechte einer Datei bestimmen. Die Gruppe und alle anderen Benutzer haben nur dann Rechte an einer Datei, wenn der Eigentümer ihnen diese Rechte einräumt. Die gesamte Einstellung der Rechte für eine Datei heißt auch deren **Zugriffsmodus**.

Zugriffsmodus

Zugriffsbeschränkung

In einem Mehrbenutzersystem, in dem private oder gruppeninterne Daten auf einem allgemein zugänglichen Medium gespeichert werden, kann der Eigentümer einer Datei durch entsprechende Zugriffsbeschränkung eine Veränderung oder das Lesen seiner Daten verbieten. Die Rechte einer Datei können für den Eigentümer, die Benutzergruppe und die sonstigen Benutzer separat und unabhängig voneinander festgelegt werden. Dadurch lassen sich bereits mit Hilfe der Zugriffsrechte die Kompetenzen und Zuständigkeiten eines Gruppenarbeitsprozesses auf die Dateien, mit denen die Gruppe arbeitet, abbilden.

14.2 Zugriffsrechte auf Dateien und Verzeichnisse

14.2.1 Grundlagen

Für jede Datei und jedes Verzeichnis im System erlaubt Linux für jede der drei Kategorien von Benutzern – Eigentümer, Mitglieder der Dateigruppe, Rest der Welt – separate Zugriffsrechte. Diese Rechte teilen sich auf in Leserecht, Schreibrecht und Ausführungsrecht.

Zugriffsrechte für Dateien

Für Dateien bedeuten diese Rechte in etwa das, was ihre Namen aussagen: Wer Leserecht hat, darf sich den Inhalt der Datei anschauen, wer Schreibrecht hat, darf ihren Inhalt ändern. Das Ausführungsrecht ist notwendig, um die Datei als Programm starten zu können.

Zum Starten eines binären »Maschinenprogramms« ist nur Ausführungsrecht nötig. Für Dateien mit Shellskripten oder anderen »interpretierten« Programmen brauchen Sie außerdem Leserecht.

Zugriffsrechte für Verzeichnisse

Bei Verzeichnissen sieht es etwas anders aus: Leserecht ist notwendig, um den Inhalt eines Verzeichnisses anschauen zu können – etwa indem Sie das `ls`-Kommando ausführen. Schreibrecht brauchen Sie, um Dateien im Verzeichnis anlegen, löschen oder umbenennen zu können. Das »Ausführungsrecht« steht für die Möglichkeit, das Verzeichnis »benutzen« zu dürfen, in dem Sinne, dass Sie mit `cd` hineinwechseln oder seinen Namen in Pfadnamen von weiter unten im Dateibaum liegenden Dateien benutzen dürfen.

In Verzeichnissen, wo Sie nur das Leserecht haben, können Sie die Dateinamen lesen, aber nichts Anderes über die Dateien herausfinden. Haben Sie für ein Verzeichnis nur »Ausführungsrecht«, dann können Sie Dateien ansprechen, solange Sie wissen, wie sie heißen.

Normalerweise hat es wenig Sinn, Schreib- und Ausführungsrecht für Verzeichnisse getrennt voneinander zu vergeben; in gewissen Spezialfällen kann es allerdings nützlich sein.

Es ist wichtig, hervorzuheben, dass Schreibrecht auf eine *Datei* für das *Löschen* der Datei völlig unnötig ist – Sie brauchen Schreibrecht auf das *Verzeichnis*, in dem die Datei steht, und sonst nichts! Da beim »Löschen« nur ein Verweis auf die tatsächliche Dateiinformation (die Inode) aus dem Verzeichnis entfernt wird, handelt es sich dabei um eine reine Verzeichnisoperation. Das `rm`-Programm warnt Sie zwar, wenn Sie versuchen, eine Datei zu löschen, auf die Sie kein Schreibrecht haben, aber wenn Sie die Operation bestätigen und Schreibrecht auf das Verzeichnis der Datei haben, steht dem erfolgreichen Löschen nichts im Weg. (Linux kennt – wie jedes andere Unix-artige System auch – keine Möglichkeit, eine Datei direkt zu »löschen«; Sie können nur alle Verweise auf die Datei entfernen, woraufhin der Linux-Kernel von sich aus entscheidet, dass niemand mehr auf die Datei zugreifen kann, und ihren Inhalt entsorgt.)

Wenn Sie dagegen Schreibrecht auf die Datei, aber nicht auf ihr Verzeichnis haben, können Sie die Datei nicht komplett löschen. Sie können aber natürlich die Länge der Datei auf 0 setzen und damit den *Inhalt* entfernen, auch wenn die Datei selbst prinzipiell noch existiert.

Für jeden Benutzer gelten diejenigen Zugriffsrechte, die »am besten passen«. Haben zum Beispiel die Mitglieder der Gruppe, der eine Datei zugeordnet ist, kein Leserecht für die Datei, der »Rest der Welt« dagegen schon, dann dürfen die Gruppenmitglieder nicht lesen. Das auf den ersten Blick verlockende Argument, dass, wenn schon der Rest der Welt die Datei lesen darf, die Gruppenmitglieder, die ja in gewissem Sinne auch Teil des Rests der Welt sind, das eigentlich auch dürfen sollten, zählt nicht.

14.2.2 Zugriffsrechte anschauen und ändern

Informationen

Informationen darüber, welche Rechte, Benutzer- und Gruppenzuordnung für eine Datei gelten, bekommen Sie mit »`ls -l`«:

```
$ ls -l
-rw-r--r--  1  hugo  users   4711  Oct 4 11:11 datei.txt
drwxr-x---  2  hugo  group2  4096  Oct 4 11:12 testdir
```

Die Zeichenkette am linken Rand der Tabelle gibt die Zugriffsrechte für den Eigentümer, die Dateigruppe und den »Rest der Welt« an (das allererste Zeichen ist nur der Dateityp und hat mit den Zugriffsrechten nichts zu tun). Die dritte Spalte nennt den Benutzernamen des Eigentümers und die vierte Spalte den Namen der Dateigruppe.

In der Rechteliste stehen »`r`«, »`w`« und »`x`« jeweils für ein vorhandenes Lese-, Schreib- und Ausführungsrecht. Steht nur ein »`-`« in der Liste, dann hat die betreffende Kategorie das betreffende Recht nicht. »`rw-r--r--`« steht also für »Lese- und Schreibrecht für den Eigentümer, aber nur Leserecht für die Gruppenmitglieder und den Rest der Welt«.

Befehl `chmod`

Als Dateieigentümer können Sie mit dem Befehl `chmod` (von *change mode*, »Zugriffsmodus ändern«) die Zugriffsrechte für die Datei setzen. Dabei können Sie die drei Kategorien durch die Abkürzungen »`u`« (*user*) für den Eigentümer (Sie selbst), »`g`« (*group*) für die Mitglieder der Dateigruppe und »`o`« (*others*) für den »Rest der Welt« bestimmen. Die Rechte selbst werden durch die schon erwähnten Kürzel »`r`«, »`w`« und »`x`« festgelegt. Mit »`+`«, »`-`« und »`=`« können Sie verfügen, ob die benannten Rechte respektive zusätzlich zu den existierenden vergeben, von den existierenden »subtrahiert« oder anstelle der bisherigen gesetzt werden sollen. Zum Beispiel:

```
$ chmod u+x datei          Ausführungsrecht für Eigentümer
$ chmod go+w datei         setzt Schreibrecht für Gruppe und RdW
$ chmod g+rw datei         setzt Lese- und Schreibrecht für die Gruppe
```

```
$ chmod g=rw,o=r datei          setzt Lese- und Schreibrecht,
                                löscht Ausführungsrecht für die Gruppe
                                setzt reines Leserecht für den Rest der Welt
$ chmod a+w datei               äquivalent zu ugo+w
```

Tatsächlich sind Rechtespezifikationen um einiges komplexer. Konsultieren Sie die `info`-Dokumentation zu `chmod`, um die Details herauszufinden.

Der Dateieigentümer ist (neben `root`) der einzige Benutzer, der die Zugriffsrechte für eine Datei oder ein Verzeichnis ändern darf. Dieses Privileg ist unabhängig von den tatsächlichen Dateirechten; der Eigentümer darf sich selbst alle Rechte entziehen, aber hindert sich dadurch nicht selber daran, sich später wieder Rechte zu erteilen.

Die allgemeine Syntax des `chmod`-Kommandos ist

```
chmod [⟨Optionen⟩] ⟨Rechte⟩ ⟨Name⟩ …
```

Es können beliebig viele Datei- oder Verzeichnisnamen angegeben werden. Die wichtigsten Optionen sind:

`-R` Wenn ein Verzeichnis angegeben wurde, werden auch die Rechte von Dateien und Verzeichnissen innerhalb dieses Verzeichnisses geändert usw.

`--reference=`⟨*Name*⟩ Verwendet die Zugriffsrechte der Datei ⟨*Name*⟩. In diesem Fall müssen keine ⟨*Rechte*⟩ angegeben werden.

Numerische Rechtedarstellung

Sie können den Zugriffsmodus einer Datei statt wie eben angegeben »symbolisch« auch »numerisch« angeben. In der Praxis ist das sehr verbreitet, wenn Sie alle Rechte für eine Datei oder ein Verzeichnis auf einmal setzen wollen, und funktioniert so: Die drei Rechtetripel werden als dreistellige Oktalzahl dargestellt – die erste Ziffer beschreibt die Rechte des Eigentümers, die zweite die Rechte der Dateigruppe und die dritte die Rechte für den »Rest der Welt«. Jede dieser Ziffern ergibt sich aus der Summe der jeweiligen Rechte, wobei Leserecht 4 zählt, Schreibrecht 2 und Ausführrecht 1. Hier sind ein paar Beispiele für gängige Rechtezuordnungen in »`ls -l`«- und oktaler Darstellung:

```
rw-r--r--  644
r--------  400
rwxr-xr-x  755
```

Mit der numerischen Rechtedarstellung können Sie nur alle Rechte auf einmal setzen – es gibt keine Möglichkeit, wie mit den »`+`«- und »`-`«-Operatoren der symbolischen Darstellung einzelne Rechte zu setzen oder zu entfernen und die anderen dabei unbehelligt zu lassen. Das Kommando

```
$ chmod 644 datei
```

entspricht also der symbolischen Form

```
$ chmod u=rw,go=r datei
```

14.2.3 Dateieigentümer und Gruppe setzen – `chown` und `chgrp`

Das Kommando `chown` erlaubt das Setzen des Datei- oder Verzeichniseigentümers und der Gruppe. Dem Befehl werden die Benutzerkennung des Besitzers und/oder die gewünschte Gruppenkennung und der Dateiname bzw. Verzeichnisname, dessen Eigentümer geändert werden soll, übergeben. Der Aufruf sieht so aus:

```
chown ⟨Benutzername⟩[:][⟨Gruppenname⟩] ⟨Name⟩ …
chown :⟨Gruppenname⟩ ⟨Name⟩ …
```

Werden Benutzername und Gruppenkennung angegeben, werden beide gesetzt; wird nur ein Benutzername angegeben, bleibt die Gruppe so, wie sie war; wird ein Benutzername mit Doppelpunkt angegeben, dann wird die Datei der primären Gruppe des Benutzers zugeordnet; wird nur ein Gruppenname angegeben (mit Doppelpunkt), dann bleibt der Eigentümer so, wie er war. Zum Beispiel:

```
# chown hugo:entw brief.txt
# chown www-data bla.html                    Neuer Benutzer www-data
# chown :entw /home/entwicklung              Neue Gruppe entw
```

`chown` unterstützt auch eine veraltete Syntax, bei der anstatt des Doppelpunkts ein einfacher Punkt verwendet wird.

Um Dateien an andere Benutzer oder beliebige Gruppen zu »verschenken«, müssen Sie Systemverwalter sein. Der Hauptgrund dafür ist, dass normale Benutzer sonst einander ärgern können, wenn das System »Kontingentierung« (Quotas) verwendet, also jeder Benutzer nur über eine bestimmte Menge Plattenplatz verfügen darf.

Mit dem Kommando `chgrp` können Sie die Gruppe einer Datei ändern, und zwar auch als normaler Benutzer – solange Sie Eigentümer der Datei sowie Mitglied der *neuen* Gruppe sind:

```
chgrp ⟨Gruppenname⟩ ⟨Name⟩ …
```

Änderungen des Dateieigentümers oder der Dateigruppe ändern die Zugriffsrechte für die verschiedenen Kategorien nicht.

Auch `chown` und `chgrp` unterstützen die Option `-R` zur rekursiven Anwendung auf eine ganze Verzeichnishierarchie.

Selbstverständlich können Sie auch mit den meisten Dateibrowsern (wie Konqueror oder Nautilus) Rechte, Gruppe und Eigentümer einer Datei ändern.

Übungen

14.1 [!2] Legen Sie eine neue Datei an. Welcher Gruppe ist diese Datei zugeordnet? Verwenden Sie `chgrp`, um die Datei einer Ihrer anderen Gruppen zuzuordnen. Was passiert, wenn Sie die Datei einer Gruppe zuordnen wollen, in der Sie nicht Mitglied sind?

14.2 [4] Vergleichen Sie die Mechanismen, die verschiedene Dateibrowser (zum Beispiel Konqueror, Nautilus, …) zum Setzen von Dateirechten, Eigentümer, Gruppe, … anbieten. Gibt es nennenswerte Unterschiede?

14.3 Eigentum an Prozessen

Linux betrachtet nicht nur die mehr oder weniger festen Daten auf einem dauerhaften Speichermedium als Objekte, die einem Eigentümer zugeordnet werden. Auch die Prozesse im System haben einen Eigentümer.

Viele Kommandos, die Sie über die Tastatur eingeben, erzeugen einen Prozess im Arbeitsspeicher des Rechners. Im normalen Betrieb befinden sich immer mehrere Prozesse gleichzeitig im Speicher, die vom Betriebssystem streng voneinander abgegrenzt werden. Die einzelnen Prozesse werden mit allen ihren Daten ei-

Prozesse haben auch Eigentümer

nem Benutzer als Eigentümer zugeordnet. Dies ist in der Regel der Benutzer, der den Prozess gestartet hat – auch hier haben Prozesse, die mit Administratorrechten gestartet wurden, die Möglichkeit, ihre Identität zu ändern, und der SUID-Mechanismus (Abschnitt 14.4) kann hier ebenfalls eingreifen.

Die Eigentümer der Prozesse werden vom Programm `ps` angezeigt, wenn es mit der Option `-u` aufgerufen wird.

```
# ps -u
USER   PID %CPU  %MEM SIZE      RSS TTY STAT  START  TIME COMMAND
bin     89  0.0   1.0  788      328   ?    S  13:27  0:00 rpc.portmap
test1  190  0.0   2.0 1100       28   3    S  13:27  0:00 bash
test1  613  0.0   1.3  968       24   3    S  15:05  0:00 vi XF86.tex
nobody 167  0.0   1.4  932       44   ?    S  13:27  0:00 httpd
root     1  0.0   1.0  776       16   ?    S  13:27  0:03 init [3]
root     2  0.0   0.0    0        0   ?   SW  13:27  0:00 (kflushd)
```

14.4 Besondere Zugriffsrechte für ausführbare Dateien

Beim Auflisten der Dateien mit dem Befehl »`ls -l`« bekommen Sie bei manchen Dateien neben den bekannten Zugriffsrechten `rwx` abweichende Anzeigen wie

```
-rwsr-xr-x   1 root shadow  32916 Dec 11 20:47 /usr/bin/passwd
```

Was soll das? Hierzu müssen wir etwas weiter ausholen:

Angenommen, das Programm `passwd` sei mit folgenden Zugriffsrechten versehen:

```
-rwxr-xr-x   1 root shadow  32916 Dec 11 20:47 /usr/bin/passwd
```

Ein normaler (unprivilegierter) Benutzer, sagen wir mal `hugo`, möchte nun sein Kennwort ändern und ruft das Kommando `passwd` auf. Als nächstes erhält er die Meldung "*permission denied*". Was ist die Ursache? Der `passwd`-Prozess (der mit den Rechten von `hugo` läuft) versucht die Datei `/etc/shadow` zum Schreiben zu öffnen und scheitert natürlich, da nur `root` die erforderliche Schreibberechtigung besitzt – dies darf auch nicht anders sein, sonst könnte jeder die Kennwörter beliebig manipulieren, etwa um das `root`-Kennwort zu ändern.

SUID-Bit

Mit Hilfe des **Set-UID-Bits** (oft kurz »SUID-Bit« genannt) kann dafür gesorgt werden, dass ein Programm nicht mit den Rechten des Aufrufers, sondern des Dateieigentümers – hier `root` – ausgeführt wird. Im Fall von `passwd` hat der *Prozess*, der `passwd` ausführt, also Schreibrecht auf die Datei `/etc/shadow`, obwohl der aufrufende Benutzer als Nicht-Systemadministrator dieses Schreibrecht sonst nicht hat. Es liegt in der Verantwortung des Programmierers von `passwd`, dafür zu sorgen, dass mit diesem Recht kein Schindluder getrieben wird, etwa indem Programmierfehler ausgenutzt werden, um beliebige Dateien außer `/etc/shadow` zu manipulieren oder andere Einträge in `/etc/shadow` außer dem Kennwortfeld des aufrufenden Benutzers zu verändern. Unter Linux funktioniert der Set-UID-Mechanismus übrigens nur für Maschinencode-Programme, nicht für Shell- oder andere Interpreter-Skripte.

Die Bell Labs hatten eine Weile lang ein Patent auf den – von Dennis Ritchie erfundenen – SUID-Mechanismus [SUID]. AT&T hatte Unix zuerst nur unter der Voraussetzung verteilt, dass nach der Erteilung des Patents Lizenzgebühren erhoben werden würden; wegen der logistischen Schwierigkeit, Jahre später von Hunderten von Unix-Installationen rückwirkend kleine Geldbeträge einzutreiben, entschloss man sich jedoch, das Patent der Allgemeinheit zur Verfügung zu stellen.

Analog zum Set-UID-Bit gibt es auch ein SGID-Bit, mit dem ein Prozess statt mit der Gruppenzugehörigkeit des Aufrufers mit der Gruppenzugehörigkeit der Programmdatei und den damit verbundenen Rechten (typischerweise zum Zugriff auf andere Dateien, die dieser Gruppe zugeordnet sind) ausgeführt wird. SGID-Bit

Die SUID- und SGID-Modi werden wie alle anderen Modi einer Datei mit dem Systemprogramm `chmod` verändert, indem Sie symbolische Schlüssel wie `u+s` (setzt das SUID-Bit) oder `g-s` (löscht das SGID-Bit) angeben. Auch in oktalen Modi können Sie diese Bits setzen, indem Sie eine vierte Ziffer ganz links hinzufügen: Das SUID-Bit hat den Wert 4, das SGID-Bit den Wert 2 – so können Sie einer Datei den Zugriffsmodus 4755 geben, um sie für alle Benutzer lesbar und ausführbar zu machen (der Eigentümer darf auch schreiben) und das SUID-Bit zu setzen. `chmod`-Syntax

Sie erkennen Programme, die mit den Rechten des Eigentümers oder der Gruppe der Programmdatei arbeiten, in der Ausgabe von »`ls -l`« durch die symbolischen Abkürzungen »`s`« anstelle von »`x`« für normal ausführbare Dateien. `ls`-Ausgabe

14.5 Besondere Zugriffsrechte für Verzeichnisse

Es gibt eine weitere Ausnahme von der Zuordnung des Eigentums an Dateien nach dem »Verursacherprinzip«: Der Eigentümer eines Verzeichnisses kann bestimmen, dass die in diesem Verzeichnis erzeugten Dateien der gleichen Benutzergruppe gehören wie das Verzeichnis selbst. Das geschieht, indem das SGID-Bit des Verzeichnisses gesetzt wird. (Da Verzeichnisse nicht ausgeführt werden können, ist das SGID-Bit für solche Sachen frei.) SGID für Verzeichnisse

Die Zugriffsrechte auf ein Verzeichnis werden durch das SGID-Bit nicht verändert. Um eine Datei in einem solchen Verzeichnis anzulegen, muss ein Benutzer das Schreibrecht in der für ihn zutreffenden Kategorie (Eigentümer, Gruppe, andere Benutzer) haben. Wenn ein Benutzer zum Beispiel weder der Eigentümer noch Mitglied der Benutzergruppe eines SGID-Verzeichnisses ist, muss das Verzeichnis für alle Benutzer beschreibbar sein, damit er neue Dateien hineinschreiben kann. Die in dem SGID-Verzeichnis erzeugte Datei gehört dann der Gruppe des Verzeichnisses, auch wenn der Benutzer selbst dieser Gruppe nicht angehört.

Der typische Anwendungsfall für das SGID-Bit auf einem Verzeichnis ist ein Verzeichnis, das einer »Projektgruppe« als Datenablage dient. (Nur) Die Mitglieder der Projektgruppe sollen alle Dateien im Verzeichnis lesen und schreiben und auch neue Dateien anlegen können. Das heißt, Sie müssen alle Benutzer, die an dem Projekt mitarbeiten, in die Projektgruppe tun (sekundäre Gruppe reicht):

```
# groupadd projekt                                  Neue Gruppe anlegen
# usermod -a -G projekt hugo                        hugo in die Gruppe
# usermod -a -G projekt susi                        susi auch
```

Jetzt können Sie das Verzeichnis anlegen und der neuen Gruppe zuordnen. Eigentümer und Gruppe bekommen alle Rechte, der Rest der Welt keine; außerdem setzen Sie noch das SGID-Bit:

```
# cd /home/projekt
# chgrp projekt /home/projekt
# chmod u=rwx,g=srwx /home/projekt
```

Wenn `hugo` jetzt eine Datei in `/home/projekt` anlegt, sollte diese der Gruppe `projekt` zugeordnet sein:

```
$ id
uid=1000(hugo) gid=1000(hugo) groups=101(projekt),1000(hugo)
$ touch /tmp/hugo.txt                          Test: gewöhnliches Verzeichnis
```

```
$ ls -l /tmp/hugo.txt
-rw-r--r-- 1 hugo hugo 0 Jan  6 17:23 /tmp/hugo.txt
$ touch /home/projekt/hugo.txt                          Projektverzeichnis
$ ls -l /home/projekt/hugo.txt
-rw-r--r-- 1 hugo projekt 0 Jan  6 17:24 /home/projekt/hugo.txt
```

Das ganze hat nur einen Schönheitsfehler, den Sie erkennen, wenn Sie sich die letzte Zeile des Beispiels anschauen: Die Datei hat zwar die richtige Gruppenzugehörigkeit, aber andere Mitglieder der Gruppe `projekt` dürfen sie trotzdem nur lesen. Wenn Sie möchten, dass alle Mitglieder von `projekt` schreiben dürfen, müssen Sie entweder nachträglich `chmod` anwenden (lästig) oder die *umask* so setzen, dass das Gruppenschreibrecht erhalten bleibt (siehe Übung 14.4).

Der SGID-Modus verändert nur das Verhalten des Betriebssystems beim Erzeugen neuer Dateien. Der Umgang mit bereits existierenden Dateien ist in diesen Verzeichnissen völlig normal. Das bedeutet beispielsweise, dass eine Datei, die außerhalb des SGID-Verzeichnisses erzeugt wurde, beim Verschieben dorthin ihre ursprüngliche Gruppe behält (wohingegen sie beim Kopieren die Gruppe des Verzeichnisses bekommen würde).

Auch das Programm `chgrp` arbeitet in SGID-Verzeichnissen völlig normal: der Eigentümer einer Datei kann sie jeder Gruppe zueignen, der er selbst angehört. Gehört der Eigentümer nicht zu der Gruppe des Verzeichnisses, kann er die Datei mit `chgrp` nicht dieser Gruppe übergeben – dazu muss er sie in dem Verzeichnis neu erzeugen.

Es ist möglich, bei einem Verzeichnis das SUID-Bit zu setzen – diese Einstellung hat aber keine Wirkung.

Linux unterstützt noch einen weiteren Spezialmodus für Verzeichnisse, bei dem das Löschen oder Umbenennen von darin enthaltenen Dateien nur dem Besitzer der jeweiligen Datei erlaubt ist:

```
drwxrwxrwt   7 root   root   1024 Apr  7 10:07 /tmp
```

sticky bit

Mit diesem `t`-Modus, dem ***sticky bit***, kann einem Problem begegnet werden, das bei der gemeinsamen Verwendung öffentlicher Verzeichnisse entstehen kann: Das Schreibrecht für das Verzeichnis erlaubt auch das Löschen fremder Dateien, unabhängig von deren Zugriffsmodus und Besitzer! Beispielsweise sind die `/tmp`-Verzeichnisse öffentlicher Raum, in dem von vielen Programmen temporäre Dateien angelegt werden. Um darin Dateien anlegen zu können, haben alle Benutzer für diese Verzeichnisse Schreibrecht. Damit hat jeder Benutzer auch das Recht, Dateien in diesem Verzeichnis zu löschen.

Normalerweise betrachtet das Betriebssystem beim Löschen oder Umbenennen einer Datei die Zugriffsrechte auf die Datei selbst nicht weiter. Wenn auf einem Verzeichnis das *sticky bit* gesetzt wird, kann eine Datei in diesem Verzeichnis anschließend nur von ihrem Eigentümer, dem Eigentümer des Verzeichnisses oder `root` gelöscht werden. Das *sticky bit* kann über die symbolische Angabe `+t` bzw. `-t` gesetzt oder gelöscht werden, oktal hat es in derselben Ziffer wie SUID und SGID den Wert 1.

Das *sticky bit* bekommt seinen Namen von einer weiteren Bedeutung, die es früher in anderen Unix-Systemen hatte: Seinerzeit wurden Programme vor dem Start komplett in den Swap-Speicher kopiert und nach dem Programmlauf wieder daraus entfernt. Programmdateien, auf denen das *sticky bit* gesetzt war, wurden dagegen auch noch nach dem Programmende im Swap-Speicher liegengelassen. Dies beschleunigte weitere Startvorgänge für dasselbe Programm, weil der anfängliche Kopiervorgang entfiel. Linux verwendet wie die meisten heutigen Unix-Systeme *demand paging*, holt also nur die

wirklich benötigten Teile des Programmcodes direkt aus der Programmdatei und kopiert gar nichts in den Swap-Speicher; das *sticky bit* hatte unter Linux niemals seine ursprüngliche Sonderbedeutung.

Übungen

14.3 [2] Was bedeutet das spezielle Zugriffsrecht »s«? Wo finden Sie es? Können Sie dieses Recht auf eine selbst erstellte Datei setzen?

14.4 [!1] Mit welchem `umask`-Aufruf können Sie eine *umask* einstellen, die im Projektverzeichnis-Beispiel allen Mitgliedern der Gruppe `projekt` das Lesen *und* Schreiben von Dateien im Projektverzeichnis erlaubt?

14.5 [2] Was bedeutet das spezielle Zugriffsrecht »t«? Wo finden Sie es?

14.6 [4] (Für Programmierer.) Schreiben Sie ein C-Programm, das ein geeignetes Kommando (etwa `id`) aufruft. Machen Sie dieses Programm SUID-`root` bzw. SGID-`root` und beobachten Sie, was passiert.

14.7 [3] Wenn Sie sie für ein paar Minuten mit einer `root`-Shell alleine lassen, könnten findige Benutzer versuchen, irgendwo im System eine Shell zu hinterlegen, die sie SUID `root` gemacht haben, um auf diese Weise nach Bedarf Administratorrechte zu bekommen. Funktioniert das mit der Bash? Mit anderen Shells?

Kommandos in diesem Kapitel

`chgrp` Setzt Gruppenzugehörigkeit von Dateien und Verzeichnissen `chgrp`(1) 209

`chmod` Setzt Rechte für Dateien und Verzeichnisse `chmod`(1) 207

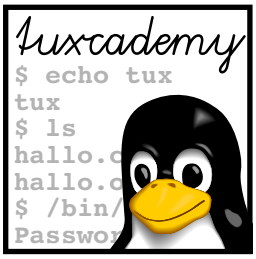
`chown` Setzt Eigentümer und Gruppenzugehörigkeit für Dateien und Verzeichnisse `chown`(1) 208

Zusammenfassung

- Linux unterscheidet für Dateien das Lese-, Schreib- und Ausführungsrecht, wobei diese Rechte getrennt für den Eigentümer der Datei, die Mitglieder der der Datei zugeordneten Gruppe und den »Rest der Welt« angegeben werden können.
- Die Gesamtheit der Rechte für eine Datei heißt auch Zugriffsmodus.
- Jede Datei (und jedes Verzeichnis) hat einen Eigentümer und eine Gruppe. Zugriffsrechte – Lese-, Schreib- und Ausführungsrecht – werden für diese beiden Kategorien und den »Rest der Welt« getrennt vergeben. Nur der Eigentümer darf die Zugriffsrechte setzen.
- Für den Systemverwalter (`root`) gelten die Zugriffsrechte nicht – er darf jede Datei lesen oder schreiben.
- Dateirechte werden mit dem Kommando `chmod` manipuliert.
- Mit `chown` kann der Systemverwalter die Eigentümer- und Gruppenzuordnung beliebiger Dateien ändern.
- Normale Benutzer können mit `chgrp` ihre Dateien einer anderen Gruppe zuordnen.
- SUID- und SGID-Bit erlauben es, Programme mit den Rechten des Dateieigentümers bzw. der der Datei zugeordneten Gruppe anstatt den Rechten des Aufrufers auszuführen.
- Das SGID-Bit auf einem Verzeichnis bewirkt, dass neue Dateien in diesem Verzeichnis der Gruppe des Verzeichnisses zugeordnet werden (und nicht der primären Gruppe des anlegenden Benutzers).
- Das *sticky bit* auf einem Verzeichnis erlaubt das Löschen von Dateien nur dem Eigentümer (und dem Systemadministrator).

Literaturverzeichnis

SUID Dennis M. Ritchie. »Protection of data file contents«. US-Patent 4,135,240. Beantragt am 9.7.1973, erteilt am 16.7.1979.



15

Linux im Netz

Inhalt

Lernziele

- Einen Überblick über Netzwerk-Konzepte haben
- Anforderungen dafür verstehen, einen Linux-Rechner in ein lokales Netz zu integrieren
- Wichtige Kommandos zur Fehlersuche kennen
- Wichtige Netzwerkdienste kennen

Vorkenntnisse

- Umgang mit Dateien (Kapitel 6) und mit einem Texteditor
- Kenntnisse über die Linux-Dateisystemstruktur (Kapitel 10)
- Vorkenntnisse im TCP/IP-Bereich und Kenntnisse im Umgang mit Netzwerkdiensten sind nützlich

15.1 Netzwerk-Grundlagen

15.1.1 Einführung und Protokolle

Im 21. Jahrhundert machen Computer erst richtig Spaß, wenn sie ans Internet angeschlossen sind – in Firmen meist über ein unternehmensweites »lokales Netz«, daheim über einen DSL-Router oder unterwegs über eine Funkverbindung (WLAN oder Mobilfunk). Es gibt diverse Methoden, ins Netz zu kommen – aber unabhängig davon, wie Ihr Rechner im Netz ist, funktioniert das Internet selbst eigentlich immer gleich.

TCP/IP

Grundlage des Internets ist eine »Protokollfamilie« namens »TCP/IP«. Ein Protokoll ist eine Verabredung darüber, wie sich Rechner miteinander unterhalten, und kann alles abdecken von bestimmten elektrischen, optischen oder Funksignalen bis hin zu Anfragen und Antworten zum Beispiel an einen Web-Server (aber nicht alles gleichzeitig). Man kann grob zwischen drei Arten von Protokollen unterscheiden:

Übertragungsprotokolle regeln den Datenverkehr (etwas salopp gesagt) auf der Ebene von Netzwerkkarten und Kabeln. Dazu gehören zum Beispiel Ethernet (fürs LAN) oder WLAN-Protokolle wie IEEE 802.11.

Kommunikationsprotokolle regeln die Kommunikation zwischen Rechnern in verschiedenen Netzen. Wenn Sie von Deutschland aus auf eine Webseite in Australien zugreifen, sorgen die Kommunikationsprotokolle IP und TCP dafür, dass die Datenbytes, die Sie hier losschicken, *down under* ankommen und umgekehrt.

Anwendungsprotokolle sorgen dafür, dass der Empfänger Ihrer Datenbytes in Australien mit den Datenbytes tatsächlich etwas anfangen kann. Das Web-Anwendungsprotokoll HTTP zum Beispiel erlaubt es Ihnen, ein Bild von einem Koalabär abzurufen; der Server in Australien interpretiert Ihre Anfrage und schickt Ihnen das Koala-Bild (und nicht das vom Känguru), und Ihr Web-Browser hier in Deutschland kann erkennen, dass Sie tatsächlich ein Bild bekommen haben und keine (textuelle) Fehlermeldung.

Diese schichtweise Organisation hat den großen Vorteil, dass jede Schicht nur mit der Schicht direkt darüber und der Schicht direkt darunter verkehren muss. Das Anwendungsprotokoll HTTP muss nicht wissen, wie Bytes von Deutschland nach Australien kommen (zum Glück!), denn diese Aufgabe delegiert es an das Kommunikationsprotokoll. Das Kommunikationsprotokoll wiederum überläßt die tatsächliche Übertragung der Bytes dem Übertragungsprotokoll.

Klassischerweise wird an dieser Stelle auf das »ISO/OSI-Referenzmodell« verwiesen, das zur Organisation eines Rechnernetzes nicht weniger als sieben (!) Schichten vorsieht. Wir ersparen Ihnen das hier aber im Interesse der Kürze.

Die Anwendungsprotokolle in TCP/IP – außer HTTP etwa SMTP (für E-Mail), SSH (für interaktive Sitzungen auf entfernten Rechnern), DNS (für die Auflösung von Rechnernamen) oder SIP (für Internet-Telefonie), unter vielen anderen – verwenden normalerweise eines von zwei Kommunikationsprotokollen, UDP oder TCP. Während UDP einen verbindungslosen, unzuverlässigen Dienst realisiert (auf Deutsch: Daten können bei der Übertragung verlorengehen oder in einer anderen Reihenfolge ankommen, als sie abgeschickt wurden), stellt TCP einen verbindungsorientierten, zuverlässigen Dienst zur Verfügung (das heißt, die Daten kommen am anderen Ende vollständig und in der richtigen Reihenfolge an).

Welches Kommunikationsprotokoll günstiger ist, hängt von der Anwendung ab. Im Web möchten Sie normalerweise nicht, dass bei der Übertragung Daten flöten gehen (ein Stück Text, Bild oder heruntergeladener

Programmcode könnte fehlen, mit ärgerlichen bis katastrophalen Ergebnissen), also ist TCP die richtige Wahl. Bei Fernsehen oder Telefonie übers Internet dagegen können Sie wahrscheinlich eher mit kurzen Aussetzern leben (ein pixeliges Bild oder ein kurzes Rauschen in der Leitung), als dass alles für eine halbe Minute komplett zum Stehen kommt, während das System ein verlorengegangenes Datenpaket neu anfordert. An dieser Stelle ist UDP besser. UDP eignet sich auch besser für Dienste wie DNS, wo kurze Anfragen sehr schnell kurze Antworten liefern sollen.

Das dritte Kommunikationsprotokoll in TCP/IP, ICMP, dient zur Steuerung und Fehlersuche und wird von Benutzerprogrammen normalerweise nicht direkt verwendet.

15.1.2 Adressierung und Routing

Damit jeder Rechner auf dem Internet gezielt angesprochen werden kann – Ihr Rechner schickt einem bestimmten Server eine Anfrage, und die Antwort auf die Anfrage muss ja auch irgendwie wieder genau zu Ihrem Rechner zurückfinden –, bekommt er eine eindeutige Adresse, die »IP-Adresse«. Im aktuell (noch) weiter verbreiteten Standard, »IPv4«, ist das eine Kombination aus vier »Oktetten«, also Zahlen von 0 bis 255, zum Beispiel `192.168.178.10`.

Ihre IP-Adresse dürfen Sie sich nicht einfach aussuchen, sondern Sie bekommen sie zugeordnet. In einer Firma erledigt das der System- oder Netzwerkadministrator und für Ihren Internet-Anschluss daheim Ihr Internet-Anbieter.

In der Firma kriegen Sie mit einiger Wahrscheinlichkeit immer dieselbe IP-Adresse. Ihr Internet-Anbieter wird Ihnen dagegen eine IP-Adresse immer nur für eine bestimmte Zeit »leihen«; beim nächsten Mal bekommen Sie dann eine andere. Das soll dem Anbieter einerseits ermöglichen, weniger Adressen zu benutzen, als er Kunden hat (denn nicht jeder Kunde ist permanent online), und Sie andererseits davon abhalten, Serverdienste anzubieten, für die eine ständig wechselnde IP-Adresse eine Behinderung darstellt.

Die IP-Adressen selbst fallen nicht einfach so vom Himmel, sondern werden – soweit möglich – mit System vergeben, um die Weiterleitung von Daten zwischen verschiedenen Netzen, das sogenannte »Routing«, zu vereinfachen. Dazu gleich mehr.

Loopback-Adresse

Rechner haben normalerweise mehr als eine IP-Adresse. Die »Loopback-Adresse« `127.0.0.1` steht auf jedem Rechner für den Rechner selbst und ist auch von außen nicht zugänglich. Ein Dienst, der auf der Adresse `127.0.0.1` angeboten wird, kann darum nur von Clients auf dem Rechner selbst angesprochen werden.

Das klingt absurd und nutzlos, hat aber durchaus Sinn: Zum Beispiel könnten Sie an der Entwicklung einer Webseite arbeiten, die Mail an Benutzer verschicken können muss (etwa mit Aktivierungs-Links für Benutzerkonten). Um das auf Ihrem Entwicklungsrechner zu testen, können Sie einen lokalen Mailserver installieren, der nur auf der Loopback-Adresse Mail annimmt – für Ihr Projekt reicht das auf jeden Fall aus, aber Sie laufen keine Gefahr, von irgendwo auf dem Internet mit Spam überschüttet zu werden.

Bei Rechnern mit Ethernet oder WLAN hat die betreffende Schnittstelle natürlich auch eine IP-Adresse – jedenfalls wenn der Rechner aktuell im Netz ist. Außerdem spricht nichts dagegen, den Netzwerkschnittstellen für Testzwecke oder besondere Konfigurationen weitere Adressen zuzuordnen.

Die Herausforderung besteht darin, dafür zu sorgen, dass irgendein beliebiger Rechner X auf dem Internet mit irgendeinem beliebigen anderen Rechner Y kommunizieren kann, solange er nur die IP-Adresse von Y kennt. (Wenn X in Deutschland steht und Y in Australien, ist es nicht 100% offensichtlich, wie die

Bytes von X zu Y gelangen sollen.) Das Zauberwort, das dies möglich macht, ist »Routing«. Und das funktioniert ungefähr so:

Routing

- Ihr Rechner X kann die IP-Adresse von Rechner Y herausfinden (dafür ist das DNS da). Vielleicht ist sie `10.11.12.13`.
- Rechner X kann feststellen, dass die IP-Adresse `10.11.12.13` nicht im selben lokalen Netz ist wie seine eigene (kein Wunder, Rechner Y ist ja in Australien). Das heißt, Rechner X kann keine Daten direkt an Rechner Y schicken (große Überraschung).

Default-Route

- Die Netzwerk-Konfiguration von Rechner X enthält eine »Default-Route«, anders gesagt ein Rezept dafür, wie Rechner X mit Daten umgehen soll, die er nicht direkt zustellen kann. Diese Default-Route kann zum Beispiel aussehen wie »Schicke alles, was Du nicht direkt zustellen kannst, an Rechner Z«. Den Rechner Z nennt man auch »Default-Gateway«.
- Auch Rechner Z – vielleicht Ihr DSL-Router – kann die Daten nicht direkt an Rechner Y zustellen. Dafür weiß er aber auch, was er mit Daten machen soll, die nicht für die direkt an ihn angeschlossenen Rechner wie X gedacht sind, nämlich weiterschicken an Ihren Internet-Anbieter.
- Dieses Spiel geht möglicherweise noch über ein paar Ebenen so weiter, bis Ihre Daten auf einem Rechner landen, der weiß, dass an `10.11.`x`.`y adressierte Daten an den australischen Internetanbieter (nennen wir ihn mal »Billabong-Net«) geschickt werden sollen, also werden sie dorthin geleitet.
- Billabong-Net weiß (hoffentlich), wie eingehende Pakete an `10.11.12.13` an ihr endgültiges Ziel geleitet werden müssen. Der tatsächliche Rechner Y steht zwar vielleicht bei einem Kunden von Billabong-Net, oder bei einem Kunden dieses Kunden, aber wenn alles richtig konfiguriert ist, passt das schon.
- Allfällige Antworten von Y an X nehmen prinzipiell den umgekehrten Weg.

Die wichtige Beobachtung an dieser Stelle ist, dass der tatsächliche Weg der Daten von Rechner X zu Rechner Y sich erst während der Zustellung ergibt. Rechner X muss keine genaue Streckenführung vorgeben, sondern verläßt sich darauf, dass jede der Stationen unterwegs das Richtige tut. Umgekehrt kommt jeder Rechner mit »lokalem« Wissen aus und muss nicht für das komplette Internet wissen, was wo ist – was ein Ding der Unmöglichkeit wäre.

Eine der Eigenschaften von IP als Kommunikationsprotokoll ist, dass das Routing sich grundsätzlich von Paket zu Paket ändern kann (auch wenn das in der Regel nicht passiert). Das macht es dem Internet möglich, flexibel auf Engpässe oder Ausfälle von Verbindungen zu reagieren.

Im Prinzip ist das so ähnlich wie bei der »gelben Post«. Sie müssen ja auch nicht wissen, welchen genauen Weg Ihre Geburtstagskarte für Tante Susi in Sydney nimmt, sondern Sie stecken nur den korrekt freigemachten Umschlag in Ihren freundlichen Nachbarschafts-Briefkasten.

Unter dem Strich läuft das Ganze darauf hinaus, dass Ihr Linux-Rechner drei Dinge wissen muss: seine eigene IP-Adresse, welche Adressen er direkt erreichen kann, und ein Default-Gateway für den Rest. Welche Adressen Ihr Rechner direkt erreichen kann, wird durch die Adresse Ihres lokalen Netzes zusammen mit einer »Subnetzmaske« beschrieben, die heutzutage zumeist als Zahl angegeben wird.

Subnetzmaske

Stellen Sie sich vor, Ihr Rechner hat die IP-Adresse `192.168.178.111` im lokalen Netz `192.168.178.0/24`. Dabei ist die 24 die Subnetzmaske – sie gibt an, dass die ersten 24 Bit Ihrer Adresse (also die ersten drei Oktette, namentlich `192.168.178`) das Netz selbst adressieren. Das letzte Oktett (oder die letzten 8 Bit,

die zur »32« fehlen) steht damit für lokale Adressen zur Verfügung. Das heißt, alle Rechner mit Adressen von `192.168.178.0` bis `192.168.178.255` sind – falls sie überhaupt existieren – direkt lokal erreichbar; für alle anderen IP-Adressen muss der Default-Gateway (der eine Adresse im lokalen Netz haben muss) herangezogen werden.

In Wirklichkeit wären in unserem Beispiel die Adressen `192.168.178.0` und `192.168.178.255` nicht für Rechner zu benutzen, weil sie eine Spezialbedeutung haben, aber wir sagen das nur der Vollständigkeit halber.

Diese wichtigen Netzwerkparameter – IP-Adresse, Subnetz(maske) und Default-Gateway – können Sie grundsätzlich mit der Hand in die Netzwerkkonfiguration Ihres Rechners eintragen. (Die Details dafür sind distributionsspezifisch.) Es ist allerdings extrem wahrscheinlich, dass in Ihrem Netz ein »DHCP-Server« zur Verfügung steht, der Ihrem Rechner die Netzwerkparameter bei Bedarf zukommen läßt, ohne dass Sie sich selbst darum kümmern müssen.

Wenn Sie also einen Linux-Rechner als Client ins Netz bringen wollen, sollte es normalerweise reichen, ihn einzuschalten und entweder das Ethernet-Kabel in die dafür vorgesehene Buchse zu stecken oder im dafür vorgesehenen Menü ein WLAN auszuwählen und gegebenenfalls das Kennwort dafür einzutippen. Sollte es damit Probleme geben, dann schreien Sie laut und nachdrücklich nach Ihrem System- oder Netzadministrator.

15.1.3 Namen und DNS

IP-Adressen sind gut, schön und wichtig, aber etwas unbequem in der Handhabung. Es wäre schließlich sehr ärgerlich, wenn Sie die Adresse `213.157.7.75` eingeben (geschweige denn sich merken) müßten, um auf den Web-Server zu kommen, wo die aktuelle Version dieser Schulungsunterlage zur Verfügung steht. `shop.linupfront.de` ist da doch um Einiges eingängiger.

Damit Sie von den bequemen Namen zu den unbequemen IP-Adressen kommen (und umgekehrt), gibt es das *Domain Name System* oder DNS. Das DNS ist eine weltweit verteilte Datenbank für Rechnernamen, IP-Adressen und noch ein paar andere Sachen, und es ist über DNS-Server (oder auch »Nameserver«) zugänglich. Ihre Firma oder Ihr Internet-Anbieter sollte einen (oder, aus Redundanzgründen besser zwei) DNS-Server für Sie betreiben.

DNS-Server

Sie können das bei Bedarf auch selbst – natürlich auf Linux-Basis –, und spätestens, wenn Sie für Ihre Firma ein nicht ganz kleines lokales Netz, einen Web- oder einen Mailserver warten, ist das meistens auch eine gute Idee. Allerdings sind wir da schon deutlich im LPIC-2-Territorium.

Die Adresse des (oder der) DNS-Server kommt als viertes Element zu den »wichtigen Netzwerkparametern« aus dem letzten Abschnitt – IP-Adresse, Subnetz(maske), Default-Gateway – dazu, die jeder Linux-Rechner haben sollte.

Ähnlich wie bei den IP-Adressen und Routen muss auch kein DNS-Server den kompletten Überblick über alle existierenden Namen haben (dafür sind es inzwischen auch einfach zu viele). In Wirklichkeit gibt es eine Hierarchie:

- Die »Root-Level-Nameserver« kennen den Teil eines Namens, der ganz rechts steht – etwa `.de` oder `.com` oder `.tv` oder was es sonst noch so gibt –, und wissen, welche Nameserver sich mit dem *Inhalt* dieser Zonen auskennen.
- Die Nameserver für `.de` (nur mal so als Beispiel) kennen alle Namen der Form *irgendwas*`.de` und wissen, welche Nameserver über die Namen unterhalb *dieser* Namen Bescheid wissen.

- Die Nameserver für einen Namen wie *irgendwas*.de (die normalerweise bei der betreffenden Firma oder deren Internet-Anbieter stehen) wissen die IP-Adresse für einen Namen wie www.*irgendwas*.de und können sie bei Bedarf liefern.

Das heißt, um einen Namen wie shop.linupfront.de »aufzulösen«, also die dazugehörige IP-Adresse zu finden, fragt Ihr Rechner zunächst einen Root-Level-Nameserver nach Nameservern für de. Dann fragt er einen dieser Nameserver nach Nameservern für linupfront.de. Schließlich fragt er einen Nameserver für linupfront.de nach der Adresse von shop.linupfront.de.

Genaugenommen macht diese Arbeit nicht »Ihr Rechner«, sondern der DNS-Server, den Ihr Rechner benutzt. Aber das ändert nichts am Prinzip.

Natürlich ist das ein ziemlich aufwendiges Verfahren, und darum merkt Ihr System sich die Ergebnisse von Anfragen für eine Weile. Wenn Sie einmal herausgefunden haben, dass shop.linupfront.de der IP-Adresse 213.157.7.75 entspricht, wird davon ausgegangen, dass das auch eine Weile so bleibt, und der Auflösungsprozess erst nach Ablauf dieser Zeit wieder angestoßen.

Der Vorteil an der Sache ist, dass wir von der Linup Front GmbH frei über Namen »unterhalb von« linupfront.de verfügen und diese auch nach Belieben in unseren DNS-Server eintragen können. Andere Leute holen sie sich direkt von da. Es wäre viel mühsamer, einen neuen Namen aufwendig beim »Internet-Amt« beantragen zu müssen und darauf zu warten, dass er dort ins offizielle Verzeichnis eingetragen wird. (Denken Sie mal an Grundbuch-Änderungen und wie lange die dauern.)

15.1.4 IPv6

IP als Kommunikationsprotokoll gibt es schon seit gut 30 Jahren, und man hat festgestellt, dass einige Annahmen, von denen man seinerzeit ausgegangen ist, wohl doch etwas naiv waren. Zum Beispiel erlaubt IPv4 (wie die aktuelle Version des Protokolls heißt) prinzipiell 2^{32}, also gut 4 Milliarden Adressen. Aufgrund von Einschränkungen des Protokolls sowie verschiedenen Ungeschicklichkeiten bei der Vergabe stehen aber praktisch keine unbenutzten IPv4-Adressen mehr zur Verfügung – und in einem Zeitalter, wo (gefühlt) fast jeder ein internetfähiges Mobiltelefon mit sich herumträgt und noch mehr Leute eins haben wollen, ist das ein definitives Problem.

Es gibt Mittel und Wege, das Problem abzumildern – zum Beispiel bekommt längst nicht jedes Internet-Handy eine vom ganzen Internet aus sichtbare Adresse zugeordnet, sondern die Betreiber schotten ihre Netze vom eigentlichen Internet ab, so dass sie mehr Adressen vergeben können (Stichwort »Adressumsetzung« oder *network address translation*, NAT) –, aber die Methoden sind ziemlich unappetitlich und führen auch zu Problemen anderswo.

Schon seit geraumer Zeit steht IPv6, der designierte Nachfolger von IPv4[1], in den Startlöchern. IPv6 räumt mit einigen Problemen von IPv4 auf, aber die Internet-Anbieter tun sich im Moment noch etwas schwer damit, IPv6 flächendeckend zur Verfügung zu stellen. Linux kommt mit IPv6 aber hervorragend zurecht, und da Sie IPv4 und IPv6 durchaus parallel betreiben können, steht einer IPv6-basierten Infrastruktur in Ihrem Unternehmen (oder gar im häuslichen LAN – viele DSL-Router unterstützen inzwischen IPv6) im Prinzip nichts im Weg. Hier sind einige der wesentlichen Eigenschaften von IPv6:

Erweiterter Adressraum Statt 32 Bit breiten Adressen verwendet IPv6 128 Bit breite Adressen, in der Hoffnung, dass das für die vorhersehbare Zukunft

[1] IPv5 hat es nie wirklich gegeben.

reicht (die Chancen stehen recht gut). IPv6-Adressen werden notiert, indem man jeweils 2 Bytes hexadezimal (Basis 16) darstellt und den Doppelpunkt als Trennzeichen verwendet:

```
fe80:0000:0000:0000:025a:b6ff:fe9c:406a
```

In jedem Viererblock dürfen führende Nullen entfernt werden:

```
fe80:0:0:0:25a:b6ff:fe9c:406a
```

Außerdem darf maximal eine Folge von Null-Blocks durch »::« ersetzt werden:

```
fe80::25a:b6ff:fe9c:406a
```

Die Loopback-Adresse, also das moralische Äquivalent zu `127.0.0.1` in IPv4, ist `::1`.

Adresszuordnung Bei IPv4 bekommen Sie von Ihrem Internet-Anbieter normalerweise eine IP-Adresse oder höchstens ein paar zugeordnet (es sei denn, Sie sind eine wirklich große Firma oder ein Internet-Anbieter – und auch für die sind die Adressen inzwischen sehr knapp). Wenn Sie mehr Adressen für Ihre Rechner brauchen, müssen Sie tricksen. Bei IPv6 bekommen Sie statt dessen ein komplettes *Netz*, namentlich ein »Subnetz-Präfix«, das von den 128 möglichen Adressbits nur 48 oder 56 festschreibt. Sie selbst können dann in mehreren Subnetzen jeweils 2^{64} Adressen vergeben, und das ist wahrscheinlich mehr, als Sie verbrauchen können (das *gesamte Internet* hat gemäß IPv4 ja nur 2^{32} Adressen – einen Bruchteil davon – zur Verfügung).

Vereinfachte Konfiguration Während bei IPv4 ein Rechner eine IP-Adresse für das lokale Netz zugeordnet bekommen muss, damit er andere Rechner erreichen kann – notfalls mit Hilfe eines DHCP-Servers –, kann ein Rechner sich mit IPv6 selbst eine Adresse geben, die zumindest für die lokale Kommunikation mit Rechnern in der unmittelbaren Nachbarschaft funktioniert. Außerdem ist es mit IPv6 für einen Rechner auch ohne DHCP möglich, Router in der Umgebung zu finden, die Daten ins Internet weiterleiten können, was verschiedene Probleme mit DHCP unter IPv4 vermeidet. IPv6 verwendet übrigens keine »Defaultroute«.

Andere Verbesserungen im Detail Das Format von IP-Datenpaketen wurde geändert, um schnelleres Routing zu ermöglichen. Außerdem definiert IPv6 Verfahren, mit denen das Subnetz-Präfix eines Netzes viel leichter geändert werden kann als die Adresse eines Netzes mit IPv4 – auch das vor dem Hintergrund, das Routing zu vereinfachen. Ferner unterstützt IPv6 verschlüsselte Netzwerkstrukturen (IPsec) und Mobilität, bei der Rechner – denken Sie an Mobiltelefone – von einem Netz in ein anderes umziehen können, ohne dass ihre IP-Adressen sich ändern oder bestehende Verbindungen abgebrochen werden (»Mobile IPv6«).

Kompatibilität Die Einführung von IPv6 betrifft nur IP – Protokolle wie TCP und UDP oder die darauf aufsetzenden Anwendungsprotokolle bleiben unverändert. Außerdem ist es, wie erwähnt, möglich, IPv4 und IPv6 parallel zu betreiben.

Übungen

15.1 [!1] Welche Transportprotokolle (außer Ethernet und IEEE-802.11) kennen Sie? Welche Kommunikationsprotokolle (außer IP, TCP und UDP)? Welche Anwendungsprotokolle außer den im Text genannten?

15.2 [!1] Wieviele nutzbare IP-Adressen enthält das Netz `10.11.12.0/22`? (Ziehen Sie wegen deren Sonderbedeutung die erste und letzte Adresse ab.)

15.3 [2] Welche Voraussetzungen müssen Sie erfüllen, um einen Domainnamen in der `de`-Top-Level-Domain registrieren zu können?

15.2 Linux als Netzwerk-Client

15.2.1 Anforderungen

Was nötig ist, um einen Linux-Rechner als (IPv4-)Client in ein existierendes Netz zu integrieren (der einzige für *Linux Essentials* interessante Anwendungsfall), haben wir ja schon im vorigen Abschnitt umrissen:

- Eine IP-Adresse für den Rechner selbst
- Eine Netzwerkadresse und -maske für den Rechner (damit er entscheiden kann, welche IP-Adressen »lokal« sind)
- Die Adresse eines Default-Gateways im lokalen Netz
- Die Adresse(n) eines (oder besser zweier) DNS-Server

Im Idealfall bekommt Ihr Rechner diese Informationen automatisch über DHCP zugewiesen, etwa beim Hochfahren oder Einstöpseln des LAN-Kabels für einen stationären Rechner oder beim Einbuchen in ein drahtloses Netz für mobile Systeme. Sollte das nicht der Fall sein, dann müssen Sie die Netzwerkparameter selber setzen. Wie das genau geht, hängt von Ihrer Distribution ab:


Bei Debian GNU/Linux und den davon abgeleiteten Distributionen steht die Netzkonfiguration in der Datei `/etc/network/interfaces`. Das Format ist größtenteils selbsterklärend, und in `/usr/share/doc/ifupdown/examples/network-interfaces.gz` steht ein kommentiertes Beispiel. Die Dokumentation finden Sie in `interfaces`(5).


Bei den SUSE-Distributionen konfigurieren Sie die Netzwerkparameter am einfachsten über den YaST (in »Netzwerkgeräte/Netzwerkkarte«). Ansonsten finden Sie Konfigurationsdateien unter `/etc/sysconfig/network`.


Bei den Red-Hat- und von Red Hat abgeleiteten Distributionen stehen Konfigurationsdateien im Verzeichnis `/etc/sysconfig/network-scripts`.

`ifconfig` Für kurzzeitige Experimente können Sie auch das Kommando `ifconfig` verwenden. Etwas wie

```
# ifconfig eth0 192.168.178.111 netmask 255.255.255.0
```

verpasst der Netzwerkschnittstelle `eth0` (Ethernet) die IP-Adresse `192.168.178.111` im lokalen Netz `192.168.178.0/24` – die `255.255.255.0` ist eine umständliche Methode, die Netzmaske (24) hinzuschreiben. Das Default-Gateway (zum Beispiel `192.168.178.1`) konfigurieren Sie mit dem Kommando `route`:

Default-Gateway

```
# route add -net default gw 192.168.178.1
```

Diese Einstellung halten nur bis zum nächsten Neustart vor!

`/etc/resolv.conf` Die Adressen von DNS-Servern stehen für gewöhnlich in der Datei `/etc/resolv.conf`, die ungefähr so aussehen könnte:

```
# /etc/resolv.conf
search example.com
nameserver 192.168.178.1
nameserver 192.168.178.2
```

(das »search example.com« sucht, wenn Sie irgendwo einen Namen ohne Punkt – etwa www – angegeben haben, automatisch nach »www.example.com«).

Wenn Ihr System das Netz automatisch konfiguriert – etwa bei der Verbindung mit WLANs –, wird der Inhalt von /etc/resolv.conf mitunter gnadenlos überschrieben. Schauen Sie gegebenenfalls in der Dokumentation nach, wie und wo Sie in Ihrer Distribution Nameserver dauerhaft konfigurieren können.

15.2.2 Fehlersuche

Sollte die einfache Methode der Netzwerkkonfiguration (Einstöpseln bzw. Einbuchen) nicht funktionieren oder sollten anderweitige Probleme auftauchen – etwa quälend lange Wartezeiten beim Zugriff auf Web-Seiten oder unerklärliche Verbindungsabbrüche –, dann sollten Sie einen System- oder Netzadministrator zu Rate ziehen oder generell jemanden, der sich mit der Materie besser auskennt als Sie. (Jedenfalls bis Sie die LPIC-2-Prüfungen abgelegt haben; spätestens ab dann wird man *Sie* herbeiholen, wenn andere Leute in Schwierigkeiten sind.)

Auf der anderen Seite kommt es immer gut an, wenn Sie selbst die gängigsten Probleme ausgeschlossen oder den Fehler schon etwas eingegrenzt haben. Das spart Ihrem Administrator möglicherweise Arbeit oder sorgt zumindest dafür, dass Sie vor ihr oder ihm nicht aussehen wie ein DAU[2], sondern wie jemand, der für voll genommen werden sollte.

Im Rest dieses Abschnitts erklären wir Ihnen die wichtigsten Werkzeuge zur Fehlersuche und wie Sie sie benutzen können.

ifconfig Das Kommando ifconfig haben wir gerade zur Netzkonfiguration kennengelernt. Mit ifconfig können Sie aber auch die Konfiguration einer Netzwerkschnittstelle abfragen:

```
$ /sbin/ifconfig eth0
eth0      Link encap:Ethernet  Hardware Adresse 70:5a:b6:9c:40:6a
          inet Adresse:192.168.178.130  Bcast:192.168.178.255▷
                                                          ◁ Maske:255.255.255.0
          inet6-Adresse: 2002:4fee:57e5:0:725a:b6ff:fe9c:406a/64▷
                                                    ◁ Gültigkeitsbereich:Global
          inet6-Adresse: fe80::725a:b6ff:fe9c:406a/64▷
                                                 ◁ Gültigkeitsbereich:Verbindung
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metrik:1
          RX packets:83769 errors:0 dropped:0 overruns:0 frame:0
          TX packets:76525 errors:0 dropped:0 overruns:0 carrier:0
          Kollisionen:0 Sendewarteschlangenlänge:1000
          RX bytes:69767848 (66.5 MiB)  TX bytes:10245590 (9.7 MiB)
          Interrupt:20 Speicher:d7400000-d7420000
```

Interessant sind hier vor allem die verschiedenen Adressenangaben:

- In der ersten Zeile der Ausgabe steht die »Hardware«- oder »MAC-Adresse« der Schnittstelle. Sie wird vom Hersteller der (hier) Ethernet-Karte fest vergeben.
- In der zweiten Zeile steht die IP(v4)-Adresse, die der Schnittstelle zugeordnet ist. Ganz rechts steht die Netzmaske in der altertümlich-umständlichen Form.
- Die dritte und vierte Zeile geben verschiedene IPv6-Adressen an. Die vierte Zeile ist die nur lokal gültige Adresse, die der Rechner sich selbst gegeben hat, während die dritte ein Netzwerkpräfix enthält, mit dem der Rechner grundsätzlich aus dem Internet erreichbar wäre.

[2] »Dümmster anzunehmender User«


Wenn Sie genau hinschauen, können Sie in der zweiten Hälfte jeder der IPv6-Adressen die MAC-Adresse der Schnittstelle wiederfinden.

Das »UP« in der fünften Zeile signalisiert, dass die Schnittstelle aktiv ist.

Wenn Sie `ifconfig` ohne Parameter aufrufen, gibt es Informationen über alle aktiven Netzwerkschnittstellen auf dem Rechner aus. Mit dem Parameter `-a` nimmt es noch die gerade nicht aktiven dazu.

`ping` Mit dem Kommando `ping` können Sie auf der untersten Ebene (IP) prüfen, ob Ihr Rechner mit anderen Rechnern kommunizieren kann. `ping` benutzt das Steuerungsprotokoll ICMP, um einen anderen Rechner um ein »Lebenszeichen« zu bitten. Kommt dieses Lebenszeichen wieder bei Ihrem Rechner an, wissen Sie, dass (a) Ihr Rechner Daten an den anderen Rechner schicken kann, und (b) der andere Rechner Daten an *Ihren* Rechner schicken kann (das eine impliziert nicht notwendigerweise das andere).

Im einfachsten Fall rufen Sie `ping` mit dem Namen des Rechners auf, mit dem Sie kommunizieren wollen:

```
$ ping fritz.box
PING fritz.box (192.168.178.1) 56(84) bytes of data.
64 bytes from fritz.box (192.168.178.1): icmp_req=1 ttl=64 time=3.84 ms
64 bytes from fritz.box (192.168.178.1): icmp_req=2 ttl=64 time=5.09 ms
64 bytes from fritz.box (192.168.178.1): icmp_req=3 ttl=64 time=3.66 ms
64 bytes from fritz.box (192.168.178.1): icmp_req=4 ttl=64 time=3.69 ms
64 bytes from fritz.box (192.168.178.1): icmp_req=5 ttl=64 time=3.54 ms
                                                    Abbruch mit Strg+C …
--- fritz.box ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 3.543/3.967/5.095/0.575 ms
```

Hier ist alles in Ordnung: Alle fünf abgeschickten Pakete sind wieder zurückgekommen, die Reihenfolge stimmt, und die Laufzeiten sind im Rahmen für ein lokales Netz. Wenn Sie statt dessen eine Weile lang nichts sehen und dann etwas wie

```
From 192.168.178.130 icmp_seq=1 Destination Host Unreachable
From 192.168.178.130 icmp_seq=2 Destination Host Unreachable
From 192.168.178.130 icmp_seq=3 Destination Host Unreachable
```

erscheint, dann liegt etwas im Argen – der Zielrechner ist nicht erreichbar.

Wenn Sie keine Verbindung zu einem entfernten Rechner im Internet bekommen, kann das Problem natürlich grundsätzlich irgendwo zwischen Ihnen und dem entfernten Rechner liegen. Wenn Sie methodisch vorgehen wollen, dann benutzen Sie die folgende Taktik:

- Prüfen Sie mit `ping`, ob Sie die Loopback-Schnittstelle `127.0.0.1` erreichen können. Wenn das nicht klappt, dann ist mit *Ihrem* Rechner kapital etwas nicht in Ordnung.
- Prüfen Sie mit `ping`, ob Sie die Adresse der Netzwerkkarte (Ethernet, WLAN, …) erreichen können, über die Sie gerade mit dem Internet verbunden sind (oder zu sein glauben). Sie finden die Adresse bei Bedarf mit etwas wie

  ```
  $ /sbin/ifconfig eth0
  ```

  heraus. Auch das sollte eigentlich funktionieren – ansonsten haben Sie ein lokales Problem.
- Prüfen Sie mit `ping`, ob Sie die Adresse Ihres Default-Gateways erreichen können. (Wenn Sie die nicht auswendig wissen, rufen Sie `route` auf: In

```
$ /sbin/route
Kernel-IP-Routentabelle
Ziel            Router     Genmask         Flags Metric Ref Use Iface
default         fritz.box  0.0.0.0         UG    0      0     0 eth0
link-local      *          255.255.0.0     U     1000   0     0 lo
192.168.178.0   *          255.255.255.0   U     0      0     0 eth0
```

sehen Sie in der `default`-Zeile unter `Ziel`, was Sie verwenden müssen – hier »`ping fritz.box`«.) Wenn das nicht funktioniert, Sie also Meldungen wie

```
Destination Host Unreachable
```

bekommen, ist in Ihrem lokalen Netz etwas nicht in Ordnung. Wenn Sie können, dann fragen Sie einen Kollegen, ob der gerade ins Internet kommt, oder probieren Sie einen anderen Rechner aus: Sollte da alles klappen, ist der Schwarze Peter wieder bei Ihrem Computer. Ansonsten – und möglicherweise auch in diesem Fall – ist es Zeit für den Systemadministrator.

Zum Beispiel könnte es sein, dass Ihre Default-Route verbogen ist und auf den falschen Rechner zeigt. (Zumindest bei manueller Konfiguration wäre das denkbar.) Das würde die Benutzer anderer Rechner, wo die Konfiguration wahrscheinlich stimmt, nicht betreffen.

- Wenn Sie auch den Default-Gateway problemlos erreichen können, dann ist das Problem entweder jenseits Ihres LAN irgendwo im Internet (und damit möglicherweise nicht nur Ihrem, sondern sogar dem Zugriff Ihres Administrators entzogen) oder weiter oben im »Protokollstapel« anzusiedeln. Es kann zum Beispiel sein, dass Sie einen entfernten Web-Server zwar über `ping` kontaktieren können, dass bei Ihrem (Firmen-?)Internetzugang aber der direkte Zugriff ins Web gesperrt ist, weil Sie einen »Proxy-Server« benutzen sollen (und das zu konfigurieren vergessen haben). Ihr Systemadministrator hilft Ihnen weiter.

Eine Netzanbindung, die manchmal funktioniert und manchmal nicht (Knick im Kabel? Mäusefraß?), können Sie mit »`ping -f`« auf die Probe stellen. Hier schickt `ping` nicht wie üblich ein Paket pro Sekunde an den anderen Rechner, sondern so schnell es kann. Für jedes gesendete Paket wird ein Punkt ausgegeben, für jedes empfangene ein »Backspace«-Zeichen, das den Punkt wieder löschen sollte. Gehen Ihnen Pakete verloren, dann entsteht eine länger werdende Zeile von Punkten.

Sind Sie nicht `root`, sondern nur ein normaler Benutzer, dann müssen Sie sich mit einem minimalen Abstand von 0,2 Sekunden zwischen zwei versandten Paketen zufriedengeben. Das Netzwerk überfluten dürfen Sie nämlich nur als Administrator.

Wenn Sie IPv6-Verbindungen prüfen wollen, müssen Sie statt `ping` das Kommando `ping6` verwenden:

```
$ ping6 ::1
```

`dig` Das Kommando `dig` dient zum Testen der Namensauflösung über das DNS. Wenn Sie nichts anderes sagen, versucht es zu einem auf der Kommandozeile angegebenen Namen eine IP-Adresse zu finden:

```
$ dig www.linupfront.de

; <<>> DiG 9.8.1-P1 <<>> www.linupfront.de
;; global options: +cmd
```

```
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 34301
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;www.linupfront.de.             IN      A

;; ANSWER SECTION:
www.linupfront.de.      3600    IN      CNAME   s0a.linupfront.de.
s0a.linupfront.de.      3600    IN      A       31.24.175.68

;; Query time: 112 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Thu Mar  1 16:06:06 2012
;; MSG SIZE  rcvd: 69
```

Diese (sehr geschwätzige) Ausgabe verrät uns in der »QUESTION SECTION«, dass wir nach `www.linupfront.de` gesucht haben (nicht dass wir das nicht sowieso wüßten). Die »ANSWER SECTION« teilt mit, dass zu dem Namen `www.linupfront.de` eigentlich gar keine IP-Adresse gehört, sondern dass `www.linupfront.de` nur ein »Spitzname« für den Rechner `s0a.linupfront.de` ist (eine beliebte Methode). Aber `s0a.linupfront.de` hat die Adresse `31.24.175.68`. Schließlich sehen wir im letzten Block, dass diese Antwort vom DNS-Server auf `127.0.0.1` kommt.

Kommt dagegen längere Zeit keine Antwort und dann schließlich etwas wie

```
; <<>> DiG 9.8.1-P1 <<>> www.linupfront.de
;; global options: +cmd
;; connection timed out; no servers could be reached
```

dann ist etwas faul im Staate Dänemark. Entweder stimmen Ihre Einstellungen in `/etc/resolv.conf` nicht, oder der Nameserver tut nicht das, was er soll.

Sie können gezielt einen bestimmten Nameserver fragen, indem Sie ihn auf der Kommandozeile benennen:

```
$ dig www.linupfront.de @fritz.box                    Frage fritz.box
```

Natürlich sollte zum Auflösen dieses Namens keine aufwendige DNS-Suche nötig sein (ansonsten hätten Sie vielleicht ein Henne-Ei-Problem); im Zweifel können Sie immer die IP-Adresse direkt angeben:

```
$ dig www.linupfront.de @192.168.178.1
```

Wenn Sie sich mit dem DNS auskennen: Sie können `dig` auch verwenden, um nach anderen RR-Typen zu suchen als `A`-Records. Sagen Sie einfach auf der Kommandozeile, was Sie haben wollen:

```
$ dig linupfront.de mx
```

Um den Namen zu einer gegebenen IP-Adresse herauszufinden (falls es einen gibt), müssen Sie die Option `-x` angeben:

```
$ dig +short -x 31.24.175.68
s0a.linupfront.de.
```

(Mit der Option `+short` liefert `dig` eine sehr knappe Antwort.)

Natürlich kann `dig` noch viel mehr, aber das Meiste davon ist nur für Leute von Interesse, die Nameserver konfigurieren oder am Laufen halten müssen. Im Zweifel finden Sie mehr Details in `dig`(1).

netstat Das Kommando `netstat` ist eine Art Schweizer Messer, das Ihnen Informationen aller Art über Ihren Rechner und seine Netzanbindung liefert. Wenn Sie einfach nur

```
$ netstat
```

aufrufen, bekommen Sie eine Liste aller aktiven Verbindungen angezeigt. Dazu zählen nicht nur TCP-Verbindungen, sondern auch lokale Verbindungen über Unix-Domain-Sockets, die genauso erdrückend umfangreich wie überaus langweilig sind. Interessanter ist es, sich zum Beispiel mit

```
$ netstat -tl
Aktive Internetverbindungen (Nur Server)
Proto Recv-Q Send-Q Local Address       Foreign Address  State
tcp        0      0 ceol:domain         *:*              LISTEN
tcp        0      0 ceol-eth.fri:domain *:*              LISTEN
tcp        0      0 *:ssh               *:*              LISTEN
tcp        0      0 ceol:ipp            *:*              LISTEN
tcp        0      0 ceol:postgresql     *:*              LISTEN
tcp        0      0 ceol:smtp           *:*              LISTEN
◁◁◁◁◁
```

eine Liste der TCP-Ports ausgeben zu lassen, auf denen auf dem lokalen Rechner Serverprogramme auf eingehende Verbindungen lauschen.

TCP und UDP verwenden »Ports«, damit derselbe Rechner gleichzeitig mehrere Dienste anbieten oder ansprechen kann. Viele Protokolle haben fest zugeordnete Portnummern – eine Liste finden Sie in der Datei `/etc/services`.

An der Beispielausgabe können Sie etwa sehen, dass der Rechner auf der Adresse `ceol` unter anderem einen DNS-Server (Dienst `domain`), einen CUPS-Server zum Drucken (Dienst `ipp`) sowie einen Mailserver (Dienst `smtp`) anbietet. Den `ssh`-Dienst (Secure Shell) bietet er sogar auf allen konfigurierten Adressen an.

»`netstat -tl`« ist ein wichtiges Werkzeug zur Fehlersuche im Zusammenhang mit Netzwerkdiensten. Wenn ein Dienst hier nicht auftaucht, von dem Sie annehmen, dass er eigentlich in der Liste vorkommen sollte, dann ist das ein Zeichen dafür, dass mit seiner Konfiguration etwas nicht stimmt – entweder sind nicht die richtige Adresse und/oder der richtige Port eingestellt, oder beim Start des Dienstes ist irgendetwas katastrophal schiefgegangen, so dass er gar nicht läuft.

Mit »`-u`« statt »`-t`« sehen Sie die UDP-basierten Serverdienste, und mit »`-p`« werden auch der Name und die PID des Prozesses ausgegeben, der den Dienst erbringt. Letztere funktioniert allerdings nur, wenn Sie das Kommando als `root` aufrufen.

Mit »`-n`« bekommen Sie das Ganze mit IP-Adressen und Portnummern statt Namen. Manchmal ist das aufschlussreicher, jedenfalls sofern Sie ein Gefühl für die Portnummern haben:

```
$ netstat -tln
Aktive Internetverbindungen (Nur Server)
Proto Recv-Q Send-Q Local Address       Foreign Address State
tcp        0      0 127.0.0.1:53        0.0.0.0:*       LISTEN
tcp        0      0 192.168.178.130:53  0.0.0.0:*       LISTEN
tcp        0      0 0.0.0.0:22          0.0.0.0:*       LISTEN
◁◁◁◁◁
```

»`netstat -s`« liefert Ihnen Statistikinformationen in der Form

```
Ip:
    145845 total packets received
    8 with invalid addresses
    0 forwarded
    0 incoming packets discarded
    145837 incoming packets delivered
    138894 requests sent out
    16 outgoing packets dropped
    172 dropped because of missing route
Icmp:
    30 ICMP messages received
    0 input ICMP message failed.
◁◁◁◁◁
```

Und »`netstat -r`« ist im Wesentlichen dasselbe wie »`route`« (ohne Parameter).

Übungen

15.4 [!2] Verwenden Sie `ping`, um sich zu vergewissern, dass Sie einen wohlbekannten Rechner im Internet erreichen können (vielleicht `www.google.com`).

15.5 [1] Benutzen Sie `dig`, um nachzuschauen, für welche IP-Adresse `www.heise.de` steht.

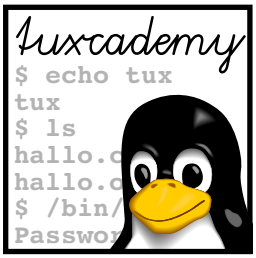
15.6 [2] Die Option `+trace` von `dig` bringt das Programm dazu, die komplette Suchkette für einen Namen zu dokumentieren, beginnend bei den Root-Level-Nameservern. Probieren Sie das für einige interessante Namen aus und prüfen Sie die Zwischenschritte.

15.7 [2] Welche Netzwerkdienste bietet Ihr Rechner an? Welche davon sind von anderen Rechnern aus erreichbar?

Kommandos in diesem Kapitel

Zusammenfassung

- Es gibt drei Arten von Netzwerkprotokollen: Transportprotokolle, Kommunikationsprotokolle und Anwendungsprotokolle.
- In einem »Protokollstapel« verkehrt jedes Protokoll nur mit den unmittelbar darunter- und darüberliegenden Protokollen.
- TCP/IP enthält die Kommunikationsprotokolle IP, TCP (zuverlässig und verbindungsorientiert) und UDP (unzuverlässig und verbindungslos), das Steuerungsprotokoll ICMP und eine Vielzahl von Anwendungsprotokollen auf TCP- oder UDP-Basis.
- Rechner im Internet haben eindeutige IP-Adressen.
- Routing erlaubt die Kommunikation zwischen nicht direkt verbundenen Rechnern.
- Um am Netzwerk teilzunehmen, benötigt ein Linux-Rechner eine IP-Adresse, eine Subnetzadresse mit Maske, ein Default-Gateway und die Adresse mindestens eines DNS-Servers.
- Das DNS ist eine verteilte Datenbank zur Abbildung von Rechnernamen auf IP-Adressen und umgekehrt (unter anderem).
- IPv6 ist der Nachfolgestandard von IPv4, der verschiedene Einschränkungen aufhebt und Verbesserungen enthält.
- `ifconfig` und `route` dienen zur manuellen Konfiguration der Netzanbindung. Distributionen haben verschiedene Methoden zur dauerhaften Netzkonfiguration.
- Die Programme `ifconfig`, `ping`, `dig` und `netstat` dienen zur Fehlersuche im Netz.



A

Musterlösungen

Dieser Anhang enthält Musterlösungen für ausgewählte Aufgaben.

1.1 Der Autor dieser Zeilen machte seine ersten Schritte auf einem »richtigen« Computer 1982 mit einem Apple II plus. Der hatte einen 6502-Prozessor mit der grandiosen Taktfrequenz von 1 MHz und einen für damalige Verhältnisse gigantischen RAM-Speicher von 48 KiB. Eine Festplatte gab es nicht, aber dafür 5¼-Zoll-Disketten, auf die jeweils 143 KiB Daten passten – und das pro Seite. Eigentlich sollte man ja nur eine Seite der Diskette benutzen, aber mit Fingerspitzengefühl oder einem speziellen Locher war es möglich, eine Schreibschutzkerbe in die gegenüberliegende Kante der Plastikhülle zu schneiden, damit man die Diskette wenden konnte. (Bei Preisen von 30 oder mehr Mark für einen Zehnerpack war das auch bitter nötig.)

2.1 Eine gute Quelle ist `http://oreilly.com/catalog/opensources/book/appa.html`. Lesen Sie unbedingt auch `http://www.cs.vu.nl/~ast/reliable-os/`.

2.2 Auf `ftp.kernel.org` steht noch ein `linux-0.01.tar.gz`.

2.3

1. Falsch. GPL-Programme dürfen zu beliebigen Preisen verkauft werden, solange der Käufer den Quellcode bekommt (usw.) und die GPL-Rechte zugesprochen bekommt.
2. Falsch. Firmen sind herzlich eingeladen, eigene Produkte auf der Basis von GPL-Software zu entwickeln, nur fallen diese Produkte auch wieder unter die GPL. Natürlich ist die Firma nicht gezwungen, ihr Produkt an die Allgemeinheit zu verschenken – es genügt, wenn der Quellcode den tatsächlichen direkten Kunden zugänglich gemacht wird, die auch die ausführbaren Programme gekauft haben, aber die dürfen damit alles machen, was die GPL ihnen erlaubt (inklusive die Software an andere Leute weiterverkaufen oder -verschenken).
3. Richtig.
4. Falsch. Sie dürfen ein Programm beliebig *benutzen*, ohne dass Sie die GPL akzeptiert haben. Die GPL regelt erst die *Weitergabe* der Software, und Sie können von der GPL vorher Kenntnis nehmen. (Interaktive Programme sollen Sie ja auf die GPL hinweisen.) Grundsätzlich gilt natürlich die Feststellung, dass nur solche Bestimmungen verbindlich sind, die der Empfänger

der Software *vor* dem Erwerb (Kauf, …) kennen konnte; da die GPL dem Empfänger aber zusätzliche Rechte gibt, die er sonst überhaupt nicht hätte – etwa das Recht der originalgetreuen oder modifizierten Weitergabe –, ist das kein Problem; man kann die GPL völlig links liegenlassen und darf trotzdem alles mit der Software machen, was das Urheberrecht zuläßt. Anders sieht das mit den EULAs proprietärer Programme aus; diese versuchen ein Vertragsverhältnis herzustellen, in dem der Käufer ausdrücklich auf Rechte *verzichtet*, die er laut Urheberrecht hätte (etwa das Recht, die Programmstruktur zu analysieren), und sowas geht natürlich, wenn überhaupt, nur vor dem Kauf.

3.2 Auf dem Textbildschirm gilt, dass in beiden Fällen die Fehlermeldung „`Login incorrect`“ erscheint, aber erst nach der Kennwortabfrage. Der Grund dafür ist, dass es sonst einfach wäre, gültige Benutzernamen zu raten (die, bei denen nicht gleich eine Fehlermeldung erscheint). So, wie das System jetzt ist, kann ein „Cracker“ nicht entscheiden, ob schon der Benutzername ungültig war oder nur das Kennwort falsch ist, was das Einbrechen ins System deutlich erschwert.

4.1 In der Loginshell lautet die Ausgabe »`-bash`«, in der zusätzlich gestarteten »Subshell« dagegen »`bash`«. Das Minuszeichen am Anfang sagt der Shell, dass sie sich wie eine Loginshell benehmen soll und nicht wie eine »normale« Shell, was Auswirkungen auf die Initialisierung hat.

4.2 `alias` ist ein internes Kommando (geht nicht anders). `rm` ist ein externes Kommando. `echo` und `test` sind bei der Bash intern, stehen aber auch als externe Kommandos zur Verfügung, weil andere Shells sie nicht intern implementieren. Bei der Bash sind sie hauptsächlich aus Effizienzgründen intern.

5.2 Probieren Sie »`apropos process`« oder »`man -k process`«.

5.5 Das Format und die Werkzeuge für Info-Dateien wurden Mitte der 1980er Jahre entwickelt. HTML war da noch überhaupt nicht erfunden.

6.1 Das aktuelle Verzeichnis ist in Linux ein Prozessattribut, das heißt, jeder Prozess hat sein eigenes aktuelles Verzeichnis (bei DOS ist das aktuelle Verzeichnis eine Eigenschaft des Laufwerks, aber das ist in einem Mehrbenutzersystem natürlich nicht tragbar). Darum muss `cd` in die Shell eingebaut sein. Wenn es ein externes Kommando wäre, würde es in einem neuen Prozess ausgeführt, würde *dessen* aktuelles Verzeichnis ändern und sich dann prompt beenden, während das aktuelle Verzeichnis der Shell unverändert bliebe.

6.4 Bekommt `ls` einen Dateinamen übergeben, gibt es Informationen nur über diese Datei aus. Bei einem Verzeichnisnamen liefert es Informationen über alle Dateien in dem betreffenden Verzeichnis.

6.5 Hierfür hat `ls` die Option `-d`.

6.6 Das ganze könnte etwa so aussehen:

```
$ mkdir -p grd1-test/dir1 grd1-test/dir2 grd1-test/dir3
$ cd grd1-test/dir1
$ vi hallo
$ cd
$ vi grd1-test/dir2/huhu
$ ls grd1-test/dir1/hallo grd1-test/dir2/huhu
grd1-test/dir1/hallo
grd1-test/dir2/huhu
```

```
$ rmdir grd1-test/dir3
$ rmdir grd1-test/dir2
rmdir: grd1-test/dir2: Directory not empty
```

Damit Sie mit `rmdir` ein Verzeichnis entfernen können, muss dieses (bis auf die zwangsläufig vorhandenen Einträge »`.`« und »`..`«) leer sein.

6.7 Auf die Suchmuster passen jeweils:

(a) `prog.c`, `prog1.c`, `prog2.c`, `progabc.c`

(b) `prog1.c`, `prog2.c`

(c) `p1.txt`, `p2.txt`, `p21.txt`, `p22.txt`

(d) `p1.txt`, `p21.txt`, `p22.txt`, `p22.dat`

(e) alle Dateien

(f) alle Dateien außer `prog` (hat keinen Punkt im Namen)

6.8 »`ls`« ohne Argumente listet den Inhalt des aktuellen Verzeichnisses auf. Verzeichnisse im aktuellen Verzeichnis werden nur namentlich aufgezählt. »`ls`« mit Argumenten dagegen (also auch »`ls *`« – `ls` bekommt das Suchmuster ja nicht zu sehen) listet Informationen über die angegebenen Argumente auf. Für Verzeichnisse heißt das, dass auch der *Inhalt* der Verzeichnisse aufgelistet wird.

6.9 Die Datei »`-l`« (in der Ausgabe des ersten Kommandos zu sehen) wird von `ls` als Option verstanden. Darum taucht sie in der Ausgabe des zweiten Kommandos auch nicht mehr auf, da bei `ls` mit Pfadnamenargumenten nur die als Argument angegebenen Dateien ausgegeben werden.

6.10 Wenn der Stern auf Dateinamen mit Punkt am Anfang passte, dann würde zum Beispiel das rekursive Lösch-Kommando »`rm -r *`« auch den Namen »`..`« bearbeiten. Damit würden nicht nur Unterverzeichnisse des aktuellen Verzeichnisses gelöscht, sondern auch das übergeordnete Verzeichnis und so weiter.

6.11 Hier sind die Kommandos:

```
$ cd
$ cp /etc/services myservices
$ mv myservices src.dat
$ cp src.dat /tmp
$ rm src.dat /tmp/src.dat
```

6.12 Wenn Sie ein Verzeichnis umbenennen, sind alle darin enthaltenen Dateien und Unterverzeichnisse automatisch unter dem neuen Verzeichnisnamen zu finden. Eine `-R`-Option ist daher vollkommen überflüssig.

6.13 Der naive Ansatz – etwas wie »`rm -file`« – schlägt fehl, da `rm` den Dateinamen als Folge von Optionen missversteht. Dasselbe gilt für Kommandos wie »`rm "-file"`« oder »`rm '-file'`«. Die folgenden Möglichkeiten funktionieren besser:

1. Mit »`rm ./-file`« ist das Minuszeichen nicht mehr am Anfang des Parameters und leitet so keine Option ein.

2. Mit »`rm -- -file`« sagen Sie `rm`, dass nach dem »`--`« definitiv keine Optionen mehr kommen, sondern nur noch Dateinamen. Das funktioniert auch bei vielen anderen Programmen.

6.14 Im Rahmen der Ersetzung von »*« wird auch die Datei »-i« erfasst. Da die Dateinamen bei der Ersetzung in ASCII-Reihenfolge in die Kommandozeile getan werden, sieht `rm` eine Parameterliste wie

```
-i a.txt b.jpg c.dat                                    oder was auch immer
```

und hält das »-i« für die *Option* `-i`, die es dazu bringt, Dateien nur nach Rückfrage zu entfernen. Wir hoffen mal, dass das reicht, damit Sie nochmal in Ruhe nachdenken.

6.15 Wenn Sie die Datei über das eine Link im Editor aufrufen und ändern, sollte anschließend auch unter dem anderen Link der veränderte Inhalt zu sehen sein. Es gibt allerdings »clevere« Editoren, die Ihre Datei beim Speichern nicht überschreiben, sondern neu abspeichern und anschließend umbenennen. In diesem Fall haben Sie hinterher wieder zwei verschiedene Dateien.

6.16 Wenn die Zieldatei eines symbolischen Links nicht (mehr) existiert, dann gehen Zugriffe auf das Link ins Leere und führen zu einer Fehlermeldung.

6.17 Auf sich selbst. Man kann das Wurzelverzeichnis daran erkennen.

6.18 Das Verzeichnis `/home` liegt auf diesem Rechner auf einer eigenen Partition und hat im Dateisystem auf jener Partition die Inodenummer 2, während das Verzeichnis `/` auf seinem eigenen Dateisystem die Inodenummer 2 hat. Da Inodenummern nur dateisystemweit eindeutig sind, kann in der Ausgabe von »`ls -i`« durchaus bei verschiedenen Dateien dieselbe Zahl stehen; das ist kein Grund zur Besorgnis.

6.19 Im Wurzelverzeichnis (und nur in diesem) haben ».« und »..« dieselbe Inode-Nummer. Programme, die ausgehend von einem bestimmten Verzeichnis die »..«-Links verfolgen, um das Wurzelverzeichnis zu finden, können so feststellen, dass sie dort angekommen sind.

6.20 Harte Links sind ununterscheidbare, also gleichberechtigte Namen für eine Datei (oder hypothetischerweise ein Verzeichnis). Jedes Verzeichnis hat aber ein »Link« namens »..«, das auf das Verzeichnis »darüber« verweist. Dieses Link kann es pro Verzeichnis nur einmal geben, was sich mit der Idee mehrerer gleichberechtigter Namen nicht verträgt. Ein anderes Argument dagegen ist, dass es für jeden Namen einen eindeutigen Pfad geben muss, der in endlich vielen Schritten zum Wurzelverzeichnis (`/`) führt. Wären Links auf Verzeichnisse erlaubt, dann könnte eine Befehlsfolge wie

```
$ mkdir -p a/b
$ cd a/b
$ ln .. c
```

zu einer Schleife führen.

6.21 Der Referenzzähler für das Unterverzeichnis hat den Wert 2 (ein Verweis entsteht durch den Namen des Unterverzeichnisses in ~, einer durch das .-Link im Unterverzeichnis selbst). Hätte das Unterverzeichnis weitere Unterverzeichnisse, dann würden die ..-Links dieser Verzeichnisse den Referenzzähler in die Höhe treiben.

6.22 Die Kette der symbolischen Links wird verfolgt, bis Sie bei etwas ankommen, was kein symbolisches Link ist. Die maximale Länge solcher Ketten ist allerdings in der Regel beschränkt (siehe Übung 6.23).

6.23 Die Untersuchung dieser Fragestellung wird einfacher, wenn man Shell-Schleifen benutzen kann (siehe Abschnitt 9.6.) Etwas wie

```
$ touch d
$ ln -s d L1
$ i=1
$ while ls -lH L$i >/dev/null
> do
>   ln -s L$i L$((i+1))
>   i=$((i+1))
> done
```

legt eine »Kette« von symbolischen Links an, wobei jedes Link auf das vorige zeigt. Dies wird fortgesetzt, bis das »`ls -lH`«-Kommando fehlschlägt. An der Fehlermeldung können Sie dann sehen, welche Länge gerade noch erlaubt ist. (Auf dem Rechner des Autors ist das Ergebnis »40«, was im wirklichen Leben keine allzu störende Einschränkung darstellen dürfte.)

6.24 Harte Links brauchen fast keinen Platz, da es sich dabei nur um zusätzliche Verzeichniseinträge handelt. Symbolische Links sind eigene Dateien und kosten darum zumindest schon mal ein Inode (jede Datei braucht ein eigenes Inode). Dazu kommt der Speicherplatz für den Namen der Zieldatei. Prinzipiell wird Plattenplatz an Dateien in Einheiten der Blockgröße des Dateisystems vergeben (also 1 KiB und mehr, meist 4 KiB), aber in den `ext-` Dateisystemen gibt es eine spezielle Ausnahme für »kurze« symbolische Links (kleiner als ca. 60 Bytes), die im Inode selber gespeichert werden und keinen Datenblock brauchen. Andere Dateisysteme wie das Reiser-Dateisystem gehen von sich aus sehr effizient mit kurzen Dateien um, so dass auch dort der Platzbedarf für symbolische Links vernachlässigbar sein dürfte.

6.25 Ein mögliches Kommando wäre »`find / -size +1024k -print`«.

6.26 Der grundlegende Ansatz ist etwas wie

```
find . -maxdepth 1 ⟨Tests⟩ -ok rm '{}' \;
```

Die ⟨*Tests*⟩ sollten dabei die Datei so eindeutig wie möglich bestimmen. Das »`-maxdepth 1`« sorgt dafür, dass Unterverzeichnisse nicht durchsucht werden. Im naheliegendsten Fall verschaffen Sie sich mit »`ls -i`« die Inodenummer der Datei (zum Beispiel 4711) und löschen Sie dann mit

```
find . -maxdepth 1 -inum 4711 -exec rm -f '{}' \;
```

6.27 Schreiben Sie in die Datei `.bash_logout` in Ihrem Heimatverzeichnis etwas wie

```
find /tmp -user $LOGNAME -type f -exec rm '{}' \;
```

oder – effizienter –

```
find /tmp -user $LOGNAME -type f -print0 \
    | xargs -0 -r rm -f
```

(in der Umgebungsvariablen `LOGNAME` steht der aktuelle Benutzername).

6.28 Verwenden Sie ein Kommando wie »`locate '*/README'`«. Natürlich würde »`find / -name README`« es auch tun, aber das braucht *viel* länger.

6.29 Direkt nach dem Anlegen steht die neue Datei nicht in der Datenbank und wird entsprechend auch nicht gefunden (Sie müssen erst `updatedb` laufen lassen). Die Datenbank bekommt auch nichts davon mit, dass Sie die Datei gelöscht haben, bis Sie wiederum `updatedb` aufrufen. – Am geschicktesten rufen Sie `updatedb` übrigens nicht direkt auf, sondern über das Shellskript, das auch Ihre Distribution benutzt (etwa `/etc/cron.daily/find` bei Debian GNU/Linux). Auf diese Weise stellen Sie sicher, dass `updatedb` dieselben Parameter verwendet wie sonst auch immer.

6.30 `slocate` sollte nur diejenigen Dateinamen ausgeben, auf die der aufrufende Benutzer auch zugreifen darf. Die Datei `/etc/shadow`, in der die verschlüsselten Kennwörter der Benutzer stehen, ist dem Systemverwalter vorbehalten (siehe auch *Linux-Administration I*).

7.1 Der reguläre Ausdruck *r*+ ist nur eine Abkürzung für *rr**, auf + könnte man also verzichten. Anders sieht es bei ? aus, dafür gibt es keinen Ersatz, jedenfalls wenn man (wie bei `grep` vs. `egrep`) annehmen muss, dass man statt *r*? auch nicht `\(\|`*r*`\)` sagen kann (GNU-`grep` unterstützt das synonyme *r*`{,1}` – siehe Tabelle 7.1 –, aber die `grep`-Implementierungen herkömmlicher Unixe kennen das nicht).

7.2 *Dafür* brauchen Sie eine Musterlösung? Machen Sie sich nicht lächerlich. – Naja … weil Sie's sind …

```
egrep '\<(Königst|T)ochter\>' frosch.txt
```

7.3 Eine Möglichkeit wäre

```
grep :/bin/bash$ /etc/passwd
```

7.4 Wir suchen nach Wörtern, die mit einer (möglicherweise leeren) Folge von Konsonanten anfangen, dann kommt ein »a«, dann wieder möglicherweise Konsonanten, dann ein »e« und so weiter. Wir müssen vor allem aufpassen, dass uns keine Vokale »durchrutschen«. Der resultierende reguläre Ausdruck ist relativ unappetitlich, darum erlauben wir uns eine kleine Schreibvereinfachung:

```
$ k='[^aeiou]*'
$ grep -i ^${k}a${k}e${k}i${k}o${k}u${k}$ /usr/share/dict/words
abstemious
abstemiously
abstentious
acheilous
acheirous
acleistous
affectious
annelidous
arsenious
arterious
bacterious
caesious
facetious
facetiously
fracedinous
majestious
```

(Nachschlagen dürfen Sie die Wörter selber.)

7.5 Probieren Sie mal

```
egrep '(\<[A-Za-z]{4,}\>).*\<\1\>' frosch.txt
```

Wir brauchen `egrep` für den Rückbezug. Die Wortklammern müssen auch um den Rückbezug herum angegeben werden (probieren Sie es mal ohne!). Umlaute werden hier der besseren Übersicht wegen ignoriert.

8.1 Eine (wahrscheinliche) Möglichkeit ist, dass das Programm `ls` nach etwa dem folgenden Schema abläuft:

```
Lies die Verzeichnisdaten in Liste l ein;
if (Option -U nicht angegeben) {
   Sortiere die Elemente von l;
}
Gib l auf die Standardausgabe aus;
```

Es würde also nach wie vor erst alles gelesen, dann sortiert (oder eben nicht sortiert) und dann ausgegeben.

Die andere Erklärung ist, dass zum Zeitpunkt des Lesens des `inhalt`-Eintrags wirklich noch nichts in die Datei geschrieben wurde; aus Effizienzgründen puffern die meisten Programme, wenn sie in Dateien schreiben sollen, ihre Ausgabe intern zwischen und führen tatsächliche Betriebssystemaufrufe zum Schreiben erst aus, wenn wirklich substanzielle Datenmengen (typischerweise 8192 Bytes) zusammengekommen sind. Dies läßt sich bei Programmen beobachten, die relativ langsam sehr viel Ausgabe erzeugen; hier wächst eine Ausgabedatei in Schritten von 8192 Bytes.

8.2 Wenn `ls` auf den Bildschirm (oder allgemein ein »bildschirmartiges« Gerät) schreibt, dann formatiert es die Ausgabe anders als wenn es in eine »richtige« Datei schreibt: Es versucht, mehrere Dateinamen in einer Zeile anzuzeigen, wenn die Länge der Dateinamen es zuläßt, und kann je nach Einstellung die Dateinamen auch entsprechend ihres Typs färben. Bei der Ausgabeumlenkung in eine »richtige« Datei werden einfach nur die Namen ohne Formatierung zeilenweise ausgegeben.

Auf den ersten Blick scheint das der Behauptung zu widersprechen, dass Programme gar nicht mitbekommen, dass ihre Ausgabe nicht auf den Bildschirm, sondern anderswohin geht. Im Normalfall ist die Behauptung richtig, aber wenn ein Programm sich ernsthaft dafür interessiert, ob sein Ausgabeziel ein bildschirmartiges Gerät (vulgo »Terminal«) ist, kann es das System danach fragen. Im Falle von `ls` ist die Überlegung dahinter einfach die, dass die Terminalausgabe normalerweise von menschlichen Benutzern angeschaut wird und man diesen möglichst viel Information bieten will. Umgeleitete Ausgabe dagegen wird von anderen Programmen verarbeitet und sollte einfach sein; daher die Beschränkung auf einen Dateinamen pro Zeile und der Verzicht auf Farben, die ja über Terminalsteuerzeichen angesprochen werden und die Ausgabe »verunreinigen« würden.

8.3 Die Shell arrangiert die Ausgabeumlenkung, bevor das Kommando aufgerufen wird. Das Kommando sieht also nur noch eine leere Eingabedatei, was in der Regel nicht zum erwarteten Ergebnis führt.

8.4 Die Datei wird von vorne gelesen und gleichzeitig hinten um alles Gelesene verlängert, sie wächst also, bis sie allen freien Plattenplatz einnimmt.

8.5 Dazu müssen Sie die Standard-Ausgabe in die Standard-Fehlerausgabe umlenken:

```
echo Fehler >&2
```

8.6 Prinzipiell spricht nichts gegen

```
… | tee bla | tee fasel | …
```

Kürzer ist allerdings

```
… | tee bla fasel | …
```

Siehe hierzu auch die Dokumentation zu `tee` (Handbuch- oder Info-Seite).

8.7 Leiten Sie die Dateiliste durch »`cat -A`«.

8.8 Eine Möglichkeit wäre »`head -n 13 | tail -n 1`«.

8.10 `tail` merkt das, gibt eine Warnung aus und macht am neuen Dateiende weiter.

8.11 Im `tail`-Fenster steht

```
Hallo
elt
```

Die erste Zeile kommt vom ersten `echo`; das zweite `echo` überschreibt den kompletten Inhalt der Datei, aber »`tail -f`« weiß, dass es die ersten sechs Zeichen der Datei (»`Hallo`« plus der Zeilentrenner) schon ausgegeben hat – es wartet nur darauf, dass die Datei länger wird, und liefert lediglich das, was neu dazu gekommen ist, nämlich »`elt`« (und ein Zeilentrenner).

8.14 Die Zeile mit dem Namen »von Traben« wird falsch einsortiert, weil in ihr das zweite Feld nicht wirklich der Vorname ist, sondern das Wort »Traben«. Wenn Sie sich die Beispiele genau anschauen, werden Sie feststellen, dass die Sortierung immer stimmt – nur halt für »Traben« statt »Gesine«. Dies ist ein eindeutiges Argument für die zweite Form der Beispieldatei, die mit den Doppelpunkten als Trenner.

8.15 Mit »`sort -k 1.4,1.8`« können Sie die Zeilen nach der Jahreszahl sortieren. Sind zwei Zeilen gemäß dem Sortierschlüssel gleich, macht `sort` einen »Notvergleich« über die ganze Zeile, der hier dazu führt, dass innerhalb der Jahre die Monate korrekt sortiert werden. Wenn Sie gerne sichergehen und ganz explizit sein möchten, können Sie natürlich auch »`sort -k 1.4,1.8 -k 1.1,1.2`« schreiben.

8.19 Benutzen Sie etwas wie

```
cut -d: -f 4 /etc/passwd | sort -u | wc -l
```

Das `cut`-Kommando isoliert die Gruppennummer in jeder Zeile der Benutzerdatenbank. »`sort -u`« liefert eine sortierte Liste aller Gruppennummern, in der jede Gruppennummer nur einmal vorkommt. »`wc -l`« schließlich zählt die Zeilen der Liste, das Ergebnis ist die Anzahl der verschiedenen primären Gruppen.

9.1 Zum Beispiel:

1. `%d-%m-%Y`
2. `%y-%j (KW%V)`
3. `%Hh%Mm%Ss`

9.2 Wir wissen das auch nicht genau, aber versuchen Sie testhalber mal etwas wie »`TZ=America/Los_Angeles date`«.

9.4 Wenn Sie eine Umgebungsvariable im Kindprozess ändern, bleibt der Wert der Variablen im Elterprozess davon unbeeinflusst. Es gibt Mittel und Wege, Informationen an den Elterprozess zurückfließen zu lassen, aber dies ist keiner davon.

9.5 Starten Sie eine neue Shell und entfernen Sie die Umgebungsvariable `PATH` aus deren Umgebung, ohne jedoch die Variable selbst zu löschen. Versuchen Sie, externe Programme zu starten. – Wenn `PATH` gar nicht existiert, startet die Shell keine externen Programme.

9.6 Eine systemunabhängige Musterlösung können wir hierfür leider nicht geben; probieren Sie es selber aus (mit `which`).

9.7 Sie sollten mit »`whereis`« zwei Dateien namens `/usr/share/man/man1/crontab.1.gz` und `/usr/share/man/man5/crontab.5.gz` finden. Die erstere enthält die Dokumentation für das eigentliche `crontab`-Kommando, die letztere die Dokumentation für das Dateiformat, das Sie mit `crontab` anlegen können. (Die Details sind für diese Aufgabe nicht wichtig; siehe *Linux-Administration I*.)

9.8 Die Bash verwendet Zeichenfolgen der Form »`!`⟨*Zeichen*⟩« zum Zugriff auf alte Kommandos (eine Alternative zu den Tastaturfunktionen wie Strg+r, die sich aus der C-Shell in die Bash hinübergerettet hat). Die Zeichenfolge »`!"`« hat aber keine Funktion, sondern gilt als Syntaxfehler.

9.9 Keiner.

9.10 Wenn der Dateiname als Parameter übergeben wird, fühlt `wc` sich bemüßigt, ihn mit der Zeilenzahl auszugeben. Wenn `wc` von der Standardeingabe liest, gibt es nur die reine Zeilenzahl aus.

9.11 Versuchen Sie etwas wie

```
#!/bin/bash
pattern=$1
shift
◁◁◁◁◁
for f
do
    grep $pattern "$f" && cp "$f" backup
done
```

Nach dem `shift` ist das Suchmuster nicht mehr der erste Parameter, und das wirkt sich auch auf »`for f`« aus.

9.12 Der `-f`-Dateitest bezieht sich, wenn er auf ein symbolisches Link angewendet wird, auf die Datei (oder Verzeichnis oder was auch immer), auf die das Link zeigt. Er ist also auch dann erfolgreich, wenn der betrachtete Name eigentlich nur ein symbolisches Link ist. (Warum hat `filetest2` dieses Problem *nicht*?)

10.2 Das können Sie mit etwas wie

```
ls /bin /sbin /usr/bin /usr/sbin | wc -l
```

bestimmen. Alternativ dazu können Sie einfach an der Eingabeaufforderung der Shell zweimal Tab drücken – die Shell antwortet dann mit etwas wie

```
Display all 2371 possibilities? (y or n)
```

und das ist – in Abhängigkeit von Ihrem `PATH` – Ihre Antwort. (Wenn Sie als normaler Benutzer angemeldet sind, dann sind die Programme in `/sbin` und `/usr/sbin` in der Regel nicht dabei.)

10.3 Benutzen Sie statt »`grep` ⟨*Muster*⟩ `*.txt`« das Kommando »`grep` ⟨*Muster*⟩ `*.txt /dev/null`«. Damit hat `grep` immer mindestens zwei Dateinamenparameter, wobei `/dev/null` die Ausgabe nicht weiter verfälscht. – Die unter Linux gebräuchliche GNU-Implementierung von `grep` unterstützt eine Option `-H`, die dasselbe bewirkt, aber das ist nicht portabel.

10.4 Bei `cp` auf eine existierende Zieldatei wird die Datei zum Schreiben geöffnet und auf die Länge 0 gesetzt, bevor die Quelldaten hineingeschrieben werden. Bei `/dev/null` führt das nur dazu, dass die Quelldaten verschwinden. Bei `mv` auf eine existierende Zieldatei wird die Zieldatei jedoch erst gelöscht – und das ist eine Verzeichnisoperation, die ungeachtet der besonderen Natur von `/dev/null` einfach den Namen `null` aus dem Verzeichnis `/dev` entfernt und eine neue Datei namens `null` mit dem Inhalt von `bla.txt` dort anlegt.

10.6 Es ist unklug, weil es zum einen nicht richtig funktioniert, zum zweiten die Daten sowieso nicht sichernswert sind, weil sie sich ständig ändern (man verschwendet also jede Menge Platz auf dem Sicherungsmedium und Zeit zum Kopieren), und zum dritten eine solche Sicherheitskopie überhaupt nicht wieder eingespielt werden kann. Unkontrollierte Schreibvorgänge zum Beispiel auf `/proc/kcore` führen mit an Sicherheit grenzender Wahrscheinlichkeit zum Systemabsturz.

11.1 Weil `AA` kürzer ist als `*2A`.

11.2 Das Hauptproblem besteht darin, den Stern auszudrücken. Im einfachsten Fall könnten Sie etwa »`A*1*2B*4*A`« schreiben. Die Komprimierung leidet natürlich darunter, dass Sie den einzelnen Stern durch drei Zeichen ausdrücken; Sie könnten als Ausnahme definieren, dass zum Beispiel ** für einen einzelnen Stern steht, aber das macht die Entkomprimierung ein bisschen komplizierter.

11.3 Verwenden Sie dazu die Kommandos »`ls -l >inhalt`« und »`tar -cvf inhalt.tar inhalt`«. Sie werden feststellen, dass das Archiv wesentlich größer als das Original ist. Das liegt an den Metadaten des Archivs. `tar` komprimiert nicht, sondern archiviert. Aus einer Datei ein Archiv, das heißt eine Datei zu erstellen, ist nicht wirklich eine brauchbare Idee.

11.4 Geben Sie beispielsweise »`touch datei{1,2,3}`« und »`tar -rvf inhalt.tar datei*`« ein.

11.5 Entpacken Sie das Archiv mit »`tar -xvf inhalt.tar`«.

11.6 Wenn Sie `etc-backup.tar` auf dem anderen Rechner auspacken (etwa weil Sie nachschauen möchten, was drinsteht) und das Archiv absolute Pfadnamen enthält, werden die Daten nicht in ein Unterverzeichnis `etc` des aktuellen Verzeichnisses geschrieben, sondern sie landen im `/etc`-Verzeichnis des anderen Rechners. Das ist mit großer Wahrscheinlichkeit nicht das, was Sie haben wollen. (Natürlich sollten Sie darauf achten, beim Auspacken eines Archivs mit *relativen* Pfadnamen nicht gerade in `/` zu stehen.)

11.7 Falls Sie `gzip` verwenden möchten, geben Sie »`gzip -9 inhalt.tar`« ein.

11.8 Achtung, für mit `gzip` komprimierte `tar`-Archive brauchen Sie zum Manipulieren mit `tar` die zusätzliche Option `-z`: »`tar -tzf inhalt.tar.gz`«. Um das ursprüngliche *Archiv* wieder herzustellen, benötigen Sie das Kommando »`gunzip inhalt.tar.gz`«.

11.9 Versuchen Sie es mit »`tar -cvzf /tmp/homearchiv.tar ~`«.

11.10 Hier ist ein Beispiel dafür, wie der Messvorgang organisiert werden kann:

In den Cache

```
$ unxz linux-3.18.tar.xz
$ for prog in gzip bzip2 xz
> do
>   echo $prog
>   time $prog -k linux-3.18.tar
> done
$ ls -l linux-3.18.tar*
```

Beachten Sie, dass wir die Option `-k` benutzen, damit die Komprimierungsprogramme das »Original« nicht löschen. (Die Ähnlichkeiten der Kommandooptionen aller Programme kommt hier nützlich.)

11.13 `unzip` bietet Ihnen an, die Datei im Archiv zu ignorieren oder umzubenennen oder die existierende Datei zu überschreiben.

11.14 Probieren Sie etwas wie

```
$ unzip files.zip "a/*" -x "*/*.c"
```

12.1 Ein einfaches `su` (ohne `/bin/` davor) würde im Prinzip auch klappen. Allerdings ist es sinnvoll, sich das `/bin/su` anzugewöhnen, weil Sie dadurch besser gegen »trojanische Pferde« geschützt sind: Ein cleverer Benutzer könnte in einem Verzeichnis in seinem `$PATH` ein Programm unterbringen, das `su` heißt und vor dem in `/bin` gefunden wird. Dann ruft er Sie wegen irgendeinem vorgeschützten Problem an seinen Rechner, Sie sagen sich »Das habe ich gleich, ich leihe mir nur mal die Terminalsitzung hier aus, um kurz `root` zu werden«, rufen nichtsahnend »`su`« auf, und das Programm Ihres cleveren Benutzers fragt Sie nach dem `root`-Kennwort, so wie das echte `su` es auch machen würde. Bis darauf, dass es das Kennwort irgendwo in eine Datei sichert, eine Fehlermeldung ausgibt und sich selbst löscht. Sie denken natürlich, dass Sie sich beim Kennwort vertippt haben, rufen nochmal »`su`« auf, bekommen diesmal das echte Programm und ab hier ist alles OK – bis darauf, dass Ihr cleverer Benutzer jetzt das `root`-Kennwort weiß. Wenn Sie von Anfang an »`/bin/su`« benutzen, kann dieser Angriff nicht so einfach funktionieren.

12.2 Mit `su` kann irgendein beliebiger Benutzer Administratorrechte bekommen, wenn er das `root`-Kennwort weiß. Mit `sudo` werden Sie dagegen nur dann Erfolg haben, wenn Sie in der `sudo`-Konfiguration als befugt eingetragen sind. `sudo` fragt Sie also nicht nach dem Kennwort, um herauszufinden, ob Sie überhaupt `root` *sein* dürfen, sondern um sicherzustellen, dass Sie *Sie* sind. (Sie könnten mal eben rausgegangen sein und irgendwer hat sich über Ihren Rechner hergemacht.) Dafür ist Ihr eigenes Kennwort genauso nützlich wie das `root`-Kennwort. Besonders praktisch ist `sudo`, wenn Sie sich die Systemadministration mit ein paar Kollegen teilen, da dann *niemand* das tatsächliche `root`-Kennwort wissen muss – Sie können etwas sehr Langes und Kompliziertes vergeben und es nach der Grundinstallation des Rechners in einem verschlossenen Umschlag in den Safe tun (für Notfälle). Bei `su` müßten *alle* Administratoren das Kennwort wissen, was Änderungen kompliziert

macht. (Die SUSE-Distributionen verwenden auch bei `sudo` das `root`-Kennwort und sind anscheinend sogar noch stolz auf diese Form von Hirnschaden.)

12.7 Hier ist die RPM-Variante:

```
$ rpm --query --all | wc -l
```

Und auf einem Debian-artigen System sollten Sie etwas verwenden wie

```
$ dpkg --list | grep ^ii | wc -l
```

13.1 Durch deren numerische UID und GID.

13.2 Das funktioniert, ist aber nicht notwendigerweise eine gute Idee. Für das System ist das dann derselbe Benutzer, das heißt, alle Dateien und Prozesse mit der entsprechenden UID gehören beiden Benutzern.

13.3 Die UIDs von Pseudobenutzern werden von (Dienst-)Programmen genutzt, die dadurch exakt definierte Zugriffsrechte bekommen.

13.4 Wer in der Gruppe `disk` ist, hat Lese- und Schreibrecht auf die Platten auf Blockebene. Mit Kenntnissen über die Dateisystemstruktur ist es einfach, eine Kopie von `/bin/sh` zu einer SUID-`root`-Shell (Abschnitt 14.4) zu machen, indem man direkt die Verwaltungsinformationen auf der Platte ändert. Damit ist Mitgliedschaft in der Gruppe `disk` im Wesentlichen äquivalent zu `root`-Rechten; Sie sollten niemanden in die Gruppe `disk` aufnehmen, dem Sie nicht das `root`-Kennwort verraten würden.

13.5 Dort finden Sie in der Regel ein »x«. Das ist ein Verweis, dass das Kennwort, welches normalerweise dort stehen würde, in einer anderen Datei steht, nämlich in der `/etc/shadow`, die im Gegensatz zu der ersten Datei nur für `root` lesbar ist.

13.6 Es gibt im Grunde zwei Möglichkeiten:

1. Nichts. In diesem Fall sollte das System Sie abweisen, nachdem Sie auch noch Ihr Kennwort eingegeben haben, weil kein Benutzerkonto mit dem rein großgeschriebenen Benutzernamen korrespondiert.
2. Das System redet ab jetzt mit Ihnen ebenfalls rein in Großbuchstaben. In diesem Fall geht Ihr Linux-System davon aus, dass Sie vor einem absolut vorsintflutlichen Terminal (1970er Jahre oder so) sitzen, das keine Kleinbuchstaben anzeigen kann, und schaltet die Verarbeitung der Eingabe- und Ausgabedaten freundlicherweise so um, dass Großbuchstaben in der Eingabe als Kleinbuchstaben interpretiert und Kleinbuchstaben in der Ausgabe als Großbuchstaben angezeigt werden. Heute ist das von eingeschränktem praktischen Nährwert (es sei denn, Sie arbeiten in einem Computermuseum), und Sie sollten sich schleunigst wieder abmelden, bevor Ihnen der Kopf platzt. Weil dieses Benehmen so atavistisch ist, spielt auch nicht jede Linux-Distribution hierbei mit.

13.7 Benutzen Sie `getent`, `cut` und `sort`, um Listen der Benutzernamen für die Datenbanken zu generieren und `comm`, um die beiden Listen zu vergleichen.

13.8 Verwenden Sie den Befehl `passwd`, wenn Sie als Benutzer `hugo` eingeloggt sind oder »`passwd hugo`« als `root`. In der Datei sollte dann ein anderer Eintrag im zweiten Feld auftauchen, das Datum der letzten Kennwortänderung (Feld 3) sollte das aktuelle Datum tragen (in welcher Einheit?).

13.9 Sie geben ihm mit »`passwd trottel`« als `root` ein neues Kennwort, denn lesen können Sie sein altes auch mit `root`-Rechten nicht.

13.10 Verwenden Sie dazu den Befehl »`passwd -n 7 -x 14 -w 2 hugo`«. Testen können Sie die Einstellungen mit »`passwd -S`«.

13.11 Zum Anlegen des Benutzers verwenden Sie das Programm `useradd`, zum Verändern der UID »`usermod -u`«. Anstatt des Login-Namens sollte den Dateien nur noch eine UID als Besitzer zugeordnet sein, zu der ja kein Login-Name mehr bekannt ist …

13.12 Für jedes der drei Konten sollte je eine Zeile in `/etc/passwd` und in `/etc/shadow` vorhanden sein. Um mit den Konten arbeiten zu können, benötigen Sie nicht unbedingt ein Kennwort (Sie können als `root` das Programm `su` verwenden), wohl aber, um sich unter der neuen Kennung einzuloggen. Eine Datei können Sie auch ohne Heimatverzeichnis anlegen und zwar in `/tmp` (falls Sie es vergessen haben, ein Heimatverzeichnis wäre für einen Benutzer aber schon nicht schlecht).

13.13 Verwenden Sie zum Löschen den Befehl `userdel`. Um die Dateien zu löschen verwenden Sie das Kommando »`find / -uid` ⟨*UID*⟩ `-delete`« .

13.14 Verwenden Sie »`usermod -u`«, dann müssen Sie die Dateien des Benutzers der neuen UID zuordnen, zum Beispiel mit »`find / -uid` ⟨*UID*⟩ `-exec chown test1 {} ';'`« oder (besser) »`chown -R --from=`⟨*UID*⟩ `test1 /`«. ⟨*UID*⟩ ist dabei jeweils die (numerische) alte UID.

13.15 Sie können dazu unter anderem `/etc/passwd` mit `vipw` editieren, oder Sie verwenden `usermod`.

13.16 Gruppen bieten die Möglichkeit, differenziert Rechte an Gruppen (ach was?) von Benutzern zu vergeben. So können Sie zum Beispiel alle Mitarbeiter Ihrer Personalabteilung einer Gruppe zuordnen und dieser Gruppe ein eigenes Arbeitsverzeichnis auf einem File-Server zuteilen. Außerdem können Gruppen dazu dienen, die Rechte zum Zugriff auf bestimmte Peripheriegeräte zu steuern (etwa über die Gruppen `disk`, `audio` oder `video`).

13.17 Verwenden Sie das Kommando »`mkdir` ⟨*Verzeichnis*⟩« zum Erstellen und »`chgrp` ⟨*Gruppenname*⟩ ⟨*Verzeichnis*⟩« zum Verschenken des Verzeichnisses. Sinnvollerweise setzen Sie dann noch das SGID-Bit, damit im Verzeichnis angelegte Dateien sofort der Gruppe gehören.

13.18 Verwenden Sie dazu die folgenden Kommandos:

```
# groupadd test
# gpasswd -a test1 test
Adding user test1 to group test
# gpasswd -a test2 test
Adding user test2 to group test
# gpasswd test
Changing the password for group test
New Password:x9q.Rt/y
Re-enter new password:x9q.Rt/y
```

Zum Gruppenwechsel verwenden Sie das Kommand »`newgrp test`«. Nur wenn Sie nicht Mitglied der entsprechenden Gruppe sind, werden Sie nach dem Kennwort gefragt.

14.1 Eine neue Datei bekommt die Gruppe, die gerade Ihre primäre Gruppe ist. Sie können eine Datei keiner Gruppe zuordnen, in der Sie nicht selber Mitglied sind – es sei denn, Sie sind `root`.

14.3 Hierbei handelt es sich um das SUID- bzw. SGID-Bit. Die Bits bewirken, dass ein Prozess die UID/GID der ausführbaren Datei und nicht des ausführenden Benutzers bekommt. Sie bekommen es mit »`ls -l`« angezeigt. Selbstverständlich können Sie alle Rechte von eigenen Dateien ändern. Sinnvoll ist zumindest das SUID-Bit aber nur bei echten ausführbaren Dateien, also nicht bei Shell-Skripten o.ä.

14.4 Eine der beiden folgenden (gleichwertigen) Möglichkeiten genügt:

```
$ umask 007
$ umask -S u=rwx,g=rwx
```

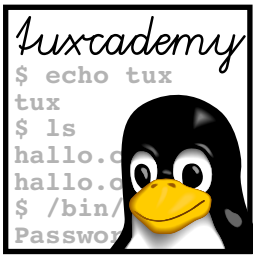
Sie werden sich vielleicht fragen, was die `x`-Bits in dieser *umask* zu suchen haben. Für Dateien sind sie in der Tat nicht relevant, da Dateien standardmäßig nicht ausführbar erzeugt werden. Allerdings könnte es sein, dass im Projektverzeichnis auch Unterverzeichnisse erwünscht sind, und dann ist es sinnvoll, diese gleich so mit Rechten zu versehen, dass man mit ihnen auch vernünftig arbeiten kann.

14.5 Das sogenannte *sticky bit* bewirkt in den betroffenen Verzeichnissen, dass nur der Besitzer einer Datei diese auch löschen/umbenennen kann. Sie finden es unter anderem beim Verzeichnis `/tmp`.

14.7 Mit der Bash funktioniert das nicht (jedenfalls nicht ohne Weiteres). Für andere Shells können wir hier nicht sprechen.

15.2 1022 ($= 2^{32-22} - 2$). Ein nützliches Programm für solche Berechnungen – wenn Sie sie nicht im Kopf machen mögen – heißt `ipcalc`.

15.3 Sie müssen eine Adresse in Deutschland haben und zwei Nameserver angeben können, die die DNS-Daten für die Domain vorhalten. Normalerweise registrieren Sie `de`-Namen nicht direkt, sondern über Ihren Internet-Anbieter oder eine andere Firma, die solche Dienstleistungen übernimmt (die zulässige Vergabestelle, das DENIC, redet nicht gerne mit Privatleuten und schreckt diese darum durch absolut astronomisch exorbitant wuchermäßige Mondpreise für eine Leistung ab, für die die anderen Firmen fünf Euro im Jahr oder so verlangen). Diese Firmen stellen Ihnen in der Regel gerne die Nameserver zur Verfügung und bieten unter Umständen sogar einen »Treuhandservice« an, bei dem ein in Deutschland ansässiger Treuhänder als Domain-»Eigentümer« fungiert, wenn Sie nicht selbst in Deutschland wohnen. Außerdem muss der Domainname anderweitig unbenutzt sein und darf nur Buchstaben, Ziffern, den Punkt, das Minuszeichen und die in der deutschen Sprache üblichen Umlaute enthalten. Manche Namen, insbesondere solche, die man in die Nähe nationalsozialistischer Propaganda rücken könnte, werden vom DENIC nicht registriert oder, falls sie doch registriert wurden und jemandem auffallen, wieder entfernt – aber dafür gibt es keine festen Regeln.



B

Beispieldateien

Als Beispiel verwenden wir an verschiedenen Stellen das Märchen vom Froschkönig, genauer gesagt »Der Froschkönig oder der eiserne Heinrich«, aus den »Deutschen Kinder- und Hausmärchen« der Gebrüder Grimm. Das Märchen drucken wir hier ganz und in der kompletten Form so wie in der Datei ab, um Vergleiche mit den Beispielen zuzulassen.

```
Der Froschkönig oder der eiserne Heinrich

In alten Zeiten, als das Wünschen noch geholfen hat, lebte einmal ein
König, der hatte wunderschöne Töchter. Die jüngste von ihnen war so
schön, daß die Sonne selber, die doch so vieles schon gesehen hat,
sich verwunderte, sooft sie ihr ins Gesicht schien.

Nahe bei dem Schlosse war ein großer, dunkler Wald, und mitten darin,
unter einer alten Linde, war ein Brunnen. Wenn nun der Tag recht heiß war,
ging die jüngste Prinzessin hinaus in den Wald und setzte sich an den Rand
des kühlen Brunnens. Und wenn sie Langeweile hatte, nahm sie eine
goldene Kugel, warf sie in die Höhe und fing sie wieder auf. Das war
ihr liebstes Spiel.

Nun trug es sich einmal zu, daß die goldene Kugel der Königstochter
nicht in die Händchen fiel, sondern auf die Erde schlug und gerade in
den Brunnen hineinrollte. Die Königstochter folgte ihr mit den Augen
nach, aber die Kugel verschwand, und der Brunnen war tief, so tief,
daß man keinen Grund sah.

Da fing die Prinzessin an zu weinen und weinte immer lauter und konnte
sich gar nicht trösten. Als sie so klagte, rief ihr plötzlich jemand
zu: »Was hast du nur, Königstochter? Du schreist ja, daß sich ein
Stein erbarmen möchte.«

Sie sah sich um, woher die Stimme käme, da erblickte sie einen Frosch,
der seinen dicken, häßlichen Kopf aus dem Wasser streckte. »Ach, du
bist's, alter Wasserpatscher«, sagte sie. »Ich weine über meine
goldene Kugel, die mir in den Brunnen hinabgefallen ist.«

»Sei still und weine nicht«, antwortete der Frosch, »ich kann wohl Rat
schaffen. Aber was gibst du mir, wenn ich dein Spielzeug wieder
heraufhole?«
```

»Was du haben willst, lieber Frosch«, sagte sie, »meine Kleider, meine Perlen und Edelsteine, auch noch die goldene Krone, die ich trage.«

Der Frosch antwortete: »Deine Kleider, deine Perlen und Edelsteine und deine goldene Krone, die mag ich nicht. Aber wenn du mich liebhaben willst und ich dein Geselle und Spielkamerad sein darf, wenn ich an deinem Tischlein neben dir sitzen, von deinem goldenen Tellerlein essen, aus deinem Becherlein trinken, in deinem Bettlein schlafen darf, dann will ich hinuntersteigen und dir die goldene Kugel heraufholen.«

»Ach, ja«, sagte sie, »ich verspreche dir alles, was du willst, wenn du mir nur die Kugel wiederbringst.« Sie dachte aber, der einfältige Frosch mag schwätzen, was er will, der sitzt doch im Wasser bei seinesgleichen und quakt und kann keines Menschen Geselle sein!

Als der Frosch das Versprechen der Königstochter erhalten hatte, tauchte er seinen Kopf unter, sank hinab, und über ein Weilchen kam er wieder heraufgerudert, hatte die Kugel im Maul und warf sie ins Gras. Die Königstochter war voll Freude, als sie ihr schönes Spielzeug wiedererblickte, hob es auf und sprang damit fort.

»Warte, warte!« rief der Frosch. »Nimm mich mit, ich kann nicht so laufen wie du!« Aber was half es ihm, daß er ihr sein Quak-quak so laut nachschrie, wie er nur konnte! Sie hörte nicht darauf, eilte nach Hause und hatte den Frosch bald vergessen.

Am andern Tag, als sie sich mit dem König und allen Hofleuten zur Tafel gesetzt hatte und eben von ihrem goldenen Tellerlein aß, da kam, plitsch platsch, plitsch platsch, etwas die Marmortreppe heraufgekrochen. Als es oben angelangt war, klopfte es an die Tür und rief. »Königstochter, jüngste, mach mir auf«

Sie lief und wollte sehen, wer draußen wäre. Als sie aber aufmachte, saß der Frosch vor der Tür. Da warf sie die Tür hastig zu, setzte sich wieder an den Tisch, und es war ihr ganz ängstlich zumute.

Der König sah wohl, daß ihr das Herz gewaltig klopfte, und sprach: »Mein Kind, was fürchtest du dich? Steht etwa ein Riese vor der Tür und will dich holen?«

»Ach, nein«, antwortete sie, »es ist kein Riese, sondern ein garstiger Frosch.«

»Was will der Frosch von dir?«

»Ach, lieber Vater, als ich gestern im Wald bei dem Brunnen saß und spielte, fiel meine goldene Kugel ins Wasser. Als ich deshalb weinte, hat sie mir der Frosch heraufgeholt. Und weil er es durchaus verlangte, versprach ich ihm, er sollte mein Spielgefährte werden. Ich dachte aber nimmermehr, daß er aus seinem Wasser käme. Nun ist er draußen und will zu mir herein.«

Da klopfte es zum zweiten Mal, und eine Stimme rief:

»Königstochter, jüngste,
Mach mir auf!
Weißt du nicht, was gestern

Du zu mir gesagt
Bei dem kühlen Brunnenwasser?
Königstochter, jüngste,
Mach mir auf!«

Da sagte der König: »Was du versprochen hast, das mußt du auch halten! Geh nur und mach ihm auf!«

Sie ging und öffnete die Tür. Da hüpfte der Frosch herein und hüpfte ihr immer nach bis zu ihrem Stuhl. Dort blieb er sitzen und rief: »Heb mich hinauf zu dir!« Sie zauderte, bis es endlich der König befahl. Als der Frosch auf dem Stuhl war, wollte er auf den Tisch, und als er da saß, sprach er: »Nun schieb mir dein goldenes Tellerlein näher, damit wir mitsammen essen können.« Der Frosch ließ sich's gut schmecken, ihr aber blieb fast jeder Bissen im Halse stecken.

Endlich sprach der Frosch: »Ich habe mich satt gegessen und bin müde. Nun trag mich in dein Kämmerlein und mach dein seidenes Bettlein zurecht!« Die Königstochter fing an zu weinen und fürchtete sich vor dem kalten Frosch, den sie sich nicht anzurühren getraute und der nun in ihrem schönen, reinen Bettlein schlafen sollte.

Der König aber wurde zornig und sprach: »Wer dir geholfen hat, als du in Not warst, den sollst du hernach nicht verachten!«

Da packte sie den Frosch mit zwei Fingern, trug ihn hinauf in ihr Kämmerlein und setzte ihn dort in eine Ecke. Als sie aber im Bette lag, kam er gekrochen und sprach: »Ich will schlafen so gut wie du. Heb mich hinauf, oder ich sag's deinem Vater!«

Da wurde sie bitterböse, holte ihn herauf und warf ihn gegen die Wand. »Nun wirst du Ruhe geben«, sagte sie, »du garstiger Frosch!« Als er aber herabfiel, war er kein Frosch mehr, sondern ein Königssohn mit schönen freundlichen Augen. Der war nun nach ihres Vaters Willen ihr lieber Geselle und Gemahl. Er erzählte ihr, er wäre von einer bösen Hexe verwünscht worden, und niemand hätte ihn aus dem Brunnen erlösen können als sie allein, und morgen wollten sie mitsammen in sein Reich gehen.

Und wirklich, am anderen Morgen kam ein Wagen herangefahren, mit acht weißen Pferden bespannt, die hatten weiße Straußfedern auf dem Kopf und gingen in goldenen Ketten. Hinten auf dem Wagen aber stand der Diener des jungen Königs, das war der treue Heinrich.

Der treue Heinrich hatte sich so gekränkt, als sein Herr in einen Frosch verwandelt worden war, daß er drei eiserne Bänder um sein Herz hatte legen lassen, damit es ihm nicht vor Weh und Traurigkeit zerspränge.

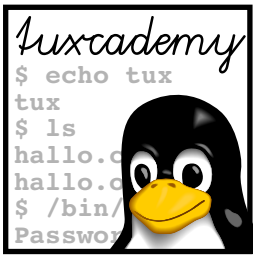
Der Wagen sollte nun den jungen König in sein Reich holen. Der treue Heinrich hob ihn und seine junge Gemahlin hinein, stellte sich wieder hinten hinauf und war voll Freude über die Erlösung seines Herrn. Als sie ein Stück des Weges gefahren waren, hörte der Königssohn, daß es hinter ihm krachte, als ob etwas zerbrochen wäre. Da drehte er sich um und rief:

»Heinrich, der Wagen bricht!«
»Nein, Herr, der Wagen nicht,

```
 Es ist ein Band von meinem Herzen,
 Das da lag in großen Schmerzen,
 Als Ihr in dem Brunnen saßt
 Und in einen Frosch verzaubert wart.«

Noch einmal und noch einmal krachte es auf dem Weg, und der Königssohn
meinte immer, der Wagen bräche. Doch es waren nur die Bänder, die vom
Herzen des treuen Heinrich absprangen, weil sein Herr nun erlöst und
glücklich war.
```

(Die Linup Front GmbH weist ausdrücklich darauf hin, dass die Autoren dieser Dokumentation jegliche Tierquälerei verurteilen.)



C

Linux-Essentials-Zertifizierung

Das *Linux Professional Institute* (LPI) ist eine herstellerunabhängige, nicht profitorientierte Organisation, die sich der Förderung des professionellen Einsatzes von Linux widmet. Ein Aspekt der Arbeit des LPI ist die Erstellung und Durchführung weltweit anerkannter, distributionsunabhängiger Zertifizierungsprüfungen beispielsweise für Linux-Systemadministratoren.

Diese Schulungsunterlage dient zur Vorbereitung auf das *Linux-Essentials*-Zertifikat. Das LPI schreibt auf seiner Web-Seite hierzu:

> Das Ziel des *Linux-Essentials*-Zertifikats ist, das Grundwissen zu definieren, das für die kompetente Nutzung eines Linux-Betriebssystems auf einem Desktop-Computersystem oder auf einem mobilen Gerät notwendig ist. Das dazugehörende *Linux-Essentials*-Programm wird Jugendlichen und denjenigen, für die Linux und Open Source neu ist, helfen und sie dabei unterstützen, den Platz von Linux und Open Source im größeren Kontext der IT-Branche zu verstehen.

Das LPI hat das für die Prüfung erforderliche Wissen in Form von **Prüfungszielen** (*objectives*) zusammengestellt und auf seinen Web-Seiten unter `http://www.lpi.org/` veröffentlicht.

Prüfungsziele

Der folgenden Tabelle können Sie entnehmen, welche Kapitel in der Schulungsunterlage den Inhalt welcher Prüfungsziele abdecken. Die Prüfungsziele selbst sind im Anschluss abgedruckt. Beachten Sie, dass die *Linux-Essentials*-Prüfungsziele nicht dazu geeignet oder vorgesehen sind, einen Einführungskurs in Linux didaktisch zu strukturieren. Aus diesem Grund orientiert diese Unterlage sich nicht an der Reihenfolge der Prüfungsziele in der Veröffentlichung des LPI, sondern weicht im Interesse einer verständlicheren und logisch so weit wie möglich aufeinander aufbauenden Präsentation davon ab.

Beachten Sie, dass die Prüfungsziele auf den Web-Seiten des LPI gelegentlich modifiziert werden und zum Beispiel zusätzliche Informationen dort erscheinen können, die es möglicherweise (noch) nicht in diese Schulungsunterlage geschafft haben. Lesen Sie zur Sicherheit die Prüfungsziele beim LPI nach.

C.1 Übersicht der Prüfungsziele

Die folgende Tabelle zeigt die Prüfungsziele der Prüfung *Linux Essentials* und die Kapitel, die diese Prüfungsziele abdecken. Die Zahlen in der rechten Spalte verweisen auf die Kapitel, die das entsprechende Material enthalten.

Nr	Gew	Titel	LXES
1.1	2	Linux Entwicklung und gängige Betriebssysteme	2
1.2	2	Wichtige Open-Source-Anwendungen	2
1.3	1	Open-Source-Software und Lizenzen verstehen	2
1.4	2	IuK-Kenntnisse und Arbeiten mit Linux	2–3, 13
2.1	2	Erste Schritte auf der Kommandozeile	4, 6, 9
2.2	2	Die Kommandozeile benutzen, um Hilfe zu finden	5
2.3	2	Verzeichnisse und Protokolldateien nutzen	6
2.4	2	Erstellen, Verschieben und Löschen von Dateien	6
3.1	2	Archivierung von Dateien auf der Kommandozeile	11
3.2	4	Suche und Entnahme von Daten aus Dateien	7–8
3.3	4	Befehle in ein Skript umwandeln	3, 9
4.1	1	Wahl eines Betriebssystems	1–2
4.2	2	Computer-Hardware verstehen	1
4.3	3	Datenspeicherung	10
4.4	2	Den Computer ins Netz einbinden	15
5.1	2	Grundlegende Sicherheit und Identifizieren von Benutzertypen	10, 13
5.2	2	Benutzer und Gruppen erstellen	13
5.3	2	Einstellungen für Dateiberechtigungen und Dateieigentum	14
5.4	1	Besondere Verzeichnisse und Dateien	6, 10, 14

C.2 Prüfungsziele für *Linux Essentials*

1.1 Linux Entwicklung und gängige Betriebssysteme

Gewicht 2

Beschreibung Kenntnisse zur Entwicklung von Linux und wichtige Distributionen.

Hauptwissensgebiete

- Open Source Philosophie
- Distributionen
- Embedded Systems

Hier ist eine auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme:

- Android
- Debian
- CentOS

1.2 Wichtige Open-Source-Anwendungen

Gewicht 2

Beschreibung Wichtige Anwendungen und deren Nutzung.

Hauptwissensgebiete

- Desktop-Anwendungen
- Server-Anwendungen
- Mobile Anwendungen
- Entwicklungssprachen
- Werkzeuge für die Paketverwaltung

Hier ist eine auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme:

- OpenOffice.org, LibreOffice, Thunderbird, Firefox
- Blender, Gimp, Audacity, ImageMagick
- Apache, MySQL, PostgreSQL
- NFS, Samba, OpenLDAP, Postfix, DNS, DHCP
- C, Java, Perl, Shell, Python, PHP

1.3 Open-Source-Software und Lizenzen verstehen

Gewicht 1

Beschreibung Open-Source-Communities und die Lizenzierung freier und Open-Source-Software.

Hauptwissensgebiete

- Lizenzen
- Free Software Foundation (FSF), Open Source Initiative (OSI)

Hier ist eine auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme:

- GPL, BSD, Creative Commons
- Free Software, Open Software, FOSS, FLOSS
- Open-Source-Geschäftsmodelle

Gut zu wissen:

- Geistiges Eigentum: Copyright, Markenzeichen und Patente
- Apache-Lizenz, Mozilla-Lizenz

1.4 IuK-Kenntnisse und Arbeiten mit Linux

Gewicht 2

Beschreibung Grundkenntnisse der Informations- und Kommunikationstechnologie (IuK) und Arbeiten mit Linux.

Hauptwissensgebiete

- Desktop Kenntnisse
- Erste Schritte auf der Kommandozeile
- Gewerbliche Nutzung von Linux, Cloud Computing und Virtualisierung

Hier ist eine auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme:

- Nutzung eines Browsers, Sicherheitsbedenken, Einstellungsoptionen, Websuche, und Speichern von Inhalten
- Terminal und Konsole
- Passworteinstellungen
- Privatsphäreeinstellungen und Werkzeuge
- Nutzung gängiger Open-Source-Anwendungen in Präsentationen und Projekten

2.1 Erste Schritte auf der Kommandozeile

Gewicht 2

Beschreibung Grundkenntnisse zur Benutzung der Linux-Kommandozeile

Hauptwissensgebiete

- Shell-Grundkenntnisse

- Kommandos eingeben und strukturieren
- Mit Optionen arbeiten
- Variable
- Globbing, Wildcard
- Quoting

Hier ist eine auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme:

- `echo`
- `history`
- Umgebungsvariable `PATH`
- `which`

Gut zu wissen:

- Variablenersetzung
- Steuerungsoperatoren `||`, `&&` und `;`

2.2 Die Kommandozeile benutzen, um Hilfe zu finden

Gewicht 2

Beschreibung Hilfekommandos benutzen und die Navigation durch die unterschiedlichen Hilfesysteme.

Hauptwissensgebiete

- Man
- Info

Hier ist eine auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme:

- `man`
- `info`
- Handbuchseiten (man pages)
- `/usr/share/doc`
- `locate`

Gut zu wissen:

- `apropos`, `whatis`, `whereis`

2.3 Verzeichnisse und Protokolldateien nutzen

Gewicht 2

Beschreibung Navigieren in Home- und System-Verzeichnissen und Protokolldateien an verschiedenen Orten.

Hauptwissensgebiete:

- Dateien, Verzeichnisse
- Versteckte Dateien und Verzeichnisse
- Home
- Absolute und relative Pfade

Hier ist eine auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme:

- Gängige Optionen für `ls`
- Rekursives Auflisten
- `cd`
- `.` und `..`
- `HOME` und `~`

2.4 Erstellen, Verschieben und Löschen von Dateien

Gewicht 2
Beschreibung Erstellen, Verschieben und löschen von Dateien und Verzeichnissen im eigenen Verzeichnis.
Hauptwissensgebiete

- Dateien und Verzeichnisse
- Beachtung von Groß- und Kleinschreibung
- Einfaches Globbing und Quoting

Hier ist eine auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme:

- `mv`, `cp`, `rm`, `touch`
- `mkdir`, `rmdir`

3.1 Archivierung von Dateien auf der Kommandozeile

Gewicht 2
Beschreibung Archivierung von Dateien im eigenen Verzeichnis.
Hauptwissensgebiete:

- Dateien, Verzeichnisse
- Archive, Komprimieren

Hier ist eine auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme:

- `tar`
- Gängige `tar`-Optionen
- `gzip`, `bzip2`
- `zip`, `unzip`

Gut zu wissen:

- Einzelne Dateien aus Archiven extrahieren

3.2 Suche und Entnahme von Daten aus Dateien

Gewicht 4
Beschreibung Suche und Entnahme von Daten aus Dateien in den Home-Verzeichnissen.
Hauptwissensgebiete

- Kommando-Pipelines
- Eingabe-/Ausgabe-Umleitung
- Wichtigste POSIX-konforme reguläre Ausdrücke (`.`, `[ ]`, `*`, `?`)

Hier ist eine auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme:

- `find`
- `grep`
- `less`
- `head`, `tail`
- `sort`
- `cut`
- `wc`

Gut zu wissen:

- Grundlegende reguläre Ausdrücke gemäß POSIX (Auszug – `[^ ]`, `^`, `$`)
- Erweiterte reguläre Ausdrücke gemäß POSIX (Auszug – `+`, `( )`, `|`)
- `xargs`

3.3 Befehle in ein Skript umwandeln

Gewicht 4
Beschreibung Wiederkehrende Befehle in einfache Skripte umwandeln.
Hauptwissensgebiete

- Grundlagen der Textbearbeitung
- Grundlagen von Shellskripten

Hier ist eine auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme:

- `/bin/sh`
- Variable
- Argumente
- `for`-Schleifen
- `echo`
- Rückgabewert
- `pico`, `nano`, `vi` (Grundlagen

Gut zu wissen:

- Bash
- `if`-, `while`- und `case`-Kommandos
- Kommandos `read`, `test` und `[`

4.1 Wahl eines Betriebssystems

Gewicht 1
Beschreibung Kenntnisse über wichtige Betriebssysteme und Linux-Distributionen.
Hauptwissensgebiete

- Unterschiede zwischen Windows, Mac und Linux
- Management des Lebenszyklus von Distributionen

Hier ist eine auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme:

- GUI versus Kommandozeile, Desktop-Konfiguration
- Wartungszyklen, Beta und Stabil

4.2 Computer-Hardware verstehen

Gewicht 2
Beschreibung Vertrautheit mit den Komponenten für den Bau von Desktop- und Server-Computern.
Hauptwissensgebiete

- Hardware

Hier ist eine auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme:

- Festplatten und Partitionen, Hauptplatinen, Prozessoren, Netzteile, optische Laufwerke, Peripheriegeräte
- Bildschirmarten
- Treiber

4.3 Datenspeicherung

Gewicht 3
Beschreibung Wo verschiedene Arten von Informationen in einen Linux-System gespeichert werden.
Hauptwissensgebiete

- Kernel
- Prozess
- `syslog`, `klog`, `dmesg`
- `/lib`, `/usr/lib`, `/etc`, `/var/log`

Hier ist eine auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme:

- Programme, Bibliotheken, Pakete und Paketdatenbanken, Systemeinstellungen
- Prozesse und Prozesstabellen, Speicheradressen, Systembenachrichtigungen und Protokollierung

4.4 Den Computer ins Netz einbinden

Gewicht 2
Beschreibung Abfragen wesentlicher Netzwerkeinstellungen und Bestimmen der Grundvoraussetzungen für einen Computer in einem Local Area Network (LAN).
Hauptwissensgebiete

- Internet, Netzwerk, Router
- Domain Name Service
- Netzwerkeinstellungen

Hier ist eine auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme:

- `route`
- `resolv.conf`
- IPv4, IPv6
- `ifconfig`
- `netstat`
- `ping`

Gut zu wissen:

- `ssh`
- `dig`

5.1 Grundlegende Sicherheit und Identifizieren von Benutzertypen

Gewicht 2
Beschreibung Verschiedene Benutzertypen in einem Linux-System.
Hauptwissensgebiete

- Root und unprivilegierte Nutzer
- Systemnutzer

Hier ist eine auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme:

- `/etc/passwd`, `/etc/group`
- `id`, `who`, `w`
- `sudo`

Gut zu wissen:

- `su`

5.2 Benutzer und Gruppen erstellen

Gewicht 2
Beschreibung Benutzer und Gruppen in einem Linux-System erstellen.
Hauptwissensgebiete

- Befehle zu Benutzern und Gruppen
- User-IDs

Hier ist eine auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme:

- `/etc/passwd`, `/etc/shadow`, `/etc/group`
- `id`, `last`
- `useradd`, `groupadd`
- `passwd`

Gut zu wissen:

- `usermod`, `userdel`
- `groupmod`, `groupdel`

5.3 Einstellungen für Dateiberechtigungen und Dateieigentum

Gewicht 2
Beschreibung Verstehen und Bearbeiten von Dateiberechtigungen und Eigentumseinstellungen.
Hauptwissensgebiete

- Datei- und Verzeichnisrechte und Dateieigentümer

Hier ist eine auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme:

- `ls -l`
- `chmod`, `chown`

Gut zu wissen:

- `chgrp`

5.4 Besondere Verzeichnisse und Dateien

Gewicht 1
Beschreibung Besondere Verzeichnisse und Dateien in einem Linuxsystem, einschließlich besonderer Zugriffsrechte.
Hauptwissensgebiete:

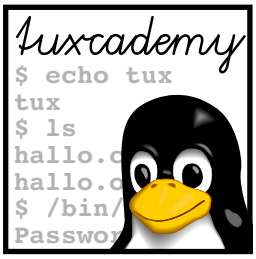
- Systemdateien, Bibliotheken
- Symbolische Links

Hier ist eine auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme:

- `/etc`, `/var`
- `/tmp`, `/var/tmp` und Sticky Bit
- `ls -d`
- `ln -s`

Gut zu wissen:

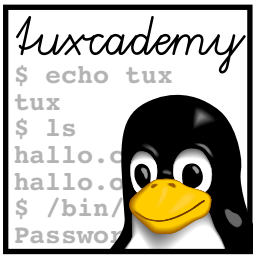
- Hardlinks
- Setuid/Setgid



D

Kommando-Index

Dieser Anhang fasst alle im Text erklärten Kommandos zusammen und verweist auf deren Dokumentation sowie die Stellen im Text, wo die Kommandos eingeführt werden.



Index

Dieser Index verweist auf die wichtigsten Stichwörter in der Schulungsunterlage. Besonders wichtige Stellen für die einzelnen Stichwörter sind durch **fette** Seitenzahlen gekennzeichnet. Sortiert wird nur nach den Buchstaben im Indexeintrag; „`~/.bashrc`“ wird also unter „B“ eingeordnet.