Bashscripting 102

Einführung in Bashscripten

von Marius Schwarz

für die (GNU) Linux UserGroup BraunSchweig

Stand September 2016

Bashscripting 102

Die Themen die wir heute Ansprechen werden, sind :

- Wie ist ein Script aufgebaut
- Was sind Dateiattribute
- Was sind Variablen

Bashscripting 102

Was ist so ein Bashscript eigentlich ?

„Ein BashScript ist eine als ausführbar markierte Textdatei, die eine Ansammlung von Befehlen enthält.“

Gut, wir merken uns, es ist ein Textfile und es muß als Ausführbar markiert sein.

Das bringt uns direkt zur nächsten Frage:

Was sind Dateiattribute ?

Bashscripting 102

Schauen wir uns mit dem Befehl „ls“ einfach mal eine Datei an :

```
[marius@eve ~]$ ls -la „tmp/convert2mp3.sh“ „AGNOSTIC FRONT - BLITZKRIEG BOP.mp3“
-rw-rw-r--. 1 marius marius 2107798  5. Sep 19:28 AGNOSTIC FRONT - BLITZKRIEG BOP.mp3
-rwx------. 1 marius marius     402 14. Jul 17:39 tmp/convert2mp3.sh
```

Gehen wir die Ausgabe von „ls“ durch:

Dateiattribute (Flags) : -rwxrwxrwx
Anzahl der Links: 1 (normal)
Username: marius
Gruppenname: marius
Dateigröße: In Bytes oder wenn -h in GB/MB/KB
Datum: Uhrzeit und Datum
Dateiname: mit Pfad oder ohne, je nach angegebenem „ls“ Argument

Bashscripting 102

Dateiattribute (Flags) : -rwxrwxrwx

Vorweg, es gibt noch mehr Attribute und mögliche Werte, hier soll es nur um die Grundlagen gehen.

Die Flags teilen sich in Gruppen () auf: (Filetype)(userrechte)(gruppenrechte)(weltrechte)

Wobei der Filetype „-“ ein normales File meint, „d“ ein Verzeichnis und „l“ einen Link.

Die Rechte (rwx) stehen für Read,Write,eXecute.

Wenn es Userrechte sind, gelten Sie für den Besitzer der Datei (im Beispiel „marius“), wenn es Gruppenrechte sind, gelten Sie für die angegebene Gruppe von Benutzern und wenn es Weltrechte sind, gelten Sie für alle, die nicht zu den beiden anderen Rechtegruppen gehören, also den Rest der Welt. Die Weltrechte werden im Englischen auch mit OTHERS bezeichnet, das wird später wichtig.

Bashscripting 102

Dateiattribute: Read,Write,eXecute

Die Regeln zu den Rechten lauten, beispielhaft für den Besitzer der Datei :

Habe ich Leserechte(Read), darf ich die Datei lesen, also mit einem anderen Programm öffnen und reinschauen.

Habe ich Schreibrechte(Write), darf ich in die Datei etwas schreiben, oder den vorhandenen Inhalt ändern.

Habe ich Ausführungsrechte(Execute), darf ich die Datei als Programm starten, unabhängig davon, ob es ein echtes Programm ist, eine MP3 Datei oder ein Textfile!

Über die Sinnhaftigkeit des Ausführens von MP3 Dateien kann man streiten, hier geht es nur um die Frage: Was darf ich !

Bashscripting 102

Wie man Dateiattribute setzt

Um selbst Dateiattribute zuzuweisen, benutzt man den Befehl „chmod“ :

Hier ein paar Beispiele:

chmod u+rwx dateiname Lesen,Schreiben,Ausführen für den Besitzer setzen
chmod g+rwx dateiname Lesen,Schreiben,Ausführen für die Gruppe setzen
chmod o+rwx dateiname (o=others) Lesen,Schreiben,Ausführen für Alle setzen
chmod u-x dateiname für den Besitzer **nicht** mehr ausführbar machen
chmod u+x dateiname für den Besitzer ausführbar machen
chmod g+x dateiname für die Gruppe ausführbar machen
chmod 755 dateiname für Besitzer RWX, die Gruppe RX und Alle RX setzen.
chmod 755 dateiname <=> chmod u+rwx g+rx o+rx dateiname

Die Attribute sind intern in BITS gespeichert und können auch mit Zahlen, statt Namen gesetzt oder gelöscht werden.

Bashscripting 102

Im Beispiel hatten wir bei einer Datei folgende Ausgabe :

```
-rwx------. 1 marius marius     402 14. Jul 17:39 tmp/convert2mp3.sh
```

Hier haben wir nur Rechte für den Besitzer, daß bedeutet, daß nur er überhaupt Zugriff hat, die Welt darf nichts, die Gruppe darf nichts.

An der Dateiendung „.sh“ sehen wir (angedeutet), daß es sich um ein Shellscript handelt. Shellscripte können heißen wie sie möchten, auch die Dateiendung kann beliebig sein.

Mit dem „file“ Befehl zur Identifikation einer Datei bekommen wir hier prompt die passende Aussage:

```
[marius@eve ~]$ file tmp/convert2mp3.sh
tmp/convert2mp3.sh: Bourne-Again shell script, ASCII text executable
```

Bashscripting 102

Wie wird jetzt aus einer Textdatei ein ShellScript ?

Dazu schauen wir uns mal die erste Zeile der Textdatei genauer an :

```
$ cat tmp/convert2mp3.sh
#!/bin/bash

FROM=$1

TO=`echo "$1" | sed -e "s/mp4$/mp2/g" -e "s/flv$/mp2/gi" -e "s/webm$/mp2/gi" -e "s/m4a$/mp2/gi"`

TO=$(basename -- "$TO")

rm -f "$TO"
ffmpeg -i "$FROM" -c:a mp3 -b:a 320k "$TO"
#ffmpeg -i "$FROM" -strict -2 -c:a:0 aac -b:a:0 192k "$TO.aac"

FROM=$TO
# TO=`echo "$FROM" | sed -e "s/Videos/Audio/g"`

TO=`echo "$TO" | sed -e "s/mp2$/mp3/g"`

lame -V 5 -b 64 -B 224 "$FROM" "$TO"
```

Bashscripting 102

Wie wird jetzt aus einer Textdatei ein ShellScript ?

„#!/bin/bash“ ist die sogenannte Shebang-Zeile[1], oder auch magische Zeile genannt.

Diese weist die ausführende Bashshell an, die Inhalte der Textdatei mit dem angegebenen Interpreter auszuführen. Klingt komisch, oder ?

„#!“ ist das Shebang, welches das nachfolgende Programm ausführt. Beispiele:

„#!/usr/bin/perl -w“ - Ausführung durch Perl Interpreter
„#!/usr/bin/php“ - Ausführung durch PHP Interpreter
„#!/bin/bash“ - Aufruf mit Bash

Wie man beim Perl Befehl sieht, kann man auch Argument zum Interpreter angeben. Das ist aber i.d.R. nicht nötig.

1) https://de.wikipedia.org/wiki/Shebang

Bashscripting 102

Beispiele:

```
#!/usr/bin/perl

use Sys::Hostname qw(hostname); # not strictly necessary; exports it by default
use IO::Socket::INET;
use File::Tail;

$SIG{INT} = \&tsktsk;

sub tsktsk {
      $output = readpipe("iptables -F sshguard");
      exit;
}
```

```
#!/usr/bin/php-cli
<?php

for($i=0;$i<256;$i++)
   {
   $c1=dechex($i);
   if(strlen($c1)==1){$c1="0".$c1;}
   $c1="=".$c1;
   $myqprinta[]=strtoupper($c1);
   $myqprintb[]=chr($i);
   }
```

Bashscripting 102

Was sind Variablen ?

Variablen sind einfach Platzhalter, denen man Zeichenfolgen zuweisen kann.

In Bash ist die Nutzung der Variablen etwas uneinheitlicher als z.B. in Perl oder PHP.

Die Namen von Variablen kann ich mir frei wählen, solange da keine reservierten Wörter oder Zeichen drin vorkommen:

FROM ist ok
FROM= ist nicht ok, weil = ein reserviertes Zeichen ist, denn mit = wird ein Wert zugewiesen:

FROM="Inhalt der Variablen FROM am Ende der Ausführung dieser Anweisung"

Wenn ich den Inhalt von FROM benutzen möchte, muß ich ein $ Zeichen davor stellen:

echo $FROM

Bashscripting 102

Was sind Variablen ?

Ein Beispiel:

```
[marius@eve ~]$ echo $FROM

[marius@eve ~]$ FROM="Jetzt ist etwas in FROM drin"
[marius@eve ~]$ echo $FROM
Jetzt ist etwas in FROM drin
[marius@eve ~]$
```

Merke: Variablen sind am Anfang immer leer!

```
[marius@eve ~]$ FROM=`ls -la /tmp`
[marius@eve ~]$ echo $FROM
```

Merke: Variablen können die Ausgaben eines Befehls (oder mehrerer) speichern.

So kann man sich Ergebnisse anderer Befehle merken und mehrfach verarbeiten

Bashscripting 102

Was sind Variablen ?

Ein Beispiel:

```
[marius@eve ~]$ FROM="Jetzt ist etwas in FROM drin"
[marius@eve ~]$ echo $FROM | grep -i Morgen
[marius@eve ~]$ echo $FROM | grep -i drin
Jetzt ist etwas in FROM drin
[marius@eve ~]$
```

Im Beispiel oben, sieht man, daß die FROM Variable mehrfach benutzt wird und Ihren Wert beibehält.

Es gibt für Variablen viele Einsatzmöglichkeiten, da man auch damit Rechnen kann. In diesem Fall nennt man das einen Zähler / Counter.

Bashscripting 102

Variablennamen

Der praktische Nutzen von Variablen ist natürlich, daß man durch den Namen auf den Inhalt schließen kann, weil man den Namen selbst vergibt. Dadurch wird das Programm u.a. auch deutlich übersichtlicher, als wenn man immer alle Befehle schreiben müßte, was man auf der vorherigen Seite sehen konnte.

Die Namen von Variablen in einem BashScript können Groß- und Kleinbuchstaben enthalten, man sollte sich selbst aber einen festen Stil angewöhnen, z.b. alle Variablennamen GROß schreiben. Diese Maßnahme erhöht die Lesbarkeit des Programms und vermeidet dadurch Fehler.

In Bash können Variablennamen auch fast beliebig lang sein, was aber nicht zur Übersichtlichkeit beiträgt.

Bashscripting 102

reservierte Variablen

Wenn man ein BashScript startet, braucht es üblicherweise ein paar Argumente, damit es sinnvolle Sachen machen, z.B. wie in unserem Beispiel das Umkonvertieren von Audio-dateien in MP3 Files. Da macht es Sinn, den zu konvertierenden Namen als Argument mit zugeben:

tmp/convert2mp3.sh filename

Damit man im Script dann diese Argument benutzen kann sind min. die folgenden Variablen schon beim Scriptstart vergeben:

$1 $2 $3 $4 $5 $6 $7 $8 $9 $0

Es gibt noch mehr, aber die spielen für uns heute keine Rolle.

Bashscripting 102

Was sind Variablen ?

In unserer Beispiel Datei sind gleich mehrere Variablen vorhanden:

```
$ cat tmp/convert2mp3.sh
#!/bin/bash

FROM=$1

TO=`echo "$1" | sed -e "s/mp4$/mp2/g" -e "s/flv$/mp2/gi" -e "s/webm$/mp2/gi" -e "s/m4a$/mp2/gi"`

TO=$(basename -- "$TO")

rm -f "$TO"
ffmpeg -i "$FROM" -c:a mp3 -b:a 320k "$TO"

FROM=$TO
TO=`echo "$TO" | sed -e "s/mp2$/mp3/g"`

lame -V 5 -b 64 -B 224 "$FROM" "$TO"
```

In Rot dabei die $1 Variable, also „vermutlich“ der Name der zu wandelnden Datei.

Bashscripting 102

Probleme bei Argumenten

Der Klassiker unter den „Dumm gelaufen“ Fehlern ist das Benutzen von Leerzeichen in Datei- oder Verzeichnisnamen, da diese als Argument einem Programm übergeben oft zu Fehlern führen.

mp3player Name der Datei.mp3

Wird nicht so funktionieren, wie das vielleicht geplant war. Damit es so funktioniert, muß man den Dateinamen, also das Argument in Anführungszeichen „“ setzen.

mp3player „Name der Datei.mp3“

Dann kann die Shell das Script korrekt aufrufen und das Programm wird korrekt ablaufen. An so etwas denkt nicht jeder, wenn er ein Script schreibt, deswegen malt es an jede Wand:

IMMER ANFÜHRUNGSZEICHEN BENUTZEN!

Bashscripting 102

Das Beispiel in Einzelschritten

```
FROM=$1
```

Die Variable FROM speichert den Inhalt des Arguments zum Befehl.

```
TO=`echo "$1" | sed -e "s/mp4$/mp2/g" -e "s/flv$/mp2/gi" -e "s/webm$/mp2/gi" -e "s/m4a$/mp2/gi"`
```

Um das Ergebnis eines Befehls zu speichern, muß bei der Zuweisung der Befehl in `` Zeichen eingebettet sein!

Echo gibt den Dateinamen aus. SED ersetzt in der Ausgabe von Echo, die Zeichenfolgen mp4, flv, webm und m4a durch „mp2“. Das „ersetzte“ Ergebnis wird in TO gespeichert.

Bashscripting 102

Das Beispiel in Einzelschritten

```
TO=$(basename -- "$TO")
```

Der Dateiname wird auf den Basisnamen gekürzt.

Beispiel:

Aus „/Musik/Bandname/Titel.mp3„ wird nur „Titel.mp3„

```
rm -f "$TO"
```

Der so ermittelte Zielname (TO) wird gelöscht. Gelöscht deshalb, weil sonst bei einer erneuten Ausführung des gleichen Scripts, die Datei schon vorhanden ist und der nachfolgende Befehl evtl. interaktiv fragt, ob er die Datei überschreiben soll. Um das zu vermeiden, wird die Datei einfach pauschal gelöscht.

Man beachte die Anführungszeichen, für den Fall, daß Leerzeichen im Dateinamen vorkommen.

Bashscripting 102

Das Beispiel in Einzelschritten

```
ffmpeg -i "$FROM" -c:a mp3 -b:a 320k "$TO"
```

Mit Hilfe von FFMPEG wird die Quelldatei (FROM) in ein MP3File (TO) umgewandelt.

```
FROM=$TO
TO=`echo "$TO" | sed -e "s/mp2$/mp3/g"`

lame -V 5 -b 64 -B 224 "$FROM" "$TO"
```

Nun wird mit FROM=$TO , der Name der Quelldatei geändert, nämlich auf die eben erstellte MP3Datei.

Nun wird im Namen noch „mp2“ mit „mp3“ ersetzt, damit die eigentliche Zieldatei den richtigen Namen hat. Die .mp2 Datei ist nur eine Zwischendatei im Umwandlungsprozess gewesen.

Mit Hilfe des besten Mp3Encoders „Lame“ wird das finale Mp3 erzeugt.

Bashscripting 102

Fazit

Man kann zwar auch in der Bashbefehlszeile mit Variablen arbeiten, aber so richtig Sinn macht das nur in Scripten.

Durch geschicktes Umwandeln des ursprünglichen (FROM) Dateinamens, dem Einsatz in zwei Variablen und ein paar Befehlen, konnte die Aufgabe, ein MP3 File aus verschiedenen Typen von Audio/Videodateien zu erstellen, so gelöst werden, daß sie immer automatisiert gelingt.

Ausblick auf die nächste Lektion:

Wir werden uns mit IF-THEN-ELSE Anweisungen beschäftigen