CET-225
Operating Systems
Experiment # 03

Experiment Title

Programming using shell Script

Assessment of CLO(s): 04

Performed on _____

Student Name:			
Roll No.		**Group**	
Semester		**Session**	

Total (Max)	**Performance (03)**	**Viva (03)**	**File (04)**	**Total (10)**
Marks Obtained				
Remarks (if any)				

Experiment evaluated by

Instructor's Name	**Engr. Bushra Aziz**

THEORY

When a user enters commands from the command line, he is entering them one at a time and getting a response from the system. From time to time it is required to execute more than one command, one after the other, and get the final result. This can be done with a ***shell program or a shell script***. A shell program is a series of Linux commands and utilities that have been put into a file by using a text editor. When a shell program is executed the commands are interpreted and executed by Linux one after the other.
A shell program is like any other programming language and it has its own syntax. It allows the user to define variables, assign various values, and so on.

Creating and Executing a Shell Program

At the simplest level, shell programs are just files that contain one or more shell or Linux commands. These programs can be used to simplify repetitive tasks, to replace two or more commands that are always executed together with a single command, to automate the installation of other programs, and to write simple interactive applications.

Shell script is just a simple text file with ".sh" extension, having executable permission. Use chmod command to change permission.

```
linux-o5t4:/home/CL2/Desktop # ls -l
total 24
-rw-r--r-- 1 CL2  users   50 Oct  4 17:22 .directory
-rw------- 1 CL2  users  552 Oct 12 11:13 .myscript.sh.kate-swp
-rw-r--r-- 1 CL2  users 2460 Oct  4 17:22 Home.desktop
-rw-r--r-- 1 CL2  users    2 Oct 11 15:42 hello.txt
-rwxrwxrwx 1 CL2  users  155 Oct 11 16:33 hi.sh
-rwxr--r-- 1 root root     0 Oct 12 11:12 myscript.sh
-rw-r--r-- 1 CL2  users 2902 Oct  4 17:22 trash.desktop
linux-o5t4:/home/CL2/Desktop # chmod 777 myscript.sh
linux-o5t4:/home/CL2/Desktop # ls -l
total 24
-rw-r--r-- 1 CL2  users   50 Oct  4 17:22 .directory
-rw------- 1 CL2  users  552 Oct 12 11:13 .myscript.sh.kate-swp
-rw-r--r-- 1 CL2  users 2460 Oct  4 17:22 Home.desktop
-rw-r--r-- 1 CL2  users    2 Oct 11 15:42 hello.txt
-rwxrwxrwx 1 CL2  users  155 Oct 11 16:33 hi.sh
-rwxrwxrwx 1 root root     0 Oct 12 11:12 myscript.sh
-rw-r--r-- 1 CL2  users 2902 Oct  4 17:22 trash.desktop
```

Process of writing and executing a script

- Open terminal.
- Navigate to the place where you want to create script using 'cd' command.
- Cd (enter) [This will bring the prompt at Your home Directory].
- touch hello.sh (Here we named the script as hello, remember the '.sh' extension is compulsory).

- vi hello.sh (nano hello.sh) [You can use your favourite editor, to edit the script].
- chmod 744 hello.sh (making the script executable).
- sh hello.sh or ./hello.sh or bash hello.sh (running the script)

The bang line

Bang line tells the kernel that a specific shell or language is to be used to interpret the contents of the file. This is the line which is written at the beginning of every program. It starts with the bang character (#!) and goes for example like this:

#!/bin/bash

This is the link to the Bourne again shell(bash). The instructions written after this bang line will be interpreted using the above-named shell.

Writing your First Script

```
#!/bin/bash # My

first script

echo "Hello World!"
```

Using Variables

Linux shell programming is full-fledged programming language and, as such, supports various types of variables. Variables have three major types:

- **Environment variables** are part of the system environment, and they do not need to be defined. They can be used in shell programs. Some of them such as PATH can also be modified within the shell program.
- **Built-in variables** are provided by the system. Unlike environment variable they cannot be modified.
- **User variables** are defined by the user when he writes the script. They can be used and modified at will within the shell program.

Environment Variables

You can use any one of the following commands to display the environment variables and their values.

printenv
OR

printenv | less
OR

printenv | more

Built-in Variables

These are special variables that Linux provides that can be used to make decisions in a program. Their values cannot be modified.
Several other built-in shell variables are important to know about when you are doing a lot of shell programming. The following table lists these variables and gives a brief description of what each is used for.

Variable	**Use**
$#	Stores the number of command-line arguments that were passed to the shell program.
$?	Stores the exit value of the last command that was executed.
$0	Stores the first word of the entered command (the name of the shell program).
$*	Stores all the arguments that were entered on the command line ($1 $2 ...).
"$@"	Stores all the arguments that were entered on the command line, individually quoted ("$1" "$2" ...).

A list of the commonly used variables in Linux:

- $BASH
- $USER
- $BASH_VERSION
- $PATH
- $TERM etc.

User Variables

Assigning values to variables

A value is assigned to a variable simply by typing the variable name followed by an equal sign and the value that is to be assigned to the variable. For example, if you wanted to assign a value of 5 to the variable count, you would enter the following command in bash or pdksh:

```
count=5
```

With tcsh you would have to enter the following command to achieve the same results:

```
set count = 5
```

With the bash and pdksh syntax for setting a variable, you must make sure that there are no spaces on either side of the equal sign. With tcsh, it doesn't matter if there are spaces or not.

Notice that you do not have to declare the variable as you would if you were programming in C or Pascal. This is because the shell language is a non-typed interpretive language. This means that you can use the same variable to store character strings that you use to store integers. You would store a character string into a variable in the same way that you stored the integer into a variable. For example:

```
name=Garry - (for pdksh and bash)

set name = Garry - (for tcsh)
```

Accessing variable values

To access the value stored in a variable precede the variable name with a dollar sign ($). If you wanted to print the value stored in the count variable to the screen, you would do so by entering the following command:

```
echo $count
```

If you omitted the $ from the preceding command, the echo command would display the word count onscreen.

Positional parameters

The shell has knowledge of a special kind of variable called a positional parameter. Positional parameters are used to refer to the parameters that were passed to a shell program on the command line or a shell function by the shell script that invoked the function. When you run a shell program that requires or supports a number of command-line options, each of these options is stored into a positional parameter. The first parameter is stored into a variable named 1, the second parameter is stored into a variable named 2, and so forth. These variable names are reserved by the shell so that you can't use them as variables you define. To access the values stored in these variables, you must precede the variable name with a dollar sign ($) just as you do with variables you define. The following shell program expects to be invoked with two parameters. The program takes the two parameters and prints the second parameter that was typed on the command line first and the first parameter that was typed on the command line second.

```
#program reverse, prints the command line parameters out in reverse #order
```

```
echo "$2" "$1"
```

If you invoked this program by entering

```
reverse hello there
```

The program would return the following output:

```
there hello
```

Exercises:

1. Write a shell program that takes three parameter (your name,your ID and Department) and displays it on the screen.

2. Write a shell program that takes a number parameters equal to the last digit of your roll number and displays the values of the built in variables such as $#, $0, and $* on the screen.

3. Write a script that uses the **read** command which takes the input from the keyboard and assigns it as the value of the variable PERSON and finally prints it on screen.