AWK

Filters

- A group of UNIX commands accepts some data as input, performs some manipulation on it and produces some output . These group of commands are called filters.
- The filters use standard input and standard output and can be used with redirection and pipelines.
- eg. head ,tail, cut, paste, sort etc.
- grep, egrep
- eg. grep sales emp.lst
- egrep '(sen|das)gupta' emp.lst

sed

- sed is a stream editor. A stream editor is used to perform basic text transformations on an input stream.
- Everything in sed is an instruction. It is a multipurpose tool which combines the work of several filters . The general form is-

  sed options 'address action' file(s)
- An instruction combines an address for selecting lines with an action to be taken.
- sed has two ways of addressing lines - by line number and by specifying a pattern which occurs in a line.

  $sed –n '/director/p' emp.lst

  # -n : for suppressing double printing , used with p

AWK

- The awk command was introduced in1977. It is named after its author Aho, Weinberger & Kernighan.
- awk utility is powerful data manipulation/scripting programming language.
- awk combines features of several filters. awk appears as gawk(GNU awk) in Linux.
- awk operates at field level. It can easily access, transform and format individual fields in a record.

- The general syntax is

  awk options ‘address{action} ‘ files
- Here, the address and action constitute an awk program. These programs are mostly one line long, but they can also include several lines.

Simple awk filtering

- A awk command specifies an address and an action.
- To select directors from emp.lst use -

  $ awk ‘/directors/{print}’ emp.lst # lists directors from emp.lst
- The address section(/directors/) selects lines that are processed in action section({print}).
- If the address is missing, the action applies to all lines of the file. If the action is missing , the entire line will be printed. Printing is the default action of awk .
- $ awk ’/director/’ emp.lst
- $ awk ’/director/{print }’emp.lst
- $ awk ’/director/{print$0}’emp.lst
- All above commands are equivalent
- For pattern matching, awk uses regular expressions in sed style.
- $ awk -F “|” ’/sa[kx]s*ena/’ emp.lst
- It also allows multiple patterns.

Demo file- stud

Rollno	Name	PCM	USA	IOM
101	Ameya	18	18	16
135	Bharat	16	16	15
145	Chaitanya	15	16	14
123	Deepak	16	16	16
125	Hari	17	16	15
156	Ganesha	17	17	16
159	Ratan	17	17	17

SPLITTING A LINE INTO FIELDS

- awk uses the special "variable" $0 to indicate the entire line. It also identifies fields by $1, $2, $3,… . These parameters also have a special meaning to the shell.
- Single-quoting an awk program protects them from interpretation by the shell.
- Unlike other UNIX filters which operate on fields, awk also uses a contiguous sequence of spaces and tabs as a single delimiter. If the sample database uses the |, then -F option must be used to specify it in programs.
- You can use awk to print the name, marks in all or some subjects :
- $ awk ‘ { print $1,$2 }’ stud

  101 Ameya

  135 Bharat

  145 Chaitanya

  123 Deepak

  125 Hari

  156 Ganesha

  159 Ratan

Note a , (comma) is used to delimit the field specifications. This ensures that each field is separated from the other by a space. If you don't put the comma, the fields will be glued together.

- User can use awk with a line address to select lines. Use the built-in variable NR to specify the line numbers:
- $ awk 'NR == 3, NR == 5 { print $1,$2 }’ stud

  145 Chaitanya

  123 Deepak

  125 Hari

- The statement NR == 3 is really a condition that is being tested, rather than an assignment. NR is just one of those built-in variables used in awk programs, and == is one of the many operators employed in comparison tests.
- $ awk ‘ { print $1,$2, $3+$4+$5 }’ stud

  101 Ameya 52

135 Bharat 47

145 Chaitanya 45

123 Deepak 48

125 Hari 48

156 Ganesha 50

159 Ratan 51

- $ awk ‘ { print $1,$2, $3,$4,$5,$3+$4+$5 }’ stud

101 Ameya 18 18 16 52

135 Bharat 16 16 15 47

145 Chaitanya 15 16 14 45

123 Deepak 16 16 16 48

125 Hari 17 16 15 48

156 Ganesha 17 17 16 50

159 Ratan 17 17 17 51

printf: FORMATTING OUTPUT

- The previous output is unformatted, but with the C-like printf statement, we can use awk as a stream formatter. awk accepts most of the formats used by the C printf function. We can produce a list of all the *students* -
- $ awk ‘ { printf "%5d %-15s %7d %7d %7d %7d\n", $1,$2,$3,$4,$5, $3+$4+$5 }' stud

101 Ameya 18 18 16 52

135 Bharat 16 16 15 47

145 Chaitanya 15 16 14 45

123 Deepak 16 16 16 48

125 Hari 17 16 15 48

156 Ganesha 17 17 16 50

159 Ratan 17 17 17 51

Built in variables

awk has several built in variables. The values are assigned automatically. The user can reassign some of them.

- **FS**-Field Separator

If the database uses | as a field separator then it must occur in BEGIN section so that the body of program knows its value before it starts processing. (Blank or tab is default)

BEGIN { FS="|" }

This is equivalent to –F option.

- **OFS** - Output Field Separator

When print statement is used with comma separated arguments, value of each argument is separated by a space in output. (Blank is default) . This can be assigned with new value as-

BEGIN { OFS="~" }

- **ORS** -

  Output record separator string (Default is new line)

- **NF** - Number of fields in a line

  This variable can be used to check the number of fields in current line.

- **NR**-

  Cumulative Number of lines read

- **FILENAME**-

  This stores the name of the current file being processed.

- **ARGC**-

  Number of arguments in command line

- **ARGV**-

  List of arguments

- **RS** –

  Input record separator character (Default is new line)

- **OFMT**-

  Output format of number

Arithmetic in awk

- You can easily, do the arithmetic with awk as follows

- $ cat > **maths**

```
{
        print $1 " + " $2 " = " $1 + $2
        print $1 " - " $2 " = " $1 - $2
```

```
        print $1 " / " $2 " = " $1 / $2
        print $1 " x " $2 " = " $1 * $2
        print $1 " mod " $2 " = " $1 % $2
}
```

- Run the awk program as follows:
- **$ awk -f maths**

```
        20 7
        20 + 7 = 27
        20 - 7 = 13
        20 / 7 =
```

- 2.8572

```
        20 x 7 = 140
        20 mod 7 = 6
        (Press CTRL + D to terminate)
```

User Defined variables in awk

- You can also define your own variable in awk program, as follows:
- $ cat > maths1

```
  {
        no1 = $1
        no2 = $2
        ans = $1 + $2
        print no1 " + " no2 " = " ans
  }
```

- Run the program as follows

```
$ awk -f maths1
        2  7
        2 + 7 = 9
```

- In the above program, no1, no2, ans all are user defined variables. Value of first and second field are assigned to no1, no2 variable respectively and the addition to ans variable.

LOGICAL AND RELATIONAL OPERATORS:

- awk supports various logical and relational operators.

  e.g.

```
$ awk '$1 == 125 || $1 == 135{
> printf "%-15s %5d %5d %5d\n", $2,$3,$4,$5 }' stud
        125 Hari       17    16     15
```

- awk also uses the II and && logical operators in the same way as used by C and the UNIX shell.
- This command looks for two numbers only in the first field ($1). The second string is searched only if (||) the first search fails.
- For negating the above condition, use the != and && operators as:

```
$1 != 125 || $1 != 135
```

Using Relational Operators

- awk can also handle numbers, both integer and floating type, and all the relational tests can be handled by awk. Using relational operators the list of students scoring greater than 50 marks can be printed-

```
$ awk '$3+$4+$5 >= 50 {  printf "%-20s %d\n", $2,  $3+$4+$5 }' stud
        101  Ameya              52
        156 Ganesha             50
        159 Ratan               51
```

Relational Operators

Operator	Significance
<	Less than
<=	Less than or equal to
= =	Equal to

!=	Not equal to
>=	Greater than or equal to
>	Greater than
~	Matches a regular expression
!~	Does not match a regular expression

File ‘store’

- Create the following file

Sr.No	Product	Qty	Unit Price
1	Pen	100	40.00
2	Rubber	60	5.00
3	Pencil	60	3.50
4	GeoBox	20	145.00
5	ColorBox	30	70.00
6	NoteBook	200	32.00
7	Steppler	30	42.00

- cat > bill1

```
{
total = $3*$4
srno = $1
item = $2
grandtotal = grandtotal + total
print srno  item  “ Rs. “ total
}
```

- **$ awk –f bill1 stores**

```
pen Rs. 4000
pencil Rs. 210
rubber Rs. 300
geobox Rs. 2900
```

colbox Rs. 2100
notebook Rs. 7400
steppler Rs. 1260

The BEGIN and END sections

- In awk program, the action part is applied to the addressed lines. If, it is required to print something before the processing of first line, eg heading, the BEGIN section is useful. Similarly the END section is useful in printing some totals after the processing is over.
- The BEGIN and END sections are optional.
- **The general form is-**

  BEGIN {action} # Both require braces

- cat > bill2

```
BEGIN{
        print ”----------------------------------------------------”
         print ” Inventory Cost”
         print ”----------------------------------------------------” }
 {
        total = $3*$4
        srno = $1
        item = $2
        grandtotal = grandtotal + total
        print srno  item  “ Rs. “ total          }
END {
         print ”----------------------------------------------------”
        print  “ Total Rs. “  grandtotal
        print  ”==================================“  }
```

- $ awk –f bill2 stores

```
 ------------------------------------------------
 Inventory Cost
------------------------------------------------
1   pen   Rs. 4000
2   pencil   Rs. 210
3   rubber   Rs. 300
4   geobox   Rs. 2900
5   colbox   Rs. 2100
6   notebook   Rs. 7400
7   steppler   Rs. 1260
------------------------------------------------
  Grand Total  Rs.=  18170
================================
```

- $ cat > bill3

```
BEGIN {
        print "------------------------------------------------"
        print " Inventory Cost including Tax"
        print "------------------------------------------------"}
{
total = $3*$4
srno = $1
item = $2
grtotal += total
print srno "   "item "   ""Rs. " total
}
END  {
        print "------------------------------------------------"
        print "  Grand Total  Rs.=  " grtotal
         print "------------------------------------------------"
        print "  Grand Total + VAT  Rs.=  " grtotal + 0.05*grtotal
         print "========================================"  }
```

- $ awk -f bill3 stores

```
------------------------------------------------
 Inventory Cost including Tax
------------------------------------------------
1   pen   Rs. 4000
2   pencil   Rs. 210
3   rubber   Rs. 300
4   geobox   Rs. 2900
5   colbox   Rs. 2100
6   notebook   Rs. 7400
7   steppler   Rs. 1260
------------------------------------------------
  Grand Total  Rs.=  18170
------------------------------------------------
  Grand Total + VAT  Rs.=  19078.5
==============================
```

if condition in awk:

- General syntax of if condition is as follows:

  Syntax:

```
if ( condition )
    {
      Statement 1
      Statement 2
      Statement N
      }
else
    {
      Statement 1
      Statement 2
      Statement N
    }
```

Loops in awk

- for loop and while loop are used for looping purpose in awk.
- **Syntax of for loop-**

```
for (expr1; condition; expr2)
{
    Statement 1
    Statement 2
    Statement N
}
```

-

```
{
sum = 0
i = 1
for (i=1; i<=10; i++)
     {
          sum = sum + i
     }
printf "Sum for 1 to 10 numbers = %d  \n", sum
 }
```

while loop

- Syntax of while loop is as follows:

```
while (condition)
  {
    statement1
    statement2
    statementN
  }
```

- While loop will continue as long as given condition is TRUE.

Reversing the number

```
{
no = $1
remn = 0
while ( no > 1 )
```

```
  {
     remn = no % 10
     no /= 10
     printf "%d" ,remn   }
printf "\nNext number please (CTRL+D to stop):";}
```

getline function

- getline function is used to read input from keyboard and then assign the data to variable.

- *Syntax:*

```
getline variable-name < "-"
|              |                  |
1              2                  3
```

1 --> getline is function name
2 --> variable-name is used to assign the value read from input
3 --> Means read from stdin (keyboard)

```
BEGIN {
    printf "Your name please:"
    getline na < "-"
    printf " your age please:"
    getline age < "-"
   print "Hello " na, ", your age is ", age
        }
       {
       printf " Enter name :"
       getline na < "-"
       printf " Enter marks :"
       getline marks < "-"
       printf " Your name and marks: %-10s %d \n", na, marks
       if (marks >=66)
```

```
        printf " You got Distinction! Congratulations!!!\n"
    else if (marks >=60)
            printf " You got First Class! Congratulations!!!\n"
        else if (marks >=45)
            printf " You got Second Class! Congratulations!"
    }
```

FUNCTIONS

- awk has several built-in functions, performing both arithmetic and string operations.
- The parameters can be passed to a function in C-style.
- When a function is used without any parameter, the symbols ( ) need not be used.
- awk even enables you to define your own functions.
- Function Description

int(x)	Returns integer value of *x*
sqrt(x)	Returns square root of *x*
length	Returns length of complete record
length(x)	Returns length of *x*
substr(sl, s2,s3)	Returns portion of string of length s3, starting from position s2 in string *s1*
index(s1, s2)	Returns position of string s2 in string s1
split(s, a)	Splits string s into array *a;* optionally returns number of fields
system("cmd")	Runs UNIX command *cmd,* and returns its exit status

emp.lst file

2365 | barun sengupta | director | personne1 | 11/05/47 | 7800

3564 | sudhir Agarwal | executive | personnel | 06/07/47 |7500

4290|jayant Choudhury | executive | production | 07/09/50 | 6000

9876 | jai Sharma | director | production | 2/03/50 | 7000

The length and length( ) Functions

- length() determines the length of its argument, and if no argument is present, then it assumes the entire line as its argument. You can use the-following statement to locate records whose length exceeds 57 characters:

  awk –F “|" 'length> 57' emp.lst

- Observe that we used length without any argument. But we provide argument
- The following program selects those people who have short names:

  awk -F"|" ‘ length($2) < 11' emp.lst # $2 is *the second field*

The index() and substr() Functions

- The index() function determines the position of a string within a larger string. This function is especially useful in validating single character fields. If you have a field which can take the values a, b, c, d or e, you can use this function to find whether this single character field can be located within the string abcde:

  x = index("abcde", “b”)

  This returns the value 2.

- The substr() function takes three arguments. The first argument s1 represents the string to be used for extraction, s2 represents the starting point of extraction, and s3 indicates the number of characters to be extracted.

- Because string values can also be used for computation, the returned string from this function can be used to select those born between 1946 and 1951.

$ awk –F ”|” 'substr($5,7,2) > 45 && substr($5,7,2) < 52' emp.lst

2365 | barun sengupta | director | personne1 | 11/05/47 | 7800 | 2365

3564 | sudhir Agarwal | executive | personnel | 06/07/47 |7500 | 2365

9876 | jai Sharma | director | production | 2/03/50 | 7000 | 9876

The split() Function :

- split(), which also takes three arguments, breaks up a string (first argument) into fields, and stores them individually in an array (second argument). The delimiter is used as the third argument. The conversion of the date format to yyyymmdd is in a simpler way:

$ awk –F “|” '{split($5,ar, "/") ; print

“19"ar[3]ar[2]ar[1]}` emp.lst

19521212

19501203

19431904

- This method is superior because it explicitly picks up the fifth field, whereas sed just transformed the only date field that it could match.