**Top Examples of Awk Command in Unix**

Awk is one of the most powerful tools in Unix used for processing the rows and columns in a file. Awk has built in string functions and associative arrays. Awk supports most of the operators, conditional blocks, and loops available in C language.   
  
One of the good things is that you can convert Awk scripts into Perl scripts using a2p utility.   
  
**The basic syntax of AWK:**

awk 'BEGIN {start\_action} {action} END {stop\_action}' filename

Here the actions in the begin block are performed before processing the file and the actions in the end block are performed after processing the file. The rest of the actions are performed while processing the file.  
  
**Examples:**  
  
Create a file input\_file with the following data. This file can be easily created using the output of ls -l.

-rw-r--r-- 1 center center  0 Dec  8 21:39 p1

-rw-r--r-- 1 center center 17 Dec  8 21:15 t1

-rw-r--r-- 1 center center 26 Dec  8 21:38 t2

-rw-r--r-- 1 center center 25 Dec  8 21:38 t3

-rw-r--r-- 1 center center 43 Dec  8 21:39 t4

-rw-r--r-- 1 center center 48 Dec  8 21:39 t5

From the data, you can observe that this file has rows and columns. The rows are separated by a new line character and the columns are separated by a space characters. We will use this file as the input for the examples discussed here.   
  
**1.** awk '{print $1}' input\_file  
  
Here $1 has a meaning. $1, $2, $3... represents the first, second, third columns... in a row respectively. This awk command will print the first column in each row as shown below.

-rw-r--r--

-rw-r--r--

-rw-r--r--

-rw-r--r--

-rw-r--r--

-rw-r--r--

To print the 4th and 6th columns in a file use awk '{print $4,$5}' input\_file  
  
Here the Begin and End blocks are not used in awk. So, the print command will be executed for each row it reads from the file. In the next example we will see how to use the Begin and End blocks.  
  
**2.** awk 'BEGIN {sum=0} {sum=sum+$5} END {print sum}' input\_file  
  
This will prints the sum of the value in the 5th column. In the Begin block the variable sum is assigned with value 0. In the next block the value of 5th column is added to the sum variable. This addition of the 5th column to the sum variable repeats for every row it processed. When all the rows are processed the sum variable will hold the sum of the values in the 5th column. This value is printed in the End block.  
  
**3.** In this example we will see how to execute the awk script written in a file. Create a file sum\_column and paste the below script in that file

#!/usr/bin/awk -f

BEGIN {sum=0}

{sum=sum+$5}

END {print sum}

Now execute the the script using awk command as   
  
awk -f sum\_column input\_file.  
  
This will run the script in sum\_column file and displays the sum of the 5th column in the input\_file.  
  
**4.** awk '{ if($9 == "t4") print $0;}' input\_file  
  
This awk command checks for the string "t4" in the 9th column and if it finds a match then it will print the entire line. The output of this awk command is

-rw-r--r-- 1 pcenter pcenter 43 Dec  8 21:39 t4

**5.** awk 'BEGIN { for(i=1;i5;i++) print "square of", i, "is",i\*i; }'  
<=  
This will print the squares of first numbers from 1 to 5. The output of the command is

square of 1 is 1

square of 2 is 4

square of 3 is 9

square of 4 is 16

square of 5 is 25

Notice that the syntax of “if” and “for” are similar to the C language.  
  
**Awk Built in Variables:**   
  
You have already seen $0, $1, $2... which prints the entire line, first column, second column... respectively. Now we will see other built in variables with examples.   
  
**FS** - Input field separator variable:  
  
So far, we have seen the fields separted by a space character. By default Awk assumes that fields in a file are separted by space characters. If the fields in the file are separted by any other character, we can use the FS variable to tell about the delimiter.   
  
**6.** awk 'BEGIN {FS=":"} {print $2}' input\_file  
OR  
awk -F: '{print $2}' input\_file  
  
This will print the result as 

39 p1

15 t1

38 t2

38 t3

39 t4

39 t5

**OFS** - Output field separator variable:   
  
By default whenever we printed the fields using the print statement the fields are displayed with space character as delimiter. For example  
  
**7.** awk '{print $4,$5}' input\_file  
  
The output of this command will be

center 0

center 17

center 26

center 25

center 43

center 48

We can change this default behavior using the OFS variable as  
  
awk 'BEGIN {OFS=":"} {print $4,$5}' input\_file

center:0

center:17

center:26

center:25

center:43

center:48

Note: print $4,$5 and print $4$5 will not work the same way. The first one displays the output with space as delimiter. The second one displays the output without any delimiter.   
  
**NF** - Number of fileds variable:  
  
The NF can be used to know the number of fields in line  
  
**8.** awk '{print NF}' input\_file  
This will display the number of columns in each row.  
  
**NR** - number of records variable:   
The NR can be used to know the line number or count of lines in a file.  
  
**9.** awk '{print NR}' input\_file  
This will display the line numbers from 1.  
  
**10.** awk 'END {print NR}' input\_file  
This will display the total number of lines in the file.  
  
**String functions in Awk:**  
Some of the string functions in awk are:  
  
index(string,search)  
length(string)  
split(string,array,separator)  
substr(string,position)  
substr(string,position,max)  
tolower(string)  
toupper(string)  
  
**Advanced Examples:**  
  
**1.** Filtering lines using Awk split function  
  
The awk split function splits a string into an array using the delimiter.  
  
The syntax of split function is  
split(string, array, delimiter)  
  
Now we will see how to filter the lines using the split function with an example.  
  
The input "file.txt" contains the data in the following format

1 U,N,UNIX,000

2 N,P,SHELL,111

3 I,M,UNIX,222

4 X,Y,BASH,333

5 P,R,SCRIPT,444

Required output: Now we have to print only the lines in which whose 2nd field has the string "UNIX" as the 3rd field( The 2nd filed in the line is separated by comma delimiter ).  
The ouptut is:

1 U,N,UNIX,000

3 I,M,UNIX,222

The awk command for getting the output is:

awk '{

split($2,arr,",");

if(arr[3] == "UNIX")

print $0

} ' file.txt

**Examples**

| **Learning Linux awk command with examples** | |
| --- | --- |
| **Linux command syntax** | **Linux command description** |
|  | |
| awk ' {print $1,$3} ' | Print only columns one and three using stdin |
| awk ' {print \_\_g5\_token58ab212c7745c} ' | Print all columns using stdin |
| awk ' /'pattern'/ {print $2} ' | Print only elements from column 2 that match pattern using stdin |
| awk -f script.awk inputfile | Just like make or sed, awk uses -f to get its' instructions from a file, useful when there is a lot to be done and using the terminal would be impractical |
| awk ' program ' inputfile | Execute program using data from inputfile |
| awk "BEGIN { print \"Hello, world!!\" }" | Classic "Hello, world" in awk |
| awk '{ print }' | Print what's entered on the command line until EOF (^D) |
| #! /bin/awk -f BEGIN { print "Hello, world!" } | awk script for the classic "Hello, world!" (make it executable with chmod and run it as-is) |
| # This is a program that prints \ "Hello, world!" # and exits | Comments in awk scripts |
| awk -F "" 'program' files | Define the FS (field separator) as null, as opposed to white space, the default |
| awk -F "regex" 'program' files | FS can also be a regular expression |
| awk 'BEGIN { print "Here is a single \ quote <'\''>" }' | Will print <'>. Here's why we prefer Bourne shells. :) |
| awk '{ if (length(\_\_g5\_token58ab212c777ee) > max) max = \ length(\_\_g5\_token58ab212c777ee) } END { print max }' inputfile | Print the length of the longest line |
| awk 'length(\_\_g5\_token58ab212c7783a) > 80' inputfile | Print all lines longer than 80 characters |
| awk 'NF > 0' data | Print every line that has at least one field (NF stands for Number of Fields) |
| awk 'BEGIN { for (i = 1; i <= 7; i++) print int(101 \* rand()) }' | Print seven random numbers from 0 to 100 |
| ls -l . | awk '{ x += $5 } ; END \ { print "total bytes: " x }' total bytes: 7449362 | Print the total number of bytes used by files in the current directory |
| ls -l . | awk '{ x += $5 } ; END \ { print "total kilobytes: " (x + \ 1023)/1024 }' total kilobytes: 7275.85 | Print the total number of kilobytes used by files in the current directory |
| awk -F: '{ print $1 }' /etc/passwd | sort | Print sorted list of login names |
| awk 'END { print NR }' inputfile | Print number of lines in a file, as NR stands for Number of Rows |
| awk 'NR % 2 == 0' data | Print the even-numbered lines in a file. How would you print the odd-numbered lines? |
| ls -l | awk '$6 == "Nov" { sum += $5 } END { print sum }' | Prints the total number of bytes of files that were last modified in November |
| awk '$1 ̃/J/' inputfile | Regular expression matching all entries in the first field that start with a capital j |
| awk '$1 ̃!/J/' inputfile | Regular expression matching all entries in the first field that **don't** start with a capital j |
| awk 'BEGIN { print "He said \"hi!\" \ to her." }' | Escaping double quotes in awk |
| echo aaaabcd | awk '{ sub(/a+/, \ "<A>"); print }' | Prints "<A>bcd" |
| ls -lh | awk '{ owner = $3 ; $3 = $3 \ " 0wnz"; print $3 }' | uniq | Attribution example; try it :) |
| awk '{ $2 = $2 - 10; print \_\_g5\_token58ab212c77bfb }' inventory | Modify inventory and print it, with the difference being that the value of the second field will be lessened by 10 |
| awk '{ $6 = ($5 + $4 + $3 + $2); print \ $6' inventory | Even though field six doesn't exist in inventory, you can create it and assign values to it, then display it |
| echo a b c d | awk '{ OFS = ":"; $2 = "" > print \_\_g5\_token58ab212c77d78; print NF }' | OFS is the Output Field Separator and the command will output "a::c:d" and "4" because although field two is nullified, it still exists so it gets counted |
| echo a b c d | awk ’{ OFS = ":"; \ $2 = ""; $6 = "new" > print \_\_g5\_token58ab212c77ddf; print NF }’ | Another example of field creation; as you can see, the field between \$4 (existing) and \$6 (to be created) gets created as well (as \$5 with an empty value), so the output will be "a::c:d::new" "6" |
| echo a b c d e f | awk ’\ { print "NF =", NF; > NF = 3; print \_\_g5\_token58ab212c77e2e }’ | Throwing away three fields (last ones) by changing the number of fields |
| FS=[ ] | This is a regular expression setting the field separator to space and nothing else (non-greedy pattern matching) |
| echo ' a b c d ' | awk 'BEGIN { FS = \ "[ \t\n]+" } > { print $2 }' | This will print only "a" |
| awk -n '/RE/{p;q;}' file.txt | Print only the first match of RE (regular expression) |
| awk -F\\ ’...’ inputfiles ... | Sets FS to \\ |
| BEGIN { RS = "" ; FS = "\n" } { print "Name is:", $1 print "Address is:", $2 print "City and State are:", $3 print "" } | If we have a record like "John Doe 1234 Unknown Ave. Doeville, MA", this script sets the field separator to  newline so it can easily operate on rows |
| awk ’BEGIN { OFS = ";"; ORS = "\n\n" } > { print $1, $2 }’ inputfile | With a two-field file, the records will be printed like this: "field1:field2  field3;field4  ...;..." Because ORS, the Output Record Separator, is set to two newlines and OFS is ";" |
| awk ’BEGIN { > OFMT = "%.0f" # print numbers as \ integers (rounds) > print 17.23, 17.54 }’ | This will print 17 and 18, because the Output ForMaT is set to round floating point values to the closest integer value |
| awk ’BEGIN { > msg = "Dont Panic!" > printf "%s\n", msg >} ' | You can use printf mainly how you use it in C |
| awk ’{ printf "%-10s %s\n", $1, \ $2 }’ inputfile | Prints the first field as a 10-character string, left-justified, and \$2 normally, next to it |
| awk ’BEGIN { print "Name Number"  print "---- ------" }  { printf "%-10s %s\n", $1, \ $2 }’ inputfile | Making things prettier |
| awk ’{ print $2 > "phone-list" }' \ inputfile | Simple data extraction example, where the second field is written to a file named "phone-list" |
| awk ’{ print $1 > "names.unsorted"  command = "sort -r > names.sorted"  print $1 | command }’ inputfile | Write the names contained in \$1 to a file, then sort and output the result to another file (you can also append with >>, like you would in a shell) |
| awk ’BEGIN { printf "%d, %d, %d\n", 011, 11, \ 0x11 }’ | Will print 9, 11, 17 |
| if (/foo/ || /bar/)  print "Found!" | Simple search for foo *or* bar |
| awk ’{ sum = $2 + $3 + $4 ; avg = sum / 3 > print $1, avg }’ grades | Simple arithmetic operations (most operators resemble C a lot) |
| awk '{ print "The square root of", \ $1, "is", sqrt($1) }' 2 The square root of 2 is 1.41421 7 The square root of 7 is 2.64575 | Simple, extensible calculator |
| awk ’$1 == "start", $1 == "stop"’ inputfile | Prints every record between start and stop |
| awk ’  > BEGIN { print "Analysis of \"foo\"" }  > /foo/ { ++n }  > END { print "\"foo\" appears", n,\  "times." }’ inputfile | BEGIN and END rules are executed exactly once, before and after any record processing |
| echo -n "Enter search pattern: " read pattern awk "/$pattern/ "’{ nmatches++ } END { print nmatches, "found" }’ inputfile | Search using shell |
| if (x % 2 == 0) print "x is even" else print "x is odd" | Simple conditional. awk, like C, also supports the ?: operators |
| awk ’{ i = 1  while (i <= 3) {  print $i  i++  } }’ inputfile | Prints the first three fields of each record, one per line. |
| awk ’{ for (i = 1; i <= 3; i++)  print $i }’ | Prints the first three fields of each record, one per line. |
| BEGIN { if (("date" | getline date\_now) <= 0) {  print "Can’t get system date" > \ "/dev/stderr"  exit 1 } print "current date is", date\_now close("date") } | Exiting with an error code different from 0 means something's not quite right. Here's and example |
| awk ’BEGIN { > for (i = 0; i < ARGC; i++) > print ARGV[i] > }’ file1 file2 | Prints awk file1 file2 |
| for (i in frequencies) delete frequencies[i] | Delete elements in an array |
| foo[4] = "" if (4 in foo) print "This is printed, even though foo[4] \ is empty" | Check for array elements |
| function ctime(ts, format) {  format = "%a %b %d %H:%M:%S %Z %Y"  if (ts == 0)  ts = systime()  # use current time as default  return strftime(format, ts) } | An awk variant of ctime() in C. This is how you define your own functions in awk |
| BEGIN { \_cliff\_seed = 0.1 } function cliff\_rand() {  \_cliff\_seed = (100 \* log(\_cliff\_seed)) % 1  if (\_cliff\_seed < 0)  \_cliff\_seed = - \_cliff\_seed  return \_cliff\_seed } | A Cliff random number generator |
| cat apache-anon-noadmin.log | \ awk 'function ri(n) \ { return int(n\*rand()); } \ BEGIN { srand(); } { if (! \ ($1 in randip)) { \ randip[$1] = sprintf("%d.%d.%d.%d", \ ri(255), ri(255)\ , ri(255), ri(255)); } \ $1 = randip[$1]; print \_\_g5\_token58ab212c792d6 }' | Anonymize an Apache log (IPs are randomized) |