Louis Schirmer | Uwe M. Schirmer

Shell Script Programmierung

> kapieren & trainieren <

Der einfache Einstieg in die Linux-Automatisierung für Systemadministration, DevOps & Co.

mitp

Als sein Sohn möchte ich insbesondere Uwe für diese Möglichkeit danken, zusammen meine ersten Erfahrungen mit dem Schreiben eines Buchs sammeln zu können.

Und ich möchte meinem Sohn für die gemeinsamen Stunden danken, die wir mit dem Schreiben dieses Buchs verbracht haben.

Gemeinsam geht unser Dank auch an den mitp-Verlag und insbesondere an Janina Vervost für ihre großartige Unterstützung und die großartige Zusammenarbeit.

Danken möchten wir auch unserer Tochter bzw. Schwester Lisa, von der die Zeichnung des Pinguins im Buch stammt.

Louis Schirmer, Uwe Schirmer

Shell Script Programmierung kapieren & trainieren

Der einfache Einstieg in die Linux-Automatisierung

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über https://portal.dnb.de/opac.htm abrufbar.

ISBN 978-3-7475-0800-8

1. Auflage 2025

www.mitp.de

E-Mail: mitp-verlag@lila-logistik.com
Telefon: +49 7953 / 7189 - 079
Telefax: +49 7953 / 7189 - 082

Lektorat: Janina Vervost
Sprachkorrektorat: Jürgen Benvenuti
Covergestaltung: Christian Kalkert
Bildnachweis: © Wanlop / stock.adobe.com
Satz: III-satz, Kiel, www.drei-satz.de

Inhaltsverzeichnis

Einleitung

Ein Dojo ist ein Ort, an dem Kampfkünste gelehrt und von Schülern geübt werden. Neben einfachen Bewegungen werden, vor allem im Karate, auch Katas trainiert. Ein Kata ist ein Kampf gegen imaginäre Gegner, in einer festgelegten Serie von einzelnen Bewegungen. Diese Katas werden geübt, bis man sie verinnerlicht hat und sie zur Routine geworden sind.

Das unterschiedliche Leistungsniveau eines jeden Schülers zeigt sich an farbigen Gürteln. Ein neuer Schüler startet mit einem weißen Gurt. Besteht ein Schüler eine Prüfung, die aus der Vorstellung der neuesten gelernten Kata und dem Anwenden von Basistechniken besteht, dann steigt er einen Gürtel auf, bis er den schwarzen Gurt, und damit den ersten Meistergrad, erreicht.

Aber was hat das mit Shell Scripting zu tun?

Übertragen auf das Shell Scripting sind Katas kurze Übungen, die jeweils einen Aspekt des Schreibens von Skripten trainieren. Unser Dojo ist dieses Buch. Ganz nach dem Prinzip von Dojos, Katas und den tollen bunten Gürteln wollen wir das Shell Scripting vermitteln. Übungen werden wiederholt, bis die Übungsaufgaben elegant und effizient gelöst werden können. So wird Wissen mit Übungen gefestigt und herausragende Leistungen auf dem Weg zum Meister des Shell Scriptings werden mit neuen Gürteln und dem Aufstieg in höhere Kapitel belohnt.

Was lernst du in diesem Buch?

Das Ziel dieses Buchs ist es, dir eine umfassende Einführung in das Thema Shell Scripting und den Umgang mit dem Terminal zu bieten. Du lernst nicht nur die Grundlagen, sondern auch fortgeschrittene Techniken, um effizient und effektiv Skripte zu schreiben, Systemprozesse zu steuern und Textdateien zu bearbeiten.

Welche Kenntnisse werden vorausgesetzt und an wen richtet sich das Buch?

Das Buch setzt grundlegende Kenntnisse im Umgang mit Computern voraus, aber es ist keine tiefgehende Vorerfahrung mit Linux oder Shell Scripting nötig. Es richtet sich an Anfänger sowie fortgeschrittene Nutzer, die ihre Kenntnisse vertiefen und erweitern möchten. Es ist ideal für Studierende, IT-Professionals und alle, die ein Interesse an Systemadministration und Automatisierung haben.

Wie ist das Buch aufgebaut und welche Themen werden abgedeckt?

Das Buch ist in sieben Kapitel gegliedert, die aufeinander aufbauen:

Kapitel 1: Einführung in Terminal und Shell

Wir starten mit einer Einführung in Terminal und Shell sowie das Bearbeiten von Dateien und Verzeichnissen. Wir wollen dir hier einen ersten Überblick über wichtige Befehle geben.

Kapitel 2: Grundlagen des Shell Scriptings

Nach den Grundlagen zu Linux und zum Terminal beginnen wir mit dem Shell Scripting. Wir zeigen dir den Aufbau eines Skripts, wie man es ausführt, Ausgaben, Variablen, Arithmetik und bedingte Befehlsausführung.

Kapitel 3: Erweiterte Skriptfunktionen

Dieses Kapitel umfasst Kommandozeilenparameter, Rückgabewerte, Schleifen und Arrays, die dir mehr Kontrolle über deine Skripte geben. Außerdem erhältst du ein tieferes Verständnis dafür, Skripte zu beenden und Fehler zu finden.

Kapitel 4: Umgang mit Ein- und Ausgaben

Wir widmen uns hier der Umleitung von Ein- und Ausgaben und der Verwendung von Pipes, um Befehle zu verketten. Zudem lernst du, Dateien zu zerlegen, zu verbinden, zu sortieren und zu verändern, damit du sicher mit Textdateien umgehen kannst.

Kapitel 5: Fortgeschrittene Textverarbeitung

Hier lernst du reguläre Ausdrücke kennen, die mächtige Such- und Filtermöglichkeiten bieten. Wir bringen dir bei, Dateien nach bestimmten Mustern zu verarbeiten und Unterschiede in Dateien zu finden. Wir zeigen dir auch, wie du Skripte sauber formatierst und die Ausführungszeiten misst.

Kapitel 6: Prozesse und Signale

Dieses Kapitel gibt dir einen tieferen Einblick in die Arbeitsweise eines Linux-Betriebssystems (Prozesse, Jobs und Signale). Zudem wirst du Funktionen kennenlernen und deren Zusammenarbeit mit Signalen verstehen.

Kapitel 7: Systemkonfiguration

Zum Abschluss lernst du, wie du dein System mithilfe von Konfigurationsdateien anpassen kannst. Wir betrachten die Systeminitialisierung und den Login-Prozess sowie zeitgesteuerte Programm- und Skriptstarts. Ein kurzer Exkurs in die Zeichencodierung rundet dieses Kapitel ab.

Downloads zum Buch

Am Ende jedes Kapitels findest du Gürtelprüfungen, mit denen du dein Können testen und das Gelernte ausprobieren kannst. Folgst du allen Prüfungen, entsteht bis zum Ende des Buchs ein vollständiges Skript.

Die Musterlösungen für die Gürtelprüfungen und die Übungen in den Kapiteln gibt es als Dateien zum Download unter:

`www.mitp.de/0799`

Kapitel 1

Weißgurt: Einführung in Terminal und Shell

Im ersten Kapitel wenden wir uns Grundlagen zu, die die Basis für späteres Wissen bilden, den Umgang mit dem System vereinfachen und dessen Aufbau verständlicher machen.

Wir erklären dir, was Terminal und Shell sind, wie du sie verwendest und wie du mit Dateien und Verzeichnissen arbeitest. Wir geben einen Überblick über wichtige Befehle und du bekommst die Möglichkeit, das Gelernte in einer Übung anzuwenden.

Um dieses Kapitel nicht aufzublähen, haben wir uns dazu entschlossen, nur eine abschließende Übung einzubauen. Du kannst die eingefügten Code-Beispiele jederzeit bei dir lokal ausprobieren.

1.1 Die richtige Linux-Distribution

Will man Linux installieren, muss man eine Entscheidung zwischen Kali Linux, Debian, Ubuntu und vielen anderen Distributionen[1] treffen. Aber was ist der Unterschied?

Es gibt mittlerweile mehr als 600 verschiedene Linux-Distributionen. All diese Distributionen unterscheiden sich voneinander und sind oft auf ganz bestimmte Anwendungsfälle zugeschnitten. Eine Gemeinsamkeit haben sie aber alle: Sie bauen auf einem Linux-Kernel auf.

Ein Kernel ermöglicht die Kommunikation zwischen Software und Hardware. Er sorgt dafür, dass die Kommunikation zwischen dem Benutzer und seinem Gerät trotz unterschiedlicher Hardware funktioniert. Die verschiedenen Linux-Distributionen nutzen verschiedene Versionen dieses Kernels, der jeweils auf andere Anforderungen und die eigene Hardware zugeschnitten ist. Eine Distribution enthält aber nicht nur verschiedene Varianten des Kernels, sondern auch unterschiedliche Software-Pakete.

1 In der Informatik spricht man von einer Distribution, wenn man ein individuell zusammengestelltes Paket von Software meint.

Der Linux-Kernel wird aber auch in Smart-Devices, dem Windows-Subsystem für Linux (WSL), Raspberry Pi OS und auf Android-Smartphones genutzt.

Welche Distribution für die eigenen Zwecke die passendste ist, darüber sollte man sich am besten selbst informieren. Wir haben uns hier für Ubuntu entschieden, weil es weitverbreitet und sehr universell einsetzbar ist. Du kannst aber auch eine andere Distribution wählen.

Unix

Häufig wirst du bei Recherchen zu Linux-Themen auf das Wort *Unix* stoßen. Dabei handelt es sich um eines der ersten, auf der Programmiersprache C basierenden Betriebssysteme. Linux orientierte sich in seiner Entstehungszeit sehr an Unixund ähnelt ihm deshalb stark. Man zählt Linux zu den Unix-ähnlichen Betriebssystemen.

1.2 Die Verzeichnisstruktur

Der FHS (»Filesystem Hierarchy Standard«) definiert einen Standard, nach dem der Großteil der Linux-Distributionen ihr Dateisystem, also Verzeichnisse und ihre Inhalte, sortiert und benennt. Durch die Verwendung eines Standards wird dafür gesorgt, dass diese Linux-Distributionen eine einheitliche Verzeichnisstruktur haben. Das ermöglicht es, Programme für verschiedene Distributionen zu entwickeln, ohne dass man dabei unterschiedliche Verzeichnisstrukturen berücksichtigen muss.

Abweichungen vom FHS

Einzelne Linux-Distributionen können von diesem Standard abweichen (z.B. GoboLinux oder NixOS). Die großen Linux-Distributionen wie Ubuntu, Kali Linux oder Debian halten sich jedoch daran.

Die Wurzel deines Verzeichnissystems ist das Root-Verzeichnis (dt. Wurzel-Verzeichnis). Alle Datei- und Verzeichnispfade nehmen hier ihren Ursprung. Stellst du dir das Verzeichnissystem als Baumdiagramm vor, dann ist das Root-Verzeichnis die Wurzel.

Ein Systempfad beginnt immer mit einem Schrägstrich (»/«) für das Root-Verzeichnis. Der FHS enthält eine ganze Liste an Verzeichnissen, die im Root-Verzeichnis liegen. Verzeichnisse sowie ihre untergeordneten Verzeichnisse und Dateien werden mit einem »/« voneinander getrennt. Gegebenen Pfaden kannst du also von links nach rechts, bis zu ihrem Ende, aus dem Root-Verzeichnis folgen.

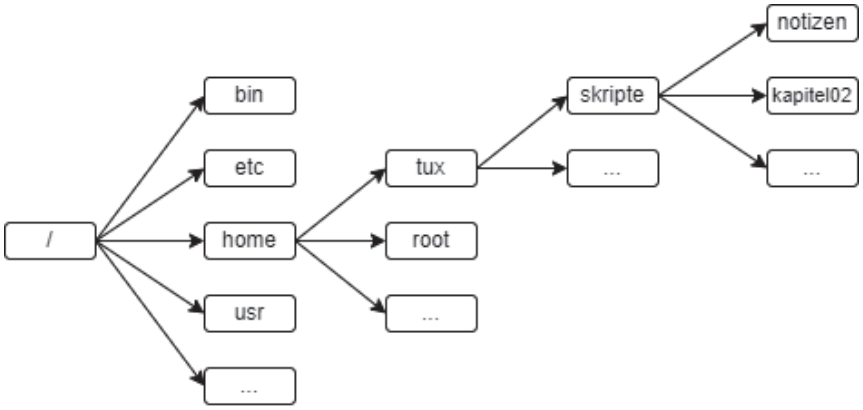

Abb. 1.1: Ausschnitt einer Verzeichnisstruktur

Der Pfad `/home/tux/wichtig` könnte also auf eine Datei `wichtig` verweisen, die in dem Verzeichnis `tux` im Verzeichnis `home` im Root-Verzeichnis liegt. `wichtig` könnte aber auch ein Unterverzeichnis im Verzeichnis `tux` sein. Ob `wichtig` ein Verzeichnis oder eine Datei ist, sieht man nur, wenn man sich `wichtig` genauer anschaut. Das geht z.B. mit dem Befehl `ls`, den wir in Abschnitt 1.3.4 erklären.

Weiterführende Informationen zum FHS

Mehr Informationen zum FHS findest du auf folgenden Seiten:

- `https://de.wikipedia.org/wiki/Filesystem_Hierarchy_Standard`
- `https://refspecs.linuxfoundation.org/FHS_3.0/fhs/index.html` (Die Spezifikation auf Englisch)

1.2.1 Das Home-Verzeichnis

Das Home-Verzeichnis ist eines der vom FHS festgelegten Verzeichnisse im Root-Verzeichnis.

Im Home-Verzeichnis `/home` hat jeder Benutzer ein eigenes Verzeichnis. Für den Benutzer Tux liegt das Home-Verzeichnis in `/home/tux`. Den Namen Tux wirst du immer wieder lesen, wir nutzen ihn als Platzhalter für den aktiven Benutzer. Taucht der Name Tux also irgendwo auf, dann ersetze ihn einfach mit deinem eigenen Benutzernamen.

Das Home-Verzeichnis eines Benutzers ist der Ort, an dem benutzerspezifische Einstellungen und Dateien abgelegt werden können. Andere Benutzer haben keinen Zugriff auf dein Home-Verzeichnis, nur du hast alle verfügbaren Berechtigungen.

Als Kurzform für das Home-Verzeichnis des aktiven Benutzers kann man die Tilde »~« nutzen.

Ist Tux also gerade der aktive Benutzer und will ein Programm im Verzeichnis `skripte` seines Home-Verzeichnisses ausführen, dann ist `~/skripte` die Kurzform für `/home/tux/skripte`. Wechselt jetzt der Benutzer und Pax möchte ein Skript in seinem Home-Verzeichnis ausführen, greift `~/skripte` auf `/home/pax/skripte` zu.

1.3 Terminal und Shell

Ein Terminal ist ein Programm, das in einem Fenster eine auf Text basierende Interaktion mit dem Betriebssystem ermöglicht. Dazu läuft im Terminal ein weiteres Programm, die Shell. Eine Shell ist ein Programm, das Befehle interpretiert und die zugrunde liegenden Programme ausführt.

Befehle sind Programme

Auch Befehle sind Programme, die von der Shell ausgeführt werden. Es kann am Anfang verwirren, dass Programme in Programmen ausgeführt werden. Um diese Verschachtelung zu verstehen, hilft es, sich vor Augen zu führen, dass das Terminal ein Programm ist, das die Shell als Programm ausführt, welche Befehle (Programme) ausführt.

Auch die grafische Benutzeroberfläche ist ein separates Programm.

Genau wie bei den Linux-Distributionen gibt es nicht die eine Shell. Jede Distribution hat zwar eine Shell, die das Terminal als Standard nutzt, meistens werden aber mehrere Shells mitgeliefert.

Ähnlich dem FHS gibt es noch einen weitreichenderen Standard für Systemkompatibilität, den POSIX – kurz für »Portable Operating System Interface for Unix« (dt. Portable Betriebssystem-Schnittstelle für Unix) –, welcher eine Reihe an Standards für Betriebssysteme definiert. Diese Standards befassen sich mit Schnittstellen und Verhaltensweisen von Unix-ähnlichen Betriebssystemen. Das heißt, dass grundlegende Funktionen wie das Arbeiten mit Dateien und einige grundlegenden Befehle der Shells gleich oder sehr ähnlich aufgebaut sind. Alles, was über die Funktionalitäten des POSIX hinausgeht, kann sich allerdings unterscheiden und ist dann nicht mehr zwischen den verschiedenen Shells kompatibel.

`bin` und `sbin`

Das Verzeichnis `/bin` enthält alle Befehle, die vom Benutzer und vom Administrator ausgeführt werden können. Das Verzeichnis `/sbin` enthält Befehle, die nur

vom Administrator ausgeführt werden dürfen. `bin`- und `sbin`-Verzeichnisse gibt es nicht nur im Wurzel-Verzeichnis, man findet sie auch an verschiedenen anderen Stellen im System (z.B. `/usr/bin`, `/usr/sbin`, `/usr/local/bin`, `/usr/local/sbin`, `~/bin`).

Die Shells sind im Verzeichnis `/bin` gespeichert. Es gibt drei Shells, von denen man schon einmal gehört haben sollte – tiefergehende Details werden wir hier jedoch nicht behandeln.

Die Bourne-Shell wurde 1979 veröffentlicht. Ihr Programmname ist *sh* und sie lag früher im Pfad `/bin/sh`. Mittlerweile verweist `/bin/sh` allerdings meistens als symbolischer Link auf die Standard-Shell der gewählten Distribution. Oft ist das dann die *Bash*-Shell, auf Ubuntu allerdings die *Dash*-Shell.

Die Bash oder auch Bourne-Again Shell ist eine Weiterentwicklung der Bourne Shell und wurde 1986 veröffentlicht. Die Bash liegt in /bin/bash. Sie ist die wohl am weitesten verbreitete Linux-Shell und erweitert die sh um viele Funktionalitäten. Bei der Ausführung eines Terminals wird auf Ubuntu die Bash als Shell genutzt.

Auch gehört haben solltest du von der Debian-Almiquist-Shell, kurz Dash. Diese liegt in /bin/dash und ist bei der Ausführung von Systemskripten die Standard-Shell von Ubuntu. Sie hat einige Unterschiede zur Bash, auch wenn beide den POSIX-Standard erfüllen. Vor allem kann Dash durch die Geschwindigkeit punkten, da sie deutlich schneller als die meisten anderen Shells ist, inklusive der Bash. Sie hat aber weniger Funktionalitäten.

In diesem Buch verwenden wir für alle Beispiele die Bash als Shell, weil sie am weitesten verbreitet ist. Damit die Skripte in anderen Shells funktionieren, musst du sie anpassen. Wo es Sinn ergibt, weisen wir auf Unterschiede zu anderen Shells hin. Wegen der zahlreichen Shells und ihrer jeweils eigenen Syntax können wir aber nicht auf alle Besonderheiten eingehen.

Weiterführende Informationen zur Shell

Mehr Informationen zur Shell und den verschiedenen Shells gibt es hier:

`https://de.wikipedia.org/wiki/Unix-Shell`

1.3.1 Das Terminal bedienen

Um das Terminal zu öffnen, gibt es mehrere Wege. Zum einen kann es mit dem entsprechenden Icon geöffnet werden, wie jede andere Anwendung auch, wenn eine grafische Benutzeroberfläche installiert ist. Alternativ kann unter Ubuntu auch die Tastenkombination Strg+Alt+T genutzt werden.

Jede Zeile des Terminals hat den gleichen Aufbau, auch wenn sich die angezeigten Informationen von Distribution zu Distribution und sogar von Installation zu Installation unterscheiden können. Jede Zeile besteht aus einem sogenannten *Prompt*, hinter dem ein Cursor blinkt. Was als Prompt angezeigt wird, kann konfiguriert werden. Auf unserem System sieht der Prompt folgendermaßen aus:

```
tux@DESKTOP-11088GB:~$
```

Hier besteht der Prompt aus dem Benutzernamen des aktiven Benutzers. Getrennt mit einem `@`-Symbol folgt der Name des Rechners. Auf ein `:` folgt der aktuelle Pfad, das sogenannte *Arbeitsverzeichnis*. Beim Start vom Terminal ist das standardmäßig das Home-Verzeichnis, welches als Tilde ~ abgekürzt wird. Ein `$`-Symbol markiert das Ende des Prompts bzw. der Eingabeaufforderung.

Um die Beispiele übersichtlicher zu machen, werden wir in diesem Buch den ersten Teil des Prompts mit »...« abkürzen, sodass lediglich der Pfad des Arbeitsverzeichnisses angezeigt wird. Der Rest ist die meiste Zeit unwichtig.

```
...:~$
```

Direkt hinter dem Prompt, an der Position des blinkenden Cursors, können eigene Texte und Befehle geschrieben und durch Drücken der Enter-Taste ausgeführt werden.

Jetzt schließen wir das Terminal wieder. Neben dem Schließen per Icon gibt es alternativ die Tastenkombination Strg+D sowie den Befehl `exit`.

```
...:~$ exit
```

Copy and Paste im Terminal

Üblicherweise kannst du nicht einfach mit Strg+C und Strg+V Text kopieren und einfügen. Wenn die Funktionen unterstützt werden, weichen die Tastenkombinationen meist ab. In der Bash sind diese üblicherweise Strg+Shift+C und Strg+Shift+V .

1.3.2 Befehle ausführen

Als Erstes müssen wir den Unterschied zwischen den Begriffen Befehl, Programm und Prozess klären. Ein *Programm* ist etwas, das ausgeführt werden kann (z.B. eine Binärdatei oder ein Skript). Ein *Prozess* ist ein Programm, das gerade läuft. Ein Programm kann mehrfach gestartet werden, dann laufen mehrere Prozesse des gleichen Programms. Ein *Befehl* ist eine Anweisung, die du in ein Terminal eingibst, um z.B. ein zugrunde liegendes Programm auszuführen.

Ein Befehl setzt sich aus drei Bestandteilen zusammen: dem Namen des Befehls, den Flags und den Parametern.

Das Leerzeichen hat die Funktion, einzelne Bestandteile voneinander zu trennen. Die einzelnen Bestandteile werden *Tokens* genannt. Das ermöglicht, dass einem Befehl z.B. mehrere Parameter oder Flags als einzelne Tokens (durch Leerzeichen voneinander getrennt) übergeben werden können.

Der erste Token ist der Name eines Befehls, den wir aufrufen wollen. Der Name, den wir eingeben, ist meistens eine Abkürzung. Für den Befehl `list` schreiben wir `ls`, für `change directory` schreiben wir `cd`. Was diese Befehle machen, erklären wir im Abschnitt 1.3.4.

Bei den meisten Befehlen stellen die folgenden Tokens die Flags dar. Diese sind in der Regel optional und passen das Standardverhalten des Befehls an. Die meisten Befehle nehmen auch mehrere Flags entgegen. Um zu kennzeichnen, dass es sich um Flags handelt, werden ein oder zwei Bindestriche vor den Namen des Flags gesetzt. Das kann dann so aussehen: `-d` oder `--directory`. Flags gibt es dabei häufig in einer Kurzform (ein Strich und ein Buchstabe) und in einer Langform (zwei Striche und der ausgeschriebene Name des Flags).

Was ein Flag bei einem Befehl bewirkt, kann je nach Befehl unterschiedlich sein. Das Flag `-d` (`--directory`) bewirkt beim `ls` Befehl z.B., dass nur Verzeichnisse aufgelistet werden. Beim `touch`-Befehl, den wir weiter unten erklären, kann man mit dem Flag `-d` (`--date`) das Erstellungsdatum einer Datei festlegen.

Übergibst du dem Befehl mehrere Flags, die aus einem Buchstaben bestehen und selbst keine weiteren Angaben benötigen, so kannst du diese zusammenfassen. Möchtest du einem Befehl die Flags `-d`, `-e` und `-f` übergeben, kannst du dies auch als `-def` schreiben. Bei Flags, die Parameter übergeben bekommen, funktioniert das nicht, weil die Parameter den Flags dann nicht mehr zugeordnet werden können.

Die letzten Tokens sind die *Argumente* oder auch *Parameter*. Diese sind die Werte, die man einem Befehl zu seiner Ausführung übergibt. Nicht alle Befehle benötigen Parameter, manchmal benötigen Befehle aber auch mehrere.

Oft sind Parameter Pfade. Pfade können als relative oder absolute Pfade übergeben werden. Einen absoluten Pfad zu übergeben bedeutet, einen Pfad vom Root-Verzeichnis aus anzugeben. Das sieht dann z.B. so aus: `/home/tux/einVerzeichnis/eineDatei`.

Relative Pfade stehen immer im Zusammenhang zum Arbeitsverzeichnis. Der angegebene relative Pfad wird immer an das Arbeitsverzeichnis angehängt. Ist das Arbeitsverzeichnis also das Home-Verzeichnis (»~«) und wird als Parameter `einOrdner` übergeben, dann wird im Hintergrund `~/einOrdner` daraus zusammengesetzt. Hier können auch längere Pfade angegeben werden. So wird `einOrdner/nochEinOrdner/eineDatei` zu `~/einOrdner/nochEinOrdner/eineDatei`.

```
...:~$ cd ..
...:/home$ cd tux
...:~$ cd ..
...:/home$ cd /home/tux
```

Ob es sich um einen absoluten oder relativen Pfad handelt, ist daran zu erkennen, ob das erste Zeichen ein »/« ist. Das »/« deutet darauf hin, dass der Pfad vom Root-Verzeichnis aus zu lesen ist (absoluter Pfad).

Wenn ein Befehl ausgeführt wird, dann gibt es Situationen, in denen man die Ausführung vorzeitig abbrechen möchte. In diesem Fall kannst du die Tastenkombination Strg+C nutzen.

1.3.3 Hilfe zur Selbsthilfe

Linux kennt sehr viele Befehle, teilweise sogar mehrere, die fast das Gleiche machen. Deswegen ist es wichtig, die verschiedenen Möglichkeiten zu kennen, die Funktionen eines Befehls, seine Parameter und Flags herauszufinden.

Jeder Befehl in Linux kennt das Flag `--help` zur Anzeige eines kurzen Hilfetexts.

```
...:~$ cd --help
cd: cd [-L|[-P [-e]] [-@]] [dir]
    Change the shell working directory.
```

Alternativ kannst du in der Bash-Shell auch den `help`-Befehl verwenden. Andere Shells haben ihre eigenen bzw. andere Dokumentationssysteme. Der `help`-Befehl zeigt auch Hilfetexte über sich selbst an.

```
...:~$ help cd
cd: cd [-L|[-P [-e]] [-@]] [dir]
    Change the shell working directory.
```

Bei beiden Varianten sollte der gleiche Hilfetext angezeigt werden.

Für jeden Befehl, der unter Linux installiert wird, werden zusätzlich *Manpages* (auf Deutsch etwa Handbuchseiten) installiert. Die *Manpages* können über den `man`-Befehl angezeigt werden. Die angezeigten Handbücher zu einem Befehl können länger sein und auch Kombinationen von Parametern und Flags mit Beispielen erklären. Sie werden in einem Programm angezeigt, das sich über Tastenkürzel bedienen lässt. Mit H kannst du eine Hilfeseite mit den Tastenkürzeln für das Anzeigeprogramm aufrufen. Über Q kannst du das Programm wieder verlassen.

```
cd(n)       Tcl Built-In Commands           cd(n)

_____________________________________________________

NAME
cd - Change working directory
SYNOPSIS
cd ?dirName?

_____________________________________________________

DESCRIPTION
Change  the  current working directory to dirName, or to the home direct
ory (as specified in the HOME environ-
          ment variable) if dirName is not given.  Returns an empty stri
ng.  Note that the ...
```

Neben Handbüchern zu einzelnen Befehlen kannst du dir auch Informationen zu API-Funktionen, Konzepten, Konfigurationsdateien und Dateiformaten anzeigen lassen.

Der `info`-Befehl ist ein anderes Dokumentationssystem, das aus dem GNU-Projekt stammt. Es bietet Texte mit Links zwischen Textpassagen (aus der Zeit vor dem Internet mit seinen Webseiten). Ein `info`-Handbuch ist wie ein digitales Buch mit Inhaltsverzeichnis und einem durchsuchbaren Index, der das Auffinden von Informationen erleichtert. Links werden dabei durch unterstrichene Texte dargestellt, die wie Links im Browser funktionieren. Du wählst sie aus, indem du den Cursor mit den Pfeiltasten auf einen Link bewegst und die Enter-Taste betätigst. Über die Leertaste kannst du zur nächsten Seite blättern und Q beendet auch hier das Anzeigeprogramm.

Was ist das GNU-Projekt?

Das GNU-Projekt wurde 1983 von Richard M. Stallman ins Leben gerufen, um ein vollständiges Betriebssystem auf der Basis von freier Software zu schaffen. Im Rahmen des Projekts sollte ein vollständig freies, Unix-ähnliches Betriebssystem entstehen. Da ein eigener Kernel des Projekts bis heute nicht für den praktischen Einsatz geeignet ist, wird das System mit dem Linux-Kernel kombiniert und GNU/Linux oder kurz Linux genannt. Die Kombination von GNU und dem Linux-Kernel bildet ein ausgereiftes, stabiles Betriebssystem. Dabei kommen die Shell, Coreutils, einige Bibliotheken und Compiler wie der gcc von GNU.

Der Name GNU ist ein rekursives Akronym von »GNU is Not Unix« (»GNU ist Nicht Unix«) und wird, um Verwechslungen zu vermeiden, wie das Tier Gnu im Deutschen ausgesprochen. Das Logo ist der Kopf einer afrikanischen Gnu-Antilope.

Allen oben aufgeführten Hilfsprogrammen und Flags ist gemeinsam, dass die Hilfetexte meistens auf Englisch geschrieben sind. Es besteht zwar die Möglichkeit, Hilfetexte und *Manpages* auf Deutsch zu installieren, das funktioniert aber nur unvollständig und man muss dann mit einem Mix aus deutschen und englischen Texten leben, weil nicht für alle Befehle deutsche Hilfetexte verfügbar sind.

Dazu kommt, dass sich die Informationen und Inhalte der verschiedenen Hilfesysteme überschneiden können.

Weitere Informationen zu Befehlen findest du auch im Internet z.B. auf den Webseiten der jeweiligen Distribution[2]. Hier kannst du im Suchschlitz auf der Seite (oben rechts) den Namen eines Befehls eingeben und dir so eine ziemlich detaillierte Beschreibung anzeigen lassen, bei größeren Distributionen neben Englisch in mehreren weiteren Sprachen, darunter auch auf Deutsch.

1.3.4 Navigieren im Dateisystem

Das Arbeitsverzeichnis ist der Pfad, in dem standardmäßig alle Befehle ausgeführt werden und von dem aus alle relativen Pfade betrachtet werden.

Wie zuvor schon erwähnt, kannst du das Arbeitsverzeichnis, also den Pfad, in dem du dich gerade befindest, an der Eingabeaufforderung ablesen. Es gibt aber auch einen Befehl, mit dem du dir das Arbeitsverzeichnis ausgeben lassen kannst: `pwd`, das steht für »print working directory«.

```
...:~$ pwd
/home/tux
```

Mit dem Befehl `ls`, für »list«, kannst du dir alle Dateien und Unterverzeichnisse des Arbeitsverzeichnisses anzeigen lassen.

Einige wichtige Flags von `ls` sind `-l`, `-a` und `-R`.

- `-l` sorgt dafür, dass ausführliche Informationen aller Inhalte ausgegeben werden,
- mit `-a` werden versteckte Dateien ausgegeben und
- mit `-R` werden rekursiv auch Unterverzeichnisse und deren Unterverzeichnisse mit ausgegeben.

```
...:~$ ls -l -a -R
...
...:~$ ls
...
```

2 Für Ubuntu ist dies z.B. `https://wiki.ubuntuusers.de/Dokumentation/`

Um den Inhalt eines bestimmten Verzeichnisses in einem vom Arbeitsverzeichnis abweichenden Verzeichnis anzuzeigen, kann als Parameter ein Pfad übergeben werden.

```
...:~$ ls /etc
```

Zum Navigieren im Verzeichnissystem wird der Befehl `cd`, für »change directory«, verwendet. Diesem muss als Parameter der Pfad übergeben werden, in den du wechseln willst. Zur Erinnerung: Pfade können absolut oder relativ angegeben werden.

Ein Beispiel mit absoluten Pfaden:

```
...:~$ cd /etc/ssh
...:/etc/ssh $ cd ~
...:~$ cd /
...:/$
```

Ein Beispiel mit relativen Pfaden:

```
...:/$ cd home
...:/home$ cd tux
...:/home/tux$
```

Um in übergeordnete Verzeichnisse, also aus einem Verzeichnis raus, zu navigieren, nutzt man zwei Punkte `..`. Mit `cd ..` springst du also in das übergeordnete Verzeichnis. Mit `../einOrdner` würdest du dich aus dem aktuellen Verzeichnis heraus- und in das benachbarte Verzeichnis `einOrdner` hineinbegeben.

```
...:~$ cd ..
...:/home$ cd ../etc
...:/etc$ cd ~
...:~$
```

Neue Verzeichnisse können mit dem Befehl `mkdir`, für »make directory«, angelegt werden.

`mkdir` nimmt beliebig viele Parameter entgegen, wobei jeder Parameter einen Pfad darstellt, der erstellt werden soll. Gibst du hier nur einen Namen an, also einen relativen Pfad, dann wird das Verzeichnis direkt im Arbeitsverzeichnis erstellt.

Der Name eines Verzeichnisses darf Groß- und Kleinbuchstaben, die Zahlen 0 bis 9 und einige Sonderzeichen wie `! % ( ) { } . - ^ ~ _ @ # $` und Leerzeichen enthalten.

```
...:~$ mkdir skripte skripte2
...:~$ ls
skripte  skripte2
...:~$ mkdir skripte/abschnitt1
...:~$ ls
skripte  skripte2
```

Es ist wichtig, dass alle Verzeichnisse im angegebenen Pfad, bis auf das letzte, bereits existieren. Möchtest du einen ganzen Pfad erstellen, dann musst du das Flag `-p` anhängen.

```
...:~$ mkdir -p verzeichnis/verzeichnis2
...:~$ ls
skripte  skripte2 verzeichnis
...:~$ ls verzeichnis
verzeichnis2
```

Dateien und Ordner können mit dem Befehl `rmdir`, für »remove directory«, gelöscht werden. Der Befehl funktioniert aber nur, wenn das Verzeichnis leer ist. Ist es das nicht, kann der Befehl `rm -r` genutzt werden, um ein Verzeichnis samt Inhalt zu löschen. Dieser Befehl wird im nächsten Abschnitt (Abschnitt 1.3.5) zu Dateien noch mal aufgegriffen und genauer erklärt. `rmdir` und `rm -r` werden einfach der Pfad des zu löschenden Verzeichnisses oder die Pfade der zu löschenden Verzeichnisse übergeben.

```
...:~$ ls
skripte  skripte2 verzeichnis
...:~$ rmdir skripte skripte2
...:~$ ls
verzeichnis
```

Mit dem Befehl `mv`, für »move«, können Verzeichnisse verschoben oder umbenannt werden. Dazu wird als erster Parameter der Pfad des zu verschiebenden Verzeichnisses und als zweiter der des Zielverzeichnisses angegeben. Es werden alle Inhalte vom ersten in das zweite Verzeichnis verschoben. Da das alte Verzeichnis gelöscht und ein neues erstellt wird, kann so durch Angabe eines neuen

Namens, ohne Änderungen am Pfad, eine Datei oder ein Verzeichnis auch umbenannt werden.

```
...:~$ mkdir skripte
...:~$ ls
skripte verzeichnis
...:~$ mv skripte skripte2
...:~$ ls
skripte2 verzeichnis
```

1.3.5 Arbeiten mit Dateien

Eine neue, leere Datei kannst du mit dem Befehl `touch` anlegen. Diesem Befehl übergibst du einfach den Pfad der zu erstellenden Datei. Es gilt wieder, dass alle Verzeichnisse im Pfad bereits existieren müssen. Für die Namen von Dateien gelten dieselben Regeln wie für Verzeichnisnamen.

```
...:~$ touch textDatei
...:~$ ls
textDatei
```

Eine neu erstellte Datei ist, wenig überraschend, leer. Zum Beschreiben kannst du einen Texteditor deiner Wahl nutzen. Hast du keinen Editor installiert oder kannst nur im Terminal arbeiten, dann ist das natürlich auch möglich.

Ein sehr leicht zu bedienender, häufig bereits vorinstallierter Editor ist `nano`. Da `nano` auch auf Ubuntu vorinstalliert ist, ist in der Regel keine manuelle Installation notwendig und du kannst direkt loslegen.

Wir öffnen `nano`, indem wir `nano` als Befehl nutzen und einen Dateipfad als Parameter übergeben.

```
...:~$ nano textDatei
```

Kurz ein paar Worte zu `nano`. Es ist ohne weitere Einstellungen im Terminal nicht möglich, mithilfe der Maus zu navigieren, es bleibt nur das Navigieren mittels Pfeiltasten. Am unteren Rand des Terminals sind weitere Tasten und ihre Bedeutung zu sehen, das »^« steht dabei für die Strg-Taste. Du kannst Programme wie `nano` aber auch in einem Modus mit Mausunterstützung starten. Bei `nano` geht das über das Flag `-m`.