De La Salle University
College of Computer Studies

SYSMGMT Laboratory Manual

Name:	Samuel Gabrielle Malijan	Date:	7-11-2018
Section:	S15	Grade:	

3.0 Unix / Linux Command Line Interface (CLI) Basics –Unix Shells and Shell Scripts

The Unix shell is a program that allows interactive use of the Unix operating system and the computer hardware. There are different shells created for the Unix system. The first shell was the Bourne shell. After the Bourne shell, the Korn and C shell was later created. The Korn shell is a superset of the Bourne shell while the C shell was “C” like (“C” as in the C programming language). Although there are several shells available, the shells have the same set of features with only subtle differences. For example, the “history” command is not available in the Bourne shell but is available in the Korn and C shells. The prompts of different shells also look different.

Today, the Linux system uses the Bourne Again shell. This “new” shell has the useful features of Korn and C shells. You could say that it is the hybrid of the Korn and C shell. Do remember that the Bourne Again shell is only in Linux versions of Unix systems.

As mentioned earlier, these shells have same set features. These features are aliasing, environment setting, variables and shell programming or also known as shell scripts. These features will be discussed in better detail in this section.

Shell scripts allow a user to make useful utilities using Unix commands or other utilities. Shell scripts also have constructs like programming languages that allow flexibility.

With a minimum background of shells, scripts and job control, the objectives of this section are:

1. Familiarization with Unix shells, it features and how to use it.
2. Familiarization with Unix shell scripts and how create and use shell scripts.

1.1. The Unix Shell

1.1.1.Shell Features

“alias” Command

An alias allows a way to give a different name or shorter name for a command. This allows a user to “personalize” some commands that are used often.

Step 1 On the prompt, enter the command **`alias ll="ls -l"`**. Again on the prompt, enter the command **`ll`**.

1a. What is the output of the command? What is the equivalent of the command?

It lists the files of the OS,and the equivalent of the command is basically List.

1b. What other command can you think of that you might need an alias? (Include the options for the command)

man ls I suppose? I will just type ml as it will help me to easily access the manual I dont understand some commands

1c. On the prompt, type the command **`unalias ll`**. Enter the command **`ll`** at the prompt. What happened? Where you able to show the listing of your directory? Why not?

Nope, because I removed the alias for such a command alias.

1d. What does the alias and **`unalias`** command do?

Alias allows you to personalize a hotkey - basically shortening the type time or gives you better key words. Unalias allows you to remove mistakes in aliasin or if you just want to remove it.

"set" Command

The set command allows setting and unsetting of flags that control shell-wide characteristics. There are a lot flags available on the Unix system. What will be presented are the useful ones.

Step 2 At the prompt, enter the command **`ls > list`** and then enter the command **`cat list`**.

2a. At the prompt, enter the command **`ls –l > list`** and then enter the command **`cat list`**. Was the text file overwritten? Why?

Yes it was overwritten, because you put the new list from the ls -l command into the textbox.

2b. At the prompt, enter the command **`set –o noclobber`** and then enter the command **`ls > list`**. Were you allowed to overwrite the file? What does the command do?

Nope, it seems that noclobber prevents overwriting of files.

2c. At the prompt, enter the command "**`set +o noclobber`**" and then enter the command **`ls > list`**. Were you allowed to overwrite the file? What does the command do?

Yes I was able to overwrite the file due to the removal of the setting in the previous step.

Step 3 At the prompt, enter the command **`set –o verbose`** and then enter the command **`ls -l`**.

3a. What does the command do?

It basically prints what the terminal is doing. For example, ls -l, it prints it after you type it in.

3b. At the prompt, enter the command **`set +o verbose`** and then enter the command **`ls –l`**. Did it turn off the verbose mode? Why?

yes, because after you enter the command ls -l it stop s printing.

Command Sequencing

Several commands can be executed in succession in the Unix system. By using the ";", commands can be sequenced together. The Unix system is also able to do conditional sequencing. With conditional sequencing, a command will only be executed if the first one is successful or the next command will only be a executed if there was an error.

Step 4 At the prompt, type the command "**`date; ls –l | more`**".

4a. What did the command do? Did it execute two commands in succession?

It printed the date along with the time, and printed the llist of files. And yes, it did.

4b. At the command prompt, type the command "**`date; ls; pwd > out.txt`**". View the text file `using` a text editor or "**`cat`**" command. What was on the "out.txt" file? (Describe)

/home/samuel basically where the file was located.

4c. At the command prompt, type the command "**`(date; ls; pwd) > out.txt`**". View the text file using a text editor or "`cat`" command. What was on the "`out.txt`" file? (Describe)

First, the date and time was printed, then the list of directories and files within the directory, and finally, the directory on where the out.txt is located.

4d. Why were the output of the two previous command were different? What was the command?

During the first command, date and ls were executed, but it was not put in the text because only pwd was the only command to be printed during the first. However, during the second, with the parenthesis, it clearly says in human language to put all of these things printed into the text file.

Step 5 At the prompt, type the command "**`mkdir dir1`**" then enter the command "**`ls –d dir1 && echo "Directory found"`**" Notice that this is a conditional command. The "**`&&`**" specifies that the next command will be executed when the first command is successful.

5a. Did it execute the second command?

Yes it said that the directory was found.
i

5b. At the prompt, type the command "**`ls –ld dir3 && echo "Directory found"`**". Did it execute the second command? Why?

It could not execture instead it gave an error message saying that the directory could not be found.

Shell Variables

Shell variables are store information required by processes so that processes can function properly. An example of a variable is how your command prompt looks like. The variable is called PS1. There are two types of shell variables: local and global (also known as environment variable). Local variables are variables that are available for the current shell session only while global variables are variables that are available to the current and all child or sub-shells that the user or the system might start

Step 6 At the prompt, enter the command "**`my_name="<your first name>"`**" and then enter the command "**`echo $my_name`**".

6a. What was the output of the command? Were you able to create a variable?

THe output of the command was Samuel, and yes it seemed so.

6b. At the prompt, enter the command "**`echo my_name`**". Was the output same as before? Is the "`$`" sign needed to display the value of the variable? Why?

it just echoed my_name. I think the $ sign is an indicator that you are trying to call a variable. It is similiar to PHP when it calls variables.

6c. What did you create? A local or global variable?

I think I made a local variable.

6d. At the prompt, enter the command "bash" and then enter the command "**`echo $my_name`**". Were you able to see your name?

After getting prompted to sudo mode - no i wasnt able to se emy name after bashing.
I think Bash resets all the variables?

6e. Without exiting the shell yet, type the command "**`my_name="<your first name> <your last name>"`**" and enter the command "**`echo $my_name`**". Were you able to see your name? Explain why you were not able to see your name earlier?

Yes, I was able to see my full name.
Because you entered the command for bash which resets al currently set variables I think.

6f. Exit the shell and enter the command "**`echo $my_name`**". Was the output same as previous shell? Why?

Nope, because it is just a local variable, not a global variable.

Step 7 At the command prompt, enter the command "**`PS1="%"`**".

7a. What happened to the prompt? Did it change?

It changed to a % prompt.

Notice that PS1 is a shell variable. It was created by the operating system. In this case, the variable PS1 holds the text or command what the prompt should look like.

7b. At the prompt, enter the command "**`set | more`**". What did it displayed? Were able to find PS1?

It displayed the contents of the BASH command, such as the bash versions and commands, and etc.
Yes I was able to, and PS1 is set to %.

7c. At the command prompt, enter the command "**`PS1="[\\u@\\h \\W]\\$"`**". Did the command prompt go back to original state?

Well, instead of a % it went to an arrow head instead.(>)

There are special characters that the shell can accept for the prompt setting. These settings are:

Setting	**What for**
\u	Username
\d	Date
\h	Hostname
\$	Dollar Sign
\W	Working directory

7d. Using the command "**`set | more`**", find the shell variable for your log-in name. Is there shell variable for it? What is the name of the variable?

LOGNAME is the name of the variable holding my name for logging in.

7e. Using the command "**`set | more`**", find the shell variable for your home directory. Is there shell variable for it? What is the name of the variable?

It is named HOME, that contains the home directory.

Initialization Files

Initialization files or scripts can contain series of commands, settings and variables. Initialization files allows a user to personalize some settings. There are two types of initialization files: system wide and personal. The system wide initialization is set by the administrator and is the default initialization file. The personal initialization file resides in the user's home directory and can be "personalized" by the user. In the bash shell, there are two initialization files, the "**`.bash_profile`**" and the "**`.bashrc`**". The first initialization file can contain shell variable settings while the second initialization file contains alias settings. The "**`.bash_profile`**" can also contain other command that can be executed during the logging in.

Step 8 At the prompt, enter the command "**`PS1=%`**". Make sure that the prompt has changed and enter the "exit" command.

8a. Login to the server and check the prompt. Were you able to change the prompt? Did it go back to its original state?

No, I was not ablet o change the prompt, and yes it did go back to its original state.

8b. Edit the "**`.bash_profile`**" using the vi text editor. At the end of the file, enter the text "**`PS1="[\u@\h \W]$"`**". Save the text file and exit. Login to the server again and check the prompt. Did the prompt change permanently?

yes, it did, to student@samuelserver~W$.

Step 9 At the prompt, enter the command "**`alias ll="ls -l"`**". Enter the command "**`ll`**".

9a. Were you able to see the effect of the alias?

No, as allias cant even be set nr can it be found.

9b. Exit the session and login to the server again. Enter the command "**`ll`**" at the prompt. Were you able to see the effect of the command? Why do you think the command was not successful?

I think because we have edited the bash_profile? And because of that somehow our user privelege has decreased? Thats my theory.

9c. Edit the "**`.bashrc`**" using the vi text editor. At the end of the file, enter the text "**`alias ll="ls-l"`**". Save the text file and exit.

Edit the "**`.bash_profile`**" using the vi text editor. Add the following lines:

```
if [ -f "$HOME/.bashrc" ]; then
    . "$HOME/.bashrc"
fi
```

This instructs the system to execute commands in the **`.bashrc`** file on login if it exist in the user home directory. Save the text file and exit.

9d. Login to the server again and check the command. Were you able to use the alias? Why so?

I wasn't able to, for some reason.

1.2. Shell Scripts

A shell script (sometimes also known as shell program) is a series of commands and utilities that have been put to a file by using a text editor. When the shell script is executed, the commands are interpreted and executed by the shell program. The commands in the shell script are executed by the shell one after the other. A shell script is like any other programming language and has its own syntax. You can define variables; assign various values, and use control and iterative statements.

After creating the shell script, the file permission of the shell script should be read and execute (at least by the owner). So that the shell can find your shell script, make a directory called “bin” in your home directory. Your shell script should be saved in the “bin” directory. The shell will be able to find it because the shell variable “PATH” enumerates “$HOME\BIN” as one of the directories to look for executables files for commands entered. You will see this later in this section.

The programming language like syntax of the shell scripts is easy to understand if you have experiences with the C programming language. One problem though is that you should be using the C shell. With the bash shell, the syntax is a little different but still easy to understand. For this section of the course, the bash shell syntax will be used. The syntax for shell scripting in the bash and C shell is provided as a supplemental material in this course. Please get it from your instructor.

A Shell Script Program

Step 10 Type the following text and save it your directory. Use the filename “`fmgr`”.

```
get_file_name()
{
  echo -n "Enter filename: "
  read mfile
}

print_no_file()
{
  echo "$mfile: No such file or directory"
}

while [ 1 ]
do
  clear
  echo "File Manager 1.0"
  echo "Please choose an option"
  echo "1 - Move a file to the recycle bin"
  echo "2 - Retrieve a file from the recycle bin"
  echo "3 - Empty the recycle bin"
  echo "4 - Delete a file permanently"
  echo "5 - Exit"
  echo -n "Choose option: "
  read option
  case $option in
    1)
      get_file_name
      if [ -f $mfile ]; then
       mv $mfile $HOME/.junk
      else
       print_no_file
      fi;;
    2)
      get_file_name
      if [ -f $HOME/.junk/$mfile ]; then
       mv $HOME/.junk/$mfile $HOME
      else
       echo "$mfile: No such file or directory"
      fi;;
    3)
      echo -n "Are you sure you want to empty recycle bin? "
      read answer
      if [ $answer="y" -o $answer="Y" ]; then
       rm $HOME/.junk/*
      fi;;
    4)
      get_file_name
      if [ -f $mfile ]; then
       rm -i $mfile
      else
       print_no_file
      fi;;
    5)
      exit;;
    *)
  esac
done
```

After saving the file, add the “`execute`” permission. Also, create a directory called “`.junk`”. The period (.) at the start of the filename will create a hidden file.

10a. Execute the shell script by entering "**`./fmgr`**" on the prompt. Did it execute?

Yes it did.

10b. Choose option number 5, this will exit the script. Execute the script again but this time, use the command "**`fmgr`**". Did it execute? What was the message?

Command not found.

Remember that commands are actually programs in the Unix system. The shell automatically looks for them in certain directories specified in the shell variable "**`$PATH`**". If you want to see the paths, just enter the command "**`echo $PATH`**".

Step 11 Create a directory called "**`bin`**" on your home directory and type in the command "**`PATH=$PATH:$HOME/bin`**". The previous command adds the "**`$HOME/bin`**" to one of the paths to look for progams. Move the "**`fmgr`**" file to the "**`bin`**" directory in your home directory. Execute the "**`fmgr`**" script and make sure you are in your home directory.

11a. Were you able to execute the file? Why were you able to execute the file even if it wasn't in the home directory?

Perhaps because you registered the place where the file is being stored.

Step 12 Copy the files "**`SYSMGMT Lab – 03 Shells and Scripting.pdf`**" and "**`rfc2186.txt`**" from the "**`/home/student/files`**" directory. If you still have the files, there is no need to copy it. Execute the "**`fmgr`**" script file. In answering the questions in this step, make sure you point out pieces of code in the script file.

12a. Put the file "**`rfc2186.txt`**" to the recycle bin and exit the script file. Is the file in the "**`.junk`**" directory?

12b. Execute the "**`fmgr`**" script file and retrieve the "**`rfc2186.txt`**" file. Is the file still in the "**`.junk`**" directory? Were you able to retrieve it?

12c. Execute the "**`fmgr`**" script file and empty the recycle bin. Is the file still in the "**`.junk`**" directory?

12d. Execute "**`fmgr`**" script file and put the file "**`rfc2186.txt`**" to the recycle bin. Empty the recycle bin and exit the script. Is the file in the "**`.junk`**" directory? Why?

12e. Execute "**`fmgr`**" script file and permanently delete the file "**`SYSMGMT Lab – 03 Shells and Scripting.pdf`**". Is the file in the "**`.junk`**" directory? Why?

Although it may look like that shell scripts are like programs, notice that shell scripts are able to execute shell commands to. Most network administrators automate tasks using shell scripts. You can even change server settings, add a batch of users, or even delete a batch of users of using shell scripts.