CE350
Lecture8

PROGRAMMING THE BASH SHELL
PART IV

by İlker Korkmaz and Kaya Oğuz

Programming the bash

- As pointed in syllabus document, there will be 5 lecture weeks to cover the programming with bash scripts.
- This lecture is the 4th part on scripting in bash.

References

- The contents of this lecture are prepared with the help of and based on:
  - the textbook, UNIX Shells by Example (Chapter5, and Chapter6)
  - the tutorial, Advanced Bash-Scripting Guide at http://tldp.org/LDP/abs/html/

Contents of Lecture8

- based on the regular expressions
  - **sed**
  - **awk**

A note:

- Midterm will be held on April 19th

Just a review example on the previous lecture concepts

- How can you print the lines, where the first word begins with K or k, of a given file?
- If you do not achieve to handle the above operation, it is advised that you should dissect the previous lectures.

Regular Expression concept

- A Regular Expression (RE) is a pattern of characters used to be matched in a search.
- The RE may directly be used via the bash operator =~ (RE-match operator).
- grep is one of the popular tools for RE.
- **sed** is another tool, which uses the RE syntax similar as in vi editor. Due to its noninteractive structure, sed lets you type your editing commands at the command line.
- **awk** is yet another tool for RE.

sed

- sed is the streamlined editor
- sed may be used with RE

How sed works

- sed processes a file or an input one line at a time and sends its output to the screen.
- sed stores the processing line in a buffer called **pattern space** or **temporary buffer**.
- sed, like grep, can be used for searching patterns. Moreover,
    - sed has the ability to replace the patterns found.
    - sed has the ability to modify (delete, insert, append) the input file.
    - sed returns exit status of 0 whether or not the pattern is found. It returns nonzero value if sed contains a syntax error.

RE with sed

- Use forward slashes, / /
- to search and print:
    - sed -n '/RE/p' fileName
        - example: sed -n '/love/p' myFile.in
        - without -n, all lines will be printed and also the lines containing the pattern will be reprinted.
- to substitute:
    - sed 's/RE/replacementString/' fileName
        - example: sed 's/love/like/g' myFile.in

sed addressing and commands

- The addressing may be used to decide which lines to edit.
- The addresses may be the decimal numbers, or RE (also called context address), or both. Without specifying an address, all lines are issued by sed.
- Table 5.1 gives the sed commands: d, s, !, r, ...
  - $ sed '/[Ll]ove/d' myfile
    - Lines containing the pattern Love or love are deleted.
  - $ sed '/Love/!d' myfile
    - Lines not containing the pattern Love are deleted.
  - $ sed '1,3d' myFile
    - Lines 1 through 3 are deleted.

How to modify a file with sed

- sed is a nondestructive editor. It will display the modifications on the screen, but it will not change the original file.
- To really effect the modifications in the file, redirection may be used.
- example:
  - $ sed '2,4d' myFile > tempFile
  - $ mv tempFile myFile

Some examples with sed

- As an introduction to sed to be used with RE, try to understand Examples 5.8, 5.9, 5.13, 5.16, and 5.18.
- sed has more abilities according to its commands (r, w, a, i, c, n, y, q, ...), which may be studied by examples available in Chapter 5 of the textbook.

awk

- awk is a UNIX programming language used for manipulating data and generating reports.
- nawk is a newer version of awk and its first initials stand for the authors of the language.
    - Alfred **A**ho, Brian **K**ernighan, and Peter **W**einberger
    - **could have been called as wak, kaw, …; however it is awk.**
- gawk is the GNU version used in Linux.
- awk scans a file or an input line by line, searching for lines that match a pattern and performing the given actions on those matched lines.

awk with RE

- This presentation includes “nawk” command, however you may use “gawk” on Linux.
- formats to be used are:
  - nawk 'pattern' fileName
  - nawk {action} fileName
  - nawk 'pattern {action}' fileName
- examples:
  - $ nawk '/Love/' myFile
    - prints all lines containing Love
  - $ nawk '{print $1}' myFile
    - prints the first fields of the lines in myFile
    - The first field starts at the left margin of the line and is delimited by a whitespace.
  - $ nawk '/Love/{print $1, $2}' myFile
    - What gets printed?

awk examples

- awk may also be used to manipulate the command outputs.
  - $ command | nawk 'pattern {action}'
  - $ df | nawk '$4 > 75000'
    - df reports the free disk space on file systems. If the fourth field of any line of the output of the command df is greater than 75000 (blocks), that line is printed.

How awk works

- Let's dissect the following command:
  - $ awk '{print $1, $3}' myFile.in
  - awk takes a line of input from the file myFile.in and puts the line into an internal variable called $0 (zero). Each line is also called a record.
  - Next, the line is broken into fields or words separated by a whitespace character. Each field is stored in a numbered variable, starting with $1.
  - awk knows about the whitespace characters according to FS, field separator, which may be changed.
  - awk uses print function with comma, which provides a space in the output. The comma is specially mapped to the internal variable OFS, output field separator. The comma generates whatever assigned to OFS.
  - The above processes continue until all lines in the file have been handled.

Records and Fields for awk

- Each line is a record and is terminated with a newline.
- The output record separator, ORS, and the input record separator, RS, can be changed.
- The number of each record is stored in awk's built-in variable, NR. After a record has been processed, NR is incremented.
- $0 variable refers to the entire record.
- Each record consists of fields separated by whitespace (blank spaces or tabs).
- NF variable stores the number of fields within the current processing record.
- Fields are stored in the variables $1, $2, …
- Field separator, FS, holds the value of the input field separator. It may be changed using the -F option.

Using field separator to parse a record

- To parse a line, use nawk -F
- example:
  - $ nawk -F: '/CE 350/{print $1, $2}' file.dat
    - The colon character “:” separates the fields.
  - $ nawk -F'[:\t]' '{print $1, $2}' file.dat
    - Either the colon, the tab, or the space can separate the fields.

Some awk examples

- What gets printed?
  - **tail /etc/passwd | awk -F: '/jane/{print $1, "Group: "$4}'**
  - **Reference:** http://bashshell.net/utilities/using-variables-with-awk/
- An example using awk with -v option:
    - What gets printed?
      - **a=4**
      - **tail /etc/passwd | awk -v myVar=$a -F: '/jane/{print $1, "Group: "$myVar, myVar}'**
- Another reference about awk:
  - http://www.gnu.org/s/gawk/manual/html_node/Patterns-and-Actions.html#Patterns-and-Actions

awk scripting

- -f option may be used to include an input file and also a script file within the command awk.
- format:
  - $ nawk -f myNawkScript myFile
  - You shall dissect **Example 6.53** on page 187 of the textbook.

awk programming

- awk is actually a programming language.
- The syntax of awk is similar to C programming language.
- The awk utility provides the UNIX/Linux shell programmers with many features, which are mainly explained in Chapter 6 of the textbook.