



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.

The *awk* command, named after its authors Aho, Weinberger and Kernighan, is one of the most powerful utilities for text manipulation. It combines features of many other filters. It can access, transform and format individual fields in a record – it is also called as a report writer. It can accept regular expressions for pattern matching, has “C” type programming constructs, variables and in-built functions. In fact, *awk* is nearly as powerful as any other programming language. However, *awk* programs are generally slow, if any alternative commands are available to do the same use them rather than using AWK.

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.



Copyright © Capgemini 2015. All Rights Reserved

4

Filters accept some data as input, perform some manipulation on it and produce some output. Some simple filters have been discussed in the previous chapters. This chapter discusses advanced filter –*awk*.

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*.



Copyright © Capgemini 2015. All Rights Reserved 5

Advanced Filter awk.

The general syntax of the awk command is:

```
awk <options> 'line specifier {action}' <file(s)>
```

Simple awk Filtering

Following is an example of a simple awk command:

It prints all lines from file *books.lst* in which pattern 'Computer' is found.

```
$ awk '/Computer/ {print}' books.lst
```

Output:

1001 Learning Unix	Computers	01/01/1998	575
1003 XML Unleashed	Computers	20/02/2000	398
1004 Unix Device Drivers	Computers	09/08/1995	650
1007 Unix Shell Programming	Computers	03/02/1993	536

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.



Copyright © Capgemini 2015. All Rights Reserved 8

The line specifier uses a context address to specify the lines that need to be taken up for processing in the action section. If the line specifier is missing, then the action is applicable to all lines of the file.

In the action part, statement {print} indicates that selected lines are to be printed. The statement {print} is equivalent to {print \$0} - \$0 being the variable for the entire line. The awk command is also capable of breaking each line into fields – each field is identified as \$1, \$2 etc.

For the purpose of identification of fields, awk uses a contiguous string of spaces as field delimiter. However, it is possible to use a different delimiter. This is specified using the -F option in awk.

\$ awk -F "|" "/Computer/ {print \$2,\$5}' books.lst

Output:

Learning Unix	575
XML Unleashed	398
Unix Device Drivers	650
Unix Shell Programming	536

For pattern matching, awk uses regular expressions of *egrep* variety.

\$ awk -F "|" "/XML|Unix/ {print \$2, \$5}' books.lst

Output:

Learning Unix	575
XML Unleashed	398
Unix Device Drivers	650
XML Applications	630
Unix Shell Programming	536

It is possible to specify line numbers in file using the inbuilt NR variable. Also, awk can use the C-like *printf* statement to format the output.

\$ awk -F "|" '\$1=="1002" {printf "%2d,%-20s",NR,\$2}' books.lst

Output:

2,Moby Dick

The -f option of awk is useful if you wish to store the line specifier or action in a separate file.

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.



Copyright © Capgemini 2015. All Rights Reserved 9

The logical operators `||` (or) and `&&` (and) are used by the `awk` command to combine conditions in the line specifier. A relational operator can be used in the line specifier, also in the action component.

The operators `==` (equal) and `!=` (not equal) can handle only fixed length strings and not regular expressions. To match regular expressions, `~` (match) and `!~` (negate) are used. The characters `^` and `$` can be used for looking for a pattern in the beginning and end of field.

To work with numbers, operators like `<` (less than), `>` (greater than), `<=` (less than or equal), `>=` (greater than or equal), `==` (equal) and `!=` (not equal) can be used.

It is possible to perform computations on numbers using C like arithmetic operators (`+`, `-`, `*`, `/`, `%`, `++`, `--`, `+=` etc).

\$ `awk -F"|"` `'/Unix/ && $5 < 600 {printf "%s,%d\n",$2,$5}' books.lst`

Output:

```
Learning Unix          ,575
Unix Shell Programming ,536
$ awk -F"|"' '$2=="Learning Unix"' books.lst
$ awk -F"|"' '$2~/Learning Unix/' books.lst
1001|Learning Unix    |Computers   |01/01/1998| 575
$ awk -F"|"' '$5<500 {
> cnt=cnt+1
> printf "%d %s\n",cnt,$2}' books.lst
1 Moby Dick
2 XML Unleashed
```

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



Copyright © Capgemini 2015. All Rights Reserved 10

awk -F "|" 'NR ==3, NR ==6 {print NR, \$0}' emp.lst

In this example, NR represents record number. It prints records from *third* record to *sixth* record.

-F is use to specify field separator.

awk '{ print NF, \$1, NR }' emp.data

This prints number of fields, contents of field 1 and record number for all records.

awk '\$3 == 0' emp.data

It prints all lines in which the value in the third field is 0.

awk '{ print NR, \$0 }' emp.data

It will print record number and record for all records

awk ' \$1 == "Susie" ' emp.data

It prints all lines in which the value in the first field is Susie.

Summary

- AWK is based on pattern matching and performing action.
- Various built in variable of AWK
- Extracting field from file using AWK



Add the notes here.

Review Questions

- Which variable is used to print number of records processed by AWK ?
- How many times action block is executed in AWK?
- Print \$0 prints whole record
 - TRUE
 - FALSE

