CHAPTER 9

Awk Command

9.1 INTRODUCTION

This facility is very much useful for small scale database applications requiring no precision. This is the most powerful command which can mimic many of the other commands. We do have variants of this command with the names nark, gawk, etc.,.

Syntax of awk command

```
awk OPTIONS `BEGIN{
            }
            {
            }
            END{
            }' filename
```

Awk command considers the given file as a database file; each line of the file is considered as a record and each word of a line is taken as a field [Venkateswarlu]. Whatever operations we wanted to execute before processing any line of the file, we have write in the BEGIN section. The operations which are required to be executed after processing all the records have to be written in END section. Instructions which are required to be executed on every record have to be written in the middle block. It is not necessary that every awk command to have all the three blocks. However, opening curly braces should immediately follow the BEGIN and END words and should be in the same line as that of BEGIN, END words. Awk supports a limited amount of C style of programming constructs. However, we can not say that it can be used in place of C language. While running, instructions in the BEGIN block are executed first, then instructions in the middle block are executed on every record and at the end instructions in END block are executed.

Normally, awk assume space or TAB as the field separator between words. However, if a file contains some other character as field separator, the same can be informed through -F option.

awk uses the following reserved parameters or variables :

NF = number of fields

NR = number of records

OFS = output field separator
$0 = current record as a whole
$1, $2, $3… = first, second, third etc., fields of current record

The following commands display the content of the given file. That is, they behave similar to cat command. Some versions of awk, the default behavior is to display the record which is read. Thus, with those awk commands we may see each line getting displayed two times. These awk versions will be having an option –n by using it we can suppress this problem.

awk ‘{ print $0} ‘ filename

awk ‘{ printf “%s”, $0 }’ filename

The following command displays given file’s (aa) content along with line numbers. Here, NR value refers to the record number. The print statement will be executed on each line of the given file. First, it reads a line. Automatically, NR value will be set to 1. Thus, the printf statement prints 1 and line content. When next line is read, NR value will be automatically incremented by 1. Like this, the content of the file will be printed along with line numbers.

awk ‘{ printf “%3d %s”, NR, $0}’ aa

In the above command, we can use print command instead of printf. We can use the following commands also to get the same effect. In awk, we can use variables where ever we want. The initial values will be considered as 0 (or null). We can apply increment, decrement operators without any hesitation. Thus, in the following commands, initially x value is considered as zero and is incremented when first line is read. Similarly, it will be further incremented by one when second line is read. Also, we can use a semicolon to separate the instructions. Of course, if the instructions are given in one per line, semicolon is not necessary. See the output of our interactive session.

```
$ awk ‘{x++
> print x, $0
> }’ aa
1 asas
2 asas
$ awk ‘{ x++ print x, $0}’ aa
awk: { x++ print x, $0}
awk: ^ syntax error
$ awk ‘{ x++; print x, $0}’ aa
1 asas
2 asas
```

Let us assume that we want awk to emulate wc command. That is, we want number of lines, words and characters to be printed in the given file. See our interactive workout at the shell.

```
$ awk '{ x+=NF; y+=length($0);} END{ print NR, x, y}' aa
2 2 8
$ awk '{ x+=NF;
>y+=length($0);}
>END{ print NR, x, y}' aa
2 2 8
```

In the above commands, we have used length function which returns the number of characters in the given argument. Here, we are sending $0 which is current line. Thus, we get number of characters in the current line. In a nutshell, each time a record is read and number of fields (NF) and characters in it are added to x and y variables respectively. Thus, x and y will be having total number of words and characters. Of course, NR value is the record number. Thus, when we ask to print NR, x and y in the END block, we will get number of lines, words and characters.

To print number of lines in file (of course wc -l does this, too)

awk 'END{print NR}' file

The following command displays UID's and username's of the legal users of the machine.

awk -F":" '{ print $3, $1}' /etc/passwd

awk -F":" '{ printf "%3d %s", $3, $1 }' /etc/passwd

Unlike cut command, awk command is does not preserve the order of the fields in output. Refer our discussion on this with cut command. That is, output of the following commands is different (unlike cut).

awk -F":" '{ print $3, $1 }' /etc/passwd

awk -F":" '{ print $1, $3 }' /etc/passwd

We can use awk command to behave like grep. For example, the following command will display those lines of the given file with the string "fleece". Here, we did not specify to print. However, print command is the default if nothing is specified.

awk '/fleece/' file

The above command is as good as the following.

awk '/fleece/{print $0}' file

awk '$0 ~ /fleece/' file

awk '$0 ~ /fleece/{print $0}' file

In the above commands, $0 refers to the current record as a whole.

The following command will select lines 14 through 30 of file.

awk 'NR==14, NR==30' file

To select a line (say 12th line) of the given file, we can run the following.

awk 'NR==12' file

The following command displays those lines of the given files, whose first field value is same as line number.

The following command will print those lines of the given file whose first field values are same as record number.

awk "NR==$1" file

To rearrange the fields 1 and 2 and put colon in between

awk '{print $2 ":" $1}' file

All lines between BEGIN and END lines (you can substitute any strings for BEGIN and END, but they must be between slashes) will be displayed.

awk '/BEGIN/,/END/' file

Substitute every occurrence of a string XYZ by the new string ABC: Requires nawk.

nawk '{gsub(/XYZ/,"ABC"); print}' file

Assuming that we have a file field separator : in the given file then the following command will print 3rd field from each line of the given file. That is, it acts like cut command.

awk -F: '{print $3}' file

To print out the last field in each line, regardless of how many fields :

awk '{print $NF}' file

Print out lengths of lines in the file

awk '{print length($0)}' somefile

or

awk '{print length}' somefile

Print out lines and line numbers that are longer than 80 characters

awk 'length > 80 {printf "%3d. %s\n", NR, $0}' somefile

In ls –l output, 7th field will be having file sizes. We want the sum of these sizes. Thus, we redirect the output ls –l command to awk through pipe as shown below. Here, in the begin block we have defined a variable total with initial value 0. For this, each file size is added. In the END block, we displayed the accumulated total.

ls -l | awk 'BEGIN{total=0} {total += $7} END{print total}'

The following will consider only regular files. We are selecting the regular files details using grep command.

ls -l |grep "^-"| awk 'BEGIN{total=0} {total += $7} END{print total}'

The following command will display "Yes", if the first field of each line is same as its line number. Otherwise, it prints "No". This will be used to test the integrity of numbered files.

$ awk 'NR == $1 {x++} END{ if (x == NR) {printf "%s", "Yes"} else { printf "%

s", "No" } }' pppp

No

$ cat pppp

1 Rama is a good boy

2 Abhi is a boy

4 Annuj is a bad

If the file pppp contains 3 in the first field, then we will get message "Yes".

The following command will print out the longest line in the given file .

```
$awk 'BEGIN {maxlength = 0} \
{ \ if (length($0) maxlength) { \
maxlength = length($0) \
longest = $0 \
} \
} \
END {print longest}' somefile
```

The above can be implemented in a single line also as :

```
awk ' length > maxlength { maxlength=length}END{print maxlength}'
                    filename
```

To find how many entirely blank lines are in a file, the following command will be useful.

```
awk '/^$/ {x++} END {print x}' somefile
```

Print out last character of field 1 of every line

```
awk '{print substr($1,length($1),1)}' somefile
```

Comment out only #include statements in a C file. This is useful if we want to run "cxref" which will follow the include links.

```
awk '/#include/{printf "/* %s */\n", $0; next} {print}' file.c | cxref
-c $*
```

If the last character of a line is a colon, print out the line. This would be useful in getting the pathname from output of ls -lR :

```
$awk '{ \
lastchar = substr($0,length($0),1) \
if (lastchar == ":") \
print $0 \
}' somefile
```

Here is the complete thing....Note that it even sorts the final output

```
$ls -lR | awk '{ \
lastchar = substr($0,length($0),1) \
if (lastchar == ":") \
dirname = substr($0,1,length($0)-1) \
else \
if ($4 20000) \
printf "%10d %25s %s\n", $4, dirname, $8 \
}' | sort -r
```

The following is used to break all long lines of a file into chunks of length 80 :

```
$awk '{ line = $0
while (length(line) 80)
{
print substr(line,1,80) line = substr(line,81,length(line)-80)
}
if (length(line) 0) print line
}' somefile.with.long.lines>whatever
```

If we want to use awk as a programming language, we can do so by not processing any file, but by enclosing a bunch of awk commands in curly braces, activated upon end of file. To use a standard UNIX "file" that has no lines, use /dev/null.

Here's a simple example :

```
awk 'END{print "hi there everyone"}' < /dev/null
```

Here's an example of using this to print out the ASCII characters :

```
$awk ' { for(i=32; i<127; i++) \
printf "%3d %3o %c\n", i,i,i \
}' </dev/null
```

Sometimes we wish to find a field which has some identifying tag, like X= in front. Suppose our file (playfile1) looked like :

50 30 X=10 Y=100 Z=-2
X=12 89 100 32 Y=900
1 2 3 4 5 6 X=1000

Then to select out the X= numbers from each line, do

```
$awk '{ for (i=1; i <=NF; i++) \
if ($i ~ /X=.*/) \
print substr($i,3) \
}' playfile1
```

Note that we used a regular expression to find the initial part: /X=.*/

Pull an abbreviation out of a file of abbreviations and their translation. Actually, this can be used to translate anything, where the first field is the thing you are looking up and the 2nd field is what you want to output as the translation.

```
nawk '$1 == abbrev{print $2}' abbrev=$1 translate.file
```

Join lines in a file that end in a dash. That is, if any line ends in -, join it to the next line. This only joins 2 lines at a time. The dash is removed.

```
$awk '/-$/ {oldline = $0 \
getline \
print substr(oldline,1,length(oldline)-1) $0 \
next} \
{print}' somefile
```

Function in nawk to round: function round(n) { return int(n+0.5) }

If you have a file of addresses with empty lines between the sections, you can use the following to search for strings in a section, and print out the whole section. Put the following into a file called "section.awk" :

BEGIN {FS = "\n"; RS = ""; OFS = "\n"} $0 ~ searchstring {print }

Assume our names are in a file called "rolodex". Then use the following nawk command when you want to find a section that contains a string. In this example, it is a person's name :

nawk -f section.awk searchstring=Wolf rolodex

We assume the following data in the file EMPLOYEE having employee ID, name, designation, department, salary, no of dependents and age in each line.

111|NB Venkateswarlu|Professor|CSE|27000|2|42

121|GV Saradamba|Professor|CHEM|32000|2|46

122|PN Rao|Assistant Professor|Civil|26000|3|54

awk -F"|" '{ printf "%s %d", $2 , $1}' EMPLOYEE

This command displays names of the employees and their ID's in a tabular fashion.

awk -F"|" ' $2 ~ /Rao/{ printf "%s %d", $2 , $1}' EMPLOYEE

This command displays names of the employees and their ID's whose name contains the string "Rao".

awk -F"|" ' $2 ~ /Ra[ou]/{ printf "%s %d", $2 , $1}' EMPLOYEE

This command displays names of the employees and their ID's whose name contains the string "Rao" or "Rau".

awk -F"|" ' $2 ~ /^Rao/{ printf "%s %d", $2 , $1}' EMPLOYEE

This command displays names of the employees and their ID's whose name starts with the string "Rao".

awk -F"|" ' $2 ~ /^Ra[ou]/{ printf "%s %d", $2 , $1}' EMPLOYEE

This command displays names of the employees and their ID's whose name starts with the string "Rao" or "Rau".

awk -F"|" ' $2 ~ /Rao$/{ printf "%s %d", $2 , $1}' EMPLOYEE

This command displays names of the employees and their ID's whose name ends with the string "Rao".

awk -F"|" ' $2 ~ /Ra[ou]$/{ printf "%s %d", $2 , $1}' EMPLOYEE

This command displays names of the employees and their ID's whose name ends with the string "Rao" or "Rau".

awk -F"|" ' $2 ~ /^Ra[ou]$/{ printf "%s %d", $2 , $1}' EMPLOYEE

This command displays names of the employees and their ID's whose name contains the strings "Rao" or "Rau".

awk -F"|" ' $2 ~ /^[rR]a[ou]$/{ printf "%s %d", $2 , $1}' EMPLOYEE

This command displays names of the employees and their ID's whose name contains the strings "Rao", "rao", "Rau" or "rau".

```
awk -F"|" ` $4 >10000 { printf "%s %d", $2 , $1}' EMPLOYEE
```

This command displays names of the employees and their ID's whose salary is more than 10000.

```
awk -F"|"'{ s+=$4 p+=$6} END { printf "%d %d", s/NR, p/NR}'
EMPLOYEE
```

This command displays average salary and average number of dependents of the employees.

```
awk -F"|"'$4<5000{s+=$4 p+=$6}END{printf "%d %d",s/NR,p/
NR}' EMPLOYEE
```

This command displays average salary and average number of dependents of the employees whose salary is less than 5000.

```
awk -F"|"'$6<50{s+=$4 p+=$6}END{printf"%d %d",s/NR,p/NR}'
EMPLOYEE
```

This command displays average salary and average number of dependents of the employees whose age is more than 50.

```
awk -F"|" ` {
        If ($4 >5000) n1++
        Else n2++
        }END{ printf "%d %d", n1, n2 }' EMPLOYEE
```

This command displays no of employees whose salary is greater than 5000 and less than 5000.

```
awk -F"|" ` {
          If ($4 >5000)
          {
          n1++
          s1+=$4
          }
          else
          {
          n2++
          s2+=$4
          }
          }END{ printf "%d %d", s1/n1, s2/n2 }' EMPLOYEE
```

This command displays average salary of employees whose salary is greater than 5000 and less than 5000.

We can use arrays also. Their initial values also taken as zeros.

```
awk -F"|" ` {
        if ($4 >5000)
        {
        s[1]++
        s[2]+=$4
        }
        else
        {
        s[3]++
        s[4]+=$4
        }
        }END{ printf "%d %d", s[2]/s[1], s[4]/s[3] }' EMPLOYEE
```

This command displays average salary of employees whose salary is greater than 5000 and less than 5000.

We can use content addressable arrays. That is, the array element indexes for these arrays can be strings rather than usual integers.

```
awk -F"|" `{ s[$3]++} END{ for(desig in s) printf "%s %d", desig,
s[desig] }' EMPLOYEE
```

The above command displays designation and number of people having that designation. Here, we are using an array s. The third field value is designation which can be Professor, Lecturer, etc.,. When awk reads a record and $3 value is Professor then s of Professor is incremented by one. This is repeated for each record. Thus, the array contains number of Professors, Lecturers and vice versa. The END block uses a special version of for loop to print the array information. The syntax of the for loop is :

for(var in arrayname)

In the above example, we have used design as var and s as arrayname. This, design acquires values as Professor, Lecturer and vice versa. The respected array content is also printed. Thus, the output will be :

Lecturer 4

Professor 6

Let us assume that we want total money spent on each department on salaries. The following awk command will be useful.

```
awk -F"|" `{ s[$4]++} END{ for(dept in s) printf "%s %d", dept, s[dept] }'
EMPLOYEE
```

```
awk `{ l=(80-length($0))/2
        I=0;
        while(I<l)
        {
        printf "%s", " "
        I++;
        }
    printf "%s", $0 }' filename
```

This program prints every line of the program centered on the screen.

```
awk '{ l=(80-length($0))/2
        for(I=0;I<l; I++)
        {
        printf "%s", " "
        }
    printf "%s", $0 }' filename
```

This program prints every line of the program centered on the screen.

9.2 CONCLUSIONS

This chapter introduces the user to most widely used UNIX command, awk. Live examples are included to explain how really awk can be used in database applications. Also, examples are included which employs the C style constructs such as for, while, if, etc., in awk.