The Shell

Introduction

The Linux/Unix shell refers to a special program that allows you to interact with it by entering certain commands from the keyboard; the shell will execute the commands and display its output on the monitor. The environment of interaction is text-based (unlike the GUI-based interaction we have been using in the previous chapters) and since it is command-oriented this type of interface is termed Command Line interface or CLI. Before the advent of GUI-based computing environments, the CLI was the only way that one can interact and access a computer system.

Up until now, there was never a need to type commands into a shell; and with the modernisation and creation of a lot of newer GUI-based tools, the shell is becoming increasingly un-required to perform many tasks. But that said, the shell is a very powerful place, and a lot is achieved through it. A lot of the front-end GUI methods of doing things have similar ways and means to get done with using the shell. Professional Linux and UNIX users find the shell very powerful, and an introduction to at least the basic shell usage is useful.

Getting to a shell

Since it is most likely that you are in the graphical desktop environment now, the underlying shell that is available is not displayed. To access a shell, try the following key combination,

Control + Alt + F1

Where F1 can be replaced by F2, F3, and so on. The graphical desktop tends to run in F7 or F8, so to go back to your graphical desktop screen, just hit Control + Alt + F7. These are virtual terminals.

Alternatively, you could get to a Terminal application, so you can have a shell while your in the graphical desktop environment (this is much preferred, and will be used throughout this Chapter). To do this, go to:

Main Menu --> System Tools --> Terminal

Or right-click on the desktop, and click on the Open Terminal option. This terminal is equivalent to the virtual terminals mentioned earlier, except now you don't have to switch screens – you can just minimize or maximize the terminal (or if you're done, you can close it).

Some useful commands

Now that you are at a terminal, you might as well input some commands. For example, when you start a shell, display such as below (or similar) will be seen (and this can be configured to your liking!):

```
[-(byte@hermione)-(pts/4)-(05:34pm:05/06/2004)-]
[-(~)>
```

The cursor blinks, waiting for input. To this, some of the more used and useful commands include:

- ls – list files in the current directory.

- cd – change working directory. If your current path is /home/username/Trash for instance, typing “cd” will bring you back to /home/username.

- mkdir – make a new directory

- rmdir – delete a directory (must be empty)

- cp – invoked such as “cp currentFile newFile”, and is used to copy files.

- mv – invoked such as “mv currentLocation newLocation”. This is used to either move or rename files.

- rm – invoked such as “rm myFile”; it is used to delete files permanently.

- pwd – prints the working (current) directory.

- cat – concatenate files (can be used to join them together), and prints its output to standard output (the terminal screen). Used like: “cat myFile”.

- less – allows for file viewing in the shell, and is most useful for text files; invoked like “less myFile”.

- find – can be used to find files via the command line. Example usage could be: “find . -name toc”, which looks at the current directory (defined by “.”) for any files with the name “toc”.

- locate – picks entries from a database, that is updated regularly; invoked via “locate myFile”. Its much quicker than find (since it only searches a database), but might not be as quick to update as find (the update of the database might happen once every day only).

- date – display the current date! This can also be used to set the date of the system (but administrator privileges are required).

- history – built-in shell command for the BASH environment that shows the last run commands.

As always, these commands just begin to scratch the surface of the capability of the shell. There are thousands of such commands available on your system! And keep in mind that each and every command comes with options, that are usually executed via the *-flag* – again, the man pages list all useful commands. For instance the command

rm -i

will prompt when deleting a file, so you have to either say 'y' if you're sure, or 'n' if you do not want to delete the file.

```
[-(/tmp)> rm -i usr.bin
rm: remove regular file `usr.bin'? y
```

A Few More Concepts and Shortcuts

Now that you've seen some commands that are useful in the shell, its important to know a few more concepts. For instance, the tilde (“~”) represents the home directory, so rather than typing /home/username it can be represented via a '~'. This means less typing for you.

```
[-(~/MyOSS-Stuff/IOSN)> pwd
/home/byte/MyOSS-Stuff/IOSN
[-(~/MyOSS-Stuff/IOSN)> cd ~
[-(~)> pwd
/home/byte
```

So in that example, I was located in /home/byte/MyOSS-Stuff/IOSN, and just by issuing a “cd ~”, the shell has brought the current working directory to /home/byte.

A dot “.” means the current directory. While “../” will mean the parent directory. This can be nested to include “../../” and so on, till it reaches the top level directory /.

Input/output Redirection and Pipes

Running a command by itself with a lot of output doesn't seem all that useful. For instance, if there are many files in a directory, running a command to list the directory like,

ls /usr/bin

will result in about 2100 lines being displayed on the screen! To actually get any useful information out of it, you might want to dump the output of the ls command to a file; or maybe use a utility like less to view it. All this is possible thanks to input/output redirection and pipes.

Input redirection is performed using < or <<, while output redirection is done via > or >>. A point to note is that when using >, it just recreates the file, even if the same filename exists, while >> concatenates the output to the same file, causing it to possibly be double in size (if its the same output).

A pipe ("|") is used to pass the output of the command not to a file, or to the screen, but to the next utility. Pipes can be nested, so you can pass the data through several utilities before you can get the useful information that you want. Let's dive into some examples!

```
1. [-(/tmp)> ls /usr/bin >> usr.bin
2. [-(/tmp)> wc -l usr.bin
3. 2171 usr.bin
4. [-(/tmp)> ls /usr/bin >> usr.bin
5. [-(/tmp)> wc -l usr.bin
6. 4342 usr.bin
7. [-(/tmp)> ls /usr/bin > usr.bin
8. [-(/tmp)> wc -l usr.bin
9. 2171 usr.bin
```
Note: the line numbers are added for clarity, and are not included in the shell output!

In line 1, the output of the directory listing of /usr/bin gets placed in a file called usr.bin. On line 2, a new utility called 'wc' is used (this is used to print the number of lines in the file (as it gets passed the -l option) – its output is at line 3. The same command is then repeated on line 4, and now, the file is double the size as per line 6! That is because the >> output redirection was used, which has concatenated the two outputs together. Notice that in line 7, a single > is used, and in line 9, it shows that the file has been over-written with the new contents.

```
[-(/tmp)> ls /usr/bin | grep cancel
cancel
cancel.cups
```

The above is an example of how a pipe is used. After listing the files, the output is passed on to a utility called grep (which basically searches for a pattern, and prints the output) and the string being searched for is "cancel". It comes back with two matches. Similarly, a command like:

ls /usr/bin | less

Will place the output of the directory listing into the less pager so that it can be scrolled through easily. And for another example as to how pipes can be nested, issuing:

```
[-(/tmp)> 'ls' /usr/bin | grep auto|wc -l
19
```

sends the output of the directory listing of /usr/bin to grep, which then searches for the string "auto", and then wc prints how many times it occurs in lines.

A useful command string that a lot of systems administrators tend to use would be:

```
[root@hermione root]# tail -f /var/log/messages
Jul  5 12:04:02 hermione last message repeated 13 times
Jul  5 16:17:17 hermione last message repeated 17 times
Jul  5 16:17:28 hermione last message repeated 18 times
Jul  5 16:17:32 hermione
```

A 'tail' displays the last ten lines of the file, and the -f option means that if there are more logs, it gets displayed (via it being appended to the bottom).

WHERE DO I GET HELP?

Rather than get frightened off the shell, there are some sources of help, in the event that you aren't sure what you're doing in the shell.

Man Pages

These are manual pages, for each and every command that resides on your system. This is a first point of reference, and it is invoked by:

man *command-name*

e.g.

```
$ man man
```

The above runs man on itself, explaining a bit about the manual page system.

Info Pages

This is the new GNU project method of distributing manuals, and info pages are a lot more comprehensive than man pages. It is invoked by:

info *command-name*

e.g.

```
$ info info
```

The above runs info on itself, and provides some useful information as to how info can be used, and how you can navigate info documents.

Other Useful Commands (for help)

While still on the topic of help, there are a few more useful commands that you want to know about:

- whatis – invoked by "whatis package-name" and it provides information about the tool that whatis recognizes (and has in its database).

- apropos – invoked by "apropos *string*", and it provides strings matching what is located in the whatis database. This is most useful when you don't know what command you want to run, but have an idea that as to what it should be dealing with (so apropos mail should provide all sorts of mail clients that are available on your system).

CONCLUSION

This is the power of Linux and UNIX command lines. There is much more to learn, as there are different shells, and different shell syntaxes available. Also, regular expressions are useful, and there are plenty more utilities available, and if a liking towards the shell is taken, shell scripts can be written to perform a lot of tasks, including backing up directories and more!

EXERCISES

1. Open up a shell on your Desktop and perform the following:

   - find the name of the directory you are in
   - list out the contents of the current directory
   - list out the contents of the directory */usr/bin*
   - check the current date and time

2. Change directory to your home directory and make a new sub-directory there named *Temp11* and change directory to it
   - copy the following files from the */etc* directory to the directory Temp11*: services, motd, fstab, hosts*

- concatenate the files copied above into one single file called *file1*
- count the number of lines present in the file *file1*
- delete the four files listed above in the directory Temp11