LAB 8

Course Code: CSC 2209 Course Title: Operating Systems

Dept. of Computer Science
Faculty of Science and Technology

Lecturer No:	**08**	**Week No:**	**08**	**Semester:**	
Lecturer:	*Name & email*				

Lecture Outline

1. First Shell Program
2. Running First Program
3. Shell Variables
4. Shell Variables Rules
5. Comments and Escape Characters
6. Reading User Input
7. Operators
8. Floating Point Calculation

First Shell Program

- **At first create a file**
  - touch hello.sh [sh is not mandatory it helps text editors to differentiate shell scripts from others]
- **Open the file with any text editor**
  - vi filename.sh [here using vi text editor]
- **Print Hello World**
  - Start with **#! /bin/bash** [location of bash]
  - echo Hello World or echo “Hello World”
  - Save Code

Running First Program

- **To run a program**
  - ./filename.sh
- **Permission denied right?**
  - Check the permission details of that file
    - ls –l filename.sh [user don't have execute permission by default]
  - Change Permission
    - chmod u+x filename.sh
    - Chmod 744 filename.sh
- **Now run the program**
  - bash filename.sh [No Permission Needed]
  - or ./filename.sh [Now it will work]
- **Now its running fine**

Shell Variables

- **While using variable use $ sign before the name**
- **Don't need declaration**
- **There are two types of shell variables**
  - System Variables
  - User defined Variables
- **System Variables:**
  - Usually maintained by Operating systems
  - Written in all capitals
  - Examples:
    - $BASH [knowing the bash location]
    - $BASH_VERSION [knowing the bash version]
    - $HOME [knowing the home directory]
    - $PWD [present working directory]

Shell Variables (cont'd)

- **User defined variables Syntax**

  Name=Alex [here Name is the variable name and Alex is value]

  ***** Don't Use Space in variable assignment like below

  Name = Alex or Name =Alex or Name= Alex *****

- **Using the variable**

  echo $Name

- **Try**
  - echo My name is $Name
  - echo "My name is "$Name
  - echo "This is "$Name" Who did this!"

Shell Variables Rules

- The shell does not care about types of variables; they may store strings, integers, real numbers - anything you like.
- Loosely coupled
- Variables in the Bourne shell do not have to be declared
- Variable Name cannot be started with numeric values
- Space is not allowed in name

Comments and Escape Characters

- **Comments**
  - Comments are used for documentation
  - It's a good programming practice
  - # is used comment any line in shell script
  - Example: # This is a comment

```
echo hello # this is a comment
```

- **Escape Characters**
  - \
  - Example: Hello \"World\" to print Hello "world"

Reading User Input

- **read command takes input from the keyboard**
- **Syntax**
  - read variablename [input will be saved in variablename]
- **Using the taken input**
  - Use the variablename with $ sign like $variablename
  - Example: The entered value is $variablename
- **Multiple values input**
  - read variablename1 variablename2 …
  - While giving multiple input use space to separate don't press enter
- **Taking input in same line (not in next line) // p flag**
  - read –p "Enter Variable" variablename (read –p comment variablename)
- **Taking silent input like password**
  - read –s "Enter Variable" variablename [silent in new line]
  - read –sp "Enter Variable" variablename [silent in same line]

Operators

- **Arithmetic Operator**
  - BASH don’t have any mechanism to perform arithmetic operations
  - It uses expr [external program] to perform
  - expr only performs integer operations
  - Floating value calculations are discussed later

    ****Most Important Things to Remember****
  - There must be spaces between operators and expressions

    2+2 is not correct it should be 2 + 2
  - It should be written like `expr 2 + 2` or $(expr 2 + 2)
  - ` This symbol is called backtick

Some Examples

- **Assuming a=20 b=10**
- **Addition will be sum=$(expr $a + $b) or**

  sum=`expr $a + $b`
- **Subtraction will be sub=$(expr $a - $b) or**

  sub=`expr $a - $b`
- **Multiplication will be multi=$(expr $a * $b) or**

  multi=`expr $a \* $b`
- **Division will be div=$(expr $a / $b) or**

  div=`expr $a / $b`
- **Modulus will be mod==$(expr $a % $b) or**

  mod=`expr $a % $b`

Operators

- Relational Operators
  - Bourne Shell supports the following relational operators that are specific to numeric values.
  - Following operators will not work for strings.

All relational operators must be inside square braces with spaces around them

[$a == $b] [CORRECT]

[$a == $b] [WRONG]

[$a==$b] [WRONG]

Operators

- **Relational operators**
  - -eq or == to check equality of 2 number [$a- eq $b]
  - -ne or != to check inequality of 2 number [$a -ne $b]
  - -gt or > to check left operand is greater or not [$a -gt $b]
  - -lt or < to check left operand is greater or not [$a -lt $b]
  - -ge or >= to check left operand is greater or equal [$a -ge $b]
  - -le or <= to check left operand is lesser or equal [$a -le $b]

Operators

- **String Operators**
  - = checks the equality [$a = $b]
  - != checks the inequality [$a != $b]
  - -z Checks if the given string operand size is zero; if it is zero length, then it returns true. [-z $a]
  - -n Checks if the given string operand size is non-zero; if it is nonzero length, then it returns true. [-n $a]

Floating Point Calculation

- Floating points can not be evaluated with expr it can be calculated with a bc utility
- num1=20.5
- num2=10
- echo “$num1+$num2” | bc
- For storing values into a variable
- num3=$(bc <<< “$num1+num2”)

*****Remember the spaces and * will not be applicable here*****

It uses simple equation strategies

Books

- Unix Shell Programming
  - Written by Yashavant P. Kanetkar