Oliver Kreipl, Dorian Karnbaum, Marc Remolt

Grundlagen der Shellskript-Programmierung

Ein Webmasters Press Lernbuch

Version 1.0.1 vom 23.6.2016

Autorisiertes Curriculum für das Webmasters Europe Ausbildungs- und Zertifizierungsprogramm.

www.webmasters-europe.org

Inhaltsverzeichnis

10 Schleifen

In dieser Lektion lernen Sie

- wie Sie denselben Code mehrfach ausführen können.
- was Schleifen sind.
- was die `while`-Schleife ist.
- was die `for`-Schleife ist.
- was Endlosschleifen sind.

10.1 Wiederholtes Ausführen von Codeblöcken

Vielleicht kennen Sie das Problem: Sie wollen im Prinzip denselben Code mehrfach ausführen, aber nicht den Code kopieren und mehrfach in dasselbe Skript schreiben.

10.2 Die while-Schleife

10.2.1 Syntax

Eine ***while*** -Schleife führt den Codeblock zwischen `do` und `done` so lange aus, wie die Bedingung hinter `while` wahr ist, also einen exit-Status von 0 zurückgibt. Hier werden fast immer Tests verwendet.

```
1 while Testbedingung
2 do
3   # Codeblock
4 done
```

Codebeispiel 10.1 *while_syntax.sh*

Sie müssen selbst dafür sorgen, dass die Bedingung irgendwann »falsch« zurückliefert. Meistens verändern Sie die Testumstände in dem Codeblock, der jedes Mal ausgeführt wird.

Beispiel

```
1 #!/bin/bash
2
3 zaehler=0
4
5 while (( zaehler < 10 ))
6 do
7   echo "Zahl: $zaehler"
8   (( zaehler++ ))
9 done
```

Codebeispiel 10.2 *while1.sh*

```
oliver@frodo:~$ bash while1.sh
Zahl: 0
Zahl: 1
Zahl: 2
Zahl: 3
Zahl: 4
Zahl: 5
Zahl: 6
Zahl: 7
Zahl: 8
Zahl: 9
```

Der Test sollte Ihnen inzwischen keine Schwierigkeiten mehr bereiten. Er prüft, ob die Shellvariable `$zaehler` kleiner als 10 ist. In Zeile 8 wird eben diese Variable in jedem Durchlauf um eins erhöht, weswegen die Schleife auch abbricht, sobald die Variable den Wert 10 erreicht.

Sorgen Sie dafür, dass die Testbedingung irgendwann scheitert, denn sonst läuft Ihre Schleife bis in alle Ewigkeit und Sie haben eine sogenannte Endlos-Schleife erzeugt. Es gibt seltene Fälle, wo Sie genau das brauchen, aber meistens ist es ein Fehler.

10.2.2 Eine while-Schleife ohne Abbruch-Modifikator

Sie können mit einer `while`-Schleife zum Beispiel auch auf das Eintreffen eines bestimmten Systemereignisses warten, zum Beispiel, wenn eine Datei gelöscht wird. Das folgende Skript läuft so lange, bis die Datei */tmp/test.txt* nicht mehr existiert, und gibt dann eine Meldung aus.

```
1 #!/bin/bash
2
3 while [ -f /tmp/test.txt ]
4 do
5   sleep 5
6 done
7
8 echo "Datei wurde gelöscht!"
```

Codebeispiel 10.3 *while2.sh*

Sie sehen richtig, der Codeblock der `while`-Schleife tut nichts! Besonders auffällig ist, dass es keinen Abbruch-Modifikator gibt. Da es von äußeren Umständen, hier von der Existenz einer Datei abhängt, ob die Bedingung wahr ist, muss im Code-Block auch keine künstliche Veränderung herbeigeführt werden, wie es im letzten Beispiel der Fall war.

Falls Sie das Kommando noch nicht kennen: `sleep` wartet die angegebene Anzahl an Sekunden und setzt erst dann das Skript fort. Alle 5 Sekunden wird also die Testbedingung geprüft, und sobald die Datei gelöscht ist, wird die Schleife abgebrochen und das `echo` auf Zeile 8 ausgegeben.

10.2.3 While zum zeilenweisen Einlesen einer Textdatei

Die `while`-Schleife in Kombination mit dem `read`-Kommando bietet eine sehr elegante Möglichkeit, Textdateien zeilenweise einzulesen. Da Sie diese Funktionalität ohnehin benötigen werden und das Beispiel außerdem noch eine interessante neue Schreibweise zeigt, will ich Ihnen diesen Code nicht vorenthalten.

Beispiel

```
1 #!/bin/bash
2
3 while read zeile
4 do
5   echo "Zeile: $zeile"
6 done < /etc/passwd
```

Codebeispiel 10.4 *while_read.sh*

Auf den ersten Blick sieht das Beispiel etwas verwirrend aus. Gehen wir also die einzelnen Punkte durch. Das Kommando `read` liest immer eine einzelne Benutzereingabe ein. Sobald der Benutzer `RETURN` drückt, merkt `read`, dass seine Aufgabe beendet ist.

Beispiel

```
oliver@frodo:~$ read zeile
Das ist eine Zeile
```

```
oliver@frodo:~$ echo $zeile
Das ist eine Zeile
```

Nach dem Wort `Zeile` hat der Benutzer `RETURN` gedrückt und `read` hat sich sofort beendet. Neu[2] für Sie ist nun, dass `read` eigentlich von der Standardeingabe liest. Statt von der Tastatur, kann `read` also auch aus einer Datei lesen, wenn Sie eine Umleitung `<` verwenden.

Was passiert nun, wenn Sie den folgenden Code ausführen? Denken Sie erst einmal darüber nach, bevor Sie es ausprobieren.

Beispiel

```
oliver@frodo:~$ read zeile < /etc/passwd

oliver@frodo:~$ echo $zeile
```

Als Ausgabe erhalten Sie nur die erste Zeile der Datei*/etc/passwd*, nicht ihren gesamten Inhalt. Da am Ende der ersten Zeile ein Zeilenumbruch steht (daher heißt es auch Zeile), beendet sich `read` schon an dieser Stelle, und der restliche Text geht verloren.

Wenn Sie dieses `read` in einer Schleife verwenden, wie in dem Beispiel gerade, wird `read` so oft aufgerufen, wie es Zeilen zum Abarbeiten gibt. Wenn `read` keine Zeile mehr erhält, legt es in `$?` einen Wert ungleich 0 ab, was zum sofortigen Ende der Schleife führt.

Die letzte Besonderheit an dieser Schleife ist, dass die Eingabeumleitung am Ende des Blocks, also nach dem `done`, steht.

```
...
```

```
done < /etc/passwd
```

2. Hoffentlich nicht wirklich neu.

Intuitiv hätten Sie die Umleitung eher hinter das `read` geschrieben, nicht wahr?

```
while read zeile < /etc/passwd

...
```

Das funktioniert aber leider nicht, da Sie die gesamte Datei an ein einziges `read` übergeben. Es wird also im ersten Durchlauf die erste Zeile an `read` übergeben, der Rest, wie eben erklärt, wird verworfen. Im zweiten Durchlauf wird an ein neues `read` wieder die gesamte Datei übergeben und wieder wird die erste Zeile verarbeitet. Das Spiel können Sie beliebig lange fortsetzen (außer, die Datei wird manuell gelöscht), denn Sie haben eine Endlosschleife gebaut, die auf ewig die erste Zeile von `/etc/passwd` ausgibt.

Wenn Sie die Umleitung nach dem `done` einfügen, wird der Inhalt der Datei nur ein einziges Mal an den Schleifenblock übergeben, und `read` pickt sich jedes Mal die nächste Zeile heraus. Wenn Sie jetzt Kopfschmerzen haben, sind Sie nicht alleine. Ich habe auch eine Weile gebraucht, bis ich das Prinzip hinter den wenigen Zeilen wirklich verstanden hatte. Aber es ist nun mal der empfohlene Weg, eine Textdatei zeilenweise abzuarbeiten. Also müssen Sie ihn kennen und wissen, was hier passiert.

10.3 Die for-Schleife

Eine ***for-Schleife*** auf der Shell arbeitet anders, als Sie es eventuell aus anderen Programmiersprachen gewohnt sind.[3] Sie durchläuft eine Liste von Werten und speichert den jeweils nächsten Wert in der angegebenen Variable. Die Liste ist eine Sammlung von beliebigen Werten, die durch Whitespace, also Leerzeichen, Tabulatoren oder Zeilenumbrüche getrennt sind.

Auch hier können Sie den Trenner durch Verändern der Variable `IFS` an Ihre Bedürfnisse anpassen.

10.3.1 Syntax

Die Syntax sieht folgendermaßen aus:

```
1 for variable in liste
2 do
3   # Codeblock
4 done
```

Codebeispiel 10.5 *for_syntax.sh*

Die Schleife durchläuft für jedes Element von `liste` die Anweisungen zwischen `do` und `done`. In jedem Schleifendurchlauf wird `variable` der nächste Wert aus `liste` zugewiesen. Als Variable gilt jeder gültige Variablenname.

Beispiel

```
1 #!/bin/bash
2
3 namen='Herbert Sandra Klaus Bettina Frank Claudia'
4
```

3. Sie könnte eher mit einer `foreach`-Schleife aus PHP verglichen werden.

```
5 for name in $namen
6 do
7   echo "Hallo $name"
8 done
```

Codebeispiel 10.6 *for1.sh*

```
oliver@frodo:~$ bash for1.sh
Hallo Herbert
Hallo Sandra
Hallo Klaus
Hallo Bettina
Hallo Frank
Hallo Claudia
```

Achten Sie darauf, dass Sie in Zeile 5 bei der ersten Variable einen Wert zuweisen, also kein `$` verwenden dürfen, und bei der zweiten auslesen, das `$` also brauchen.

10.3.2 Dynamisch erzeugte Listen

Die Liste nach `in` muss von Ihnen nicht unbedingt wörtlich geschrieben werden. Jedes Kommando, das eine derartige Liste erzeugt, ist hier erlaubt. Besonders häufig wird hier die Kommandosubstitution verwendet.

Beispiel

```
1 #!/bin/bash
2
3 for file in $(find /etc -type f 2> /dev/null)
4 do
5   if grep 'root' $file > /dev/null 2>&1
6 then
7   echo "Der Begriff 'root' wurde in $file gefunden."
8 fi
9 done
```

Codebeispiel 10.7 *for2.sh*

```
oliver@frodo:~$ bash for2.sh
Der Begriff 'root' wurde in /etc/skel/.bashrc gefunden.
Der Begriff 'root' wurde in /etc/bash.bashrc gefunden.
Der Begriff 'root' wurde in /etc/logrotate.d/dpkg gefunden.
Der Begriff 'root' wurde in /etc/logrotate.d/apache2 gefunden.
Der Begriff 'root' wurde in /etc/init.d/umountroot gefunden.
...
Der Begriff 'root' wurde in /etc/postfix/post-install gefunden.
Der Begriff 'root' wurde in /etc/postfix/postfix-script gefunden.
Der Begriff 'root' wurde in /etc/postfix/master.cf gefunden.
```

Der `find`-Befehl gibt alle Dateien (ohne die Verzeichnisse selbst) unterhalb von */etc* zurück, und diese Ausgabe wird als Liste an die `for`-Schleife übergeben. In jedem Durchlauf enthält `$file` also einen Dateinamen mit Pfad, z. B. */etc/passwd*.

In Zeile 5 wird in dem Codeblock der `for`-Schleife eine `if` -Anweisung ausgeführt. Dort wird mittels `grep` getestet, ob in der angegebenen Datei der Begriff `root` vorkommt. Wenn ja, wird eine entsprechende Erfolgsmeldung ausgegeben.

Es ist also problemlos möglich, Anweisungen in Shellskripten ineinander zu verschachteln. Sie können übrigens auch Schleifen wieder in Schleifen verschachteln, aber das erspare ich Ihnen vorerst.

Eine weitere Möglichkeit, Listen dynamisch zu erzeugen, sind die Shellexpansionen, die Sie in Lektion 5 »Shell-Expansion und Strings« kennengelernt haben.

Beispiel

```
#!/bin/bash

for zahl in {1..10}
do
  echo $zahl
done
```

Codebeispiel 10.8 *for3.sh*

```
oliver@frodo:~$ bash for3.sh
1
2
3
4
5
6
7
8
9
10
```

10.4 Zusammenfassung

Schleifen erlauben es Ihnen, denselben Codeblock mehrfach auszuführen, ohne ihn mehrfach hinschreiben zu müssen. Sie sind ein unverzichtbares Hilfsmittel, um Shellskripte kurz und übersichtlich zu halten.

In dieser Lektion haben Sie die beiden Schleifen `while` und `for` kennengelernt und einige Anwendungen ausprobiert.

Testen Sie Ihr Wissen

1. Wie lange wird eine `while`-Schleife ausgeführt?
2. Wie sorgen Sie dafür, dass die Bedingung einer `while`-Schleife irgendwann unwahr wird?
3. Wie oft wird eine `for`-Schleife ausgeführt?
4. Wodurch werden die Elemente der Liste getrennt? Wie können Sie das Verhalten verändern?
5. Wie können Sie die Ausgabe eines Kommandos als Liste für eine `for`-Schleife verwenden?