LAB NO: 1 **Date:**

UNIX SHELL COMMANDS

Objectives:

1. To recall the UNIX special characters and commands.

2. To describe basic commands.

1. UNIX shell, special characters and commands

What is shell?

A shell is an environment in which one can run commands, programs, and shell scripts. There are different flavors of shells, just as there are different flavors of operating systems. Each flavor of shell has its own set of recognized commands and functions.

Shell Prompt:

The prompt, $, which is called command prompt, is issued by the shell. While the prompt is displayed, one can type a command. The shell reads the input entered by user. Pressing Enter key is must for the execution. It determines the command to be executed by looking at the first word of the input. A word is an unbroken set of characters. Spaces or tabs are used to separate words.

Following is a simple example of date command which displays current date and time:

```
$date
Thu Jan 1 08:30:19 IST 2016
```

Shell Types:

In UNIX there are two major types of shells:

1. The Bourne shell. If you are using a Bourne-type shell, the default prompt is the $ character.
2. The C shell. If you are using a C-type shell, the default prompt is the % character.

There are again various subcategories for Bourne Shell which are listed as follows:

- Bourne shell (sh)
- Korn shell (ksh)
- Bourne Again shell (bash)
- POSIX shell (sh)

The different C-type shells follow:

- C shell (csh)
- TENEX/TOPS C shell (tcsh)

The original UNIX shell was written in the mid-1970s by Stephen R. Bourne while he was at AT&T Bell Labs in New Jersey. The Bourne shell was the first shell to appear on UNIX systems, thus it is referred to as "the shell". The Bourne shell is usually installed as /bin/sh on most of the versions of UNIX. For this reason, it is the shell of choice for writing scripts to use on several different versions of UNIX.

Special Characters

It is important to know about meaning of symbols and characters that are used to serve special purpose in unix environment. This means that few characters: a) cannot be used in certain situations, b) may be used to perform special operations, or, c) must be "escaped" if one wants to use them in a normal way.

Character	Description
\	Escape character. If user want to reference a special character, one must "escape" it with a backslash first. Example: touch /tmp/filename*
/	Directory separator, used to separate a string of directory names. Example: /usr/src/linux
.	Current directory. Can also "hide" files when it is the first character in a filename.
..	Parent directory
~	User's home directory
*	Matches 0 or more characters in a filename, or by itself, all files in a directory. Example: pic*2002 can represent the files pic2002, picJanuary2002, picFeb292002, etc.
?	Matches a single character in a filename. Example: hello?.txt can represent hello1.txt, helloz.txt, but not hello22.txt
[ ]	Can be used to represent a range of values, e.g. [0-9], [A-Z], etc. Example: hello[0-2].txt represents the names hello0.txt, hello1.txt, and hello2.txt
|	Used to redirect the output of one command into another command. Example: ls | more
>	Redirects the output of one command into a new file. If file already exists, it will be over written. Example: ls > myfiles.txt
>>	Redirects the output of a command onto the end of an existing file. Example: echo "Mary 555-1234" >> phonenumbers.txt
<	Redirect a file as input to a program. Example: more < phonenumbers.txt
;	Allows to execute multiple commands on a single line. Example: cd /var/log ; less messages

&&	Command separator as above, but only executes the second command if the first command is executed successfully without errors. Example: cd /var/logs && less messages
&	Execute a command in the background, and immediately get your shell back. Example: find / -name core > /tmp/corefiles.txt &

2. Shell commands and getting help

Executing Commands

Most common commands are located in your shell's "PATH", meaning that one can just type the name of the program to execute it. Example: Typing "ls" will execute the "ls" command. Shell's "PATH" variable includes the most common program locations, such as /bin, /usr/bin, /usr/X11R6/bin and others. To execute commands that are not in the current PATH, user has to give the complete location of the command. [PATH is an environmental variable. To display the value of PATH variable execute echo $PATH]

Examples: /home/bob/myprogram
./program (Execute a program in the current directory)
~/bin/program (Execute program from a personal bin directory)

[Before executing the program, the program file has to be granted with execution permission. For granting execute permission the command **chmod +x program** has to be executed.]

Command Syntax

Commands can be executed by passing the arguments or without passing the arguments. To serve special purposes from the command argument has to be passed. Typical command syntax can look something like this:

command [-argument] [-argument] [--argument] [file]

Examples:	ls	List files in current directory
	ls -l	Lists files in "long" format
	ls -l --color	As above, with colourized output
	cat filename	Show contents of a file
	cat -n filename	Show contents of a file, with line numbers

Getting Help

When user is stuck and need help with a Linux command, help is usually only a few keystrokes away! Help on most Linux commands is typically built right into the commands themselves, available through online help programs ("man pages" and "info pages").

Many commands have simple “help” screens that can be invoked with special command flags. These flags usually look like “-h” or “--help”. Example: grep --help

“Man Pages” The best source of information for most commands can be found in the online manual pages, known as “man pages” for short. To read a command's man page, type “man command”.

Examples:	man ls	Get help on the “ls” command.
	man man	A manual about how to use the manual!

To search for a particular word within a man page, type “/word”. To quit from a man page, just type the “Q” key.

Sometimes, you might not remember the name of Linux command and you need to search for it. For example, if you want to know how to change a file's permissions, you can search the man page descriptions for the word “permission” like this: man -k permission

If you look at the output of this command, you will find a line that looks something like:

chmod - change file access permissions Now you know that “chmod” is the command you were looking for.

Typing “man chmod” will show you the chmod command's manual page!

3. Commands for Navigating the Linux file systems

The first thing you usually want to do when learning about the Linux filesystem is take some time to look around and see what's there! These next few commands will: a) Tell you where you are, b) take you somewhere else, and c) show you what's there. The following are the various commands used for Linux file system navigation.

a. ***pwd*** “Print Working Directory”. Shows the current location in the directory tree.
b. ***cd*** “Change Directory”. When typed all by itself, it returns you to your home directory.
c. ***cd directory*** Change into the specified directory name. Example: cd /usr/src/linux
d. ***cd ~*** “~” is an alias for your home directory. It can be used as a shortcut to your “home”, or other directories relative to your home.
e. ***cd ..*** Move up one directory. For example, if you are in /home/vic and you type “cd ..”, you will end up in /home.
f. ***cd -*** Return to previous directory. An easy way to get back to your previous location!
g. ***ls*** List all files in the current directory, in column format.
h. ***ls directory*** List the files in the specified directory. Example: ls /var/log
i. ***ls -l*** List files in “long” format, one file per line. This also shows you additional info about the file, such as ownership, permissions, date, and size.
j. ***ls -a*** List all files, including “hidden” files. Hidden files are those files that begin with a “.”
k. ***ls –ld directory*** A “long” list of “directory”, but instead of showing the directory contents, show the directory's detailed information. For example, compare the output of the following two commands:
    i. ls -l /usr/bin

ii. ls -ld /usr/bin

l. ***ls /usr/bin/d**** List all files whose names begin with the letter "d" in the /usr/bin directory.

3.1 Filenames, Wildcards, and Pathname Expansion

Sometimes you need to run a command on more than one file at a time. The most common example of such a command is ls, which lists information about files. In its simplest form, without options or arguments, it lists the names of all files in the working directory except special hidden files, whose names begin with a dot (.). If you give ls filename arguments, it will list those files—which is sort of silly: if your current directory has the files duchess and queen in it and you type ls duchess queen, the system will simply print those filenames. But sometimes you want to verify the existence of a certain group of files without having to know all of their names; for example, if you use a text editor, you might want to see which files in your current directory have names that end in .txt. Filenames are so important in UNIX that the shell provides a built-in way to specify the pattern of a set of filenames without having to know all of the names themselves. You can use special characters, called wildcards, in filenames to turn them into patterns. The following provides the list of the basic wildcards.

Wildcard	Matches
?	Any single character
*	Any string of characters
[set]	Any character in set
[!set]	Any character not in set

The ? wildcard matches any single character, so that if your directory contains the files program.c, program.log, and program.o, then the expression program.? matches program.c and program.o but not program.log. The asterisk (*) is more powerful and far more widely used; it matches any string of characters. The expression program.* will match all three files in the previous paragraph; text editor users can use the expression *.txt to match their input files.

(aside from the leading dot, which "hides" the file); it's just another character. For example, ls * lists all files in the current directory; you don't need *.* as you do on other systems. Indeed, ls *.* won't list all the files—only those that have at least one dot in the middle of the name. Assume that you have the files bob, darlene, dave, ed, frank, and fred in your working directory.

Expression	Yields
fr*	frank fred
*ed	ed fred
b*	bob
e	darlene dave ed fred
r	darlene frank fred
*	bob darlene dave ed frank fred
d*e	darlene dave

g* g*

Notice that * can stand for nothing: both *ed and *e* match ed. Also notice that the last example shows what the shell does if it can't match anything: it just leaves the string with the wildcard untouched.

The remaining wildcard is the set construct. A set is a list of characters (e.g., abc), an inclusive range (e.g., a-z), or some combination of the two. If you want the dash character to be part of a list, just list it first or last.

Using the set construct wildcards are as follows:

Expression	**Matches**
[abc]	a, b, or c
[.,;]	Period, comma, or semicolon
[-_]	Dash or underscore
[a-c]	a, b, or c
[a-z]	All lowercase letters
[!0-9]	All non-digits
[0-9!]	All digits and exclamation point
[a-zA-Z]	All lower- and uppercase letters
[a-zA-Z0-9_-]	All letters, all digits, underscore, and dash

In the original wildcard example, program.[co] and program.[a-z] both match program.c and program.o, but not program.log. An exclamation point after the left bracket lets you "negate" a set. For example, [!.;] matches any character except period and semicolon; [!a-zA-Z] matches any character that isn't a letter. To match ! itself, place it after the first character in the set, or precede it with a backslash, as in [\!].

The range notation is handy, but you shouldn't make too many assumptions about what characters are included in a range. It's safe to use a range for uppercase letters, lowercase letters, digits, or any subranges thereof (e.g., [f-q], [2-6]). Don't use ranges on punctuation characters or mixed-case letters: e.g., [a-Z] and [A-z] should not be trusted to include all of the letters and nothing more.

The process of matching expressions containing wildcards to filenames is called wildcard expansion or globbing. This is just one of several steps the shell takes when reading and processing a command line; another that we have already seen is tilde expansion, where tildes are replaced with home directories where applicable.

However, it's important to be aware that the commands that you run only see the results of wildcard expansion. That is, they just see a list of arguments, and they have no knowledge of how those

arguments came into being. For example, if you type ls fr* and your files are as on the previous page, then the shell expands the command line to ls fred frank and invokes the command ls with arguments fred and frank. If you type ls g*, then (because there is no match) ls will be given the literal string g* and will complain with the error message, g*: No such file or directory. This is different from the C shell's wildcard mechanism, which prints an error message and doesn't execute the command at all.

Here is an example that should help make things clearer. Suppose you are a C programmer. This means that you deal with files whose names end in .c (programs, also known as source files), .h (header files for programs), and .o (object code files that aren't human-readable), as well as other files. Let's say you want to list all source, object, and header files in your working directory. The command ls *.[cho] does the trick. The shell expands *.[cho] to all files whose names end in a period followed by a c, h, or o and passes the resulting list to ls as arguments. In other words, ls will see the filenames just as if they were all typed in individually—but notice that we required no knowledge of the actual filenames whatsoever! We let the wildcards do the work.

The wildcard examples that we have seen so far are actually part of a more general concept called pathname expansion. Just as it is possible to use wildcards in the current directory, they can also be used as part of a pathname. For example, if you wanted to list all of the files in the directories /usr and /usr2, you could type ls /usr*. If you were only interested in the files beginning with the letters b and e in these directories, you could type ls /usr*/[be]* to list them.

4. Working With Files and Directories

These commands can be used to: find out information about files, display files, and manipulate them in other ways (copy, move, delete). The various commands used for working with files and directories are:

a. ***file*** Find out what kind of file it is. For example, “file /bin/ls” tells us that it is a Linux executable file.
b. ***cat*** Display the contents of a text file on the screen. For example: cat file.txt would displays the file content.
c. ***head*** Display the first few lines of a text file. Example: head /etc/services
d. ***tail*** Display the last few lines of a text file. Example: tail /etc/services
e. ***tail -f*** Display the last few lines of a text file.
f. ***cp*** Copies a file from one location to another. Example: cp mp3files.txt /tmp (copies the mp3files.txt file to the /tmp directory)
g. ***mv*** Moves a file to a new location, or renames it. For example: mv mp3files.txt /tmp (copy the file to /tmp, and delete it from the original location)
h. ***rm*** Delete a file. Example: rm /tmp/mp3files.txt
i. ***mkdir*** Make Directory. Example: mkdir /tmp/myfiles/

j. ***rmdir*** Remove Directory. rmdir will only remove direcotry when it is empty. use of rm -R will remove the directory as well as any files and subdirectories as long as they are not in use. Be careful though, make sure you specify the correct directory or you can remove a lot of stuff quickly.
Example: rmdir /tmp/myfiles/

5. Commands used for searching directory

The following commands are used to find files. "ls" is good for finding files if you already know approximately where they are, but sometimes you need more powerful tools such as these:

a. ***which*** Shows the full path of shell commands found in your path. For example, if you want to know exactly where the "grep" command is located on the filesystem, you can type "which grep". The output should be something like: /bin/grep
b. ***whereis*** Locates the program, source code, and manual page for a command (if all information is available). For example, to find out where "ls" and its man page are, type: "whereis ls" The output will look something like: ls: /bin/ls /usr/share/man/man1/ls.1.gz
c. ***locate*** A quick way to search for files anywhere on the filesystem. For example, you can find all files and directories that contain the name "mozilla" by typing: locate mozilla
d. ***find*** A *very* powerful command, but sometimes tricky to use. It can be used to search for files matching certain patterns, as well as many other types of searches. A simple example is: find . -name "*.sh". This example starts searching in the current directory "." and all subdirectories, looking for files with "mp3" at the end of their names.

6. Piping and Re-Direction

Before we move on to learning even more commands, let's side-track to the topics of piping and re-direction. The basic UNIX philosophy, therefore by extension the Linux philosophy, is to have many small programs and utilities that do a particular job very well. It is the responsibility of the programmer or user to combine these utilities to make more useful command sequences.

6.1 Piping Commands Together

The pipe character, "|", is used to chain two or more commands together. The output of the first command is "piped" into the next program, and if there is a second pipe, the output is sent to the third program, etc. For example: ls -la /usr/bin | less
In this example, we run the command "ls -la /usr/bin", which gives us a long listing of all of the files in /usr/bin. Because the output of this command is typically very long, we pipe the output to a program called "less", which displays the output for us one screen at a time.

6.2 Redirecting Program Output to Files

There are times when it is useful to save the output of a command to a file, instead of displaying it to the screen. For example, if we want to create a file that lists all of the MP3 files in a directory, we can do something like this, using the ">" redirection character: ls -l /home/vic/MP3/*.mp3 > mp3files.txt. A similar command can be written so that instead of creating a new file called mp3files.txt, we can append to the end of the original file:ls -l /home/vic/extraMP3s/*.mp3 >> mp3files.txt

7. **Shortcuts**
   a. ctrl+c Halts the current command
   b. ctrl+z Stops the current command,
   c. ctrl+d Logout the current session, similar to exit
   d. ctrl+w Erases one word in the current line
   e. ctrl+u Erases the whole line
   f. !! Repeats the last command
   g. exit Logout the current session

Lab exercises

1. Execute all the commands explained so far in this manual
2. Explore the following commands along with their various options. (Some of the options are specified in the bracket)
   a. cat (variation used to create a new file and append to existing file)
   b. head and tail (-n, -c)
   c. cp (-n, -i, -f)
   d. mv (-f, -i) [try (i) mv dir1 dir2 (ii) mv file1 file2 file3 ... directory]
   e. rm (-r, -i, -f)
   f. rmdir (-r, -f)
   g. find (-name, -type)
3. List all the file names satisfying following criteria
   a. Containing only alphabets of length 4.
   b. Having maximum length 4
   c. Starting with a vowel.
   d. Has the extension .txt.
   e. Containing atleast one digit.
   f. Does not contain any of the vowels as the start letter.