O'REILLY®

Vom Autor des Bestsellers »Linux – kurz & gut«

Produktiv auf der Linux-Kommandozeile

Sicher und souverän mit Linux arbeiten

Daniel J. Barrett

Übersetzung von
Kathrin Lichtenberg

Produktiv auf der Linux-Kommandozeile

Sicher und souverän mit Linux arbeiten

Daniel J. Barrett

Deutsche Übersetzung von Kathrin Lichtenberg

O'REILLY®

Daniel J. Barrett

Lektorat: Ariane Hesse
Übersetzung: Kathrin Lichtenberg
Korrektorat: Sibylle Feldmann, *www.richtiger-text.de*
Satz: III-satz, *www.drei-satz.de*
Herstellung: Stefanie Weidner
Umschlaggestaltung: Karen Montgomery, Michael Oréal, *www.oreal.de*
Druck und Bindung: mediaprint solutions GmbH, 33100 Paderborn

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über *http://dnb.d-nb.de* abrufbar.

ISBN:
Print 978-3-96009-205-6
PDF 978-3-96010-740-8
ePub 978-3-96010-741-5
mobi 978-3-96010-742-2

1. Auflage 2023

Wieblinger Weg 17
69123 Heidelberg

Authorized German translation of the English edition of *Efficient Linux at the Command Line*
ISBN 9781098113407

Dieses Buch erscheint in Kooperation mit O'Reilly Media, Inc. unter dem Imprint »O'REILLY«.
O'REILLY ist ein Markenzeichen und eine eingetragene Marke von O'Reilly Media, Inc. und wird mit Einwilligung des Eigentümers verwendet.

Hinweis:
Dieses Buch wurde auf PEFC-zertifiziertem Papier aus nachhaltiger Waldwirtschaft gedruckt. Der Umwelt zuliebe verzichten wir zusätzlich auf die Einschweißfolie.

Schreiben Sie uns:
Falls Sie Anregungen, Wünsche und Kommentare haben, lassen Sie es uns wissen: *kommentar@oreilly.de*.

5 4 3 2 1 0

Inhalt

Vorwort

Dieses Buch erlaubt Ihnen, Ihre Fähigkeiten auf der Linux-Kommandozeile weiterzuentwickeln und auszubauen, damit Sie schneller, geschickter und effizienter arbeiten können.

Wenn Sie so sind wie die meisten Linux-Anwender, dann haben Sie die ersten Schritte mit der Kommandozeile während der Arbeit gemacht. Vielleicht haben Sie auch ein Buch zur Einführung gelesen oder Dinge während der Installation von Linux einfach ausprobiert. Ich schrieb dieses Buch, um Ihnen dabei zu helfen, weiterzugehen und das Niveau Ihrer Kenntnisse und Fertigkeiten auf der Linux-Kommandozeile zu verbessern. Das Buch ist voller Techniken und Konzepte, die hoffentlich Ihren Umgang mit Linux umkrempeln und Ihre Produktivität ankurbeln. Es führt Sie also über die reinen Grundlagen hinaus.

Eine Kommandozeile ist die einfachste aller Benutzerschnittstellen, sie stellt aber auch die größte Herausforderung dar. Sie ist einfach, weil sie Ihnen nichts weiter präsentiert als einen Prompt, der darauf wartet, dass Sie irgendeinen Befehl ausführen:[1]

```
$
```

Sie ist eine Herausforderung, weil alles, was nach dem Prompt kommt, Ihrer Verantwortung unterliegt. Es gibt keine freundlichen Icons, Buttons oder Menüs, die Sie führen und leiten. Stattdessen ist jeder Befehl, den Sie eintippen, ein kreativer Akt. Das gilt für einfache Befehle wie das Auflisten von Dateien:

```
$ ls
```

genauso wie für aufwendigere Befehle wie diesen hier:

```
$ paste <(echo {1..10}.jpg | sed 's/ /\n/g') \
        <(echo {0..9}.jpg | sed 's/ /\n/g') \
  | sed 's/^/mv /' \
  | bash
```

1 In diesem Buch wird der Linux-Prompt (oder die Eingabeaufforderung, wie man den Prompt auch nennt) als Dollarzeichen dargestellt. Ihr Prompt kann auch anders aussehen.

Falls Sie gerade mit offenem Mund auf diesen Befehl starren und denken: »Was zum Henker ist *das*?« oder »Ich würde niemals so einen komplexen Befehl benutzen«, dann ist dieses Buch genau richtig für Sie.[2]

Was Sie lernen werden

Dieses Buch wird Sie bei drei wichtigen Fertigkeiten schneller und effektiver machen:

- Auswählen oder Konstruieren von Befehlen zum Lösen eines vorliegenden geschäftlichen Problems.
- Effizientes Ausführen dieser Befehle.
- Navigieren durch das Linux-Dateisystem.

Am Ende werden Sie verstehen, was hinter den Kulissen passiert, wenn Sie einen Befehl ausführen, sodass Sie die Ergebnisse besser vorhersagen können (und keine abergläubischen Vorstellungen entwickeln). Sie werden ein Dutzend unterschiedlicher Methoden sehen, um Befehle aufzurufen, und lernen, wann Sie welche von ihnen am besten einsetzen. Sie werden außerdem praktische Tipps erhalten und Tricks kennenlernen, mit denen Sie Ihre Produktivität steigern können, wie etwa:

- das schrittweise Aufbauen komplexer Befehle aus einfachen, um echte Probleme zu lösen, wie etwa das Verwalten von Passwörtern oder das Generieren von 10.000 Testdateien,
- das intelligente Organisieren Ihres Home-Verzeichnisses, damit Sie Zeit sparen und nicht lange nach Dateien suchen müssen,
- das Umwandeln von Textdateien und Abfragen dieser Dateien, wie etwa Datenbanken, um geschäftliche Ziele zu erreichen,
- das Kontrollieren der Point-and-click-Funktionen von Linux von der Kommandozeile aus, wie etwa das Kopieren und Einfügen mithilfe der Zwischenablage und das Beziehen und Verarbeiten von Webdaten, ohne die Hände von der Tastatur nehmen zu müssen.

Vor allem lernen Sie ganz allgemein die besten Vorgehensweisen, damit Sie ganz unabhängig von den Befehlen, die Sie einsetzen, erfolgreicher beim täglichen Einsatz von Linux werden und auf dem Arbeitsmarkt konkurrenzfähiger sind. Dies ist das Buch, das ich mir gewünscht hätte, als ich Linux gelernt habe.

Was dieses Buch nicht ist

Dieses Buch dient nicht der Optimierung Ihres Linux-Computers, damit dieser effizienter läuft. Es macht *Sie* effizienter im Umgang mit Linux.

2 Sie lernen den Zweck dieses mysteriösen Befehls in Kapitel 8 kennen.

Dieses Buch ist auch keine allumfassende Referenz für die Kommandozeile – es gibt Hunderte von Befehlen und Funktionen, die ich nicht erwähne. Hier geht es um Fachwissen. Das Buch lehrt eine sorgfältig ausgewählte Menge an Kommandozeilenwissen in einer praktischen Anordnung und Reihenfolge, um Ihre Fertigkeiten auszubauen. Falls Sie eine Art Referenz wünschen, probieren Sie es mit meinem Taschenbuch »Linux – die wichtigen Befehle kurz & gut« (O'Reilly).

Zielgruppe und Voraussetzungen

Ich gehe davon aus, dass Sie Erfahrungen mit Linux haben; *dieses Buch ist keine Einführung*. Stattdessen ist es für Benutzerinnen und Benutzer gedacht, die ihre Kenntnisse und Fertigkeiten auf der Kommandozeile verbessern wollen, wie Studenten, Systemadministratoren, Softwareentwickler, Sicherheitsverantwortliche, Testingenieure und ganz allgemein Linux-Enthusiasten. Auch erfahrene Linux-Anwender dürften hier noch fündig werden, vor allem wenn sie den Umgang mit Befehlen eher durch Ausprobieren gelernt haben und ihr Verständnis von den dahinterliegenden Konzepten verbessern wollen.

Um wirklich einen Nutzen aus diesem Buch zu ziehen, sollten Sie bereits mit den folgenden Themen vertraut sein (falls Sie es nicht sind, finden Sie in Anhang A Informationen zum Auffrischen Ihrer Erinnerung):

- Anlegen und Bearbeiten von Textdateien mit einem Texteditor wie `vim` (`vi`), `emacs`, `nano` oder `pico`.
- Grundlegende Befehle zum Umgang mit Dateien, wie `cp` (Kopieren), `mv` (Verschieben oder Umbenennen), `rm` (Entfernen oder Löschen) und `chmod` (Ändern der Dateiberechtigungen).
- Grundlegende Befehle zum Betrachten von Dateien, wie `cat` (Betrachten einer ganzen Datei) und `less` (seitenweises Betrachten).
- Grundlegende Verzeichnisbefehle, wie `cd` (Wechseln des Verzeichnisses), `ls` (Auflisten der Dateien in einem Verzeichnis), `mkdir` (Anlegen eines Verzeichnisses), `rmdir` (Entfernen eines Verzeichnisses) und `pwd` (Anzeigen Ihres aktuellen Verzeichnisnamens).
- Grundlagen zu Shell-Skripten: Speichern von Linux-Befehlen in einer Datei, Ausführbarmachen einer Datei (mit `chmod 755` oder `chmod +x`) und Ausführen der Datei.
- Betrachten der in Linux enthaltenen Dokumentation, der sogenannten Manpages, mit dem Befehl `man` (Beispiel: `man cat` zeigt die Dokumentation des Befehls `cat` an).
- Superuser werden mithilfe des Befehls `sudo`, sodass Sie vollständigen Zugriff auf Ihr Linux-System erhalten (Beispiel: `sudo nano /etc/hosts` bearbeitet die Systemdatei */etc/hosts*, die vor normalen Benutzern geschützt ist).

Wenn Sie darüber hinaus auch noch gebräuchliche Kommandozeilenfunktionen wie das Pattern Matching (Musterabgleich) für Dateinamen (mit den Symbolen * und ?), die Umleitung von Eingaben/Ausgaben (< und >) sowie Pipes (|) kennen, sind Sie gut für alles Weitere gerüstet.

Ihre Shell

Ich gehe davon aus, dass Ihre Linux-Shell `bash` ist, die Standard-Shell in den meisten Linux-Distributionen. Immer wenn ich von der »Shell« schreibe, meine ich `bash`. Die meisten Ideen, die ich vorstelle, gelten auch für andere Shells, wie `zsh` oder `dash`; in Anhang B finden Sie Unterstützung beim Übersetzen der Beispiele aus diesem Buch für andere Shells. Ein Großteil des Materials funktioniert auch problemlos und unverändert im Terminal des Apple Mac, das standardmäßig `zsh` einsetzt, obwohl man auch `bash` ausführen kann.[3]

Konventionen in diesem Buch

Folgende typografische Konventionen kommen in diesem Buch zum Einsatz:

Kursiv
: Kennzeichnet neue Begriffe, URLs, E-Mail-Adressen, Dateinamen und Dateierweiterungen.

`Nichtproportionalschrift`
: Wird für Programmlistings verwendet und kennzeichnet beim Einsatz innerhalb eines Absatzes Programmelemente wie Variablen- oder Funktionsnamen, Datenbanken, Datentypen, Umgebungsvariablen, Anweisungen und Schlüsselwörter.

`Nichtproportionalschrift fett`
: Zeigt Befehle oder anderen Text, der wörtlich vom Anwender eingetippt werden sollte. Wird außerdem gelegentlich in Befehlsausgaben eingesetzt, um interessante Stellen hervorzuheben.

`Nichtproportionalschrift kursiv`
: Kennzeichnet Text, der durch nutzereigene Werte oder Werte, die sich aus dem Kontext ergeben, ersetzt werden soll. Wird außerdem für kurze Hinweise rechts von Codelistings verwendet.

`Nichtproportionalschrift hervorgehoben`
: Dient in komplexen Programmlistings dazu, die Aufmerksamkeit auf bestimmte Textteile zu lenken.

3 Die `bash`-Version auf macOS ist uralt, und ihr fehlen wichtige Funktionen. Um die `bash` zu aktualisieren, lesen Sie den Artikel »Upgrading Bash on macOS« (*https://oreil.ly/35jux*) von Daniel Weibel.

Tipp
Dieses Element kennzeichnet einen Tipp oder einen Vorschlag.

Hinweis
Dieses Element kennzeichnet einen allgemeinen Hinweis.

Warnung
Dieses Element kennzeichnet eine Warnung.

Die Verwendung von Codebeispielen

Es steht zusätzliches Material (Codebeispiele, Übungen usw.) zum Herunterladen unter *https://efficientlinux.com/examples* zur Verfügung.

Falls Sie eine technische Frage oder ein Problem bei der Verwendung der Codebeispiele haben, schicken Sie bitte eine E-Mail an *bookquestions@oreilly.com*.

Dieses Buch soll Ihnen helfen, Ihre Arbeit zu erledigen. Falls ein Beispielcode in diesem Buch angeboten wird, dürfen Sie ihn im Allgemeinen in Ihren Programmen und Dokumentationen nutzen. Sie müssen nicht extra Kontakt mit uns aufnehmen und um Erlaubnis bitten, es sei denn, Sie reproduzieren einen beträchtlichen Teil des Codes. So erfordert zum Beispiel das Schreiben eines Programms, das einige Codeschnipsel aus diesem Buch verwendet, keine Erlaubnis. Wollen Sie dagegen Beispiele aus O'Reilly-Büchern verkaufen oder vertreiben, brauchen Sie eine Erlaubnis. Das Beantworten einer Frage, indem Sie dieses Buch und Beispielcode daraus zitieren, erfordert keine Erlaubnis. Nehmen Sie dagegen einen beträchtlichen Teil des Beispielcodes aus diesem Buch in die Dokumentation Ihres Produkts auf, ist eine Erlaubnis notwendig.

Wir wissen eine Nennung zu schätzen, verlangen sie aber nicht. Eine Nennung umfasst üblicherweise Titel, Autor, Verlag und ISBN, zum Beispiel: »*Efficient Linux at the Command Line* von Daniel J. Barrett (O'Reilly). Copyright 2022 Daniel Barrett, 978-1-098-11340-7«.

Falls Sie der Meinung sind, dass Ihre Nutzung der Codebeispiele außerhalb des Bereichs des fairen Einsatzes liegt, den wir oben beschrieben haben, schreiben Sie uns an *permissions@oreilly.com*.

Danksagungen

Es war eine Freude, dieses Buch zu schreiben. Mein Dank geht an die fantastischen Leute bei O'Reilly, vor allem die Lektoren Virginia Wilson und John Devins, die Herstellungsredakteure Caitlin Ghegan und Gregory Hyman, die Content-Managerin Kristen Brown, die Korrektorin Kim Wimpsett, die Indexredakteurin Sue Klefstad und das überaus hilfsbereite Tools-Team. Danken möchte ich außerdem den Fachkorrektoren dieses Buchs, Paul Bayer, John Bonesio, Dan Ritter und Carla Schroder, für ihre vielen hilfreichen Kommentare und Anmerkungen. Danke außerdem an die Boston Linux Users Group für die Titelvorschläge. Ein besonderer Dank geht an Maggie Johnson bei Google für ihre freundliche Erlaubnis, dieses Buch schreiben zu dürfen.

Meinen tief empfundenen Dank richte ich darüber hinaus an Chip Andrews, Matthew Diaz und Robert Strandh, mit denen ich vor 35 Jahren an der The Johns Hopkins University studiert habe. Sie haben mein neu erwachtes und wachsendes Interesse an Unix erkannt und dem Informatikfachbereich zu meinem großen Erstaunen empfohlen, mich als seinen nächsten Systemadministrator anzuheuern. Dieser kleine Akt des Vertrauens hat den weiteren Verlauf meines Lebens verändert. (Robert gebührt außerdem Dank für den Tipp zum Blindschreiben in Kapitel 3.) Danke auch an die Schöpfer und Hüter von Linux, GNU Emacs, Git, AsciiDoc und vielen anderen Open-Source-Werkzeugen – ohne diese klugen und großzügigen Menschen sähe meine Karriere heute vermutlich ganz anders aus.

Und natürlich danke ich an dieser Stelle meiner wunderbaren Familie, Lisa und Sophia, für ihre Liebe und Geduld.

TEIL I
Kernkonzepte

Die ersten vier Kapitel zielen darauf ab, schnell Ihre Effizienz zu erhöhen. Entsprechend werden Konzepte und Techniken behandelt, die sofort nützlich sein sollten. Sie lernen, Befehle mit Pipes zu kombinieren, die Verantwortlichkeiten der Linux-Shell zu verstehen, Befehle aus der Vergangenheit schnell abzurufen und zu bearbeiten und mit großer Geschwindigkeit durch das Linux-Dateisystem zu navigieren.

KAPITEL 1

Befehle kombinieren

Wenn Sie unter Windows, macOS oder den meisten anderen Betriebssystemen arbeiten, verbringen Sie vermutlich die meiste Zeit damit, Anwendungen wie Webbrowser, Textverarbeitungen, Tabellenkalkulationen und Spiele auszuführen. Eine typische Anwendung ist vollgepackt mit Funktionen: Sie enthält praktisch alles, von dem die Entwickler dachten, dass Sie als Anwender es gebrauchen könnten. Deshalb sind die meisten Anwendungen autark bzw. unabhängig, sie hängen nicht von anderen Anwendungen ab. Manchmal kopieren Sie vielleicht Daten zwischen verschiedenen Anwendungen hin und her, aber im Großen und Ganzen sind diese eigenständig und arbeiten getrennt voneinander.

Die Linux-Kommandozeile ist anders. Anstelle von riesigen Anwendungen mit einer Unmenge an Funktionen bietet Linux Tausende kleiner Befehle mit jeweils nur wenigen Funktionen an. Der Befehl `cat` zum Beispiel gibt Dateien auf dem Bildschirm aus, und das war's dann auch schon. `ls` listet die Dateien in einem Verzeichnis auf, `mv` benennt Dateien um und so weiter. Jeder Befehl hat einen einfachen, ziemlich eindeutig definierten Zweck.

Was ist, wenn Sie etwas Komplizierteres erledigen müssen? Keine Sorge. Linux erlaubt Ihnen, ganz einfach *Befehle zu kombinieren*, sodass ihre einzelnen Eigenschaften zusammenspielen, damit Sie Ihr Ziel erreichen können. Diese Art des Arbeitens führt zu einer ganz anderen Denkweise. Anstatt zu fragen: »Welche Anwendung sollte ich starten?«, um ein bestimmtes Ergebnis zu erzielen, stellen Sie die Frage: »Welche Befehle sollte ich miteinander kombinieren?«

In diesem Kapitel lernen Sie, wie Sie Befehle in unterschiedlichen Kombinationen anordnen und ausführen, damit sie das machen, was Sie wollen. Um es nicht zu sehr zu verkomplizieren, stelle ich Ihnen nur sechs Linux-Befehle und ihre grundlegendsten Anwendungen vor, damit Sie sich auf den komplexeren und interessanteren Teil konzentrieren können – das Kombinieren dieser Befehle –, ohne erst noch aufwendig etwas anderes lernen zu müssen. Es ist ein bisschen so, als würden Sie mit nur sechs Zutaten das Kochen oder lediglich mit Hammer und Säge das Tischlern erlernen. (In Kapitel 5 packe ich weitere Befehle in Ihren Linux-Werkzeugkasten.)

Sie kombinieren Befehle mithilfe von *Pipes*, einer Linux-Funktionalität, die die Ausgabe eines Befehls mit der Eingabe eines anderen verbindet. Wenn ich die einzelnen Befehle vorstelle (`wc`, `head`, `cut`, `grep`, `sort` und `uniq`), demonstriere ich auch

gleich ihre Verwendung mit Pipes. Manche Beispiele sind ganz praktisch für den täglichen Einsatz mit Linux, während andere lediglich Beispiele darstellen, um eine wichtige Eigenschaft zu verdeutlichen.

Eingabe, Ausgabe und Pipes

Die meisten Linux-Befehle lesen die Eingabe von der Tastatur, schreiben die Ausgabe auf den Bildschirm oder beides. Linux benutzt spezielle Namen für dieses Lesen und Schreiben:

stdin (die »Standardeingabe«)
: Der Eingabestrom, den Linux von Ihrer Tastatur einliest. Wenn Sie am Prompt irgendeinen Befehl eintippen, liefern Sie Daten an die Standardeingabe.

stdout (die »Standardausgabe«)
: Der Ausgabestrom, den Linux auf Ihren Bildschirm schreibt. Wenn Sie den Befehl `ls` zum Anzeigen von Dateinamen ausführen, erscheinen die Ergebnisse auf der Standardausgabe.

Jetzt kommt die coole Stelle. Sie können die Standardausgabe eines Befehls mit der Standardeingabe eines anderen Befehls verbinden, sodass der erste Befehl den zweiten speist. Beginnen wir mit dem vertrauten Befehl `ls -l` zum Auflisten eines großen Verzeichnisses, wie etwa */bin*, im Langformat:

```
$ ls -l /bin
total 12104
-rwxr-xr-x 1 root root 1113504 Jun  6  2019 bash
-rwxr-xr-x 1 root root  170456 Sep 21  2019 bsd-csh
-rwxr-xr-x 1 root root   34888 Jul  4  2019 bunzip2
-rwxr-xr-x 1 root root 2062296 Sep 18  2020 busybox
-rwxr-xr-x 1 root root   34888 Jul  4  2019 bzcat
...
-rwxr-xr-x 1 root root    5047 Apr 27  2017 znew
```

Dieses Verzeichnis enthält viel mehr Dateien, als Ihr Bildschirm Zeilen anzeigen kann. Daher scrollt die Ausgabe schnell über die Anzeige und verschwindet dann. Schade, dass `ls` die Informationen nicht bildschirmweise anzeigt, pausiert, bis Sie eine Taste drücken, und dann weitermacht. Aber halt: Ein anderer Linux-Befehl hat genau diese Funktion. Der Befehl `less` zeigt eine Datei Bildschirm für Bildschirm an:

```
$ less myfile
```

Sie können diese beiden Befehle miteinander verbinden, da `ls` auf die Standardausgabe schreibt und `less` von der Standardeingabe lesen kann. Verwenden Sie eine Pipe, um die Ausgabe von `ls` an die Eingabe von `less` zu senden:

```
$ ls -l /bin | less
```

Dieser zusammengesetzte Befehl zeigt den Inhalt des Verzeichnisses bildschirmweise an. Der senkrechte Strich (|) zwischen den Befehlen ist das Linux-Pipe-Symbol.[1] Es verbindet die Standardausgabe des ersten mit der Standardeingabe des nächsten Befehls. Jede Kommandozeile, die Pipes enthält, wird als *Pipeline* bezeichnet.

Den Befehlen ist im Allgemeinen nicht bewusst, dass sie Teil einer Pipeline sind. `ls` glaubt, dass es auf den Bildschirm schreibt, obwohl seine Ausgabe an `less` umgeleitet wird. Und `less` glaubt, dass es von der Tastatur liest, obwohl es tatsächlich die Ausgabe von `ls` einliest.

Was ist ein Befehl?

Das Wort *Befehl* besitzt in Linux drei unterschiedliche Bedeutungen, wie Abbildung 1-1 zeigt:

Ein Programm
: Ein ausführbares Programm, das mit einem einzigen Wort benannt ist und über dieses Wort ausgeführt wird, wie `ls`, oder ein ähnliches Merkmal, das in die Shell eingebaut ist, wie `cd` (ein *Shell-Befehl* oder *Shell-Builtin*).[2]

Ein einfacher Befehl
: Ein Programmname (oder Shell-Befehl), dem optional noch Argumente folgen, wie `ls -l /bin`.

Ein zusammengesetzter Befehl
: Mehrere einfache Befehle, die als Einheit behandelt werden, wie etwa die Pipeline `ls -l /bin | less`.

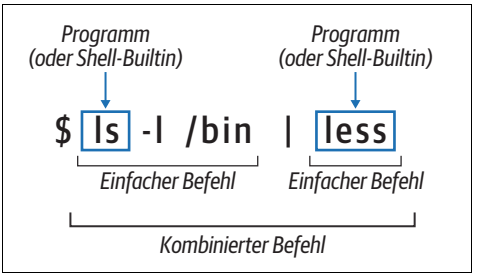

Abbildung 1-1: Programme, einfache Befehle und zusammengesetzte Befehle werden alle als »Befehle« bezeichnet.

In diesem Buch verwende ich das Wort *Befehl* auf all diese Arten. Normalerweise ergibt sich aus dem Kontext, wie ich es meine. Falls nicht, werde ich die spezielleren Begriffe verwenden.

1 Auf PC-Tastaturen erzeugen Sie das Pipe-Symbol mit der Tastenkombination Alt Gr + < und auf Mac-Tastaturen mit der Tastenkombination Alt-7.

2 Der POSIX-Standard nennt diese Form des Befehls ein *Utility*.

Sechs Befehle für den Einstieg

Pipes sind ein wesentlicher Teil des Linux-Know-hows. Beginnen wir das Entwickeln Ihrer Pipeline-Fähigkeiten mit einer kleinen Gruppe von Linux-Befehlen, damit Sie gerüstet sind, egal welche Befehle später auf Sie zukommen.

Die sechs Befehle – `wc`, `head`, `cut`, `grep`, `sort` und `uniq` – besitzen zahlreiche Optionen und Betriebsarten, die ich für den Augenblick einmal überspringe, damit wir uns voll und ganz auf die Pipes konzentrieren können. Wenn Sie mehr über die einzelnen Befehle erfahren wollen, rufen Sie den Befehl `man` auf. Dieser zeigt die jeweils komplette Dokumentation an, zum Beispiel:

```
$ man wc
```

Um die sechs Befehle in Aktion zu demonstrieren, verwende ich eine Datei namens *animals.txt*, die Informationen zu einigen O'Reilly-Büchern enthält und in Beispiel 1-1 zu sehen ist.

Beispiel 1-1: In der Datei animals.txt

```
python    Programming Python         2010    Lutz, Mark
snail     SSH, The Secure Shell      2005    Barrett, Daniel
alpaca    Intermediate Perl          2012    Schwartz, Randal
robin     MySQL High Availability    2014    Bell, Charles
horse     Linux in a Nutshell        2009    Siever, Ellen
donkey    Cisco IOS in a Nutshell    2005    Boney, James
oryx      Writing Word Macros        1999    Roman, Steven
```

Jede Zeile enthält vier Fakten über ein O'Reilly-Buch, die jeweils durch ein einzelnes Tabulatorzeichen (Tab-Zeichen) getrennt sind: das Tier auf dem vorderen Cover, den Buchtitel, das Jahr der Veröffentlichung und den Namen des ersten Autors.

Befehl #1: wc

Der Befehl `wc` gibt die Anzahl der Zeilen, Wörter und Zeichen in einer Datei aus:

```
$ wc animals.txt
  7  51 325 animals.txt
```

`wc` berichtet, dass die Datei *animals.txt* 7 Zeilen, 51 Wörter und 325 Zeichen enthält. Wenn Sie die Zeichen »von Hand« nachzählen, werden Sie einschließlich der Leerzeichen und Tabs nur 318 Zeichen vorfinden, aber `wc` bezieht auch das unsichtbare Newline-Zeichen ein, das am Ende jeder Zeile steht.

Die Optionen `-l`, `-w` und `-c` weisen `wc` an, nur die Anzahl der Zeilen, Wörter bzw. Zeichen auszugeben:

```
$ wc -l animals.txt
7 animals.txt
$ wc -w animals.txt
51 animals.txt
```

```
$ wc -c animals.txt
325 animals.txt
```

Das Zählen ist eine solch nützliche, vielseitig einsetzbare Aufgabe, dass die Autoren von wc den Befehl so gestaltet haben, dass er mit Pipes funktioniert. Er liest von der Standardeingabe, wenn Sie den Dateinamen weglassen, und schreibt auf die Standardausgabe. Verwenden wir ls, um den Inhalt des aktuellen Verzeichnisses aufzulisten und ihn über eine Pipe an wc weiterzuleiten, damit dieser die Zeilen zählt. Diese Pipeline beantwortet die Frage: »Wie viele Dateien sind in meinem aktuellen Verzeichnis sichtbar?«

```
$ ls -1
animals.txt
myfile
myfile2
test.py
$ ls -1 | wc -l
4
```

Die Option -1, die ls anweist, seine Ergebnisse in einer einzigen Spalte auszugeben, ist hier nicht unbedingt nötig. Um zu erfahren, weshalb ich sie verwendet habe, lesen Sie den Kasten »ls ändert sein Verhalten, wenn es umgeleitet wird«.

wc ist der erste Befehl, den Sie in diesem Kapitel gesehen haben. Sie können daher mit Pipes eigentlich noch nicht viel anstellen. Leiten Sie zum Spaß einmal die Ausgabe von wc mit einer Pipe auf sich selbst um. Damit sehen Sie, dass ein Befehl mehr als einmal in einer Pipeline vorkommen kann. Dieser zusammengesetzte Befehl gibt an, dass wc vier Wörter ausgegeben hat: drei Integer und einen Dateinamen:

```
$ wc animals.txt
  7  51 325 animals.txt
$ wc animals.txt | wc -w
4
```

Warum sollten Sie jetzt anhalten? Fügen Sie der Pipeline einen dritten wc-Befehl hinzu und zählen Sie die Zeilen, Wörter und Zeichen in der Ausgabe »4«:

```
$ wc animals.txt | wc -w | wc
      1       1       2
```

Die Ausgabe zeigt eine Zeile (die die Zahl 4 enthält), ein Wort (die Zahl 4 selbst) und zwei Zeichen an. Wieso zwei? Weil die Zeile »4« mit einem unsichtbaren Newline-Zeichen endet.

Das sind jetzt genug alberne Pipelines mit wc. Wenn Sie mehr Befehle kennengelernt haben, zeigt sich auch, wie praktisch Pipelines tatsächlich sind.

ls ändert sein Verhalten, wenn es umgeleitet wird

Im Gegensatz zu nahezu jedem anderen Linux-Befehl ist sich ls darüber bewusst, wenn die Standardausgabe der Bildschirm ist oder ob es umgeleitet wurde (in eine

Pipe oder anderswohin). Der Grund dafür ist Benutzerfreundlichkeit. Wenn der Bildschirm die Standardausgabe ist, ordnet `ls` seine Ausgabe in mehreren Spalten an, die sich besser lesen lassen:

```
$ ls /bin
bash        dir         kmod      networkctl       red     tar
bsd-csh     dmesg       less      nisdomainname    rm      tempfile
...
```

Wird die Standardausgabe dagegen umgeleitet, erzeugt `ls` nur eine einzige Spalte. Ich demonstriere das, indem ich die Ausgabe von `ls` mit einer Pipe an einen Befehl leite, der einfach seine Eingabe reproduziert, wie etwa `cat`:[3]

```
$ ls /bin | cat
bash
bsd-csh
bunzip2
busybox
...
```

Dieses Verhalten kann zu eigenartig aussehenden Ergebnissen führen, wie das folgende Beispiel zeigt:

```
$ ls
animals.txt   myfile   myfile2    test.py
$ ls | wc -l
4
```

Der erste `ls`-Befehl schreibt alle Dateinamen auf eine Zeile, der zweite Befehl jedoch berichtet, dass `ls` vier Zeilen erzeugt hat. Falls Ihnen dieses eigenwillige Verhalten von `ls` nicht bewusst ist, finden Sie diese Diskrepanz möglicherweise verwirrend.

`ls` besitzt Optionen, um sein Standardverhalten außer Kraft zu setzen. Zwingen Sie `ls`, eine einzige Spalte auszugeben, indem Sie die Option `-1` verwenden, oder erzwingen Sie mehrere Spalten, indem Sie die Option `-C` einsetzen.

Befehl #2: head

Der Befehl `head` gibt die ersten Zeilen einer Datei aus. Geben Sie die ersten drei Zeilen von *animals.txt* aus und nutzen Sie dazu `head` mit der Option `-n`:

```
$ head -n3 animals.txt
python      Programming Python       2010    Lutz, Mark
snail       SSH, The Secure Shell    2005    Barrett, Daniel
alpaca      Intermediate Perl        2012    Schwartz, Randal
```

3 Abhängig von Ihren Einstellungen könnte `ls` noch weitere Formatierungseigenschaften verwenden, wie etwa Farben, wenn es auf den Bildschirm schreibt, die jedoch nicht zum Einsatz kommen, wenn es umgeleitet wird.

Falls Sie mehr Zeilen anfordern, als in der Datei enthalten sind, gibt head die gesamte Datei aus (wie cat). Lassen Sie die Option -n weg, liefert head standardmäßig zehn Zeilen (-n10).

Für sich allein genommen, ist head ganz praktisch, um einen Blick auf den Anfang einer Datei zu werfen, wenn einen der restliche Inhalt nicht interessiert. Der Befehl ist schnell und effizient, selbst bei sehr großen Dateien, weil er nicht die ganze Datei lesen muss. Darüber hinaus schreibt head auf die Standardausgabe, sodass es sich gut für den Einsatz in Pipelines eignet. Zählen Sie die Wörter in den ersten drei Zeilen von *animals.txt*:

```
$ head -n3 animals.txt | wc -w
20
```

Für noch mehr Pipeline-Spaß kann head auch von der Standardeingabe lesen. Eine gebräuchliche Anwendung besteht darin, die Ausgabe eines anderen Befehls zu reduzieren, wenn Sie nicht alles davon sehen wollen, etwa bei einer langen Verzeichnisauflistung. Listen Sie zum Beispiel die ersten fünf Dateinamen aus dem Verzeichnis */bin* auf:

```
$ ls /bin | head -n5
bash
bsd-csh
bunzip2
busybox
bzcat
```

Befehl #3: cut

Der cut-Befehl gibt eine oder mehrere Spalten aus einer Datei aus. Geben Sie zum Beispiel alle Buchtitel aus *animals.txt* aus; diese stehen in der zweiten Spalte:

```
$ cut -f2 animals.txt
Programming Python
SSH, The Secure Shell
Intermediate Perl
MySQL High Availability
Linux in a Nutshell
Cisco IOS in a Nutshell
Writing Word Macros
```

cut bietet zwei Möglichkeiten, um zu definieren, was eine »Spalte« ist. Die erste besteht darin, anhand des Felds (-f) zu schneiden, wenn die Eingabe aus Strings (Feldern) besteht, die jeweils durch ein einzelnes Tab-Zeichen getrennt sind. Praktischerweise ist dies genau das Format der Datei *animals.txt*. Der gerade gezeigte cut-Befehl gibt dank der Option -f2 das zweite Feld jeder Zeile aus.

Um die Ausgabe zu kürzen, leiten Sie sie mit einer Pipe an head, sodass nur die ersten drei Zeilen ausgegeben werden: