Operating Systems

LAB 2

SHELL Programming (Part I)

Objectives:

The aim of this laboratory is to learn and practice SHELL scripts by writing small SHELL programs.
The following are the primary objectives of this lab session:

- ✓ **Understanding what is a SHELL script**
  - ❖ What is a SHELL script
  - ❖ Different kinds of SHELLs in UNIX
- ✓ **Why and where it is used**
- ✓ **First simple SHELL script**
- ✓ **SHELL variables**
  - ❖ User defined variables
  - ❖ System variables
  - ❖ Read only variables and wiping out variables
  - ❖ Assigning values to variables
  - ❖ Reading input

****
****

Understanding what is a SHELL script

Shell

A shell is a command line interpreter. It takes commands and executes them. As such, it implements a programming language. Three most widely used shells in UNIX are Bourne shell, C shell, and Korn shell.

Shell scripts

A shell script or a shell program is a series of commands put in a file and executed by the Shell. We will use shell to create shell scripts.

Why and where it is used

Why Shell scripts?

Since the user cannot interact with the kernel directly, Shell programming skills are a must to be able to exploit the power of UNIX to the fullest extent. A shell script can be used for variety of tasks and some of them are listed below.

Uses of Shell scripts

1. Customizing your work environment. For Example Every time you login, if you want to see the current date, a welcome message, and the list of users who have logged in you can write a shell script for the same.

2. Automating your daily tasks. For example, to back up all the programs at the end of the day.

3. Automating repetitive tasks.

4. Executing important system procedures, like shutting down the system, formatting a disk, creating a file system etc.

5. Performing some operations on many files.

First simple SHELL script

Create a file named SS1 (shell script 1) and type the following as content.

Example1:

```
# SS1
# Usage: SS1
ls
who
pwd
```

Save the file and type the file name in command line and the file is executed as follows.

console> SS1

SS1: Command not found.

The reason for this message is, the path of this command should be specified.

console> ./SS1
SS1: Permission Denied.

Now you do not have the permission to execute the file. That is the default permission given for any file is with out execute permission. How to know the default permission? Type the command **umask.** This command will give the masked permissions of a file while creation.

Change the permissions using **chmod** command and execute the file again.

SHELL variables

The variables in the Shell are classified as

- **User defined variables**: defined by the user for his use (e.g. age=32).
- **Environmental variables**: defined by shell for its own operations (PATH, HOME, TERM, LOGNAME, PS1, SHELL etc.).
- **Predefined variables:** reserved variables used by the shell and UNIX commands for specifying the exit status of command, arguments to the shell scripts, the formal parameters etc.

Examples:

```
name=Ali
echo $name
echo Hello $name ! , Welcome to $HOME
```

To read standard input into a shell script use the **read** command. For example:

Example2:

```
# SS2
# Usage: SS2
# An interactive shell script
echo What is your name\?
read name
echo Hello $name. Assalam-o-Alaikum.
```

This prompts the user for input, assigns this to the variable name and then displays the value of this variable to standard output.

If there is more than one word in the input, each word can be assigned to a different variable. Any words left over are assigned to the last named variable. For example:

Example3:

```
# SS3
# Usage: SS3
echo "Please enter your surname\n"
echo "followed by your first name: \c"
read name1 name2
echo "Welcome to CSE Dept., UET, $name2 $name1"
```

Example4:

```
# SS4
# Usage: SS4
# This script takes two file names and copies the first file into the second one
echo “Please Enter source file name: \c”
read source
echo “Enter the target file name :\c”
read target
cp $source $target
echo file $source  is copied into the $target
```

Command Substitution:

Format for command substitution is:
var=`command` (where ‘ ‘ is back quote)

Examples:

echo ‘date’ # It will display the output of date command
echo there are ‘who | wc –l’ users working on the system # see output of this

Arithmetic in SHELL script

Various forms for performing computations on shell variables using **expr** command are:

expr val_1 op val_2 (Where op is operator)
expr $val_1 op $val_2
val_3=’expr $val_1 op $val_2`

Examples:

```
expr 5 + 7        # Gives 12
expr 6 – 3        # Gives 3
expr 3 \* 4       # Gives  12
expr 24 / 3       # Gives  8

sum='expr 5 + 6'
echo $sum         # Gives 11
```

Example 5:

```
# SS5
# Usage: SS5
a=12
b=90
echo sum is $a + $b          # Will display sum is 12 + 90
echo sum is `expr $a + $b`   # Gives sum is 102
```

Run all the programs given in the Lab Notes, and observe the output for each program.

Practice different examples and show them in lab report.