

4

Keine Angst vor der Shell

Inhalt

4.1	Warum?	62
4.2	Was ist die Shell?	62
4.3	Kommandos	63
4.3.1	Wozu Kommandos?	63
4.3.2	Wie sind Kommandos aufgebaut?	64
4.3.3	Arten von Kommandos	65
4.3.4	Noch mehr Spielregeln	66

Lernziele

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

Vorkenntnisse

- Grundlegende Kenntnisse im Umgang mit Computern sind hilfreich

4.1 Warum?

Mehr als andere moderne Betriebssysteme basiert Linux (wie Unix) auf der Idee, textuelle Kommandos über die Tastatur einzugeben. Manch einem mag das vor-sintflutlich 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 »Kultur-schock« vergleichbar dem, den ein Mensch des 21. Jahrhunderts erleidet, wenn er plötzlich an den Hof von Kaiser Barbarossa versetzt würde: Kein Internet, schlechte Tischmanieren und furchtbare Zahnärzte!

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

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

4.2 Was ist die Shell?

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



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

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

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

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



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

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

Die Shell übernimmt dabei im Einzelnen folgende Aufgaben:

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

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

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

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

Übungen



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

4.3 Kommandos

4.3.1 Wozu Kommandos?

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

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

3. Der Rechner führt die erhaltene Anweisung aus.

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

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

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

4.3.2 Wie sind Kommandos aufgebaut?

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

Syntax

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

Wörter

Erstes Wort: Kommando

Parameter

Groß- und Kleinschreibung




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



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:

Optionen

- 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«.

lange Optionen

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

Die allgemeine Befehlsstruktur lässt sich also folgendermaßen darstellen: Befehlsstruktur

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

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



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

4.3.3 Arten von Kommandos

In Shells gibt es prinzipiell zwei Arten von Kommandos:

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

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

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

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

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

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

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

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

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

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

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

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

Übungen



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

4.3.4 Noch mehr Spielregeln

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

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

```
$&;(){}[]*?!<>«
```

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

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

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



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

Kommandos in diesem Kapitel

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

Zusammenfassung

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