Capgemini

UNIX

Lesson 13 : AWK Programming

Lesson Objectives

- At the end of the session you is able to understand:
  - How to write simple *awk* scripts.

7.1: Advanced Filter - awk
Introduction

AWK

- Based on *pattern matching* and *performing action.*
- We have seen how *grep* uses pattern.
- Limitations of the *grep* family are:
  - No options to identify and work with fields.
  - Output formatting, computations etc. is not possible.
  - Extremely difficult to specify patterns or regular expression always.
- AWK overcomes all these drawbacks.

7.1: Advanced Filter - awk
Introduction

AWK

- Named after Aho, Weinberger, Kernigham.
- As powerful as any programming language.
- Can access, transform and format individual fields in records.

7.1: Advanced Filter - awk
Contents

Syntax:

- awk <options> ‘line specifier {action}’ <files>
- Example:
  - awk '{ print $0 }' emp.data
- This program prints the entire line from the file *emp.data*.
- $0 refers to the entire line from the file *emp.data.*

7.1: Advanced Filter - awk
AWK variables

Variable List:

- **$0:** Contains contents of the full current record.
- **$1..$n:** Holds contents of individual fields in the current record.
- **NF:** Contains number of fields in the input record.
- **NR:** Contains number of record processed so far.
- **FS:** Holds the field separator. Default is *space* or *tab.*
- **OFS:** Holds the output field separator.
- **RS:** Holds the input record separator. Default is a new line.
- **FILENAME:** Holds the input file name.
- **ARGC:** Holds the number of Arguments on Command line
- **ARGV:** The list of arguments

7.1: Advanced Filter - awk Example

awk '{ print $1 $2 $3 }' emp.data

- This prints the *first, second* and *third* column from file *emp.data.*

awk '{ print }' emp.data

- Prints all lines (all the fields) from file *emp.data.*

7.2 AWK variables
Overview

Line specifier and action option are optional, either of them needs to be specified.

If line specifier is not specified, it indicates that all lines are to be selected.

{action} omitted, indicates print (default).

Fields are identified by special variable $1, $2, ….;

Default delimiter is a contiguous string of spaces.

Explicit delimiter can be specified using -F option

- Example: awk -F "|" '/sales/{print $3, $4}' emp.lst

Regular expression of *egrep* can be used to specify the pattern.

7.2 AWK variables
Examples

awk '$3 > 0 { print $1, $2 * $3 }' emp.data

- Checks for $3 (third field) value. If it is greater than 0, then it prints the first column and the multiplication of the second and the third columns.

7.2 AWK variables
Examples

Line numbers can be selected using NR built-in variable.

- awk -F "|" 'NR ==3, NR ==6 {print NR, $0}' emp.lst
- awk '{ print NF, $1, NR }' emp.data
- awk '$3 == 0' emp.data
- awk '{ print NR, $0 }' emp.data
- awk ' $1 == "Susie" ' emp.data

7.3 Logical and Relational operators
Logical and Relational operator

Logical Operator &&, ||

```
$awk - F “|” ‘$3 ==“director” || $3 == “chairman”{printf “%-20s”,$2}’ emp.lst
```

Relational Operators : <, <=, ==, !=, >=, >, ~, !~

```
$awk -F “|” ‘$6>7500 {printf “%20s”, $2}’ emp.lst
$awk -F “|” ‘$3 == “director” || $6>7500 {print $0}’ emp.lst
```

7.3 Logical and Relational operators
Logical and Relational operators

== tries to find perfect match.

- String may have trailing spaces.
- To overcome this you can use “~” and “!~” (match and negate of match) with the regular expression.
- Example:

```
$awk -F “|” ‘$2~/director/||$2~/g.m/{printf $0}’ emp.lst
    $3 ~/^g.m/    # Beginning with g.m
```

7.4 Arithmetic operators
Arithmetic operators

Computation on numbers is done:

- +, -, *, /, % operator available.
- No type-declaration for variables.
- Variables are initialized to zero.
- ++, -- (Increment / Decrement variable)
- +=, -= (Add/Sub expression to preceeding variable)
- *=, /= (Mult/Div expression to preceeding variable)
- %= (Modulus after dividing preceeding variable)

```
$awk -F "|" '$3 == "director" || $6>7500 {
        >kount = kount+1
        >printf "%3d%-20s\n", kount,$2}' emp.lst
```

Can also use C constructs like kount++, Kount+=sum etc.

7.5 awk command
awk command

-f option

- awk program can be written in a separate file and used in awk.
  - Example:

```
$awk -F "|" -f emp.awk    emp.lst
```

- Single quoted contents are written in this file without quotes.

```
$cat emp.awk
        $3 == "Director" {print $2,$4,$5}
        $
```

7.6: BEGIN and END Section
BEGIN and END Section

BEGIN and END Section:

- Format: (i) BEGIN {action} (ii) END {action}

```
awk<options> 'BEGIN {action}
                    line specifier {action}
                    END {action}' <files>
```

7.6 BEGIN and END Section Example

```
$cat emp.awk

        BEGIN {  printf "\n\t Employee details \n\n"

        }

        $6>7500{

        # increment sr. no. and sum salary

        kount++; tot+=$6

        printf "%d%-20s%d\n", kount, $2, $6

        }

        END {  printf "\n The Avg. Sal. Is %6d\n", tot/kount

        }

        $awk -F "|" -f emp.awk emp.lst
```

7.6 BEGIN and END Section Example

Example 1:

```
awk '
BEGIN
 {
  print "NAME   RATE  HOURS";
  print ""
  }
  {
 print
}
' emp.data
```

Example 2:

```
awk '
END
 {
    print NR, "employees"
 }
' emp.data
```

O/P : 6 employees

7.7 Positional parameters and shell variable
Positional parameters and shell variable

- **Requires entire awk command to be in the shell script.**
- **Differentiate positional parameter and field identifier**
  - Positional parameter should be single-quoted in an awk program.
    - Example: $3 > '$1'

7.8 Built in functions
Numeric Functions

int(x)
- Returns integer value of x.

sqrt(x)
- Returns square root of x.

index(s1,s2)
- Returns the position of string s2 in s1.

length()
- Returns length of the argument.

7.8 Built in functions
String

substr(s1,s2,s3)

- Returns portion of string of length - s3,starting position s2 in string s1.

tolower(str)

- Convert to lower case.

toupper(str)

- Convert to upper case.

split(s,a)

- Split the string s into array a.
- Optionally it returns the number of fields.
  - Example:

```
awk -F":" 'split{$5,arr,"/";print "20arr[3]arr[2]arr[1]}'  emp.lst
```

```
awk -F":" 'substr($5,7,2) > 45 && substr($5,7,2) < 52' emp.lst
```

- Splits 5th Field (date) into an array and prints in form “YYYYMMDD”
- Retrieves those born between 1946 and 1951.

7.9: Control statements
Control statements

```
if (<condition>)
{
        <Block of statements>
}
else
{
        <Block of statements>
}
```

```
While (Condition)
{
        <Block of statements>
}
for (<expr1>;<cond1>;<expr2>)
{
                <Block of
statements>
}
```

7.9: Control statements
Example

```
awk '
{
   if (NF != 3)   {
      print $0, "number of fields is not equal to 3“
   if ($2 < 3.35)   {
      print $0, "rate is below minimum wage"    }
   if ($2 > 10)   {
      print $0, "rate exceeds $ 10 per hour"    }
   if ($3 < 0)   {
      print $0, "negative hours worked"    }
   if ($3 > 60)   {
      print $0, "too many hours worked"    }
}
' emp.data
```

7.10 Pattern Matching
Pattern matching

Print all lines with pattern Susie:

```
awk ' /Susie/ ' emp.data
```

Print all lines in which 4th field matches with pattern Asia:

```
awk’ {    if ($4 ~ /Asia/)
                    {   print    }
                          }
                ' countries.data
```

7.10 Pattern Matching Example

Print any line that matches exactly three digits

```
awk ‘
{
 if (/^[o-9][o-9][o-9]$/)
  {
    print
  }
}
' numbers.data
```

print any line that consists only digits

```
awk '
 {
   If (/^[o-9]+$/)
   {
     print
   }
}
' numbers.data
```

SUMMARY

- AWK is based on pattern matching and performing action.
- Commands enclosed in BEGIN section gets executed first.
- Then. it performs main action part on all records.
- Commands enclosed in END section gets executed.

Review Questions

❖In ------ section report header can be printed

❖------ section helps to print totals

❖Print $0 prints whole record

- TRUE
- FALSE