Shell Programming

- Sameer Khan

Agenda

- **Handling error code and return code**
  - Managing various return code like $?,$$ and $! Etc
  - Suppressing error and printing user defined messages.
- Reading and manipulating text files
- Using awk, sed , grep, tr in script (Not full details)
  - Using awk to break cols and fetching particular match.
  - Using awk to filer based on user delimiter.
  - Using sed to manipulate (search and replace pattern) file.
- **Building complex syntax**

- **Enabling debug mode to debug any issue.**
  - Enabling vx mode to debug error.

Handling error code and return code

- $? - The exit status of the last command executed
- $$ - The process ID of the current script
- $! - The process number of the last background command

- Script2 &
- bpid= $!

Handling error code and return code

- Write a shell program to append contents of **file1** to end of **file2**. Then check if append was successful. – hint: use $?
- Write a shell script to list currently running processes every 5 sec. Write another shell script which executes this script in background, waits for 30 sec and then kills the background script.

Redirection

- “<“ , “>” and “|” are used for redirection of input and output.
- Example

ps -aef | grep CLAGENT

In this example, output of command “ps -aef” is redirected as input for command “grep CLAGENT”

echo “This is an example” **>** sample.txt

In this example, output of echo command is redirected and stored into a file

```
while read line

do
let COUNT=$COUNT+1

done < filename.txt
```

Redirection of error messages

- “1” is considered as std-output
- “2” is considered as std-error
- What will result.txt contain if there is no directory with name sameer
  - find ./sameer/ -name “*.c” > result.txt
- To capture error message in result.txt we will have to redirect std-error also to it
  - find ./sameer/ -name “*.c’ > result.txt 2>&1
- What if I neither want to capture the error nor want it to be displayed on screen ?
  - find ./sameer/ -name “*.c’ > result.txt 2>/dev/null

exercise

- Write a shell script to go to your home directory, find list of all c-files in directory “tutorials”. All error messages from find command should be ignored (not displayed on screen). Redirect this output as input to grep command to find files which have “system” in their names.

Reading and manipulating text files

- awk, cut , sed, tr, grep, tail, head etc are used for reading file contents and manipulating them.
- man <command>
  - man gives detailed description of commands
- Write a script to find line containing word “Error” and print 30 lines from there. – Hint use grep, tail/head

Reading and manipulating text files

http://www.grymoire.com/Unix/Sed.html

- **sed – stream editor**. This is useful in editing files from scripts without manually opening files using editors.
- Format sed option script file
- Example:
- Sed “/NAME/” students.txt

sed -ie “s/NAME/Sameer/g” students.txt

sed s/day/night/ **<** old **>** new

sed -ie “/Sameer/d” students.txt

- "&." It corresponds to the pattern found.
- sed 's/[a-z][a-z]*/REM/' <old >new

  sed 's/[a-z][a-z]*/(&) (&)/' <old >new
- If you want to switch two words around, you can remember two patterns and change the order around:

  sed 's/\([a-z]\)\([a-z]*\)/(& \1 \2) /g'

sed

- combining multiple commands is to use a *-e* before each command:
- sed -e 's/a/A/' -e 's/b/B/' <old >new

- The "-n" option will not print anything unless an explicit request to print is found
- sed -n 's/PATTERN/&/p' file

Reading and manipulating text files

- **Awk** :- reads input line by line, tokenizes line into tokens. Allows to write sub scripts for manipulating based on tokens.
- Basic Format: *pattern { action }*
- Following is format for action
  BEGIN { print "START" } ← *Action before any line is read*

  { print }

  END { print "STOP" } ← *Action after last line is read*
- Simple awk examples:
  - BEGIN { x=5 }
    { print x, $x}
  - ps -aef | awk '/CLAGENT/ && !/awk/ {print $0}‘
  - UPTIME=`uptime | awk '{if (($4 == "days,") || ($4 == "hrs,")) print $3; else print 0}'`

AWK

- AWK supports C type syntax
- Arithmatic operators

+ - * / % ++ --

  - c-type assignments
    - = += -= *= /= %=
  - Relational operators
    - == > >= < <= !=
  - Regular Expressions
    - ~= !=
- Relational Operator
  - && || !
- Commands supported by AWK

if (conditional) statement [else statement]
while (conditional) statement
for (expression ; conditional ; expression) statement
for (variable in array) statement
break
continue
{ [statement] ...}
variable=expression
print [expression-list] [> expression]
printf format [, expression-list] [> expression]
next
exit

AWK – Built in Vars

- "$0" refers to the entire line that AWK reads in
- $n refers to nth token ($1 $2 etc)
- FS – refers to field separator token
- OFS – Output Field Separator
- NF - Number of fields (tokens generated)
- NR – Number of record or line number
- FILENAME - Name of the file being read.

AWK - script

- ```
  #!/bin/sh
  awk '
  BEGIN { print "File\tOwner" }
  { print $8, "\t", $3}
  END { print " - DONE -" }
  '
  ```

- AWK is also an interpretor, you can save yourself a step and make the file executable by add one line in the beginning of the file

- ```
  #!/bin/awk -f
  BEGIN { print "File\tOwner" }
  { print $8, "\t", $3}
  END { print " - DONE -" }
  ```

AWK – dynamic variables

- In following example $column will not be printed correctly.
- #!/bin/sh
  #NOTE - this script does not work!
  column=$1
  awk '{print $column}'

  instead another $ is required to access it as follows
- #!/bin/sh
  column=$1
  awk '{print $'$column'}'

AWK – string functions

- index(string,search)
- length(string)
- split(string,array,separator)
- substr(string,position)
- substr(string,position,max)

AWK - Exercise

- **Exercise**: write shell script to read all lines containing Motorola and not Mobility. Append “MSG” to beginning of each line in the output.

- awk ‘/Motorola/ && !/Mobility/ {printf (“MSG %s\n“, $0)}’ file.txt

Reading and manipulating text files

- **cut**: this also tokenizes input lines like awk
- Example:
  - ps -aef | grep CLAGENT | grep -v "grep" | tr -s " " | cut –d " " -f8
- Write a shell script to find IP address of system usinf “ifconfig” command.

Reading and manipulating text files

- Exercise
  - Write a script to find a tar file “monitor.tar” in your home dir. Find list of files in tar file. Add the total sizes of files which are tarred with subdirectory script.

Example, if tar contains following files

a /home/script/kernelOops.sh 7K

a /home/script/monitorKeysForOops.sh 2K

a /home/ssh/keys.sh 5K

Script should report total script size as 9K

- Write a script to read a file and print only the last word from it.
- Write a script to read a file and print 3rd token if 2nd word is “day” or 1st word is “date” else print an error message for that line.
  - awk '{if (($1 == "date") || ($2 == "day")) print $3; else print "Error "}'` file.txt

- Write a script to get pid of current script. Then list all tasks running with same parent as current scripts parent.

Exercise

- Write a script to capture cpu usage using top command. Print the Error if cpu usage exceeds 50%
- Write a script to find memory usage of a particular task (ps command/ pmap -d <pid>)

Debugging shell script

- sh –vx script.sh
- bash –vx script.sh

BackUp

root@hapWibbSc2:~# time ps -aef | awk '/CLAGENT/ && !/awk/ {print $0}' | wc -l |awk '{print $1}'
1

real 0m0.027s
user 0m0.008s
sys 0m0.020s

root@hapWibbSc2:~# time ps -aef | grep CLAGENT | grep -v "grep" | tr -s " " | cut -d " " -f8 |wc |tr -s " " | cut -d " " -f2
1

real 0m0.043s
user 0m0.012s
sys 0m0.028s

Links:
http://www.grymoire.com/Unix/Awk.html