Linux, Knoppix, Mac OS X, Open Source: Vorteile von Unix et al. in Chemie & Biologie

Teil 12: Awk – Skalare, Arrays, Hashes

Röbbe Wünschiers

Ein wesentliches Merkmal einer jeden Programmiersprache sind Variablen. Sie können ganz einfache Notizbuch ähnliche Funktionen übernehmen, aber auch kompliziertere Aufgaben, z.B. die Rolle einer Zuordnungstabelle erfüllen. Schließlich können über Variablen auch Informationen der Arbeitsumgebung (Benutzer, aktuelles Verzeichnis, usw.) abgefragt und verarbeitet werden.

So nützlich wie Variablen in der Mathematik sind, um z.B. Gleichungen eine allgemeine Gültigkeit zu verleihen, so nützlich sind sie auch bei der Programmierung. In dieser Ausgabe möchte ich Ihnen die Verwendung von Variablen in Awk-Skripten vorstellen. Sie werden bald sehen, dass Variablen nicht gleich Variablen sind. Wir müssen zwischen Skalaren, Arrays und Hashes unterscheiden.

Skalare

Skalare sind die einfachste Form von Variablen. Sie bestehen aus einem Namen und einem zugeordneten Wert. Aus der Mathematik kennen wir Zuordnungen der Form x=3. Der Variablenname ist *x*, der Wert 3. Auf die selbe Art und Weise können in Awk Variablen definiert werden.

Terminal 1

```
01    $ awk 'BEGIN{x=3; print x}'
02    3
03    $ awk 'BEGIN{x=3; print x, y}'
04    3
05    $ awk 'BEGIN{x=3; y="clb";
       print x, y}'
06    3 clb
07    $
```

Tabelle 1: **Variablen Zuweisung** – Die folgenden Befehle dienen der Zuweisung von Zahlenwerten zu Variablen.

Zuweisung	Funktion
x = 5	Einfache Zuweisung (x=5)
x += 5	Addiert 5 zu dem Wert der Variablen *x* (x=x+5)
x -= 5	Subtrahiert 5 zu dem Wert der Variablen *x* (x=x-5)
x *= 5	Multipliziert den Wert der Variablen *x* mit 5 (x=x*5)
x /= 5	Dividiert den Wert der Variablen *x* durch 5 (x=x/5)
x ^= 5	Potenziert den Wert der Variablen *x* mit 5 (x=x^5)

Da die kurzen Skripte in Terminal 1 keine Eingabedatei verarbeiten, schreiben wir alle Kommandos in einen `BEGIN` Block (siehe CLB 08/04). In Zeile 1 weisen wir mit dem Befehl `x=3` der Variablen *x* den Wert 3 zu. Durch `print x` geben wir den Wert von *x* wieder aus. Das Beispiel in Zeile 3 zeigt, dass eine Variable der kein Wert zugewiesen wurde, leer ist. Für y wird nichts ausgegeben. Einer Variablen kann sowohl eine Zahl, wie auch ein Textstring zugewiesen werden. Im Gegensatz zu Zahlen, müssen Textstrings immer von Anführungszeichen umgeben werden. Variablennamen können fast beliebig lang sein, dürfen aber keine Sonderzeichen oder Leerzeichen enthalten. Zudem müssen sie mit einem Buchstaben beginnen. Es ist sehr sinnvoll Variablen einen Namen zu geben, der ihren Inhalt widerspiegelt. Bei größeren Programmen mit vielen Variablen verliert man sonst schnell die Übersicht.

Variablen Zuweisungen

Es gibt verschiedene Wege, wie Sie Variablen Werte zuweisen können (Tabelle 1). Den einfachsten Fall haben Sie in Terminal 1 kennen gelernt. Es gibt aber auch Zuweisungskommandos von der Art: addiere zum aktuellen Wert der Variablen *x* den Wert y.

Terminal 2

```
01    $ awk 'BEGIN{x=3; y=x; x+=5;
       y^=4; print x,y,x^y}'
02    8 81 1.41348e+73
03    $ awk 'BEGIN{x="clb";
       y="heft";
      z=x y; xy=x+y; print z, xy}'
04    clbheft
05    $ awk 'BEGIN{x=3; y=x; x+=5;
       xy=x y; print xy}'
06    83
07    $
```

Zeile 1 von Terminal 2 zeigt Ihnen einfache Beispiele für Wertezuordnungen. In Zeile 3 addieren wir zwei Variablen denen jeweils ein Textstring zugewiesen ist. Beachten Sie, dass Zeichenketten nicht durch die + Operation addiert werden. Diese bezieht sich ausschließlich auf Zahlen. Stattdessen werden die Variablen einfach durch ein Leerzeichen voneinander getrennt. Werden auf Variablenwerte Zahlenoperationen angewandt, dann werden sie auch als Zahlen bewertet.

Deshalb ist “hello“ + “world“ = 0. Umgekehrt gilt das Gleiche, weshalb 8 plus 3 gleich “83“ ist. Dies ist in Zeile 5 in Terminal 2 gezeigt. In der Regel sollte man versuchen, das Vermischen von Zahlen- und Textstrings zu vermeiden.

Integrierte Variablen

Mittels integrierter Variablen können Sie mit Awk, oder Awk mit Ihnen, kommunizieren (Tabelle 2). Die Bezeichnung integriert, oder *in-build*, meint also keinen Variablentyp, wie Skalar, Array oder Hash (siehe unten), sondern ist funktionell gemeint und unterscheidet sich von benutzerdefinierten Variablen. Die wichtigsten in Awk integrierten Variablen haben Sie bereits kennen gelernt. Die Variable *$0* enthält die aktuelle bearbeitete Zeile. Diese ist in Felder aufgeteilt, die in den Variablen *$1*, *$2*, usw. gespeichert sind (siehe Abbildung 2 in CLB 07/04). Diesen Variablen ordnet Awk automatisch einen Wert zu. Zu den integrierten Variablen gehören auch jene, dessen Wert von dem Benutzer verändert werden kann und mit denen Sie das Verhalten von Awk bestimmen können. Solchen Variablen sind wir in der letzten Ausgabe begegnet. Dort konnten wir durch Zuordnung von Werten zu den Variablen *RS (record separator)*, *FS (field separator)*, *ORS (output record separator)* und *OFS (output field seperator)* bestimmen, welche Trennzeichen Awk bei der Dateneingabe und -ausgabe verwenden soll (siehe Tabelle 2 in CLB 08/04).

Neben diesen, gibt es drei weitere wichtige integrierte Variablen, die uns Informationen über den Programmablauf liefern (Tabelle 2). Die Variable *FILENAME* enthält den Namen der zur Zeit eingelesen Datei. *NR (number of records)* gibt an, welche Zeile der Datei gerade bearbeitet wird und der Wert von *NF (number of fields)* entspricht der Anzahl der Felder in der aktuellen Zeile. Was als Zeile bzw. Feld erkannt wird, hängt wiederum von den Variablen *RS* und *FS* ab. Terminal 3 zeigt die Dynamik dieser Variablen.

Terminal 3

```
01    $ cat > test.txt
02    Dies ist Zeile 1.
03    Hier Nummer 2.
04    3
05    Die Letzte!
06    $ awk '{print FILENAME, NR,
       NF}' test.txt
07    test.txt 1 4
08    test.txt 2 3
09    test.txt 3 1
10    test.txt 4 2
11    $
```

Die Variable *NR* wird einfach durch gezählt, d.h., wenn Sie mehrere Dateien direkt hintereinander bearbeiten, dann ist es mit *NR* nicht möglich herauszufinden welche Zeile gerade bearbeitet wird.

Variable	Bedeutung	Standardeinstellung
FILENAME	Name der aktuell bearbeiteten Datei	
NR	Aktuell bearbeitete Zeilennummer (*number of record*)	
FNR	Aktuell bearbeitete Zeilennummer der aktuell bearbeiteten Datei (*file number of record*)	
NF	Anzahl der Felder (Wörter) der aktuell bearbeitenden Zeile (*number of fields*)	
$0	Aktuelle bearbeitete Zeile	
$1...$n	Felder der aktuellen Zeilen, gemäß den Einstellungen von *FS*	
RS	Zeilen-Trennzeichen bei der Eingabedatei (*record separator*)	Zeilenumbruch (\ n)
FS	Feld-Trennzeichen bei der Eingabedatei (*field separator*)	Leerzeichen oder Tabulator (\ t)
ORS	Zeilen-Trennzeichen bei der Datenausgabe (*output record separator*)	Zeilenumbruch (\ n)
OFS	Feld-Trennzeichen bei der Datenausgabe (*output field separator*)	Leerzeichen

Tabelle 2: **Integrierte Variablen** – Beschreibung der wichtigsten integrierten Variablen. Mit diesen Variablen können Sie das Verhalten von Awk bestimmen oder den Programmablauf kontrollieren.

Schauen wir uns ein Beispiel an. Dazu erstellen wir eine zweite Datei, indem wir die Datei *test.txt* kopieren und als *play.txt* abspeichern. Der entsprechende Befehl lautet: `cp test.txt play.txt`. Jetzt rufen wir, wie in Terminal 4 gezeigt, Awk mit beiden Dateien auf.

Terminal 4

```
01    $ awk '{print FILENAME, NR,
       NF}' test.txt play.txt
02    test.txt 1 4
03    test.txt 2 3
04    test.txt 3 1
05    test.txt 4 2
06    play.txt 5 4
07    play.txt 6 3
08    play.txt 7 1
09    play.txt 8 2
10    $
```

Um dieses Problem zu umgehen gibt es die Variable *FNR (file number of record)*. Im Gegensatz zu *NR*, wird *FNR* bei der Bearbeitung einer neuen Datei automatisch auf Null gestellt. Wir können die Variable *FNR* auch als Muster verwenden, um bestimmte Zeilen, oder einen Bereich von Zeilen auszuwählen, der bearbeitet werden soll. In Terminal 5 ist im ersten Awk-Skript die Dynamik der Variable *FNR*, zusammen mit *NR* und *NF*, gezeigt. Im zweiten Skript in Zeile 10 ist gezeigt, wie *FNR* dazu genutzt werden kann, nur die erste und dritte Zeile einer jeden Datei auszugeben. Wenn Sie sich an Teil 11 dieser Serie in 09/04 erinnern, dann handelt es sich um zwei durch *Oder* (`||`) logisch verknüpfte numerische Relationen (`FNR==1` bzw. `FNR==3`).

Terminal 5

```
$ awk '{print FILENAME, NR,
 FNR, NF}' test.txt play.txt
test.txt 1 1 4
test.txt 2 2 3
test.txt 3 3 1
test.txt 4 4 2
play.txt 5 1 4
play.txt 6 2 3
play.txt 7 3 1
play.txt 8 4 2
$ awk 'FNR==1||FNR==3
 {print $0}' test.txt
 play.txt
Dies ist Zeie 1.
3
Dies ist Zeie 1.
3
$
```

Alle Variablen die wir bis hierhin betrachtet haben sind skalare Variablen. D.h, einem Variablennamen ist genau ein Wert zugewiesen. Im folgenden werden wir Variablen kennen lernen, die mehrere Werte beinhalten können.

Arrays

Arrays sind Variablen die Listen beinhalten. Dies können Listen von Zahlen, Textstrings oder beides gemischt sein. Die einzelnen Listenelemente, oder Arrayelemente, werden über Indices angesprochen, die in eckigen Klammern hinter dem Variablennamen angegeben werden. Zeilen 1-3 in Terminal 6 sollen Ihnen einen Eindruck von der Zuordnung von Werten zu Arrayelementen geben. In Zeile 1 ordnen wir dem ersten Element (Index 0) des Arrays *temp* den Wert CLB zu und fragen dann den Wert des dritten Arrayelements (Index 2) ab. Da wir *temp[2]* keinen Wert zugewiesen haben, ist er natürlich leer. In Zeile 2 klappt es dann. Beachten Sie, dass der Index des ersten Arrayelementes 0 ist.

Terminal 6

```
$ awk 'BEGIN{temp[0]="CLB";
 print temp[2]}'
$ awk 'BEGIN{temp[2]="CLB";
 print temp[2]}'
CLB
$ awk 'BEGIN{
 liste="Eins-2-Drei";
 split(liste,temp,"-");
 for(item in temp){
 print "Item: "item
 " - Wert: "
 temp[item]}}'
Item: 1 - Wert: Eins
Item: 2 - Wert: 2
Item: 3 - Wert: Drei
$
```

```
BEGIN{
  liste="Eins-2-Drei"
  split(liste,temp,"-")
  for(item in temp){
   print "Item: "item" - Wert: "temp[item]
  }
}
```

Skript 1: *splitliste.awk* – Das ordentlich formatierte Skript aus Zeile 4 in Terminal 6.

Nun lassen Sie uns die Zeile 4 in Terminal 6 betrachten, die hier aus Platzgründen auf 7 Zeilen verteilt ist (siehe auch Skript 1). Bei einem Skript von dieser Länge lohnt es sich schon, es in einer Datei zu speichern und Awk mit dem Dateinamen aufzurufen. Der Datei geben wir den Namen *splitliste.awk*; der Programmaufruf sähe dann wie folgt aus: `awk -f splitliste.awk`. In Skript 1 sehen Sie das Skript aus Zeile 4 in Terminal 6. Die Zeileneinschübe machen es übersichtlicher. Zudem können wir jetzt auf die Semikolons verzichten, die bei der einzeiligen Version die Befehle voneinander getrennt haben. Aber nun zurück zur Funktion des Skriptes. Zunächst weisen wir der Variablen *liste* den Textstring „Eins-2-Drei" zu. Dann wenden wird den Befehl `split` an, um den Inhalt der Variablen *liste* anhand des Trennzeichens „-" in einzelne Elemente aufzutrennen, die in dem Array *temp* gespeichert werden. Dann verwenden wir eine so genannte Schleife, um alle Elemente des Arrays *temp* auf den Bildschirm auszugeben. Das Kommando `for(item in temp){...}` lässt sich lesen als: Kopiere den Index eines Elements des Arrays *temp* in der Variablen *item* und führe ... aus. Tue dies für alle Elemente des Arrays *temp*. Die Schleife wird also so oft durchlaufen, wie der Array Elemente hat. Der Befehl, den wir für jedes Arrayelement ausführen, lautet `print "Item: "item" - Wert: "temp[item]`. Wir geben also eine Mischung von Text ("Item: " bzw. " - Wert: ") und dem Inhalt der Variablen *item* und *temp[item]* aus. Sie sollten sich mit diesem Skript gut vertraut machen, da Arrays sehr hilfreiche Datenstrukturen sind.

```
BEGIN{
  split(ARGV[1]liste,temp,ARGV[2])
  for(item in temp){
   print "Item: "item" - Wert: "temp[item]
  }
}
```

Skript 2: *splitinput.awk* – Eine Erweiterung des Skriptes aus Skript 1, das Argumente aus der Kommandozeile aufnimmt.

Variable	Bedeutung
ARGC	Anzahl der Kommandozeilen Parameter
ARGV[n]	Array der Kommandozeilen Parameter; *ARGV[1]* bis *ARGV[ARGC-1]*

Tabelle 3: **Kommandozeilen Parameter** – Die Argumente der Kommandozeilen und deren Anzahl kann mit diesen beiden Variablen abgefragt werden.

Kommandozeilen Parameter

Awk verwendet einen Array um Kommandozeilen Parameter an ein Skript zu übergeben. Was bedeutet das? Wenn Sie einen Shellbefehl wie `cp` (kopieren) ausführen, dann geben Sie zusätzlich die Quell- und Zieldatei an. Das sind Kommandozeilen Parameter. Auf die selbe Art können Sie von einem Awk-Skript Kommandozeilen Parameter aufrufen. Wir könnten z.B. das Skript aus Terminal 6 flexibler gestalten, indem wir den Textstring und das Trennzeichen beim Aufruf des Skriptes angeben dürfen. Das Skript in Skript 2 erfüllt genau diese Funktion. Der Array, der die Kommandozeilen Parameter enthält heißt *ARGV* (Tabelle 3). Terminal 7 veranschaulicht die Funktion des Skriptes.

Terminal 7

```
$ awk -f splitinput.awk
 eins/zwei/drei /
Item: 1 - Wert: eins
Item: 2 - Wert: zwei
Item: 3 - Wert: drei
$
```

Beim Aufruf des Skriptes *splitinput.awk* geben Sie gleichzeitig, jeweils von Leerzeichen getrennt, den Textstring und das Trennzeichen an. Der erste Kommandozeilen Parameter, „eins/zwei/drei“ wird automatisch als zweites Element, und der zweite Kommandozeilen Parameter als drittes Element, in dem Array *ARGV* gespeichert. Finden Sie selbst heraus, was das erste Element, *ARGV[0]*, enthält.

Erinnern Sie sich noch, wie man aus einem Awk-Skript ein eigenständiges Programm generieren kann? Werfen Sie einen Blick in Teil 10 in CLB 08/04 und in Skript 3. Nachdem Sie das Skript mit dem Befehl `chmod u+x split.awk` ausführbar gemacht haben, können Sie es wie in Terminal 8 gezeigt aufrufen.

Terminal 8

```
$ ./split.awk a-b-c -
Item: 2 - Wert: b
Item: 3 - Wert: c
Item: 1 - Wert: a
$
```

Arrays sind also Listen, deren einzelnen Elemente über numerische Indices angesprochen werden können. Awk erlaubt uns aber auch Textstrings als Indices zu verwenden.

```
#!/usr/bin/awk -f
BEGIN{
  split(ARGV[1]liste,temp,ARGV[2])
  for(item in temp){
    print "Item: "item" - Wert: "temp[item]
  }
}
```

Skript 3: *split.awk* – Das Skript aus Skript 2 als selbständiger Befehl.

Hashes

Hashes, oder assoziative Arrays, sind eine Erweiterung des normalen Arrays. Hier darf der Index aus einem Textstring bestehen. Das macht Hashes vielseitig verwendbar. Bei assoziativen Arrays wird der Index meistens als Schlüssel (*key*) bezeichnet. Um die Funktion von Hashes kennen zu lernen, speichern Sie das folgende Skript als *translate.awk ab*.

Skript *translate.awk*

```
BEGIN{
  atom["C"]="Kohlenstoff"
  atom["H"]="Wasserstoff"
  atom["O"]="Sauerstoff"
  atom["H2O"]="Wasser"
  print atom[ARGV[1]]
}
```

In diesem Skript verwenden wir einen Hash namens *atom* als eine Zuordnungstabelle, die wie ein Wörterbuch genutzt werden kann. Die Ausführung ist in Terminal 9 gezeigt.

Terminal 9

```
$ awk -f translate.awk O
Sauerstoff
$ awk -f translate.awk H2O
Wasser
$
```

Hashes als Zuordnungstabelle sind natürlich nur eine von vielen möglichen Anwendungen. Ich denke, dass Ihnen das Beispiel einen guten Einblick in die Möglichkeiten gibt. Es gibt auch einen integrierten Hash, den ich Ihnen als letztes noch vorstellen möchte.

Tabelle 4: **Umgebungsvariablen** – Um den Wert von Shell-Variablen abzufragen, setzen Sie den Namen der Shell-Variablen als Schlüssel (Index) in dem Hash ENVIORN ein.

Variable	Bedeutung
ENVIRON["n"]	Hash zur Ausgabe der Umgebungsvariablen

Shell Variablen

Shell Variablen oder Umgebungsvariablen enthalten Informationen über den Computer an dem Sie arbeiten. Sehen wir uns einige Beispiele an.

Terminal 10

```
01    $ echo $SHELL
02    /bin/bash
03    $ echo $USER
04    rw
05    $ echo $PWD
06    /Users/rw/CLB/Unix-12
07    $
```

In den Zeilen 1, 3 und 5 in Terminal 10 geben wir den Wert einiger Umgebungsvariablen mit dem `echo` Befehl aus (siehe Teil 2 in CLB 12/03). Diese Variablen sind Teil der Shell in der wir arbeiten und werden daher auch als Shell-Variablen bezeichnet. Über einen Hash namens *ENVIRON* (Tabelle 4) können wir auf diese Variablen auch aus Awk zugreifen.

Terminal 11

```
01    $ awk 'BEGIN{print
       ENVIRON["SHELL"]}'
02    /bin/bash
03    $
```

Beachten Sie, dass das Dollarzeichen, welches beim Abruf der Variable in der Shell mittels `echo` vorangestellt werden muss, bei Awk nicht angegeben werden darf.

Ich hoffe, ich konnte Ihnen einen guten Überblick über Variablen wie Skalare, Arrays und Hashes verschaffen. Wir werden in den kommenden Ausgaben immer wieder mit Variablen arbeiten. Sie werden dann auch eine Reihe von Befehlen kennen lernen, die z.B. Arrays nach bestimmten Elementen durchsuchen oder sortieren. Für den Moment ist es nur wichtig, dass Sie das allgemeine Konzept verstehen und die Unterschiede zwischen Skalaren, Array und Hashes erkennen.

Neue Befehle in dieser Ausgabe

`split(string,array,sep)`	Teilt einen Textstring in einen Array
`for(var in array){...}`	Gibt alle Indices eines Arrays aus