Louis Schirmer | Uwe M. Schirmer

Shell Script Programmierung

> kapieren & trainieren <

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

mitp

Inhaltsverzeichnis

Einleitung 11

1 Weißgurt: Einführung in Terminal und Shell 15
1.1 Die richtige Linux Distribution 15
1.2 Die Verzeichnisstruktur 16
1.2.1 Das Home-Verzeichnis 17
1.3 Terminal und Shell 18
1.3.1 Das Terminal bedienen 19
1.3.2 Befehle ausführen 20
1.3.3 Hilfe zur Selbsthilfe 22
1.3.4 Navigieren im Dateisystem 24
1.3.5 Arbeiten mit Dateien 27
1.3.6 Zugriffsrechte 29
1.3.7 Die Umgebungsvariablen 34
1.3.8 Programme nachinstallieren (Paketmanagement-Systeme) . 35
1.3.9 Übersicht über wichtige Befehle 36
1.4 Die erste Gürtelprüfung 36
1.4.1 Aufgabenstellung 37
1.4.2 Musterlösung 37

2 Gelbgurt: Grundlagen des Shell Scriptings 39
2.1 Das erste Skript 39
2.1.1 Der Shebang 40
2.1.2 Maskierung 42
2.1.3 Einfache Textausgaben 43
2.1.4 Mehr zum Ausführen von Befehlen 46
2.1.5 Kommentare 48
2.1.6 Ausführen von Skripten 48
2.1.7 Übung: Eine Textausgabe mit Escape-Sequenzen 49
2.2 Variablen 50
2.2.1 Bezeichner 50
2.2.2 Erstellen und löschen 51
2.2.3 Datentypen 53
2.2.4 Variablen-, Parameter- und Kommandosubstitution 54
2.2.5 Übung: Variable ausgeben 57
2.2.6 Aufbauendes Wissen zu Umgebungsvariablen 57
2.3 Fortgeschrittene Ausgaben 59
2.3.1 printf 59

2.3.2 Mehr zu ANSI-Escape-Sequenzen 62
2.3.3 Übung: Ladebalken 64
2.4 Eingaben 65
2.4.1 Übung: Begrüßung 67
2.5 Rechnen 67
2.5.1 Rechnen mit let 69
2.5.2 Der expr-Befehl 70
2.6 Bedingte Anweisungen 70
2.6.1 Der Befehl test 70
2.6.2 Programmlauf kontrollieren mit if und else 74
2.6.3 Übung: Benutzerdialog 77
2.6.4 Mehrfachverzweigung über switch und case 78
2.6.5 Übung: Benutzerdialog mit Mehrfachverzweigung 80
2.6.6 Bedingungen und Rechnen 80
2.7 Gürtelprüfung 81
2.7.1 Aufgabenstellung 82
2.7.2 Musterlösung 83

3 Orangegurt: Erweiterte Skriptfunktionen 85
3.1 Grundlagen zu Kommandozeilenparametern 85
3.2 Schleifen 86
3.2.1 for-Schleife für Zahlenwerte 87
3.2.2 for-Schleife für Aufzählungen 88
3.2.3 for-Schleife mit Dateinamensubstitution 90
3.2.4 Übung: Dateien und Verzeichnisse analysieren 91
3.2.5 while-Schleife 91
3.2.6 until-Schleife 93
3.2.7 Übung: Dem Skript Parameter übergeben 93
3.3 Kommandozeilenparameter (Fortsetzung) 94
3.3.1 shift-Befehl 97
3.3.2 Übung: Eine Summe berechnen 97
3.3.3 Kommandozeilen-Flags mit getopts auswerten 98
3.3.4 Übung: Skript mit Flags 100
3.4 Arrays 100
3.4.1 Der POSIX-konforme Weg 101
3.4.2 Arrays in der Bash 102
3.4.3 Übung: Arrays und Schleife 105
3.5 Exit 106
3.5.1 Das Skript beenden 107
3.5.2 Exit-Status und bedingte Anweisungen 108
3.5.3 Übung: Das Programm mit Fehlercode abbrechen 109
3.6 Fehlersuche 110
3.6.1 xtrace-Modus 112

3.6.2 noexec-Modus . . . 112
3.6.3 Leere Variablen erkennen . . . 113
3.6.4 Skript bei einem Exit-Status abbrechen . . . 114
3.7 Gürtelprüfung . . . 114
3.7.1 Aufgabenstellung . . . 115
3.7.2 Musterlösung . . . 115

4 Grüngurt: Umgang mit Ein- und Ausgaben . . . 117
4.1 Umleitungen von Ein- und Ausgaben . . . 118
4.1.1 Ein- und Ausgabekanäle . . . 118
4.1.2 Dateideskriptoren . . . 121
4.1.3 Dateien zeilenweise einlesen . . . 124
4.1.4 Übung: Wörter zählen . . . 127
4.2 Ein- und Ausgabeweiterleitung mit Pipes . . . 127
4.2.1 Named Pipes . . . 130
4.2.2 Ausgabe vervielfältigen mit »tee« . . . 131
4.2.3 Dezimalzahlen mit »bc« und Pipes . . . 132
4.2.4 Übung: Flüssigkeiten und Dezimalzahlen . . . 134
4.3 Arbeiten mit Dateien . . . 134
4.3.1 Dateien zerlegen und zusammenfügen . . . 135
4.3.2 Weitere Optionen für Textdateien . . . 137
4.3.3 Dateien analysieren . . . 138
4.3.4 Texte sortieren . . . 141
4.3.5 Übung: Inhalte eines Verzeichnis analysieren . . . 142
4.3.6 Inhalte in Dateien suchen . . . 144
4.3.7 Übung: Suchen eines Texts in einer Datei . . . 147
4.3.8 Duplikate finden . . . 148
4.3.9 Übung: Sortieren einer Dateiauswertung . . . 150
4.3.10 Inhalte von Dateien verändern . . . 150
4.3.11 Einfügen, Anfügen und Löschen von Zeilen . . . 152
4.3.12 Übung: Einfacher Texteditor . . . 157
4.4 Gürtelprüfung . . . 158
4.4.1 Aufgabenstellung . . . 159
4.4.2 Musterlösung . . . 160

5 Blaugurt: Fortgeschrittene Textverarbeitung . . . 163
5.1 Reguläre Ausdrücke . . . 163
5.1.1 Reguläre Ausdrücke am Beispiel von grep . . . 164
5.1.2 Übung: Namensvarianten finden . . . 172
5.1.3 Übung: Erkennen von E-Mail-Adressen . . . 172
5.1.4 Übung: Datumsangaben erkennen . . . 173
5.2 Arbeiten mit Strings . . . 173
5.2.1 Strings zusammenfügen . . . 174

5.2.2 Pattern Matching, Globbing und Platzhalter 175
5.2.3 Strings mit Parametersubstitution manipulieren 177
5.2.4 Bedingte Anweisungen mit Strings 182
5.2.5 Übung: Pattern Matching anwenden 184
5.3 Dateien miteinander vergleichen und kombinieren 185
5.3.1 Dateien patchen 187
5.3.2 Inhalte mit »join« vereinigen 188
5.3.3 Spalten ausschneiden 189
5.3.4 Spalten zeilenweise ausgeben 190
5.3.5 Übung: Benutzer und Home-Verzeichnisse 192
5.4 Skripte formatieren 193
5.4.1 Styleguides 193
5.4.2 Codequalität verbessern mit einem Linter 193
5.5 Ausführungszeit eines Skripts oder Befehls messen 195
5.6 Systeminformationen und Logdateien 196
5.6.1 Filtern über Befehle 197
5.7 Gürtelprüfung 198
5.7.1 Aufgabenstellung 198
5.7.2 Musterlösung 199

6 Braungurt: Prozesse und Signale 203
6.1 Prozesse 203
6.1.1 Prozessattribute 204
6.1.2 Prozesse verwalten 206
6.1.3 Übung: MyPs 212
6.1.4 Job-Kontrolle 213
6.1.5 Übung: Asynchrone Prozesse 217
6.1.6 Prioritäten anpassen 218
6.1.7 Daemons 219
6.2 Funktionen 221
6.2.1 Übung: Verarbeitung abbrechen 222
6.2.2 Gültigkeit von Funktionen 222
6.2.3 Werte an Funktionen übergeben 224
6.2.4 Geltungsbereiche von Variablen 225
6.2.5 Werte in Funktionen zurückgeben 226
6.2.6 Übung: Funktion mit Parametern 227
6.2.7 Übung: Funktionsbibliothek 227
6.3 Signale 228
6.3.1 Signale senden 230
6.3.2 Signale abfangen 231
6.3.3 Signale ignorieren 233
6.3.4 Signale zurücksetzen 233
6.3.5 Übung: Benutzerfreundliches Abbrechen 233

6.4 Einsatz von Signalen in Skripten 234
6.4.1 Abbrechen von Skripten verhindern 235
6.4.2 Geordnetes Abbrechen von Skripten 235
6.4.3 Geordnetes Beenden eines Skripts oder der Shell 236
6.4.4 Konfigurationsdateien neu einlesen 236
6.4.5 Übung: Konfiguration einlesen 237
6.5 Erweiterte Eingaben 238
6.5.1 Überprüfen von Eingaben 239
6.5.2 Übung: Skript mit Menü 240
6.6 Gürtelprüfung 241
6.6.1 Aufgabenstellung 241
6.6.2 Musterlösung 241

7 Schwarzgurt: Systemkonfiguration 243
7.1 Systeminitialisierung 243
7.1.1 Systemstart 243
7.1.2 Init 244
7.1.3 Das Init-System »systemd« 245
7.1.4 Lesen der Konfigurationen 246
7.1.5 Units und Targets 246
7.1.6 Logging während des Systemstarts 248
7.1.7 Nützliche systemd-Hilfsprogramme 249
7.1.8 Systemweite Umgebungsvariablen 251
7.1.9 Eigene Units erstellen 252
7.1.10 Übung: Eine eigene Unit erstellen 255
7.2 Start von Shells 255
7.2.1 Login-Prozess 256
7.2.2 Nicht-Login-Shells 258
7.2.3 Das System konfigurieren 258
7.2.4 Übung: Bash-Aliasse anlegen 261
7.2.5 Übung: Umgebungsvariablen anlegen 262
7.3 Zeitgesteuertes Ausführen von Skripten 262
7.3.1 Crontab 262
7.3.2 Übung: Einrichten eines Cronjobs 269
7.3.3 Anacron 270
7.3.4 At 272
7.4 Zeichencodierung 275
7.5 Skripte für das System sichtbar machen 277
7.6 Gürtelprüfung 278
7.6.1 Aufgabenstellung 278
7.6.2 Musterlösung 278

Stichwortverzeichnis 281

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.

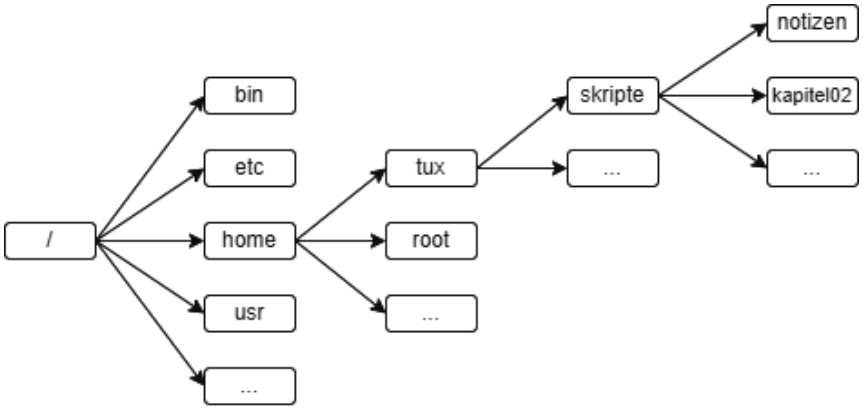

Abb. 1.1: Ausschnitt einer Verzeichnisstruktur

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`.

Abb. 1.2: So sieht nano nach dem Start aus.

Drücken wir z.B. die Tasten Strg+X, um die Datei zu schließen, werden wir noch gefragt, ob wir unsere Änderungen speichern wollen, was wir mit einem Y und dem Bestätigen des Dateinamens per Enter dann auch tun. Natürlich kannst du auch zwischenspeichern, das geschieht mit den Tasten Strg+S.

Es ist aber lästig eine Datei immer im Editor zu öffnen, um den Inhalt zu sehen. Zur Ausgabe des Inhalts einer Datei in der Konsole kann der Befehl `cat`, für »concatenate«, genutzt werden. Der eigentliche Zweck von `cat` ist das Zusammenfügen mehrerer Dateien, aber der Befehl eignet sich auch gut für eine Ausgabe des Inhalts. `cat` nimmt als Parameter einen Dateipfad und gibt den Inhalt der Datei in der Konsole aus. Mit dem Flag `-n` werden die Zeilen nummeriert ausgegeben.

```
...:~$ cat -n textDatei
...
```

Vorsicht

Öffnest du aus Versehen eine Binärdatei mit `cat`, kann es sein, dass du den Befehl `reset` nutzen musst, um dein Terminal wieder in einen benutzbaren Zustand zu versetzen.

Mit dem `cp`-Befehl (»copy«) kann eine Kopie der Datei erzeugt werden. Dazu wird als erster Parameter der Dateipfad der zu kopierenden Datei übergeben und als zweiter Parameter der Dateipfad der zu erstellenden Datei. Existiert die Zieldatei bereits, wird lediglich der Inhalt überschrieben.

```
...:~$ cp textDatei textDatei2
```

Der Befehl `mv` (»move«) zum Verschieben funktioniert für Dateien genauso wie für Verzeichnisse. Es werden zwei Dateipfade als Parameter übergeben, und die Inhalte der ersten Datei werden in die neu erstellte zweite Datei geschrieben. Da die alte Datei gelöscht wird, kann der Befehl auch genutzt werden, um Dateien umzubenennen.

```
...:~$ mv textDatei2 textDatei3
```

Wird als zweiter Parameter lediglich ein bereits bestehendes Verzeichnis angegeben, heißt die Datei im neuen Verzeichnis identisch zur Ursprungsdatei. Ist die Zieldatei bereits vorhanden, wird der Inhalt der Datei wie bei `cp` überschrieben.

```
...:~$ touch textDatei
...:~$ mkdir testVerzeichnis
...:~$ mv textDatei testVerzeichnis/textDatei3
...:~$ cd testVerzeichnis
...:~/testVerzeichnis$ ls
textDatei3
...:~/testVerzeichnis$ mv textDatei3 textDatei4
...:~/testVerzeichnis$ ls
textDatei4
```

Um Dateien zu löschen, nutzt man den `rm`-Befehl, den du im letzten Abschnitt zum Verzeichnissystem schon kennengelernt hast. Dieser Befehl nimmt als Parameter den Namen der Datei entgegen, welche gelöscht werden soll.

Mit dem Flag `-d` kann mit `rm` auch ein Verzeichnis angegeben werden, welches gelöscht werden soll. Dieses muss genau wie bei `rmdir` aber leer sein. Um auch nicht leere Verzeichnisse zu löschen, wird das Flag `-r` genutzt. So kannst du wie im letzten Abschnitt angegeben auch Verzeichnisse löschen, die noch Dateien oder Ordner enthalten:

```
...:~/testVerzeichnis$ cd ..
...:~$ rm -r testVerzeichnis
```

1.3.6 Zugriffsrechte

Linux ist ein Mehrbenutzersystem, das bedeutet, dass es auf den Einsatz mit mehreren Benutzern ausgelegt ist. Hier sind Zugriffsrechte, die angeben, was man sehen und was man ausführen darf, besonders wichtig.

Ein normaler Benutzer hat vollen Zugriff auf sein eigenes Home-Verzeichnis, ansonsten kann er aber nur auf wenige andere Ordner zugreifen. Der Administrator (root) hat dagegen volle Rechte im gesamten Dateisystem.

Auch wenn man das Linux-System selbst eingerichtet hat, ist man nicht automatisch Administrator. Bei der Installation wird man in der Regel dazu aufgefordert, einen Benutzer-Account anzulegen. Es ist nämlich nicht sinnvoll, normale Arbeiten als Administrator durchzuführen. Wenn du z.B. auf einer bösartigen Webseite surfst oder ein bösartiges Programm ausführst, würdest du das eigene System sofort gefährden, wenn du immer den vollen Zugriff hättest. Deswegen wirst du normalerweise als Benutzer arbeiten und dich nur als Administrator anmelden, wenn du Dinge am System konfigurieren oder verändern möchtest.

Es gibt unter Linux auch die Möglichkeit, sich kurzzeitig zum Administrator zu machen, um mal eben einen Befehl auszuführen, den man sonst nicht ausführen könnte. Das geht mit dem `sudo`-Befehl (»superuser do«). Wenn du `sudo` einem Befehl voranstellst, wird er mit Administrator-Rechten ausgeführt. Dazu muss das System aber entsprechend eingerichtet sein. Je nachdem, wie `sudo` konfiguriert wurde, wird man dann noch aufgefordert, das Administrator-Passwort einzugeben, aber nicht bei jedem weiteren `sudo`-Aufruf[3].

```
...:~$ sudo ls
[sudo] password for tux:
...
```

Was der Benutzer mit einer Datei oder einem Verzeichnis anstellen darf, wird über den Eigentümer und die Zugriffsrechte entschieden.

Jede Datei hat einen Eigentümer und eine Gruppe, der sie zugeordnet ist. Wenn du eine Datei mit `touch` erstellst, bekommt sie automatisch dich als Benutzer und Gruppe zugewiesen.

```
...:~$ touch neueDatei
...:~$ ls -l neueDatei
-rw-r--r-- 1 tux tux 0 Mar  7 17:59 neueDatei
```

Der Befehl `ls` mit dem Flag `-l` zeigt ein paar zusätzliche Informationen an, unter anderem die Zugriffsrechte (`-rw-r--r--`, erklären wir gleich), den Eigentümer (erstes `tux`) und die Gruppe (zweites `tux`). Danach kommen noch die Größe der Datei, der Erstellungszeitpunkt und natürlich der Name.

3 Wie lange sich `sudo` einen erfolgreichen Login merkt, kann man konfigurieren, die Standardeinstellung liegt bei 5 Minuten.

Stichwortverzeichnis

-- 87, 102
? 176
* 176
&& 47, 75
# 48, 94
#! 40
% 59
++ 87
|| 47, 75

A

Abbruchbedingung 87
Administrator 19
  anmelden 30
  Zugriffsrechte 30
alarm 228
Alias 258
  anlegen 261
  löschen 260
Anacron 268, 270
  Jobs definieren 271
  Umgebungsvariablen 270
  Zeitstempel 270
Anker 171
ANSI 45
  Farbcodes 46
ANSI-Escape-Sequenz 62
ANSI-Steuerzeichen 64
APT 35
Arbeitsverzeichnis 20
  ausgeben 24
Archiv
  durchsuchen 144
Argument
  Parameter 21
arithmetischer Kontext 67
Array 100, 102
  nachträglich Elemente hinzufügen 104
  Strings konvertieren 103
ASCII 275
Asterisk 265
asynchron 228
At 272
  Datumsangabe 272
  geplante Jobs ausgeben 275
  Jobnummer 274
  Jobs löschen 275
  Queue 274
  Schlüsselwörter 273
  Zeitangabe 272
aufteilen 135
Aufzählung 87
Ausführungszeit
  messen 195
Ausgabe 118
Auslagerungsspeicher 209
Authentifizierungsversuche 196
Automatisierung 262

B

Backtick 57
Bash 19
bc 132
Bedingte Anweisungen 70
Bedingung 74, 86
  mehrere 75
  verschachtelte 77
Befehl 18, 20
  abbrechen 22
  Hilfetext 22
  mehrere ausführen 46
  Übersicht 36
Benutzer 192
  ändern 33
  ausgeben 192
  ID 204
  Login 256
Benutzerdialog 77
Benutzereingabe 65
Benutzername 17
Benutzersignal 229
Berechtigungen 17
Bezeichner 50
bg 215
Bibliothek 227
bin 18
Binärdatei 135
  vergleichen 187
Binärer Operator 73
BIOS 243
Blackscreen 244
Bootloader 243
Bourne-Again Shell
  bash 19
Bourne-Shell 19, 67
Buffer 208
Byte-Offset 145

C

Cache 209
case 78
cat 117, 136
checkbashisms 194
CMD 205
cmp 187
Code
  auslagern 198
Cron 262
Cronjob 262
  Ausgabe in Datei speichern 268
  Ausgaben umleiten 268
  einrichten 269
  Fehlermeldungen 268
  Intervall 266
  Schrittgröße von Intervallen 266
  Sonderzeichen für Zeitangaben 267
  Wochentage 266
  Zeitangaben 266
  Zeitangaben Kurzform 267
Crontab 262
  Flags 263
Cursor
  neu setzen 63
  Position speichern 63
  verschieben 63
cut 189

D

Daemon 219, 244
  anzeigen 206
  inetd 219
  systemd 219
daemon-reload 250
Dash 19
Datei 134
  analysieren 138
  anzeigen 24
  durchsuchen 144
  Eigenschaften prüfen 73
  Ende 126
  erstellen 27
  gepackte 139
  Inhalt ändern 150
  Inhalt anzeigen 28
  kopieren 28
  lesen aus 118
  löschen 26
  mehrere Änderungen 156
  mehrere durchsuchen 146
  patchen 187
  schreiben in 118
  Typ 138
  umbenennen 27, 29
  vergleichen 185
  verknüpfen 188
  verschieben 29
  Zeile anfügen 152
  Zeile einfügen 155
  Zeile löschen 153
  zerlegen 135
  zusammenfügen 135
Dateideskriptor 121
  erstellen 121
  freigeben 123
Dateien
  löschen 29
Dateiende
  Platzhalter 154
Dateinamensubstitution 90
Dateisystem 16
Dateityp 31
  überprüfen 70
Datenstruktur 100
Datentyp 53
  beschränken 54
dd 238
Debian-Almiquist-Shell
  Dash 19
declare 52, 54, 103
Default-Wert 79
Dezimalzahl 54
  rechnen 132
Dienst
  Daemon 219
  starten 247
diff 185, 187
Display Manager 256, 257
Distribution 15
Dokumentation 23
Dollar-Zeichen 55
Doppelpunkt 56
Duplikate
  entfernen 149
  finden 148

E

echo 43, 118
Editor 27
egrep 144
Eigentümer 30
eilf 76
Eingabe 65, 238
  Trennzeichen 65
  überprüfen 239
  weiterleiten 120
Eingabeaufforderung
  Prompt 24
Einstellung 134
else 76
Elternprozess 34, 203
  anzeigen 207
E-Mail 135, 173
Endlosschleife 126
env 41
EOF 121
EOT 274

esac 79
escape 93
Escape-Sequenz 43, 45
  ANSI 44
Escaping 42
exec 121
Exit-Status 106, 114, 125
expr 70
Extended-RE 164

F

Fehler
  finden 194, 196
Fehlercode 106, 109
Fehlermeldung 118
Fehlersuche 110
Feld 190
fg 215
fgrep 144
FHS 16
Fibonacci 93
FIFO 128, 130
file 138
First-in-First-out 128
Flag 21
  auswerten 98
  Kurzform 21
  zusammenfassen 21
Fließkommazahl
  Dezimalzahl 54
Float 54
fmt 138, 141
for 87, 88
Formatangabe 59
Formatierung 193
  aufheben 44
Funktion 221, 241
  entfernen 223
  exportieren 222
  Gültigkeit 222
  lokale Variable 225
  mehrere Rückgabewerte 227
  Rückgabewert 226

G

gdm3 257
Gerätesignal 229
getopts 98
  Fehlermeldung anpassen 99
getty 256
GID 204
GitLab 193
Globbing 175
GNU-Projekt 23
Gravis
  Backtick 57
grep 144, 198, 239
  reguläre Ausdrücke 164
  Rückgabewert 146
größer-gleich 72
Gruppe 30
  ändern 33
Gürtelprüfung 36

H

Hallo Welt 221
Handler 231
  Signal-Handler 234
head 137
here document 121
here string 121
Hilfetext 22
HOME 34
Home-Verzeichnis 17
  ausgeben 192
htop 206, 207
  Funktionstasten 210
  Sortierung 209
httpd 219

I

Identifier 50
if 74
IFS 101
Index 103
  ausgeben 109
inetd 219
info 23
INI-Datei 252
  Abschnitte 252
  Aufbau 253
Init 244, 252
  alte Systeme 245
Initialisierungsdatei
  INI-Datei 252
Instanz 203
Integer 54
  vergleichen 71
Interpreter 39
  angeben 40
  Flag 40

J

Job 213
  Hintergrund 213
  im Hintergrund starten 215
  stoppen 214
  Vordergrund 214
  warten auf 216
jobs 214
join 188
journalctl 248
  Flags 248

K

Kanal 118, 124
  auf Kanal umleiten 119
  nicht anzeigen 119

Standardausgabekanal 118
Standardeingabekanal 118
Standardfehlerkanal 118
umleiten 118
Kernel 15, 243
kill 230
Kleene, Stephan 164
kleiner-gleich 72
Kommandosubstitution 56
Kommandozeilenparameter 85, 94, 198
Anzahl 94
Kommentar 48
Kommunikation 228
Konfigurationsdatei 34, 237, 243
Abschnitt 252
Benennung 253
benutzerspezifische 257
neu einlesen 236
neu laden 250
systemd 246
Konstante 52
löschen 52
Kontrollstrukturen 70

L

Ladebalken 64
LANG 251
launchd 245
Lesezeiger 124
less 197
let 69
Link
symbolischer 40
Linter 193
Linux 15, 23
Liste 100
Load average 209
local 225
Logdatei 196
durchsuchen 197
Logging 245
Systemstart 248
Logikfehler 110
Login 256
mit grafischer Oberfläche 257
Login-Shell 256, 258
Log-Level 249
ls 175

M

macOS 245
Mail Transfer Agent 264
Manpage 22
maskieren 93
Maskierung 42
Master Boot Record 243
Medien-Typ 139
Mehrfachverzweigung 78
Mem 208
Mimetype
Medien-Typ 139
mkfifo 130
mknod 130
Modulo-Operator 68

N

Nachkommastellen 61
Named Pipe 130
erstellen 130
löschen 131
nano 27
Navigieren 25
Negations-Operator 93
Netzwerk-Dienste 219
nice 205, 218
noexec-Modus 112

O

Oder-Operator 73
Oktalmodus 32
Operator 69, 87
binärer 73
Kurzform 69
OPTARG 98
OPTIND 98
Ordner
Verzeichnis 26

P

Paketmanager 35
Paketverwaltungsdatei
durchsuchen 144
Parameter 21, 97
als String speichern 95
mehr als 9 97
Parametersubstitution 55, 86
Strings 177
paste 190
patch 187
Patchdatei 187
PATH 34, 277
Pattern Matching 175
Performance 196
Pfad
absoluter 21
erstellen 26
relativer 21
PGID 230
pgrep 144
PID 204, 210
Pipe 47, 127, 128
Befehle verbinden 128
mehrere Ausgaben 131
Named Pipe 130
Syntax 128
Unnamed Pipe 128
Pipe-File 130
Platzhalter 80, 90, 175, 177

Dateianfang 154
Dateiende 154
reguläre Ausdrücke 165
Zeilenanfang 154
Zeilenende 154
Port 219
Positionsparameter 85
setzen 101
POSIX 18
prüfen 194
POSIX-RE 164
postfix 264
Potenz 68
Potenz-Operator 133
PPID 204
PRCE 164
printf 59
Breite 60
Flags 61, 62
Platzhalter 60, 62
Präzision 60
Priorität 218
proc 210
Dateien 211
Verzeichnisse 211
Programm 20
installieren 35
Prompt 20
Prozent-Zeichen 59
Prozess 20, 203
asynchron 217
Attribute 204, 205
beenden 229
besondere IDs 230
Gruppe 230
ID 204, 230
Init 244
Priorität 208, 218
Systemstart 244
verwaister 244
verwalten 206
Zombie 244
Zustand 204, 205
Prozesskontrollblock 203
Prozessorauslastung 208
Prozess-Substitution 192
Prozesstabelle 203, 244
ps 206
Pseudo-Dateisystem 210
pstree 206
PWD 34

Q

Quantifizierer 168
Quoting 42
schwaches 43
starkes 42

R

raw-Mode 238
read 65, 118, 120, 124, 126
Flag 66
readonly 52
Rechenoperationen 68
Rechnen 67
Bedingungen 80
Dezimalzahlen 132
RegEx
Reguläre Ausdrücke 163
Reguläre Ausdrücke 163
Alternativen 170
Backslash 165
erweiterte 165
Fragezeichen 165
Gruppierung 170
Pipe 170
Platzhalter 165
Punkt 165
Quantifizierer 168
Sonderzeichen 171
speichern 170
Zeichenauswahl 166
Zeichenklassen 165
renice 218
Reservierte Wörter 42
return 226
RFC 172
Root-Verzeichnis 16
Rückgabe
in Variable speichern 56
Runlevel 248
Run Queue 209

S

sbin 18
scale 133
Scheduler 203, 205
Schleife 86
Abbruchbedingung 87
for 87, 88
Liste 88
Platzhalter 90
Trennzeichen 88
until 93
Vergleich 87
while 91
Schlüsselwörter 42
sed 153
Kommandos 153
Semikolon 46
sendmail 264
seq 191
Service 219
Service-Typ 253
set 101, 111
sh 19
Shebang 40

Shell 18
Flag 111
geordnet beenden 236
Konfigurationsdateien 255
konfigurieren 258
Modus 256
ShellCheck 193
Shell-Variable 50
shift 97
SIGALRM 228
SIGHUP 228
SIGINT 228
deaktivieren 235
SIGKILL 233
Signal 204, 228
0 230
abfangen 231
auflisten 231
Hauptgruppen 229
ignorieren 233
in Skripten behandeln 234
senden 230
Übersicht 229
zurücksetzen 233
Signal-Handler 234
SIGSTOP 233
Skript 39
abbrechen 109
abbrechen verhindern 235
ausführen 48
Ausführungszeit messen 195
beenden 107
Ende 40
Endung 39
geordnet beenden 236
leere Zeilen 40
Name 39
sichtbar machen 277
zeitgesteuert ausführen 262
sleep 64
Sonderzeichen 42, 53
sort 141
Sortieren 142, 150
source 222
Spalte
ausschneiden 189
zeilenweise ausgeben 190
Speicherauslastung 208
Speichergröße 135
Speichern 241
split 135
Spracheinstellungen 276
sshd 219
SSH 219
Standardausgabe 127
Standardeingabe 126, 127
String 53, 173
Groß- und Kleinschreibung konvertieren 181
Länge überprüfen 71
lexikografische Anordnung 182
Vergleich 182
zusammenfügen 174
strings 140
stty 228
Styleguide 193
Subshell 256, 257, 258
Substitution 90
Substring 178
entfernen 179
ersetzen 180
Suchen 163, 200
im Verzeichnis 147
in Datei 144
sudo 30, 250
Summe 97
Superdaemon 219
Swp 209
Symlink 147
synchron 228
Syntaxfehler 110
finden 112
syslog 248
System
aktualisieren 35
herunterfahren 250
neu starten 250
systemctl 249
systemd 219, 245
Unit-Konfigurationsdatei 252
Systemdienste 219
systemd-inhibit 250
systemd-Journal 248
Konfiguration 249
Prioritäten Logeinträge 249
Systeminformationen 196
Systemprozess
anzeigen 206
Systemsignal 229
Systemstart 243, 245
Systemzustand 247
SysVinit 244

T

tac 137
tail 137, 197
Target 245, 247
symbolischer Link 248
Task
Prozess 203
Tasks 209
Tastendruck abfragen 238
Tastenkombination 228
tee 131
Temporäre Dateien 235
Terminal 18
Copy and Paste 20
öffnen 19

schließen 20
Tastenkombination 228
test 182
Exit-Status 108
Testdatei
erzeugen 136
Testumgebung 184
Text
extrahieren 140
sortieren 141
Textausgabe 43, 59
Textdatei 117
analysieren 140
aufteilen 136, 137
formatieren 138
rückwärts zusammenfügen 137
Texteditor 157
Thread 204
Tilde 18
time 195
Timer 255
Token 21
top 206, 207
tr 150
trap 231
Trennzeichen 101
Tux 17
type 223

U

Ubuntu 16
UEFI 243
UID 204
Umgebungsvariable 34, 57, 106
? 73
anlegen 262
anzeigen 34, 250, 251
dauerhaft ändern 260
löschen 34
Spracheinstellungen 276
systemweit 251
systemweite setzen 251
temporär erstellen 58
Umleitung 118
Umleitungsarten 124
Und-Operator 73
uniq 148
Unit 245, 246
Abhängigkeiten 247
automatisch starten 254
beenden 249
Dateisystemmanagement 247
einmalig starten 254
erstellen 255
Konfiguration überprüfen 254
konfigurieren 252, 253
starten 249
Target 254
Unix 16
Unnamed Pipe 128
unset 52, 223
until 93
Uptime 209
USER 34
UTF-8 53, 139, 276

V

Variable 50, 225
Bezeichner 50
definieren 51
erstellen 51
exportieren 225
Geltungsbereich 225
initiieren 51
leere erkennen 113
löschen 52
Standardwert 55
weiterleiten an Standardeingabe 121
Wertzuweisung 51
Variablensubstitution 54
Vergleich 70, 71
Vergleichsoperator 71, 74
Verzeichnis
anzeigen 24
durchsuchen 147
erstellen 25
löschen 26, 29
umbenennen 27
verschieben 26
wechseln 25

W

Wahrheitswert
umkehren 72
wait 216
wc 140
while 91
Whitespace 166
Wildcard 79, 90, 175, 176
Wurzel-Verzeichnis siehe Root-Verzeichnis 16

X

xtrace-Modus 112

Z

Zahl 54
vergleichen 71
Zeichencodierung 275
Umgebungsvariablen 276
Zeichenkette 53, 173
Zeichensatz 53
Zeitgesteuertes Ausführen 262
Zombieprozess 244
Zugriffsrechte 29, 31
ändern 31
Oktalmodus 32