3.3 Lektion 2

Zertifikat:	Linux Essentials
Version:	1.6
Thema:	3 Die Macht der Befehlszeile
Lernziel:	3.3 Von Befehlen zum Skript
Lektion:	2 von 2

Einführung

Im letzten Abschnitt haben wir dieses einfache Beispiel verwendet, um das Bash-Skripting zu demonstrieren:

```
#!/bin/bash

# A simple script to greet a single user.

if [ $# -eq 1 ]
then
  username=$1

  echo "Hello $username!"
else
  echo "Please enter only one argument."
fi
echo "Number of arguments: $#."
```

- Alle Skripte sollten mit einem *Shebang* beginnen, der den Pfad zum Interpreter definiert.
- Alle Skripte sollten Kommentare enthalten, um ihre Verwendung zu beschreiben.
- Dieses spezielle Skript arbeitet mit einem Argument, das beim Aufruf an das Skript übergeben wird.
- Dieses Skript enthält eine *if-Anweisung*, die die Bedingungen einer eingebauten Variablen `$#` testet. Diese Variable wird auf die Anzahl der Argumente gesetzt.
- Wenn die Anzahl der an das Skript übergebenen Argumente gleich 1 ist, dann wird der Wert des ersten Arguments an eine neue Variable namens `username` übergeben und das Skript gibt einen Gruß aus. Andernfalls erscheint eine Fehlermeldung.
- Schließlich gibt das Skript die Anzahl der Argumente aus. Dies ist für die Fehlersuche nützlich.

Dies ist ein nützliches Beispiel, um einige weitere Funktionen von Bash-Skripting zu erklären.

Exit Codes

Sie werden feststellen, dass unser Skript zwei mögliche Zustände hat: Entweder es druckt `"Hello <user>!"` oder es gibt eine Fehlermeldung aus, was für viele unserer Kern-Utilities ganz normal ist. Denken Sie an `cat`, mit dem Sie zweifellos sehr vertraut werden.

Lassen Sie uns eine erfolgreiche Verwendung von `cat` mit einer Situation vergleichen, in der es fehlschlägt. Erinnern wir uns, dass unser obiges Beispiel ein Skript namens `new_script.sh` ist.

```
$ cat -n new_script.sh

     1  #!/bin/bash
     2
     3  # A simple script to greet a single user.
     4
     5  if [ $# -eq 1 ]
     6  then
     7    username=$1
     8
     9    echo "Hello $username!"
    10  else
    11    echo "Please enter only one argument."
    12  fi
    13  echo "Number of arguments: $#."
```

Dieser Befehl ist erfolgreich, und Sie werden feststellen, dass das `-n` -Flag auch Zeilennummern gedruckt hat, die sehr hilfreich beim Debuggen von Skripten sind, aber bitte beachten Sie, dass sie *nicht* Teil des Skripts sind.

Jetzt werden wir den Wert einer neuen eingebauten Variablen `$?` überprüfen. Im Moment beachten Sie einfach die Ausgabe:

```
$ echo $?
0
```

Betrachten wir nun eine Situation, in der `cat` fehlschlägt. Zuerst erhalten wir eine Fehlermeldung und überprüfen dann den Wert von `$?`.

```
$ cat -n dummyfile.sh
cat: dummyfile.sh: No such file or directory
$ echo $?
1
```

Die Erklärung für dieses Verhalten ist folgende: Jede Ausführung des Dienstprogramms `cat` gibt einen *Exit Code* zurück. Ein Exit Code sagt uns, ob der Befehl erfolgreich war oder ein Fehler auftrat. Ein Exit Code von *null* zeigt an, dass der Befehl erfolgreich abgeschlossen wurde, was für fast jeden Linux-Befehl

gilt, mit dem Sie arbeiten. Jeder andere Exit Code weist auf einen Fehler hin. Der Exit Code des *letzten auszuführenden Befehls* wird in der Variablen `$?` gespeichert.

Exit Codes werden normalerweise nicht von menschlichen Benutzern gesehen, aber sie sind sehr nützlich beim Schreiben von Skripten. Stellen Sie sich ein Skript vor, bei dem wir Dateien auf ein entferntes Netzlaufwerk kopieren. Es gibt viele Möglichkeiten, wie die Kopieraufgabe fehlgeschlagen sein kann: Unser lokaler Rechner ist nicht mit dem Netzwerk verbunden oder das Remote-Laufwerk ist voll. Indem wir den Exit Code unseres Kopierprogramms überprüfen, können wir den Benutzer auf Probleme beim Ausführen des Skripts aufmerksam machen.

Es ist gute Praxis, Exit Codes zu implementieren, also werden wir das jetzt tun. Wir haben zwei Pfade in unserem Skript: Erfolg und Misserfolg. Wir nutzen *null*, um Erfolg anzuzeigen, und *eins* für den Misserfolg.

```
 1  #!/bin/bash
 2
 3  # A simple script to greet a single user.
 4
 5  if [ $# -eq 1 ]
 6  then
 7    username=$1
 8
 9    echo "Hello $username!"
10    exit 0
11  else
12    echo "Please enter only one argument."
13    exit 1
14  fi
15  echo "Number of arguments: $#."
```

```
$ ./new_script.sh Carol
Hello Carol!
$ echo $?
0
```

Beachten Sie, dass der Befehl `echo` in Zeile 15 vollständig ignoriert wurde und das Skript mit `exit` sofort beendet wird, so dass diese Zeile nie gefunden wird.

Behandlung mehrerer Argumente

Bisher kann unser Skript nur einen einzigen Benutzernamen auf einmal verarbeiten, eine beliebige Anzahl von Argumenten außer einem wirft einen Fehler. Wir wollen herausfinden, wie wir dieses Skript vielseitiger gestalten können.

Ein Benutzer könnte instinktiv dazu übergehen, mehr Positionsvariablen wie `$2`, `$3` usw. zu verwenden. Leider können wir aber die Anzahl der Argumente, die ein Benutzer benötigt, nicht vorhersehen. Um dieses Problem zu lösen, ist es hilfreich, mehr eingebaute Variablen einzuführen.

Wir werden die Logik unseres Skripts ändern: Null Argumente sollen einen Fehler werfen, aber eine beliebige Anzahl von Argumenten sollte erfolgreich sein. Dieses neue Skript wird `friendly2.sh` heißen.

```
 1  #!/bin/bash
 2
 3  # a friendly script to greet users
 4
 5  if [ $# -eq 0 ]
 6  then
 7    echo "Please enter at least one user to greet."
 8    exit 1
 9  else
10    echo "Hello $@!"
11    exit 0
12  fi
```

```
$ ./friendly2.sh Carol Dave Henry
Hello Carol Dave Henry!
```

Es gibt zwei eingebaute Variablen, die alle an das Skript übergebenen Argumente enthalten: `$@` und `$*`. Meist verhalten sich beide gleich, Bash wird die Argumente *parsen* und jedes Argument trennen, wenn es auf ein Leerzeichen stößt. Der Inhalt von `$@` sieht also so aus:

0	1	2
Carol	Dave	Henry

Wenn Sie mit anderen Programmiersprachen vertraut sind, erkennen Sie diese Art von Variable vielleicht als *Array*. Arrays sind in Bash einfach zu erstellen, indem Sie Leerzeichen zwischen Elementen wie der Variablen `FILES` im Skript `arraytest` unten setzen:

```
FILES="/usr/sbin/accept /usr/sbin/pwck /usr/sbin/chroot"
```

Es enthält eine Liste mit mehreren Elementen. Das ist bisher nicht sehr hilfreich, da wir noch keine Möglichkeit eingeführt haben, diese Elemente individuell zu behandeln.

For-Schleifen

Kommen wir noch einmal zu dem zuvor gezeigten Beispiel `arraytest`. Hier legen wir ein eigenes Array namens `FILES` an. Wir brauchen eine Methode, um diese Variable zu “entpacken” und nacheinander auf jeden einzelnen Wert zugreifen zu können. Dafür nutzem wir eine Struktur namens *for-Schleife*, die es in allen Programmiersprachen gibt. Es gibt zwei Variablen, auf die wir zugreifen werden: eine ist der Bereich, und die andere ist für den individuellen Wert, an dem wir gerade arbeiten. Dies ist das gesamt Skript:

```
#!/bin/bash

FILES="/usr/sbin/accept /usr/sbin/pwck /usr/sbin/chroot"

for file in $FILES
do
  ls -lh $file
```

```
done
```

```
$ ./arraytest
lrwxrwxrwx 1 root root 10 Apr 24 11:02 /usr/sbin/accept -> cupsaccept
-rwxr-xr-x 1 root root 54K Mar 22 14:32 /usr/sbin/pwck
-rwxr-xr-x 1 root root 43K Jan 14 07:17 /usr/sbin/chroot
```

Im obigen Beispiel `friendly2.sh` sehen Sie, dass wir mit einer Reihe von Werten arbeiten, die in einer einzigen Variablen `$@` enthalten sind. Der Übersichtlichkeit halber nennen wir die letztgenannte Variable `username`:

```
 1  #!/bin/bash
 2
 3  # a friendly script to greet users
 4
 5  if [ $# -eq 0 ]
 6  then
 7    echo "Please enter at least one user to greet."
 8    exit 1
 9  else
10    for username in $@
11    do
12      echo "Hello $username!"
13    done
14    exit 0
15  fi
```

Denken Sie daran, dass die Variable, die Sie hier definieren, beliebig benannt werden kann und dass alle Zeilen in `do… done` einmal für jedes Element des Arrays ausgeführt werden. Betrachten wir die Ausgabe unseres Skripts:

```
$ ./friendly2.sh Carol Dave Henry
Hello Carol!
Hello Dave!
Hello Henry!
```

Nehmen wir nun an, wir wollen unseren Output etwas menschlicher gestalten, indem wir unseren Gruß in eine Zeile setzen.

```
 1  #!/bin/bash
 2
 3  # a friendly script to greet users
 4
 5  if [ $# -eq 0 ]
 6  then
 7    echo "Please enter at least one user to greet."
 8    exit 1
 9  else
10    echo -n "Hello $1"
11    shift
12    for username in $@
13    do
```

```
13    do
14      echo -n ", and $username"
15    done
16    echo "!"
17    exit 0
18  fi
```

Einige Anmerkungen:

- Die Verwendung von `-n` mit `echo` unterdrückt den Zeilenumbruch nach der Ausgabe, d.h. alle Ausgaben werden auf dieselbe Zeile gedruckt, und der Zeilenumbruch folgt erst nach dem `!` in Zeile 16.
- Der Befehl `shift` entfernt das erste Element unseres Arrays. So wird aus:

0	1	2
Carol	Dave	Henry

das Folgende:

0	1
Dave	Henry

Betrachten wir die Ausgabe:

```
$ ./friendly2.sh Carol
Hello Carol!
$ ./friendly2.sh Carol Dave Henry
Hello Carol, and Dave, and Henry!
```

Reguläre Ausdrücke bei der Fehlerprüfung

Nehmen wir an, wir wollen alle Argumente, die der Benutzer eingibt, überprüfen, z.B. dass alle an `friendly2.sh` übergebenen Namen *nur Buchstaben* enthalten und alle Sonderzeichen oder Zahlen einen Fehler werfen. Für eine solche Fehlerprüfung nutzen wir `grep`.

Erinnern Sie sich daran, dass wir reguläre Ausdrücke mit `grep` verwenden können.

```
$ echo Animal | grep "^[A-Za-z]*$"
Animal
$ echo $?
0


$ echo 4n1ml | grep "^[A-Za-z]*$"
$ echo $?
1
```

Das `^` und das `$` kennzeichnen den Anfang und das Ende der Zeile, das `[A-Za-z]` kennzeichnet einen Bereich von Buchstaben, Groß- oder Kleinschreibung. Das `*` ist ein *Quantisierer* und modifiziert unseren Buchstabenbereich von null bis zu (beliebig) vielen Buchstaben. Zusammenfassend lässt sich sagen, dass unser `grep` erfolgreich ist, wenn die Eingabe *ausschließlich* Buchstaben umfasst — andernfalls schlägt es fehl.

Im nächsten Schritt müssen wir dafür sorgen, dass `grep` Exit Codes zurückgibt, abhängig davon, ob es eine Übereinstimmung gab oder nicht. Eine positive Übereinstimmung gibt `0` zurück, andernfalls wird der Wert `1` zurückgegeben. Wir können dies verwenden, um die Argumente in unserem Skript zu testen.

```
 1  #!/bin/bash
 2
 3  # a friendly script to greet users
 4
 5  if [ $# -eq 0 ]
 6  then
 7    echo "Please enter at least one user to greet."
 8    exit 1
 9  else
10    for username in $@
11    do
12      echo $username | grep "^[A-Za-z]*$" > /dev/null
13      if [ $? -eq 1 ]
14      then
15        echo "ERROR: Names must only contains letters."
16        exit 2
17      else
18        echo "Hello $username!"
19      fi
20    done
21    exit 0
22  fi
```

In Zeile 12 leiten wir die Standardausgabe an `/dev/null` um, was eine einfache Möglichkeit ist, sie zu unterdrücken, denn wir wollen keine Ausgabe des Befehls `grep` sehen, sondern nur seinen Exit Code testen, was in Zeile 13 geschieht. Beachten Sie auch, dass wir einen Exit Code `2` verwenden, um ein ungültiges Argument anzuzeigen. Es ist gute Praxis, verschiedene Exit Codes zu verwenden, um verschiedene Fehler anzuzeigen; so kann ein erfahrener Benutzer diese Exit Codes zur Fehlersuche nutzen.

```
$ ./friendly2.sh Carol Dave Henry
Hello Carol!
Hello Dave!
Hello Henry!
$ ./friendly2.sh 42 Carol Dave Henry
ERROR: Names must only contains letters.
$ echo $?
2
```

Geführte Übungen

1. Lesen Sie das folgende `script1.sh` :

```
#!/bin/bash

if [ $# -lt 1 ]
then
  echo "This script requires at least 1 argument."
  exit 1
fi

echo $1 | grep "^[A-Z]*$" > /dev/null
if [ $? -ne 0 ]
then
  echo "no cake for you!"
  exit 2
fi

echo "here's your cake!"
exit 0
```

   Wie lautet die Ausgabe der folgenden Befehle?

```
  ./script1.sh


  echo $?


  ./script1.sh cake


  echo $?


  ./script1.sh CAKE


  echo $?
```

2. Lesen Sie das folgende Skript `script2.sh` :

```
for filename in $1/*.txt
do
   cp $filename $filename.bak
done
```

   Beschreiben Sie den Zweck des Skripts so, wie Sie es verstehen.

Offene Übungen

1. Erstellen Sie ein Skript, das beliebig viele Argumente vom Benutzer entgegennimmt, und drucken Sie nur solche Argumente, die Zahlen größer als 10 sind.

Zusammenfassung

In diesem Abschnitt haben Sie gelernt:

- Was Exit Codes sind, was sie bedeuten und wie man sie implementiert.
- Wie man den Exit Code eines Befehls überprüft.
- Was for-Schleifen sind, und wie man sie mit Arrays verwendet.
- Wie man `grep`, reguläre Ausdrücke und Exit Codes verwendet, um Benutzereingaben in Skripten zu überprüfen.

Befehle, die in den Übungen verwendet werden:

`shift`
: Entfernt das erste Element eines Arrays.

Spezielle Variablen:

`$?`
: Enthält den Exit Code des zuletzt ausgeführten Befehls.

`$@`, `$*`
: Enthält alle Argumente, die an das Skript übergeben werden, als Array.