Kapitel 3 Kommandos: Überblick und Dateiverwaltung

In diesem Kapitel …

- lernen Sie die Philosophie der Kommandozeile kennen
- erleben Sie, dass sich auch mit der Kommandozeile gut arbeiten lässt
- lernen Sie, wie Sie grundlegende Dateioperationen ausführen können
- erfahren Sie einiges über Zugriffsrechte auf Dateien und Verzeichnisse
- lernen Sie fortgeschrittene Kommandos zum Archivieren und Komprimieren von Dateien kennen

3.1 Einleitung: Der Linux-Werkzeugkasten

Die Arbeit auf der Kommandozeile unterscheidet sich grundlegend vom Arbeiten unter der grafischen Oberfläche. Typisch für die grafische Oberfläche sind mächtige Programmpakete, die Komplettlösungen für ganze Problemklassen bereitstellen: The Gimp für Bildbearbeitung, LibreOffice für Büroaufgaben, Kontact für E-Mail, Firefox für den Zugriff auf und die Darstellung von Web-Seiten und so weiter. Zwar gibt es auch E-Mail-Programme und Web-Browser für die Kommandozeile, aber die meisten solcher Programme sind nicht nur eine abgespeckte Version der grafischen Variante. Typische Kommandozeilen-Programme lösen Probleme oft nicht direkt, sondern dienen vor allem als Werkzeug, um spezielle Problemlösungen zu generieren. Normal für die Kommandozeile sind also Programme wie `find`, das nicht viel mehr kann, als Dateien zu finden, oder `grep`, das nicht viel mehr kann, als Dateien nach bestimmten Texten zu durchsuchen. Typische Unix-Programme begnügen sich oft damit, nur eine Sache zu erledigen, diese aber richtig.

Werkzeugkastenprinzip

Dieses »Werkzeugkastenprinzip« ist eine der Grundlagen von Unix – und damit Linux. 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 zum Beispiel nicht jedes Programm eine eigene Sortierfunktion, sondern viele Programme greifen auf das Kommando `sort` zurück. Dieser modulare Aufbau hat neben der Vereinfachung für die Programmierer, die nicht ständig neue Sortierroutinen schreiben oder zumindest in ihre Programme einbauen müssen, den Vorteil, dass bei einer Fehlerkorrektur oder Optimierung von `sort` alle Programme, die darauf zugreifen, ebenfalls profitieren, und das, ohne dass man sie explizit anpassen muss (meistens jedenfalls).

Ein Beispiel mag dies verdeutlichen: Der Aufruf von

```
$ man bash
```

zeigt die Handbuchseite des Programms `bash` an. In Wirklichkeit passiert aber Folgendes: `man` sucht und findet die Handbuchsei-

te `/usr/share/man/man1/bash.1.gz`. Da sie komprimiert ist, ruft es `gzip` auf, um sie zu dekomprimieren. Danach wird die Handbuchseite für die Ausgabe auf einem Terminal durch `groff` formatiert und durch `less` oder `more` seitenweise angezeigt.

Damit dieser Werkzeugkastenansatz erfolgreich funktionieren kann, muss das System aber einige Bedingungen erfüllen:

- Programme müssen miteinander kombiniert werden können. Dies wird zum einen durch das Konzept der Pipeline erreicht, das es erlaubt, eine Art Fließbandverarbeitung zu realisieren (Abschnitt 4.1). Zum anderen stellt die Kommandozeile durch ihre Programmiermöglichkeiten die Infrastruktur bereit, komplexe Kombinationen von Kommandos zu bilden (Abschnitt 12.2).
- Programme müssen miteinander kommunizieren können, sie müssen die gleiche Sprache sprechen. Unter Unix haben sich dafür (zeilenorientierte) Textdateien etabliert. Nur weil die meisten Werkzeuge Textdateien lesen und schreiben (Abschnitt 4.2), sind sie miteinander kompatibel.
- Wenn Sie mehrere Werkzeuge kombinieren müssen, um Ihr jeweiliges Problem zu lösen, kann dies schnell in viel Tipperei ausarten. Die Kommandozeile muss Sie so unterstützen, dass Sie effektiv arbeiten können.

➪ »Opening the Software Toolbox« in der GNU-`coreutils`-Dokumentation (aufzurufen mit »`info coreutils 'Opening the software toolbox'`«)

➪ B. W. Kernighan, R. Pike, *The Unix Programming Environment* (Prentice Hall, Inc., 1984), `http://cm.bell-labs.com/cm/cs/upe/`

3.2 Arbeit auf der Kommandozeile

3.2.1 Der Kommandointerpreter – Die Shell

Was ist eine Shell? Für den Anwender ist eine direkte Kommunikation mit dem Betriebssystem nicht möglich. Das geht nur über Programme, die die Systemaufrufe des Systemkerns ansprechen. Irgendwie müssen Sie also solche Programme starten können. Diese Aufgabe übernimmt die Shell, ein besonderes Anwen-

dungsprogramm, 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 Muschelschale (engl. *shell*) umschließt, das dadurch nicht direkt sichtbar ist. Die Shell selbst ist nur ein Programm unter vielen, das auf das Betriebssystem zugreift.

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 1970er Jahre für »Unix Version 7« von Stephen L. Bourne entwickelt. Diese nach ihm benannte Bourne-Shell enthält alle grundlegenden Funktionen und erfreute sich weiter Verbreitung, ist heute aber nur mehr sehr selten in ihrer ursprünglichen Form anzutreffen. Daneben zählen die C-Shell, an der *University of California in Berkeley* entstanden 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.

Bourne-Shell

C-Shell

Korn-Shell

Bourne-Again-Shell

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

Da die Shell – ungeachtet ihrer zentralen Bedeutung – ein normales Anwendungsprogramm ist, kann sie leicht durch andere Programme ersetzt werden. Wenn im System mehrere verschiedene Shells zur Verfügung stehen (der Regelfall), lässt sich mit folgenden Befehlen zwischen diesen umschalten:

`sh` für eine einfache, an die klassische Bourne-Shell angelehnte Shell (falls vorhanden – bei Linux ist auch `sh` oft die 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`.

Aktuelle Shell?

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

Shell verlassen

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

Bash als Standard

Alle erwähnten Shells unterscheiden sich mehr oder weniger in Syntax und Programmierung, die Auswahl bleibt letztendlich den persönlichen Vorlieben des Anwenders überlassen. Da sich jedoch, wie erwähnt, auf Linux-Systemen die Bash als Standard etabliert hat und diese Shell auch für die LPIC-1-Prüfung vorausgesetzt wird, konzentrieren wir uns auf diese Variante.

3.2.2 Kommandos

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 überlegt sich ein Kommando und gibt dieses per Tastatur oder per Maus ein.
3. Der Rechner führt die erhaltene Anweisung aus.

Eingabeaufforderung

Die Shell zeigt eine Eingabeaufforderung (engl. *prompt*) auf dem Bildschirm an. Die Eingabeaufforderung ist frei konfigurierbar; traditionell besteht sie für normale Benutzer aus »`$ `« oder »`% `« und für den Systemadministrator aus einem »`# `«. (Viele Linux-Distributionen verwenden aufwendigere Eingabeaufforderungen.)

Sollte die Shell am Ende einer Eingabezeile noch nicht zufrieden sein, beispielsweise weil Sie Anführungszeichen aufgemacht, aber nicht geschlossen haben, so macht sie dies durch eine andere Eingabeaufforderung (»`> `«) kenntlich:

```
$ echo 'Anfang
> Ende'
Anfang
Ende
```

Wie ist ein Kommando aufgebaut? Unter einem Kommando verstehen wir grundsätzlich eine längere Folge von Zeichen, die durch die Eingabetaste abgeschlossen und danach von der Shell ausgewertet wird. Kommandos sind zumeist der englischen Sprache entlehnt und ergeben in ihrer Gesamtheit eine eigene »Kommandosprache«. Diese Sprache muss gewissen Regeln, einer Syntax, gehorchen, damit sie von der Shell »verstanden« werden kann.

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 um das eigentliche Kommando.

Wörter

Erstes Wort: Kommando

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 kleingeschrieben und nicht anders verstanden.

Groß- und Kleinschreibung

Parameter

Alle weiteren Wörter in der Kommandozeile sind Parameter, die das Kommando genauer spezifizieren. Die Shell übergibt die zur Trennung dienenden Leerzeichen jedoch bei der Kommandoausführung nicht mit, sie dienen lediglich zur Abgrenzung der Parameter vom Kommando. Deutlich wird dies an einem kleinen Beispiel.

Das Kommando echo gibt alle übergebenen Parameter, getrennt durch jeweils ein Leerzeichen, auf dem Bildschirm aus, also:

```
$ echo schauen wir mal
schauen wir mal
$ echo ein   riesiges    loch
ein riesiges loch
```

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

Optionen

- Parameter, die mit einem Minuszeichen »-« beginnen, heißen Optionen. Diese können auftreten, müssen es aber nicht, sie sind eben »optional«. Bildlich gesprochen handelt es sich dabei um »Schalter«, mit denen Sie Aspekte des jeweiligen Kommandos ein- oder ausschalten können. Möchten Sie mehrere Optionen übergeben, können diese (meistens) hinter einem Minus-

zeichen zusammengefasst werden, die Parameterfolge »`-a -l -F`« entspricht also »`-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 mit zwei Minuszeichen[1] eingeleitet und können nicht zusammengefasst werden: »`bla --fasel --blubb`«.

Lange Optionen

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

Argumente

Arten von Kommandos In Shells gibt es prinzipiell zwei Arten von Kommandos:

Interne Kommandos (zum Beispiel `cd`, `exit` oder `type`) 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.

Externe Kommandos (zum Beispiel `sort`, `who` oder `ls`) Das sind ausführbare Dateien, die im Dateisystem üblicherweise in den Verzeichnissen `/bin` oder `/usr/bin` abgelegt sind. Auch vom Benutzer selbst erstellte Kommandos gehören zu dieser Kategorie.

Extern oder intern?

Um die Art eines Kommandos herauszufinden, können Sie das Kommando `type` benutzen. Übergeben Sie hier den Befehlsnamen als Argument, wird die Art des Kommandos oder der entsprechende Verzeichnispfad auf dem Bildschirm ausgegeben, etwa

```
$ type echo
echo is a shell builtin
$ type sort
sort is /usr/bin/sort
```

[1] Zumindest ist dies bei Programmen des GNU-Projekts so. Bei Programmen, die keine kurzen Optionen unterstützen, können auch lange Optionen mit nur einem Minuszeichen vorkommen.

3.2.3 Die Shell als komfortables Werkzeug

Wer zum ersten Mal von einem grafisch orientierten Betriebssystem aus kommend mit einer Kommandozeile in Berührung kommt, empfindet in der Regel ein gewisses Unbehagen und weiß gar nicht so recht, was er tun soll. Dabei verfügt die Bash über viele hilfreiche Funktionen, die das Arbeiten mit ihr so angenehm machen können, dass sie eine ernstzunehmende Alternative zu grafischen Benutzeroberflächen darstellt:

Kommandoeditor Die Kommandozeilen lassen sich wie mit einem Texteditor bearbeiten. Die Schreibmarke kann also in der Zeile hin- und herbewegt werden, Zeichen können beliebig gelöscht oder hinzugefügt werden usw., bis die Eingabe durch Betätigen der Eingabetaste beendet wird. Dabei imitiert die Bash das Verhalten des Editors `emacs`, wahlweise auch das des `vi`, und deckt damit die beiden bedeutendsten Editoren unter Linux ab.

➪ Abschnitt »READLINE« in `bash`(1)

Kommandoabbruch Bei den vielen Linux-Kommandos kann es durchaus vorkommen, dass ein Name verwechselt oder ein falscher Parameter übergeben wurde. Aus diesem Grund können Sie ein Kommando während der Durchführung abbrechen. Hierzu müssen Sie nur die Tasten Strg+c gleichzeitig drücken. Die Reaktion auf diese Tastenkombination kann unter Umständen etwas auf sich warten lassen.

Die Vorgeschichte Die Shell nimmt alle Tastatureingaben des Anwenders entgegen und merkt sich diese. Sie können sich dann mittels der Cursortasten ↑ und ↓ durch die zuletzt eingegebenen Befehle bewegen. Diese Liste wird beim ordnungsgemäßen Verlassen des Systems in der versteckten Datei `~/.bash_history` gespeichert und steht nach dem nächsten Anmelden wieder zur Verfügung. Die Liste kann im Übrigen auch mit Strg+r durchsucht werden.

`history`

Das Kommando `history` gibt die komplette Vorgeschichte aus (mit Zeilennummern verbrämt). »`history 20`« gibt nur die letzten 20 Kommandos aus (Sie verstehen schon). Die Zeilennummern können Sie verwenden, um Kommandos aus der Vorgeschichte direkt zu recyceln, indem Sie etwas wie

```
$ history
  ⊲⊲⊲⊲⊲
    333  make install
  ⊲⊲⊲⊲⊲
$ !333
make install
⊲⊲⊲⊲⊲
```

eingeben. Über !-Ausdrücke können Sie Kommandos aus der Vorgeschichte in zahllosen fremdartigen und wundervollen Variationen weiterverwenden. Lesen Sie die Details in der Bash-Dokumentation nach und staunen Sie – eingedenk der Tatsache, dass in der LPI-101-Prüfung nur die elementaren Grundlagen gefragt werden. Auch das `history`-Kommando selbst hat übrigens die eine oder andere Option zu bieten.

➭ Abschnitte »HISTORY« und »HISTORY EXPANSION« in `bash(1)`
➭ »`help history`«

Autovervollständigung Eine große Arbeitserleichterung ist die Fähigkeit der Bash zur automatischen Komplettierung von Kommando- bzw. Dateinamen. Durch Drücken der Tab-Taste werden unvollständige Eingaben vollendet, sofern der gewünschte Kommando- oder Dateiname eindeutig identifiziert werden kann. Für die Kommandonamenvervollständigung zieht die Bash neben den internen Kommandos alle ausführbaren Dateien in den Verzeichnissen heran, die in der Variable `PATH` aufgezählt sind; für die Dateinamenvervollständigung betrachtet sie die im aktuellen bzw. angegebenen Verzeichnis befindlichen Dateien. 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. Kommandoname noch immer unvollendet sein kann. Ein erneutes Drücken von Tab listet dann die übrigen Möglichkeiten auf.

Komplettierung von Kommando- bzw. Dateinamen

Die Eingabe eines einzelnen Buchstabens mit zweimaligem Drücken der Tab-Taste liefert eine Liste aller verfügbaren Befehle mit dem gewünschten Anfangsbuchstaben. Dies ist zum Beispiel hilfreich, um sich die Schreibweise selten gebrauchter Befehle ins Gedächtnis zurückzurufen.

Befehlsliste

Tastaturkürzel	Funktion
↑ bzw. ↓	durch zuletzt eingegebene Kommandos blättern
Strg+r	History 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+k	bis Zeilenende löschen
Strg+l	Bildschirm löschen
Strg+c	Kommando abbrechen
Strg+d	Eingabe abbrechen (in der Login-Shell: Abmelden)

Tabelle 3.1: Tastaturkürzel innerhalb der Bash (Auswahl)

In der Ausgabe blättern Neue Zeilen werden erwartungsgemäß immer unterhalb der letzten ausgegeben. Der Bildschirm lässt jedoch nur die Darstellung einer bestimmten Zeilenanzahl zu. Wird diese überschritten, verschwinden die ältesten Zeilen nach oben aus dem sichtbaren Bereich. Das bedeutet jedoch nicht, dass das System diese vergessen hätte. Es verfügt nämlich über einen Speicher, der die Inhalte mehrerer kompletter Bildschirme aufnehmen kann. Möchten Sie sich also weiter zurückliegende Darstellungen noch einmal anschauen, können Sie mit den Tasten ⇑+Bild ↑ bzw. ⇑+Bild ↓ zwischen den Bildschirmseiten vor- und zurückblättern. Eingaben sind aber stets nur am Ende der letzten Seite, also in der aktuellen Zeile, möglich. – Das ist eigentlich keine Leistung der Bash, sondern eine des Treibers für virtuelle Konsolen oder des grafischen Terminalemulators. Der Umfang des gespeicherten alten Materials hängt darum von diesem Programm ab und kann deutlich verschieden ausfallen.

Tastaturkürzelübersicht Tabelle 3.1 gibt eine Übersicht über die wichtigsten in der Bash verwendeten Tastaturkürzel.

Sonderzeichen Die Schreibweise eines Kommandos ist bei der Interpretation durch die Shell entscheidend. So ist es z. B. ein Unterschied, ob ein Kommando groß- oder kleingeschrieben wird (in