Linux von Anfang an, zweiter Teil

Lian Herzberg

07.12.2025

5 Zusammenfassung

5.1 Befehle auf der Kommandozeile

In den ersten Abschnitten haben wir gelernt, die Kommandozeile souverän zu bedienen. Wenn wir jetzt neue Befehle dazulernen, folgt deren Verwendung immer dem vertrauten Schema:

```
> befehl -option parameter
```

Zum Beispiel:

```
> ls -l /home
```

5.2 Funktionsweise von Programmen

Wir haben auch gelernt, dass alle Programme Eingaben (Input) verarbeiten und Ausgaben (Output) zurückgeben können. Das gilt auf der Kommandozeile mit der Standardeingabe und der Standardausgabe (die Standardfehlerausgabe ist hier als eine besondere Art von Ausgabe zu betrachten), aber es gilt darüber hinaus für jede Software, die jemals geschrieben wurde.

Abbildung 1: Jedes Programm muss nicht, aber kann Input verarbeiten und Output produzieren.

6 Unix als Standard

Der Unix-Standard bildet noch heute die Grundlage für viele moderne Betriebssysteme. Linux-Distributionen gelten als unix-artig: Sie folgen in vielen Fällen der

Unix-Designphilosophie und halten sich teilweise an Unix-Standards. Die verschiedenen BSDs sind auch heute in fortlaufender Entwicklung, sie waren ursprünglich Unix-Derivate, sind aber inzwischen komplett umgeschrieben worden.

Der Aufbau und die Benennung bestimmter Verzeichnisse, Konfigurationsdateien und Befehle ist unter vielen unix-artigen Systemen identisch. Dadurch ist der Wechsel zwischen Linux-Distributionen einfach und auch die Verwendung von BSD-Systemen für erfahrene Linux-Admins leicht zu erlernen.

Der POSIX-Standard stellt die gemeinsame Grundlage von Unix-kompatiblen Betriebssystemen dar. Dieser Standard definiert eine Programmierschnittstelle. Eine Programmierschnittstelle dient dazu, beim Schreiben von Software auf Systemfunktionen zugreifen zu können. Dazu gehören Funktionen wie das Lesen und Schreiben von Dateien, der Zugriff auf die Standardeingabe und Standardausgabe, oder auch der Zugriff auf Hardware.

Die Unix-Philosophie beinhaltet auch eine Reihe von Grundregeln, an die unix-artige Systeme sich mal mehr, mal weniger halten. Dazu gehören zum Beispiel die Regel, dass jede Funktion über eine Datei zugänglich gemacht wird: Auf eurem Linux-System gibt es eine Datei für euren Datenträger, eine Datei für eure CPU und eine Datei für eure Grafikkarte.

Eine weitere Philosophie ist der Ansatz, dass jedes Programm genau eine Funktion zu erfüllen hat. Zusätzliche Funktionen sind demnach vorzugsweise durch separate Programme umzusetzen statt durch Erweiterung eines bestehenden Programms.

Natürlich gibt es hier viel Potenzial für Meinungsverschiedenheiten und abweichende Interpretationen. Dies führt dazu, dass die Softwarelandschaft unter unix-artigen Betriebssystemen sehr divers ist und es für jeden Geschmack und jeden Anwendungsfall unterschiedliche Software gibt.

7 Unix-Dateiberechtigungen

Im Unix-Standard kennen wir genau drei Berechtigungen: Lesen, Schreiben und Ausführen (read, write und execute).

Die Leseberechtigung gibt bei Dateien das Recht, ihren Inhalt auszulesen und bei Verzeichnissen das Recht, ihren Inhalt anzuzeigen.

Die Schreibberechtigung gibt bei Dateien das Recht, den Inhalt der Datei zu verändern, die Datei zu verschieben oder zu löschen, und bei Verzeichnissen das Recht, in sie hineinzuschreiben, sie zu verschieben oder das Verzeichnis oder seine Inhalte zu löschen.

Die Ausführberechtigung gibt bei Dateien das Recht, sie auszuführen – was natürlich nur bei Programmen Sinn macht. Bei Verzeichnissen bedeutet diese Berechtigung hingegen, dass sie betreten werden dürfen.

Wir kennen im Unix-Standard außerdem drei Kategorien von Benutzern: die Besitzerin (owner) einer Datei, die Gruppe (group), und alle anderen. Das bedeutet, wir haben dreimal drei Optionen, die entweder »Ja« oder »Nein« lauten können. Um diese Optionen zu speichern, brauchen wir somit neun Bits.

7.1 Darstellung von Berechtigungen

Stellen wir einmal eine Tabelle auf, um unsere neun Optionen darzustellen, und füllen sie mit einem Beispielwert aus:

	Lesen (r)	Schreiben (w)	Ausführen (x)	als Dezimalzahl
Besitzer	1	1	0	6
Gruppe	1	0	0	4
Alle	1	0	0	4

Zum Umrechnen in eine Dezimalzahl lesen wir die Berechtigungen jeweils als eine ganz normale Binärzahl. Die Binärzahl 001 entspricht der Dezimalzahl 1, die Binärzahl 010 entspricht der Dezimalzahl 2 und die Binärzahl 100 entspricht der Dezimalzahl 4. Die jeweils gesetzten Bits werden addiert, darum entspricht die Binärzahl 110 der Dezimalzahl 4+2+0=6.

So erhalten wir für das angegebene Beispiel den Berechtigungcode 644. Die erste Ziffer gibt die Berechtigungen für die Besitzerin an, die zweite Ziffer die Berechtigungen für die Gruppe und die dritte Ziffer die Berechtigungen für alle anderen.

Eine weitere Darstellung der Berechtigungen erhalten wir bei Ausführen des Befehls `ls -l`. Im ersten Feld werden die Berechtigungen für jede Datei angegeben. Die allererste Stelle ist dabei für besondere Eigeschaften definiert, bei Verzeichnissen steht hier ein `d` (für »directory«). Anschließend steht dort ein Code nach dem Schema:

```
rw-r--r--
```

Gesetzte Berechtigungsbit werden mit ihrem jeweiligen Kürzel markiert (r für Lesen, w für Schreiben, x für Ausführen). Nicht gesetzte Berechtigungen werden mit einem Bindestrich dargestellt.

Zur Übung können nach dem selben Schema noch einmal folgende Berechtigungscodes in Tabellenschreibweise dargestellt werden: 510, 777, 310.

	Lesen (r)	Schreiben (w)	Ausführen (x)	als Dezimalzahl
Besitzer	1	0	1	5
Gruppe	0	0	1	1
Alle	0	0	0	0

	Lesen (r)	Schreiben (w)	Ausführen (x)	als Dezimalzahl
Besitzer	1	1	1	7
Gruppe	1	1	1	7
Alle	1	1	1	7

	Lesen (r)	Schreiben (w)	Ausführen (x)	als Dezimalzahl
Besitzer	0	1	1	3
Gruppe	0	0	1	1
Alle	0	0	0	0

Spätestens beim zweiten Beispiel wird klar, dass mit diesem System eine fundamentale Art von Zugriffskontrolle erfolgt. Wir sollten uns berechtigterweise unwohl damit fühlen, einer Datei diese Berechtigungen zuzuweisen.

7.2 Setzen von Berechtigungen

Um Dateiberechtigungen zu modifizieren, wird der Befehl `chmod` verwendet. Damit wir diesen Befehl gefahrlos ausprobieren können, beginnen wir mit einer kleinen Wiederholungsaufgabe.

Aufgabe: Erstelle auf der Kommandozeile ein Verzeichnis namens `linuxkurs`, wechsle in dieses Verzeichnis und erstelle mit `echo` eine Textdatei namens `textdatei.txt`.

Nun können wir uns mit `ls` die Berechtigungen unserer erstellten Textdatei anzeigen lassen:

```
> ls -l textdatei.txt
```

Die Ausgabe wird in etwa so aussehen:

```
-rw-r----- 1 skye skye    6 Dec  7 09:43 textdatei.txt
```

Die Felder der Ausgabe können wir in diesem Abschnitt nur teilweise verstehen, der Vollständigkeit halber werden sie an dieser Stelle benannt:

- Berechtigungen
- Anzahl der Links (bei Dateien meistens 1, bei Verzeichnissen meistens 2)
- Besitzer
- Gruppe
- Dateigröße in Bytes
- Datum der letzten Änderung
- Dateiname

Nun modifizieren wir die Berechtigungen unserer Textdatei.

```
> chmod 777 textdatei.txt
> ls -l textdatei.txt
```

Wir sehen, dass nun sämtliche Berechtigungen gesetzt sind.

```
> chmod 000 textdatei.txt
> ls -l textdatei.txt
```

Nun sind keine Berechtigungen mehr gesetzt. Wenn wir nun versuchen, die Datei zu löschen, erhalten wir eine Fehlermeldung:

```
> rm textdatei.txt
```

Da wir allerdings die Besitzerin der Datei sind, können wir die Berechtigungen wieder ändern. Wir stellen den Ausgangszustand wieder her:

```
> chmod 640 textdatei.txt
> ls -l textdatei.txt
```

Um auszuprobieren, wie sich Berechtigungen auf das Verhalten von Verzeichnissen auswirken, wechseln wir ins übergeordnete Verzeichnis.

```
> cd ..
```

Wenn wir nun den Befehl `ls -l` ausführen, sehen wir in einer langen Liste unser Verzeichnis `linuxkurs`. Allerdings zeigt der Befehl

```
> ls -l linuxkurs/
```

... uns den Inhalt des Verzeichnisses an und nicht etwa die Informationen des Verzeichnisses selbst. Hierfür lernen wir eine weitere Option des `ls`-Befehls kennen:

```
> ls -ld linuxkurs/
```

Um zu demonstrieren, was der Entzug der einzelnen Berechtigungen bewirkt, führen wir folgende Befehle nacheinander aus:

```
> chmod 000 linuxkurs/
> ls -ld linuxkurs/
> ls -l linuxkurs/
> cd linuxkurs/
> rm linuxkurs/textdatei.txt
```

Wir sehen, dass wir weder das Verzeichnis lesen, noch seinen Inhalt ändern dürfen.

```
> chmod 100 linuxkurs/
> ls -l linuxkurs/
> cd linuxkurs/
pwd
```

Nun dürfen wir das Verzeichnis betreten, aber seinen Inhalt nicht lesen.

```
> cd ..
> chmod 400 linuxkurs/
> ls -l linuxkurs/
> cd linuxkurs/
```

Nun dürfen wir den Inhalt des Verzeichnisses auflisten, das Verzeichnis aber nicht betreten.

```
> chmod 500 linuxkurs/
> cd linuxkurs/
> ls -l
> cd ..
> rm -r linuxkurs
```

Jetzt fehlt uns nur noch die Schreibberechtigung.

```
> chmod 700 linuxkurs/
> ls -l linuxkurs/
```

Wer aufmerksam die Ausgabe der `ls`-Befehle betrachtet hat, hat vielleicht gemerkt, dass die Änderung der Berechtigungen immer nur das Verzeichnis selbst betroffen haben, während die Berechtigungen der Textdatei unverändert blieben. In unserer Dateisystem-Baumstruktur haben wir nur den Knotenpunkt des Verzeichnisses modifiziert, aber nicht seine darunter liegenden Äste und Endpunkte:

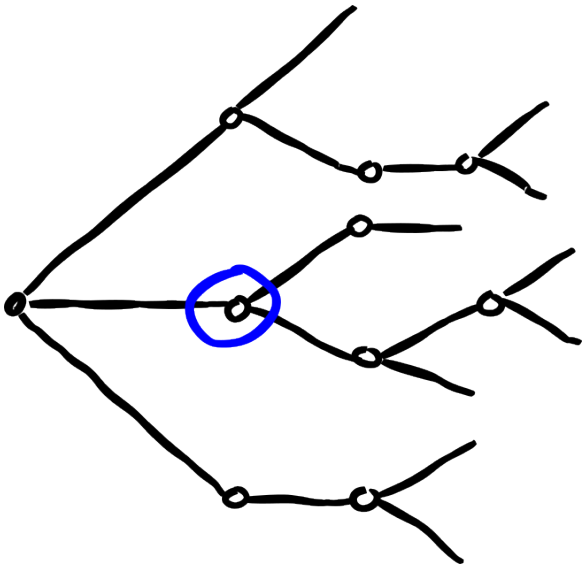

Abbildung 2: Modifikation eines einzelnen Knotenpunktes in der Baumstruktur des Dateisystems.

Wenn wir stattdessen den gesamten Ast modifizieren wollen, nennt man das eine rekursive Aufgabe. Hierfür müssen wir dem `chmod`-Befehl noch eine weitere Option mitgeben:

```
> chmod -R 777 linuxkurs
> ls -l linuxkurs
```

Wenn ein Befehl sich auf einen gesamten Ast in der Dateistruktur bezieht, wird diese Option meistens mit `-R` oder `-r` zugänglich gemacht. Hier ist Vorsicht geboten: Die Groß- und Kleinschreibung ist von Befehl zu Befehl unterschiedlich, manchmal heißen die Optionen auch ganz anders. Wer nicht aufpasst, hat schnell eine ganz falsche Option ausgewählt. Die Man-Pages helfen, die richtigen Optionen zu finden.

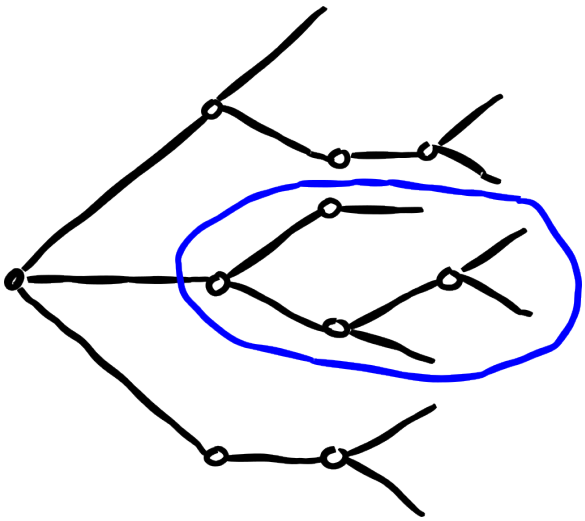

Abbildung 3: Modifikation eines ganzes Astes in der Baumstruktur des Dateisystems.

8 Benutzer und Gruppen

Im Unix-Standard ist jede Datei und jedes Verzeichnis genau einem Benutzer und genau einer Gruppe zugeordnet. Diese Zuordnung sehen wir in der Listenansicht von `ls`.

Jeder Benutzer muss sich in mindestens einer Gruppe befinden, kann aber Mitglied von beliebig vielen Gruppen sein. Welchen Gruppen der aktuelle Benutzer zugeordnet ist, können wir mit dem folgenden Kommando anzeigen:

```
> groups
```

Die Ausgabe enthält eine Liste von Gruppen. Dabei ist die erste angezeigte Gruppe die Standardgruppe (default group) des Benutzers. Diese Standardgruppe gibt an, welcher Gruppe neu erstellte Dateien des Benutzers standardmäßig zugeordnet werden.

Auf Systemebene sind Gruppen und Benutzer durch Zahlen codiert. Diese heißen Group ID (GID) und User ID (UID).

Der Superuser (root) hat erweiterte Rechte zur Systemadministration. Er ist auf jedem Linux-System vorhanden, auch wenn wir kein Passwort für diesen Benutzer gesetzt haben.

8.1 Wo sind Benutzer und Gruppen definiert?

Wir lernen nun eine der schönsten Eigenschaften eines offenen Linux-Desktop-Systems kennen: den ungehinderten und direkten Zugang zur internen Konfiguration des Systems.

Gruppen und Benutzer sind in zwei speziellen Dateien im Verzeichnis `/etc` festgelegt. Dieses Verzeichnis enthält systemweite Konfigurationen, von denen wir im Verlauf dieses Kurses noch einige kennenlernen werden.

Aufgabe: Lasst euch den Inhalt von `/etc/passwd` auf die Kommandozeile ausgeben.

Diese Datei enthält eine Liste aller im System angelegten User. Der Name ist leicht irreführend, denn heutzutage werden die Passwörter nicht mehr in dieser Datei gespeichert. Sie enthält zwar noch ein Feld für das Passwort, doch hier steht immer ein `x`.

Wir sehen, dass außer unserem eigenen Benutzer und dem root-Benutzer noch viele weitere User existieren. Häufig wird für Systemdienste jeweils ein eigener Benutzer angelegt, damit dieser Dienst mit unprivilegierten Rechten laufen kann. Dies ist ein Sicherheitsmechanismus – falls der Dienst eine Sicherheitslücke aufweist und von außen angegriffen wird, kann die Angreiferin nicht automatisch auf das restliche System zugreifen, sondern erhält nur die Rechte dieses speziellen Benutzers.

In der `passwd`-Datei sind in jeder Zeile folgende Felder enthalten:

- Username
- x anstelle eines Passwortes
- User ID
- Group ID der Standardgruppe
- Beschreibung (hier habt ihr vielleicht euren Namen eingegeben)
- Home-Verzeichnis (meistens `/home/username`)
- Login-Shell (bei Benutzern für Systemdienste meistens `nologin`)

Die Passwörter sind in einer weiteren Systemdatei gespeichert, nämlich in `/etc/shadow`. Wenn wir versuchen, uns als normaler User diese Datei ausgeben zu lassen, erhalten wir eine Fehlermeldung. Diese erklärt sich durch Betrachten der Berechtigungen:

```
> ls -l /etc/shadow
```

Wir sehen auch, dass die Datei dem Benutzer und der Gruppe `root` gehört. Das bedeutet, dass auch der folgende Befehl fehlschlagen wird:

```
> chmod 777 /etc/shadow
```

Obwohl in dieser Datei die Passwörter nicht im Klartext, sondern in Form eines Hashes stehen, sollte der Inhalt der `shadow`-Datei niemals weitergegeben werden.

Informationen über die Gruppen auf unserem System finden wir in `/etc/group`. Dort enthält jede Zeile folgende Felder:

- Gruppenname
- x statt des Gruppenpassworts
- Group ID
- Mitglieder

8.2 Der Superuser Root

Jedes Linux-System besitzt einen Root-Benutzer. Allerdings ist es gut möglich, dass für diesen Benutzer kein Passwort gesetzt ist.

Wenn wir ein separates Root-Passwort gesetzt haben, können wir uns als Superuser mit dem Root-Passwort authentifizieren:

```
> su -
```

Nach erfolgreicher Passworteingabe können wir uns mit dem Befehl `whoami` vergewissern, dass wir tatsächlich als Root angemeldet sind.

Bei vielen modernen Desktop-Linux-Distributionen wird standardmäßig kein Root-Passwort gesetzt. Dies ist eine Sicherheitsmaßnahme, um zu verhindern, dass nichttechnische User ein zu schwaches (leicht zu merkendes und leicht zu knackendes) Administratorpasswort setzen, oder dass sie dieses Passwort vergessen.

Stattdessen wird der Befehl `sudo` verwendet.

Sudo macht, was sein Name sagt: Er tut ein ganz bestimmtes Ding mit Superuser-Rechten. Üblicherweise verwenden wir `sudo` in Zusammenhang mit einem bestimmten Kommando, das mit privilegierten Rechten ausgeführt werden soll, zum Beispiel (bitte aufpassen, wer euch über die Schulter schaut):

```
> sudo cat /etc/shadow
```

Hier wird nach Bestätigung unseres eigenen Passworts ein einziger Befehl mit Root-Rechten ausgeführt. Dabei ist der Befehl `sudo` nicht automatisch jedem Systembenutzer zugänglich! Jeder User muss einzeln diese Berechtigung erhalten, was bei der Installation des Systems automatisch für euren Standardbenutzer eingerichtet wird.

Wir können dem `sudo`-Befehl eine Option mitgeben, um uns eine Root-Shell zu holen:

```
> sudo -s
> whoami
```

Wer kein Root-Passwort gesetzt hat, kann in der `shadow`-Datei in der Spalte für das Passwort ebenfalls ein `x` sehen. Um ein Root-Passwort zu setzen, kann in einer Root-Shell der folgende Befehl ausgeführt werden:

```
> passwd
```

Dies ist nur in Einzelfällen empfehlenswert. Root-Passwörter sollten immer besonders sicher sein (eine lange Kette von zufälligen Zeichen oder eine noch längere Kette zufälliger Wörter). Der Root-Benutzer kann mit `passwd` auch die Passwörter anderer Systemuser neu setzen, andere User können mit diesem Kommando ausschließlich ihr eigenes Passwort verändern.

Der Root-User darf mit vollen Administrationsrechten uneingeschränkt Änderungen am System vornehmen. Deshalb ist die Arbeit in einer Root-Shell immer vorsichtig und mit Bedacht durchzuführen. Es ist sehr leicht, sich durch ungeprüfte

Kommandos oder uninformiertes Ausprobieren das System zu zerschießen oder – schlimmer – wertvolle Benutzerdateien unwiederbringlich zu löschen.

Auch, wenn wir an dieser Stelle noch nichts vom Kernel und von der Hardware wissen, dennoch der Hinweis: Auch der Root-User agiert im Userspace. Er kann nicht direkt auf die Hardware zugreifen, zum Beispiel direkt auf einzelne Speicherblöcke des Datenträgers schreiben, einzelne Register auslesen oder der Soundkarte Daten zur Audio-Ausgabe schicken. Diese Funktionen laufen im Kernelspace ab, was eine zusätzliche Ebene der Sicherheit darstellt.

8.3 Die Grenzen des Berechtigungssystems

Systembenutzer und Dateiberechtigungen beziehen sich immer auf das laufende System. Im laufenden System kann ein User ohne `sudo`-Berechtigung und ohne Root-Passwort sich (abgesehen von eventuellen Sicherheitslücken) keinen Zugriff auf die Dateien anderer User oder auf die Systemkonfiguration verschaffen.

Eine Methode, dieses Berechtigungssystem zu umgehen, ist es somit, ein anderes System zu starten, auf dem wir vollen Zugriff haben. Wenn ich beispielsweise ein Laptop habe, dessen Systempasswörter ich nicht kenne, kann ich problemlos den Datenträger ausbauen und in meinem eigenen Computer stecken. Auf meinem eigenen System ist die Laptop-SSD nur ein weiterer Datenträger, auf den ich mit meinem eigenen Root-Zugang uneingeschränkt zugreifen kann.

Noch einfacher ist es, stattdessen ein Live-System zu starten. Auch dort kann ich sofort mit Root-Rechten auf alle eingebauten Datenträger zugreifen, ohne dass ich deren Zugangspasswörter kennen muss.

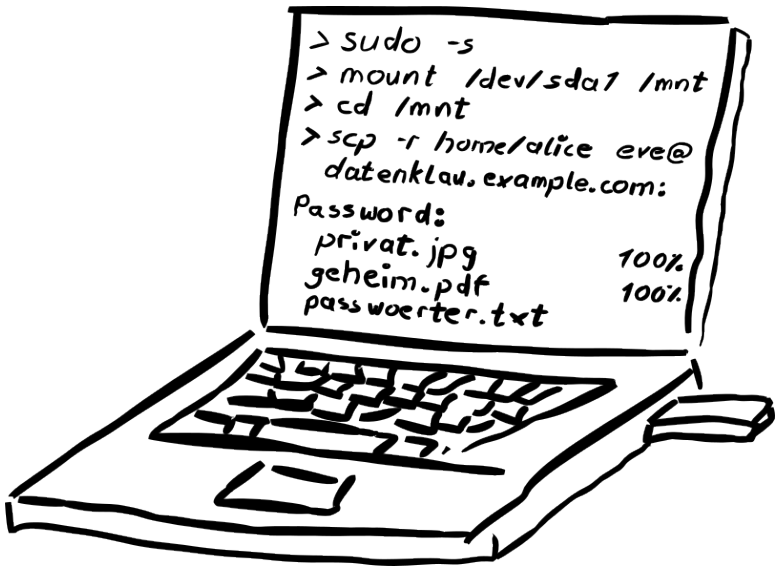

Abbildung 4: Wenn die Daten nicht verschlüsselt sind, reicht oft ein USB-Stick mit einem Live-System, um an alles dranzukommen.

Das bedeutet, dass dieses Berechtigungssystem unsere Daten nicht schützen kann, wenn der Angreifer physikalischem Zugriff auf das Gerät hat. Um diese Art von Schutz zu gewährleisten, müssen unsere Datenträger verschlüsselt werden. In modernen Smartphones ist dies standardmäßig der Fall. Viele Linux-

Distributionen bieten uns bei der Systeminstallation eine sichere Verschlüsselung durch das einfache Setzen eines Häkchens und eines Passworts.