CHARUTAR VIDYA MANDAL'S
SEMCOM
Vallabh Vidyanagar

Faculty Name: Ami D. Trivedi
Class: TYBCA (Semester-V)
Subject: US05EBCA01 (Basics of UNIX Operating System)

*UNIT – 4 (Shell Scripting using Filters)

DEFINITION OF A FILTER

Filter is a UNIX command which takes a character stream as standard input, manipulate its contents and generates a similar stream as standard output.

OR

A filter is a program which can receive a flow of data from standard input, process (or filter) it and send the results to the standard output.

The shell's redirection and piping features can be used with these commands.

A filter is unaware of the source and destination of its data.

BASIC FILTERS

1. grep family

Full form of grep is: **g**lobal search for **r**elational **e**xpression & **p**rint

UNIX has a special family of commands for handling search requirements. Principal member of this family is the **grep** command. grep is easy to use.

The grep command can be used to search for a specified pattern in one or more files, and displays the matching output on standard output.

Output of grep depends on the options used. grep can display

- Lines containing the selected pattern
- Lines not containing the selected pattern (-v)
- Line numbers where the pattern occurs (-n)
- Number of lines containing the pattern (-c)
- Filenames where the pattern occurs (-l) and so on.

Syntax: grep options pattern filename(s)

Syntax of grep treats its first argument as the pattern and the rest as filenames. When grep fails in locating the pattern, it silently returns the prompt.

Following is a list of options (flags) that can be used with grep command:

-v	to find lines not matching the specified pattern.
-n	to display the relative line number before each line in the output.
-c	to display the count of lines in which the pattern was found without displaying the lines.
-l	to list just the filenames in which the specified pattern has been found.
-e	to match multiple patterns.
-E	using | delimiter we can locate multiple patterns from the file.
-i	to search, ignoring the case of the letter.
-x	to match the patterns exactly with a line.

.	It matches a single character.
*****	It matches zero or more occurrences of the previous character to *.
[ ]	Any one character from the square bracket.
^	Used for matching at the end of a line
$	Used for matching at the beginning of a line
( )	(x1|x2)x3 – Matches x1x3 or x2x3

We will apply different grep option on following file with name “stud.lst”. File contain 7 records and 4 fields namely student id (combination of class and roll number), name of student, name of event, pointes scored in an event.

```
fy01   |   nil o’brien          |sports   |100
sy01   |   anuj roy             |music    |  50
sy02   |   sam o’bryan          |dance    |125
ty01   |   virali sengupta      |dance    |175
fy02   |   mansi dutta          |sports   |  75
fy03   |   dev dasgupta         |music    |150
ty02   |   shabd datta          |sports   |  25
```

1) To display lines containing the pattern

$ grep sports stud.lst

```
fy01   |   nil o’brien          |sports   |100
fy02   |   manas dutta          |sports   |  75
ty02   |   shabd datta          |sports   |  25
```

Pattern is not quoted here. Quoting is required only when search string contains multiple words or it uses any of the shell character like *, ? and so on.

- **grep with multiple files**

$ grep music stud.lst stud1.lst

```
stud.lst:    sy01   |   anuj roy          |music    |  50
stud.lst:    fy03   |   dev dasgupta      |music    |150
stud1.lst:   pg01   |   arth rai          |music    |200
stud1.lst:   pg07   |   viraj mittal      |music    |  50
```

When grep is used with multiple filenames then every line is prefixed by its filename.

2) To count occurrences of the pattern (-c)

```
$ grep -c sports stud.lst
3
```

The **-c (count)** option here counts the occurrences of pattern sports. Answer will be in number.

If we use this command with multiple files then filename will be prefixed to the line count.

```
$ grep –c sports stud*.lst
stud.lst:3
stud1.lst:2
```

3) To display lines not containing the pattern (-v)

$ grep –v sports stud.lst

```
sy01   |   anuj roy             |music    |  50
sy02   |   sam o’bryan          |dance    |125
ty01   |   virali sengupta      |dance    |175
fy03   |   dev dasgupta         |music    |150
```

The **-v (inverse)** option selects all the lines that does not contain pattern sports here.

4) Displaying line numbers (-n)

$ grep -n music stud.lst

```
2:sy01   |   anuj roy          |music    |  50
6:fy03   |   dev dasgupta      |music    |150
```

The **-n (number)** option can be used to display the line numbers containing the pattern along with the lines.

Line numbers are shown at the beginning of each line. Line number and actual line is separated by **:**.

If we will use this option with multiple filenames, then we will have 2 additional fields – filename and line number.

$ grep -n music stud*.lst

```
stud.lst:2:sy01    |   anuj roy          |music   |  50
stud.lst:6:fy03    |   dev dasgupta      |music   |150
stud1.lst:1:pg01   |   arth rai          |music   |200
stud1.lst:4:pg07   |   viraj mittal      |music   |  50
```

5) **Displaying filenames (-l)**

$ grep -l music *.lst

```
stud.lst
stud1.lst
```

The **-l (list)** option displays only names of files where a pattern is found.

6) **Ignoring Case (-i)**

$ grep -i 'SAM' stud.lst

```
sy02   |   sam o'bryan      |dance   |125
```

When we look for a name but not sure about case, grep offers –i (ignore) option. This makes the match case insensitive.

7) **Matching multiple pattern (-e)**

$ grep -e dutta -e data stud.lst

```
fy02   |   mansi dutta      |sports   | 75
ty02   |   shabd datta      |sports   | 25
```

We can match multiple patterns with one grep command. Every pattern should start with –e option.

We can store all patterns in a separate file, one pattern per line. Now use –f option of grep.

$ grep –f pattern.lst stud.lst

8) **Matching multiple pattern (-E)**

We can use –E option with delimiter | (pipe) to locate multiple patterns.

$ grep -E 'sengupta | dasgupta' stud.lst
OR
$ grep -E '(sen | das) gupta' stud.lst

```
ty01   |   virali sengupta   |dance   |175
fy03   |   dev dasgupta      |music   |150
```

9) **Matching pattern exactly with a line (-x)**

If you want to search for a pattern that is the only string in a line, use the -x option.

file1	**$ grep -x "course" file1**
new course course title course	course

10) Rectangular brackets [ ]

$ grep “d[au]tta” stud.lst

```
fy02   |   mansi dutta        |sports   |  75
ty02   |   shabd datta        |sports   |  25
```

When we enclose group of characters within a pair of rectangular brackets, it matches for a single character in the group. [au] matches either an **a or u**.

We can use **range** in [ ]. For e.g. **[0-9] or [a-m]** etc. ASCII value of the character at left hand side of **-** must be lower than on right hand side character.

We can **negate** the character class using **^ (caret)**. For e.g. [^a-z]. It matches all characters other than a-z.

11) The *

The * (asterisk) refers to immediately preceding character. It matches zero or more occurrences of the previous character. In other words, the previous character can occur many times or not at all.

Consider 3 patterns : aarval, agarval and aggarval. First does not contain g, second contains 1 g and third contains 2 g. Use * after character g in the pattern.

$ grep “ag*arval” stud.lst

12) The Dot (.)

A dot matches a single character. The pattern **f…** will match a four character pattern beginning with **f**.

13) ^ (caret)

^ is used for matching at the beginning of line.

$ grep “^f” stud.lst

```
fy01   |   nil o’brien        |sports   |100
fy02   |   mansi dutta        |sports   |  75
fy03   |   dev dasgupta       |music    |150
```

14) $

$ is used to match at the end of pattern.

$ grep “1..$” stud.lst

```
fy01   |   nil o’brien        |sports   |100
sy02   |   sam o’bryan        |dance    |125
ty01   |   virali sengupta    |dance    |175
fy03   |   dev dasgupta       |music    |150
```

This command will display all lines where points are greater than or equal to 100.

15) Quoting in grep

$ grep ‘mansi dutta’ stud.lst

```
fy02   |   mansi dutta        |sports   |  75
```

When we use multiword string as the pattern, we must quote the pattern. If we will not use quotes then it will search for mansi but dutta will be considered as a file name. Fails to open such a file will display error message.

Note: To search a pattern containing single quote for e.g. **nil o’brien** we need to enclose it in double quotes. Single quote do not protect single quotes.

2. expr (Computation and String handling)

expr command combines two functions in one:

1) Perform arithmetic operations on integers.
2) Manipulate strings

Arithmetic Functions

expr can perform the basic four arithmetic operations as well as the modulus (remainder) function

Syntax: expr op1 math-operator op2

Examples:

```
$ echo `expr 6 + 3`        $ x=3 ; y=5          Note: Multiple assignments
9                          $ expr $x - $y
$ echo `expr 6 \* 3`       -2
18                         $ expr $y / $x
$ expr 3 + 5               1
8                          $ expr 13 % 5
$ expr 3 \* 5              3
15
```

For e.g.

```
$ echo "expr 6 + 3"    # It will print expr 6 + 3
$ echo 'expr 6 + 3'    # It will print expr 6 + 3
```

The operand (+, - , * etc) must be enclosed on both side by **whitespace**. Multiplication operand (*) has to be escaped. Otherwise shell will consider it as filename metacharacter.

expr can handle only integers. So division operation generates (gives) only integer part.

expr is often used with command substitution to assign a variable. We need to enclose expr in back quotes while doing command substitution. For e.g.

```
$ x=6 ; y=2 ; z=`expr $x + $y`        Note: expr enclosed in back quotes
$ echo $z
8
```

While evaluating an expression expr performs various arithmetic operations according to following priorities:

/, *, % - First priority
+, -   - Second priority

String Handling

expr's string handling facilities are not elegant. expr can perform 3 important string functions:

1) Find length of string
2) Extract a substring
3) Locate the position of a character in a string

bc – the calculator

If you want to perform simple arithmetic expression in UNIX, use the bc command. It is a text-based calculator.

bc performs only integer computation and truncates the decimal portion. e.g. 9/5 will give answer 1. To enable floating-point computation, you have to set **scale** to the number of digits of precision before you type the expression.

```
$ echo "scale=2; 17 / 7"|bc
2.42
```

3. sed (The Stream Editor)

sed is a multipurpose tool which combines the work of several filters. It is derived from **ed** line editor (original editor of UNIX). It is designed by Lee McMahon.

sed is used to perform non interactive operations. It acts on a data stream. That why known as stream editor.

Everything in **sed** is an instruction. An instruction combines an **address** for selecting lines with an **action** to be taken on them.

Syntax: sed options 'address action' file(s)

The address and action are enclosed within single quotes. Action can be simply display or editing like insertion, deletion or substitution of text.

We will apply different sed option on file "stud.lst". (Refer stud.lst of grep command)

3.1 Line Addressing

Addressing in sed is done in 2 ways:

- By line number (like 3,7p)
- By specifying a pattern (like /From:/p)

$ sed '3q' stud.lst Note: Quit after line number 3

```
fy01  |  nil o'brien      |sports  |100
sy01  |  anuj roy         |music   | 50
sy02  |  sam o'bryan      |dance   |125
```

sed also uses p (print) command to print the output.

$ sed '1,2p' stud.lst

```
fy01  |  nil o'brien      |sports  |100
fy01  |  nil o'brien      |sports  |100
sy01  |  anuj roy         |music   | 50
sy01  |  anuj roy         |music   | 50
sy02  |  sam o'bryan      |dance   |125
ty01  |  virali sengupta  |dance   |175
```
……..more lines where each line displayed only once

By default, sed prints all lines on standard output in addition to the lines affected by the action. So the addressed lines (first two) are displayed twice. But we don't want this.

1) Suppressing Duplicate Line Printing (-n)

To overcome the problem of printing duplicate lines, we should use –n option whenever we use p command.

$ sed –n '1,2p' stud.lst

```
fy01  |  nil o'brien      |sports  |100
sy01  |  anuj roy         |music   | 50
```

To select the last line of the file, use **$**.

$ sed –n '$p' stud.lst

```
ty02  |  shabd datta      |sports  | 25
```

2) Reversing Line Selection Criteria (!)

We can use sed's negation operator (!) with any action. So selecting first two lines is same as not selecting lines 3 through end.

$ sed –n '3,$!p' stud.lst Note: Don't print lines 3 to the end

3) Selecting Line from the Middle

sed can also select lines from the middle of a file. This is not possible with head and tail command.

$ sed –n ‘3,4p’ stud.lst

```
sy02   |   sam o’bryan       |dance   |125
ty01   |   virali sengupta   |dance   |175
```

4) Selecting Multiple Sections

Sed is not restricted to selecting contiguous groups of lines. By placing each instruction on a separate line, we can select multiple sections.

$ sed –n ‘1,2p
7,9p
$p’ stud.lst

We can place all these instructions in a single line by using –e option.

$ sed -n -e ‘1,2p -e 7,9p -e $p’ stud.lst

3.2 Context Addressing

The second form of addressing allows us to specify a pattern rather than line numbers. This is known as context addressing. Here, pattern has a **/** on both the side.

$ sed –n ‘/From: /p’ mail.lst

```
From: Bharat <bharat@yahoo.com>
From: Ganga <ganga@gmail.com>
………..
```

$ sed –n “/o’br[iy][ae]n/p” stud.lst

```
fy01   |   nil o’brien       |sports   |100
sy02   |   sam o’bryan       |dance    |125
```

$ sed –n “/o’br[iy][ae]n/p /dutta/p” stud.lst Note: Match either o’bryan or o’brien or dutta

3.3 Substitution

sed’s strongest feature is substitution. It is achieved through **s (substitute)** command.

It allows us to replace a pattern in its input with something else.

Syntax: [address]s / expression1 / string2 / flag

Here expression1 is replaced by string2 in all lines specified by the address. If the address is not specified, the substitution will be performed for all lines containing expression1.

To replace all **| (pipe) with :**

$ sed ‘s / | / : /’ stud.lst | head -2

```
fy01   :   nil o’brien       |sports   |100
sy01   :   anuj roy          |music    |  50
```

Notice that only first (left most) occurrence of | has been replaced. We need to use g (global) flag to replace all the pipes. g flag is used for global substitution.

$ sed ‘s / | / : /g’ stud.lst | head -2

```
fy01   :   nil o’brien       :sports   :100
sy01   :   anuj roy          :music    :  50
```

To limit the substitution to portion of a file:

$ sed ‘1,3s / | / : /g’ stud.lst

```
fy01   :  nil o’brien        :sports  :100
sy01   :  anuj roy           :music   :  50
sy02   :  sam o’bryan        :dance   :125
```

Substitution can be performed for any string.

$ sed -n ‘s / br[iy]an / brown /p’ stud.lst

All brian and bryan will be replaced by brown. Due to –n, lines will be displayed also.

- **Performing multiple substitutions**

$ sed ‘s / br[iy]an / brown /g
s/ [mp]ike / hike/g’ stud.lst

3.4 <u>Writing to Files (w)</u>

The w (write) command writes the selected lines to a separate file. We can save the lines contained within <FORM> and </FORM> tags in a separate file.

$ sed ‘/<FORM> /, /<\ /FORM> /w newform.html’ form.html

Here, the form contents are extracted and saved in newform.html.

In </FORM> tag, / needs an escaping because / is sed’s pattern delimiter.

We can save all form segments from all html files in a single file:

$ sed ‘/<FORM> /, /<\ /FORM> /w newform.html’ *.html

We can search for multiple patterns and store the matched lines in different files – all in one shot.

```
$ sed ‘/<FORM> /, /<\ /FORM> /w newform.html
       /<FRAME> /, /<\ /FRAME> /w newframe.html
       /<TABLE> /, /<\ /TABLE> /w newtable.html’ *.html
```

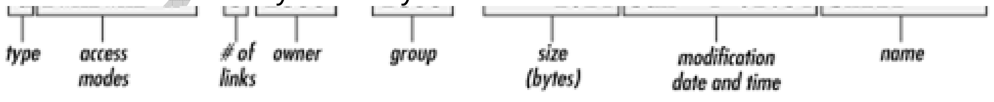
<u>PROCESSING THE OUTPUT OF COMMANDS LIKE ls, ps, who, etc.</u>

Commands like ls, ps, who etc. gives the output in column format. i.e. Output of such commands contain multiple columns providing useful information for the user.

<u>ls (listing)</u>

ls command is used to display all files and directories.

```
$ ls -l
total 2
 -rw-r- -r- -    1   ty01     tybca    8         25 Feb 10:00    file1
 -rw-r- -r- -    1   ty01     tybca    8         26 Feb 12:00    file2
```



<u>ps (process status)</u>

like a file, process has many attributes. ps command is used to display the attributes of a process. By default ps displays the processes associated with a user at the terminal.

```
$ ps
 PID              TTY       TIME   CMD
 100             tty03   00:00:01   sh
 200             tty03   00:00:00   ps
Process id    Terminal      Time       Command
```

who

UNIX system is used my multiple users. We may be interested in knowing the people who are currently using the system.

$ who

```
ty01      tty01      Jan 01      10.:00
ty02      tty02      Jan 01      12:00
ty03      tty03      Jan 01      11:00
Users     Terminal   Login date  Login time
```

To Process output of commands with columnar output, we can use:

1. cut command
2. awk utility

1. cut command

We can use the cut command in a Unix / Linux command pipeline. For example, we can use the cut command with the ls command to get a list of filenames in the current directory, like this:

$ ls –l | cut –c44-

This command can be read as:

- Run the "ls -l" command.
- Pipe the output of that command into the cut command.
- The cut command prints everything from each line starting at column 44 through the end of the line (-c44-).

The -c option lets you deal with columns or character positions. In this example all the output from each line starting in column 44 and going to the end of the line. If we want only columns 40 through 50, we can specify like:

$ ls –l | cut –c44-50

That particular command doesn't make much sense in the real world, since filenames can all be different lengths, but it does show how to use a range of columns with the Linux cut command.

2. awk utility

$ ls -l | awk '{print $1, $2, $3, $4}'

where $1, $2 , $3 are the fields that you wish to print. Awk will delimit each field in the listing of "ls -l" by a space.

Each successive field can be printed by providing the field number. So print $1, $2, $3, $4 will print out the first 4 fields.

PROCESSING DATA IN TEXT FILES (FIXED-WIDTH FORMAT AND DELIMITED FORMAT)

Delimited file format

Data text available in raw, misaligned and unformatted way may be difficult to read.

In this raw text formatting, selected keyboard characters (or "delimiters") are used to maintain the data's tabular organization.

To separate the data normally which are contained in different columns and text fields, we can insert comma, pipe, colon, hash or tabbed spaces etc.

The meaning of the data values are kept intact although the file formatting is represented differently.

Unfortunately, not all applications will interpret the tabbed spaces and comma characters as proper delimiters. It happens when such delimiters become a part of your data. For e.g. in a file with comma is separator and comma will come as part of currency value.

Consequently, your data can be misinterpreted and your text file tasks impossible.

Fixed width format

A fixed record format contains data in each line of textual information (known as a "record"). Record is divided into fields of fixed byte lengths.

When you create a fixed record file, you determine how much space is allocated for each field in the first row.

The spacing remains consistent down the entire column. If the vertically aligned values fall short of the required field length, the field will be padded with trailing spaces or, if the values exceed it, will be cut off.

These inflexible fixed lengths eliminate the need for potentially misinterpreted delimiters that can make your work difficult.

Creating a fixed record file, however, is never difficult. It only needs the exact calculation of column positions to access the different fields.

cut command to process data in text files (FIXED-WIDTH format & DELIMITED format)

We can use the cut command to extract data from each line of a text file. cut command will split the file vertically.

This command can be used for a file that contains data records so that each line consists of one or more fields separated by delimiters or tab.

Following is a list of flags that can be used with the cut command:

-c <character list>	to specify a list of characters to be cut from each line.
-f <field list>	to specify a list of fields to be cut from each line.
-d character	to specify a delimiter

Example for Delimited file format

Let us assume that we have a file called stud.lst. File has 4 fields separated by a pipe delimiter. We will apply different cut option on file “stud.lst”. (Refer stud.lst of grep command)

```
fy01  |  nil o’brien      |sports  |100
sy01  |  anuj roy         |music   | 50
sy02  |  sam o’bryan      |dance   |125
ty01  |  virali sengupta  |dance   |175
fy02  |  mansi dutta      |sports  | 75
fy03  |  dev dasgupta     |music   |150
ty02  |  shabd datta      |sports  | 25
```

In this file, the fields are separated by | (pipe) characters.

If you want to extract the first field, use the following command:

```
$ cut –d “|” -f1,2 stud.lst
fy01  |  nil o’brien
sy01  |  anuj roy
sy02  |  sam o’bryan
………………..
```

Here pipe delimiter is specified in cut command with –d option. –f1,2 means we want to extract field number 1 and 2.

If we want to extract may contiguous fields from a file:

$ cut –d "|" –f1-5,8 file1

Here it will extract total 6 fields: 1 to 5 and 8th field from file1.

More examples:

$ cut –d "|" –f1-3,7- file1 Note: Field 1,2,3,7 to last field.

Example for Fixed width file format

Let us assume that we have a file called marks.lst. File has 5 fields. All fields are having fixed number of bytes. They are aligned to create a column format look. We will apply different cut option on file "marks.lst".

```
11  Ansh   120  100  90
22  Binoy  80   50   80
33  Rahat  40   150  50
44  Malav  160  125  70
```

In this file, the fields are stored at fixed positions. First 2 positions are reserved for roll number. Position 3 to 8 reserved for name, 9 to 11 for mark 1, 13 to 16 for mark 2 and 18 to 21 for mark 3.

If you want to cut first field (position 1 and 2), use the following command:

$ cut –c1-2 marks.lst OR $ cut –c-2 marks.lst Note: Position 1 can be omitted.

```
11
22
33
44
```

If you want to cut multiple fields from different positions, use the following command:

$ cut –c1-10, 15-20, 50-55 file1

If the position 55 is last column position then it can be omitted as well.

$ cut –c-10, 15-20, 50- file1

Note:

We may sometimes need

1. To generate attractive formatted output or
2. To perform processing on different fields of every / selected record.

In this situation, a while read loop construct can be used. In this loop, every record will be read in a sequence from a given data file into a string.

Now, we can apply cut command to split the string into fields. These extracted fields can be stored into local variable of script for further processing.

Following script will extract fields from every record of marks.lst. Total of mark 1, 2 and 3 will be calculated and display on screen as an output.

Example:

```
sum=0
while read s1
do
        rno=$(echo $s1 | cut –d “|” –f1)
        sname=$(echo $s1 | cut –d “|” –f2)
        m1=$(echo $s1 | cut –d “|” –f3)
        m2=$(echo $s1 | cut –d “|” –f4)
        m3=$(echo $s1 | cut –d “|” –f5)
        sum=`expr $m1 + $m2 + $m3`
        echo $sum
done < marks.lst
```

Output:

```
310
210
240
345
```

Disclaimer

The study material is compiled by Ami D. Trivedi. The basic objective of this material is to supplement teaching and discussion in the classroom in the subject. Students are required to go for extra reading in the subject through library work.