Introduction to awk

Matt Warner

Contents

1 What is awk?

- Created by:

  Aho, Weinberger and Kernighan.
- awk is a scripting language used for manipulating data and generating reports
- Versions of awk:

  awk, nawk, mawk, pgawk, ...
- GNU awk: gawk

2 What can you do with awk?

- awk operation:
  - reads a file line by line
  - splits each input line into fields
  - compares input line/fields to pattern performs action(s) on matched lines
- Useful for:
  - transforming data files
  - producing formatted reports
- Programming constructs
  - format output lines
  - arithmetic and string operations
  - conditionals and loops

3 Basic awk invocation

Syntax:

```
awk 'script' file(s)
```

```
awk -f scriptfile file(s)
```

Common option: -F (to change the field seperator)

4 Basic awk script

- consists of patterns & actions:

Syntax:

```
pattern {action}
```

- If pattern is missing, action is applied to all lines
- if action is missing, the matched line is printed
- Must have either pattern or action

Example:

```
# print all lines containing string "for" in testfile

awk '/for/ { print }' testfile
```

5 awk variables

awk reads input line into buffers ***record*** and ***fields***

- field buffer:
  - One for each field in the current record
  - Variable names: $1, $2, ...
- record buffer:
  - $0 holds the entire record

5.1 More variables

NR Number of the current record

NF Number of fields in the current record

Also:

FS Field sperator (default=whitespace)

Example:

Say we had a file called emps:

```
~$ cat emps
Tom Jones      4424      5/12/66     543354
Mary Adams     5346      11/4/63     28765
Sally Chang    1654      7/22/54     6500000
Billy Black    1683      9/23/44     336500

$ awk '/Tom/ { print $0 }' emps
Tom Jones      4424      5/12/66     543354

$ awk '{ print NR, $0 }' emps
1 Tom Jones      4424      5/12/66     543354
2 Mary Adams     5346      11/4/63     28765
3 Sally Chang    1654      7/22/54     6500000
4 Billy Black    1683      9/23/44     336500
```

Example: Space as Field Separator

```
~$ cat emps
Tom Jones      4424      5/12/66     543354
Mary Adams     5346      11/4/63     28765
Sally Chang    1654      7/22/54     6500000
Billy Black    1683      9/23/44     336500

awk '{ print NR, $1, $2, $5 }' emps

1 Tom Jones      543354
2 Mary Adams     28765
3 Sally Chang    6500000
4 Billy Black    336500
```

Example: Colon as Field Seperator

```
~$ cat emps2
Tom Jones:4424:5/12/66:543354
Mary Adams:5346:11/4/63:28765
Sally Chang:1654:7/22/54:650000
Billy Black:1683:9/23/44:336500

awk -F: '/Jones/{ print $1, $2 }' emps2
Tom Jones 4424

awk -F: '/Jones/{print $1, $3}' emps2
Tom Jones 5/12/66
```

Example: File Processing

$ cat input

```
Jan  13  25  15  115
Feb  15  32  24  22
Mar  15  24  34  228
Apr  31  52  63  420
May  16  34  29  208
Jun  31  42  75  492
Jul  24  34  67  436
Aug  15  34  47  316
Sep  15  53  67  277
Oct  29  54  68  525
Nov  20  87  82  577
Dec  17  35  61  401
Jan  21  36  64  620
Feb  26  58  80  652
Mar  24  75  70  495
Apr  21  70  74  514


# prints all records
awk '{print}' input

# prints only first field of each record
awk '{print $1}' input
```

6 Simple Patterns

- BEGIN
  - Matches before the first line of input
  - Used to create header for report
- END
  - Matches after the last line of input
  - Used to create footer for report

6.1 More Patterns

- expression patterns: whole line vs. explicit field match
  - whoe line /regExp/
  - field match $2 ~/regExp/
- range patterns
  - specified as from and to:
    example: /regExp/,/regExp/

Example:

```
$ cat input
Jan  13  25  15  115
Feb  15  32  24  22
Mar  15  24  34  228
Apr  31  52  63  420
May  16  34  29  208
Jun  31  42  75  492
Jul  24  34  67  436
Aug  15  34  47  316
Sep  15  53  67  277
Oct  29  54  68  525
Nov  20  87  82  577
Dec  17  35  61  401
Jan  21  36  64  620
Feb  26  58  80  652
Mar  24  75  70  495
Apr  21  70  74  514


$ cat script
BEGIN {
  print "January Sales Revenue"
}
$1 ~ /Jan/ {
  print $1, $2+$3+$4, $5
}
END {
  print NR, " records processed"
}
# invoking script
awk -f script input

#OUTPUT
January Sales Revenue
Jan 53 115
Jan 121 620
16 records processed
```

7 awk actions

- basic expressions
- output: print, printf
- decisions: if
- loops: for, while

8 awk Expressions

Consists of: **operands** and **operators**

- operands:
  - numeric and string constants
  - variables
  - functions and regular expressions
- operators:
  - assignment: = ++ – += -= *= /=
  - arithmetic: + - * % ^
  - logical: && || !
  - relational: > < >= <= == !=
  - match: ~ !~
  - string concatenation: space

9 awk Variables

- Created via assignment

Syntax:

```
var = expression
```

- types: number (not limited to integer)
- variables come into existence when first used
- type of variable depends on its use
- variables are initilized to either o or “”

Example:

```
cat input
```

Jan 13 25 15 115
Feb 15 32 24 22
Mar 15 24 34 228
Apr 31 52 63 420
May 16 34 29 208
Jun 31 42 75 492
Jul 24 34 67 436
Aug 15 34 47 316
Jan 21 36 64 620

```
cat script
BEGIN {
  print "January Sales Revenue"
  count = 0
  sum = 0
}
$1 ~ /Jan/ && NF == 5 {
  print $1, $2+$3+$4, $5
  count++
  sum+= $5
}
END {
  print count, " records produce: ", sum
}
# invoke
awk -f script input

# OUTPUT
January Sales Revenue
Jan 53 115
Jan 121 620
2 records produce: 735
```

10 awk output: print

- Writes to standard output
  - Output is terminated by newline
- If called with no parameter, it will print $0
- Printed parameters are seperated by blank
- Print control characters are allowed:
  - \n \a \t \b ...

Print examples:

```
$ cat grades
john 85
andrea 89
jasper 84

# Normal Output
$ awk '{print $1, $2}' grades

# values seperated by comma
$ awk '{print $1 "," $2}' grades

$ awk '{print $1 "," $2, " some more"}' grades
```

11 printf: Formatting output

Syntax:

`printf(format-string, var1, var2, ...)`

- Each format specifier within "format-string" requires additional argument of matching type
  - %d, %i decimal integer
  - %c single character
  - %s string of characters
  - %f floating point number

11.1 Format specifier modifers

- between "%" and letter
  - %10s
  - %7d
  - %10.4f
  - %-20s
- Meaning:
  - width of field, field is printed right justified ("-" for left justify)
  - Precision: number of digits after decimal point

12 awk Example: list of products

101: propeller 97.95
102: trailer hitch 97.95
103: sway bar 49.99
104: fishing line 0.99
105: mirror 4.99
106: cup holder 2.49
107: cooler 14.89
108: wheel 49.99
109: transom 199.00
110: pulley 9.88
111: lock 31.00
112: boat cover 120.00
113: premium fish bait 1.00

```
BEGIN {
  FS= ":"
  print "Marine Parts R Us"
  print "Main catalog"
  print "Part-id\tname\t\t\t price"
  print "=================================="
}
{
  printf("%3d\t%-20s\t%6.2f\n", $1, $2, $3)
}
END {
  print "=================================="
  print "Catalog has", NR, "parts"
}
```

13 Typical awk script

Divided into three major parts:

BEGIN {Begin's Actions}	Preprcessing
Pattern {Action} Pattern {Action} Pattern {Action}	Body
END {End's Actions}	Postprocessing

Note:-

Comment lines start with #

14 awk Array

- awk allows one-dimensional arrays
  - index can be number or string
  - elements can be string or number
- array need not be declared
  - its size
  - its element type
- array elements are created when first used
  - initalized to 0 or “”

14.1 Arrays

Syntax:

```
arrayName[index] = value
```

Examples:

```
list[1] = "some value"
list[2] = 123

list["other"] = "oh my !"
```

Array as Map

```
Age["Robert"] = 46
Age["George"] = 22
Age["Juan"] = 19
```

14.2 Array Examples

Input file:

1 clothing 3141
1 computers 9161
1 textbooks 21321
2 clothing 3252
2 computers 12321
2 supplies 2242
2 textbooks 15462

Desired output: **summary of department sales**

Complete program:

```
{
  deptSales[$2] += $3
}
END {
  for (x in deptSales)
    print x, deptSales[x]
}
```

15 awk built-in functions

- **arithmetic**

  ex: sqrt, rand

- **string**

  ex: index, length, split, substr, sprintf, tolower, toupper

- **misc.**

  ex: system, systime

The split function

Syntax:

`split(string, array, fieldsep)`

This divides **string** into pieces separated by **fieldsep**
It then stores the pieces in **array**
if **fieldsep** is omitted, the value of FS is used

Example:

```
split("26:Miller:Comedian", fields, ":")

# Results
fields[1] = "26"
fields[2] = "Miller"
fields[3] = "Comedian"
```

16 awk control structures

- **Conditional**
  - if-else
- **Repetition**
  - for
    - with counter
    - with array index
  - while

    also: break, continue

17 If Statement

Syntax:

```
if (conditional expression)
  statement-1
else
  statement-2
```

Example:

```
if ( NR < 3 )
  print $2
else
  print $3
```

Note:-

Use compound { } for more than one statement.
{
... ...
}

If Statement for arrays

Syntax:

```
if (value in array)
  statement-1
else
  statement-2
```

Example:

```
if ("clothing" in deptSales)
  print deptSales["clothing"]
else
  print "not found"
```

18 for Loop

Syntax:

```
for (initialization; limit-test; update)
  statement
```

Example:

```
for (i=1; i <= 10; i++)
  print "The square of ", i, " is ", i*i
```

for Loop for arrays

Syntax:

```
for (var in array)
  statement
```

Example:

```
for (x in deptSales) {
  print x
  print deptSales[x]
}
```

19 while Loop

Syntax:

```
while (logical expression)
  statement
```

Example:

```
i=1
while (i <= 10) {
  print "The square of ", i, " is ", i*i
  i=i+1
}
```

20 loop control statements

- **break**

  exits loop

- **continue**

  skips the rest of current iteration, continues with next iteration