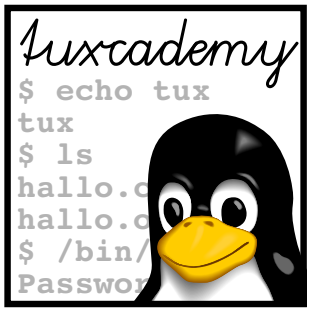
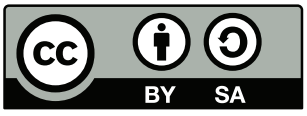
Version 4.0

Linux-Grundlagen

für Anwender und Administratoren





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

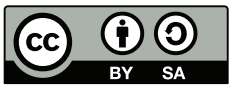
Diese Schulungsunterlage ist inhaltlich und didaktisch auf Inhalte der Zertifizierungsprüfung LPI-101 (LPIC-1, Version 4.0) des Linux Professional Institute *abgestimmt. Weitere Details stehen in Anhang C.*

Das Linux Professional Institute *empfiehlt keine speziellen Prüfungsvorbereitungsmaterialien oder -techniken – wenden Sie sich für Details an* `info@lpi.org`.

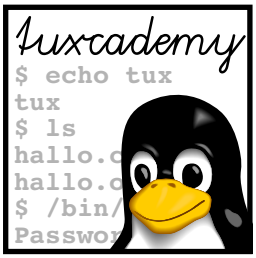
Das tuxcademy-Projekt bietet hochwertige frei verfügbare Schulungsunterlagen zu Linux- und Open-Source-Themen – zum Selbststudium, für Schule, Hochschule, Weiterbildung und Beruf.
Besuchen Sie `https://www.tuxcademy.org/`! Für Fragen und Anregungen stehen wir Ihnen gerne zur Verfügung.

Linux-Grundlagen für Anwender und Administratoren

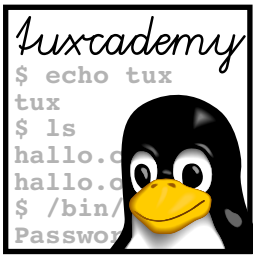
Revision: `grd1:d0ad6b9d16863aeb:2015-08-04`
`grd1:a13e1ba7ab759bab:2015-08-04` 1–11, B–C
`grd1:HnLafg42pqEurl5k7wWY8u`



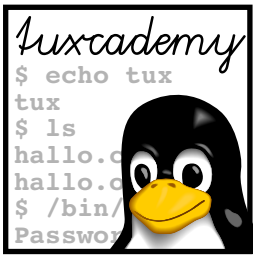
Autoren: Tobias Elsner, Anselm Lingnau
Technische Redaktion: Anselm Lingnau ⟨*anselm@tuxcademy.org*⟩
Gesetzt in Palatino, Optima und DejaVu Sans Mono



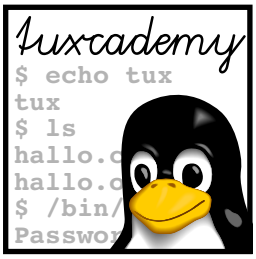
Inhalt



Tabellenverzeichnis



Abbildungsverzeichnis



Vorwort

Dieser Kurs ist eine Einführung in die Anwendung von Linux. Dem Teilnehmer soll nicht nur die Benutzung dieses Betriebssystems nach Anleitung, sondern als Basis für spätere Administrationsaufgaben auch ein grundlegendes Verständnis des Systems vermittelt werden.

Der Kurs wendet sich an Benutzer von Computern, die schon grundlegende EDV-Anwenderkenntnisse haben und bereits mit einer grafischen Benutzeroberfläche vertraut sind. Kenntnisse anderer Betriebssysteme sind zwar hilfreich, werden aber nicht vorausgesetzt. Sowohl ambitionierte Anwender als auch Administratoren lernen in diesem Kurs den Umgang mit dem Linux-Betriebssystem.

Nach Abschluss des Kurses sind die Teilnehmer in der Lage, das Betriebssystem Linux auf elementarer Ebene zu nutzen. Die Anwendung der grafischen Oberfläche und üblicher Kommandozeilen-Programme ist in Grundzügen bekannt und muss gegebenenfalls noch geübt werden. Der erfolgreiche Abschluss dieses Kurses oder vergleichbare Kenntnisse sind Voraussetzung für den erfolgreichen Besuch weiterer Linux-Kurse und für eine Zertifizierung beim *Linux Professional Institute*.

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!

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.

Übungsaufgaben

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.

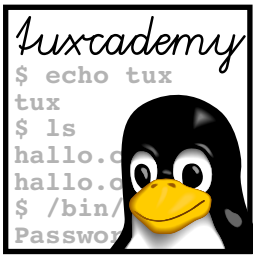
Wir sind bemüht, diese Schulungsunterlage möglichst aktuell, vollständig und fehlerfrei zu gestalten. Trotzdem kann es passieren, dass sich Probleme oder Ungenauigkeiten einschleichen. Wenn Sie etwas bemerken, was Sie für verbesserungsfähig halten, dann lassen Sie es uns wissen, etwa indem Sie eine elektronische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

LPIC-1-Zertifizierung

Diese Unterlage ist Teil eines Kurskonzepts zur Vorbereitung auf die LPIC-1-Zertifizierung. Anhang C enthält hierzu mehr Informationen.



1

Einführung

Inhalt

Lernziele

- Linux mit seinen Eigenschaften und seiner Entstehungsgeschichte kennen
- Verstehen, was Kernel und Distributionen sind
- Die Begriffe »GPL«, »freie Software« und »Open Source« einordnen können

Vorkenntnisse

- Kenntnisse eines anderen Betriebssystems sind hilfreich, um Parallelen und Unterschiede erkennen zu können

1.1 Was ist Linux?

Linux ist ein Betriebssystem. Als Betriebssystem stellt es die elementaren Funktionen zum Betrieb eines Rechners zur Verfügung. Anwendungsprogramme bauen auf dem Betriebssystem auf. Es bildet die Schnittstelle zwischen der Hardware und den Anwendungsprogrammen, aber auch die Schnittstelle zwischen Hardware und Mensch (Benutzer). Ohne ein Betriebssystem ist der Computer nicht in der Lage, unsere Eingaben zu »verstehen« bzw. zu verarbeiten.

Die verschiedenen Betriebssysteme unterscheiden sich in der Weise, wie sie die oben genannten Aufgaben ausführen. Linux ist in seiner Funktionalität und seiner Bedienung dem Betriebssystem Unix nachempfunden.

1.2 Die Geschichte von Linux

Die Entstehungsgeschichte von Linux ist etwas Besonderes in der Computer-Welt. Während die meisten anderen Betriebssysteme kommerzielle Produkte von Firmen sind, wurde Linux von einem Studenten als Hobbyprojekt ins Leben gerufen. Inzwischen arbeiten weltweit Hunderte von Profis und Enthusiasten daran mit – von Hobbyprogrammierern und Informatikstudenten bis zu Betriebssystemexperten, die von renommierten Computerfirmen dafür bezahlt werden, Linux weiterzuentwicken. Grundlage für die Existenz eines solchen Projekts ist das Internet: Die Linux-Entwickler nutzen intensiv Dienste wie E-Mail, verteilte Revisionskontrolle und das World Wide Web und haben so das Betriebssystem Linux zu dem gemacht, was es heute ist. Linux ist also das Ergebnis einer internationalen Zusammenarbeit über Länder- und Firmengrenzen hinweg, nach wie vor geleitet von Linus Torvalds, dem ursprünglichen Autor.

Um den Hintergrund von Linux erklären zu können, müssen wir etwas weiter ausholen: Unix, das Vorbild von Linux, entstand ab 1969. Es wurde von Ken
Bell Laboratories Thompson und seinen Kollegen bei den Bell Laboratories (dem Forschungsinstitut des US-amerikanischen Telefonmagnaten AT&T) entwickelt[1]. Es verbreitete sich rasch vor allem in Universitäten, da die Bell Labs die Quellen und Dokumentation zum Selbstkostenpreis abgaben (AT&T durfte aufgrund kartellrechtlicher Einschränkungen keine Software verkaufen). Unix war zunächst ein Betriebssystem für die PDP-11-Rechner von Digital Equipment, wurde aber im Laufe der 1970er Jahre auf andere Plattformen portiert – ein vergleichsweise einfaches Unterfangen, da die Unix-Software, einschließlich des Betriebssystemkerns, zum allergrößten Teil in der von Dennis Ritchie speziell für diesen Zweck erfundenen Program-
C miersprache **C** geschrieben war. Die vielleicht wichtigste Portierung war die auf
VAX die PDP-11-Nachfolgeplattform VAX an der University of California in Berkeley, die als »BSD« (kurz für *Berkeley Software Distribution*) in Umlauf kam. Im Laufe der Zeit entwickelten verschiedene Computerhersteller unterschiedliche Unix-Varianten teils auf der Basis des AT&T-Codes, teils auf der Basis von BSD (z. B. Sinix von Siemens, Xenix von Microsoft (!), SunOS von Sun Microsystems, HP/UX von Hewlett-Packard oder AIX von IBM). Auch AT&T selbst durfte schließlich Unix
System V verkaufen – die kommerziellen Versionen System III und (später) System V. Das führte zu einer ziemlich unübersichtlichen Fülle verschiedenener Unix-Produkte. Es kam nie wirklich zu einer Standardisierung, aber man kann in etwa zwischen BSD- und System-V-nahen Unix-Varianten unterscheiden. Zum größten Teil wur-
SVR4 den die BSD- und die System-V-Entwicklungslinien in »System V Release 4« zusammengeführt, das die wesentlichen Eigenschaften beider Strömungen aufweist.

Die allerersten Teile von Linux wurden 1991 von Linus Torvalds, einem damals 21-jährigen Studenten aus Helsinki, entwickelt, als dieser die Möglichkeiten des Intel-386-Prozessors in seinem neuen PC genauer untersuchte. Nach einigen Mo-

[1] Der Name »Unix« ist ein Wortspiel mit »Multics«, dem Betriebssystem, an dem Ken Thompson und seine Kollegen vorher mitgearbeitet hatten. Das frühe Unix war viel simpler als Multics. Wie es dazu kam, den Namen mit »x« zu schreiben, ist nicht mehr bekannt.

Bild 1.1: Ken Thompson (sitzend) und Dennis Ritchie (stehend) an einer PDP-11, ca. 1972. (Abdruck mit freundlicher Genehmigung von Lucent Technologies.)

naten war aus den Assemblerstudien ein kleiner, lauffähiger Betriebssystemkern entstanden, der in einem Minix-System eingesetzt werden konnte – Minix war ein kleines Unix-artiges Betriebssystem, das der Informatikprofessor Andrew S. Tanenbaum an der Freien Universität Amsterdam für seine Studenten geschrieben hatte. Das frühe Linux hatte ähnliche Eigenschaften wie ein Unix-System, enthielt aber keinen Unix-Quellcode. Linus Torvalds gab den Programmcode über das Internet frei, und die Idee wurde mit Begeisterung von vielen Programmierern aufgegriffen und weiter entwickelt. Die im Januar 1992 herausgegebene Version 0.12 war bereits ein stabil laufender Betriebssystemkern. Es gab – dank Minix – den `gcc` (GNU C-Compiler), die `bash`, `emacs` und viele der anderen GNU-Hilfsprogramme. Dieses Betriebssystem wurde über anonymes FTP weltweit verteilt. Die Zahl der Programmierer, Tester und Unterstützer wuchs rasend schnell. Das ermöglichte in kurzer Zeit Fortschritte, von denen mächtige Softwareunternehmen nur träumen können. Innerhalb weniger Monate wurde aus dem Minikernel ein ausgewachsenes Betriebssystem mit ziemlich vollständiger (wenn auch simpler) Unix-Funktionalität.

Minix

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 PDA 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.

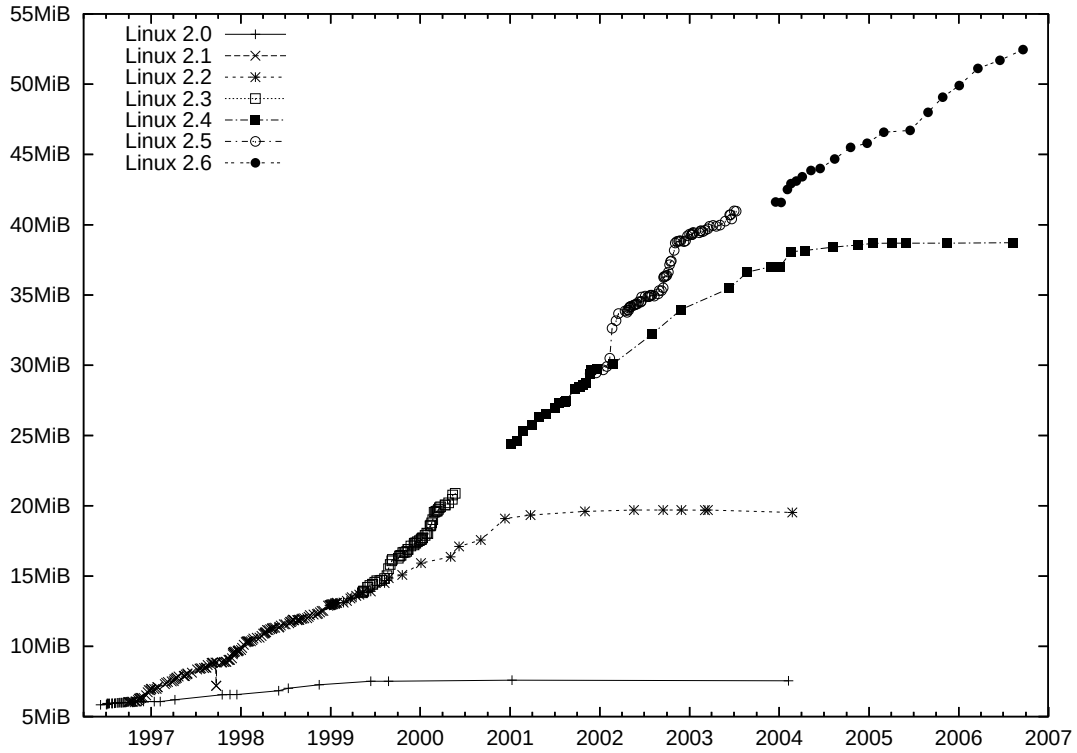


Bild 1.2: Die Weiterentwicklung von Linux, gemessen an der Größe von `linux-*.tar.gz`. Jede Marke entspricht einer Linux-Version. In den gut 10 Jahren von Linux 2.0 bis Linux 2.6.18 hat der Umfang des komprimierten Linux-Quellcodes sich nahezu verzehnfacht.

Übungen

1.1 [4] Suchen Sie im Internet nach der berühmt-berüchtigten Diskussion zwischen Andrew S. Tanenbaum und Linus Torvalds, in der Tanenbaum sagt, Linus Torvalds wäre mit etwas wie Linux bei ihm im Praktikum durchgefallen. Was halten Sie davon?

1.2 [2] Welche Versionsnummer hat der älteste Linux-Kern-Quellcode, den Sie noch finden können?

1.3 Freie Software, »Open Source« und die GPL

GPL
Free Software Foundation
Freie Software
Open Source

Linux steht seit Beginn der Entwicklung unter der *GNU General Public License* (**GPL**) der *Free Software Foundation* (FSF). Die FSF wurde von Richard M. Stallman, dem Autor des Editors Emacs und anderer wichtiger Programme, mit dem Ziel gegründet, qualitativ hochwertige Software »frei« verfügbar zu machen – in dem Sinne, dass Benutzer »frei« sind, sie anzuschauen, zu ändern und im Original oder geändert weiterzugeben, nicht notwendigerweise in dem Sinne, dass sie nichts kostet[2]. Insbesondere ging es ihm um ein frei verfügbares Unix-artiges Betriebssystem, daher »GNU« als (rekursive) Abkürzung für *"GNU's Not Unix"*. Die wesentliche Aussage der GPL liegt darin, dass in ihrem Sinne geschützte Software zwar jederzeit verändert und auch verkauft werden darf, aber immer zusammen mit dem (gegebenenfalls veränderten) Quellcode weitergegeben werden muss – deshalb auch *Open Source* – und der Empfänger dieselben Rechte der Veränderung und Weitergabe erhalten muss. Es hat darum keinen großen Sinn, GPL-Software

[2] Die FSF sagt "free as in speech, not as in beer"

»pro Rechner« zu verkaufen, da der Empfänger das Recht bekommt, die Software so oft zu kopieren und zu installieren, wie er mag. (Was man allerdings durchaus »pro Rechner« verkaufen darf, ist Unterstützung und Service für die GPL-Software.) Neue Software, die durch Erweiterung oder Veränderung von GPL-Software entsteht, muss als »abgeleitetes Werk« ebenfalls unter die GPL gestellt werden.

Die GPL bezieht sich also auf den Vertrieb der Software, nicht den Gebrauch, und gibt dem Empfänger der Software Rechte, die er sonst überhaupt nicht hätte – zum Beispiel das Recht zur Vervielfältigung und Weitergabe der Software, das nach dem Gesetz dem »Urheber« vorbehalten bleibt (darum »Urheberrecht«). Sie unterscheidet sich deshalb deutlich von den *End User License Agreements* »proprietärer« Software, die dem Benutzer normalerweise Rechte zu *nehmen* versuchen. (Manche EULAs wollen dem Empfänger der Software zum Beispiel verbieten, in der Öffentlichkeit schlecht – oder überhaupt – über das Produkt zu reden.)

EULAs

Die GPL ist eine *Lizenz*, kein Vertrag, da sie dem Empfänger der Software einseitig etwas erlaubt (wenngleich unter Bedingungen). Der Empfänger der Software muss die GPL nicht ausdrücklich »akzeptieren«. Bei den gängigen EULAs handelt es sich dagegen um *Verträge*, da der Empfänger der Software im Austausch für das Recht, die Software benutzen zu dürfen, auf gewisse Rechte verzichten soll. Aus diesem Grund müssen EULAs auch ausdrücklich akzeptiert werden. Die rechtlichen Anforderungen hierfür liegen recht hoch – etwa müssen in Deutschland einem Software-Käufer die EULA-Bedingungen vor dem Kauf bekannt sein, damit sie wirksamer Bestandteil des Kaufvertrags werden können. Da die GPL die Rechte des Käufers (insbesondere was den Gebrauch der Software angeht) in keiner Weise dem gegenüber einschränkt, was er bei irgendeinem Sachkauf zu erwarten hätte, gelten diese Anforderungen nicht für die GPL; die zusätzlichen Rechte, die der Käufer unter der GPL erhält, sind eine Art Sonderbonus.

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.

GPLv3

Nicht verwechseln sollten Sie GPL-Software auch mit »Public-Domain-Software«. Letztere gehört niemandem, jeder darf damit machen, was er möchte (dieses Konzept existiert im deutschen Recht übrigens nur als »Gemeinfreiheit«, die eintritt, wenn der Urheber länger als 70 Jahre tot ist – im Computerbereich ist das noch nicht relevant). Die Urheberrechte an GPL-Software liegen aber in der Regel auch weiterhin beim Entwickler oder den Entwicklern, und die GPL sagt sehr deutlich, was mit der Software gemacht werden darf und was nicht.

Public Domain

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.

Andere »freie« Lizenzen

Es gibt andere »freie« Software-Lizenzen, die dem Empfänger der Software ähnliche Rechte einräumen, etwa die »BSD-Lizenz«, die die Integration von entsprechend lizenzierter Software in eigene, nicht freie Produkte gestattet. Die GPL gilt aber als konsequenteste der freien Lizenzen in dem Sinne, dass sie 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 betreffenden Programme freigegeben. 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.

Freiheits-Kriterien
Debian Free Software Guidelines

Wann gilt eine Software als »frei« oder »Open Source«? Dafür gibt es keine verbindlichen Regeln, aber weithin anerkannt sind die *Debian Free Software Guidelines* [DFSG]. Die FSF fasst ihre Kriterien in den *Four Freedoms* zusammen, die für eine freie Software gegeben sein müssen:

- Man muss das Programm für jeden beliebigen Zweck benutzen dürfen (Freiheit 0)
- Man muss studieren können, wie das Programm funktioniert, und es an seine 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. Gängige Freie-Software-Lizenzen wie die GPL und die BSD-Lizenz erfüllen diese Freiheiten natürlich.

Mehrfach-Lizenzierung

Im übrigen darf der Urheber einer Software diese durchaus unter verschiedenen Lizenzen verteilen, etwa alternativ unter der GPL und einer »kommerziellen« Lizenz, die den Empfänger von den Anforderungen der GPL, etwa der Freigabe

von Quellcode für abgeleitete Werke, freistellt. Auf diese Weise können zum Beispiel Privatanwender und Autoren freier Software kostenlos in den Genuss einer leistungsfähigen Unterprogrammbibliothek wie zum Beispiel des Grafikpakets »Qt« (veröffentlicht von Qt Software – ehemals Troll Tech –, einem Tochterunternehmen von Nokia) kommen, während Firmen, die ihren eigenen Programmcode nicht frei zur Verfügung stellen wollen, sich von der GPL »freikaufen« müssen.

Übungen

1.3 [!2] 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.

1.4 [4] Manche Softwarelizenzen verlangen, dass bei Änderungen am Inhalt einer Datei aus der Softwaredistribution die Datei einen neuen Namen bekommen muss. Ist so lizenzierte Software »frei« gemäß der DFSG? Was halten Sie davon?

1.4 Linux – Der Kernel

Betriebssystemkern

Genaugenommen bezeichnet der Begriff »Linux« nur den Betriebssystemkern, der die eigentlichen Aufgaben des Betriebssystems übernimmt. Den Betriebssystemkern nennt man (neudeutsch) auch *Kernel*. Er erledigt elementare Aufgaben wie Speicher- und Prozessverwaltung und die Steuerung der Hardware. Anwenderprogramme müssen sich an den Kernel wenden, wenn sie zum Beispiel auf Dateien auf der Platte zugreifen wollen. Der Kernel prüft solche Vorgänge und kann auf diese Weise dafür sorgen, dass niemand unerlaubten Zugriff auf fremde Dateien erhält. Außerdem kümmert sich der Kernel darum, dass alle Prozesse im System (und damit alle Benutzer) den ihnen jeweils zustehenden Anteil an der verfügbaren Rechenzeit erhalten.

Versionen

Natürlich gibt es nicht nur einen Linux-Kernel, sondern es existieren viele verschiedene Versionen. Bis zur Kernel-Version 2.6 wurde zwischen stabilen »Anwender-Versionen« und instabilen »Entwickler-Versionen« wie folgt unterschieden:

stabile Versionen

- In Versionsnummern wie 1.*x*.*y* oder 2.*x*.*y* bezeichnet ein gerades *x* stabile Versionen. In stabilen Versionen sollen keine durchgreifenden Änderungen gemacht werden; es sollen vor allem Fehlerkorrekturen vorgenommen werden, und ab und zu werden Treiber für neu herausgekommene Hardware oder andere sehr wichtige Eigenschaften neu hinzugefügt, die aus den Entwicklungs-Kernels »zurückportiert« wurden.

Entwicklungs-Versionen

- Die Versionen mit ungeradem *x* sind Entwicklungs-Versionen, die nicht für den Produktiveinsatz geeignet sind. Sie enthalten oft unzureichend getesteten Code und sind eigentlich für die Leute gedacht, die sich aktiv an der Entwicklung von Linux beteiligen wollen. Da Linux ständig weiter entwickelt wird, gibt es auch ständig neue Kernelversionen. Die Änderungen betreffen zumeist Anpassungen an neue Hardware oder die Optimierung verschiedener Subsysteme, manchmal auch komplett neue Erweiterungen.

Kernel 2.6

In Kernel 2.6 hat sich die Vorgehensweise geändert: Statt wie bisher nach einer mehr oder kurzen Stabilisierungsphase die Version 2.7 für Neuentwicklungen anzufangen, haben Linus Torvalds und die anderen Kernel-Entwickler sich entschieden, die Weiterentwicklung von Linux näher an der Endanwender-Version zu halten. Dadurch soll die Divergenz von Entwickler- und stabilen Versionen vermieden werden, wie sie sich zum Beispiel im Vorfeld der Veröffentlichung von Linux 2.6 zu einem großen Problem entwickelt hatte – vor allem weil Firmen wie SUSE oder Red Hat sich viel Mühe gemacht hatten, interessante Eigenschaften der Entwicklerversion 2.5 in ihre Versionen der 2.4-Kernels zurück zu übertragen, bis zum Beispiel ein SUSE-2.4.19-Kernel Aberhunderte von Unterschieden zum »offiziellen« Linux 2.4.19 aufwies.

Die aktuelle Methode besteht darin, vorgeschlagene Änderungen und Erweiterungen in einer neuen Version des Kernels »probezufahren«, die dann in einem kürzeren Zeitraum für »stabil« erklärt wird. Beispielsweise folgt auf die Version 2.6.37 eine Entwicklungsphase, in der Linus Torvalds Erweiterungen und Änderungen für die Version 2.6.38 entgegennimmt. Andere Kernelentwickler (oder wer sonst Lust darauf hat) haben Zugriff auf Linus' interne Entwicklerversion, die, wenn sie vernünftig genug aussieht, als »Release-Kandidat« 2.6.38-rc1 herausgegeben wird. Damit beginnt eine Stabilisierungsphase, in der dieser Release-Kandidat von mehr Leuten getestet wird. Reparaturen führen zu weiteren Release-Kandidaten, bis die neue Version stabil genug aussieht, dass Linus Torvalds sie zur Version 2.6.38 erklären kann. Danach schließt sich eine Entwicklungsphase für 2.6.39 an und so weiter.

Release-Kandidat

Parallel zu Linus Torvalds' »offizieller« Version wartet Andrew Morton eine experimentellere Version, den sogenannten »`-mm`-Baum«. Hierin werden größere und durchgreifendere Änderungen getestet, bis sie reif dafür sind, von Linus in den offiziellen Kernel aufgenommen zu werden.

`-mm`-Baum

Einige andere Entwickler bemühen sich um die Wartung der »stabilen« Kernels. Es könnte etwa Kernels mit den Versionsnummern 2.6.38.1, 2.6.38.2, … geben, in denen jeweils nur kleine und überschaubare Änderungen gemacht werden, etwa um gravierende Fehler oder Sicherheitslücken zu reparieren und Linux-Distributoren die Gelegenheit zu geben, auf längerfristig unterstützte Kernel-Versionen zurückgreifen zu können.

Version 3.0

Am 21. Juli 2011 gab Linus Torvalds offiziell die Version 3.0 des Linux-Kerns frei. Eigentlich hätte das die Version 2.6.40 sein müssen, aber er wollte das Schema für die Versionsnummern vereinfachen. Die »stabilen« Kernels auf der Basis von 3.0 werden entsprechend mit 3.0.1, 3.0.2, … bezeichnet, und in Torvalds' Entwicklungslinie folgt dann 3.1-rc1 usw. bis 3.1 und so fort.

Linus Torvalds legte sehr großen Wert auf die Feststellung, dass es zwischen den Kernels 2.6.39 und 3.0 keine großen Schritte bei der Funktionalität gab – jedenfalls nicht mehr als zwischen irgend zwei anderen aufeinanderfolgenden Kernels der 2.6-Serie –, sondern dass es sich nur um eine Umnummerierung handelte. Die Idee des zwanzigjährigen Linux-Jubiläums wurde ins Spiel gebracht.

»offizielle« Kernels

Quellcode für »offizielle« Kernels können Sie im Internet von `ftp.kernel.org` bekommen. Allerdings setzen die wenigsten Linux-Distributoren Originalkernels ein. Die Kernels der gängigen Distributionen sind zumeist mehr oder weniger umfassend modifiziert, etwa indem weitere Treiber hinzugefügt oder Eigenschaften integriert werden, die von der Distribution erwünscht, aber nicht Teil des Standard-Kernels sind. Der Kernel des *SUSE Linux Enterprise Servers 8* zum Beispiel enthielt angeblich rund 800 Modifikationen gegenüber dem Standardkernel. (Die Änderungen im Linux-Entwicklungsprozess sollten dazu geführt haben, dass die Differenzen heutzutage nicht mehr ganz so krass sind.)

Distributions-Kernels

Kernel-Struktur

Die meisten Kernel sind heute modular aufgebaut. Frühere Kernel waren das

nicht, das heißt, sie bestanden aus einem einzigen Stück Code, das sämtliche Aufgaben erfüllte, etwa die Unterstützung bestimmter Hardware. Wollte man neue Hardware zum System hinzufügen oder eine neue Eigenschaft, etwa ein anderes Dateisystem, verwenden, musste man sich aus den Quellen einen neuen Kernel übersetzen, ein sehr zeitintensiver Vorgang. Um das zu umgehen, wurde der Kernel mit der Fähigkeit ausgestattet, zusätzliche Eigenschaften in Form von Modulen einzubinden.

Module

Module sind Programmteile, die dynamisch (also zur Laufzeit des Kernels) hinzugeladen und auch wieder entfernt werden können. Möchten Sie beispielsweise einen neuen Netzwerk-Adapter verwenden, müssen Sie keinen neuen Kernel kompilieren, sondern lediglich ein neues Kernelmodul hinzuladen. Moderne Linux-Distributionen unterstützen eine automatische Hardwareerkennung, die die Eigenschaften des Systems analysiert und die richtigen Module findet und konfiguriert.

Hardwareerkennung

Übungen

1.5 [1] Welche Versionsnummer hat der aktuelle stabile Linux-Kern? Der aktuelle Entwicklerkern? Welche Linux-Kernversionen werden noch gewartet?

1.5 Die Eigenschaften von Linux

Als moderner Betriebssystemkern hat Linux eine Reihe von Eigenschaften, die zum Teil heute zum »Industriestandard« gehören (also auch bei ähnlichen Systemen in äquivalenter Form anzutreffen sind) und zum Teil Alleinstellungsmerkmale bilden.

Prozessoren

- Linux unterstützt eine große Auswahl von Prozessoren und Rechnerarchitekturen, von Mobiltelefonen (das extrem erfolgreiche Betriebssystem »Android« von Google basiert wie einige andere Systeme in diesem Bereich auf Linux) über PDAs und Tablet-Computer, alle möglichen Sorten von neuen und alten PC-artigen Rechnern, Serversystemen unterschiedlicher Art bis hin zu den größten Großrechnern (in der Rangliste der schnellsten Rechner der Welt läuft die weit überwiegende Mehrheit unter Linux).

  Ein großer Vorteil von Linux im Mobilbereich ist, dass es im Gegensatz zu Microsoft Windows die stromsparenden und leistungsfähigen ARM-Prozessoren unterstützt, auf denen die meisten mobilen Geräte aufbauen. Mit »Windows RT« hat Microsoft 2012 eine auf ARM laufende, teilweise zur Intel-Version kompatible Version von Windows 8 veröffentlicht, die im Markt aber nicht besonders gut ankam.

Hardware

- Von allen aktuell vorhandenen Betriebssystemen unterstützt Linux das breiteste Spektrum an Hardware. Zwar stehen für manche der allerneuesten Komponenten nicht sofort Treiber zur Verfügung, aber auf der anderen Seite arbeitet Linux noch mit Geräten zusammen, die Systeme wie Windows längst hinter sich gelassen haben. Ihre Investitionen in Drucker, Scanner, Grafikkarten und ähnliches werden also optimal geschützt.

Multitasking

- Linux unterstützt »präemptives Multitasking«, das heißt, mehrere Prozesse laufen – scheinbar oder bei Systemen mit mehr als einer CPU auch tatsächlich – zur selben Zeit. Dabei können diese Prozesse sich nicht in die Quere kommen oder einander ungebührlich blockieren; der Systemkern sorgt dafür, dass jeder Prozess gemäß seiner Priorität Rechenzeit zugeteilt bekommt.

  Heute ist das nichts Besonderes mehr; als Linux neu war, war es noch eher bemerkenswert.

Bei entsprechend sorgfältig konfigurierten Systemen reicht das bis in den Echtzeitbereich, und es gibt Linux-Varianten, die tatsächlich für die Steuerung von Industrieanlagen eingesetzt werden, wo »harte« Echtzeitfähigkeit, also garantierte schnelle Antwortzeiten auf extern eintretende Ereignisse, gefordert ist.

- Linux unterstützt mehrere Benutzer auf demselben Rechner, sogar gleichzeitig (über das Netz, über seriell angeschlossene Terminals oder auch mehrere Bildschirme, Tastaturen und Mäuse am selben Rechner). Für jeden Benutzer können getrennte Zugriffsrechte vergeben werden. mehrere Benutzer
- Linux kann problemlos parallel zu anderen Betriebssystemen auf demselben Rechner installiert werden, so dass man abwechselnd Linux oder ein anderes System starten kann. Über »Virtualisierung« kann ein Linux-System in unabhängige Teile aufgeteilt werden, die von aussen wie eigenständige Rechner aussehen und selbst unter Linux oder anderen Betriebssystemen laufen. Hierfür stehen verschiedene freie oder auch kommerzielle Lösungen zur Verfügung. Virtualisierung
- Linux nutzt die verfügbare Hardware effizient aus. Die heute üblichen Zweikern-Prozessoren werden genauso mit Arbeit versorgt wie die 4096 Prozessorkerne eines SGI-Altix-Servers. Linux läßt keinen Arbeitsspeicher (RAM) brachliegen, sondern benutzt ihn als Plattencache; umgekehrt wird der verfügbare Arbeitsspeicher sinnvoll eingesetzt, um Arbeitslasten zu bewältigen, die deutlich größer sind als das RAM im Rechner. Effizienz
- Linux ist auf Quellcodeebene kompatibel zu POSIX, System V und BSD und erlaubt so die Nutzung fast aller im Quellcode verfügbaren Unix-Programme. POSIX, System V und BSD
- Linux verfügt nicht nur über leistungsfähige eigene Dateisysteme mit Eigenschaften wie Journaling, Verschlüsselung und Logical Volume Management, sondern gestattet auch den Zugriff auf Dateisysteme zahlreicher anderer Betriebssysteme (etwa die FAT-, VFAT- und NTFS-Dateisysteme von Microsoft Windows) entweder auf lokalen Platten oder via Netzwerk auf entfernten Servern. Linux selbst kann als Dateiserver in Linux-, Unix- und Windows-Netzen fungieren. Dateisysteme
- Der TCP/IP-Stack von Linux ist anerkanntermaßen mit der leistungsfähigste in der Industrie (was daran liegt, dass ein großer Teil der Forschung und Entwicklung in diesem Bereich auf der Basis von Linux erbracht wird). Er unterstützt IPv4 und IPv6 und alle wichtigen Optionen und Protokolle. TCP/IP-Stack
- Linux bietet leistungsfähige und elegante Grafikumgebungen für die tägliche Arbeit und mit X11 ein sehr verbreitetes netzwerktransparentes Grafik-Basissystem. Auf den gängigen Grafikkarten wird 3D-beschleunigte Grafik unterstützt. Grafikumgebungen
- Alle wichtigen »Produktivitätsanwendungen« stehen zur Verfügung – Office-Programme, Web-Browser, Programme zum Zugriff auf elektronische Post und andere Kommunikationsmedien, Multimedia-Programme, Entwicklungsumgebungen für die verschiedensten Programmiersprachen und noch vieles mehr. Die meisten dieser Programme werden ohne Zusatzkosten mitgeliefert oder lassen sich problemlos und günstig über das Internet beziehen. Dasselbe gilt für Server für alle wichtigen Internet-Protokolle und für unterhaltsame Spiele. Software

Die Flexibilität von Linux bewirkt, dass das System nicht nur auf allen möglichen Rechnern der PC-Klasse eingesetzt werden kann (auch »alte Möhren«, auf denen kein aktuelles Windows mehr läuft, können noch im Kinderzimmer oder als Dateiserver, Router oder Mailserver gute Dienste leisten), sondern sich auch im »Embedded-Bereich«, also für fertige Geräte der Netzinfrastruktur oder Un- Embedded-Bereich

terhaltungselektronik, immer weiter verbreitet. Sie finden Linux zum Beispiel in der FRITZ!Box von AVM und ähnlichen WLAN-, DSL- und Telefoniegeräten, in diversen Set-Top-Boxen für das digitale Fernsehen, in digitalen Videorecordern, Digitalkameras, Kopierern und vielen anderen Geräten. Sogar auf Pfandflaschen-Automaten im Supermarkt hat der Autor dieser Zeilen schon Linux booten gesehen. Oft wird das nicht an die große Glocke gehängt, aber die Hersteller schätzen neben der Leistung und Bequemlichkeit von Linux an sich auch die Tatsache, dass bei Linux keine Lizenzkosten »pro verkauftem Gerät« anfallen wie bei vergleichbaren Betriebssystemen.

Sicherheitslücken

Ein weiterer Pluspunkt für Linux und freie Software ist die Weise, wie in der Szene mit Sicherheitslücken umgegangen wird. Sicherheitslücken sind in freier wie auch proprietärer Software einigermaßen unvermeidlich – jedenfalls hat noch niemand ein Programm interessanter Größe geschrieben und in Umlauf gebracht, das auf lange Sicht völlig frei von ihnen gewesen wäre. Insbesondere kann man auch nicht sagen, dass freie Software keine Sicherheitslücken hat. Die Unterschiede sind eher auf der philosophischen Ebene zu suchen:

- Ein Anbieter proprietärer Software hat in der Regel kein großes Interesse daran, Sicherheitslücken in seinem Code zu reparieren – er wird so lange wie irgend möglich versuchen, Probleme zu vertuschen und die möglichen Gefahren abzuwiegeln, denn im besten Fall bedeutet das ständige Veröffentlichen von »Patches« für Sicherheitsprobleme schlechte PR (»wo Rauch ist, ist auch Feuer«; die Konkurrenz, die gerade mal nicht im Rampenlicht steht, lacht sich ins Fäustchen), und im schlimmsten Fall hohe Kosten und jede Menge Ärger, wenn schädlicher Code im Umlauf ist, der die Sicherheitslücken ausnutzt. Außerdem gibt es wie üblich die Gefahr, bei der Korrektur eines Fehlers drei neue einzubauen, weswegen das Reparieren von Fehlern in veröffentlichter Software sich betriebswirtschaftlich nicht wirklich rechnet.

- Ein Anbieter freier Software gewinnt nichts dadurch, Informationen über Sicherheitslücken zu unterdrücken, denn der Quellcode steht allgemein zur Verfügung und jeder kann die Probleme nachvollziehen. Es ist eher eine Frage des Stolzes, bekannte Sicherheitslücken möglichst schnell in Ordnung zu bringen. Die Tatsache, dass der Quellcode allgemein zur Verfügung steht, bedeutet auch, dass Dritte es leicht haben, den Code auf Probleme zu untersuchen, die dann proaktiv repariert werden können. (Es wird gerne postuliert, dass die Verfügbarkeit des Quellcodes auch Cracker und ähnliches zwielichtiges Gesindel geradezu anlockt. Tatsache ist aber, dass solche Gestalten anscheinend kein gravierendes Problem damit haben, in proprietären Systemen wie Windows, wo der Quellcode *nicht* zur Verfügung steht, große Mengen von Sicherheitslücken zu finden. Der Unterschied ist also, wenn er überhaupt vorhanden ist, nicht gravierend.)

- Speziell im Fall von Software, die sich mit Kryptografie (also dem Verschlüsseln und Entschlüsseln vertraulicher Informationen) befasst, läßt sich argumentieren, dass die Verfügbarkeit des Quellcodes eine unabdingbare Vorbedingung dafür ist, Vertrauen aufbauen zu können, dass ein Programm tatsächlich das tut, was es soll, also das behauptete Verschlüsselungsverfahren vollständig und korrekt implementiert. Hier hat Linux eindeutig die Nase vorn.

Linux in Firmen

Linux wird heute weltweit sowohl im privaten als auch im professionellen Bereich (Firmen, Forschungseinrichtungen, Hochschulen) eingesetzt. Eine besondere Rolle spielt es dabei als System für Webserver (Apache), Mailserver (Sendmail, Postfix), Fileserver (NFS, Samba), Print-Server (LPD, CUPS), ISDN-Router, X-Terminal, Workstation usw. Linux ist aus den Rechenzentren der Industrie nicht mehr

Öffentliche Verwaltung

wegzudenken. Auch die Entscheidung von Kommunen wie der Stadt München, die Rechner der öffentlichen Verwaltung sehr weitgehend auf Linux umzustellen,

Unterstützung durch IT-Firmen

setzt Zeichen. Dazu kommt, dass namhafte IT-Firmen wie IBM, Hewlett-Packard,

Dell, Oracle, Sybase, Informix, SAP, Lotus etc. ihre Produkte auf Linux abstimmen oder in unterstützten Versionen für Linux anbieten. Ferner werden mehr und mehr Rechner von Haus aus mit Linux ausgeliefert oder zumindest vom Hersteller auf Linux-Kompatibilität getestet.

Übungen

1.6 [4] Stellen Sie sich vor, Sie sind verantwortlich für die EDV einer kleinen Firma (20–30 Mitarbeiter). Im Büro gibt es etwa 20 Büroarbeitsplätze und zwei Server (einen Datei- und Druckerserver sowie einen Mailserver bzw. Web-Proxyserver). Bisher läuft alles unter Windows. Betrachten Sie die folgenden Szenarien:

- Der Datei- und Druckerserver wird durch einen Linux-Rechner mit Samba (einem Linux/Unix-basierten Serverprogramm für Windows-Clients) ersetzt.
- Der Mailserver bzw. Web-Proxyserver wird durch einen Linux-Rechner ersetzt.
- Die 20 Büroarbeitsplätze werden durch Linux-Rechner ersetzt.

Kommentieren Sie die verschiedenen Szenarien und stellen Sie kurze Listen der Vor- und Nachteile auf.

1.6 Linux-Distributionen

Linux im engeren Sinne umfaßt nur den Betriebssystem-Kern. Um damit arbeiten zu können, benötigen Sie noch eine Vielzahl an System- und Anwendungsprogrammen, Bibliotheken, Dokumentationen usw. »Distributionen« sind nichts anderes als eine aktuelle Auswahl davon zusammen mit eigenen Programmen (insbesondere Werkzeugen zum Installieren und Administrieren), die von Firmen oder anderen Organisationen vertrieben werden, möglicherweise zusammen mit weiteren Leistungen wie Unterstützung, Dokumentation oder Aktualisierungen. Unterschiede gibt es vor allem in der Programmauswahl, den Administrationswerkzeugen, den Zusatzleistungen und dem Preis.

Distributionen

»Fedora« ist eine frei verfügbare Linux-Distribution, die unter der Federführung der amerikanischen Firma *Red Hat* entwickelt wird. Sie ist der Nachfolger der »Red Hat Linux«-Distribution; die Firma Red Hat hat sich aus dem Privatkundengeschäft zurückgezogen und zielt mit ihren Distributionen unter dem Namen »Red Hat« auf Firmenkunden. Red Hat wurde 1993 gegründet und ging im August 1999 an die Börse; das erste Red-Hat-Linux kam im Sommer 1994 heraus, das letzte (die Version 9) Ende April 2004. »Red Hat Enterprise Linux« (RHEL), das aktuelle Produkt, erschien zum ersten Mal im März 2002. Fedora ist, wie gesagt, ein frei verfügbares Angebot und dient als Entwicklungsplattform für RHEL, es ist unter dem Strich der Nachfolger von Red Hat Linux. Red Hat macht Fedora nur zum Download verfügbar; während Red Hat Linux noch als »Kiste« mit CDs und Handbüchern verkauft wurde, überläßt Red Hat das jetzt Drittanbietern.

Red Hat und Fedora

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

SUSE

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. Wie Red Hat bot SuSE auch ein Unternehmens-Linux an, den *SuSE Linux Enterprise Server* (SLES) mit einigen Ablegern wie einem speziellen Server für Mail und Groupware (den *SuSE Linux OpenExchange Server* oder SLOX). Außerdem bemühte sich SuSE darum, ihre Distribution auch auf den MDT- und Großrechnern von IBM zur Verfügung zu stellen.

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.) Auch SUSE hat inzwischen wie Red Hat den Schritt gemacht, die »Privatkunden«-Distribution zu öffnen und als »openSUSE« frei verfügbar zu machen (die früheren Versionen erschienen immer erst mit einigen Monaten Verzögerung zum Download). Im Gegensatz zu Red Hat bietet Novell/SUSE aber nach wie vor eine »Kiste« an, die zusätzlich proprietäre Software enthält. Verkauft werden außerdem derzeit unter anderem der SLES und eine Desktop-Plattform für Unternehmen, der *SUSE Linux Enterprise Desktop* (SLED).

Attachmate
Micro Focus

Anfang 2011 wurde Novell von der Firma Attachmate übernommen, die 2014 selber von Micro Focus gekauft wurde. Beides sind Firmen, die vor allem in der Welt der traditionellen Großrechner aktiv sind und sich im Linux- und Open-Source-Bereich bisher nicht besonders hervorgetan haben. Auf SUSE an sich und deren Produkte haben diese Manöver soweit keinen großen Einfluss gehabt.

YaST

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

Debian-Projekt

Im Gegensatz zu den beiden großen Linux-Distributionsfirmen Red Hat und Novell/SUSE ist das **Debian-Projekt** 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.

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 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.

Versionen

Zu jedem Zeitpunkt existieren mindestens drei Versionen von Debian

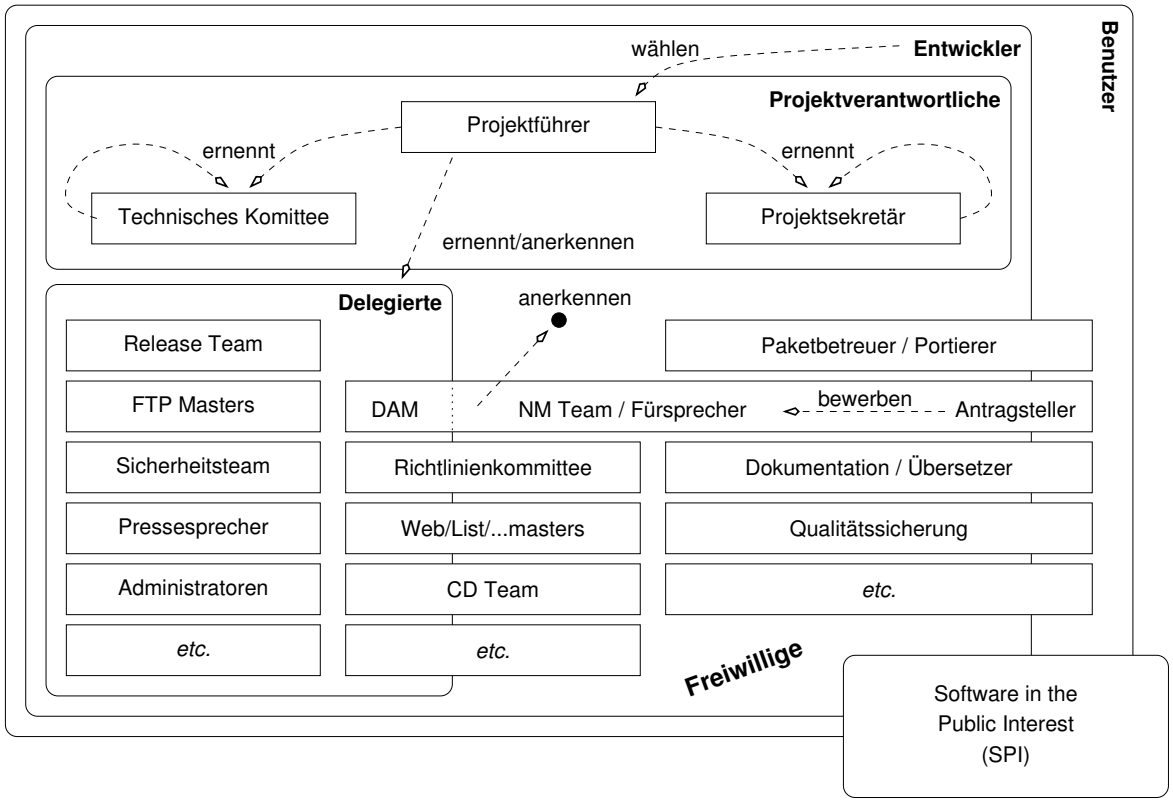


Bild 1.3: Organisationsstruktur des Debian Projekts. (Grafik von Martin F. Krafft.)

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`-Releases; 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.

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 (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.

Ableger-Projekte

Einer der populärsten Debian-Ableger ist Ubuntu, 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. Ubuntu basiert auf dem `unstable`-Zweig von Debian GNU/Linux und verwendet in weiten Teilen dieselben Werk-

Ubuntu

Ziel von Ubuntu

zeuge etwa zur Softwareverteilung, allerdings sind Debian- und Ubuntu-Softwarepakete nicht notwendigerweise miteinander kompatibel.

Ubuntu vs. Debian

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. SUSE/Red Hat

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 …

Weitere Distributionen

Außer diesen Distributionen gibt es noch viele weitere, etwa Mageia oder Linux Mint als kleinere »allgemein nützliche« Distributionen, 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.

Gemeinsamkeiten

Obwohl es eine Unzahl an Distributionen gibt, verhalten sie 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 entgegenwirken. Zu nennen sind hier der *Filesystem Hierarchy Standard* (FHS) oder die *Linux Standard Base* (LSB).

Übungen

1.7 [2] Einige Linux-Hardwareplattformen wurden oben aufgezählt. Für welche Plattformen sind tatsächlich Linux-Distributionen auf dem Markt? (Tipp: `http://www.distrowatch.org/`)

Zusammenfassung

- Linux ist ein Unix-artiges Betriebssystem.
- 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.
- Die GPL ist die bekannteste Freie-Software-Lizenz. Sie strebt danach, dass die Empfänger von Software diese modifizieren und weiterverteilen dürfen, diese »Freiheiten« dabei aber auch für künftige Empfänger erhalten bleiben. Verkaufen darf man GPL-Software trotzdem.
- »Open Source« bedeutet für den Anwender im Wesentlichen dasselbe wie »Freie Software«.
- Neben der GPL gibt es auch andere freie Lizenzen. Software kann vom Urheber auch unter mehreren verschiedenen Lizenzen gleichzeitig verteilt werden.
- Linux ist eigentlich nur der Betriebssystemkern. Man unterscheidet »stabile« und »Entwicklerkerne«. Stabile Kerne sind für Endanwender gedacht, während Entwicklerkerne nicht funktionieren müssen und Interimsversionen der Linux-Weiterentwicklung darstellen.
- Es gibt zahlreiche Linux-Distributionen, die einen Linux-Kern und zusätzliche Software vereinigen, dokumentieren und bequem installier- und administrierbar machen sollen.

Literaturverzeichnis

DFSG »Debian Free Software Guidelines«. `http://www.debian.org/social_contract`

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

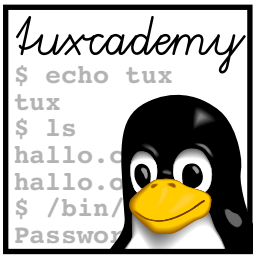
GPL91 Free Software Foundation, Inc. »GNU General Public License, Version 2«, Juni 1991. `http://www.gnu.org/licenses/gpl.html`

LR89 Don Libes, Sandy Ressler. *Life with UNIX: A Guide for Everyone*. Prentice-Hall, 1989. ISBN 0-13-536657-7.

Rit84 Dennis M. Ritchie. »The Evolution of the Unix Time-sharing System«. *AT&T Bell Laboratories Technical Journal*, Oktober 1984. **63**(6p2):1577–93. `http://cm.bell-labs.com/cm/cs/who/dmr/hist.html`

RT74 Dennis M. Ritchie, Ken Thompson. »The Unix Time-sharing System«. *Communications of the ACM*, Juli 1974. **17**(7):365–73. Der klassische Artikel über Unix.

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



2

Die Bedienung des Linux-Systems

Inhalt

Lernziele

- Sich beim System anmelden und abmelden können
- Den Unterschied zwischen normalen Benutzerkonten und dem Systemadministrator kennen

Vorkenntnisse

- Grundlegende Kenntnisse im Umgang mit Computern sind hilfreich

Bild 2.1: Die Anmeldebildschirme einiger gängiger Linux-Distributionen

2.1 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 (Bild 2.1 zeigt einige repräsentative Beispiele.)

Wundern Sie sich nicht, wenn Sie beim Eingeben Ihres Kennworts nur Sternchen 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

2.1 [!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 – im Schulungszentrum – Ihr Trainer.)

2.2 [!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?

2.2 An- und Ausschalten

Rechner ausschalten

Einschalten darf einen Linux-Rechner normalerweise jeder, der an den Schalter kommt (lokale Gepflogenheiten können anders aussehen). Allerdings sollten Sie einen Linux-Rechner nicht einfach so ausschalten – es könnten noch Daten im Hauptspeicher stehen, die eigentlich auf die Festplatte gehören und so verlorengehen, oder – was schlimmer wäre – die Daten auf der Festplatte könnten völlig durcheinanderkommen. Außerdem könnten andere Benutzer über das Netz auf dem Rechner angemeldet sein, durch das plötzliche Abschalten überrascht werden und wertvolle Arbeit verlieren. Aus diesem Grund werden wichtige Rechner normalerweise nur vom Systemverwalter »heruntergefahren«. Arbeitsplatzrechner für eine einzige Person dagegen können Sie heutzutage meist auch über die grafische Oberfläche sauber herunterfahren; je nach Systemeinstellung genügen dazu die Privilegien eines normalen Benutzers, oder Sie müssen das Kennwort des Systemverwalters eingeben.

Übungen

2.3 [2] Prüfen Sie, ob Sie Ihr System als normaler Benutzer sauber herunterfahren dürfen, und probieren Sie es gegebenenfalls aus.

2.3 Der Systemadministrator

`root`

Als normaler Benutzer sind Ihre Rechte im System beschränkt. Sie dürfen zum Beispiel in manche Dateien nicht schreiben (eigentlich die meisten, nämlich alle außer Ihren eigenen) und manche Dateien nicht einmal lesen (etwa die Datei, in der die verschlüsselten Kennwörter aller Benutzer stehen). Für Systemverwaltungszwecke gibt es allerdings eine Benutzerkennung, für die diese Einschränkung nicht gilt – der Benutzer »`root`« darf alle Dateien lesen und schreiben und auch sonst diverse Dinge tun, die den anderen Benutzern nicht offenstehen.

Administrator- oder, wie man auch sagt, root-Rechte zu haben ist ein Privileg, andererseits aber auch eine Gefahr – deswegen sollten Sie sich nur als root anmelden, wenn Sie tatsächlich Administrationsaufgaben ausführen müssen, und nicht zum E-Mail-Lesen oder Internet-Surfen.

Stellen Sie sich einfach vor, Sie seien Spider-Man: »Mit großer Macht kommt große Verantwortung.« Auch Spider-Man trägt seinen Elastan-Anzug nur, wenn er muss …

Vor allem sollten Sie es vermeiden, sich am grafischen Login als root anzumelden, da dann die gesamte grafische Oberfläche mit root-Rechten läuft, was ein Sicherheitsrisiko darstellt – Grafikoberflächen wie KDE enthalten riesige Mengen Code, die in der Regel nicht so gründlich auf mögliche Sicherheitslücken abgeklopft werden wie die im Vergleich relativ kompakte textorientierte Shell. Üblicherweise können Sie mit dem Befehl »/bin/su -« auf der Kommandozeile die Identität (und damit die Rechte) von root annehmen. su fragt nach dem root-Kennwort und startet dann eine neue Shell, in der Sie so arbeiten können, als hätten Sie sich als root angemeldet. Diese Shell können Sie später mit dem Kommando exit verlassen.

Grafische Oberfläche als root: riskant

Identität von root annehmen

Sie sollten sich angewöhnen, das Programm su immer über seinen vollständigen Pfadnamen aufzurufen – »/bin/su -«. Ansonsten könnte es sein, dass ein Benutzer Sie foppt, indem er Sie wegen irgendeines unerklärlichen Fehlers an seinen Rechner holt und Sie dazu verleitet, in einem seiner Fenster »su« zu sagen und das root-Kennwort einzugeben. Was Sie in dem Moment nicht wissen, ist, dass der clevere Benutzer selber ein »trojanisches« su-Programm geschrieben hat, das nichts tut außer das eingegebene Kennwort in eine Datei zu schreiben, die Fehlermeldung für vertippte Kennwörter auszugeben und sich selbst zu löschen. Wenn Sie es dann zähneknirschend nochmal probieren, dann bekommen Sie das echte su – und Ihr Benutzer ist fortan im Besitz der begehrten Administratorprivilegien …

Dass Sie tatsächlich über Systemverwalterprivilegien verfügen, erkennen Sie meist daran, dass die Eingabeaufforderung mit dem Zeichen »#« aufhört. Die Eingabeaufforderung für normale Benutzer endet gemeinhin auf »$« oder »>«.

Eingabeaufforderung für root

In Ubuntu können Sie sich standardmäßig gar nicht als root anmelden. Statt dessen erlaubt das System dem bei der Installation als erstes angelegten Benutzer, Kommandos mit Administratorrechten auszuführen, indem er das Kommando sudo davorsetzt. Mit

```
$ sudo chown hugo datei.txt
```

könnte er zum Beispiel die Datei datei.txt dem Benutzer hugo überschreiben – eine Operation, die dem Administrator vorbehalten ist.

Neue Versionen von Debian GNU/Linux bieten ein ähnliches Arrangement an wie Ubuntu.

In der KDE-Oberfläche haben Sie es übrigens ganz einfach, beliebige Programme als root auszuführen: Wählen Sie im »KDE«-Menü – normalerweise der Eintrag ganz links in der Leiste, da wo ein Windows-Rechner das »Start«-Menü hat – den Punkt »Befehl ausführen …«. In dem dann erscheinenden Dialogfenster können Sie einen Befehl eingeben. Bevor Sie ihn ausführen, klicken Sie auf die Schaltfläche »Einstellungen«; es öffnet sich ein Bereich mit zusätzlichen Einstellungen, wo Sie »Mit anderer Benutzerkennung« anklicken (hilfreicherweise ist root der Standardwert). Sie müssen dann nur noch das root-Kennwort im vorgesehenen Feld eingeben (Bild 2.2).

root und KDE

Alternativ dazu können Sie im selben Dialogfenster im Kommandofeld auch »kdesu« vor das eigentliche Kommando schreiben. Dann werden Sie im Anschluss nach dem root-Kennwort gefragt.

kdesu

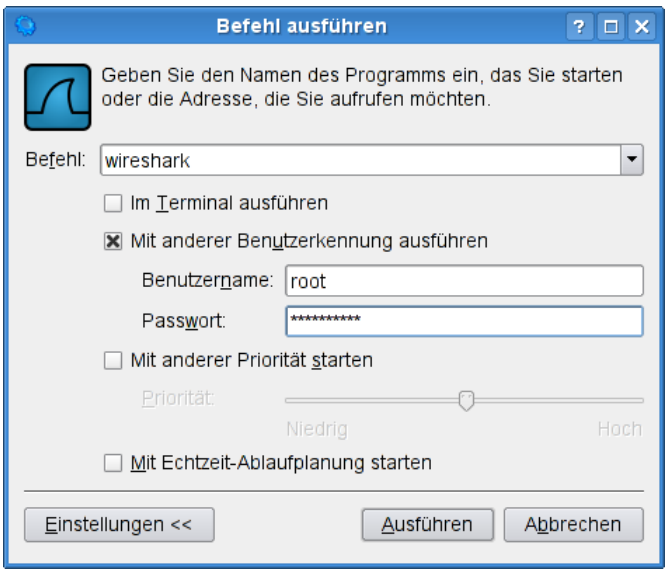


Bild 2.2: Programme als anderer Benutzer ausführen in KDE

Übungen

2.4 [!1] Verwenden Sie das Programm `su`, um Systemverwalterprivilegien zu erlangen. Rufen Sie das Kommando `id` auf, um sich zu überzeugen, dass Sie tatsächlich `root` sind – es sollte etwas ausgeben wie

```
# id
uid=0(root) gid=0(root) groups=0(root)
```

– und wechseln Sie zurück zu Ihrer normalen Benutzerkennung.

2.5 [5] (Für Programmierer.) Schreiben Sie ein überzeugendes »trojanisches« `su`-Programm. Versuchen Sie, Ihren Systemverwalter damit aufs Glatteis zu führen.

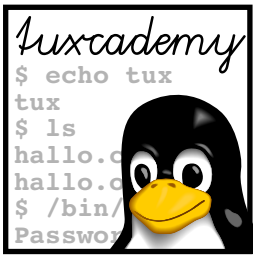
2.6 [2] Versuchen Sie, mit dem »Befehl ausführen …«-Dialog von KDE das Programm `id` in einer Terminalsitzung auszuführen. Kreuzen Sie dazu das entsprechende Feld in den erweiterten Einstellungen an.

Kommandos in diesem Kapitel

exit	Beendet eine Shell	bash(1)	35
id	Gibt UID und GIDs eines Benutzers aus	id(1)	36
kdesu	Startet unter KDE ein Programm als anderer Benutzer	KDE: help:/kdesu	35
logout	Beendet eine Sitzung („Abmelden“)	bash(1)	34
su	Startet eine Shell unter der Identität eines anderen Benutzers	su(1)	35
sudo	Erlaubt normalen Benutzern das Aufrufen bestimmter Kommandos mit Administratorprivilegien	sudo(8)	35

Zusammenfassung

- Vor dem Benutzen eines Linux-Rechners müssen Sie sich (meistens) mit Benutzername und Kennwort anmelden und hinterher wieder abmelden.
- Für den Benutzer `root` gelten die normalen Zugriffsrechte nicht – er darf (tendenziell) alles. Diese Privilegien sollten so sparsam wie möglich eingesetzt werden.
- Sie sollten sich nicht als `root` in der grafischen Oberfläche anmelden, sondern lieber bei Bedarf zum Beispiel mit `su` die Identität des Administrators annehmen.



3

Keine Angst vor der Shell

Inhalt

Lernziele

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

Vorkenntnisse

- Grundlegende Kenntnisse im Umgang mit Computern sind hilfreich

3.1 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.

3.2 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.

Wenn im System mehrere verschiedene Shells zur Verfügung stehen (der Regelfall), können Sie mit den folgenden Befehlen zwischen diesen umschalten: Wechseln zwischen Shells

`sh` für die klassische Bourne-Shell (falls vorhanden – auf den meisten Linux-Systemen ist auch `sh` eine Bash).

`bash` für die »Bourne-Again-Shell« (Bash).

`ksh` für die Korn-Shell.

`csh` für die C-Shell.

`tcsh` für die »Tenex-C-Shell«, eine erweiterte und verbesserte Version der gewöhnlichen C-Shell. Auf vielen Linux-Systemen ist das Programm `csh` in Wirklichkeit ein Verweis auf die `tcsh`.

Für den Fall, dass Sie nicht mehr wissen sollten, mit welcher Shell Sie gerade arbeiten, hilft die Eingabe von »`echo $0`«, die so ziemlich überall funktioniert und als Ausgabe die Bezeichnung der Shell liefert.

Übungen

3.1 [2] Wie viele verschiedene Shells sind auf Ihrem Rechner installiert und welche? (*Tipp:* Prüfen Sie die Datei `/etc/shells`.)

3.2 [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?

3.3 Kommandos

3.3.1 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 »`$ `«.)

3.3.2 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 3.3.4.

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.

3.3.3 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.

Extern oder intern? 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

```
$ 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

3.3 [2] Welche der folgenden Kommandos sind bei der Bash extern und welche intern realisiert: `alias`, `echo`, `rm`, `test`?

3.3.4 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

```
$&;(){}[]*?!<>«
```

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: Maskierung

```
$ 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.

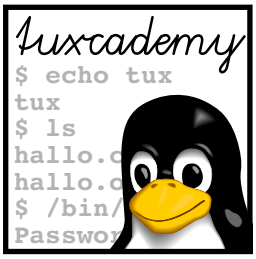
Die anderen Sonderzeichen zu erklären würde an dieser Stelle zu weit führen. Die meisten tauchen anderswo in dieser Schulungsunterlage auf – oder beziehen Sie sich auf die Bash-Dokumentation.

Kommandos in diesem Kapitel

bash	Die „Bourne-Again-Shell“, ein interaktiver Kommandointerpreter	bash(1)	41
csh	Die „C-Shell“, ein interaktiver Kommandointerpreter	csh(1)	41
date	Gibt Datum und Uhrzeit aus	date(1)	44
echo	Gibt alle seine Parameter durch Leerzeichen getrennt auf der Standardausgabe aus	bash(1), echo(1)	44
help	Zeigt Hilfe für bash-Kommandos	bash(1)	44
konsole	Ein „Terminalemulator“ für KDE	KDE: help:/konsole	42
ksh	Die „Korn-Shell“, ein interaktiver Kommandointerpreter	ksh(1)	41
sh	Die „Bourne-Shell“, ein interaktiver Kommandointerpreter	sh(1)	41
tcsh	Die „Tenex-C-Shell“, ein interaktiver Kommandointerpreter	tcsh(1)	41
type	Bestimmt die Art eines Kommandos (intern, extern, Alias)	bash(1)	44
xterm	Ein „Terminalemulator“ für das X-Window-System	xterm(1)	42

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.



4

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)

4.1 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.

4.2 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 4.1: 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

4.3 Die Handbuchseiten

4.3.1 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.

4.3.2 Struktur

Der Aufbau der Handbuchseiten folgt lose der in Tabelle 4.1 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 4.2: 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 4.2 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.

4.3.3 Kapitel

Kapitel

Jede Handbuchseite gehört zu einem »Kapitel« im konzeptuellen Gesamthandbuch (Tabelle 4.2). 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.

4.3.4 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

4.1 [!1] Schauen Sie sich die Handbuchseite zum Programm `ls` an. Benutzen Sie dafür das textbasierte Programm `man` und – falls vorhanden – den Konqueror-Browser.

4.2 [2] Welche Handbuchseiten auf Ihrem System beschäftigen sich (jedenfalls ihrem NAME-Abschnitt nach) mit Prozessen?

4.3 [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 -`«).

4.4 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,

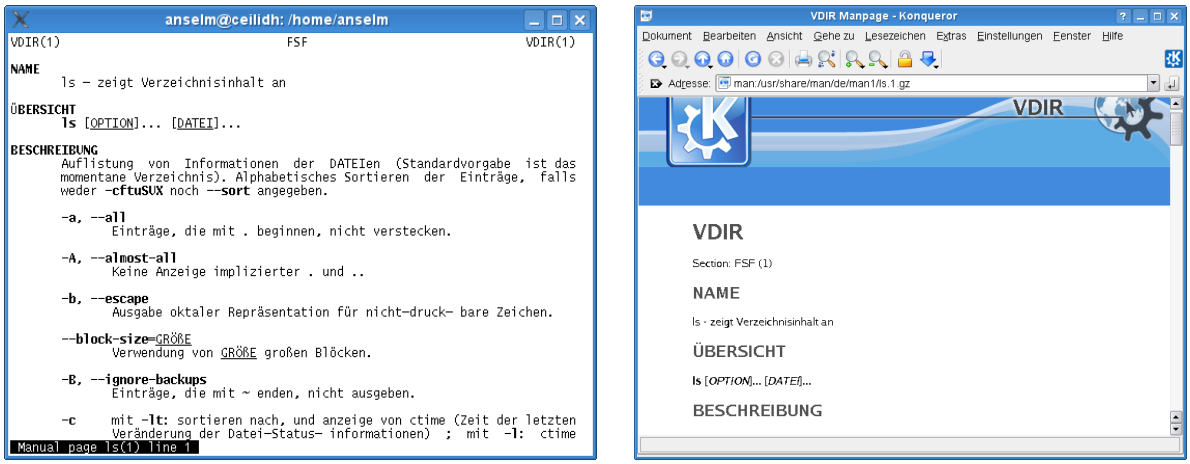


Bild 4.1: Eine Handbuchseite im Textterminal (links) und im Konqueror (rechts)

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

4.4 [!1] Schauen Sie sich die Info-Seite zum Programm `ls` an. Probieren Sie das textbasierte `info` und, falls vorhanden, den Konqueror-Browser aus.

4.5 [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?

4.5 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

4.6 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

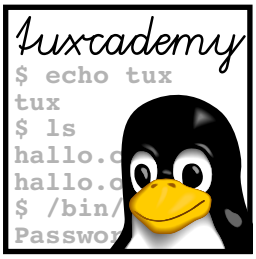
Der große Vorteil an Open-Source-Software ist übrigens nicht nur die riesige Menge an Dokumentation, sondern auch, dass die meisten Dokumente ähnlich wenig Restriktionen unterliegen wie die Software selbst. Dadurch ist eine umfassendere Zusammenarbeit von Softwareentwicklern und Dokumentationsautoren 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) 50
groff	Programm zur druckreifen Aufbereitung von Texten	groff(1) 50
help	Zeigt Hilfe für bash-Kommandos	bash(1) 48
info	Zeigt GNU-Info-Seiten auf einem Textterminal an	info(1) 52
less	Zeigt Texte (etwa Handbuchseiten) seitenweise an	less(1) 50
man	Zeigt Handbuchseiten des Systems an	man(1) 48
manpath	Bestimmt den Suchpfad für Handbuchseiten	manpath(1) 50
whatis	Sucht Handbuchseiten mit dem gegebenen Stichwort in der Beschreibung	whatis(1) 51

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.



5

Editoren: vi und emacs

Inhalt

Lernziele

- Die Editoren vi und emacs kennenlernen
- Textdateien anlegen und verändern können

Vorkenntnisse

- Arbeit mit der Shell (siehe Kapitel 2)

5.1 Editoren

Um Texte komfortabel erfassen oder verändern zu können, stehen auf allen Betriebssystemen entsprechende Hilfsprogramme zur Verfügung. Entsprechend seiner Aufgabe »Textbearbeitung« wird ein derartiges Programm als Editor (lat. *edire*, »bearbeiten«) bezeichnet.

Was tut ein Editor?

Zunächst sollte der Funktionsumfang eines Texteditors genauer umrissen werden. Mit dem einfachen Bearbeiten einzelner Zeichen sind komplexere Aufgaben nur schwer zu meistern. Gute Editoren stellen daher Funktionen zur Verfügung, mit denen gleich ganze Wörter oder Zeilen ausgeschnitten, kopiert oder eingefügt werden können. Für lange Dateien ist ferner ein Mechanismus sinnvoll, welcher den Text nach bestimmten Zeichenfolgen durchforstet. In Erweiterung dazu können ausgefeilte »Suchen und Ersetzen«-Befehle mühsame Aufgaben wie »Ersetze jedes `x` durch ein `u`« spürbar vereinfachen. Je nach Editor stehen noch wesentlich mächtigere Hilfsfunktionen zum Bearbeiten von Texten zur Verfügung.

Unterschied zu Textverarbeitungen

Im Unterschied zu Textverarbeitungen wie OpenOffice.org Writer oder Microsoft Word stellen Editoren jedoch normalerweise keine gestalterischen Elemente wie verschiedene Schriftschnitte (Times, Helvetica, Courier, …), Schriftattribute (fett, kursiv, unterstrichen, …), Satzvorlagen (Blocksatz, …) oder ähnliches zur Verfügung – sie sind vor allem zum Eingeben und Ändern von reinen Textdateien gedacht, wo so etwas eher stören würde.

Natürlich spricht nichts dagegen, mit einem Texteditor Eingabedateien für Textsatzsysteme wie `groff` oder LaTeX zu schreiben, die diese Elemente aus dem FF beherrschen. Nur sehen Sie davon mitunter nichts in der ursprünglichen Eingabe – was auch ein Vorteil sein kann: Schließlich lenken viele dieser Elemente beim Schreiben eher ab, und die Autoren werden verleitet, bei der Texteingabe am Aussehen des Textes herumzubasteln, statt sich auf den Inhalt zu konzentrieren.

Syntaxhervorhebung

Was die meisten aktuellen Texteditoren heutzutage unterstützen, ist Syntaxhervorhebung, also die Kenntlichmachung bestimmter Elemente in einem Programmtext – Kommentare, Variablennamen, Schlüsselwörter, Zeichenketten – durch Farbe oder besondere Schriftarten. Dies sieht auf jeden Fall gefällig aus, wenn auch die Frage, ob das wirklich beim Programmieren hilft, noch nicht durch entsprechende psychologische Studien entschieden werden konnte.

In den nachfolgenden Abschnitten nehmen wir zwei gängige Linux-Editoren unter die Lupe. Allerdings beschränkt sich die Betrachtung dabei auf die grundlegenden Funktionsweisen, denn im Prinzip könnte man zu jedem dieser Programme eine mehrtägige Schulung durchführen. Ebenso wie bei den Shells ist es letztendlich dem persönlichen Geschmack des Benutzers überlassen, mit welchem Editor er seine Texte manipuliert.

Übungen

5.1 [2] Welche Texteditoren sind auf Ihrem System installiert? Wie können Sie das herausfinden?

5.2 Der Standard – `vi`

5.2.1 Überblick

Der einzige Editor, der wohl in allen Linux-Systemen installiert ist, heißt `vi` (engl. *visual*, »sichtbar«). Aus praktischen Gründen handelt es sich dabei oftmals nicht um das Originalprogramm (das aus BSD stammt und inzwischen deutlich in die Jahre gekommen ist), sondern um Ableger wie `vim` (engl. *vi improved*, »verbessert«)

`vi`: heute meist Klon

oder elvis; diese Editoren sind jedoch so weit kompatibel zu vi, dass wir sie der Einfachheit halber mit dem Original in einen Topf werfen.

Einer der ersten Bildschirmeditoren

vi, ursprünglich entwickelt von Bill Joy für BSD, war einer der ersten gebräuchlichen »bildschirmorientierten« Editoren für Unix. Das bedeutet, dass nicht mehr nur einzelne Zeilen bearbeitet werden können, sondern der gesamte Bildschirm als Arbeitsfläche zur Verfügung steht. Was heutzutage eine Selbstverständlichkeit darstellt, war damals eine Innovation – was nicht daran liegt, dass die Programmierer vorher unfähig gewesen wären, sondern dass Textterminals mit freier Ansteuerung von beliebigen Zeichenpositionen auf dem Bildschirm, eine zwingende Voraussetzung für Programme wie den vi, erst zu jener Zeit zu erschwinglichen Preisen auf den Markt kamen. Aus Rücksichtnahme auf die damals älteren Systeme mit Fernschreibern oder *glass ttys*, also Terminals, die fernschreibermäßig nur unten am Bildschirmrand neue Zeichen schreiben konnten, unterstützt der vi unter dem Namen ex noch einen zeilenorientierten Modus.

Niedrige Anforderungen an Tastatur

Man konnte sich selbst bei den fortschrittlichen Terminals der damaligen Zeit allerdings nicht darauf verlassen, dass die Tastatur über das normale Buchstabenfeld hinaus noch weitere Funktionstasten etwa zur Cursorsteuerung enthielt – die heute üblichen PC-Standardtastaturen wären damals durchaus als luxuriös, wenn nicht gar überladen empfunden worden. Diese Tatsache rechtfertigt das eigentümliche Bedienungskonzept des vi, das man heute mit Recht als vorsintflutlich ansehen kann. Man kann es niemandem übelnehmen, wenn er oder sie den vi deswegen ablehnt. Trotzdem kann es nicht schaden, wenn Sie zumindest rudimentäre vi-Kenntnisse haben, selbst wenn Sie sich für Ihre tägliche Arbeit einen anderen Editor aussuchen – was Sie unbedingt tun sollten, wenn der vi Ihnen nicht zusagt. Es ist ja nicht so, dass es keine Auswahl gäbe, und auf kindische Spielchen wie »Wer den vi nicht benutzt, ist kein richtiger Linux-Anwender« lassen wir uns nicht ein. Die heute üblichen grafischen Oberflächen wie KDE enthalten zum Beispiel auch sehr schöne und leistungsfähige Editoren.

Es gibt in der Tat noch einen primitiveren Editor als den vi – das Programm ed. Eigentlich gebührt der Titel »einziger Editor, der garantiert auf jedem Unix-System vorhanden ist«, dem ed viel eher als dem vi, aber der ed als reiner Zeileneditor mit Benutzungsoberfläche aus der Fernschreiberzeit ist selbst Hardcore-Unix-Verfechtern zu ungenießbar. (Von der Idee her ist ed zu vergleichen mit dem DOS-Programm EDLIN, ed ist aber ungleich leistungsfähiger als das Programm aus Redmond.) Der Grund dafür, dass ed trotz der Existenz von Dutzenden bequemerer Texteditoren immer noch vorhanden ist, ist etwas unoffensichtlich, aber sehr Unix-artig: ed akzeptiert Kommandos über seine Standardeingabe und kann darum in Shellskripten benutzt werden, um Textdateien automatisch zu verändern. Der ed erlaubt Editieroperationen auf der gesamten Datei und kann damit prinzipiell mehr als sein Kollege, der »Stromeditor« sed, der nur seine Standardeingabe auf seine Standardausgabe kopiert und dabei gewisse Veränderungen vornehmen kann; normalerweise würde man den sed verwenden und nur in Ausnahmefällen auf den ed zurückgreifen, aber der ist trotzdem hin und wieder nützlich.

5.2.2 Grundlegende Funktionen

Puffer

Das Pufferkonzept Die Arbeit von vi erfolgt mit Hilfe sogenannter **Puffer**. Rufen Sie vi mit einem Dateinamen als Argument auf, so wird der Inhalt dieser Datei in einen Puffer gelesen. Falls keine Datei entsprechenden Namens existiert, wird ein leerer Puffer angelegt.

Alle im Editor vorgenommen Änderungen am Text werden zunächst nur innerhalb des Puffers durchgeführt. Um diese Korrekturen dauerhaft auf dem Datenträger zu verewigen, muss der Pufferinhalt explizit in die Datei zurückgeschrieben werden. Sollen stattdessen die bisher durchgeführten Änderungen verworfen werden, beenden Sie vi einfach, ohne vorher den Pufferinhalt abzuspeichern – die Datei auf dem Datenträger bleibt unverändert.

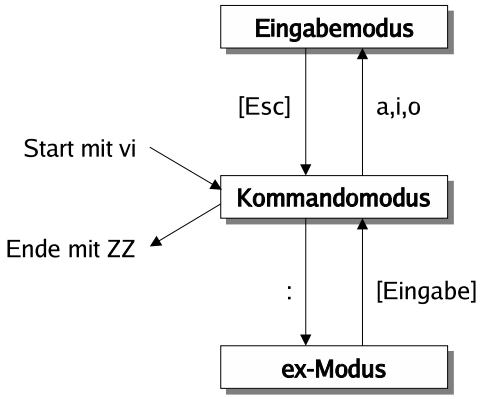


Bild 5.1: Arbeitsmodi von vi

Neben einem Dateinamen als Argument können vi beim Aufruf wie gewohnt auch Optionen anvertraut werden. Aufgrund derer relativ geringen Bedeutung sei hierzu auf die zugehörige Dokumentation verwiesen.

Arbeitsmodi

Arbeitsmodi Wie eingangs erwähnt, zeichnet sich vi durch seine ausgefallene Bedienung aus. Dieser Editor kennt nämlich drei verschiedene **Arbeitsmodi**:

Kommandomodus Alle Tastatureingaben stellen Kommandos dar, die nicht auf dem Bildschirm erscheinen und zumeist nicht durch Drücken der Eingabetaste bestätigt werden müssen.

In diesem Modus befinden Sie sich direkt nach dem Start. Seien Sie vorsichtig: Jeder Tastendruck könnte eine Aktion auslösen.

Eingabemodus Alle Tastatureingaben werden als Text behandelt und sind auf dem Monitor zu sehen. vi verhält sich hier wie ein heute gebräuchlicher Editor, allerdings mit eingeschränkten Manövrier- und Korrektur-Möglichkeiten.

Kommandozeilenmodus Im Kommandozeilenmodus können längere Befehle eingegeben werden. Diese werden üblicherweise durch einen Doppelpunkt eingeleitet und mit der Eingabetaste abgeschlossen.

Im Eingabemodus sind nahezu alle Kommandos zur Positionierung der Schreibmarke oder zur Textkorrektur unzulässig, was einen häufigen Wechsel zwischen Eingabe- und Kommandomodus erforderlich macht. Da es zudem – je nach der tatsächlich benutzten vi-Implementierung und deren Konfiguration – schwierig ist, zu erkennen, in welchem Modus sich vi gerade befindet, wird dem Einsteiger die Arbeit nicht gerade erleichtert. Eine Übersicht über die Arbeitsmodi von vi gibt Bild 5.2.

Denken Sie daran: Der vi fing an zu einer Zeit, als man nur den »Buchstabenblock« der Tastatur hatte (127 ASCII-Zeichen). An einer Mehrfachbelegung der Tasten führte darum kein Weg vorbei.

Kommandomodus

Nach dem Aufruf mit vi ohne Dateinamen gelangen Sie zunächst in den Kommandomodus. Im Unterschied zu den meisten anderen Editoren ist also nach dem Start keine direkte Texteingabe möglich. Auf dem Bildschirm blinkt links oben der Cursor, darunter befindet sich eine Spalte mit Tilden. In der letzten Zeile, auch als Statuszeile bekannt, werden neben Informationen zum momentanen Zustand des Editors (wenn die Implementierung das so macht) oder der aktuellen Datei noch die Cursorposition angezeigt (Bild 5.2).

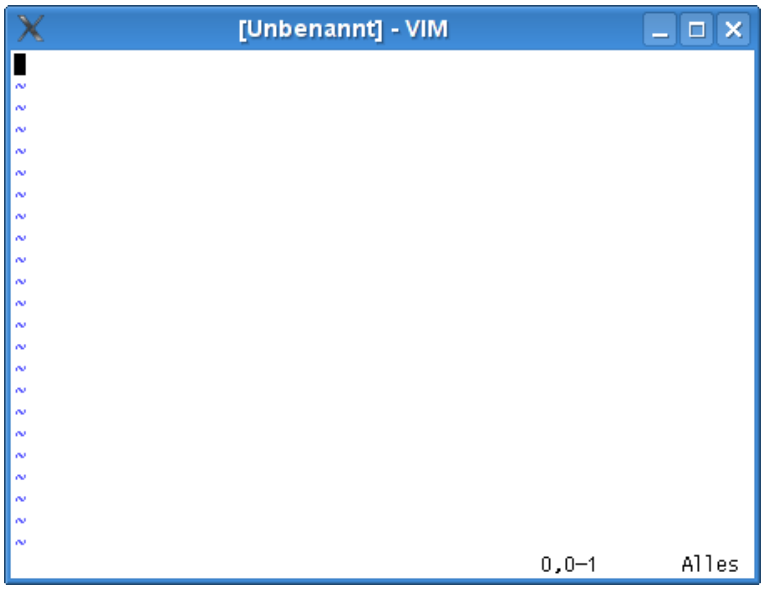


Bild 5.2: Der Editor vi (direkt nach dem Start)

Tabelle 5.1: Tastaturbefehle für den Eingabemodus von vi

Tastaturkommando	Wirkung
a	hängt neuen Text hinter dem aktuellen Zeichen an
A	hängt neuen Text am Zeilenende an
i	fügt neuen Text vor dem aktuellen Zeichen ein
I	fügt neuen Text am Zeilenanfang ein
o	fügt eine neue Zeile unter der aktuellen ein
O	fügt eine neue Zeile über der aktuellen ein

Tabelle 5.2: Tastaturbefehle zur Cursorpositionierung in vi

Tastaturkommando	Wirkung
h oder ←	ein Zeichen nach links
l oder →	ein Zeichen nach rechts
k oder ↑	ein Zeichen nach oben
j oder ↓	ein Zeichen nach unten
0	zum Zeilenanfang springen
$	zum Zeilenende springen
w	zum nächsten Wort springen
b	zum vorherigen Wort springen
f ⟨*Zeichen*⟩	zum nächsten ⟨*Zeichen*⟩ in der Zeile springen
Strg + f	eine Bildschirmseite vorblättern
Strg + b	eine Bildschirmseite zurückblättern
G	zur letzten Zeile der Datei springen
⟨*n*⟩ G	zur Zeile Nummer ⟨*n*⟩ springen

Wenn Ihr vi keine Statusinformationen anzeigt, versuchen Sie Ihr Glück einmal mit Esc `:set showmode` ↩.

Eingabemodus

Erst durch ein Kommando wie a (engl. *append*, »Anhängen«), i (engl. *insert*, »Einfügen«) oder o (engl. *open*, »Öffnen«) wechselt vi in den Eingabemodus. In der Statuszeile erscheint etwas wie »`-- INSERT --`«, und Tastatureingaben werden als Text entgegengenommen.

Die möglichen Kommandos zum Übergang in den Eingabemodus sind in Tabelle 5.1 aufgelistet, wobei zwischen Groß- und Kleinschreibung unterschieden wird. Um den Eingabemodus wieder zu verlassen, genügt ein Druck auf die Esc -Taste, und Sie finden sich im Kommandomodus wieder. Dort schreibt vi durch Eingabe von Z Z den Inhalt des Puffers auf den Datenträger und beendet sich anschließend.

Wollen Sie die vorgenommenen Änderungen lieber verwerfen, müssen Sie den Editor beenden, ohne dabei den Pufferinhalt abzuspeichern. Dies geschieht mit dem Befehl : `q!` ↩ (engl. *quit*, »aufhören«). Der einleitende Doppelpunkt zeigt, dass es sich hierbei um einen Befehl im Kommandozeilenmodus handelt.

Kommandozeilenmodus

Durch Eingabe von : im Kommandomodus wechselt vi in den Kommandozeilenmodus. Dies ist daran zu erkennen, dass in der untersten Zeile der Doppelpunkt und dahinter der Cursor erscheint. Alle weiteren Tastatureingaben werden nun an den Doppelpunkt angehängt, bis der Befehl durch die Eingabetaste (↩) abgeschlossen wird und vi in den Kommandomodus zurückkehrt. Im Kommandozeilenmodus verarbeitet vi die zeilenorientierten Kommandos seines *alter ego*, des Zeileneditors ex.

Auch zum zwischenzeitlichen Abspeichern des Pufferinhalts existiert ein ex-Kommando, dieses lautet : `w` (engl. *write*, »schreiben«). Mit dem Befehl : `x` oder der Kombination : `wq` wird der Pufferinhalt abgespeichert und danach der Editor beendet, beide Möglichkeiten sind also identisch mit dem Kommando Z Z.

Bewegung im Text Im Eingabemodus werden neu eingegebene Zeichen in die aktuelle Zeile aufgenommen, durch die Eingabetaste wird eine neue Zeile angelegt. Zwar ist mit den Pfeiltasten eine Bewegung im Text möglich, aber nur innerhalb der aktuellen Zeile können Sie die zuletzt getippten Buchstaben mit der Taste ⇐ wieder entfernen – ein Erbe der zeilenorientierten Vorgänger von vi. Weitergehende Bewegungen im Text sind nur im Kommandomodus möglich (Tabelle 5.2).

Tabelle 5.3: Tastaturbefehle zur Textkorrektur in vi

Tastaturkommando	Wirkung
x	löscht das Zeichen unter dem Cursor
X	löscht das Zeichen vor dem Cursor
r ⟨*Zeichen*⟩	ersetzt das Zeichen unter dem Cursor durch ⟨*Zeichen*⟩
d w	löscht ab Cursor bis Wortende
d $	löscht ab Cursor bis Zeilenende
d 0	löscht ab Cursor bis Zeilenanfang
d f ⟨*Zeichen*⟩	löscht ab Cursor bis zum nächsten ⟨*Zeichen*⟩ in der Zeile
d d	löscht aktuelle Zeile
d G	löscht ab aktueller Zeile bis zum Textende
d 1 G	löscht ab aktueller Zeile bis zum Textanfang

Tabelle 5.4: Tastaturbefehle zur Textersetzung in vi

Tastaturkommando	Wirkung
c w	ab Cursor bis Wortende ersetzen
c $	ab Cursor bis Zeilenende ersetzen
c 0	ab Cursor bis Zeilenanfang ersetzen
c f ⟨*Zeichen*⟩	ab Cursor bis zum nächsten ⟨*Zeichen*⟩ in der Zeile ersetzen
c / `abc`	ab Cursor bis zur nächsten Zeichenfolge `abc` ersetzen

Haben Sie den Cursor nun an die richtige Textstelle dirigiert, können Sie mit der Korrektur beginnen, die ebenfalls im Kommandomodus erfolgen muss.

Löschen von Zeichen Zum Löschen von Zeichen existiert der Befehl d (engl. *delete*, »Löschen«), der immer durch ein Folgezeichen konkretisiert werden muss (Tabelle 5.3). Um Korrekturen zu erleichtern, können Sie den aufgelisteten Befehlen einen Wiederholungsfaktor voranstellen, zum Beispiel löscht die Eingabe 3 x die nächsten drei Zeichen.

Wiederholungsfaktor

Falls Sie im Eifer des Gefechtes zu viel ver(schlimm)bessert haben, können Sie mit u (*undo*, »ungeschehen machen«) je nach Systemeinstellungen nur die letzte oder gar alle vorgenommenen Änderungen nacheinander zurücknehmen.

undo

Überschreiben von Zeichen Das Kommando c (engl. *change*, »wechseln«) dient zum Überschreiben eines ausgewählten Textbereichs. Genau wie d ist c ein Kombinationskommando, verlangt also eine zusätzliche Definition. Nach der Befehlseingabe wird der angegebene Textausschnitt entfernt und automatisch in den Eingabemodus gewechselt. Dort können Sie den neue Text einfügen und mit Esc wieder in den Kommandomodus zurückkehren (Tabelle 5.4).

Überschreiben

5.2.3 Erweiterte Funktionen

Textteile ausschneiden, kopieren und einfügen Ein häufiges Problem bei der Textbearbeitung ist, dass eine bereits existierende Passage an eine andere Position verschoben oder kopiert werden soll. Auch hierfür hat vi passende Kombinationsbefehle parat, für die die für c aufgelisteten Zusätze gültig sind. y (engl. *yank*, »zerren«) kopiert einen Text in den Zwischenspeicher, ohne den Ursprungstext zu

verändern. d hingegen verschiebt den ausgewählten Text in den Zwischenspeicher, der Textbereich wird also aus seiner ursprünglichen Position ausgeschnitten und ist nur noch im Zwischenspeicher vorhanden. (Wir haben das schon weiter oben als »Löschen« vorgestellt.)

Natürlich gibt es auch einen Befehl, um Textbausteine aus dem Zwischenspeicher wieder einzufügen. Dies erfolgt mit p hinter bzw. mit P vor der momentanen Cursorposition (engl. *paste*, »aufkleben«).

26 Zwischenspeicher

Eine Besonderheit von vi ist, dass er für derartige Operationen nicht nur einen, sondern gleich 26 Zwischenspeicher anbietet. Das ergibt Sinn, wenn mehrere unterschiedliche Textblöcke häufiger in den Text eingebaut werden sollen. Diese Zwischenspeicher sind fortlaufend von »a« bis »z« benannt und werden mit einer Kombination aus Anführungszeichen und Speichernamen aufgerufen. Die Befehlssequenz " c y 4 w kopiert also die nächsten vier Wörter in den Zwischenspeicher namens c; mit " g p wird der Inhalt des Zwischenspeichers g hinter der momentanen Cursorposition eingefügt.

Textsuche mit regulären Ausdrücken Wie es sich für einen ordentlichen Editor gehört, stellt vi ausgeklügelte Suchfunktionen zur Verfügung. Neben exakten Zeichenfolgen können mit Hilfe von »regulären Ausdrücken« auch ganze Mengen von Zeichenfolgen aufgespürt werden. Um den Suchvorgang einzuleiten, müssen Sie im Kommandomodus einen Schrägstrich / eingeben. Dieser erscheint daraufhin gefolgt vom Cursor in der letzten Zeile. Nachdem der Suchbegriff eingetippt worden ist, startet die Eingabetaste den Suchvorgang. Dieser wird von der aktuellen Cursorposition bis zum Textende, also textabwärts, durchgeführt. Soll stattdessen textaufwärts gesucht werden, muss die Suchfunktion nicht mit / , sondern mit ? aufgerufen werden. Sobald vi eine passende Zeichenfolge gefunden hat, wird die Suche beendet und der Cursor auf den ersten Buchstaben dieser Sequenz gesetzt. Mit n wird die Suche textabwärts, mit N hingegen textaufwärts fortgesetzt.

Details über reguläre Ausdrücke finden Sie in den Abschnitten 7.1 und 7.2. Lassen Sie sich beim ersten Lesen dieser Unterlage nicht ins Bockshorn jagen, sondern denken Sie sich »Zeichenkette« überall da, wo »regulärer Ausdruck« steht.

Suchen und Ersetzen Mit dem Auffinden von Zeichenketten alleine ist es oftmals nicht getan, darum bietet vi zusätzlich die Möglichkeit, die gefundenen Textteile durch andere zu ersetzen. Diese Aufgabe erledigt ein ex-Kommando mit der Syntax:

: [⟨*Startzeile*⟩,⟨*Endzeile*⟩]s/⟨*reg. Ausdruck*⟩/⟨*Ersetzung*⟩[/g]

Die in eckigen Klammern stehenden Parameter sind optional, müssen also nicht angeführt werden. Was bedeuten nun die einzelnen Bestandteile des Befehls?

Textbereich

⟨*Startzeile*⟩ und ⟨*Endzeile*⟩ legen den Bereich fest, in dem die Ersetzung durchgeführt werden soll. Ohne diese Angabe wird nur die aktuelle Zeile bearbeitet! Außer Zeilennummern sind zum Beispiel auch ein Punkt als Symbol für die aktuelle sowie ein Dollarzeichen für die letzte Zeile zulässig – bitte diese Zeichen in dieser Funktion nicht mit denselben Zeichen in regulären Ausdrücken (Abschnitt 7.1) verwechseln:

```
:5,$s/rot/blau/
```

ersetzt in jeder Zeile das erste Vorkommen von rot durch blau, wobei die ersten vier Zeilen nicht berücksichtigt werden.

```
:5,$s/rot/blau/g
```

Tabelle 5.5: ex-Kommandos von vi

Kommando	Wirkung
: w ⟨*Dateiname*⟩	schreibt den kompletten Pufferinhalt in die angegebene Datei
: w! ⟨*Dateiname*⟩	schreibt den Pufferinhalt auch in eine schreibgeschützte Datei
: e ⟨*Dateiname*⟩	liest die angegebene Datei in den Puffer
: e #	liest die letzte Datei in den Puffer
: r ⟨*Dateiname*⟩	fügt den Inhalt der angegebenen Datei hinter der aktuellen Zeile ein
: ! ⟨*Shellkommando*⟩	führt das angegebene Shellkommando aus und kehrt danach zu vi zurück
: r! ⟨*Shellkommando*⟩	fügt die Ausgabe des Kommandos hinter der aktuellen Zeile ein
: s/⟨*Muster*⟩/⟨*Ersatz*⟩	sucht nach dem Suchmuster ⟨*Muster*⟩ und ersetzt es durch ⟨*Ersatz*⟩
: q	beendet vi
: q!	beendet vi ohne Rückfrage
: x oder : wq	sichert den Pufferinhalt und beendet danach vi

ersetzt in denselben Zeilen *jedes* Vorkommen von rot durch blau. (Passen Sie auf: Auch die Brotmaschine wird zur Bblaumaschine.)

Statt Zeilennummern, ».« und »$« sind als Start- und Endangaben auch reguläre Ausdrücke in Schrägstrichen erlaubt:

```
:/^ANFANG/,/^ENDE/s/rot/blau/g
```

ersetzt rot durch blau nur in Zeilen, die zwischen einer Zeile mit ANFANG am Anfang und einer mit ENDE am Anfang stehen.

Nach dem Kommandonamen s und einem Schrägstrich müssen Sie den gesuchten regulären Ausdruck angeben. Nach einem weiteren Schrägstrich steht unter ⟨*Ersetzung*⟩ die Zeichenfolge, mit welcher der ursprüngliche Text überschrieben werden soll.

Rückbezug auf Gefundenes

Für dieses Argument existiert eine spezielle Sonderfunktion: mit einem & kann auf die Zeichenkette verwiesen werden, auf die der gesuchte reguläre Ausdruck im konkreten Fall gepasst hat, d. h. »:s/Zebra/&fink« verwandelt jedes Zebra im Suchbereich in einen Zebrafinken – eine Aufgabe, an der die Gentechnologie noch einige Zeit scheitern dürfte.

Befehle im Kommandozeilenmodus Im bisherigen Text haben wir schon einige Befehle des Kommandozeilenmodus, auch als ex-Modus bekannt, beschrieben. Neben diesen existieren noch einige andere, die alle aus dem Kommandomodus heraus mit einem Doppelpunkt eingeleitet und der Eingabetaste abgeschlossen werden müssen (Tabelle 5.5).

Übungen

5.2 [5] (Für Systeme mit vim, z. B. die SUSE-Distributionen.) Finden Sie heraus, wie die interaktive vim-Einführung aufgerufen wird, und arbeiten Sie sie durch.

5.3 Der Herausforderer – Emacs

5.3.1 Überblick

Der Texteditor emacs (»editing macros«[1]) wurde von Richard M. Stallman ursprünglich als Erweiterung des antiquierten Editors TECO entwickelt[2]. Später schrieb Stallman unter diesem Namen ein ganz neues Programm, das ohne TECO auskommt. Ganz dem GNU-Gedanken entsprechend ist der Quelltext des Editors frei verfügbar. Dadurch lässt sich das Programm flexibel an die Wünsche der Benutzer anpassen oder bei Bedarf ergänzen. Der Emacs steht unter fast jeder Plattform zur Verfügung.

Außerdem gibt es Dutzende von Emacs inspirierte Editoren, die in der Regel Funktionalität und Erweiterbarkeit gegen reduzierten Platzbedarf tauschen. Der bekannteste dürfte der »MicroEmacs« sein, der leider schon seit einer Weile nicht mehr weiterentwickelt wird. Ein anderer bekannter Emacs-Ableger ist zum Beispiel der jove. Auch die Emacs-Epigonen – oder zumindest jeweils einige – gibt es nicht nur auf Linux-Rechnern, sondern auf allen nennenswerten Plattformen inklusive Microsoft Windows.

Emacs-Ableger

Emacs selbst bietet wesentlich mehr Funktionen als vi und kann getrost als eigene Arbeitsumgebung angesehen werden, insbesondere da er in einem Dialekt der Programmiersprache Lisp erweitert werden kann. Eine Vollinstallation von Emacs erlaubt es, neben der eigentlichen Funktion als Texteditor auch Dateioperationen wie Kopieren, Löschen etc. durchzuführen, E-Mails zu senden und zu verwalten, einen Kalender oder einen Taschenrechner aufzurufen, Tetris zu spielen, sich einer Psychoanalyse zu unterziehen[3] und vieles mehr. Dieser Funktionsumfang und die weite Verbreitung des Emacs bedingen, dass dieses Programm oft schon als neuer Standardeditor in Linux-Systemen gilt[4].

Diverse Funktionen

Für die LPI-Zertifizierung ist nur der vi von Bedeutung. Wenn Sie also nur für die Prüfung lernen wollen und nicht an einer Horizont-Erweiterung interessiert sind, können Sie den Rest dieses Abschnitts getrost überspringen.

5.3.2 Grundlegende Funktionen

Puffer Ebenso wie vi ist emacs bildschirmorientiert aufgebaut und arbeitet mit Puffern, hält also die zu bearbeitende Textdatei komplett im Arbeitsspeicher und führt Änderungen zunächst nur dort aus. Im Unterschied zu vi kann emacs mehrere Puffer gleichzeitig verwalten, also können mehrere Dateien gleichzeitig bearbeitet und Text zwischen den Dateien ausgetauscht werden. Auch die unterste Zeile der Bildschirmdarstellung, in der Befehle angenommen sowie Benutzerinformationen ausgegeben werden, hat ihren eigenen Mini-Puffer.

mehrere Puffer

Da emacs keine unterschiedlichen Arbeitsmodi kennt, müssen Sie hier nicht in einen Kommandomodus wechseln, um Editorbefehle ausführen zu können. Statt dessen werden die Editorfunktionen mit der Taste Strg in Kombination mit anderen Tasten wie etwa x oder c sowie einer weiteren, frei definierbaren Taste, in der Regel Esc, aufgerufen.

[1] Es gibt auch diverse andere scherzhafte Expansionen, etwa »Escape-Meta-Alt-Control-Shift« für die Vorliebe des Programms für spezielle Tastenkombinationen, oder »*Eight Megabytes and Constantly Swapping*«; als der Emacs neu war, waren 8 Megabyte RAM noch eine Menge, so dass man das ganze heute als »*Eight Hundred Megabytes …*« liest – und bis es »*Eight Thousand*« heißt, kann es wohl auch nicht mehr lange dauern …

[2] TECO war berühmt für seine absolut unmögliche Benutzungsschnittstelle. Wenn Ihnen der vi konterintuitiv vorkommt, dann sollten Sie nicht mal daran denken, sich mit TECO zu beschäftigen. Ein beliebter Sport unter den TECO-Anwendern der späten Siebziger Jahre war es, den eigenen Vornamen als TECO-Kommando einzugeben und vorherzusagen, was wohl passieren würde – nicht immer eine leichte Aufgabe.

[3] Ehrlich! Versuchen Sie mal Esc x tetris ↩ oder Esc x doctor ↩.

[4] Emacs-Verfechter behaupten, dass der einzige Existenzzweck des vi darin besteht, das Makefile für den Emacs zu editieren.

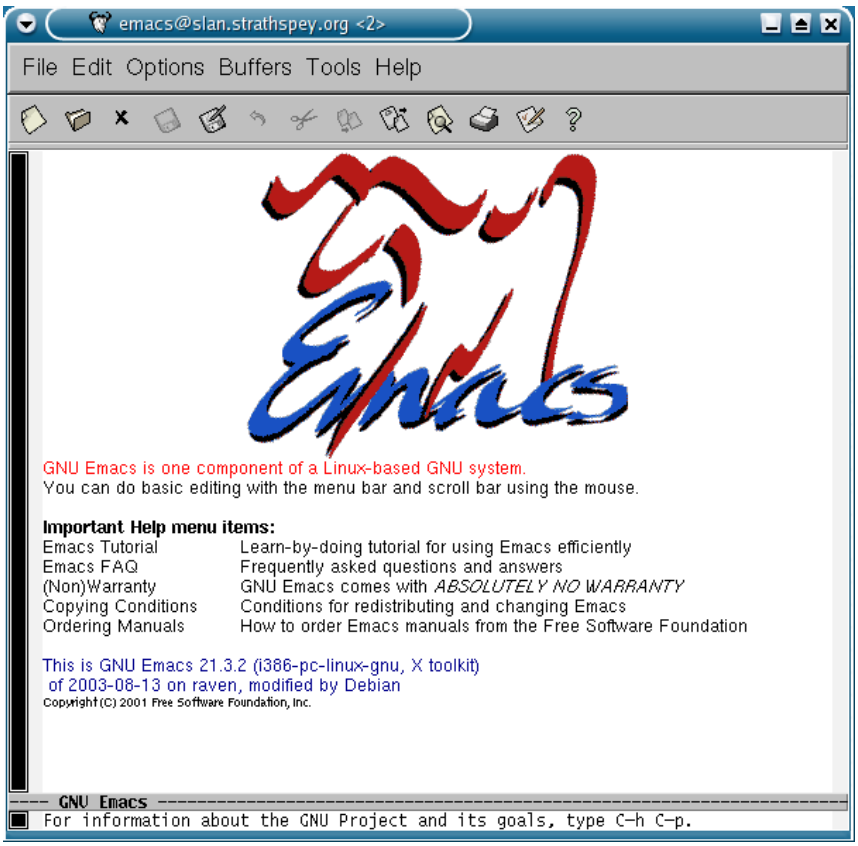


Bild 5.3: Der `emacs`-Startbildschirm

Tabelle 5.6: Mögliche Pufferzustände in `emacs`

Zeichenfolge	Pufferzustand
`-:---`	Inhalt seit dem letzten Speichern unverändert
`-:**-`	Inhalt verändert, aber noch nicht gespeichert
`-:%%-`	Dateiinhalt ist nur lesbar, nicht änderbar

Im Emacs-Jargon heißt die Funktion der Esc-Taste »Meta«. Sie können dafür auch eine der zusätzlichen umstellerartigen Tasten verwenden, an denen die heutigen PC-Tastaturen so reich sind. Typischerweise ist die Alt-Taste links neben der Leertaste entsprechend vorkonfiguriert.

Start, Ende und Hilfe Nach dem Laden von `emacs` erscheint zunächst ein Startbildschirm mit einigen wichtigen Hinweisen. Dieser sieht in etwa so aus wie in Bild 5.3 zu sehen. Während in der Handbuchseite nur die Startparameter für `emacs` aufgelistet sind, kann innerhalb des Editors mit Strg+h das eingebaute Hilfesystem aufgerufen werden. Dort findet sich alles, was das Herz begehrt, von Tastaturbefehlen bis zu Suchmustern und vieles mehr. Auch ein *tutorial*, also ein Lernprogramm, ist vorhanden, das den Anwender Schritt für Schritt mit den Grundlagen der Bedienung von `emacs` vertraut macht. Startbildschirm

Die oberste Bildschirmzeile zeigt eine Menüleiste, die jedoch ohne grafische Oberfläche nicht mit der Maus angesprochen werden kann. Da deren Tastaturbedienung sehr umständlich, die Bedienung mit der Maus dagegen simpel und nicht erklärenswert ist, wird die Menüzeile hier nicht weiter betrachtet. Menüleiste

In der vorletzten Zeile des Bildschirms finden Sie den Namen des aktuellen Puffers, der in der Regel dem Namen der geladenen Datei entspricht. Wurde beim Start von `emacs` kein Dateiname angegeben, nennt der Editor den Puffer `*scratch*`. Statuszeile

Daneben werden auch Informationen über die aktuelle Zeilennummer (hinter L, hier also Zeile 1) sowie die Position des sichtbaren Teils des Textes in Relation zum Text insgesamt (hier All, also der gesamte Text, sonst ein Prozentwert) angezeigt. Am Zeilenanfang kann der Zustand des Puffers abgelesen werden (Tabelle 5.6).

Wie bei den meisten Editoren (außer vi) können Sie direkt nach dem Programmstart Text eingeben. Die Schreibmarke lässt sich frei im Text bewegen, einzelne Zeichen können Sie mit der ⇐-Taste löschen, die Eingabetaste fügt eine neue Zeile ein – alles arbeitet wie gewohnt. Reichen Zeilen über den Bildschirmrand hinaus, werden sie am rechten Rand mit einem Rückstrich markiert und in einer neuen Bildschirmzeile fortgesetzt. In der eigentlichen Datei findet dabei kein automatischer Zeilenumbruch statt, die reale Zeilenlänge kann also eine Bildschirmzeile deutlich übertreffen.

Zum Schluss die wohl wichtigsten Angaben des Startbildschirms: Mit der Tastenfolge Strg+x u können noch nicht auf dem Datenträger gespeicherte Änderungen rückgängig gemacht werden. Der Befehl Strg+x Strg+c beendet emacs, wobei das Programm freundlicherweise fragt, ob nicht gesicherte Pufferinhalte abgespeichert werden sollen.

Texte laden und speichern Möchten Sie einen Text im emacs bearbeiten, muss sich dieser im aktuellen Puffer befinden. Zu diesem Zweck dient der Befehl Strg+x Strg+f. Nach Eingabe des Puffernamens, der üblicherweise dem Dateinamen des Textes entspricht, wird der entsprechende Puffer aktiviert. Existiert noch kein Puffer dieses Namens, legt der Editor einen neuen an und versucht, die entsprechende Datei zu laden. Dabei unterstützt emacs die bereits von der Shell bekannte Vervollständigung von Dateinamen durch die Tab-Taste.

Mit dem Kommando Strg+x Strg+v wird für die angegebene Datei kein neuer Puffer angelegt. Stattdessen wird der Inhalt des aktuellen Puffers überschrieben und der Puffername angepasst. Dabei geht natürlich der ursprüngliche Pufferinhalt verloren.

Weiterhin können Sie mit Strg+x i eine Datei an der aktuellen Cursorposition in einen Pufferinhalt einfügen. Der Inhalt des aktuellen Puffers wird mit Strg+x Strg+s gespeichert. Hat der Puffer noch keinen Namen, verlangt das Programm eine solche Angabe, ansonsten erscheint direkt die Meldung »Token-Wrote ⟨*Dateiname*⟩« in der Mini-Puffer-Zeile. Beim ersten Speichervorgang wird die ursprüngliche Datei durch eine an den Dateinamen angehängte Tilde zur Sicherungskopie umfunktioniert.

Soll der Inhalt eines bereits benannten Puffers unter einer anderen Bezeichnung abgespeichert werden, müssen Sie das Kommando Strg+x Strg+w benutzen. Das Programm verlangt dann die Angabe des neuen Dateinamens und lässt die ursprüngliche Datei unbehelligt.

Zwischen Puffern können Sie mit dem Kommando Strg+x b hin und her wechseln. Das Kommando Strg+x Strg+b zeigt eine Pufferliste.

Bewegung im Text Im Regelfall dienen die Cursortasten dazu, sich zeichenweise durch den Text zu bewegen. Besonders ältere Terminals verfügen oftmals nicht über solche Tasten, daher existieren gleichbedeutende Tastaturbefehle. Auf diese Weise können Sie auch wort- oder satzweise durch den Text springen (Tabelle 5.7).

Löschen von Zeichen Um ein beliebiges Zeichen zu entfernen, müssen Sie zunächst der Cursor an die jeweilige Stelle bringen. Dann stehen die in Tabelle 5.8 aufgeführten Kommandos zur Auswahl.

5.3.3 Erweiterte Funktionen

Textteile ausschneiden und einfügen Sobald mehr als ein einzelnes Zeichen gelöscht wurde, befindet sich diese Zeichenkette im sogenannten »*kill*«-Puffer. In Wahrheit hat also statt eines Löschvorgangs ein Ausschneiden stattgefunden.

Tabelle 5.7: Tastaturbefehle zur Cursorpositionierung in `emacs`

Tastaturkommando	Wirkung
Strg + f oder →	ein Zeichen nach rechts (engl. *forward*, »vorwärts«)
Strg + b oder ←	ein Zeichen nach links (engl. *back*, »zurück«)
Strg + n oder ↓	eine Zeile nach unten (engl. *next*, »nächste«)
Strg + p oder ↑	eine Zeile nach oben (engl. *previous*, »vorige«)
Esc f	auf Leerzeichen vor nächstem Wort springen
Esc b	auf erstes Zeichen im vorherigen Wort springen
Strg + a	zum Zeilenanfang springen
Strg + e	zum Zeilenende springen
Esc a	zum Satzanfang springen
Esc e	zum Satzende springen
Strg + v	eine Bildschirmseite weiterblättern
Esc v	eine Bildschirmseite zurückblättern
Esc <	an den Textanfang springen
Esc >	an das Textende springen

Tabelle 5.8: Tastaturbefehle zum Löschen von Zeichen in `emacs`

Tastaturkommando	Wirkung
Strg + d	löscht das Zeichen »unter« dem Cursor (engl. *delete*, »löschen«)
Esc d	löscht vom Zeichen unter dem Cursor bis zum Ende des Worts, zu dem das Zeichen gehört
Strg + k	löscht von der Cursorposition bis zum Zeilenende (engl. *kill*, »umbringen«). Ein zweites Strg + k löscht das Zeilenende selbst.
Esc k	löscht den Satz, in dem der Cursor steht
Strg + w	löscht den Bereich zwischen der Marke (gesetzt mit Strg + ␣) und der Cursorposition

Tabelle 5.9: Tastaturbefehle zur Textkorrektur in emacs

Tastaturkommando	Wirkung
Strg+t	tauscht das Zeichen unter dem Cursor mit dem unmittelbar vorigen (steht der Cursor am Zeilenende, wird das vorige Zeichen mit dem vorvorigen vertauscht)
Esc t	tauscht das Wort, das vor dem Cursor anfängt, mit dem, das nach dem Cursor anfängt. Interpunktion wird dabei nicht bewegt.
Strg+x Strg+t	tauscht die aktuelle mit der vorhergehenden Zeile
Esc c	schreibt den Buchstaben unter dem Cursor groß und alle folgenden im Wort klein
Esc u	schreibt alle Buchstaben ab der Cursorposition im Wort groß
Esc l	schreibt alle Buchstaben ab der Cursorposition im Wort klein

Der Inhalt des »*kill*«-Puffers können Sie nun mit Strg+y hinter der Cursorposition in den aktuellen Text einfügen. Auch ältere Zeichenketten stehen noch zur Verfügung. Durch Eingabe von Esc y wird der eingefügte Text durch den jeweils nächstälteren Inhalt des »*kill*«-Puffers ersetzt.

Korrektur von Tippfehlern Besonders weniger geübten Maschinenschreibern passiert es ab und an, dass zwei Buchstaben in der falschen Reihenfolge oder Großbuchstaben klein getippt werden – diese Aufzählung ließe sich beliebig fortsetzen. Zur Behebung solcher Fehler hält emacs einige Kommandos bereit, die in Tabelle 5.9 beschrieben sind.

Textteile suchen Aufgrund des großen Funktionsumfangs der Textsuche im emacs werden hier nur die beiden grundlegenden Suchmöglichkeiten vorgestellt: Der Befehl Strg+s erlaubt die Suche ab der Cursorposition zum Textende hin, während Strg+r in die umgekehrte Richtung, also aufwärts, sucht. Die gesuchte Zeichenkette wird im Mini-Puffer angezeigt, wobei bereits während der Eingabe der Cursor an die nächste passende Stelle des Textes springt.

Weitere Funktionen Wie eingangs erwähnt, besitzt der Emacs einen enormen Funktionsumfang, der weit über die Anforderungen an einen Texteditor hinausgeht. Einige dieser Besonderheiten sollen hier genannt werden:

Unterstützung bestimmter Dateitypen
- emacs ist in der Lage, anhand der Endung des Dateinamens bestimmte Dateitypen zu erkennen. Wird etwa ein C-Programmquelltext mit der Endung `.c` geöffnet, wechselt der Editor umgehend in den C-Modus. Hier werden automatische Einrückungen sowie Klammerprüfungen vorgenommen und so dem Programmierer die Arbeit erleichtert. Selbst der Aufruf des Compilers ist direkt aus dem Editor möglich. Dasselbe gilt für die meisten anderen gängigen Programmiersprachen.

Neue Mail
- Erhalten Sie während der Arbeit mit emacs eine E-Mail, wird dies umgehend in der Statuszeile gemeldet. Der Befehl Esc x `rmail` wechselt in den Mailmodus, wo Sie Nachrichten lesen, verfassen und senden können.

Verzeichniseditor
- Dateioperationen können Sie aus emacs heraus mit »dired« (engl. *directory editor*, »Verzeichniseditor«) durchführen. Die Eingabe von Strg+x d zeigt eine Liste aller Dateien im aktuellen Verzeichnis, durch die Sie sich mit den

Cursortasten frei bewegen können. Durch Betätigen der Eingabetaste wird die mit dem Cursor markierte Datei in den Editor geladen. Weiterhin sind auch das Löschen, Kopieren oder Umbenennen von Dateien möglich.

- Mit einem Dialekt der Programmiersprache Lisp, die ursprünglich zur flexiblen Programmierung künstlicher Intelligenz entwickelt worden war, können im `emacs` Makros erstellt werden. Da es sich um eine listenorientierte Programmiersprache handelt, bestehen kaum Ähnlichkeiten zu anderen Makrosprachen, es handelt sich aber um ein sehr mächtiges Werkzeug. Interessierte Anwender seien auf das umfangreiche Handbuch zu Emacs-Lisp verwiesen. Emacs-Lisp

Übungen

5.3 [5] Arbeiten Sie die interaktive Emacs-Einführung durch. (Die englischsprachige Version erhalten Sie über Strg+h t; eine deutschsprachige ist über das »Help«-Menü des Emacs zu erreichen.)

5.4 Andere Editoren

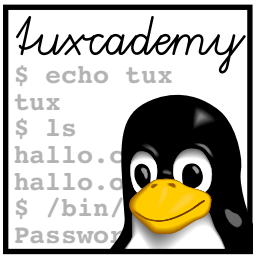
Wir haben schon durchblicken lassen, dass Ihre Editor-Auswahl ebensoviel mit Ihren persönlichen Vorlieben zu tun hat und wohl auch ebensoviel über Sie als Benutzer aussagt wie die Wahl Ihres Autos: Fahren Sie einen polierten BMW oder reicht Ihnen ein verbeulter Golf? Oder muss es doch ein Land-Rover sein? Was die Wahlmöglichkeiten angeht, so steht der Editor-Markt dem Automarkt wohl kaum nach. Wir haben Ihnen zwei nicht ganz unbedeutende Kandidaten präsentiert, aber es gibt natürlich auch noch einige andere. `kate` unter KDE und `gedit` unter GNOME zum Beispiel sind übersichtliche und leicht erlernbare Editoren mit einer grafischen Oberfläche, die für die Aufgaben eines »normalen Benutzers« sicher ausreichen. Durchstöbern Sie ruhig einmal die Paketlisten Ihrer Distribution und schauen Sie, ob Sie dort den Editor Ihrer Träume finden können.

Kommandos in diesem Kapitel

`ed`	Primitiver zeilenorientierter Editor	`ed`(1)	57
`elvis`	Populärer „Klon“ des `vi`-Editors	`elvis`(1)	56
`emacs`	Leistungsfähiger bildschirmorientierter Editor	`emacs`(1), Info: `emacs`	63
`ex`	Leistungsfähiger zeilenorientierter Editor (eigentlich `vi`)	`vi`(1)	57
`jove`	Von `emacs` inspirierter Editor	`jove`(1)	64
`sed`	Datenstromorientierter Texteditor, kopiert Eingabe unter Änderungen auf Ausgabe	`sed`(1)	57
`vi`	Bildschirmorientierter Texteditor	`vi`(1)	56
`vim`	Populärer „Klon“ des `vi`-Editors	`vim`(1)	56

Zusammenfassung

- Texteditoren sind wichtig zum Ändern von Konfigurationsdateien und zum Programmieren. Sie bieten oft besondere Eigenschaften, die diese Aufgaben erleichtern.
- Der `vi` ist ein traditioneller, sehr verbreiteter und leistungsfähiger Editor mit eigenwilliger Bedienung.
- Der Emacs ist ein modernerer Editor mit vielen Sonderfunktionen, der als freie Software zur Verfügung steht.



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 (siehe Kapitel 2)
- Umgang mit einem Texteditor (siehe Kapitel 5)

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 4 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.

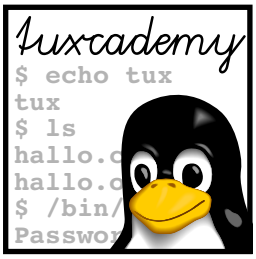
6.30 [1] Überzeugen Sie sich, dass `slocate` funktioniert, indem Sie als gewöhnlicher Benutzer nach Dateien wie `/etc/shadow` suchen.

Kommandos in diesem Kapitel

cd	Wechselt das aktuelle Arbeitsverzeichnis der Shell	bash(1)	75
convmv	Konvertiert Dateinamen zwischen Zeichenkodierungen	convmv(1)	73
cp	Kopiert Dateien	cp(1)	82
find	Sucht nach Dateien, die bestimmte Kriterien erfüllen	find(1), Info: find	90
less	Zeigt Texte (etwa Handbuchseiten) seitenweise an	less(1)	90
ln	Stellt („harte“ oder symbolische) Links her	ln(1)	85
locate	Sucht Dateien über ihren Namen in einer Dateinamensdatenbank	locate(1)	94
ls	Listet Dateien oder den Inhalt von Verzeichnissen auf	ls(1)	76
mkdir	Legt neue Verzeichnisse an	mkdir(1)	78
more	Zeigt Textdaten seitenweise an	more(1)	90
mv	Verschiebt Dateien in andere Verzeichnisse oder benennt sie um	mv(1)	84
pwd	Gibt den Namen des aktuellen Arbeitsverzeichnisses aus	pwd(1), bash(1)	76
rm	Löscht Dateien oder Verzeichnisse	rm(1)	84
rmdir	Entfernt (leere) Verzeichnisse	rmdir(1)	78
slocate	Sucht Dateien über ihren Namen in einer Datenbank und beachtet dabei deren Zugriffsrechte	slocate(1)	96
updatedb	Erstellt die Dateinamensdatenbank für locate	updatedb(1)	95
xargs	Konstruiert Kommandozeilen aus seiner Standardeingabe	xargs(1), Info: find	93

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 5)

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).

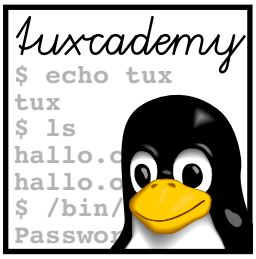
7.5 [4] Geben Sie ein Kommando an, das im »Froschkönig« alle Zeilen sucht und ausgibt, in denen irgendein mindestens vierbuchstabiges Wort zweimal auftritt.

Kommandos in diesem Kapitel

`egrep`	Sucht in Dateien nach Zeilen mit bestimmtem Inhalt, erweiterte reguläre Ausdrücke erlaubt	`grep`(1)	102
`fgrep`	Sucht in Dateien nach Zeilen bestimmten Inhalts, keine regulären Ausdrücke erlaubt	`fgrep`(1)	102
`grep`	Sucht in Dateien nach Zeilen mit bestimmtem Inhalt	`grep`(1)	101

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 (Kapitel 2)
- Umgang mit einem Texteditor (Kapitel 5)
- 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`

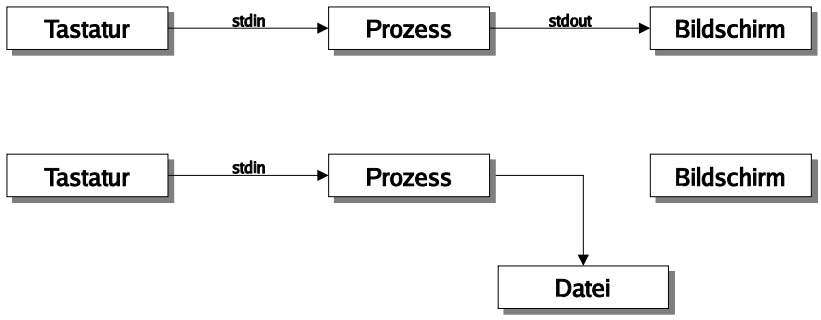


Bild 8.1: Standardkanäle unter Linux

und `stderr` für die Standard-Eingabe, Standard-Ausgabe und Standard-Fehlerausgabe. 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.

Standard-Eingabe umleiten

Mit <, dem »Kleiner-Als«-Zeichen, können Sie den Standard-Eingabe-Kanal umleiten. Nun wird statt der Tastatureingabe der Inhalt der angegebenen Datei ausgewertet:

```
$ 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.

Mit dem <<-Operator können Sie allerdings Eingabedaten für ein Programm direkt aus den Zeilen übernehmen, die dem Programmaufruf in der Shell folgen. Dies ist für den interaktiven Einsatz weniger interessant als für den Gebrauch in Shellskripten, soll aber der Vollständigkeit halber trotzdem hier erwähnt werden. Man spricht von »Hier-Dokumenten« (engl. *here documents*). Im Beispiel

```
$ grep Linux <<ENDE
Rosen sind rot,
Veilchen sind blau,
Linux ist super,
das weiß ich genau.
ENDE
```

besteht die Eingabe von grep aus den Folgezeilen bis zu der Zeile mit dem Inhalt »ENDE«. Die Ausgabe des Kommandos ist

```
Linux ist super,
```

Wird bei einem Hier-Dokument die Zeichenkette, die das Ende des Hier-Dokuments bezeichnet, ohne Anführungszeichen angegeben, so werden in den Zeilen des Hier-Dokuments Shellvariable ersetzt und Kommandosubstitution (mit `…` oder $(…)) vorgenommen. Steht die End-Zeichenkette allerdings in Anführungszeichen (egal ob einfachen oder doppelten), bleiben die Zeilen des Hier-Dokuments so, wie sie sind. Vergleichen Sie die Ausgabe von

```
$ cat <<EOF
Heutiges Datum: `date`
EOF
```

mit der von

```
$ cat <<"EOF"
Heutiges Datum: `date`
EOF
```

Zu guter Letzt: Wird das Hier-Dokument nicht mit »<<<«, sondern mit »<<-« eingeleitet, so werden alle Tabulatorzeichen am Anfang jeder Zeile des Hier-Dokuments entfernt. Dies macht es möglich, Hier-Dokumente in Shellskripten sinnvoll einzurücken.

kombinierte Umleitung

Selbstverständlich ist auch die kombinierte Umleitung von Ein- und Ausgabekanal zulässig. Die Ausgabe des Wortzähl-Beispiels wird hier in eine Datei namens `wortzahl` geschrieben:

```
$ wc -w <frosch.txt >wortzahl
$ cat wortzahl
1255
```

Standard-Fehler-Kanal

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.

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

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:

direkte Verknüpfung mehrerer Befehle

Pipeline

```
$ 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
```

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 ./
```

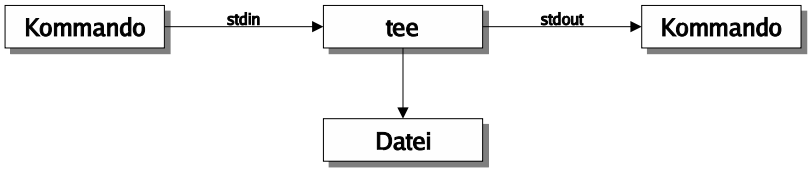


Bild 8.2: Das Kommando `tee`

```
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

Werkzeugkastenprinzip

Eine der Grundlagen von Unix – und damit Linux – ist das »Werkzeugkastenprinzip«. Das System verfügt über eine große Menge von Systemprogrammen, 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.

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`.

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` und `tac`

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).

Textdateien

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.

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`

Tabelle 8.3: Optionen für `tac` (Auswahl)

Option	Wirkung
`-b`	(engl. *before*) Die Eingabe wird so interpretiert, dass der Trenner jeweils *vor* einem Teil steht (und ausgegeben wird), nicht dahinter.
`-r`	(engl. *regular expression*) Der Trenner wird als regulärer Ausdruck interpretiert.
`-s` *s*	(engl. *separator*) Gibt einen anderen Trenner *s* (statt `\n`) an. Der Trenner darf mehrere Zeichen lang sein.

`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. Die ganze Angelegenheit ist aber etwas subtiler; siehe hierzu Übung 8.21.

Zeilen in umgekehrter Reihenfolge ausgeben

Das Kommando `tac` ist vom Namen her »`cat` rückwärts« und funktioniert auch so: Es liest als Parameter benannte Dateien oder seine Standardeingabe und gibt die gelesenen Zeilen in *umgekehrter* Reihenfolge wieder aus:

```
$ tac <<ENDE
Alpha
Beta
Gamma
ENDE
Gamma
Beta
Alpha
```

Trenner

Da hört die Ähnlichkeit aber auch schon auf: `tac` unterstützt nicht dieselben Optionen wie `cat`, sondern hat ein paar eigene (Tabelle 8.3). Zum Beispiel können Sie mit der Option `-s` einen alternativen Trenner einstellen, an dem `tac` sich beim Umdrehen der Eingabe orientiert – normal ist das Trennzeichen der Zeilentrenner, so dass zeilenweise umgedreht wird. Siehe etwa:

```
$ echo A:B:C:D | tac -s :
D
C:B:A:$ _
```

(mit der neuen Eingabeaufforderung direkt an die letzte Zeile geklebt). Diese auf den ersten Blick total eigenartige Ausgabe erklärt sich wie folgt: Die Eingabe besteht aus den vier Teilen »`A:`«, »`B:`«, »`C:`« und »`D\n`« (der Trenner, hier »`:`«, gilt als an den jeweils vorhergehenden Teil angehängt, und der Zeilentrenner `\n` wird von `echo` beigesteuert). Diese Teile werden in umgekehrter Reihenfolge ausgegeben, das heißt, zuerst kommt »`D\n`« und dann die anderen drei, jeweils ohne irgendein zusätzliches Trennzeichen dazwischen (es ist ja schon ein völlig zufriedenstellender Trenner vorhanden); die nächste Eingabeaufforderung der Shell wird direkt (ohne Schaltung in die nächste Zeile) an die Ausgabe angehängt. Die Option `-b` betrachtet den Trenner nicht als an den vorhergehenden Teil angehängt, sondern als vor den nachfolgenden Teil gestellt; unser Beispiel würde unter »`tac -s : -b`« also die Ausgabe

```
:D
:C:BA$ _
```

liefern (denken Sie's durch!).

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
```

Tabelle 8.4: Optionen für od (Auszug)

Option	Wirkung
`-A` *r*	Basis der Positionsangabe am Zeilenanfang. Gültige Werte sind: `d` (dezimal), `o` (oktal), `x` (hexadezimal), `n` (keine Positionsangabe ausgeben).
`-j` *o*	Am Anfang der Eingabe werden *o* Bytes übersprungen und erst dann mit der Ausgabe begonnen.
`-N` *n*	Maximal *n* Bytes ausgeben.
`-t` *t*	Ausgabe gemäß Typangabe *t*. Es können mehrere `-t`-Optionen auftreten, dann wird für jede der `-t`-Optionen eine Zeile im entsprechenden Format ausgegeben. Möglichkeiten für *t*: `a` (benanntes Zeichen), `c` (ASCII-Zeichen), `d` (Dezimalzahl mit Vorzeichen), `f` (Gleitkommazahl), `o` (Oktalzahl), `u` (vorzeichenlose Dezimalzahl), `x` (Hexadezimalzahl). An die Optionen außer `a` und `c` können Sie eine Ziffer anhängen, die angibt, wie viele Bytes der Eingabe jeweils gemeinsam interpretiert werden sollen. Details hierfür und für Breitenangaben durch Buchstaben finden sich in od(1). Hängen Sie an eine Option ein `z` an, werden rechts auf der Zeile die druckbaren Zeichen der Zeile ausgegeben.
`-v`	Gibt auch doppelt vorkommende Zeilen komplett aus.
`-w` *w*	Gibt *w* Bytes pro Zeile aus; Standardwert ist 16.

wenn Sie nach dem ersten `echo` in einem anderen Fenster das Kommando

```
$ tail -f /tmp/hallo
```

gestartet haben.

8.3.3 Mit der Lupe – od und hexdump

`cat`, `tac`, `head` und `tail` funktionieren am besten mit Textdateien: Beliebige Binärdaten können zwar prinzipiell verarbeitet werden, aber gerade die letzteren drei Programme gehen standardmäßig am liebsten mit Dateien um, die erkennbare Zeilen haben. Trotzdem ist es ab und zu nützlich, sich vergewissern zu können, welche Daten nun ganz genau in einer Datei stehen. Ein Hilfsmittel dafür ist das

od

Kommando `od` (kurz für *octal dump*), das beliebige Daten in unterschiedlichen Formaten anzeigen kann. Binärdaten lassen sich zum Beispiel byteweise oder wortweise in oktaler, hexadezimaler, dezimaler oder ASCII-Darstellung präsentieren. Die Standard-Darstellungsweise von `od` ist wie folgt:

```
$ od /etc/passwd | head -3
0000000 067562 072157 074072 030072 030072 071072 067557 035164
0000020 071057 067557 035164 061057 067151 061057 071541 005150
0000040 060563 064163 067562 072157 074072 030072 030072 071072
$ _
```

Zeilenformat

Ganz links steht die (oktale) Position in der Datei, an der die Ausgabezeile beginnt. Die acht folgenden Zahlen entsprechen jeweils zwei Bytes aus der Datei, als Oktalzahl interpretiert. Nützlich ist das nur unter ganz speziellen Umständen.

Zum Glück unterstützt `od` Optionen, mit denen Sie das Format der Ausgabe in

Option `-t`

sehr weiten Grenzen ändern können (Tabelle 8.4). Am wichtigsten ist die Option `-t`, die angibt, wie die `od`-Datenzeilen aussehen sollen. Für byteweise hexadezimale Ausgabe können Sie zum Beispiel

```
$ od -txC /etc/passwd
0000000 72 6f 6f 74 3a 78 3a 30 3a 30 3a 72 6f 6f 74 3a
0000020 2f 72 6f 6f 74 3a 2f 62 69 6e 2f 62 61 73 68 0a
```

```
0000040 73 61 73 68 72 6f 6f 74 3a 78 3a 30 3a 30 3a 72
◁◁◁◁◁
```

verwenden (die Positionsangabe bleibt oktal). Dabei steht das `x` für »hexadezimal« und das `C` für »byteweise«. Wenn Sie außer den Hexadezimalzahlen noch die entsprechenden Zeichen sehen wollen, können Sie ein `z` anhängen:

```
$ od -txCz /etc/passwd
0000000 72 6f 6f 74 3a 78 3a 30 3a 30 3a 72 6f 6f 74 3a  >root:x:0:0:root:<
0000020 2f 72 6f 6f 74 3a 2f 62 69 6e 2f 62 61 73 68 0a  >/root:/bin/bash.<
0000040 73 61 73 68 72 6f 6f 74 3a 78 3a 30 3a 30 3a 72  >sashroot:x:0:0:r<
◁◁◁◁◁
```

Nicht druckbare Zeichen (hier das `0a` – ein Zeilentrenner – am Ende der zweiten Zeile) werden durch ».« vertreten.

Sie können auch mehrere Typangaben hintereinandergehängt oder in separaten `-t`-Optionen angeben. Dann bekommen Sie eine Zeile pro Typangabe: mehrere Typangaben

```
$ od -txCc /etc/passwd
0000000 72 6f 6f 74 3a 78 3a 30 3a 30 3a 72 6f 6f 74 3a
          r   o   o   t   :   x   :   0   :   0   :   r   o   o   t   :
0000020 2f 72 6f 6f 74 3a 2f 62 69 6e 2f 62 61 73 68 0a
          /   r   o   o   t   :   /   b   i   n   /   b   a   s   h  \n
0000040 73 61 73 68 72 6f 6f 74 3a 78 3a 30 3a 30 3a 72
          s   a   s   h   r   o   o   t   :   x   :   0   :   0   :   r
◁◁◁◁◁
```

(was dasselbe wäre wie »`od -txC -tc /etc/passwd`«).

Eine Folge von Zeilen, die in der Ausgabe identisch zu der unmittelbar davorstehenden Zeile wären, wird durch einen Stern (»*«) am linken Rand abgekürzt: identische Ausgabezeilen

```
$ od -tx -N 64 /dev/zero
0000000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
0000100
```

(`/dev/zero` liefert Nullen in beliebigen Mengen, und die `od`-Option `-N` beschränkt die Ausgabe auf 64 davon.) Mit der Option `-v` findet diese Abkürzung nicht statt:

```
$ od -tx -N 64 -v /dev/zero
0000000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000100
```

Eine ganz ähnliche Leistung erbringt das Programm `hexdump` (oder kurz `hd`). Zunächst unterstützt es Ausgabeformate, die denen von `od` sehr ähneln, auch wenn die dafür nötigen Optionen völlig anders heißen. So entspricht das Kommando `hexdump`

```
$ hexdump -o -s 446 -n 64 /etc/passwd
```

im Wesentlichen dem ersten Beispiel für `od` weiter oben. Die meisten Optionen von `od` haben ziemlich direkte Äquivalente in `hexdump`.

Ein wesentlicher Unterschied zwischen `hexdump` und `od` ist `hexdumps` Unterstützung für Ausgabeformate. Hiermit können Sie wesentlich detaillierter festlegen, wie die Ausgabe aussehen soll, als mit `od`. Betrachten Sie das folgende Beispiel: Ausgabeformate

```
$ cat hexdump.txt
0123456789ABC◁◁◁◁◁ XYZabc◁◁◁◁◁ xyz
$ hexdump -e '"%x-« hexdump.txt
33323130-37363534-◁◁◁◁◁-a7a79-$
```

Die folgenden Punkte sind bemerkenswert:

- Das Ausgabeformat »`"%x"`« gibt 4 Bytes der Eingabedatei in hexadezimaler Form aus – »`30`« ist die hexadezimale Darstellung von 48, dem numerischen Wert des Zeichens »`0`« im ASCII. Das angehängte »`-`« wird so ausgegeben, wie es ist.
- Die 4 Bytes werden in umgekehrter Reihenfolge ausgegeben, das ist ein Artefakt der Intel-Prozessorarchitektur.
- Die doppelten Anführungszeichen sind Bestandteil der Syntax von `hexdump` und müssen darum durch einfache Anführungszeichen (oder etwas Äquivalentes) davor geschützt werden, dass die Shell sie entfernt.
- Das $ am Schluss ist die nächste Eingabeaufforderung der Shell; von sich aus gibt `hexdump` keinen Zeilentrenner aus.

Die möglichen Ausgabeformate richten sich (bequemerweise für Programmierer) nach denen, die von der Funktion `printf`(3) in Programmiersprachen wie C, Perl, `awk` usw. verwendet werden (auch die Bash unterstützt ein `printf`-Kommando). Entnehmen Sie die Details der Dokumentation!

`hexdump`-Ausgabeformate sind deutlich vielschichtiger als das eben gezeigte simple Beispiel. Wie bei `printf` üblich können Sie etwa eine »Feldbreite« für die Ausgabe bestimmen:

```
$ hexdump -e '"%10x-"' hexdump.txt
  33323130-  37363534-  ◁◁◁◁◁ a7a79-$
```

(Hier erscheint jede Sequenz von Hexadezimalzahlen – acht Zeichen lang – rechtsbündig in einem zehn Zeichen breiten Feld.)

Wiederholungszahl

Sie können auch angeben, wie oft ein Format »ausgeführt« wird:

```
$ hexdump -e '4 "%x-" "\n"' hexdump.txt
33323130-37363534-42413938-46454443-
4a494847-4e4d4c4b-5251504f-56555453-
5a595857-64636261-68676665-6c6b6a69-
706f6e6d-74737271-78777675-a7a79-
```

Die Kommandos dieses Abschnitts vorangestellte `4` gibt an, dass das Format »`"%x"-`« viermal angewendet werden soll. Danach geht es mit dem nächsten Format – »`"\n"`« – weiter, das einen Zeilentrenner erzeugt. Anschließend fängt `hexdump` wieder vorne an.

Zeilentrenner

Byteanzahl

Ferner können Sie sagen, wie viele Bytes ein Format verarbeiten soll (bei den numerischen Formaten haben Sie meist die Wahl zwischen 1, 2 und 4):

```
$ hexdump -e '/2 "%x-" "\n"' hexdump.txt
3130-
3332-
◁◁◁◁◁
7a79-
a-
```

(»`/2`« ist hier eine Abkürzung für »`1/2`«, darum erscheint jedes `%x`-Format pro Zeile nur einmal.) Wiederholungs- und Byteanzahl lassen sich kombinieren:

Tabelle 8.5: Optionen für `tr`

Option	Wirkung
`-c` (*complement*)	ersetzt alle Zeichen, die *nicht* in $\langle s_1 \rangle$ stehen, durch Zeichen aus $\langle s_2 \rangle$
`-d` (*delete*)	entfernt alle Zeichen, die in $\langle s_1 \rangle$ stehen, ohne Ersatz
`-s` (*squeeze*)	mehrere gleiche, aufeinander folgende Zeichen aus $\langle s_2 \rangle$ werden zusammengefasst

```
$ hexdump -e '4/2 "%x-" "\n"' hexdump.txt
3130-3332-3534-3736-
3938-4241-4443-4645-
<<<<<
7675-7877-7a79-a-
```

Sie können auch verschiedene Ausgabeformate mischen:

```
$ hexdump -e '"%2_ad" "%2.2s" 3/2 " %x" " %1.1s" "\n"'▷
 ◁ hexdump.txt
0 01 3332 3534 3736 8
9 9A 4342 4544 4746 H
<<<<<
```

Hier geben wir die ersten beiden Zeichen der Datei als Zeichen (nicht als numerische Codes) aus (»`"%2.2s"`«), dann die Codes von dreimal zwei Zeichen in hexadezimaler Schreibweise (»`3/2 " %x"`«), gefolgt von einem weiteren Zeichen als Zeichen (»`"%1.1s"`«) und einem Zeilentrenner. Danach geht es mit dem Format wieder von vorne los. Das »`"%2_ad"`« am Zeilenanfang liefert die aktuelle Position in der Datei (gezählt in Bytes vom Dateianfang) in dezimaler Schreibweise in einem zwei Zeichen breiten Feld.

Übungen

8.12 [2] Was ist der Unterschied zwischen den Typangaben »`a`« und »`c`« von `od`?

8.13 [3] Das »Gerät« `/dev/random` liefert zufällige Bytes (siehe Abschnitt 10.3). Verwenden Sie `od` mit `/dev/random`, um der Shellvariablen `r` eine dezimale Zufallszahl zwischen 0 und 65535 (einschließlich) zuzuweisen.

8.4 Textbearbeitung

8.4.1 Zeichen für Zeichen – `tr`, `expand` und `unexpand`

Der Befehl `tr` dient dazu, innerhalb eines Textes einzelne Zeichen gegen andere auszutauschen oder komplett zu löschen. `tr` ist ein reines Filterkommando, es kann also nicht auf benannte Dateien angewendet werden, sondern arbeitet ausschließlich mit den Standardkanälen.

Austauschoperationen

Für Austauschoperationen lautet die Syntax dieses Kommandos »`tr` $\langle s_1 \rangle$ $\langle s_2 \rangle$«. Die beiden Parameter sind Zeichenketten, die die Austauschoperation beschreiben: Im einfachsten Fall wird das erste Zeichen von $\langle s_1 \rangle$ durch das erste Zeichen von $\langle s_2 \rangle$ ersetzt, das zweite Zeichen von $\langle s_1 \rangle$ durch das zweite Zeichen von $\langle s_2 \rangle$ und so weiter. Ist $\langle s_1 \rangle$ länger als $\langle s_2 \rangle$, so wird für die »überstehenden« Zeichen in $\langle s_1 \rangle$ das letzte Zeichen von $\langle s_2 \rangle$ benutzt; ist $\langle s_2 \rangle$ länger als $\langle s_1 \rangle$, werden die zusätzlichen Zeichen in $\langle s_2 \rangle$ ignoriert.

Ein kleines Beispiel zur Illustration:

Tabelle 8.6: Zeichen und Zeichenklassen für `tr`

Klasse	Bedeutung
`\a`	Control-G (ASCII 7), Glockenton
`\b`	Control-H (ASCII 8), »*backspace*«
`\f`	Control-L (ASCII 12), »*formfeed*«
`\n`	Control-J (ASCII 10), »*linefeed*«
`\r`	Control-M (ASCII 13), »*carriage return*«
`\t`	Control-I (ASCII 9), Tabulatorzeichen
`\v`	Control-K (ASCII 11), vertikaler Tabulator
`\`*kkk*	das durch die drei Oktalziffern *kkk* benannte Zeichen
`\\`	ein Rückstrich
`[`*c*`*`*n*`]`	in $\langle s_2 \rangle$: das Zeichen *c* *n*-mal
`[`*c*`*]`	in $\langle s_2 \rangle$: das Zeichen *c* so oft, dass $\langle s_2 \rangle$ so lang ist wie $\langle s_1 \rangle$
`[:alnum:]`	alle Buchstaben und Ziffern
`[:alpha:]`	alle Buchstaben
`[:blank:]`	aller horizontaler Freiplatz
`[:cntrl:]`	alle Kontrollzeichen
`[:digit:]`	alle Ziffern
`[:graph:]`	alle druckbaren Zeichen (ohne Leerzeichen)
`[:lower:]`	alle Kleinbuchstaben
`[:print:]`	alle druckbaren Zeichen (inklusive Leerzeichen)
`[:punct:]`	alle Satzzeichen
`[:space:]`	aller horizontaler oder vertikaler Freiplatz
`[:upper:]`	alle Großbuchstaben
`[:xdigit:]`	alle Hexadezimalziffern (`0`–`9`, `A`–`F`, `a`–`f`)
`[:`*c*`:]`	alle Zeichen, die äquivalent zu *c* sind (im Moment nur *c* selbst)

```
$ tr AEiü aey <bsp.txt >neu1.txt
```

Mit diesem Kommando werden in der Datei `bsp.txt` alle »A« durch »a«, alle »E« durch »e« sowie alle »i« bzw. »ü« durch »y« ersetzt und das Resultat als `neu1.txt` gespeichert.

Bereiche

Es ist erlaubt, Folgen von Zeichen durch Bereiche der Form »*m*-*n*« auszudrücken. Dabei muss *m* in der Sortierreihenfolge vor *n* stehen. Mit der Option `-c` wird

Komplement

nicht der Inhalt von $\langle s_1 \rangle$ ersetzt, sondern dessen »Komplement«, also alle Zeichen, die *nicht* in $\langle s_1 \rangle$ enthalten sind. Das Kommando

```
$ tr -c A-Za-z ' ' <bsp.txt >neu1.txt
```

ersetzt alle Nichtbuchstaben in `bsp.txt` durch Leerzeichen.

Zeichenklassen

Es ist auch möglich, Zeichenklassen der Form `[:`*k*`:]` zu benutzen (die möglichen Klassennamen ergeben sich aus Tabelle 8.6); in vielen Fällen ist das sinnvoll, um Befehle zu konstruieren, die auch in anderen Sprachumgebungen funktionieren. In einer deutschsprachigen Umgebung enthält die Zeichenklasse »`[:alpha:]`« zum Beispiel auch die Umlaute, eine »handgestrickte« Klasse der Form »`A-Za-z`«, die für Englisch stimmt, dagegen nicht. Für Zeichenklassen gelten noch einige weitere Einschränkungen, die Sie am besten der Dokumentation für `tr` entnehmen (siehe »`info tr`«).

Löschen von Zeichen

Zum Löschen von Zeichen müssen Sie nur $\langle s_1 \rangle$ angeben: Das Kommando

```
$ tr -d a-z <bsp.txt >neu2.txt
```

löscht alle Kleinbuchstaben aus der Datei `bsp.txt`. Außerdem können Sie Folgen gleicher Zeichen in der Eingabe durch ein einziges Zeichen in der Ausgabe ersetzen: Das Kommando

```
$ tr -s '\n' <bsp.txt >neu3.txt
```

entfernt Leerzeilen, indem Folgen von Zeilenendezeichen durch jeweils eins ersetzt werden.

Mit der Option `-s` (»*squeeze*«) ist es ferner möglich, beim Ersetzen zwei verschiedene Ursprungszeichen durch zwei gleiche zu ersetzen und diese dann (wie bei `-s` mit nur einem Argument) zu einem zusammenzufassen. Aus allen »A«, »E« sowie Folgen dieser Zeichen in der Datei `bsp.txt` wird ein »Ä« in `neu3.txt`: »*squeeze*«

```
$ tr -s AE Ä <bsp.txt >neu3.txt
```

Der »Tabulator« – bekannt von der guten alten Schreibmaschine – ist eine bequeme Möglichkeit, beim Programmieren oder allgemein bei der Eingabe von Texten Einrückungen zu realisieren. Nach Konvention sind in verschiedenen Spalten (normalerweise alle 8 Spalten, also in den Positionen 8, 16, 24, …) »Tabulatorstopps« gesetzt, und die gängigen Editoren rücken beim Druck auf die Tab-Taste nach rechts an den nächsten Tabulatorstopp – wenn Sie die Tab-Taste drücken, während die Eingabemarke in Spalte 11 auf dem Bildschirm steht, zum Beispiel an Spalte 16. Trotzdem werden die aus dem Druck auf Tab resultierenden »Tabulatorzeichen« als solche in die Datei geschrieben, und manche Programme können sie nicht richtig interpretieren. Das Kommando `expand` hilft hier: Es liest die durch die übergebenen Parameter bezeichneten Dateien (ersatzweise – Sie wissen schon – seine Standard-Eingabe) und gibt sie auf seiner Standard-Ausgabe aus, wobei alle Tabulatorzeichen durch die richtige Anzahl von Leerzeichen ersetzt werden, damit die Tabulatorstopps alle 8 Zeichen erhalten bleiben. Mit der Option `-t` können Sie einen anderen »Multiplikator« für die Tabulatorstopps angeben; gängig ist zum Beispiel »`-t 4`«, was die Stopps in die Spalten 4, 8, 12, 16, … setzt. Tabulator

Tabulatorzeichen expandieren

Wenn Sie bei `-t` mehrere durch Komma getrennte Zahlen angeben, werden Tabulatorstopps genau auf die benannten Spalten gesetzt: »`expand -t 4,12,32`« setzt Tabulatorstopps in die Spalten 4, 12 und 32. Weitere Tabulatorzeichen in einer Zeile werden durch einzelne Leerzeichen ersetzt.

Die Option `-i` (engl. *initial*) bewirkt, dass nur Tabulatorzeichen am Zeilenanfang zu Leerzeichen expandiert werden.

Das Kommando `unexpand` bewirkt im Großen und Ganzen das Gegenteil von `expand`: Alle Folgen von Tabulator- und Leerzeichen am Anfang von Zeilen in der Eingabe (wie üblich benannte Dateien oder die Standard-Eingabe) werden durch die minimale Folge von Tabulator- und Leerzeichen ersetzt, die die gewünschte Einrückung realisieren. Eine Zeile mit einem Tabulatorzeichen, zwei Leerzeichen, einem weiteren Tabulatorzeichen und neun Leerzeichen am Anfang zum Beispiel wird – Standard-Tabulatorstopps in jeder achten Spalte vorausgesetzt – durch eine Zeile ersetzt, die mit drei Tabulatorzeichen und einem Leerzeichen anfängt. Die Option `-a` (engl. *all*) bewirkt, dass *alle* Folgen von zwei oder mehr Tabulator- oder Leerzeichen »optimiert« werden und nicht nur die am Zeilenanfang. Leerzeichen durch Tabulatorzeichen ersetzen

Übungen

8.14 [!2] Der berühmte römische Feldherr Julius Caesar verwendete angeblich für die Übermittlung geheimer Botschaften das folgende Verschlüsselungsverfahren: Der Buchstabe »A« wurde ersetzt durch »D«, »B« durch »E« und so weiter; »X« entsprechend durch »A«, »Y« durch »B« und »Z« durch »C« (wenn wir mal die heute üblichen 26 Buchstaben zugrundelegen und den Umstand ignorieren, dass die alten Römer kein J, kein K, kein W und

kein Y hatten). Stellen Sie Sich vor, Sie sind ein Programmierer in Caesars Legion. Wie lauten die `tr`-Aufrufe, um eine Botschaft des Feldherrn zu verschlüsseln und wieder zu entschlüsseln?

8.15 [3] Mit welchem `tr`-Kommando können Sie alle Vokale in einem Text durch einen gegebenen einzigen ersetzen? Denken Sie an das Kinderlied:

```
DREI CHINESEN MIT DEM KONTRABASS
DRAA CHANASAN MAT DAM KANTRABASS
```

8.16 [3] Wie würden Sie eine Textdatei so umformen, dass alle Satzzeichen entfernt und jedes Wort auf eine einzelne Zeile gestellt wird?

8.17 [2] Geben Sie ein `tr`-Kommando an, mit dem die Zeichen »`a`«, »`z`«, und »`-`« aus der Standardeingabe entfernt werden können.

8.18 [1] Wie würden Sie sich davon überzeugen, dass `unexpand` tatsächlich Leer- und Tabulatorzeichen durch eine »optimale« Folge ersetzt?

8.4.2 Zeile für Zeile – `fmt`, `pr` und so weiter

Während die Kommandos aus dem vorigen Abschnitt die Eingabezeichen einzeln oder in kleinen Gruppen betrachtet haben, gibt es in Linux auch viele Kommandos, die ganze Eingabezeilen bearbeiten. Einige davon stellen wir im folgenden vor.

Zeilenumbruch

Das Programm `fmt` umbricht die Zeilen der Eingabe (wie gewöhnlich den auf der Kommandozeile benannten Dateien oder der Standard-Eingabe entnommen) so, dass sie maximal eine bestimmte vorgegebene Länge haben – 75 Zeichen, wenn nicht mit der Option `-w` anders festgelegt. Dabei bemüht es sich durchaus um gut aussehende Ausgabe.

Betrachten wir einige Beispiele für `fmt` (die Datei `frosch0.txt` entspricht der Datei `frosch.txt`, bis darauf, dass die erste Zeile in jedem Absatz um zwei Zeichen eingerückt ist):

```
$ head frosch0.txt
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
```

Im ersten Beispiel reduzieren wir die Zeilenlänge auf 40 Zeichen:

```
$ fmt -w 40 frosch0.txt
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
⊲⊲⊲⊲
```

Beachten Sie, dass die zweite Zeile des ersten Absatzes um zwei Leerzeichen eingerückt ist, ohne dass unmittelbar einleuchtet, warum. Die Lösung des Rätsels ist, dass `fmt` normalerweise nur Folgen von Zeilen mit derselben Einrückung als Absatz betrachtet, der für einen »gemeinsamen« Zeilenumbruch in Frage kommt. Die eingerückte erste Zeile des ersten Absatzes der Beispieldatei gilt also als eigener »Absatz«, wird für sich umbrochen und alle resultierenden Zeilen mit der Einrückung der ersten (und einzigen) Zeile des »Absatzes« in der Eingabe versehen. Die zweite und die folgenden Zeilen in der Eingabe gelten als unabhängiger weiterer »Absatz« und werden ebenfalls separat umbrochen (mit der Einrückung der zweiten Zeile).

`fmt` versucht Leerzeilen, Wortzwischenraum und Einrückung in der Eingabe beizubehalten. Zeilenumbrüche macht es am liebsten am Satzende und ungern nach dem ersten oder vor dem letzten Wort im Satz. Ein »Satzende« aus der Sicht von `fmt` ist entweder das Ende eines Absatzes oder ein Wort, das mit ».«, »?« oder »!«, gefolgt von zwei (!) Leerzeichen, aufhört.

Mehr Details über die Funktionsweise von `fmt` stehen in der Info-Seite des Programms (Tipp zum Finden: `fmt` ist Bestandteil der GNU-Coreutils).

Im nächsten Beispiel verwenden wir die Option `-c` (engl. *crown-margin mode*), um das gerade gezeigte Phänomen zu vermeiden:

```
$ fmt -c -w 40 frosch0.txt
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
◁◁◁◁◁
```

Hier orientiert die Einrückung des (kompletten) Absatzes sich an den ersten zwei Zeilen der Eingabe; die Einrückung dieser beiden Zeilen wird erhalten, und die weiteren Zeilen der Eingabe folgen der Einrückung der zweiten Zeile.

Zum Schluss noch ein Beispiel mit langen Zeilen:

```
$ fmt -w 100 frosch0.txt
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
◁◁◁◁◁
```

Hier hätten wir ebenfalls `-c` angeben können, um die »kurze« erste Zeile der Absätze zu vermeiden. Ohne diese Option gilt sie wieder als eigenständiger Absatz und wird nicht mit den Folgezeilen verschmolzen.

Der Name des Kommandos `pr` (engl. *print*, »drucken«) mag zunächst in die Irre führen. Es gibt Dateien nicht, wie vielleicht naheliegend, auf einem Drucker aus – hierzu dient `lpr`. Statt dessen ermöglicht `pr` die Formatierung eines Textes, der entweder aus einer Datei (bzw. Dateien) oder der Standard-Eingabe stammt, mit

Tabelle 8.7: Optionen von `pr`

Option	Wirkung
`-`*n*	ergibt eine Darstellung in *n* Spalten (zwar sind beliebige Werte erlaubt, aber nur Zahlen von 2 bis 5 sind normalerweise sinnvoll)
`-h` *t*	(engl. *header*) Gibt statt des Dateinamens den Text *t* in jedem Seitenkopf aus
`-l` *n*	(engl. *length*) Legt die Anzahl der Zeilen pro Seite fest, voreingestellt sind 66
`-n`	(engl. *number*) Versieht jede Zeile mit einer fünfstelligen Zeilennummer, die durch ein Tabulatorzeichen abgetrennt ist
`-o` *n*	(engl. *offset*) Rückt den Text *n* Zeichen vom linken Rand ein
`-t`	(engl. *omit*) unterdrückt die je fünf Kopf- und Fußzeilen
`-w` *n*	(engl. *width*) Legt die Anzahl von Zeichen pro Zeile fest, voreingestellt ist 72

Seitenumbrüchen, Einrückung und Kopf- und Fußzeilen. Auch hier sind natürlich umfangreiche Kontrollmöglichkeiten gegeben. Die wichtigsten Optionen für `pr` finden Sie in Tabelle 8.7.

Zur Veranschaulichung der Arbeitsweise von `pr` ein etwas komplexeres Beispiel:

```
$ fmt -w 34 frosch.txt | pr -h "Grimms Märchen" -2


2003-10-20 21:03                Grimms Märchen                  Seite 1


Der Froschkönig oder der eiserne    antwortete der Frosch, >>ich kann
Heinrich                            wohl Rat schaffen. Aber was gibst
                                    du mir, wenn ich dein Spielzeug
In alten Zeiten, als das Wünschen   wieder heraufhole?«
noch geholfen hat, lebte einmal
ein König, der hatte wunderschöne   >>Was du haben willst, lieber
Töchter. Die jüngste von ihnen war  Frosch«, sagte sie, >>meine
so schön, daß die Sonne selber,     Kleider, meine Perlen und
die doch so vieles schon gesehen    Edelsteine, auch noch die goldene
hat, sich verwunderte, sooft sie    Krone, die ich trage.«
ihr ins Gesicht schien.
◁◁◁◁◁
```

Hier verwenden wir `fmt`, um den Text des »Froschkönigs« in einer langen, schmalen Spalte zu formatieren, und `pr`, um den Text dann zweispaltig zu Papier (bzw. zu Bildschirm) zu bringen.

Zeilennumerierung

Auf Zeilennumerierung spezialisiert ist das Kommando `nl`. Wenn nichts anderes angegeben wurde, numeriert es die nichtleeren Zeilen der Eingabe (die wie gewöhnlich aus benannten Dateien oder von der Standard-Eingabe kommen darf) fortlaufend durch:

```
$ nl frosch.txt
     1  Der Froschkönig oder der eiserne Heinrich

     2  In alten Zeiten, als das Wünschen noch geholfen hat, lebte einmal ein
     3  König, der hatte wunderschöne Töchter. Die jüngste von ihnen war so
     4  schön, daß die Sonne selber, die doch so vieles schon gesehen hat,
     5  sich verwunderte, sooft sie ihr ins Gesicht schien.

     6  Nahe bei dem Schlosse war ein großer, dunkler Wald, und mitten darin,
◁◁◁◁◁
```

Tabelle 8.8: Optionen für `nl` (Auswahl)

Option		Wirkung
`-b` *s*	(*body style*)	Numeriert die Zeilen des Körpers gemäß *s*. Mögliche Werte für *s* sind `a` (alle Zeilen numerieren), `t` (nur nichtleere Zeilen numerieren), `n` (gar keine Zeilen numerieren) und `p`⟨*Ausdruck*⟩ (nur die Zeilen numerieren, die auf den regulären Ausdruck ⟨*Ausdruck*⟩ passen). Standardwert ist `t`.
`-d` *p*[*q*]	(*delimiter*)	Verwendet die beiden Zeichen *pq* statt »`\:`« in Trennzeilen. Wurde nur *p* angegeben, so ist *q* weiterhin »`:`«.
`-f` *s*	(*footer style*)	Formatiert die Fußzeilen gemäß *s*. Die möglichen Werte für *s* entsprechen denen bei `-b`; Standardwert ist `n`.
`-h` *s*	(*header style*)	Analog zu `-f`, für Kopfzeilen.
`-i` *n*	(*increment*)	Erhöht die Zeilennummer pro Zeile um *n*.
`-n` *f*	(*number format*)	Bestimmt das Format für die Zeilennummern. Mögliche Werte für *f*: `ln` (linksbündig ohne führende Nullen), `rn` (rechtsbündig ohne führende Nullen), `rz` (rechtsbündig mit führenden Nullen).
`-p`	(*page*)	Setzt die Zeilennummer zwischen logischen Seiten *nicht* auf den Anfangswert zurück.
`-v` *n*		Beginnt die Numerierung bei Zeilennummer *n*.
`-w` *n*	(*width*)	Die Zeilennummer wird *n* Zeichen breit ausgegeben.

Das ist zunächst einmal nichts, was Sie nicht auch mit »`cat -b`« hinkriegen würden. Zum einen erlaubt `nl` aber die genauere Steuerung der Numerierung:

```
$ nl -b a -n rz -w 5 -v 1000 -i 10 frosch.txt
01000   Der Froschkönig oder der eiserne Heinrich
01010
01020   In alten Zeiten, als das Wünschen noch geholfen hat, lebte einmal ein
01030   König, der hatte wunderschöne Töchter. Die jüngste von ihnen war so
01040   schön, daß die Sonne selber, die doch so vieles schon gesehen hat,
01050   sich verwunderte, sooft sie ihr ins Gesicht schien.
01060
01070   Nahe bei dem Schlosse war ein großer, dunkler Wald, und mitten darin,
01080   unter einer alten Linde, war ein Brunnen. Wenn nun der Tag recht heiß war,
01090   ging die jüngste Prinzessin hinaus in den Wald und setzte sich an den Rand
◁◁◁◁◁
```

Im Einzelnen stehen die Optionen für das Folgende (siehe auch Tabelle 8.8): »`-b a`« sorgt dafür, dass alle Zeilen numeriert werden und nicht – wie im vorigen Beispiel – nur die nichtleeren. »`-n rz`« formatiert die Zeilennummern rechtsbündig mit führenden Nullen, »`-w 5`« sorgt für eine Breite von fünf Spalten. »`-v 1000`« lässt die Numerierung bei 1000 beginnen, und »`-i 10`« erhöht die Zeilennummer pro Zeile um 10 (nicht, wie sonst, 1).

Zeilennummern pro Seite

Zum anderen kann `nl` auch mit einer logischen Seiteneinteilung der Eingabe umgehen. Dazu dienen die »magischen« Zeichenketten »`\:\:\:`«, »`\:\:`« und »`\:`« wie im folgenden Beispiel gezeigt:

```
$ cat nl-test
\:\:\:
Kopf der ersten Seite
\:\:
Erste Zeile der ersten Seite
Zweite Zeile der ersten Seite
Letzte Zeile der ersten Seite
\:
```

Tabelle 8.9: Optionen für `wc` (Auswahl)

Option	Wirkung
`-l` (*lines*)	gibt Anzahl der Zeilen aus
`-w` (*words*)	gibt Anzahl der Wörter aus
`-c` (*characters*)	gibt Anzahl der Zeichen aus

```
Fuß der ersten Seite
\:\:\:
Kopf der zweiten Seite
(Zwei Zeilen hoch)
\:\:
Erste Zeile der zweiten Seite
Zweite Zeile der zweiten Seite
Vorletzte Zeile der zweiten Seite
Letzte Zeile der zweiten Seite
\:
Fuß der zweiten Seite
(Zwei Zeilen hoch)
```

Kopf- und Fußbereiche

Jede (logische) Seite hat einen Kopf- und einen Fußbereich sowie einen »Körper« (engl. *body*), wo der eigentliche Text steht. Der Kopfbereich wird mit »`\:\:\:`« eingeleitet und mit »`\:\:`« vom Körper getrennt. Der Körper wiederum endet bei einer »`\:`«-Zeile. Kopf und Fuß können auch wegfallen.

In der Standardeinstellung numeriert `nl` die Zeilen jeder Seite beginnend bei 1; Kopf- und Fußzeilen werden nicht numeriert:

```
$ nl nl-test

       Kopf der ersten Seite

     1  Erste Zeile der ersten Seite
     2  Zweite Zeile der ersten Seite
     3  Letzte Zeile der ersten Seite

       Fuß der ersten Seite

       Kopf der zweiten Seite
       (Zwei Zeilen hoch)

     1  Erste Zeile der zweiten Seite
     2  Zweite Zeile der zweiten Seite
     3  Vorletzte Zeile der zweiten Seite
     4  Letzte Zeile der zweiten Seite

       Fuß der zweiten Seite
       (Zwei Zeilen hoch)
```

Die »`\:…`«-Trennzeilen werden in der Ausgabe jeweils durch eine Leerzeile ersetzt.

Zeilen, Wörter, Zeichen zählen

Der Name des Kommandos `wc` stellt die Abkürzung für *»word count«* (engl. für »Wörter zählen«) dar. Entgegen dieser Bezeichnung lässt sich nicht nur die Anzahl der Wörter, sondern auch der Zeichen überhaupt und der Zeilen der Eingabe (Dateien, Standard-Eingabe) ermitteln. Dies geschieht mit in Tabelle 8.9 aufgeführten Optionen. Ein »Wort« aus der Sicht von `wc` ist eine Folge von einem oder mehreren Buchstaben. Ohne Option werden alle drei Werte in der Reihenfolge aufgelistet, wie sie in Tabelle 8.9 erscheinen:

```
$ wc frosch.txt
 159 1255 7406 frosch.txt
```

Mit den Optionen in Tabelle 8.9 können Sie die Ausgabe von `wc` auf bestimmte der Werte beschränken:

```
$ ls | wc -l
     13
$ _
```

Das Beispiel zeigt, wie mit `wc` durch Zählen der Ausgabezeilen von `ls` die Anzahl der Einträge im aktuellen Verzeichnis bestimmt werden kann.

Übungen

8.19 [1] Numerieren Sie die Zeilen der Datei `frosch.txt` in Zweierschritten beginnend bei 100.

8.20 [3] Wie können Sie die Zeilen einer Datei in *umgekehrter* Reihenfolge numerieren, also etwa so:

```
159  Der Froschkönig oder der eiserne Heinrich
158
157  In alten Zeiten, als das Wünschen noch geholfen hat, lebte einmal ein
156  König, der hatte wunderschöne Töchter. Die jüngste von ihnen war so
◁◁◁◁◁
```

(*Tipp:* Zweimal umgedreht gibt wieder das Original).

8.21 [!2] Wie unterscheidet sich die Ausgabe des Kommandos »`wc a.txt b.txt c.txt`« von der des Kommandos »`cat a.txt b.txt c.txt | wc`«?

8.5 Datenverwaltung

8.5.1 Sortierte Dateien – `sort` und `uniq`

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«.

Standardeinstellung

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
```

[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.

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.

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

Sortieren in Feldern

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.

`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`«.

Trennmarkierung

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.)

```
$ 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
```

Tabelle 8.10: 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

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 8.24.

genauere Feldbestimmung

Das zu sortierende Feld können Sie noch genauer bestimmen, wie das folgende Beispiel zeigt:

```
$ 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 8.24 manifestiert sich auch hier noch.)

andere Trennzeichen

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:

```
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 Startnummer (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 8.10; 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

8.22 [!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).

8.23 [3] Wie können Sie die Teilnehmerliste aufsteigend nach den Vereinsnamen und innerhalb der Vereine absteigend nach der Startnummer sortieren? (*Tipp:* Dokumentation lesen!)

8.24 [!2] Was ist der »Schönheitsfehler«, von dem in den Beispielen die Rede ist, und warum tritt er auf?

8.25 [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.

8.26 [3] Wie können Sie eine sortierte Liste aller Wörter in einer Textdatei aufstellen? Jedes Wort soll in der Liste nur einmal erscheinen. (*Tipp:* Übung 8.16)

8.5.2 Spalten und Felder – `cut`, `paste` & Co.

Spalten ausschneiden

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:

absolute Spalten

Eine Möglichkeit ist die absolute Bearbeitung von Spalten. Diese Spalten ent-

sprechen 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.

```
$ 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
```

Tabelle 8.11: Optionen für `join` (Auswahl)

Option		Wirkung
`-j1` *n*		Verwendet Feld *n* der ersten Datei als »Vereinigungsfeld« ($n \geqslant 1$). Synonym: `-1` *n*.
`-j2` *n*		Verwendet Feld *n* der zweiten Datei als »Vereinigungsfeld« ($n \geqslant 1$). Synonym: `-2` *n*.
`-j` *n*	(*join*)	Abkürzung für »`-j1` *n* `-j2` *n*«
`-o` *f*	(*output*)	Spezifikation für die Ausgabezeilen. *f* ist eine durch Komma getrennte Folge von Feldspezifikationen, wobei jede entweder die Ziffer »`0`« oder eine Feldnummer *m.n* ist. Dabei steht die »`0`« für das »Vereinigungsfeld«, *m* ist 1 oder 2, und *n* ist eine Feldnummer in der ersten bzw. zweiten Datei.
`-t` *c*		Das Zeichen *c* wird als Feldtrenner für Ein- und Ausgabe verwendet.

```
$ 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.

»Relationales« Zusammenfügen von Dateien

Auch das Kommando `join` setzt Zeilen aus zwei Dateien zusammen, arbeitet aber wesentlich komplizierter als `paste`. Statt einfach die ersten Zeilen, zweiten Zeilen, … aneinanderzuhängen, orientiert es sich an jeweils einem ausgezeichneten Feld der Zeilen aus beiden Dateien und »vereinigt« zwei Zeilen nur, wenn die jeweiligen Werte in diesem Feld übereinstimmen. `join` implementiert also den gleichnamigen Operator aus der Relationenalgebra, so wie man ihn von SQL-Datenbanken her kennt – auch wenn die tatsächliche Ausführung deutlich primitiver und ineffizienter ist, als eine echte Datenbank das ermöglichen würde.

Beispiel

Trotzdem kann `join` durchaus nützlich werden. Stellen wir uns vor, der große Tag ist gekommen und der Marathonlauf des TSV Lahmhausen hat stattgefunden. Die Schiedsrichter waren fleißig und haben nicht nur alle Zeiten gestoppt, sondern diese auch in eine Datei `zeiten.dat` eingetragen. Die erste Spalte ist dabei immer die Startnummer, die zweite die erreichte Zeit (der Einfachheit halber in ganzen Sekunden):

```
$ cat zeiten.dat
45:8445
123:8517
217:8533
93:8641
154:8772
119:8830
13:8832
117:8954
57:9111
26:9129
```

Wir wollen nun diese Datei mit der Teilnehmerliste zusammenbringen, damit wir jedem Sportler bzw. jeder Sportlerin seine bzw. ihre erreichte Zeit zuordnen können. Dazu müssen wir die Ergebnisdatei zunächst nach den Startnummern sortieren:

```
$ sort -n zeiten.dat >zeiten-s.dat
```

Anschließend können wir die Zeilen der Datei `zeiten-s.dat` mit dem Programm `join` mit den entsprechenden Zeilen der modifizierten Teilnehmerdatei aus dem

paste-Beispiel zusammenbringen – join setzt standardmäßig voraus, dass seine Eingabedateien zeilenweise nach dem »Vereinigungsfeld« sortiert sind, und dass das »Vereinigungsfeld« das erste Feld auf der Zeile ist.

```
$ 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
$ sort -n tn-startnr.dat \
>  | join -t: zeiten-s.dat - >tn-zeiten.dat
$ cat tn-zeiten.dat
13:8832:Schleicher:Detlef:TSV Lahmhausen:Herren
26:9129:von Traben:Gesine:TV Jahnstein:Damen
45:8445:Zielinger:Hannes:TV Jahnstein:Herren
57:9111:Fluschinsky:Käthe:Sportfreunde Renntal:Damen
93:8641:Schwitz:Sieglinde:Sportfreunde Renntal:Damen
117:8954:Rasbichel:Ulla:TSV Lahmhausen:Damen
119:8830:Rasbichel:Katja:TSV Lahmhausen:Damen
123:8517:Schulz:Hugo:SV Schnaufenberg:Herren
154:8772:Springinsfeld:Karlheinz:TV Jahnstein:Herren
217:8533:Flöttmann:Fritz:Sportfreunde Renntal:Herren
```

Die resultierende Datei tn-zeiten.dat muss jetzt nur noch nach den Zeiten sortiert werden:

```
$ sort -t: -k2,2 tn-zeiten.dat
45:8445:Zielinger:Hannes:TV Jahnstein:Herren
123:8517:Schulz:Hugo:SV Schnaufenberg:Herren
217:8533:Flöttmann:Fritz:Sportfreunde Renntal:Herren
93:8641:Schwitz:Sieglinde:Sportfreunde Renntal:Damen
154:8772:Springinsfeld:Karlheinz:TV Jahnstein:Herren
119:8830:Rasbichel:Katja:TSV Lahmhausen:Damen
13:8832:Schleicher:Detlef:TSV Lahmhausen:Herren
117:8954:Rasbichel:Ulla:TSV Lahmhausen:Damen
57:9111:Fluschinsky:Käthe:Sportfreunde Renntal:Damen
26:9129:von Traben:Gesine:TV Jahnstein:Damen
```

Dies ist ein schönes Beispiel dafür, dass mit Linux-»Bordmitteln« auch durchaus komplexe Text- und Datenverarbeitung möglich ist. Im »wirklichen Leben« würden Sie die gezeigten Befehlsfolgen natürlich als Shellskripte vorbereiten und die entsprechenden Schritte auf diese Weise automatisieren.

Übungen

8.27 [!2] Generieren Sie eine neue Version der Datei teilnehmer.dat (der mit der festen Spaltenbreite), in der die Startnummer und die Vereinszugehörigkeit nicht auftauchen.

8.28 [!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.

8.29 [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.

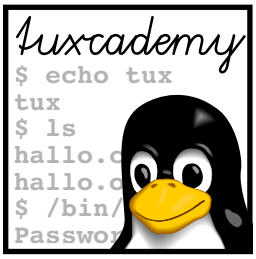
8.30 [3] Wie viele verschiedene Gruppen werden von Benutzern auf Ihrem System als primäre Gruppen benutzt? (Die primäre Gruppe eines Benutzers ist das vierte Feld in der Datei `/etc/passwd`.)

Kommandos in diesem Kapitel

cat	Hängt Dateien aneinander	cat(1)	113
cut	Extrahiert Felder oder Spalten aus seiner Eingabe	cut(1)	132
expand	Ersetzt Tabulatorzeichen in der Eingabe durch äquivalente Leerzeichen	expand(1)	121
fmt	Umbricht die Zeilen der Eingabe auf eine bestimmte Breite	fmt(1)	122
hd	Abkürzung für `hexdump`	hexdump(1)	117
head	Zeigt den Anfang einer Datei an	head(1)	115
hexdump	Gibt Dateiinhalte in hexadezimaler (oktaler, …) Form aus	hexdump(1)	117
join	Führt die Zeilen zweier Dateien „relational“ zusammen	join(1)	135
nl	Numeriert die Zeilen der Eingabe	nl(1)	124
od	Zeigt die Bytes einer Datei in dezimaler, oktaler, hexadezimaler, ... Darstellung	od(1)	116
paste	Fügt verschiedene Eingabedateien zeilenweise aneinander	paste(1)	134
pr	Bereitet seine Eingabe zum Drucken auf – mit Kopfzeilen, Fußzeilen usw.	pr(1)	123
reset	Setzt den Bildschirmzeichensatz auf einen „vernünftigen“ Wert	tset(1)	113
sort	Sortiert die Zeilen seiner Eingabe	sort(1)	127
tac	Zeigt eine Datei von hinten nach vorne an	tac(1)	114
tail	Zeigt das Ende einer Datei an	tail(1)	115
tee	Kopiert die Standardeingabe in die Standardausgabe und außerdem in Dateien	tee(1)	111
tr	Tauscht Zeichen in der Standardeingabe gegen andere aus oder löscht sie	tr(1)	119
unexpand	„Optimiert“ Tabulator- und Leerzeichen in der Eingabe	unexpand(1)	121
uniq	Ersetzt Folgen von gleichen Zeilen in der Eingabe durch die erste solche	uniq(1)	131
wc	Zählt Zeilen, Wörter und Zeichen in seiner Eingabe	wc(1)	126

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.
- Das Kommando `tr` dient zum Austauschen oder Löschen einzelner Zeichen. `expand` und `unexpand` konvertieren Tabulatorzeichen in Leerzeichen und umgekehrt.
- Mit `pr` kann man Daten zum Drucken aufbereiten (nicht drucken).
- Mit `wc` kann man die Zeilen, Wörter und Zeichen der Standardeingabe (oder der angegebenen Datei(en)) zählen.
- `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
- Mit Vordergrund- und Hintergrund-Prozessen umgehen können

Vorkenntnisse

- Shell-Grundkenntnisse (Kapitel 3)
- Dateiverwaltung und einfache Filterkommandos (Kapitel 6, Kapitel 8)

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.

Variable setzen

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:

```
$ bla=fasel
$ echo bla
bla
$ echo $bla
fasel
```

(achten Sie auf den Unterschied).

Umgebungsvariable
Shellvariable

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.

`export`

Mit dem Kommando `export` können Sie eine existierende Shellvariable zur Umgebungsvariable erklären:

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 3 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 3.3.3).

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 5) 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 5 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.

Die Shell merkt sich standardmäßig die letzten 500 Kommandos; ändern können Sie das, indem Sie die gewünschte Zahl in die Variable `HISTSIZE` schreiben. Die Variable `HISTFILESIZE` gibt an, wie viele Kommandos in die `HISTFILE`-Datei geschrieben werden – normalerweise ebenfalls 500.

Außer über die Pfeiltasten können Sie die Vorgeschichte auch über »magische« Zeichenfolgen in neuen Kommandos ansprechen. Diese Zeichenfolgen ersetzt die Shell als allererstes, direkt nach dem Einlesen der Kommandozeile. Die Ersetzung erfolgt in zwei Schritten:

- Zunächst wird bestimmt, welches Kommando aus der Vorgeschichte zur Ersetzung herangezogen wird. Die Zeichenfolge `!!` steht dabei für das unmittelbar vorige Kommando, `!-`*n* bezieht sich auf das *n*-te Kommando vor dem aktuellen (`!-2` also zum Beispiel auf das vorletzte) und `!`*n* auf das Kommando mit der Nummer *n* in der Vorgeschichte. (Das Kommando `history` gibt die komplette Vorgeschichte mit Nummern aus.) `!xyz` selektiert das jüngste Kommando, das mit `xyz` anfängt, und `!?xyz` das jüngste Kommando, das `xyz` *enthält*.

- Anschließend wird entschieden, welcher Teil des ausgewählten Kommandos »recyclet« wird und wie. Wenn Sie nichts sagen, wird das komplette Kommando eingesetzt; ansonsten gibt es einige Auswahlmöglichkeiten. Alle diese Auswahlmöglichkeiten werden durch einen Doppelpunkt (»`:`«) von der Kommandoauswahl-Zeichenfolge getrennt.

  n Wählt das *n*-te Wort. Wort 0 ist dabei der Befehl selbst.

  ^ Wählt das erste Wort (direkt nach dem Befehl).

  $ Wählt das letzte Wort.

  m-*n* Wählt die Wörter *m* bis *n*.

  *n** Wählt alle Wörter ab dem *n*-ten.

  n- Wählt alle Wörter ab dem *n*-ten, bis auf das letzte.

  Einige Beispiele zur Verdeutlichung:

  !-2:$ Wählt das letzte Wort des vorletzten Kommandos.

  !!:0- Wählt das komplette letzte Kommando bis auf das letzte.

  !333^ Wählt das erste Wort aus Kommando 333.

  Das letzte Beispiel ist übrigens kein Tippfehler; wenn das erste Zeichen der Teilauswahl aus der Liste `^$*-%` kommt, darf der Doppelpunkt wegfallen. – Studieren Sie, wenn Sie mögen, die Bash-Dokumentation (Abschnitt HISTORY), um herauszufinden, was die Shell noch alles Interessantes auf Lager hat. Auswendig lernen müssen Sie das von uns (und dem LPI) aus nicht.

Die Vorgeschichte ist eine der Sachen, die die Bash von der C-Shell übernommen hat, und wer die 1980er Jahre nicht selbst miterlebt hat, kann sich möglicherweise nur schwer vorstellen, wie die Welt vor der Erfindung des interaktiven Kommandozeilen-Editierens ausgesehen haben mag. (Für Windows-Anwender liegt diese Zeit noch nicht *so* lange zurück.) Damals galt die Vorgeschichte mit all ihren `!`-Selektoren und Transformationen als die beste Idee seit der Erfindung des vorgeschnittenen Brots; heute hingegen übt die Dokumentation dieselbe Sorte morbide Faszination aus wie die Gebrauchsanleitung für eine Dampfmaschine aus Kaisers Zeit.

Noch ein paar Anmerkungen zum `history`-Befehl: Ein Aufruf wie

```
$ history 33
```

(mit einer Zahl als Parameter) gibt nur die letzten so vielen Zeilen der Vorgeschichte aus. »`history -c`« leert die Vorgeschichte. Es gibt noch einige andere Optionen; schauen Sie in die Bash-Dokumentation oder probieren Sie »`help history`«.

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)

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.

Komplettierung von Kommando- bzw. Dateinamen

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.

Befehlsliste

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.

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.

Rückgabewert

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 5.) Dazu müssen Sie nur die Shell aufrufen und den Namen der Datei als Parameter übergeben:

```
$ bash meine-kommandos
```

Shellskript

Eine solche Datei bezeichnet man auch als **Shellskript**, und die Shell hat umfangreiche Möglichkeiten zur Programmierung, auf die wir an dieser Stelle nicht näher eingehen können. (Die Linup-Front-Schulungsunterlage *Linux für Fortgeschrittene* erklärt die Programmierung von Shellskripten sehr ausführlich.)

Sub-Shell

Wenn Sie ein Shellskript wie oben gezeigt 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:

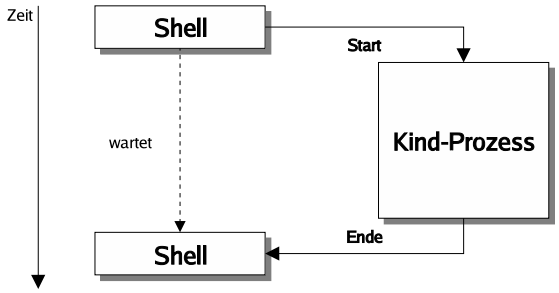


Bild 9.1: Die synchrone Arbeitsweise der Shell

```
$ 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 Vorder- und Hintergrundprozesse

Wird ein Kommando eingegeben und mit der ↩-Taste bestätigt, führt die Shell die erhaltene Anweisung aus. Interne Kommandos bearbeitet die Shell direkt; bei externen Kommandos erzeugt die Shell einen **Kindprozess**, der zur Ausführung dieses Befehls dient und sich anschließend wieder beendet. Ein Prozess unter Unix ist ein Programm, das läuft; dasselbe Programm kann mehrmals gleichzeitig (zum Beispiel von verschiedenen Benutzern) ausgeführt werden und korrespondiert dann mit mehreren Prozessen. Jeder Prozess kann Kindprozesse anlegen (auch wenn die meisten es – im Gegensatz zu Shells – nicht tun).

Kindprozess

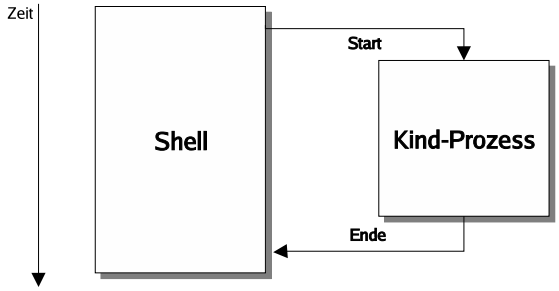


Bild 9.2: Die asynchrone Arbeitsweise der Shell

Normalerweise wartet die Shell nun, bis der Kindprozess seine Arbeit erledigt und sich beendet hat. Das merken Sie daran, dass während der Laufzeit des Kindprozesses keine neue Eingabeaufforderung erscheint. Nach Beendigung des Kindprozesses wird dessen **Rückgabewert** gelesen und ausgewertet, und erst dann erscheint wieder ein Prompt. Die Ausführung der Shell wird quasi mit dem Kind-Prozess abgeglichen. Diese »synchrone« Arbeitsweise ist in Bild 9.1 verdeutlicht; als Benutzer sehen Sie zum Beispiel etwas wie

Rückgabewert

```
$ sleep 10
                                        Ca. 10 Sekunden lang passiert nichts
$ _
```

(eigentlich kennen Sie das schon).

Soll die Shell nicht warten, bis der Kind-Prozess seine Aufgabe erledigt hat, müssen Sie ihr dies durch das Anhängen eines kaufmännischen Und-Zeichens (`&`) an die Kommandozeile mitteilen. Während der Ableger dann im Hintergrund abgearbeitet wird, erscheint nach einer kurzen Meldung zum **Hintergrund-Prozess** sofort wieder die Eingabeaufforderung der Shell:

Hintergrund-Prozess

```
$ sleep 10 &
[2] 6210
                                        Und dann sofort wieder:
$ _
```

Diese Arbeitsweise heißt »asynchron«, da die Shell nicht untätig auf das Ende des Kindprozesses wartet (s. Bild 9.2).

Die Ausgabe »`[2] 6210`« bedeutet, dass das System den Prozess mit der Nummer (oder »Prozess-ID«) 6210 als »Job« Nr. 2 angelegt hat. Auf Ihrem System dürften diese Zahlen vom Beispiel hier abweichen.

Das `&` verhält sich syntaktisch in Wirklichkeit wie der Semikolon, kann also auch als Trennzeichen zwischen Kommandos dienen. Siehe hierzu Übung 9.10.

Hier ein paar Tipps für erfolgreiche Hintergrundverarbeitung:

- Der Hintergrundprozess sollte keine Tastatureingaben erwarten, da die Shell nicht entscheiden kann, welchem Prozess – Vordergrund oder Hintergrund – die Eingaben zuzuordnen sind. Gegebenenfalls kann der Hintergrundprozess seine Eingabe einer Datei entnehmen.
- Der Hintergrundprozess sollte keine Terminalausgaben machen, da sich diese mit den Ausgaben von Vordergrund-Aktivitäten überlagern können. Fehlermeldungen können bei Bedarf in eine Datei umgeleitet oder komplett verworfen werden.

Tabelle 9.3: Optionen für `jobs`

Option	Wirkung
`-l` (long)	zeigt zusätzlich die PID an
`-n` (notify)	zeigt nur Prozesse, die seit dem letzten Aufruf von `jobs` beendet worden sind
`-p` (process)	zeigt nur die PID an

- Wird der Eltern-Prozess abgebrochen, werden meist auch alle zugehörigen Kind-Prozesse (und demzufolge auch deren Kinder usw.) beendet. Davon ausgenommen sind nur Prozesse, die sich komplett von ihren Eltern lossagen, etwa weil sie Systemdienste im Hintergrund erbringen sollen.

Jobkontrolle

Befinden sich mehrere Prozesse im Hintergrund, verliert man leicht die Übersicht. Daher stellt die Bash ein eingebautes Kommando zur Verfügung, mit dem Sie sich über den Zustand von Hintergrundprozessen informieren können – `jobs`. Wird `jobs` ohne Zusätze eingegeben, erhalten Sie eine Liste mit der Jobnummer, dem Prozesszustand und dem Kommandotext. Das sieht dann etwa folgendermaßen aus:

```
$ jobs
[1]   Done                    sleep
$ _
```

In diesem Fall ist der Prozess mit der Jobnummer 1 bereits beendet worden (engl. *done*, erledigt), ansonsten sehen Sie die Meldung »`Running`« (laufend). Das Kommando `jobs` kennt verschiedene Optionen, wobei die wichtigsten in Tabelle 9.3 aufgeführt sind.

Die Bash ermöglicht es, einen Vordergrundprozess mit der Tastenkombination Strg+z anzuhalten. Dieser Prozess wird dann von `jobs` mit dem Status »`Stopped`« gekennzeichnet. Mit dem Befehl `bg` (engl. *background* = Hintergrund) lässt sich ein solcher Prozess in den Hintergrund schicken, wo er dann weiter bearbeitet wird (ansonsten bleibt er angehalten bis zum Sankt-Nimmerleins-Tag oder zum nächsten System-Neustart, was auch immer eher eintritt). So bugsiert die Eingabe »`bg %5`« den Prozess mit der Jobnummer 5 in den Hintergrund.

Umgekehrt können Sie aus mehreren Hintergrundprozessen einen auswählen und mit dem Shell-Kommando `fg` (engl. *foreground*, Vordergrund) in den Vordergrund holen. Die Syntax von `fg` entspricht vollständig der des Kommandos `bg`.

Beenden können Sie einen Vordergrundprozess in der Bash mit der Tastenkombination Strg+c, einen Hintergrundprozess direkt mit dem Kommando `kill`, wobei Sie wie bei `bg` ein Prozentzeichen gefolgt von der Jobnummer angeben.

Übungen

9.9 [2] Verwenden Sie ein geeignet spektakuläres Programm (etwa die OpenGL-Demonstration `glxgears` unter X11, alternativ vielleicht »`xclock -update 1`«), um mit Hintergrundprozessen und Jobkontrolle zu experimentieren. Vergewissern Sie sich, dass Sie Prozesse im Hintergrund starten können, dass Sie Vordergrundprozesse mit Strg+z anhalten und mit `bg` in den Hintergrund schicken können, dass `jobs` die Hintergrundprozesse auflistet und so weiter.

9.10 [3] Beschreiben (und erklären) Sie die Unterschiede zwischen den folgenden drei Kommandozeilen:

```
$  sleep 5 ; sleep 5
$  sleep 5 ; sleep 5 &
```

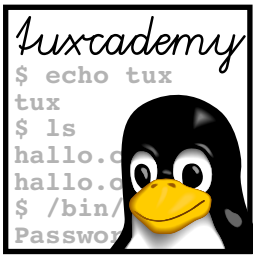
```
$ sleep 5 & sleep 5 &
```

Kommandos in diesem Kapitel

`.`	Liest eine Datei mit Shell-Kommandos so ein, als ob sie auf der Kommandozeile eingegeben worden wäre	bash(1)	149
bg	Lässt einen (angehaltenen) Prozess im Hintergrund weiterlaufen	bash(1)	151
date	Gibt Datum und Uhrzeit aus	date(1)	140
env	Gibt die Prozessumgebung aus oder startet Programme mit veränderter Umgebung	env(1)	142
export	Definiert und verwaltet Umgebungsvariable	bash(1)	141
fg	Holt einen Hintergrundprozess zurück in den Vordergrund	bash(1)	151
glxgears	Zeigt sich drehende Zahnräder unter X11, mit OpenGL	glxgears(1)	151
hash	Zeigt und verwaltet „gesehene“ Kommandos in der bash	bash(1)	144
history	Zeigt die zuletzt verwendeten bash-Kommandos an	bash(1)	146
jobs	Berichtet über Hintergrundprozesse	bash(1)	151
kill	Hält einen Hintergrundprozess an	bash(1), kill(1)	151
set	Verwaltet Shellvariable	bash(1)	142
source	Liest eine Datei mit Shell-Kommandos so ein, als ob sie auf der Kommandozeile eingegeben worden wäre	bash(1)	149
unset	Löscht Shell- oder Umgebungsvariable	bash(1)	142
whereis	Sucht ausführbare Programme, Handbuchseiten und Quellcode zu gegebenen Kommandos	whereis(1)	144
which	Sucht Programme in PATH	which(1)	144
xclock	Zeigt eine Uhr an	xclock(1x)	151

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.
- Externe Programme können auch asynchron, also im Hintergrund gestartet werden. Die Shell gibt dann sofort eine neue Eingabeaufforderung aus.



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.

10.5 Wechselmedien

Das explizite Einbinden von Wechselmedien ist ein mühsames Geschäft, und das explizite Lösen einer Einbindung vor dem Entfernen noch viel mehr – dabei kann gerade letzteres zu Ärger führen, wenn Sie das Medium physikalisch entfernen, bevor Linux komplett damit fertig ist. Linux versucht ja, das System dadurch zu beschleunigen, dass es langsame Vorgänge wie das Schreiben von Daten auf Medien nicht sofort erledigt, sondern irgendwann später, wenn der »richtige Moment« da ist, und wenn Sie natürlich Ihren USB-Stick herausziehen, bevor die Daten wirklich dort gelandet sind, dann haben Sie im besten Fall nichts gewonnen, und im schlimmsten Fall sind die anderen Daten dort im Chaos versunken.

Als Benutzer einer grafischen Arbeitsumgebung auf einem modernen Linux-System haben Sie es leicht: Wenn Sie ein Medium einlegen oder anstöpseln – egal ob Audio-CD, USB-Stick oder Digitalkamera –, erscheint ein Dialog, der Ihnen diverse interessante Aktionen anbietet, die Sie mit dem Medium ausführen wollen könnten. »Einbinden« ist normalerweise eine davon, und das System überlegt sich auch einen schönen *mount point* für Sie. Entfernen können Sie das Medium später ebenso bequem über ein Sinnbild auf dem Bildschirmhintergrund oder in der Kontrollleiste der Arbeitsumgebung. Hierüber müssen wir in dieser Unterlage nicht im Detail reden.

Anders sieht es aus, wenn Sie auf der Kommandozeile arbeiten. Dort müssen Sie Wechselmedien nämlich wie im vorigen Abschnitt skizziert explizit ein- und aushängen. Als »normaler Benutzer« dürfen Sie das aber, wie erwähnt, nicht für beliebige Medien an beliebigen Stellen, sondern nur für solche Medientypen, die Ihr Systemverwalter dafür vorgesehen und auch schon mit »vorgekochten« *mount points* versehen hat. Sie erkennen diese daran, dass sie in der Datei `/etc/fstab` mit der Option `user` oder `users` gekennzeichnet sind:

```
$ grep user /etc/fstab
/dev/hdb        /media/cdrom0   udf,iso9660 ro,user,noauto 0 0
/dev/sda1       /media/usb      auto    user,noauto         0 0
/dev/mmcblk0p1  /media/sd       auto    user,noauto         0 0
```

Für die Details der Einträge in `/etc/fstab` müssen wir Sie auf die Linup-Front-Schulungsunterlage *Linux-Administration I* vertrösten (O. K., `fstab`(5) geht auch, aber unsere Unterlage ist netter); für uns hier und heute soll reichen, dass in diesem Beispiel drei Sorten von Wechselmedien in Frage kommen, nämlich CD-ROMs (der erste Eintrag), USB-basierte Medien wie USB-Sticks, Digitalkameras oder MP3-Player (der zweite Eintrag), und SD-Karten (der dritte Eintrag). Als »normaler Benutzer« müssen Sie sich an die vorgegebenen *mount points* halten und können (nachdem Sie das betreffende Medium platziert haben) Dinge sagen wie

```
$ mount /dev/hdb                                   für die CD-ROM
$ mount /media/cdrom0                                      dito
$ mount /dev/sda1                                  für den USB-Stick
$ mount /media/sd                                  für die SD-Karte
```

Das heißt, Linux erwartet *entweder* den Gerätenamen *oder* den *mount point*; das jeweilige Pendant dazu ergibt sich aus der Datei `/etc/fstab`. Aushängen mit `umount` erfolgt analog.

Die Option `user` in `/etc/fstab` ist dafür verantwortlich, dass das funktioniert (sie bewirkt auch noch ein paar andere Dinge, die wir hier nicht im Detail besprechen müssen). Die Option `users` tut grundsätzlich dasselbe; der Unterschied zwischen den beiden – und merken Sie sich das, es könnte in der Prüfung vorkommen – besteht darin, dass bei `user` nur derjenige Benutzer das Dateisystem wieder *aushängen* darf, der es ursprünglich eingehängt hat. Bei `users` darf es jeder Benutzer (!). (Und `root` darf es sowieso immer.)

Übungen

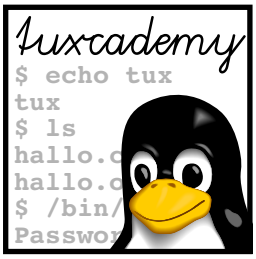
10.7 [1] Legen Sie eine Diskette ins Laufwerk, hängen Sie sie ein, kopieren Sie eine Datei (zum Beispiel `/etc/passwd`) auf die Diskette und hängen Sie die Diskette dann wieder aus. (Sollten Sie ein fortschrittliches System vor sich haben, das keine Disketten mehr annimmt, dann machen Sie dasselbe mit einem USB-Stick oder anderem geeigneten Wechselmedium.)

Kommandos in diesem Kapitel

dmesg	Gibt den Inhalt des Kernel-Nachrichtenpuffers aus	dmesg(8)	162
file	Rät den Typ einer Datei anhand des Inhalts	file(1)	154
free	Zeigt die Speicherauslastung und die Auslastung des Swap-Bereichs an	free(1)	161
klogd	Akzeptiert Protokollnachrichten des Systemkerns	klogd(8)	162
mkfifo	Legt FIFOs (benannte Pipes) an	mkfifo(1)	155
mknod	Legt Gerätedateien an	mknod(1)	155
syslogd	Bearbeitet Systemprotokoll-Meldungen	syslogd(8)	162
uptime	Gibt die Zeit seit dem letzten Systemstart sowie die CPU-Auslastung aus	uptime(1)	160

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.
- Wechselmedien müssen vor Gebrauch in den Linux-Verzeichnisbaum eingehängt und nachher wieder ausgehängt werden. Hierzu dienen die Kommandos `mount` und `umount`. Grafische Arbeitsumgebungen machen das mitunter bequemer.



11

Dateien archivieren und komprimieren

Inhalt

Lernziele

- Die Begriffe »Archivierung« und »Komprimierung« verstehen
- Mit `tar` und `cpio` umgehen können
- Dateien mit `gzip` und `bzip2` komprimieren und entkomprimieren können

Vorkenntnisse

- Arbeit mit der Shell (Kapitel 3)
- 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 archivieren mit `cpio`

`cpio` (nicht zu verwechseln mit dem goldenen Roboter aus *Star Wars*) ist genau wie `tar` ein Archivprogramm, das allerdings etwas anders funktioniert. `cpio` kennt im wesentlichen drei Betriebsmodi, die durch Optionen angesteuert werden:

- Mit der Option `-o` wird ein Archiv auf der Standardausgabe erzeugt. Das klingt etwas konterintuitiv – merken Sie es sich so, dass das Archiv die *Ausgabe* (engl. *output*) des Kommandos ist. `cpio` liest eine Liste der zu archivierenden Dateien von der Standardeingabe.
- Mit `-i` wird ein Archiv von der Standardeingabe wieder eingelesen (*input*).
- Mit der Option `-p` (*pass-through*) werden Dateien aus einer Verzeichnishierarchie in eine andere kopiert, ohne dass tatsächlich ein Archiv verwendet wird. Die Liste der Dateien kommt wie bei `-o` von der Standardeingabe; das Zielverzeichnis muss als Parameter übergeben werden.

Austausch von Datenbeständen

`cpio` erzeugt und verarbeitet eine ganze Reihe verschiedener Archivformate und ist deshalb gut für den Austausch von Datenbeständen zwischen unterschiedlichen Systemen geeignet. Unter Linux wird es allerdings selten verwendet. Obwohl `cpio` im Bezug auf die Syntax etwas komplizierter ausfällt, hat es einige entscheidende Vorteile: Es kann auch auf anderen Systemen als Linux Gerätedateien sichern, es speichert die Informationen effizienter als `tar`, fehlerhafte Stellen auf dem Sicherungsmedium werden beim Extrahieren der Daten übersprungen.

Das folgende Beispiel archiviert alle Dateien, deren Namen mit `data` beginnen und sich im Heimatverzeichnis von »`hugo`« befinden, in die Datei `archive.cpio`:

```
$ cd /home/hugo
$ find . -maxdepth 1 -name 'data*' | cpio -ov >archive.cpio
```

Da `cpio` die Namen der zu archivierenden Dateien von seiner Standardeingabe liest, ist `find` der geborene Partner. Die Option »`-maxdepth 1`« sorgt dafür, dass `find` nicht in Unterverzeichnisse absteigt.

Wenn Sie find zusammen mit cpio verwenden, um ganze Verzeichnishierarchien zu archivieren, sollten Sie sicherheitshalber die find-Option -depth benutzen. Sie sorgt dafür, dass Verzeichnisnamen immer *nach* dem Inhalt der betreffenden Verzeichnisse ausgegeben werden, und hilft gegen restriktive Zugriffsrechte auf Verzeichnissen.

Probleme mit Zeilentrennern in Dateinamen können Sie umgehen, indem Sie eine Konstruktion der Form

```
$ find ... -depth -print0 | cpio --null ...
```

verwenden. In diesem Fall gibt find nicht wie üblich einen Dateinamen pro Zeile aus (was zu Verwirrung führt, wenn mitten in einem Dateinamen eine neue Zeile anfängt), sondern trennt die Dateinamen in der Ausgabe durch Nullbytes. Die Option --null passt cpio entsprechend an.

Auspacken können Sie die Datei wie folgt:

```
$ cpio -iv <archive.cpio
```

Dabei ist zu beachten, dass cpio existierende Dateien, die dieselbe oder eine spätere Modifikationszeit als die Version der Datei im Archiv haben, nicht überschreibt (es sei denn, Sie geben die Option -u an) und auch keine Verzeichnisse anlegt (es sei denn, Sie geben die Option -d an.) Die Option -t gibt die Namen der extrahierten Dateien aus.

Sie können auf der Kommandozeile Shell-Dateisuchmuster angeben (sinnvollerweise innerhalb von Anführungszeichen), um nur bestimmte Dateien aus dem Archiv auszupacken. Beachten Sie, dass diese Suchmuster im Gegensatz zur Shell auch ein ».« am Anfang eines Dateinamens und »/« mitten in einem Pfadnamen erfassen.

GNU cpio ist nicht besonders gut dokumentiert. Die beste Informationsquelle ist die Info-Seite.

Übungen

11.7 [!2] Verpacken Sie die Dateien aus dem Archiv aus Übung 11.5 in einem cpio-Archiv. Packen Sie das Archiv an einer anderen Stelle auf dem System wieder aus.

11.4 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.

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

Grundidee

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

11.8 [2] Komprimieren Sie das tar-Archiv aus Übung 11.5 mit maximaler Verkleinerung.

11.9 [!3] Sehen Sie sich den Inhalt des komprimierten Archivs an. Stellen Sie das ursprüngliche tar-Archiv wieder her.

11.10 [!2] Wie gehen Sie vor, wenn Sie den gesamten Inhalt Ihres Heimatverzeichnisses in eine mit gzip komprimierte Datei packen möchten?

11.5 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.)

11.6 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

[1] Von der eigentlich naheliegenden Idee, das Entkomprimierungsprogramm `xunz` zu nennen, hat der Autor anscheinend Abstand genommen; ist vielleicht auch besser so.

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.

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

11.11 [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?

11.12 [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`?

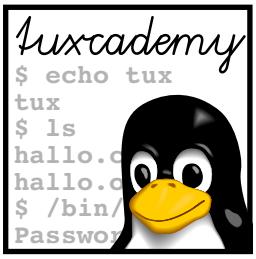
Kommandos in diesem Kapitel

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.
- `cpio` ist ein anderes, weniger verbreitetes Archivprogramm.
- `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`.

Literaturverzeichnis

RFC1951 P. Deutsch. »DEFLATE Compressed Data Format Specification version 1.3«, Mai 1996. `http://www.ietf.org/rfc/rfc1951.txt`



A

Musterlösungen

Dieser Anhang enthält Musterlösungen für ausgewählte Aufgaben.

1.2 Auf `ftp.kernel.org` steht noch ein `linux-0.01.tar.gz`.

1.3

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.

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.

1.5 Für diese Aufgabe können wir in einer gedruckten Schulungsunterlage natürlich keine korrekte Lösung angeben. Schauen Sie sich um – auf `ftp.kernel.org` oder in der wöchentlichen Ausgabe von `http://lwn.net/`.

2.2 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.

2.5 Der Anstand verbietet es uns, hier ein entsprechendes Programm abzudrucken. Einfach wird es allerdings mit Hilfe der (verpönten) C-Funktion `getpass`(3).

3.2 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.

3.3 `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.

4.2 Probieren Sie »`apropos process`« oder »`man -k process`«.

4.5 Das Format und die Werkzeuge für Info-Dateien wurden Mitte der 1980er Jahre entwickelt. HTML war da noch überhaupt nicht erfunden.

5.1 Theoretisch können Sie natürlich jedes Programm im System starten und schauen, ob es sich wie ein Texteditor benimmt … aber das würde wohl doch etwas länger dauern, als diese Aufgabe wert ist. Sie können zum Beispiel mit etwas wie »`apropos edit`« anfangen und schauen, welche der aufgezählten Handbuchseiten mit einem Texteditor korrespondieren (und nicht mit einem Editor für Grafiken, Icons, X-Ressourcen oder anderes). Die Texteditoren aus Arbeitsumgebungen wie KDE oder GNOME haben oft keine Handbuchseiten, sondern werden innerhalb der Arbeitsumgebung dokumentiert. Hier kann es nützlich sein, in den Menüs der Arbeitsumgebung nach einem Untermenü wie »Editoren« zu suchen. Die dritte Möglichkeit besteht darin, sich mit einem Kommando des Paketverwaltungssystems – etwa »`rpm -qa`« oder »`dpkg -l`« – eine Liste der installierten Softwarepakete anzeigen zu lassen und darin nach Texteditoren zu suchen.

5.2 Das Programm heißt `vimtutor`.

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 z. B. die Linup-Front-Schulungsunterlage *Linux für Fortgeschrittene*.) 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).

8.12 Bei »`a`« werden unsichtbare Zeichen mit ihren symbolischen Namen wie »`cr`« oder »`lf`« benannt, bei »`c`« durch Rückstrichsequenzen wie »`\r`« oder »`\n`«. Das Leerzeichen erscheint unter »`a`« als »`sp`«.

8.13 Der gewünschte Wertebereich (0 ... 65535 = $2^{16}-1$) entspricht gerade dem in zwei Byte darstellbaren Zahlenbereich. Wir müssen also zwei Byte aus `/dev/random` lesen und als Dezimalzahl ausgeben:

```
$ r=`od -An -N2 -tu2 /dev/random`
$ echo $r
4711
```

Die Option `-N2` liest zwei Bytes, und `-tu2` formatiert sie als vorzeichenlose 2-Byte-Dezimalzahl. `-An` unterdrückt die Positionsangabe (siehe Tabelle 8.4).

8.14 Probieren Sie mal:

```
$ echo "ALEA IACTA EST" | tr A-Z D-ZA-C
DOHD LDFWD HVW
$ echo "DOHD LDFWD HVW" | tr A-Z X-ZA-W
ALEA IACTA EST
```

Ähnlich dem Caesarschen Verfahren ist das »ROT13«-Verfahren, das im USENET benutzt wird, beispielsweise um unanständige Witze zu veröffentlichen, ohne dass empfindliche Personen sie versehentlich zu lesen bekommen (für eine »echte« Verschlüsselung vertraulicher Inhalte ist das Verfahren natürlich ein bißchen schwach). ROT13 wird durch das Kommando »`tr A-Za-z N-ZA-Mn-za-m`« beschrieben; der Vorteil dieses Verfahrens ist, dass man den Originaltext wiederbekommt, wenn man es zweimal hintereinander anwendet. Die ROT13-Routine in einem Newsreader kann also zum Ver- und Entschlüsseln benutzt werden, was den Code vereinfacht.

8.15 Der einfache Weg ist natürlich »`tr AEIOU AAAAA`«. Mit weniger Tippaufwand reicht auch »`tr AEIOU A`«.

8.16 Eine Möglichkeit dafür ist

```
$  tr -cs '[:alpha:]' '\n'
```

Das »`-c [:alpha:] \n`« wandelt alle Nichtbuchstaben in Zeilenendezeichen um, die Option `-s` sorgt dafür, dass Folgen dieser Zeilenendezeichen entfernt und durch ein Zeilenende ersetzt werden. Es ist sinnvoll, die Parameter in Anführungszeichen zu setzen, um sicherzustellen, dass die eckigen Klammern nicht von der Shell bearbeitet werden. (Wenn Sie das für deutschsprachige Texte machen wollen, sollten Sie vorher die Umgebungsvariable `LANG` auf `de_DE` o. ä. setzen, damit Umlaute und »ß« als Buchstaben anerkannt werden.)

8.17 Damit das funktioniert, muss das Zeichen »`-`« am Ende von $\langle s_1 \rangle$ stehen – steht es am Anfang, sieht »`-az`« aus wie eine Kommandooption (die `tr` nicht versteht), steht es in der Mitte, so sieht »`a-z`« aus wie ein Bereich. Alternativ, aber tippaufwendiger können Sie das Zeichen »`-`« auch durch »`[=-=]`« umschreiben (siehe Tabelle 8.6), was an einer beliebigen Position stehen darf.

8.18 Mit etwas wie »`cat -T`« (siehe Dokumentation) oder »`od -tc`«.

8.19 Das Kommando hierfür ist »`nl -v 100 -i 2 frosch.txt`«.

8.20 Das Kommando `tac` ist unser Freund:

```
$  tac frosch.txt | cat -n | tac
```

(Statt »`cat -n`« hätte es auch »`nl -ba`« oder ähnliches getan.)

8.21 Beim ersten Kommando betrachtet `wc` alle Eingabedateien separat und gibt auch noch eine Gesamtsumme aus. Beim zweiten Kommando betrachtet `wc` seine Standardeingabe, wo die Tatsache, dass es ursprünglich um drei separate Dateien gibt, nicht mehr in Erscheinung tritt. Darum gibt es beim zweiten Kommando nur eine Ausgabezeile und nicht vier wie beim ersten.

8.24 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.

8.25 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.

8.26 Mit der Lösung von Übung 8.16 ist das Ganze ziemlich einfach:

```
$ tr -cs '[:alpha:]' '\n' | sort -uf
```

Die Option `-u` von `sort` sorgt dafür, dass von einer Folge von gleichen Wörtern nur das erste ausgegeben wird. Durch `-f` werden Groß- und Kleinbuchstaben gleich behandelt (»`LC_COLLATE=de_DE`« würde das mit erledigen).

8.30 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`« (siehe auch Übung 8.26) 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.

9.10 Bei der ersten Kommandozeile müssen Sie insgesamt 10 Sekunden auf eine neue Eingabeaufforderung der Shell warten. Bei der zweiten beträgt die Wartezeit 5 Sekunden, danach wird das zweite `sleep` im Hintergrund gestartet. Bei der dritten Kommandozeile erscheint die neue Eingabeaufforderung sofort, und es werden zwei Hintergrundprozesse erzeugt.

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.)

11.8 Falls Sie `gzip` verwenden möchten, geben Sie »`gzip -9 inhalt.tar`« ein.

11.9 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`«.

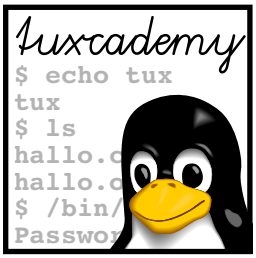
11.10 Versuchen Sie es mit »`tar -cvzf /tmp/homearchiv.tar ~`«.

11.11 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.)



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.

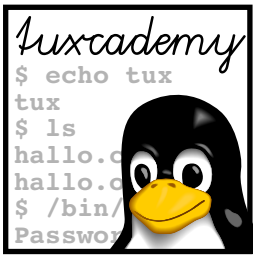
Der Wagen sollte nun den jungen König in sein Reich holen. Der treue Heinrich hob ihn und seine junge Gemahlin hinein, stellte sich wieder hinten hinauf und war voll Freude über die Erlösung seines Herrn. Als sie ein Stück des Weges gefahren waren, hörte der Königssohn, daß es hinter ihm krachte, als 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

LPIC-1-Zertifizierung

C.1 Überblick

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.

Mit der „LPIC-1"-Zertifizierung des LPI können Sie nachweisen, dass Sie über grundlegende Linux-Kenntnisse verfügen, wie sie etwa für Systemadministratoren, Entwickler, Berater oder Mitarbeiter bei der Anwenderunterstützung sinnvoll sind. Die Zertifizierung richtet sich an Kandidaten mit etwa 1 bis 3 Jahren Erfahrung und besteht aus zwei Prüfungen, LPI-101 und LPI-102. Diese werden in Form von computerorientierten Multiple-Choice- und Kurzantworttests über die Prüfungszentren von Pearson VUE und Thomson Prometric angeboten oder können auf Veranstaltungen wie dem LinuxTag oder der CeBIT zu vergünstigten Preisen auf Papier abgelegt werden. Das LPI veröffentlicht auf seinen Web-Seiten unter `http://www.lpi.org/` die **Prüfungsziele**, die den Inhalt der Prüfungen umreißen.

Prüfungsziele

Die vorliegende Unterlage ist Teil eines Kurskonzepts der Linup Front GmbH zur Vorbereitung auf die Prüfung LPI-101 und deckt damit einen Teil der offiziellen Prüfungsziele ab. Details können Sie den folgenden Tabellen entnehmen. Eine wichtige Beobachtung in diesem Zusammenhang ist, dass die LPIC-1-Prüfungsziele nicht dazu geeignet oder vorgesehen sind, einen Einführungskurs in Linux didaktisch zu strukturieren. Aus diesem Grund verfolgt unser Kurskonzept keine strikte Ausrichtung auf die Prüfungen oder Prüfungsziele in der Form „Belegen Sie Kurs x und y, machen Sie Prüfung p, dann belegen Sie Kurs a und b und machen Sie Prüfung q". Ein solcher Ansatz verleitet viele Kurs-Interessenten zu der Annahme, sie könnten als absolute Linux-Einsteiger n Kurstage absolvieren (mit möglichst minimalem n) und wären anschließend fit für die LPIC-1-Prüfungen. Die Erfahrung lehrt, dass das in der Praxis nicht funktioniert, da die LPI-Prüfungen geschickt so angelegt sind, dass Intensivkurse und prüfungsorientiertes „Büffeln" nicht wirklich helfen.

Entsprechend ist unser Kurskonzept darauf ausgerichtet, Ihnen in didaktisch sinnvoller Form ein solides Linux-Basiswissen zu vermitteln und Sie als Teilnehmer in die Lage zu versetzen, selbständig mit dem System zu arbeiten. Die LPIC-1-Zertifizierung ist nicht primäres Ziel oder Selbstzweck, sondern natürliche Folge aus Ihren neuerworbenen Kenntnissen und Ihrer Erfahrung.

C.2 Prüfung LPI-101

Die folgende Tabelle zeigt die Prüfungsziele der Prüfung LPI-101 (Version 4.0) und die Unterlagen, die diese Prüfungsziele abdecken. Die Zahlen in den Spalten für die einzelnen Unterlagen verweisen auf die Kapitel, die das entsprechende Material enthalten.

Nr	Gew	Titel	GRD1	ADM1
101.1	2	Hardware-Einstellungen ermitteln und konfigurieren	–	5–6
101.2	3	Das System starten	–	8–10
101.3	3	Runlevel/Boot-Targets wechseln und das System anhalten oder neu starten	–	9–10
102.1	2	Festplattenaufteilung planen	–	6
102.2	2	Einen Boot-Manager installieren	–	8
102.3	1	Shared Libraries verwalten	–	11
102.4	3	Debian-Paketverwaltung verwenden	–	12
102.5	3	RPM- und YUM-Paketverwaltung verwenden	–	13
103.1	4	Auf der Kommandozeile arbeiten	3–4	–
103.2	3	Textströme mit Filtern verarbeiten	8	–
103.3	4	Grundlegende Dateiverwaltung	6, 11	7.3
103.4	4	Ströme, Pipes und Umleitungen verwenden	8	–
103.5	4	Prozesse erzeugen, überwachen und beenden	9.6	4
103.6	2	Prozess-Ausführungsprioritäten ändern	–	4
103.7	2	Textdateien mit regulären Ausdrücken durchsuchen	7–8	–
103.8	3	Grundlegendes Editieren von Dateien mit vi	5, 7	–
104.1	2	Partitionen und Dateisysteme anlegen	–	6–7
104.2	2	Die Integrität von Dateisystemen sichern	–	7
104.3	3	Das Ein- und Aushängen von Dateisystemen steuern	–	7
104.4	1	Platten-Quotas verwalten	–	7.4
104.5	3	Dateizugriffsrechte und -eigentümerschaft verwalten	–	3
104.6	2	Harte und symbolische Links anlegen und ändern	6	–
104.7	2	Systemdateien finden und Dateien am richtigen Ort platzieren	6, 10	–

C.3 LPI-Prüfungsziele in dieser Schulungsunterlage

103.1 Auf der Kommandozeile arbeiten

Gewicht 4

Beschreibung Kandidaten sollten in der Lage sein, über die Kommandozeile mit Shells und Kommandos zu interagieren. Dieses Prüfungsziel setzt die Bash als Shell voraus.

Wichtigste Wissensgebiete

- Einzelne Shellkommandos und einzeilige Kommandofolgen verwenden, um einfache Aufgaben auf der Kommandozeile zu lösen
- Die Shellumgebung verwenden und anpassen, etwa um Umgebungsvariable zu definieren, zu verwenden und zu exportieren
- Die Kommando-Vorgeschichte verwenden und ändern
- Kommandos innerhalb und außerhalb des definierten Suchpfads aufrufen

Hier ist eine auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme:

- `bash`
- `echo`
- `env`
- `export`
- `pwd`
- `set`
- `unset`
- `man`
- `uname`
- `history`
- `.bash_history`

103.2 Textströme mit Filtern verarbeiten

Gewicht 3
Beschreibung Kandidaten sollten in der Lage sein, Filter auf Textströme anzuwenden.
Wichtigste Wissensgebiete

- Textdateien und Ausgabeströme durch Text-Filter schicken, um die Ausgabe mit Standard-UNIX-Kommandos aus dem GNU-textutils-Paket zu verändern

Hier ist eine auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme:

- `cat`
- `cut`
- `expand`
- `fmt`
- `head`
- `join`
- `less`
- `nl`
- `od`
- `paste`
- `pr`
- `sed`
- `sort`
- `split`
- `tail`
- `tr`
- `unexpand`
- `uniq`
- `wc`

103.3 Grundlegende Dateiverwaltung

Gewicht 4
Beschreibung Kandidaten sollten in der Lage sein, die grundlegenden Linux-Kommandos zur Verwaltung von Dateien und Verzeichnissen zu verwenden.
Wichtigste Wissensgebiete

- Einzelne Dateien und Verzeichnisse kopieren, verschieben und entfernen
- Mehrere Dateien und Verzeichnisse rekursiv kopieren
- Dateien und Verzeichnisse rekursiv entfernen
- Einfache und fortgeschrittene Dateinamen-Suchmuster in Kommandos verwenden
- `find` verwenden, um Dateien auf der Basis ihres Typs, ihrer Größe oder ihrer Zeiten zu finden und zu bearbeiten
- `tar`, `cpio` und `dd` verwenden

Hier ist eine auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme:

- `cp`
- `find`
- `mkdir`
- `mv`
- `ls`
- `rm`
- `rmdir`
- `touch`
- `tar`
- `cpio`
- `dd`
- `file`
- `gzip`
- `gunzip`
- `bzip2`
- `xz`
- Dateisuchmuster

103.4 Ströme, Pipes und Umleitungen verwenden

Gewicht 4
Beschreibung Kandidaten sollten in der Lage sein, Ströme umzuleiten und zu verbinden, um Textdaten effizient zu verarbeiten. Zu diesen Aufgaben gehören das Umleiten der Standardeingabe, Standardausgabe und Standardfehlerausgabe, das Weiterleiten der Ausgabe eines Kommandos an die Eingabe eines anderen Kommandos, die Verwendung der Ausgabe eines Kommandos als Argumente für ein anderes Kommando und das Senden der Ausgabe sowohl an die Standardausgabe als auch eine Datei.
Wichtigste Wissensgebiete

- Umleiten der Standardeingabe, Standardausgabe und Standardfehlerausgabe
- Weiterleiten der Ausgabe eines Kommandos an die Eingabe eines anderen Kommandos (Pipe)
- Verwenden der Ausgabe eines Kommandos als Argumente für ein anderes Kommando
- Senden der Ausgabe sowohl an die Standardausgabe als auch eine Datei

Hier ist eine auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme:

- `tee`
- `xargs`

103.5 Prozesse erzeugen, überwachen und beenden

Gewicht 4
Beschreibung Kandidaten sollten einfache Prozessverwaltung beherrschen.
Wichtigste Wissensgebiete

- Jobs im Vordergrund und Hintergrund ablaufen lassen
- Einem Programm signalisieren, dass es nach dem Abmelden weiterlaufen soll
- Aktive Prozesse beobachten
- Prozesse zur Ausgabe auswählen und sortieren
- Signale an Prozesse schicken

Hier ist eine auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme:

- `&`
- `bg`
- `fg`
- `jobs`
- `kill`
- `nohup`
- `ps`
- `top`
- `free`
- `uptime`
- `pgrep`
- `pkill`
- `killall`
- `screen`

103.7 Textdateien mit regulären Ausdrücken durchsuchen

Gewicht 2
Beschreibung Kandidaten sollten in der Lage sein, Dateien und Textdaten mit regulären Ausdrücken zu manipulieren. Dieses Prüfungsziel umfasst die Erstellung einfacher regulärer Ausdrücke, die mehrere Beschreibungselemente enthalten. Es umfasst ebenfalls den Einsatz von Werkzeugen, die reguläre Ausdrücke zum Durchsuchen eines Dateisystems oder von Dateiinhalten verwenden.
Wichtigste Wissensgebiete

- Einfache reguläre Ausdrücke mit mehreren Beschreibungselementen aufstellen
- Werkzeuge verwenden, die mit regulären Ausdrücken Dateisysteme oder Dateiinhalte durchsuchen

Hier ist eine auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme:

- `grep`
- `egrep`
- `fgrep`
- `sed`
- `regex`(7)

103.8 Grundlegendes Editieren von Dateien mit vi

Gewicht 3

Beschreibung Kandidaten sollten in der Lage sein, Textdateien mit `vi` zu editieren. Dieses Prüfungsziel umfasst `vi`-Navigation, grundlegende `vi`-Modi, Einfügen, Ändern, Löschen, Kopieren und Finden von Text.

Wichtigste Wissensgebiete

- Mit `vi` in einem Dokument navigieren
- Grundlegende `vi`-Modi verwenden
- Text einfügen, ändern, löschen, kopieren und finden

Hier ist eine auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme:

- `vi`
- `/`, `?`
- `h`, `j`, `k`, `l`
- `i`, `o`, `a`
- `c`, `d`, `p`, `y`, `dd`, `yy`
- `ZZ`, `:w!`, `:q!`, `:e!`

104.6 Harte und symbolische Links anlegen und ändern

Gewicht 2

Beschreibung Kandidaten sollten in der Lage sein, harte und symbolische Links auf eine Datei anzulegen und zu verwalten.

Wichtigste Wissensgebiete

- Links anlegen
- Harte und/oder symbolische Links identifizieren
- Dateien kopieren vs. verlinken
- Links verwenden, um Systemadministrationsaufgaben zu unterstützen

Hier ist eine auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme:

- `ln`
- `ls`

104.7 Systemdateien finden und Dateien am richtigen Ort platzieren

Gewicht 2

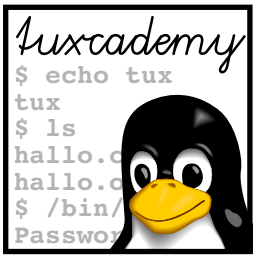
Beschreibung Kandidaten sollten mit dem Filesystem Hierarchy Standard (FHS) vertraut sein und typische Dateiorte und Verzeichnisklassifizierungen kennen.

Wichtigste Wissensgebiete

- Die korrekten Orte von Dateien unter dem FHS kennen
- Dateien und Kommandos auf einem Linux-System finden
- Den Ort und den Zweck wichtiger Dateien und Verzeichnisse gemäß dem FHS kennen

Hier ist eine auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme:

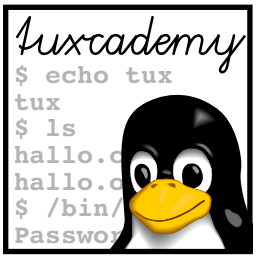
- `find`
- `locate`
- `updatedb`
- `whereis`
- `which`
- `type`
- `/etc/updatedb.conf`



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.