Einführung in AWK

Holger Jakobs (bibjah@bg.bib.de)

2003-03-31

Inhaltsverzeichnis

1 Vorstellung des Programms

(SVID2, XPG2, XPG3, POSIX.2) awk kann, wie grep, nach Ausdrücken in Dateien suchen. Aber awk kann noch viel mehr, z. B. Textdateien datenbankähnlich verwalten und sogenannte Reports generieren. Ganz allgemein kann über awk gesagt werden, dass es sich um ein Programm zur Bearbeitung von Dateien mit Zeilenstruktur handelt. **Tip:** Zu awk gibt es ganze Bücher.

awk ist im Standard-Lieferumfang aller bekannten UNIX-Varianten enthalten. Noch wesentlich leistungsfähiger als die „Standard“-awks ist der GNU-awk, der oft mit gawk aufgerufen werden muss. Bei Linux ist mit dem Namen awk direkt gawk verbunden. Auch für fast alle anderen Betriebssysteme gibt es eine (meist kostenlose) awk-Implementation, sogar für MS-DOS.

awk lehnt sich syntaktisch in einigen Dingen an die Programmiersprache C an, z. B. bei der Verwendung von geschweiften Klammern für Anweisungsblöcke. awk liest die angegebenen Dateien (oder von der Standardeingabe, wenn keine Dateien angegeben wurden) und führt bestimmte Aktionen mit den Datensätzen (= Zeilen) aus. Die Aufrufsyntax ist folgende:

```
awk [-Fc] -f Programmdatei [ Dateien ]
```
oder
```
awk [-Fc] 'Programm' [ Dateien ]
```

Bei der ersten Variante stehen die Anweisungen in einer Programmdatei, bei der zweiten Variante direkt in der Kommandozeile, sinnvoll für einfachere Aktionen oder bei der Verwendung in Shell Scripts. Die Aktionen von awk werden mit allen angegebenen Dateien ausgeführt und das Ergebnis erscheint auf der Standardausgabe, die natürlich umgelenkt werden kann.

Wird die Option -f nicht angegeben, wird der erste (Nicht-Options-) Parameter von awk als Programm angesehen. Die Option -F gefolgt von einem Zeichen legt dieses Zeichen als Feldtrennzeichen (field separator) fest. Normalerweise werden Felder durch einen Whitespace (Blank, Tab oder Kombinationen davon) getrennt.

2 awk-Anweisungen

Jede awk-Anweisung besteht aus einem Muster und einer Aktion, die für all die Zeilen ausgeführt wird, die auf das Muster passen. Die Aktion wird immer in geschweifte Klammern eingeschlossen und steht auf derselben Zeile neben dem Muster (genauer: beginnt zumindest in derselben Zeile). Passt eine Zeile auf die Muster mehrerer awk-Anweisungen, so werden alle zugehörigen Aktionen für diese Zeile der Reihe nach ausgeführt. Das heißt auch, dass die Reihenfolge aller Aktionen beliebig ist, sofern die Muster disjunkt sind. Ein Muster kann sein

- ein Suchmuster, d. h. eine Zeichenkette bzw. ein regulärer Ausdruck zwischen Slashes, z. B. würde /::/ auf alle Zeilen passen, in denen zwei Doppelpunkte hintereinander vorkommen; /^A.*x$/ dagegen auf alle Zeilen, die mit einem großen „A“ beginnen und mit einem „x“ enden.
- eine Bedingung, d. h. ein logischer Ausdruck. Z. B. wäre die Bedingung $0 ~ "^A" von allen Zeilen erfüllt, die mit einem großen „A“ beginnen (ginge natürlich auch mit dem Suchmuster /^A/.

Folgende Operatoren stehen zur Verfügung: ~ (enthält), !~ (enthält nicht), <, <=, == (gleich), != (ungleich), >=, >, in (ist Element von; siehe Abschnitt über Arrays). Es können auch mehrere Bedingungen mit && (und) und || (oder) verknüpft werden.

Es gibt noch zwei besondere „Muster“, BEGIN und END, deren zugehörige Aktionen ganz zu Beginn und ganz am Ende des awk-Programms genau einmal ausgeführt werden, z. B. für eine Kopfzeile und eine Fußzeile in einem Report.

Die Anweisungen müssen nicht am Anfang bzw. am Ende stehen; die Anordnung ist beliebig! Die `BEGIN`-Anweisung wird vor Verarbeitung der ersten Eingabezeile ausgeführt, die `END`-Anweisung nach Verarbeitung der letzten Eingabezeile.

Die in `{` und `}` eingeschlossene Aktion gibt an, was `awk` tun soll, wenn die Bedingung erfüllt ist. Wenn man keine Aktion angibt, wird die gerade bearbeitete Zeile ausgegeben, was man explizit auch durch `{print}` angeben kann (und der Klarheit wegen sollte).

Gibt man `print` Parameter mit, so werden nur diese ausgegeben. Es kann sich bei den Parametern um Variablen oder Literale handeln. Die Ausgabe eines `print`-Befehls kann in eine Datei umgeleitet werden (`>`), an eine Datei angehängt werden (`>>`) oder auch durch eine Pipe (`|`) mit einem anderen Programm verbunden werden. Werden die Parameter des `print`-Befehls nicht mit Kommas getrennt, schreibt `awk` sie einfach hintereinander. Bei Verwendung von Kommas werden sie mit dem (Ausgabe-)Feldtrennzeichen (normalerweise Blank) getrennt.

In einer Anweisung können mehrere Aktionen durch Semikolon getrennt enthalten sein. Kommentarzeilen im Programm werden mit einem # eingeleitet.

2.1 Variablen in awk

Die in Tabelle 1 aufgeführten Variablen stehen in `awk` immer zur Verfügung.

`NR`	aktuelle Zeilennummer (im gesamten Eingabestrom)
`FNR`	aktuelle Zeilennummer der aktuellen Eingabedatei
`FILENAME`	aktuelle Eingabedatei
`$0`	aktuelle Zeile
`NF`	Anzahl der Felder in der aktuellen Zeile
`$1` bis `$NF`	Felder der aktuellen Zeile
`FS`	Feldtrennzeichen (default: Whitespace)
`OFS`	Ausgabe-Feldtrennzeichen (default: Blank)
`RS`	Zeilentrennzeichen (default: lf)
`ORS`	Ausgabe-Zeilentrennzeichen (default: lf)
`ENVIRON["..."]`	Umgebungsvariablen, wenn in der Shell `LOGNAME=ANNA`, dann hat in `awk` `ENVIRON["LOGNAME"]` den Wert `ANNA`.

Tabelle 1: Variablen in AWK

Weitere Variablen können beliebig verwendet werden. Ihre Deklaration ist nicht erforderlich und auch nicht möglich. Die Verwendung einer Variablen legt sie an, sofern sie nicht schon existiert. Variablen können Skalar oder auch Arrays sein (siehe Abschnitt 4 auf der nächsten Seite). Es gibt keine Unterscheidung von Datentypen.

3 Funktionen und Operatoren in awk

awk bietet unter anderen die in Tabelle 2 aufgeführten Funktionen. Schlagen Sie ggf. weitere Funktionen in der Online-Hilfe nach (~>man awk)!

length (str)	Anzahl der Zeichen in str
int (num)	Ganzzahliger Anteil von num
index (s1, s2)	Position von s2 in s1 oder 0, wenn s2 nicht in s1 enthalten ist.
split(str, arr, del)	Splittet die Zeichenkette str in mehrere durch den Delimiter del getrennte Teile auf und speichert die Teile in Elementen des Array arr ab.
sprintf (fmt, args)	Formatiert args entsprechend dem in der fmt-Zeichenkette angegebenen Format. Rückgabe ist die sich ergebende Zeichenkette. Die Arbeitsweise entspricht der gleichnamigen C-Funktion.
substr (str, pos, len)	Gibt die Teilzeichenkette von str zurück, die an der Stelle pos beginnt und len Zeichen lang ist.

Tabelle 2: einige der Funktionen in AWK

Folgende Operatoren kennt awk: *, /, % (Divisionsrest), +, -, = (Zuweisung), ++, --, +=, -=, *=, /=

4 Assoziative Arrays

Die assoziativen Arrays gehören zu den leistungsfähigsten Eigenschaften von awk. Die Arrays können nicht nur über Zahlen indexiert werden, sondern auch über beliebige Zeichenketten, was natürlich dem üblichen Empfinden eines 3GL-Programmierers widerspricht. Indem man einem Array-Element einen Wert zuweist, legt man gleichzeitig auch die Array-Variable an. Es können beliebig viele Elemente in beliebiger Reihenfolge hinzugefügt werden; eine Dimensionierung gibt es nicht.

```
array[string]=value
```

Bei string kann es sich um eine feststehende Zeichenkette oder auch um eine beliebige Zeichenketten-Variable, einen Ausdruck oder auch ein Feld des aktuellen Datensatzes handeln. Um die Verarbeitung aller Elemente eines Arrays zu erleichtern, gibt es eine besondere for-Schleife.

```
for (element in array) { aktion }
```

Die Variable `element` nimmt alle Index-Werte, die im `array` vorkommen, der Reihe nach an und führt für jeden die `aktion` aus. Mit dem Operator `in` kann man testen, ob ein bestimmter Indexwert in einem Array enthalten ist oder nicht; mit `delete` kann man ein Array-Element löschen:

```
if (element in array) { aktion }
delete array [element]
```

5 Einfache awk-Beispielprogramme

```
~>awk '{ print }' dateiname
```

Kopiert sämtliche Zeilen der angegebenen Datei auf die Standardausgabe, denn „keine Bedingung“ heißt, dass die Aktion für alle Zeilen ausgeführt werden soll. So kann `awk` den Befehl `cat` ersetzen.

```
~>awk '/jenny/' dateiname
```

Gibt alle Zeilen der angegebenen Datei auf der Standardausgabe aus, die die Zeichenkette „`jenny`“ enthalten. Keine Aktion heißt, dass die aktuelle Zeile auf die Standardausgabe geschrieben werden soll. So ersetzt `awk` den `grep`.

```
~>awk '$1 ~ "h" { print $1, $2 }' dateiname
```

Gibt die ersten beiden Felder aller Zeilen aus, in denen im ersten Feld ein „`h`“ enthalten ist.

```
~>awk 'END { print NR, "Zeilen." }' dateiname
```

Gibt die Anzahl der Zeilen in der Datei aus, gefolgt von der Zeichenkette „`Zeilen`“.

5.1 awk-Programme in Dateien

Nun einige `awk`-Programme, die nicht auf der Kommandozeile mit eingegeben werden, sondern eigenständige Dateien sind:

- Anzeige eines Dateiinhaltes mit einer unterstrichenen Kopfzeile (Die Datei muss natürlich entsprechend formatiert sein, damit die Ausgabe schön ist.):

```
BEGIN {
  print "Artikel-Nr.    Bezeichnung   Anzahl   Preis"
  print "-----------------------------------------"
}
{ print }
```

- Aufteilung von Dateiinhalten abhängig vom Inhalt der Zeilen. Anwendungsbeispiel: Terminkalender mit 1 Zeile pro Termin, in dem die Namen aller Personen, die über den Termin informiert werden müssen, enthalten sind. Eine Zeile kann also für mehrere Personen Bedeutung haben und wird daher evtl. an mehrere verschickt.

```
BEGIN { print "Termin-Mail" }
/Mueller/ { print | "mail mueller" }
/Meier/   { print | "mail meier"   }
/Schmitz/ { print | "mail schmitz" }
END { print "Terminlisten versandt." }
```

- Ausgabe von Array-Inhalten sortiert. Da `awk` selbst nicht sortieren kann, bemühen wir das `sort`-Programm. Wir möchten die kumulierten Telefoneinheiten pro Person nach Anzahl dieser Einheiten absteigend sortiert haben. Alle Ausgaben, die wir in das `sort` hineinpipen, werden von diesem sortiert. Das Shell-Kommando muss in Anführungsstrichen stehen, damit `awk` es nicht für eine Variable hält. Natürlich könnte man das Kommando auch in eine Variable schreiben und diese dort nennen. Die Pipe zum `sort` wird mit dem Ende des `awk` geschlossen. Möchte man sie vorher schließen, z. B. um die Summe aller Einheiten unter der Liste auszugeben, so muss man explizit ein `close ("sort -n -k2")` angeben. Das Kommando muss genau so geschrieben werden wie vorher. (Eine passende Datendatei mit Telefondaten steht in `~bibjah/FORALL/AWK/telefon`).

```
$5 > 0  { Einheiten [$1] += $5; summe += $5}
END {
  for (name in Einheiten) {
    printf "\%-10s \%5d\n", name, Einheiten [name] | "sort -n -k2"
  }
  close ("sort -n -k2")      # Lassen Sie das probehalber mal weg!
  print "Summe der Einheiten: ", summe
}
```

- Anzeige der Namen aller Benutzer, die kein Password haben. Dieses `awk`-Programm `nopasswd` muss natürlich mit der Datei `/etc/passwd` (oder der Ausgabe von `ypcat passwd` aufgerufen werden.

```
BEGIN {
  print "Folgende User haben kein Password:"
  FS = ":"  # Felder in /etc/passwd werden durch : getrennt
}
$2 == "" { print $1 }
END { print "Ende." }
```

6 Interaktives Arbeiten mit awk

Üblicherweise ist awk nicht interaktiv. Aber da die Möglichkeit besteht, Shell-Kommandos auszuführen und deren Ergebnis „hereinzupipen“, kann man auch interaktive Programme schreiben. Grundsätzlich funktioniert das so (natürlich innerhalb einer awk-Aktion):

```
while ("ll" | getline zeile) { print zeile }
```

Hier wird das Shell-Kommando `ll` ausgeführt, das Ergebnis nach awk gepipet, wo es mit `getline` Zeile für Zeile in die Variable `zeile` gelesen und mittels `print` wieder ausgegeben wird. `getline` kann aus einer Pipe oder aus der aktuellen Eingabedatei lesen und das gelesene in `$0` (aktueller Eingabesatz, das ist default) oder in einer Variablen (wie hier in `zeile`) ablegen. Das Kommando kann natürlich nicht nur `ll` sein, sondern auch andere, z. B. ein `cat` oder ein `read` mit `echo`:

```
while ("cat" | getline) { print $1, $2 }
```

Die Eingabe kann über mehrere Zeilen gehen und wird mit `^D` (End-of-File) beendet, um `cat` zu beenden. Die Zeilen werden jeweils als Datensatz abgelegt.

```
"read x; echo $x" | getline; print $1, $2, $3
```

Es wird nur eine Zeile als Datensatz eingelesen.

7 „Normales“ Programmieren in awk

Im allgemeinen werden Eingabedateien in awk irgendwie verarbeitet. Das Arbeiten wie in klassischen Programmiersprachen, d. h. interaktiv ohne Datei-Ein- und -Ausgabe ist eigentlich nicht vorgesehen. Da es sich bei awk aber um eine vollständige Programmiersprache handelt, spricht nichts dagegen, einfache Programmieraufgaben mit awk zu lösen. Dazu schreibt man das Programm in die `BEGIN`-Anweisung.

```
~>awk 'BEGIN { gewünschte Anweisungen }'
```

8 awk-Scripts als ausführbare Dateien

Mit einer entsprechenden Notation kann man awk-Scripts genau wie Shell Scripts direkt ausführbar machen, so dass sie bei Benutzung wie compilierte Programme erscheinen. Dazu schreibt man in die Datei an den Anfang folgendes und macht die Datei mit einem passenden `chmod`-Kommando ausführbar:

```
#!/usr/bin/awk -f
```

Hinweis: Der Pfad zum awk-Interpreter kann von Maschine zu Maschine variieren und muss entsprechend angepasst werden. Wichtig ist, dass obige Zeile wirklich ganz am Anfang der Datei steht, die ersten beiden Bytes müssen `#!` lauten.

9 Wenn es für awk zu kompliziert wird

Nicht alle Aufgaben sind mit awk sinnvoll lösbar. Wird es komplexer,kann man zwar auch bei awk benutzerdefinierte Funktionen benutzen, aber richtige Unterprogramm-Module (Pakete, Bibliotheken) gibt es nicht. Auch geht es nicht mehr weiter, wenn man eine grafische Oberfläche gestalten möchte.

In solchen Fällen muss man auf eine „größere“ Sprache ausweichen. Hier bieten sich an: Perl, Python und Tcl als die beliebtesten plattformunabhängigen und leistungsfähigen Scriptsprachen. Eine Entscheidung zwischen diesen ist immer eine sehr persönliche, weil alle diese Sprachen quasi alles können. Bei der Gestaltung grafischer Oberflächen greifen alle auf Tk zurück, das eigentlich zu Tcl gehört. Aus diesem Grund ist die Integration von Tcl mit Tk auch die beste und einfachste.

Bezüglich Leistungsfähigkeit (z. B. bei der Ausführungsgeschwindigkeit und bei regulären Ausdrücken) oder der Verwaltung von Programmbibliotheken stehen sie sich in nichts nach. Perl ist bei der Web-Programmierung weiter verbreitet als die anderen, Tcl hat das sauberste Sprachkonzept und ist leicht lesbar, Python ist objektorieniert. Für jede diese Sprachen lassen sich also gute Argumente finden.

$RCSfile: awk.tex,v $ $Date: 2003/03/31 06:47:41 $ $Revision: 1.3 $