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

Teil 15: Awk – Mathematische und textbezogene Befehle

Röbbe Wünschiers

In den vorausgegangenen fünf Teilen dieser Serie haben Sie alles wesentliche für den gezielten Umgang mit Awk gelernt. Nur eines habe ich Ihnen bislang unterschlagen: Awks Befehlsschatz. Eine Vielzahl vorhandener Befehle ermöglichen die Durchführung mathematischer Operationen oder das Editieren von Texten. Sollte Ihnen eine Funktion fehlen, dann ist es ohne weiteres möglich Awk mit eigenen Befehlen zu bereichern.

Mathematische Befehle

Über die Grundrechenarten hinaus stehen in Awk eine Reihe mathematischer, oder besser, numerischer Befehle zur Verfügung. Sie sind in der Tabelle 1 zusammengestellt. Ich denke es ist berechtigt zu fragen, weshalb ein exotischer Befehl wie der Arctangens, nicht aber der Tangens implementiert ist. Wahrscheinlich verlassen sich Alfred Aho, Peter Weinberger und Brian Kernighan, die Programmierer von Awk, auf die Mathematikkenntnisse der Anwender. Tabelle 2 gibt daher eine kleine Übersicht über indirekt vorhandene Funktionen. Die Anwendung der numerischen Funktionen in Awk-Skripten ist äußerst einfach. In Terminal 1 sind einige Beispiele gezeigt.

Terminal 1

```
01    $ awk 'BEGIN{print sin(90)}'
02    0.893997
03    $ awk 'BEGIN{print sin(90)*3}'
04    2.68199
05    $ awk 'BEGIN{print
      int(sin(90)*3)}'
06    2
07    $ awk 'BEGIN{print
      int(sin(90)*3)**8}'
08    256
09    $ awk 'BEGIN{print
      int(sin(90)*3)**8/4}'
10    64
11    $ awk 'BEGIN{print
      int(sin(90)*3)**8/4*rand()}'
12    15.2184
13    $
```

Weiter unten werde ich ein Beispiel vorstellen, wie Sie eigene Befehle definieren können. In Zeile 11 in Terminal 1 verwenden wir mit `rand()` einen Befehl zur Erzeugung einer Zufallszahl.

Zufallsgenerator

Der Zufallsgenerator von Awk bedarf einer gesonderten Betrachtung, insbesondere vor dem Hintergrund, dass es keine Zufallsgeneratoren gibt. Ein Zufallsgenerator erzeugt lediglich eine zufällig erscheinende Zahl, oder eine Folge von Zahlen, denen aber immer eine starre Formel zugrunde liegt. Im Gegensatz zu vielen anderen Programmiersprachen ist Awk in diesem Punkt sehr transparent. Schauen Sie sich das Beispiel in Terminal 2 an.

Terminal 2

```
01    $ awk 'BEGIN{for(i=1; i<5; i++){
      print rand()}}'
02    0.237788
03    0.291066
04    0.845814
05    0.152208
06    $ awk 'BEGIN{for(i=1; i<5; i++){
      print rand()}}'
07    0.237788
08    0.291066
09    0.845814
10    0.152208
11    $ awk 'BEGIN{for(i=1; i<5; i++){
      print rand()}}'
12    0.237788
13    0.291066
14    0.845814
15    0.152208
16    $
```

Dreimal hintereinander geben wir eine Reihe von vier Zufallszahlen aus. Es ist sicher kein Zufall, dass alle Ausgaben gleich sind. Awk stellt deshalb einen Befehl zur Initialisierung des Zufallsgenerators zur Verfügung: `srand(x)` *(seed random number generator)*. Dabei ist *x* der Startwert zur Berechnung einer Folge von Zufallszahlen.

Terminal 3

```
$ awk 'BEGIN{for(i=1; i<5; i++){
srand(); print rand()}}'
0.237788
0.237788
0.237788
0.237788
$ awk 'BEGIN{for(i=1; i<5; i++){
srand(); print rand()}}'
0.601451
0.601451
0.601451
0.601451
$
```

In Terminal 3 verwenden wir `srand()` ohne die Angabe einer Zahl. In diesem Fall wird die aktuelle Uhrzeit verwendet. Das Ergebnis ist aber immer noch nicht zufriedenstellend. Da das Durchlaufen der Schleife weniger als eine Sekunde beansprucht, ist das Ergebnis wiederum immer gleich. Erst das Platzieren der Initialisierung vor die Schleife liefert brauchbare Ergebnisse, wie in Terminal 4 zu sehen ist.

Terminal 4

```
$ awk 'BEGIN{srand();
for(i=1; i<5; i++) {
print rand()}}'
0.702238
0.259245
0.588448
0.428487
$ awk 'BEGIN{srand();
for(i=1; i<5; i++) {
print rand()}}'
0.413192
0.465976
0.00741673
0.367096
$
```

Zufallsgeneratoren spielen bei der Simulation beliebiger Prozesse, wie z.B. der Brownschen Molekularbewegung oder dem beliebten Würfel eine wichtige Rolle. In Terminal 5 werden fünf virtuelle Würfel geworfen. Beim Kniffeln könnte ich bereits mit einer kleinen Straße aufwarten...

Terminal 5

```
$ awk 'BEGIN{ORS=" "; srand();
for(i=1; i<=5; i++) {
print int(rand()*6+1)};
print "\n"}'
6 4 3 2 1
$
```

Das Skript *kniffel.awk* zählt die Anzahl der Würfe mit fünf Würfeln und stoppt, wenn alle Würfel die selben Augen zeigen. Versuchen Sie selbst, das Skript dahingehend zu verändern, die Chance von 6 Richtigen im Lotto zu ermitteln.

Skript kniffel.awk

```
BEGIN{
  ORS=" "; srand()
  do{
    for(i=1; i<=5; i++){
      w[i]=int(rand()*6+1)
      print w[i]
     }
    print " "a"ter Wurf\n"; a++
  }
  while (w[1]!=w[2] ||
         w[2]!=w[3] ||
         w[3]!=w[4] ||
         w[4]!=w[5])
}
```

Textbezogene Befehle

Die große Stärke von Awk liegt natürlich in seinen Befehlen zur Bearbeitung von Strings (Zeichenketten). In Tabelle 3 ist eine Übersicht über die wichtigsten Befehle gezeigt. Nachfolgend ist jede von ihnen anhand eines simplen Beispiels beschrieben.

`substr(string, start, länge)`
Der `substr`-Befehl liefert einen Teil des Strings *string* mit *länge* Zeichen von Zeichen *start* an. Der String selbst wird nicht verändert.

Terminal 6

```
$ awk 'BEGIN{s="abcdeabcd";
print substr(s,2,3),s}'
bcd abcdeabcd
$
```

In Zeile 1 in Terminal 6 extrahieren wir beginnend vom zweiten Zeichen des Strings *s* 3 Zeichen.

`gsub(regex, ersetzten, string)`
Der Befehl `gsub` (*global substitution*) ersetzt in dem String *string* alle Muster die auf den regulären Ausdruck *regex* passen durch den String *ersetzten*. Der Befehl gibt die Anzahl der Ersetzungen zurück.

Terminal 7

```
$ awk 'BEGIN{s="abcdeabcd";
print gsub(/b./,"-",s),s}'
2 a-dea-d
$
```

Das Skript in Terminal 7 ersetzt alle vorkommen des regulären Ausdrucks `/b./` ("b" und genau ein beliebiges Zeichen) in dem String *s* ("abcdeabcd") mit einem Minuszeichen. Es werden zwei Ersetzungen durchgeführt.

sub(regex, ersetzen, string)
Dieser Befehl gleicht dem vorangegangenen, allerdings wird nur der erste Treffer des regulären Ausdrucks ersetzt.

Terminal 8
```
01    $ awk 'BEGIN{s="abcdeabcd";
      print sub(/b./,"-",s),s}'
02    1 a-deabcd
03    $
```

Im Gegensatz zu dem Beispiel in Terminal 7 wird in Terminal 8 nur der erste Treffer des regulären Ausdrucks `/b./` in dem String "abcdeabcd" ersetzt.

gensub(regex, ersetzen, n, string)
Der Befehl `gensub` (*general substitution*) gleicht den vorangegangenen Befehlen `gsub` und `sub`, allerdings erlaubt er die exakte Angabe, dass nur der n-te Treffer oder aber alle Treffer ersetzt werden sollen. Im Gegensatz zu `gsub` und `sub` wird der ursprüngliche String nicht verändert, sondern der veränderte String zurückgegeben.

Terminal 9
```
01    $ awk 'BEGIN{s="abcdeabcd";
      print gensub(/./,"-",5,s),s}'
02    abcd-abcd abcdeabcd
03    $
```

In dem Beispiel in Terminal 9 wird in dem String *s* das 5-te Zeichen durch ein Minuszeichen ersetzt. Der reguläre Ausdruck `/./` trifft auf jedes beliebige Zeichen zu.

index(string, suche)
Dieser Befehl sucht den String *suche* in dem String *string* und gibt dessen Position (n-tes Zeichen von links) wieder.

Terminal 10
```
01    $ awk 'BEGIN{s="abcdeabcd";
      print index(s,"d"),s}'
02    4 abcdeabcd
03    $
```

Das Skript in Terminal 10 sucht nach dem ersten Vorkommen des Zeichens "d" in dem String *s* und gibt dessen Position (4) aus.

length(string)
Der Befehl `length` gibt die Länge des Strings *string* wieder.

Terminal 11
```
01    $ awk 'BEGIN{s="abcdeabcd";
      print length(s),s}'
02    9 abcdeabcd
03    $
```

Das Beispiel in Terminal 11 erklärt sich wohl von selbst.

match(string, regex, array)
Dieser Befehl liefert die erste Position, an welcher der reguläre Ausdruck *regex* zutrifft. Dieser Treffer wird in dem Arrayelement *array[0]* gespeichert. Die Arrayelemente 1 bis n enthalten eventuell in dem regulären Ausdruck enthaltene Zwischenspeicher (siehe CLB 04/2004, Terminal 6). Schauen wir uns das Beispiel in Terminal 12 an.

Terminal 12
```
01    $ awk ,BEGIN{s="abcdeabcd";
      print match(s,/d./, a),s;
      for (i in a) {
      print "a["i"]: ",a[i]}}'
02    4 abcdeabcd
03    a[0]:  de
04    $ awk ,BEGIN{s="abcdeabcd";
      print match(s,/d(.)/, a),s;
      for (i in a) {
      print "a["i"]: ",a[i]}}'
05    4 abcdeabcd
06    a[0]:  de
07    a[1]:  e
08    $
```

In Zeile 1 suchen wir in dem String *s* den regulären Ausdruck `/d./`, also ein "d" gefolgt von einem beliebigen Zeichen. Dieser reguläre Ausdruck verwendet keine Zwischenspeicher. Als Ergebnis erhalten wir in Zeile 2 die erste Position, an welcher der reguläre Ausdruck passt und den unveränderten String *s* und, in Zeile 3, den Treffer als erstes Element des Arrays *a*. Erinnern Sie sich? Das erste Arrayelement hat den Index 0 (siehe CLB 10/2004). Der reguläre Ausdruck im zweiten Skript in Zeile 4 verwendet einen Zwischenspeicher: das beliebige Zeichen nach dem "d" wird gespeichert (kenntlich durch die Klammern). Der Wert des Zwischenspeichers steht als zweites Element (Index 1) des Arrays *a* zur Verfügung. Es können auch mehrere Zwischenspeicher belegt werden, wie das folgende Beispiel zeigt.

Terminal 13
```
01    $ awk 'BEGIN{s="abcdeabcd";
      print match(s,/d((.).)/, a),s;
      for (i in a) {
      print "a["i"]: ",a[i]}}'
02    4 abcdeabcd
03    a[0]:  dea
04    a[1]:  ea
05    a[2]:  e
06    $
```

split(string, array, regex)
Den `split`-Befehl haben Sie bereits in Teil 12 (CLB 10/2004) kennen gelernt. Er trennt einen String an

jenen Stellen in Teilstrings auf, die durch einen regulären Ausdruck gekennzeichnet sind. Die Teilstrings werden in dem Array *array* abgelegt. Im Falle von *split* hat das erste Arrayelement den Index 1.

Terminal 14

```
01    $ awk 'BEGIN{s="abcdeabcd";
      print split(s,a,/c/),s;
      for (i in a) {
      print "a["i"]: ",a[i]}}'
02    3 abcdeabcd
03    a[1]:  ab
04    a[2]:  deab
05    a[3]:  d
06    $
```

Anstelle des regulären Ausdrucks kann auch ein String verwendet werden.

`asort(array, ziel)`
Der Befehl `asort` (*array sorting*) liefert die Anzahl der Elemente in Array *array*. Darüber hinaus kopiert er alle Elemente des Arrays *array* in den Array *ziel* und sortiert sie dabei alphabetisch nach ihren Werten.

Terminal 15

```
01    $ awk ,BEGIN{s="Die CLB ist
      super"; split(s,a," ");
      asort(a,z); for (i in z) {
      print "z["i"]: ",z[i]}}'
02    z[4]:  super
03    z[1]:  CLB
04    z[2]:  Die
05    z[3]:  ist
06    $
```

In Terminal 15 speichern wir die durch Leerzeichen getrennten Worte des Strings *s* in den Array *a* ab, der dann wiederum alphabetisch sortiert wird. Beachten Sie, dass die Ausgabe der Arrayelemente mithilfe der Array-orientierten `for`-Schleife leider nicht sortiert verläuft. Dies lässt sich aber umgehen, wenn wir eine ausführliche `for`-Schleife verwenden (siehe Teil 14 in CLB 12/2004).

Terminal 16

```
01    $ awk 'BEGIN{s="Die CLB ist super";
      split(s,a," "); n=asort(a,z);
      for (i=1; i<=n; i++) {
      print "z["i"]: ",z[i]}}'
02    z[1]:  CLB
03    z[2]:  Die
04    z[3]:  ist
05    z[4]:  super
06    $
```

Das Skript in Zeile 1 in Terminal 16 macht sich zunutze, dass der `asort` Befehl die Anzahl der Arrayelemente liefert. Diese speichern wir in der Variable *n* ab und lassen die Schleife entsprechend n-mal durchlaufen, um alle Elemente des sortierten Arrays auszulesen.

`tolower(string)`
Dieser Befehl konvertiert einen String in Kleinbuchstaben. Der original String wird nicht verändert, sondern das Ergebnis der Umwandlung zurückgegeben.

Terminal 17

```
01    $ awk 'BEGIN{s="Die CLB ist
      super"; u=tolower(s);
      print u" - "s}'
02    die clb ist super -
      Die CLB ist super
03    $
```

`toupper(string)`
Invers zum vorangegangenen Befehl, konvertiert `toupper` einen String in Großbuchstaben.

Terminal 18

```
01    $ awk 'BEGIN{s="Die CLB ist
      super"; u=toupper(s);
      print u" - "s}'
02    DIE CLB IST SUPER -
      Die CLB ist super
03    $
```

Sie haben jetzt eine Vielzahl von Befehlen kennen gelernt die Awk standardmäßig zur Verfügung stellt. Was ist aber, wenn Sie einen nicht vorhandenen Befehl, wie z.B. den Tangens oder die Berechnung des Logarithmus zur Basis 2 häufig benötigen? Dann können Sie sich ihre eigenen Befehle definieren.

Eigene Befehle

Das Definieren eigener Befehle ist denkbar einfach. Das entscheidende Kommando lautet `function`, gefolgt von dem Namen des neuen Befehls, den notwendigen Parametern und den entsprechenden Kommandos. Am besten schauen wir uns wieder ein einfaches Beispiel an.

Terminal 19

```
01    $ awk 'BEGIN{print tan(3)}
      function tan(x) {return
      sin(x)/cos(x)}'
02    -0.142547
03    $ awk 'function tan(x) {
      return sin(x)/cos(x)}
      BEGIN{print tan(3)}'
04    -0.142547
05    $
```

Unser selbst erstellter Befehl soll den Tangens einer Zahl in Radiant berechnen. Dazu definieren wir die

Funktion `tan` mit dem Befehl `function`. Unser neuer Befehl erwartet einen Eingabewert und muss entsprechend aufgerufen werden: `tan(x)`. Als Ergebnis liefert `tan` den Quotienten aus dem Sinus und dem Kosinus der Zahl *x*. Die Rückgabe des Ergebnisses initiiert das Kommando `return`. Wie das Beispiel in Terminal 19 zeigt, kann der Befehl vor oder nach dem eigentlichen Befehlsblock definiert werden. Selbstverständlich können auch Befehle definiert werden, die mehr als einen Eingabeparameter benötigen, wie das folgende Beispiel zeigt.

Terminal 20

```
01    $ awk 'BEGIN{print
      mylog(2,1024)} function
      mylog(b,n) {return
      log(n)/log(b)}'
02    10
03    $
```

In Terminal 20 definieren wir den Befehl `mylog`, der den Logarithmus von *n* zur Basis *b* ($\log_b n$) berechnet. Es werden also zwei Parameter benötigt: die zu logarithmierende Zahl *n* und die Basis *b*.

Damit ist die Serie über Awk abgeschlossen. Wie immer gilt: Übung macht den Meister. Falls die Beiträge dieser Serie Ihnen einmal nicht weiter helfen, dann können Sie mit dem Befehl `man awk`, `man gawk` oder `man nawk` die "Betriebsanleitung" zu Awk ansehen. Für alle, die noch tiefer in die Programmierung mit Awk einsteigen möchten, habe ich einige Bücher in die Literaturliste aufgenommen. Vielleicht konnten Sie ja mit Hilfe von Linux oder Awk ein anschauliches Problem aus Ihrer Arbeitswelt lösen, das wir in der Zukunft hier vorstellen können.

Neue Befehle in dieser Ausgabe

Zusätzlich zu den Befehlen in Tabellen 1 und 3:

`srand()`	Zufallsgenerator initialisieren
`function`	Definition eigener Befehle
`return`	initiiert Rückgabe eines Werter einer Funktion

Tabelle 3: Textbasierte Befehle

Funktion	Beschreibung
`index(s,p)`	Liefert die Position des Substrings *p* in String *s*
`length(s)`	Liefert die Länge des Strings *s*
`split(s,a,p)`	Spaltet String *s* an Treffern zu *p* und speichert das Ergebnis in Array *a*
`substr(s,p,m)`	Extrahiert aus dem String *s* ab Position *p* *m* Zeichen
`sub(r,p,s)`	Ersetzt das erste Vorkommen des Musters *r* im String *s* durch String *p*
`gsub(r,p,s)`	Ersetzt alle Vorkommen des Musters *r* durch String *p* in String *s*
`match(s,r,a)`	Liefert die erste Position auf die das Muster *r* im String *s* zutrifft; eventuelle vorkommende Zwischenspeicher werden im Array *a* gespeichert
`tolower(s)`	Liefert den String *s* in Kleinbuchstaben
`toupper(s)`	Liefert den String *s* in Großbuchstaben

Tabellen

Tabelle 1: Mathematische Befehle

Funktion	Bedeutung
`x+y`	x plus y
`x-y`	x minus y
`x*y`	x mal y
`x/y`	x dividiert durch y
`x**y`	x hoch y
`sqrt(x)`	Quadratwurzel aus x
`exp(x)`	e hoch x
`log(x)`	Natürlicher Logarithmus aus x
`sin(x)`	Der Sinus aus x in Radiant
`cos(x)`	Der Kosinus aus x in Radiant
`atan2(x,y)`	Arctangens aus x/y in Radiant
`int(x)`	Liefert den ganzzahligen Anteil von x
`rand()`	Liefert eine Zufallszahl zwischen 0 und 1

Tabelle 2: Abgeleitete numerische Funktionen

Funktion	Implementierung
tan(x)	sin(x) * cos(x)
$\log_x(y)$	$\log_i(y)/\log_i(x)$
$\mathrm{sqrt}_n(x)$	$x^{1/n}$
tan(x)	sin(x)/cos(x)
asin(x)	$\mathrm{atan2}(x, (1.-x^2)^{0.5})$
acos(x)	$\mathrm{atan2}((1.-x^2)^{0.5}, x)$

Literatur

[1] Herold, H (2003) awk & sed . Die Profitools zur Dateibearbeitung und -editierung. Verlag Addison-Wesley, ISBN 3-8273-2094-1

[2] Robbins, A (2001) Effective AWK Programming. O'Reilly Verlag, ISBN 0-596-00070-7

[3] Robbins, A (2002) sed & awk kurz & gut. O'Reilly Verlag, ISBN 3-89721-246-3

[4] Thesing, S (2004) SED & AWK GE-PACKT. Mitp-Verlag, ISBN 3-8266-1427-5

[5] Wünschiers, R (2004) Computational Biology: Unix/Linux, Data Processing and Programming. Springer Verlag, ISBN 3-540-21142-X