

LESEPROBE

Linux CLI

Die Macht der universellen Linux-Befehle

Markus Edenhauser, MA MSc

Copyright ©
Dezember 2025
1. Auflage
Selbstverlag

Markus Edenhauser, MA MSc
Völser Straße 41
6020 Innsbruck
Österreich
pixeledi.eu/impressum



Du bist mein sudo fürs Leben - mit dir läuft alles mit den richtigen Rechten.



Inhaltsverzeichnis

0	Einleitung	6
0.1	Warum die Befehlszeile wichtig ist	6
0.2	Dein Einstieg in die Praxis	6
0.3	Universelle Linux-Befehle	6
0.4	Übungsdateien und E-Book herunterladen	7
0.5	Übungsdateien ausführen	7
0.6	Üben statt Auswendiglernen	8
0.7	Schreibweise von Shortcuts	8
0.8	Warum ich Denglisch verwende	8
0.9	Warum CLI trotz KI wichtig bleibt	8
0.10	KI-Transparenz-Hinweis	9
0.11	MIT-Lizenz und Haftungsausschluss	10
1	Die Welt der Betriebssysteme	11
1.1	Die großen Betriebssysteme	11
1.2	Die Linux-Familie	11
1.3	Die Unix-Familie	12
1.4	Unix und seine Philosophie	12
1.5	True Unix und Unix-Like Systeme	12
1.6	Auf welchen Systemen funktionieren die CLI-Befehle?	13
1.7	Fokus auf universelle CLI-Befehle	13
1.8	Fazit	14
2	Linux unter Windows nutzen	15
2.1	WSL - Linux direkt unter Windows	15
2.2	Linux in VirtualBox	16
2.3	Erster Test mit deinem Linux	17
3	Das Terminal	19
3.1	Das Terminal öffnen	19
3.2	Shell und Terminal	19
3.3	Erste einfache Befehle	22
3.4	Navigation mit den Pfeiltasten	23
3.5	Kommandozeilenargumente	23
3.6	Optionen an Befehle übergeben	25
3.7	Unterschied zwischen Argumenten und Optionen	25
3.8	Beispiele mit cal	25
3.9	Langformen von Optionen	26
4	Hilfe und Informationen in der Linux-CLI	28
4.1	Die wichtigsten Werkzeuge	28
4.2	Der Befehl man	28
4.3	Navigation in Man-Pages	28
4.4	Die Befehle type und which	29

5	Orientierung in der Linux-Befehlszeile	31
5.1	Wichtige Befehle	31
5.2	Root-Verzeichnis	31
5.3	Das Home-Verzeichnis	32
5.4	Typische Ordner im Home-Verzeichnis	33
5.5	Die Kurzschreibweise ~	33
5.6	pwd	33
5.7	ls	35
5.8	ls Optionen	35
5.9	Mit cd Verzeichnisse wechseln	37
5.10	Einen Ordner nach oben wechseln	38
5.11	Relative und absolute Pfade	39
5.12	Absolute Pfade	40
6	Dateien und Verzeichnisse erstellen	41
6.1	Dateinamen unter Linux	41
6.2	Dateien mit touch erstellen	42
6.3	Verzeichnisse mit mkdir erstellen	43
6.4	Zusammenfassung	44
7	Der Texteditor nano	45
7.1	Bearbeiten einer Datei in nano	45
7.2	Speichern und Beenden in nano	46
7.3	Suchen und Ersetzen	47
7.4	Nützliche weitere Shortcuts	48
8	Der Texteditor vim	49
8.1	Die Familie der Vi-Editoren	50
8.2	Arbeiten mit vim	51
8.3	Navigation in vim	52
8.4	Text ersetzen	54
9	Dateien verwalten	57
9.1	Dateien mit rm löschen	57
9.2	Dateien in anderen Verzeichnissen löschen	58
9.3	Warnung für neugierige Terminal-Abenteurer	60
9.4	Ordner verschieben mit mv	60
9.5	Dateien umbenennen mit mv	62
9.6	Dateien kopieren mit cp	64
10	Tastaturkürzel und Shortcuts	67
10.1	Warum Tastaturkürzel?	67
10.2	Textbearbeitung Killing & Yanking	68
10.3	Verlauf und History	68
11	Arbeiten mit Dateien	70
11.1	cat	70

11.2	less	71
11.3	tac und rev	72
11.4	head und tail	73
11.5	wc	74
11.6	Fazit	76
12	Standard-Streams und Umleitungen	77
12.1	Standard-Streams verstehen	78
12.2	Ausgabe in Dateien umleiten >	78
12.3	Ausgabe anhängen »	79
12.4	Fehlerausgabe umleiten 2>	80
12.5	Fehler und Ausgabe trennen oder kombinieren	80
12.6	Zusammenfassung	80
13	Arbeiten mit Pipelines	82
13.1	Was sind Pipelines?	82
13.2	Erstes Beispiel mit Übungsdateien	82
13.3	Ausgabe weiterverarbeiten	83
13.4	tr	84
13.5	Unterschied zwischen Redirect und Pipe	86
13.6	tee Ausgabe speichern und gleichzeitig weiterleiten	86
14	Shell-Expansionen	88
14.1	Grundlagen der Pfaderweiterung	88
14.2	Zeichenklassen []	89
14.3	Tilde-Erweiterung	91
14.4	Die Magie der geschweiften Klammern	92
14.5	Arithmetische Erweiterung	95
14.6	Einfaches vs. doppeltes Quoting	97
14.7	Reihenfolge der Ausführung	99
15	Suchen und finden	100
15.1	Dateien nach Namen finden	100
15.2	Dateien nach Änderungszeit finden	100
15.3	Kombination mit grep	101
16	Einführung in grep	103
16.1	Diverse Optionen	103
16.2	Kontext rund um den Treffer	104
16.3	Piping zu grep	106
17	Dateiberechtigungen im Überblick	108
17.1	Symbolische Schreibweise	109
17.2	Oktale Schreibweise	110
17.3	Best Practices	111
17.4	Einen Testbenutzer anlegen	111
17.5	Besitzer einer Datei ändern	112

17.6 Zusammenfassung	112
18 Benutzer und Administratorrechte in Linux	114
18.1 Der Benutzer root und sudo	115
18.2 Der besondere Benutzer root	115
18.3 User tux kann kein sudo ausführen	116
18.4 Zusammenfassung	117
19 Umgebungsvariablen und Alias	118
19.1 Umgebungsvariablen verstehen	118
19.2 Eigene Variablen definieren	119
19.3 Variable dauerhaft speichern	119
19.4 Alias - Befehle abkürzen	120
19.5 Alias und Variablen kombinieren	121
19.6 Alias anzeigen oder löschen	121
19.7 Mini-Test	121
19.8 Fazit	122
20 Einführung ins Bash-Scripting	123
20.1 Was ist ein Bash-Skript?	123
20.2 Ein Blick in ein Bash-Skript	123
20.3 Unser erstes Skript	124
20.4 Erklärung der Befehle	125
20.5 Die Shebang-Zeile verstehen	125
20.6 Die PATH-Variable	126
20.7 Skript zum PATH hinzufügen	127
20.8 Fazit	128
21 Cron - Aufgaben automatisch planen	129
21.1 Was ist Cron?	129
21.2 Die Crontab öffnen	129
21.3 Aufbau eines Cronjobs	129
21.4 Dein erster Cronjob	130
21.5 Cronjobs anzeigen und löschen	131
21.6 Fehlerbehandlung in Cronjobs	131
21.7 Automatisches Backup mit Cron	132
21.8 Das Backup mit Cron automatisieren	133
21.9 Cronjobs testen und Fehler provozieren	133
21.10 Cron-Dienst prüfen	133
21.11 Fazit	134
22 Exkurs sed und awk	135
22.1 Text ersetzen und löschen mit sed	135
22.2 Grundlagen von awk Daten zeilenweise verarbeiten	135
22.3 Beispiele mit deinen Testdaten	136
22.4 Vergleich grep, awk und sed	136
22.5 Wann welches Tool sinnvoll ist	136

23 Abschließende Worte	137
23.1 Bonus	137
23.2 Downloadlinks	138
23.3 Über den Autor	139
23.4 Weiteres vom Autor	140
23.5 Stichwortverzeichnis	141

0 Einleitung

Willkommen zum **Linux CLI - Die Macht der universellen Linux-Befehle**! Ich bin Edi, dein Trainer in diesem Kursbuch. Ich unterrichte sowohl vor Ort als auch online und habe schon vielen Menschen geholfen, in die Linux- und Mikrocontroller-Welt einzusteigen. Ich habe Spaß daran, komplexe Themen einfach und klar zu vermitteln, und freue mich darauf, dich auf deinem Weg in die Welt der Befehlszeile zu begleiten.

0.1 Warum die Befehlszeile wichtig ist

Die Arbeit mit der **Befehlszeile** kann anfangs einschüchternd wirken. Viele vermeiden sie lieber, dabei ist sie eines der mächtigsten Werkzeuge überhaupt. Je früher du den Umgang mit ihr lernst, desto schneller verstehst du dein System. Ziel dieses Kurses ist es, dir volle Kontrolle über dein System zu geben und dich sicher im Umgang mit der Befehlszeile zu machen.

Du lernst, wie du die Befehlszeile effizient nutzt, um Aufgaben schneller und präziser zu erledigen.

Wir behandeln wichtige Konzepte und grundlegende Befehle und tauchen dabei auch tiefer in einzelne Themen ein, um ein solides Verständnis aufzubauen.

0.2 Dein Einstieg in die Praxis

Dieser Kurs ist bewusst **praxisorientiert** aufgebaut. Nimm dir regelmäßige Zeit, die Übungen **selbst** auszuprobieren - sie wurden speziell für dieses Buch entwickelt. Nur durch eigenes Tippen und Ausführen der Befehle wirst du wirklich sicher im Umgang mit der **Befehlszeile**. Reines Lesen ist wie Fernsehen: informativ, aber ohne Übung bleibt das Gelernte nicht hängen.

0.3 Universelle Linux-Befehle

In der Linux-Welt gibt es eine riesige Menge an Befehlen - viele davon stammen aus zusätzlichen Paketen oder Tools, die du erst installieren musst. In diesem Kurs konzentrieren wir uns jedoch auf die **universellen Befehle**, also jene, die **standardmäßig** auf den meisten Systemen bereits vorhanden sind - egal ob du **Debian**, **Arch**, **Fedora** oder ein anderes Linux nutzt.

Diese Befehle sind das **Grundgerüst jedes Systems** und bilden das Rückgrat der täglichen Arbeit in der Kommandozeile. Sie funktionieren fast überall - auch auf **Servern**, **minimalen Systemen** oder in **Docker-Containern**, wo du oft keine grafische Oberfläche hast.

Wenn du diese Befehle sicher beherrschst, kannst du dich auf (fast) **jedem Linux-System zurechtfinden**, ohne vorher etwas installieren zu müssen. Alles, was du in diesem Buch lernst, basiert auf diesen **Standard-Tools**, die dir als Fundament dienen, bevor du später komplexere oder spezialisierte Programme entdeckst.

0.4 Übungsdateien und E-Book herunterladen

Alle Unterlagen findest du im GitHub-Repository: <https://github.com/pixelEDI/linux-cli-course>

Für den Einstieg brauchst du nur **linux_cli_dummydata.sh**. Das Skript erstellt automatisch alle benötigten Ordner und Dateien - perfekt zum Üben, Bearbeiten, Suchen und Manipulieren. Wenn du alles zurücksetzen möchtest, lösche die erzeugten Dateien und führe das Skript einfach erneut aus.

Im Repository findest du immer die aktuelle Version der Code-Beispiele. Du kannst das Projekt klonen oder als ZIP-Datei herunterladen - ohne Anmeldung. Ohne Git: <> **Code** → **Download ZIP** anklicken, um die Dateien direkt im Browser zu laden.

Den Download-Link für das PDF findest du am Ende des Buches im Kapitel **Downloadlinks**.

0.5 Übungsdateien ausführen

Falls du bereits ein Linux-System hast, kannst du die Übungsdaten schon herunterladen und das Skript ausführen. Andernfalls kommen wir später darauf zurück, wenn du dein System wie in Kapitel 2 gezeigt eingerichtet hast. Wenn du noch keine Vorkenntnisse mit dem Terminal hast, wirf am besten einen Blick in Abschnitt 3.1, dort wird das Terminal genauer erklärt.

Ich habe ein kleines Bash-Skript vorbereitet, das deine Übungsumgebung (viele Ordner und Test-Dateien) automatisch aufbaut. Am besten wechselst du zuerst in dein Home-Verzeichnis:

```
1 cd ~
```

Lege dort die heruntergeladene Datei **linux_cli_dummydata.sh** ab. Damit sie ausgeführt werden kann, musst du ihr einmalig die **Ausführungsrechte** geben:

```
1 chmod +x linux_cli_dummydata.sh
```

Jetzt kannst du das Skript starten:

```
1 ./linux_cli_dummydata.sh
```

Nach ein paar Sekunden ist deine komplette Übungsumgebung erstellt - mit allen Dateien und Ordnern, die du in den kommenden Kapiteln brauchst. Ab jetzt kannst du direkt loslegen und die Befehle selbst ausprobieren.

Tipp: **Lösche** nach jedem Kapitel den **Ordner** und **führe** das **Skript** erneut aus. So hast du immer einen frischen Stand und kannst den Übungen und Inhalten der Kapitel optimal folgen.

0.6 Üben statt Auswendiglernen

Lern die Befehle durch regelmäßiges Anwenden, nicht durch Auswendiglernen. Viele bleiben mit der Zeit von selbst im Gedächtnis. Wenn du etwas vergisst, ist das kein Problem - wichtig ist nur zu wissen, dass es den Befehl gibt und wo du ihn nachschlagen kannst.

Mach dir keinen Druck. Halte dir Notizen oder eine kleine Übersicht bereit und nutze die man-Pages oder Online-Hilfen. Entscheidend ist nicht, alles zu wissen, sondern zu wissen, wo du die Informationen findest.

0.7 Schreibweise von Shortcuts

In meinen Büchern schreibe ich Tastenkombinationen oft so:

1 **CTRL + A**

Das bedeutet, dass du gleichzeitig die **Strg-** und die **A-Taste** drücken sollst. Ich verwende bewusst **Großbuchstaben**, um Verwechslungen zu vermeiden - zum Beispiel zwischen einem kleinen **l** und einem großen **I**. Die Befehle kannst du in der Regel auch in **Kleinbuchstaben** eingeben. Wenn die Groß- und Kleinschreibung tatsächlich eine Rolle spielt, weise ich ausdrücklich darauf hin.

0.8 Warum ich Denglisch verwende

Viele Kursteilnehmer haben mich in der Vergangenheit gefragt, warum ich in meinen Kursen nicht ausschließlich Deutsch verwende. Die Antwort ist einfach: **Die IT-Welt spricht Englisch**. Fast alle Hilfeseiten, Fehlermeldungen, Dokumentationen und Foren sind auf Englisch verfasst. Wenn du lernst, mit englischen Begriffen umzugehen, findest du schneller Lösungen und verstehst die Werkzeuge besser. Darum verwende ich in diesem Kurs bewusst englische Begriffe, da sie zum täglichen Arbeiten in der IT dazugehören.

0.9 Warum CLI trotz KI wichtig bleibt

1. Verstehen statt nur ausführen

KI kann dir zwar Befehle liefern („Wie installiere ich XYZ?“), aber wenn du nicht verstehst, was diese Befehle tun, tappst du im Dunkeln. Du kannst Fehler nicht einschätzen oder sicher beheben. Du erkennst nicht, wenn ein Befehl potenziell gefährlich ist. Du kannst keine Befehle anpassen oder automatisieren. Ein solides Grundverständnis der CLI gibt dir also Kontrolle statt nur Vertrauen in eine KI.

2. Effizienz und Automatisierung

Wer sich in der CLI wohlfühlt, arbeitet deutlich schneller. Skripte für wiederkehrende Aufgaben, Batch-Verarbeitung großer Datenmengen oder Automatisierung von Deployments, Backups und Tests werden einfacher. KI

kann dir diese Skripte schreiben, aber du kannst sie nur wirklich nutzen, wenn du verstehst, was sie tun.

3. Fundament für viele IT-Bereiche

Die Linux-CLI ist die Grundlage für DevOps, Cloud Computing, Administration, Data Science (etwa mit Bash, SSH, Cronjobs), Cybersecurity und Entwicklung mit Build-Systemen, Containern und CI/CD.

Ohne CLI-Kenntnisse wirst du in all diesen Bereichen schnell an Grenzen stoßen.

4. KI ist ein Werkzeug, kein Ersatz für Verständnis

KI kann dir den Weg zeigen, aber du musst wissen, was du tust. Es ist wie mit einem Navigationssystem: Wenn du nie gelernt hast, Karten zu lesen, bist du verloren, sobald das GPS falsche Daten liefert.

Fazit

Die Linux-CLI zu lernen lohnt sich mehr denn je. KI macht den Lernprozess einfacher, sie ersetzt ihn aber nicht.

Mit KI kannst du interaktiv lernen, dir Übungen oder Lernpfade generieren lassen und sofortige Erklärungen bekommen. Aber das Wissen selbst ist deine Macht. Wenn du die CLI beherrscht und KI nutzt, bist du unschlagbar produktiv.

0.10 KI-Transparenz-Hinweis

Heutzutage ist oft schwer zu erkennen, ob ein Video echt ist oder ob ein Text überhaupt von einem Menschen überprüft wurde. Überall tauchen KI-generierte Inhalte auf und vieles wirkt austauschbar oder künstlich. Deshalb möchte ich offen darlegen, wie dieses Buch entstanden ist.

Dieses Buch wurde vollständig von mir selbst erstellt und inhaltlich ausgearbeitet. Ich habe KI-Tools nur unterstützend genutzt, ähnlich wie man Suchmaschinen, Foren oder Dokumentationen verwendet.

Alle Inhalte, Beispiele und Erklärungen stammen von mir. Künstliche Intelligenz kam lediglich zum Einsatz, um Formulierungen zu verfeinern, Ideen zu strukturieren oder technische Details zu diskutieren und zu verfeinern.

Die Verantwortung und Urheberschaft liegen vollständig bei mir. KI hat mir geholfen, aber sie hat nichts eigenständig erstellt. Sie war ein Werkzeug und ist nicht der Autor.

0.11 MIT-Lizenz und Haftungsausschluss

Dieses Buch und alle dazugehörigen Übungsdateien stehen unter der **MIT-Lizenz**. Das bedeutet: Du darfst die Beispiele **frei verwenden, verändern und weitergeben**, solange der ursprüngliche Urheber genannt wird - in diesem Fall **pixeledi**.

Ich übernehme **keine Haftung** für eventuelle Schäden oder Datenverluste, die durch die Verwendung der gezeigten Befehle oder Skripte entstehen können. Alle Beispiele sind mit größter Sorgfalt erstellt und getestet, richten sich aber an Lernende, die in einer sicheren Umgebung experimentieren.

Wenn du also Kommandos wie `rm`, `chmod` oder `sudo` ausprobierst, mach das am besten in deinem Übungsordner oder in einer virtuellen Umgebung. So lernst du gefahrlos - und genau das ist das Ziel dieses Kurses.

Viel Spaß beim Ausprobieren, Lernen und Entdecken der Linux-Welt!

1 Die Welt der Betriebssysteme

Wer schon einmal ein Video von mir auf YouTube oder in einem Online-Kurs gesehen hat, weiß: Ich bin ein begeisterter Linux-Fan. Ich liebe es, mich stundenlang in Konfigurationen zu vertiefen, Systeme zu verbessern und ständig Neues über Linux zu lernen.

Schau gerne regelmäßig auf meinem YouTube-Kanal vorbei und lass dich von aktuellen Themen rund um Linux inspirieren: www.youtube.com/@pixeledi

In diesem Kapitel bekommst du einen Überblick über die wichtigsten Betriebssysteme und erfährst, wie sie miteinander verbunden sind.

1.1 Die großen Betriebssysteme

Grundsätzlich gibt es vier große Welten:

- **Windows** - das bekannteste System, das viele im Alltag verwenden. Es ist weit verbreitet und wird vor allem im Büro- und Gaming-Bereich eingesetzt.
- **Linux** - das freie und offene System mit vielen Varianten. Es bietet Flexibilität, Stabilität und Kontrolle. Ideal für Server, Entwickler und alle, die ihr System verstehen wollen.
- **macOS** - basiert auf Unix und kombiniert Stabilität mit einer benutzerfreundlichen Oberfläche. Besonders beliebt bei Kreativen und Entwicklern. Es hat ein starkes Terminal, das viele Unix- und Linux-Befehle unterstützt.
- **Unix** - das ältere, aber sehr einflussreiche System, das die Basis vieler moderner Betriebssysteme bildet. Auch heute steckt es in Servern, macOS und iOS.

1.2 Die Linux-Familie

Linux ist keine einzelne Software, sondern eine ganze Familie von Systemen, die alle den **Linux-Kernel** nutzen. Die Wahl einer Distribution ist oft Geschmackssache. Trotzdem haben sich einige Systeme besonders bewährt.

Beispiele:

- **Debian** - sehr stabil und zuverlässig, mit gut getesteten Versionen. Ideal für Server. **Ubuntu** und **Linux Mint** basieren auf Debian. Ich verwende Debian selbst gern auf Servern, weil es einfach funktioniert.
- **Arch Linux** - wird laufend aktualisiert und ist immer auf dem neuesten Stand. Es richtet sich an Nutzer, die ihr System im Detail verstehen möchten. Bekannte Abkömmlinge sind **Manjaro**, **EndeavourOS** oder **CachyOS**. Ich selbst nutze Arch seit Jahren und hatte bisher keine Probleme.

Weitere beliebte Distributionen:

- **Fedora** - modern und experimentierfreudig
- **openSUSE** - mit *Leap* (stabil) und *Tumbleweed* (Rolling Release)
- **Kali Linux** / **Parrot Security** - für Sicherheit und Penetration Tests
- **Raspberry Pi OS** - für Bastelprojekte und kleine Geräte
- und noch vieleeeeeee mehr!

Jede Distribution hat ihren eigenen Fokus. Der große Vorteil von Linux ist, dass du die Variante wählen kannst, die zu deinem Stil passt.

1.3 Die Unix-Familie

Unix selbst wird heute selten direkt verwendet, bleibt aber die Grundlage vieler moderner Systeme. **macOS** und **iOS** bauen auf Unix-Prinzipien auf, genauso wie viele Serverbetriebssysteme.

Auch **FreeBSD**, **OpenBSD** und **NetBSD** sind bekannte Vertreter. Sie sind besonders stabil, sicher und werden oft in spezialisierten Bereichen genutzt. Ich sehe **OpenBSD** häufig dort, wo Sicherheit an erster Stelle steht.

Ein besonderes Beispiel ist **TrueNAS**:

- **TrueNAS CORE** basiert auf FreeBSD und ist für NAS-Server mit dem ZFS-Dateisystem optimiert.
- **TrueNAS SCALE** basiert auf Linux (Debian) und erweitert das Konzept um Container und Virtualisierung.

Unix lebt also weiter - als Fundament vieler moderner Systeme, vom Desktop bis zum Rechenzentrum.

1.4 Unix und seine Philosophie

Unix wurde Ende der 1960er Jahre bei **Bell Labs** entwickelt. Es war von Anfang an auf Stabilität, Einfachheit und Klarheit ausgelegt.

Wichtige Prinzipien sind:

- **Mehrbenutzerbetrieb** - mehrere Menschen können gleichzeitig am System arbeiten.
- **Hierarchisches Dateisystem** - alles ist logisch strukturiert in Ordnern.

Die **Unix-Philosophie** lautet:

Mach eine Sache, und mach sie gut.

Das bedeutet, dass kleine Programme jeweils nur eine Aufgabe übernehmen, sich aber miteinander kombinieren lassen. Diese Idee lebt in der **Bash** und in fast allen Linux-Werkzeugen weiter.

1.5 True Unix und Unix-Like Systeme

Der Begriff *Unix* ist geschützt und darf nur von Systemen verwendet werden, die den offiziellen Standard erfüllen - dazu gehört **macOS**. **Linux** dagegen ist ein

LESEPROBE

Kurz und lang

Kurzform	Langform	Bedeutung
-r	--reverse	Reihenfolge umkehren
-u	--universal	UTC-Zeit anzeigen
-h	--help	Hilfe anzeigen

- Kurzformen sind schneller.
- Langformen sind leichter zu lesen.

Beispiel: date

```
1 date -u
2 date --universal
```

Beide Befehle zeigen die Zeit in UTC.

Beispiel: sort

```
1 sort colors.txt
2 sort -r colors.txt
3 sort --reverse colors.txt
```

Alle drei funktionieren gleich. Die Langform ist besser zu verstehen.

Hinweise

- Nicht jeder Befehl hat Langformen.
- Langformen beginnen immer mit zwei Minuszeichen.
- Du kannst sie nicht kombinieren.
- Mit diesem Befehl siehst du alle Optionen:

```
1 befehl --help
```

Fazit

Langformen machen Befehle klarer und lesbarer. Sie sind gut zum Lernen und für den Anfang.

Beispiel:

```
1 sort --reverse colors.txt
```

ist leichter zu verstehen als

```
1 sort -r colors.txt
```

Im nächsten Kapitel lernst du, Optionen mit Werten zu kombinieren.

4 Hilfe und Informationen in der Linux-CLI

In diesem Kapitel geht es darum, wie du dir selbst helfen kannst, wenn du einmal nicht weiter weißt.

Die Linux-Kommandozeile bringt viele Werkzeuge mit, die dir zeigen, was ein Befehl tut, welche Optionen es gibt und wie du sie verwendest.

Du musst also keine langen Listen auswendig lernen - du lernst stattdessen, **wo du nachsehen kannst**.

So bekommst du Schritt für Schritt ein besseres Verständnis für das System und wirst sicherer im Umgang mit der CLI.

Kurz gesagt: Dieses Kapitel zeigt dir, wie du Antworten findest, ohne Google oder ChatGPT zu fragen.

4.1 Die wichtigsten Werkzeuge

Das wichtigste Werkzeug ist der Befehl **man**, kurz für *manual*.

Damit öffnest du die integrierten Handbuchseiten (Man-Pages) zu fast jedem Befehl im System.

Beispiel:

```
1 man ls
```

zeigt das Handbuch zum Befehl **ls**.

4.2 Der Befehl man

Die Syntax ist einfach:

```
1 man <befehl>
```

Beispiele

```
1 man cal
```

Zeigt die Dokumentation für den Befehl **cal**.

```
1 man date
```

Zeigt die Handbuchseite zu **date**.

```
1 man echo
```

Öffnet die Dokumentation zu **echo**.

4.3 Navigation in Man-Pages

Wenn du eine Man-Page geöffnet hast, kannst du dich mit diesen Tasten bewegen:

Taste	Funktion
↑ / ↓	Zeilenweise scrollen
Leertaste	Eine Seite nach unten
b	Eine Seite nach oben
/ + Suchbegriff	Nach Text suchen
n	Nächstes Suchergebnis
h	Hilfe anzeigen
q	Beenden

Beispiel: Öffne mit `man ls` das Handbuch, tippe `/sort` und drücke Enter, um nach `sort` zu suchen. Mit `n` springst du weiter, mit `q` verlässt du die Seite.

Diese Steuerung funktioniert, weil `man` intern das Programm `less` verwendet - ein Textbetrachter, den du später noch genauer kennenlernenst.

Tipps

- Man-Pages sind in Abschnitte unterteilt, meist mit **NAME**, **SYNOPSIS**, **DESCRIPTION** und **OPTIONS**.
- Wenn du dir unsicher bist, kannst du auch einfach schreiben:

```
1 man man
```

Das zeigt dir die Dokumentation zum Befehl `man` selbst.

- Mit diesem Befehl findest du alle Handbuchseiten, in denen ein bestimmtes Wort vorkommt:

```
1 man -k time
```

So kannst du alle Befehle durchsuchen, die etwas mit `time` zu tun haben.

Fazit

Mit `man` hast du direkten Zugriff auf die vollständige Dokumentation deines Systems. Du brauchst keine Internetverbindung und kein Zusatzprogramm - alles ist schon da. Das ist die wichtigste Grundlage, um dich sicher in der Kommandozeile zu bewegen.

4.4 Die Befehle `type` und `which`

Neben `man` gibt es noch kleine, aber nützliche Befehle, mit denen du herausfinden kannst, **was ein Befehl genau ist** und **woher er kommt**.

`type`

Zeigt an, **welcher Art** ein Befehl ist - also ob er ein eingebauter Shell-Befehl, ein externes Programm, eine Funktion oder ein Alias ist.

Beispiele:

```
1  type cd
2  # cd ist ein Shell-Builtin
3
4  type clear
5  # clear ist /usr/bin/clear
6
7  type ls
8  # ls ist ein Alias für 'ls --color=auto'
```

which

Zeigt den Pfad eines ausführbaren Programms, falls es eines ist.

```
1  which bash
2  # /usr/bin/bash
```

Das ist nützlich, wenn du wissen willst, welche Version eines Programms ausgeführt wird, besonders wenn du eigene Skripte oder mehrere Versionen installiert hast.

Tipp: Wenn du wissen willst, **was** ein Befehl ist, nimm **type**. Wenn du wissen willst, **wo** er liegt, nimm **which**.

5 Orientierung in der Linux-Befehlszeile

In dieser Sektion lernst du, wie du dich **im Dateisystem über die Kommandozeile bewegst** - also, wie du weißt, *wo du bist, was sich dort befindet* und *wie du woanders hingelängst*.

Die Kommandozeile ist nicht nur ein Werkzeug, um Befehle auszuführen - sie ist auch dein **Navigator durch das gesamte System**.

Sobald du verstehst, wie man sich in Verzeichnissen bewegt, kannst du Dateien finden, anlegen, verschieben oder löschen - und das alles sehr schnell.

5.1 Wichtige Befehle

In diesem Kapitel beschäftigen wir uns mit drei der wichtigsten Befehle, die du ständig brauchen wirst:

- **pwd** - zeigt das **aktuelle Verzeichnis** an
- **ls** - listet den **Inhalt eines Verzeichnisses** auf
- **cd** - wechselt in ein **anderes Verzeichnis**

Diese drei Kommandos sind das Fundament jeder Arbeit im Terminal.

Wir sprechen außerdem über:

- **Wichtige Verzeichnisse** wie dein *Home-Verzeichnis* und das *Root-Verzeichnis*
- **Relative und absolute Pfade** - also, wie du dich im System gezielt bewegst

Damit legst du die Grundlage, um später komplexere Befehle mit Pfaden und Dateien sicher zu nutzen.

5.2 Root-Verzeichnis

Unter Linux - egal ob Debian, Ubuntu, Fedora oder Arch - ist alles Teil **eines einzigen, hierarchischen Dateisystems**.

Im Gegensatz zu Windows gibt es keine Laufwerksbuchstaben wie **C:** oder **D:**. Alles beginnt bei einem Punkt: **/** - dem sogenannten **Root-Verzeichnis**.

Das Root-Verzeichnis ist die **oberste Ebene** im Linux-Dateisystem.

Jede Datei und jeder Ordner befindet sich irgendwo darunter.

Du kannst es dir wie den Stamm eines Baums vorstellen - alles andere sind Verzweigungen.

Beispiel:

```
1 cd /
2 ls
```

LESEPROBE

9 Dateien verwalten

In diesem Kapitel lernst du drei zentrale Befehle kennen, die du in der Linux-Kommandozeile ständig verwenden wirst:

- **rm** - zum **Löschen** von Dateien und Ordnern
- **mv** - zum **Verschieben oder Umbenennen** von Dateien
- **cp** - zum **Kopieren** von Dateien und Verzeichnissen

Diese drei Kommandos sind das Fundament jeder Dateiverwaltung im Terminal.

Warum das wichtig ist

Mit diesen Befehlen kannst du deine Dateien direkt über die Kommandozeile verwalten - ohne Dateimanager oder Maus.

Egal, ob du Skripte schreibst, Backups anlegst oder auf einem Server arbeitest - du wirst sie regelmäßig brauchen.

Beispielübersicht

Aufgabe	Befehl
Datei löschen	<code>rm datei.txt</code>
Datei verschieben	<code>mv datei.txt zielordner/</code>
Datei kopieren	<code>cp datei.txt kopie.txt</code>

9.1 Dateien mit rm löschen

Nachdem wir im Verlauf dieses Buches bereits gelernt haben, wie man Dateien und Ordner erstellt, schauen wir uns nun an, **wie man sie wieder löscht**. Dafür verwenden wir den Linux-Befehl **rm**, der zu den am häufigsten genutzten (und gefährlichsten) gehört.

rm steht für **remove** und wird verwendet, um **Dateien** aus dem Dateisystem zu löschen.

Beispiel:

```
1 rm dateiname.txt
```

Damit wird die angegebene Datei sofort entfernt - **ohne Papierkorb, ohne Rückfrage** (es sei denn, du verwendest eine Schutzoption).

Beispiel mit unseren Testdaten

Wechsle in dein Übungsverzeichnis, das du mit dem Setup-Skript erstellt hast:

```
1 cd linux_cli_kurs/02_files
```

Zeige die Dateien an:

```
1 ls
```

Du solltest unter anderem die Datei `shells.txt` sehen. Diese löschen wir nun:

```
1 rm shells.txt
```

Prüfe danach noch einmal mit:

```
1 ls
```

Die Datei `shells.txt` ist jetzt verschwunden.

Sicherheitsmodus: Mit Bestätigung löschen

Wenn du versehentlich etwas löschen könntest, verwende die Option `-i` (interaktiv):

```
1 rm -i passwd_sample.txt
```

Das System fragt dich:

```
1 rm: remove regular file 'passwd_sample.txt'? y
```

Erst wenn du `y` (Yes) bestätigst, wird gelöscht.

9.2 Dateien in anderen Verzeichnissen löschen

Du kannst Dateien auch direkt mit Pfad löschen, ohne vorher das Verzeichnis zu wechseln:

```
1 rm 01_basics/opinion.txt
```

Oder mehrere Dateien gleichzeitig:

```
1 rm 01_basics/hello.txt 01_basics/today.txt
```

Wichtiger Hinweis

Wenn du eine Datei mit `rm` löschst, ist sie **dauerhaft weg** - es gibt **keinen Papierkorb** wie unter Windows oder macOS. Deshalb:

Lösche niemals leichtfertig - prüfe mit `ls`, bevor du `rm` verwendest!

Mit `rm` kannst du Dateien schnell und direkt entfernen. Im Zusammenspiel mit deinen Übungsdateien aus `linux_cli_kurs` kannst du das gefahrlos ausprobieren und so ein Gefühl für den Befehl entwickeln, bevor du ihn bei echten Systemdateien einsetzt.

Ordner mit `-d` und `-r` löschen

Nachdem du nun weißt, wie man einzelne **Dateien mit `rm`** löscht, schauen wir uns an, wie man **Ordner** entfernt.

Das funktioniert ebenfalls mit `rm`, allerdings mit ein paar zusätzlichen Optionen.

Leere Ordner löschen mit `-d`

Wenn du einen **leeren Ordner** löschen möchtest, kannst du die Option `-d` verwenden (steht für *directory*).

Beispiel aus deinem Übungsverzeichnis:

```
1 cd linux_cli_kurs/02_files
2 mkdir testleer
3 ls
```

Jetzt ist der Ordner `testleer` sichtbar. Löschen kannst du ihn mit:

```
1 rm -d testleer
```

Prüfe danach:

```
1 ls
```

Der Ordner `testleer` ist verschwunden. Vermutlich wirst du diese Option nicht sehr oft brauchen, aber die nächste ist dafür umso wichtiger.

Ordner mit Inhalt löschen mit `-r`

Wenn ein Ordner **Dateien oder Unterordner** enthält, funktioniert `rm -d` **nicht**. Dann brauchst du die Option `-r` (steht für *recursive*), um alles im Ordner inklusive Unterverzeichnissen zu entfernen.

Beispiel: Im Setup-Skript wurde der Ordner `02_files/backups` erstellt. Diesen kannst du so löschen:

```
1 rm -r backups
```

Damit wird der gesamte Ordner **rekursiv** gelöscht - also inklusive aller Dateien, die darin liegen.

Sicherheitsmodus: `-ri`

Wenn du beim Löschen von ganzen Verzeichnissen lieber eine Rückfrage möchtest, kombiniere `-r` mit `-i`:

```
1 rm -ri 04_arch_is_the_best
```

Das System fragt dich dann bei jeder Datei:

```
1 rm: remove regular file 'manifesto.txt'? y
2 rm: remove regular file 'sticker.txt'? y
3 rm: remove regular file 'update_commands.txt'? y
```

So kannst du jede Löschung bestätigen.

Ohne Rückfragen löschen: `-f`

Wenn du **keine Bestätigung** möchtest und auch **Fehler ignorieren** willst (zum Beispiel, wenn eine Datei nicht existiert), kannst du `-f` (*force*) verwenden:

```
1 rm -rf backups
```

Diese Kombination löscht alles im angegebenen Ordner **sofort und ohne Nachfrage**. Sei hier besonders vorsichtig - ein falscher Pfad kann dein gesamtes System leeren.

9.3 Warnung für neugierige Terminal-Abenteurer

`rm -rf /*` ist der legendäre Selbstzerstörungsbefehl unter Linux. Er löscht **alles** - wirklich **alles** - ab dem Wurzelverzeichnis `/`, also das komplette System inklusive deiner letzten Hoffnung. Das bedeutet: Nach dem Ausführen hängt das System, und beim nächsten Start hast du im Grunde **kein Betriebssystem mehr**, weil du es schlichtweg gelöscht hast.

Also immer zweimal hinschauen, bevor du irgendetwas mit `rm -rf` aus dem Internet kopierst - sonst lernst du sehr schnell, was ein frisches System wirklich bedeutet.

Fazit

Option	Bedeutung	Beispiel
<code>-d</code>	Löscht leere Ordner	<code>rm -d testleer</code>
<code>-r</code>	Löscht Ordner mit Inhalt (rekursiv)	<code>rm -r backups</code>
<code>-ri</code>	Fragt vor jedem Löschen nach	<code>rm -ri 04_arch_is_the_best</code>

Nutze besonders `rm -r` vorsichtig. Dieser Befehl kann ganze Verzeichnis-Strukturen löschen. Am besten übst du mit deinem Kursverzeichnis `linux_cli_kurs`, bevor du ihn bei echten Daten einsetzt.

9.4 Ordner verschieben mit `mv`

Nachdem wir gesehen haben, wie man Ordner erstellt und löscht, geht es nun darum, sie mit `mv` zu verschieben.

```
1 mv <quellordner> <zielordner>
```

Wenn `<zielordner>` **existiert**, wird der Quellordner **in** diesen Zielordner verschoben. Wenn er **nicht existiert**, wird der Quellordner **umbenannt**.

Beispiel mit unseren Testdaten

Hier empfiehlt es sich, das Skript noch einmal auszuführen, damit wir wieder den Ausgangszustand unserer Testdaten hergestellt haben.

Wechsle in dein Übungsverzeichnis:

```
1 cd ~/linux_cli_kurs
2 ls
```

Du solltest dort Ordner wie `01_basics`, `02_files`, `03_scripts`, `04_arch_is_the_best` usw. sehen.

Einen Ordner verschieben

Wir verschieben den Ordner `03_scripts` in den Ordner `02_files`:

```
1 mv 03_scripts 02_files/
```

Jetzt befindet sich `03_scripts` **innerhalb** von `02_files`. Überprüfen kannst du das so:

```
1 ls 02_files
```

Du solltest dort den neuen Unterordner `03_scripts` sehen.

Schauen wir uns das an einem kleinen Beispiel an. Im Ordner `01_basics` liegt bereits die Datei `distros.txt`. Wir erstellen darin testweise einen neuen Unterordner `colors` mit einer kleinen Datei:

```
1 mkdir 01_basics/colors
2 echo -e "rot\ngrün\nblau" > 01_basics/colors/colors.txt
3
4 mkdir 01_basics/colors2
5 echo -e "rot\ngrün\nblau" > 01_basics/colors2/colors.txt
```

Der Unterschied mit oder ohne /

Das `/` am Ende eines Ordernamens zeigt der Shell an, dass du den **Inhalt des Ordners** meinst. Bei `mv` ist das jedoch egal, beide Varianten verhalten sich identisch.

Beispiel:

```
1 mv 01_basics/colors/ 02_files/
2 mv 01_basics/colors2 02_files
```

In beiden Fällen wird der Ordner `colors` samt Inhalt nach `02_files` verschoben.

LESEPROBE

Mit der Option `-d` zeigt `uniq` **nur doppelte Einträge** an. Das ist praktisch, wenn du herausfinden möchtest, welche Artikel mehrfach in deiner Liste stehen.

```
1 sort 02_files/big_list.txt | uniq -c
```

Damit zeigt `uniq` jede Zeile zusammen mit der **Anzahl ihrer Vorkommen** an, zum Beispiel:

```
1      3 Ananas
2     10 Apfel
3      3 Banane
4      4 Birne
5      ...
```

Wenn du nur wissen willst, **wie oft Ananas vorkommt**, kannst du das Ergebnis mit `grep` filtern. Zu `grep` gibt es später noch ein eigenes Kapitel, in dem wir uns diesen Befehl genauer ansehen.

```
1 sort 02_files/big_list.txt | uniq -c | grep Ananas
```

→ Ausgabe zum Beispiel:

```
1      3 Ananas
```

So kannst du direkt ablesen, wie oft Ananas in der Datei steht.

Praxisbeispiele

```
1 ls 06_invoices | sort -n
```

`ls 06_invoices` listet alle Dateien im Rechnungsverzeichnis auf. Mit `sort -n` werden diese **nach Nummern sortiert** - also 001..., 002..., 003... usw. Das hilft, Rechnungen oder Dateien chronologisch anzuordnen.

```
1 ls 06_invoices | sort -nr | head -n 5
```

Hier sortierst du die Dateien **numerisch und rückwärts** (`-nr`) und gibst anschließend mit `head -n 5` **die fünf höchsten Rechnungsnummern** aus. Das ist eine einfache Methode, um die **zuletzt erstellten oder aktuellsten Rechnungen** anzuzeigen.

```
1 sort 01_basics/distros.txt > 01_basics/distros_sorted.txt
```

Hier wird die sortierte Ausgabe **in eine neue Datei geschrieben**. `>` leitet die Ausgabe in `distros_sorted.txt` um, ohne die Originaldatei zu verändern. So kannst du sortierte Versionen von Dateien speichern und später weiterverwenden.

Mit `sort` bringst du Ordnung in jede Textdatei - egal ob Listen, Logdaten oder nummerierte Dateien. In Kombination mit `uniq`, `head` oder Pipes kannst du schnell Muster erkennen, Duplikate bereinigen und Daten übersichtlich strukturieren.

11.6 Fazit

Diese Befehle bilden das Fundament der Arbeit mit Dateien in der Linux-Kommandozeile. Mit ihnen kannst du Textdateien lesen, filtern, sortieren, zählen und analysieren. Sie sind unverzichtbar, wenn du Logs prüfst, Daten sammelst oder Skripte schreibst und in Kombination entfalten sie ihre volle Stärke. Dabei verwenden wir stellenweise schon Befehle wie **Pipes (|)** und **grep**, auch wenn wir sie erst in einem späteren Kapitel ausführlich behandeln. Hier helfen sie einfach, die Beispiele praxisnah und sinnvoll zu gestalten.

12 Standard-Streams und Umleitungen

In diesem Kapitel lernst du ein zentrales Konzept der Linux-Kommandozeile kennen:

Standard-Streams und Umleitungen (Redirections).

Diese Mechanismen bilden die Grundlage dafür, wie Programme in Linux miteinander kommunizieren, Daten ausgeben, lesen und weiterverarbeiten. Die hier gezeigten Inhalte wirst du in vielen Skripten wiederfinden. Genau deshalb schauen wir uns in diesem Kapitel die Unterschiede und Einsatzmöglichkeiten im Detail an.

An manchen Stellen greifen wir hier auch vor und verwenden Befehle wie **Pipes** (**|**) und **grep**.

Jedes Programm in der Kommandozeile verfügt über drei Datenströme:

Stream	Beschreibung	Standard-Kanal
STDIN	Standard Input - Eingabe vom Benutzer oder aus einer Datei	Tastatur
STDOUT	Standard Output - normale Ausgabe eines Befehls	Terminal
STDERR	Standard Error - Fehlermeldungen oder Warnungen	Terminal

Diese drei Kanäle sind die Basis jeder Kommunikation zwischen Prozessen in Unix- und Linux-Systemen.

Durch das **Umleiten (Redirecten)** dieser Streams kannst du:

- Ausgaben in Dateien speichern, statt sie im Terminal zu sehen
- Fehlerausgaben getrennt protokollieren
- Befehle miteinander verketteten, sodass die Ausgabe des einen zur Eingabe des nächsten wird

So wird die Kommandozeile zu einem mächtigen Werkzeug für Automatisierung, Datenverarbeitung und Debugging.

Überblick der Themen

In diesem Kapitel lernst du:

- Wie man Ausgaben in Dateien umleitet (> und >>)