ALICIA NOORS

LINUX BEFEHLSREFERENZ

SCHNELLEINSTIEG IN DIE ARBEIT MIT DER KONSOLE, REGULÄREN AUSDRÜCKEN UND SHELLSCRIPTING

VORWORT

Ohne Unix und Linux gäbe es kaum das Internet wie wir es heute kennen. Linux kommt in den letzten Jahren nicht nur auf Webservern zum Einsatz, sondern findet vermehrt Einzug in die IT-Landschaften vieler Betriebe und Organisationen. Dafür sind nicht zuletzt die Genügsamkeit was Hardware-Anforderungen angeht und die Tatsache, dass Linux und die meisten Serverdienste und Programme die darauf laufen kostenlos sind, verantwortlich...

Also müssen sich immer mehr Administratoren mit Linux beschäftigen und genau für sie habe ich dieses Buch geschrieben. Ein kleiner kompakter Helfer, der die wichtigsten Befehle, das Arbeiten mit regulären Ausdrücken und die Grundlagen der Shellprogrammierung erklärt.

Ziel dieses Buches ist es keinesfalls die Manpages (Hilfetexte zu Linux-Dateien und Befehlen) zu ersetzen. Vielmehr geht es darum dem Leser die gängisten Anwendungsfälle der diversen Befehle näher zu bringen und anhand praktischer Beispiele die einzelnen CLI-Befehle zu erläutern. Dazu werden die einzelnen Befehle thematisch Gruppiert und alphabetisch sortiert, um so ein schnelles Auffinden und Nachlesen der wichtigsten Anwendungsfälle zu ermöglichen.

Für tiefergehende Informationen können Sie die jeweiligen Manpages mit

```
man [befehl]
```

aufrufen.

Außerdem besitzen die meisten Befehle eine Kurzhilfe, die mit dem Schalter `-h` und/oder `--help` aufgerufen werden kann.

In diesem Sinne wünsche ich Ihnen einen reibungslosen Einstieg in das meiner Meinung nach beste Betriebsystem der Welt!

Ihre
Alicia Noors

INHALTSVERZEICHNIS

EINFÜHRUNG IN DIE KOMMANDOZEILE

Die Bash (Bourne-again Shell) ist eine freie Unix-Shell und Teil des GNU-Projekts. Sie ist heute auf vielen Unix- und Linux-Systemen die Standard-Shell. Neben der Bash gibt es noch weitere Shells wie `ksh` (Korn Shell), `csh` (C Shell), uvm.

Die Bourne-again Shell ist weitgehend kompatibel zur Bourne-Shell (`sh`) und beherrscht zusätzlich sowohl die meisten Funktionen der Kornshell (`ksh`) als auch Teile der C-Shell, wie zum Beispiel den Kommando-Verlauf, die `$RANDOM`-Variable und die POSIX-Form der Kommando-Substitution `$([kommando])`. Einige Elemente wie etwa die `**`-Wildcard wurden von der Z Shell (`zsh`) übernommen und Sie wurde um Funktionen wie zB Ganzzahlarithmetik oder eine Vereinfachung der I/O-Umleitungen erweitert.

Die Bash bietet mit ihrer Benutzer-Konfigurationsdatei (`~/.bashrc`) außerdem die Möglichkeit, jedem Benutzer eigene Einstellungen, wie zB eine individuelle Gestaltung des Prompts oder Alias-Namen für Befehle sitzungsübergreifend zu verwenden.

All das macht die Bash zur ersten Wahl als Shell. Jeder der öfter auf der Kommandozeile arbeitet, wird die Vorteile und die Komfort-Funktionen schnell zu schätzen wissen. Daher werden wir uns in diesem Buch auch ausschließlich mit der Bash und dem Bash-Scripting beschäftigen.

In Linux sind die grafische Oberfläche und das Betriebssystem von einander getrennt so ist es möglich und

auch üblich, Server gänzlich ohne GUI-Umgebung zu betreiben. Dies verkleinert die Anzahl der installierten Pakete enorm und damit auch die Angriffsfläche die Hackern zur Verfügung steht.

Ein solches System wird dann ausschließlich über die Kommandozeile verwaltet und betrieben.

Im Grunde könnte man meinen, dass die Arbeit mit der Kommandozeile nicht besonders komfortabel sei allerdings bietet die Bash hierzu einige Vereinfachungen.

Zuerst wäre da die Befehls-History zu nennen mit der man zuvor abgesetzte Befehle wieder aufrufen kann. Hierzu kann man mit der Pfeil aufwärts bzw. Pfeil abwärts Taste die zuletzt ausgeführten Befehle durchblättern.

Alternativ dazu kann man mit `Strg + r` nach einer Zeile im Befehlsverlauf suchen. Nach drücken dieser Tastenkombination verändert sich der Prompt zu `(reverse-i-search)`‘`: und wir können die ersten Zeichen des Befehls eingeben den wir suchen. Es wird dann zur ersten passenden Fundstelle gesprungen. Mit `Strg + r` können wir dann zur nächsten Fundstelle springen.

Will man einen laufenden Befehl abbrechen dann kann man dies in der Regel mit `Strg + c` erreichen. Alternativ dazu kann man mit `Strg + z` einen Befehl anhalten ohne ihn ganz abzubrechen. Danach kann man den Befehl mit `fg` oder `bg` wieder fortsetzen.

Ein häufiges Problem, dass Anfänger plagt, ist die Tastenkombination `Strg + s` ... diese speichert nicht etwa eine Eingabe ab, sondern friert das Terminal ein. Es wirkt also auf den ersten Blick so als wäre das Terminal abgestürzt doch mit der Tastenkombination `Strg + q` können

wir das Terminal wieder aufwecken. Speziell wenn man mit einem Editor arbeitet tappt man gern in diese Falle.

Ein weiterer Stolperstein für Windows-Umsteiger ist die Tatsache, dass Unix- und Linux-Systeme Case-sensitive sind - also zwischen Groß- und Kleinschreibung unterscheiden wird. Somit sind `Abc.txt`, `ABC.txt` und `abc.txt` drei verschiedene Dateien, die auch im gleichen Ordner liegen dürfen. Genauso verhält es sich mit den Optionen der Programme - hier wären `-R` und `-r` ebenfalls zwei verschiedene Optionen.

Oftmals kommt es vor, dass wir Programm-Ausgaben zwischenspeichern oder an ein anderes Programm weiterleiten müssen. Dazu stehen uns gleich zwei Werkzeuge zur Verfügung:

Pipes

... erlauben es die Ausgabe an ein Programm umzuleiten. Oftmals wird soetwas gemacht um lange Ausagaben in einen Pager (Dateibetrachter) zu leiten oder um Zeilen zu zählen.

```
user@acerian:~$ ps ax | wc -l
250
```

Mit dem `|`-Zeichen (Pipe) wird die Ausgabe von `ps ax` als Eingabe für `wc -l` verwendet. Damit zählen wir die Zeilen der Ausgabe. Eine so lange Ausgabe wie hier mit 250 Zeilen können wir auch an einen Dateibetrachter weiterleiten:

```
user@acerian:~$ ps ax | less
      PID TTY      STAT     TIME COMMAND
        1 ?        Ss       0:18 /sbin/init
        2 ?        S        0:00 [kthreadd]
        3 ?        S        0:31 [ksoftirqd/0]
    ... Ausgabe gekürzt
:
```

Umleitungen / Umlenkung / Ausgabeumlenkung

... erlauben es Ausgaben von Programmen in eine Datei umzuleiten oder eine Datei als Eingabe für ein Programm zu verwenden. Ein sehr gängiges Beispiel ist hier das Sichern und Rückspielen von MySQL-Datenbanken:

```
user@acerian:~$ mysqldump -u root dvwa > dvwa.sql
```

Mit `>` wird die Ausgabe des Befehls (in dem Fall der SQL-Code der Datanbank) in die Datei `dvwa.sql` geschreiben. Sollte diese Datei bereits existieren, dann wird die Datei überschrieben.

```
user@acerian:~$ mysql -u root dvwa < dvwa.sql
```

Beim wieder Einspielen einer Datenbank wird der Umleitungs-"Pfeil" einfach umgedreht. Das `<` zeigt auch grafisch deutlich an, wie der Fluss der Daten ist.

Natürlich gibt es auch Fälle in denen man die Datei nicht überschreiben möchte - so zB wenn man eine Log-Datei führen will. Hierzu gibt es die Umleitungs-Operatoren `>>` und `2>`:

```
bash cronjob.sh >> conjob.log 2> conjob.errlog
```

Mit `>>` werden die Ausgaben der Programme an die Datei `conjob.log` angehängt. An dieser Stelle sollte ich erwähnen, dass es unter Unix und Linux zwei Ausgabe-Kanäle gibt. Über `Stdout` laufen alle normalen Ausgaben einer Anwendung und `Stderr` liefert alle Fehlermeldungen. In der Regel landen beide Kanäle als Ausgabe am Bildschirm.

Durch `2>` lässt sich allerdings der `Stderr`-Kanal an eine separate Datei umleiten. Mit `2>&1` kann man `Stderr` auch in die gleiche Datei umleiten wie `Stdout`. zB `befehl > out.txt 2>&1` bedeutet die Ausgabe von `befehl` (`Stdout` und `Stderr`) nach `out.txt` umzuleiten.

Oftmals kommt es vor, dass ein Befehl länger dauert - typisch wäre hierfür ein aufwändigeres Script, ein Backup, das Packen oder Entpacken von Daten, etc. Wenn wir nicht wollen, dass das Terminal während der Ausführung eines Kommandos blockiert wird können wir mit `&` einen Befehl im Hintergrund starten:

```
user@acerian:~$ bash machwas.sh &
[1] 14996
user@acerian:~$ ls | wc -l
186
user@acerian:~$ Script beendet!
[1]+ Fertig bash machwas.sh
user@acerian:~$
```

Hier zeigt uns die Ausgabe von `[1] 14996` an, dass das Shellscript `machwas.sh` mit der PID (eindeutige Prozessnummer) `14996` im Hintergrund läuft. Das `ls | wc -l` zeigt gut, dass wir währenddessen das Terminal-Fenster weiter benutzen können und sobald der Befehl fertig ist wird die Ausgabe des Scripts (Script beendet!) angezeigt. Durch einen Tastendruck erhalten wir nochmals die Info von System, dass das Script beendet wurde.

!! - letzten Befehl nochmal ausführen

Eine weitere Möglichkeit den zuvor verwendeten Befehl erneut aufzurufen ist `!!`. Dies ist meist sinnvoll wenn wir einen Befehl versehentlich als normaler User abgesetzt haben der administrative Rechte benötigt. In so einem Fall können wir mit `sudo!!` den letzten Befehl als Administrator wiederholen und wir ersparen uns etwas Tipparbeit.

Vervollständigung

Mit Hilfe der Tab-Taste (Tabulator) lassen sich Datei- und Ordnernamen sowie Befehle vervollständigen. Sehen wir uns dazu ein Beispiel an:

```
user@acerian:~$ cd D[TAB]
user@acerian:~$ cd Do
```

Geben wir `cd D` ein und drücken `[TAB]` wird ein o ergänzt. Dies ist kein Fehler aber nach dem o kann die Bash nicht eindeutig feststellen welchen Ordner wir öffnen wollen.

```
user@acerian:~$ cd Do[TAB][TAB]
Dokumente/ Downloads/
```

Durch zweifaches drücken von `[TAB]` wird dann eine Liste der möglichen Optionen ausgegeben - hier zB `Dokumente/` und `Downloads/`

```
user@acerian:~$ cd Dow[TAB]
user@acerian:~$ cd Downloads/
```

Sobald wir nun ein w ergänzen wird eindeutig klar in welchen Ordner wir wechseln wollen und mit `[TAB]` kann der restliche Ordnername ergänzt werden.

Alternativen

Durch die Verwendung von `{}` kann einem Befehl eine Liste von Alternativen übergeben werden. zB `ls {*.pdf,*.docx,*.rtf,*.txt}`

Wichtig hierbei ist, dass die einzelnen Einträge mit einem, getrennt werden müssen und es keine Leerräume in der Liste geben darf!

ABSOLUTE VS. RELATIVE ADRESSIERUNG

Die relative Adressierung geht von dem Punkt im Dateisystem aus an dem wir uns aktuell befinden. Um dies zu demonstrieren verwende ich den Befehl `cd`.

```
user@acerian:~$ cd Downloads/
user@acerian:~/Downloads$
```

Im oberen Beispiel zeigt das `:~$` an, dass wir im Home-Ordner stehen und ein `cd Downloads/` bringt uns dann in den Unterordner Downloads.

In diesem zusammenmang gibt es noch zwei spezielle Ordner:

`./` ist der aktuelle Ordner
`../` ist der übergeordnete Ordner

So springen wir mit `cd ..` eine Ebene nach oben im Verzeichnisbaum.

```
user@acerian:~/Downloads$ cd /tmp/.font-unix/
user@acerian:/tmp/.font-unix$
```

Die absolute Adressierung geht vom Wurzel-Verzeichnis (/) aus und gibt immer den kompletten Pfad an.

So gut wie immer wenn es um Pfadangaben geht, können Sie wahlweise mit absoluter oder relativer Adressierung arbeiten. Eine der Ausnahmen ist hier zB die Benutzerverwaltung bei der Pfade zur Shell und zum Benutzer-Ordner absolut anzugeben sind.

ADMINISTRATOR (ROOT) VS. NORMALER USER

Unix- und Linux-Systeme haben einen Administrator-Account mit dem Namen `root`. Dieser Account hat in der Regel die UID (User-ID) `0` und GID (Gruppen-ID) `0`.

Befehle, die `root`-Rechte erfordern werde ich auch unter `root` ausführen um Sie auf diesen Umstand hinzuweisen.

Aus Sicherheitsgründen sollte man **nicht** mit dem `root`-Account dauerhaft am Rechner arbeiten und sich schon gar nicht in der grafischen Umgebung als `root` anmelden! Viele Systeme lassen dies auch gar nicht zu, ohne umfangreiche Konfigurationsänderungen am System.

Daher gibt es einige Wege sich in einem Terminal dauerhaft oder zeitweise `root`-Rechte zu verschaffen:

su (`switch user`)

Erlaubt es den User zu wechseln. Ohne Angabe eines Usernamens wechselt man zum Benutzer `root`. Hierbei kann mit der zusätzlichen Angabe von `-` ein vollständiger Login ausgeführt und somit alle Start-Dateien des Users `root` ausgeführt werden.

```
user@acerian:~$ su -
Passwort:
root@acerian:~# id
uid=0(root) gid=0(root) Gruppen=0(root)
root@acerian:~# su maxtest
maxtest@acerian:/root$ id
uid=1001(maxtest) gid=1001(fibu) Gruppen=1001(fibu)
```

Wie Sie anhand der Ausgabe von `id` sehen wechseln wir zwischen den unterschiedlichen Usern herum. Ein normaler User wird hierzu natürlich zur Angabe des Account-Passworts aufgefordert in den er wechseln möchte. `root` hingegen kann auch ohne Passwort sofort in jeden Account schlüpfen.

sudo (`superuser do`)

Kann dazu verwendet werden einen Befehl mit `root`-Rechten auszuführen. Nach der Eingabe des User-Passworts wird ein Befehl mit `root`-Rechten gestartet.

Natürlich kann nicht jeder User `sudo` verwenden. Die dazu berechtigten User werden in der Datei `/etc/sudoers` konfiguriert. Darüber hinaus ist es so möglich nur einen bestimmten Teil der `root`-Privilegien (zB nur die Userverwaltung) an einen bestimmten User zu deligieren.

```
user@acerian:~$ mkdir /opt/myapp
mkdir: das Verzeichnis „/opt/myapp“ kann nicht angelegt werden:
Keine Berechtigung
user@acerian:~$ sudo mkdir /opt/myapp
[sudo] Passwort für user:
user@acerian:~$
```

Die Verwaltung erfolgt mit:

```
root@acerian:~# visudo -f /etc/sudoers
```

In dieser Datei können einzelne Befehle oder Befehlsgruppen (über einen Aliasnamen zusammengefasst) einem User bzw. einer Gruppe zugewiesen werden. Hierzu ein Beispiel:

```
Cmd_Alias NETWORK = /sbin/ip, /sbin/ifdown, /sbin/ifup %admin
ALL=(ALL) ALL
%netadmin ALL=NETWORK
```

Hier wird zuerst der Alias `NETWORK` mit den drei Befehlen `ip`, `ifdown` und `ifup` definiert.

Danach werden den Mitgliedern der Linux-Gruppe `admin` alle `root`-Rechte zugestanden und den Mitgliedern der Gruppe `netadmin` wird das Ausführen der drei zuvor definierten Kommandos als `root` erlaubt.