2.1 Grundlagen der Befehlszeile – Lektion 1

Zertifikat:	Linux Essentials
Version:	1.6
Thema:	2 Sich auf einem Linux-System zurechtfinden
Lernziel:	2.1 Grundlagen der Befehlszeile
Lektion:	1 von 2

Einführung

Moderne Linux-Distributionen stellen eine Vielzahl grafischer Benutzeroberflächen bereit, aber ein Administrator muss stets wissen, wie man mit der **Befehlszeile, auch *Shell* genannt,** arbeitet. Die Shell ist ein Programm, das eine **textbasierte Kommunikation zwischen dem Betriebssystem und dem Benutzer** ermöglicht.

Es gibt **verschiedene Shells** unter Linux, hier nur einige wenige:

- Bourne-again shell (Bash)
- C shell (csh oder tcsh, eine erweiterte csh)
- Korn shell (ksh)
- Z shell (zsh)

Die **meistgenutzte Shell** unter Linux ist die **Bash**, die auch in den folgenden Beispielen und Übungen zum Einsatz kommt.

Bei der Arbeit mit einer interaktiven Shell gibt der Benutzer Befehle an der sogenannten ***Eingabeaufforderung*** oder ***Prompt*** ein. In jeder Linux-Distribution kann der Standard-Prompt etwas anders aussehen, folgt aber in der Regel dieser Struktur:

```
username@hostname aktuelles_verzeichnis shell_typ
```

Unter Ubuntu oder Debian GNU/Linux wird die Eingabeaufforderung für einen **normalen Benutzer** wahrscheinlich so aussehen:

```
carol@mycomputer:~$
```

Der **Prompt des Superusers** sieht dann so aus:

```
root@mycomputer:~#
```

Unter **CentOS oder Red Hat** Linux sieht die Eingabeaufforderung für einen normalen Benutzer hingegen so aus:

```
[dave@mycomputer ~]$
```

Und der **Prompt des Superusers** (CentOS und RedHat) sieht so aus:

```
[root@mycomputer ~]#
```

Schauen wir uns die einzelnen Komponenten dieser Struktur an:

`username`
: Name des Benutzers, der die Shell ausführt

`hostname`
: Name des Hosts, auf dem die Shell läuft. Es gibt auch den Befehl `hostname`, mit dem man den Hostname eines Systems anzeigt oder ändert.

`aktuelles_verzeichnis`
: Das Verzeichnis, in dem sich die Shell gerade befindet. Die **Tilde (`~`)** bedeutet, dass sich die Shell im **Heimatverzeichnis des aktuellen Benutzers** befindet.

`shell_typ`
: `$` zeigt an, dass die Shell von einem **normalen Benutzer** ausgeführt wird.

  `#` zeigt an, dass die Shell vom **Superuser `root`** ausgeführt wird.

Da wir keine besonderen Privilegien benötigen, werden wir in den folgenden Beispielen einen unprivilegierten Prompt verwenden; der besseren Übersicht halber verwenden wir einfach `$` als Prompt.

Aufbau der Befehlszeile

Die meisten Befehle auf der Befehlszeile folgen derselben Grundstruktur:

```
Befehl  [Option(en)/Parameter...]  [Argument(e)...]
```

Betrachten wir exemplarisch den folgenden Befehl:

```
$ ls -l /home
```

Zu den einzelnen Bestandteilen:

Befehl:
: Programm, das der Benutzer ausführt — im obigen Beispiel `ls`.

Option(en)/Parameter:
: Ein “**`Schalter`**”, der das Verhalten des Befehls in irgendeiner Weise verändert, z.B. `-l` im obigen Beispiel. Auf Optionen kann in **Kurz- oder Langform** zugegriffen werden, z.B. ist **`-l` identisch mit `--format=long`**.

  Mehrere **Optionen lassen sich kombinieren**, und für die Kurzform können die Buchstaben in der Regel zusammengeschrieben werden. Beispielsweise machen die folgenden Befehle alle dasselbe:

  ```
  $ ls -al
  $ ls -a -l
  $ ls --all --format=long
  ```

Argument(e)
: Zusätzliche Angaben, die vom Programm benötigt werden, etwa Dateiname oder Pfad (`/home` im obigen Beispiel).

Der einzige obligatorische Teil dieser Struktur ist der Befehl selbst. Im Allgemeinen sind alle anderen Elemente optional, aber ein Programm kann die Angabe bestimmter Optionen, Parameter oder Argumente erfordern.

> **Note** Die meisten Befehle liefern einen Überblick über mögliche Optionen, wenn sie mit dem **Parameter** `--help` aufgerufen werden. Weitere Informationsquellen zu einzelnen Linux-Befehlen werden wir noch kennenlernen.

Befehlstypen

Die Shell unterstützt zwei Arten von Befehlen:

Interne Befehle (Builtins)

Diese sind Teil der Shell selbst und keine eigenständigen Programme. Es gibt etwa 30 solcher Befehle, deren Hauptzweck es ist, Aufgaben innerhalb der Shell auszuführen (z.B. `cd`, `set`, `export`).

Externe Befehle

Diese befinden sich in einzelnen Dateien. In der Regel sind es binäre Programme oder Skripte. Wird ein Befehl ausgeführt, der kein Builtin ist, sucht die Shell mit der **Variablen `PATH` nach einer ausführbaren Datei** mit dem Namen des Befehls.

Der Befehl `type` zeigt, welchen Typs ein bestimmter Befehl ist:

```
$ type echo
echo is a shell builtin
$ type man
man is /usr/bin/man
```

Quoting

Als Linux-Nutzer müssen **Sie Dateien oder Variablen** auf verschiedene Weise erstellen oder manipulieren, was bei der Arbeit mit kurzen Dateinamen und einzelnen Werten einfach ist. Es wird allerdings **komplizierter, wenn Leerzeichen, Sonderzeichen und Variablen im Spiel sind**. Shells bieten eine Funktion namens *Quoting*, die solche Daten mit verschiedenen Arten von Anführungszeichen kapselt (" ", ' ').

In Bash gibt es drei Arten von Anführungszeichen:

- **Doppelte Anführungszeichen**
- **Einfache Anführungszeichen**
- **Escape-Zeichen**

Beispielsweise verhalten sich die folgenden Befehle aufgrund von Quoting nicht in der gleichen Weise:

```
$ TWOWORDS="two words"
$ touch $TWOWORDS
$ ls -l
-rw-r--r-- 1 carol carol     0 Mar 10 14:56 two
-rw-r--r-- 1 carol carol     0 Mar 10 14:56 words
$ touch "$TWOWORDS"
$ ls -l
-rw-r--r-- 1 carol carol     0 Mar 10 14:56  two
```

```
-rw-r--r-- 1 carol carol     0 Mar 10 14:58 'two words'
-rw-r--r-- 1 carol carol     0 Mar 10 14:56  words
$ touch '$TWOWORDS'
$ ls -l
-rw-r--r-- 1 carol carol     0 Mar 10 15:00 '$TWOWORDS'
-rw-r--r-- 1 carol carol     0 Mar 10 14:56  two
-rw-r--r-- 1 carol carol     0 Mar 10 14:58 'two words'
-rw-r--r-- 1 carol carol     0 Mar 10 14:56  words
```

Note Die Zeile mit `TWOWORDS=` **ist eine Bash-Variable**, die wir selbst erstellt haben. Wir werden die Variablen später einführen. Dies soll Ihnen nur zeigen, wie sich das Quoting auf die Ausgabe von Variablen auswirkt.

Doppelte Anführungszeichen

Doppelte Anführungszeichen weisen die Shell an, den Text zwischen den Anführungszeichen ("....") **als reguläre Zeichen zu übernehmen; alle Sonderzeichen verlieren ihre Bedeutung — mit Ausnahme von `$` (Dollarzeichen), `\` (Backslash) und `` ` `` (Backquote),** so dass Variablen, Befehlsersetzung und arithmetische Funktionen weiterhin verwendet werden können.

So wird beispielsweise die Ersetzung der Variablen `$USER` durch die doppelten Anführungszeichen nicht beeinflusst:

```
$ echo I am $USER
I am tom
$ echo "I am $USER"
I am tom
```

Ein Leerzeichen hingegen verliert seine Bedeutung als Argumententrenner:

```
$ touch new file
$ ls -l
-rw-rw-r-- 1 tom students 0 Oct 8 15:18 file
-rw-rw-r-- 1 tom students 0 Oct 8 15:18 new
$ touch "new file"
$ ls -l
-rw-rw-r-- 1 tom students 0 Oct 8 15:19 new file
```

Wie Sie sehen, erzeugt der Befehl `touch` im **ersten Beispiel zwei einzelne Dateien**; der Befehl interpretiert die beiden Zeichenketten als einzelne Argumente. Im **zweiten Beispiel** interpretiert der Befehl **beide Zeichenketten als *ein* Argument**, also nur eine Datei. Sie sollten allerdings Leerzeichen in Dateinamen vermeiden und stattdessen einen Unterstrich (`_`) oder einen Punkt (`.`) verwenden.

Einfache Anführungszeichen

Einfache Anführungszeichen **kennen keine Ausnahmen** wie die doppelten Anführungszeichen: Sie **widerrufen jede spezielle Bedeutung** für jedes Zeichen. Nehmen wir eines der Beispiele von oben:

```
$ echo I am $USER
I am tom
```

Bei der Verwendung der einfachen Anführungszeichen sehen Sie ein anderes Ergebnis:

```
$ echo 'I am $USER'
```

```
I am $USER
```

Der Befehl zeigt nun die Zeichenkette exakt an, ohne die Variable zu ersetzen.
Escape-Zeichen

Wir können *Escape-Zeichen* (Escape Characters) nutzen, um spezielle Bedeutungen von Zeichen aus der Bash zu entfernen. Zurück zur Umgebungsvariablen `$USER`:

```
$ echo $USER
carol
```

Wir sehen, dass standardmäßig der Inhalt der Variable im Terminal angezeigt wird. Wenn wir jedoch dem Dollar-Zeichen ein **Backslash-Zeichen (\) voranstellen**, wird die **besondere Bedeutung des Dollar-Zeichens aufgehoben**. Dies wiederum lässt Bash den Wert der ech:

```
$ echo \$USER
$USER
```

Wenn Sie sich erinnern, können wir mit einfachen Anführungszeichen ein ähnliches Ergebnis erzielen, da sie dafür sorgen, dass der Inhalt zwischen den Anführungszeichen zeichengenau ausgegeben wird. Das Escape-Zeichen funktioniert jedoch anders, indem es Bash anweist, jede spezielle Bedeutung, die das nachfolgende Zeichen haben könnte, zu ignorieren.

Geführte Übungen

1. Teilen Sie die folgenden Zeilen in die Bestandteile Befehl, Option(en)/Parameter und Argument(e) auf:
   - Beispiel: `cat -n /etc/passwd`

Befehl:	`cat`
Option:	`-n`
Argument:	`/etc/passwd`

   - `ls -l /etc`

Befehl:	ls
Option:	-l
Argument:	/etc

   - `ls -l -a`

Befehl:	
Option:	ls -l -a
Argument:	

   - `cd /home/user`

Befehl:	cd
Option:	
Argument:	/home/user

2. Bestimmen Sie den Befehlstyp:
   Beispiel:

`pwd`	Shell-Builtin
`mv`	Externer Befehl
`cd`	built in
`cat`	extern
`exit`	built in

3. Lösen Sie die folgenden Befehle mit Anführungszeichen auf:
   Beispiel:

`echo "$HOME is my home directory"`	`echo /home/user is my home directory`
`touch "$USER"`	Legt datei mit dem namen vom User
`touch 'touch'`	legt datei mit den namen touch an

Offene Übungen

1. Legen Sie mit einem Befehl und unter Verwendung der Klammer-Erweiterung (*Brace Expansion*) in der Bash (siehe Manpage der Bash) 5 von 1 bis 5 nummerierte Dateien mit dem Präfix `game` an (`game1`, `game2`,…).
   touch game{1,2,3,4,5}
2. Löschen Sie alle 5 Dateien, die Sie gerade mit nur einem Befehl erstellt haben, unter Verwendung eines anderen Sonderzeichens (siehe *Pathname Expansion* in den Man Pages der Bash).
   rm game{1,2,3,4,5}
3. Gibt es andere Möglichkeiten, zwei Befehle miteinander interagieren zu lassen? Welche sind das?
   mit ; oder &&

Zusammenfassung

In dieser Lektion haben Sie gelernt:

- Konzepte der Linux-Shell
- Was ist die Bash-Shell
- Die Struktur der Kommandozeile
- Eine Einführung ins Quoting

Befehle, die in den Übungen verwendet werden:

`bash:` Die beliebteste Shell auf Linux-Rechnern.

`echo:` Gibt Text im Terminal aus.

`Ls`: Listet den Inhalt eines Verzeichnisses auf.

`type`: Zeigt an, wie ein bestimmter Befehl ausgeführt wird.

`touch`: Erstellt eine leere Datei oder aktualisiert das Änderungsdatum einer bestehenden Datei.

`hostname`: Zeigt oder ändert den Hostnamen eines Systems.