Lab No. 10
Writing & Executing Shell Scripts, I/O, Variables, and Operators

Objective
This lab introduces the fundamental concepts of shell scripting along with its basic constructs.

Activity Outcomes:
On completion of this lab students will be able to:

- Understand the process of writing and executing shell scripts
- Use input/output commands
- Use variables and operators

Instructor Notes
As pre-lab activity, read Chapter 24, 26, 28 & 34 from the book "The Linux Command Line", William E. Shotts, Jr.

1) Useful Concepts

Introduction to Shell
Shell is an Interface between the user and the operating system. It is simply a program that is used to start other programs. It takes the commands from the keyboard and gives them to the operating system to perform the particular task. There are many different shells, but all derive several of their features from the Bourne shell, a standard shell developed at Bell Labs for early versions of UNIX. Linux uses an enhanced version of the Bourne shell called bash or the "Bourne-again" shell. The bash shell is the default shell on most Linux distributions, and /bin/sh is normally a link to bash on a Linux system. Other examples of Linux shells include: Korn, C Shell, tcsh, bash, zsh, etc.

Shell Script
In the simplest terms, a shell script is a file containing a series of commands. The shell reads this file and carries out the commands as though they have been entered directly on the command line. Shell scripts are the equivalent of batch fi les in MS-DOS, and can contain long lists of commands, complex flow control, arithmetic evaluations, user-defined variables, user-defined functions, and sophisticated condition testing. The basic advantage of shell scripting includes:

- Shell script can take input from user, file and output them on screen.
- Useful to create our own commands.
- Save lots of time.
- To automate some task of day today life.
- System Administration part can be also automated.

Writing and Executing Shell Script
We need to perform following steps to create and execute shell script

1. **Write Script:** Shell scripts are ordinary text files. We can use any text editor such as gedit to write shell script

2. **Make the shell script executable:** next step is to make the script executable. We can do it using the chmod command. For example if the script file is saved with the name test then me can make it executable in the following way:

```
sudo  chmod  777  test
```

3. **Place the script at some suitable place:** The shell automatically searches certain directories for executable files when no explicit pathname is specified. For maximum convenience, we will place our scripts in these directories. For example, we can move the executable file test to /usr/bin directory using the following command:

```
sudo  mv test /usr/bin
```

Now, we can execute the test file just like normal commands.

Writing First Shell Script:

Now, we write a simple Shell script to demonstrate the above mention steps. This script just shows a *"Hello World"* message on the terminal screen. Open the gedit and write the following code.

```
1 #!/bin/bash
2 #this is the first script
3 echo "Hello World"
```

The first line of the code tells about the shell for which we are writing the script. The second line is a comment while the third line displays the message on the terminal screen. After writing this code save it with the name test. Now, make it executable using the chmod command and move it to /usr/bin directory. Now, we can run this script by just writing test on the command line.

Shell Variables

Shell variables can be categorized as system-defined, parameter or user-defined. The details are given below:

1. **System defined** variables are already defined and included in the environment when we login. Their names are in upper case letters. Some of them are as follows:
   HISTFILE – filename of the history file. Default is $HOME/.sh history.
   HISTSIZE – Maximum number of commands retained in the history file
   HOME – The pathname of your home directory
   IFS – Inter Filed Separator. Zero or more characters treated by the shell as delimiters in parsing a command line into words. Default – Blank, Tab and Newline in succession
   PATH – List of directories separated by colon that are searched for executable
   PWD – The present working directory
   RANDOM – This is a random number from 0 – 32767. Its value will be different every time you examine it.
   SHELL – The pathname of the shell

$# – The number of parameters passed
$$ – The PID of the parent process i.e. shell
$? – exit status of last command run

2. **Parameter variables** contain the values of the parameters passed to a shell script.
   - $0 Path of the program
   - $1, $2 … Store the parameters given to the script
   - $* A list of all parameters, separated by the character defined in IFS
   - $@ Same as $* except the parameters are separated by space character
3. **User defined** variables are subject to the following rules:
   - Variable names can begin with a letter or an underscore and can contain an arbitrary number of letters, digits and underscores
   - No arbitrary upper limit to the variables you can define or use
   - It is not necessary to declare a variable before using it
   - Variables are of loosely typed
   - A variable retains its value from the time it is set – whether explicitly by you or implicitly by the shell – until its value is changed or the shell exits
   - To retrieve the value, precede the variable name with a **dollar sign ($)**
   - Quotes are used to specify values which contain spaces or special characters.
     - Double quotes (") prevent shell interpretation of special characters except $ and `.
     - Single quotes (') prevent shell interpretation of all special characters.
     - The backslash (\) prevents the shell from interpreting the next character specially.

Input/Out-put Commands

echo- The echo command is used to display a line of text/string on standard output or a file. We use quotation marks to display text/string. To display the value of a variable, we need to place $ symbol before the variable name.

```
echo "The value of x is $x"
```

read command- The read built-in command is used to read a single line of standard input i.e. keyboard. This command can be used to read keyboard input or, when redirection is employed, a line of data from a file. The syntax of read command is:

```
read  -options  arguments
```

Where options is one or more of the available options listed below and variable is the name of one or more variables used to hold the input value. If no variable name is supplied, the shell variable REPLY contains the line of data.

```
#! /bin/bash
echo "Please Enter a number"
read num
echo "You Entered $num"
```

Common options for read command are:

Option	Description
-a	*array* Assign the input to *array*, starting with index zero.
-n num	Read *num* characters of input, rather than an entire line.
-p prompt	-p *prompt* Display a prompt for input using the string *prompt*.
-t second	Timeout. Terminate input after *seconds*. read returns a non-zero exit status if an input times out
-u fd	Use input from file descriptor *fd*, rather than standard input

Operators

An operator is a symbol that usually represents an action or process. There various types of operators supported by bash shell.
Arithmetic Operators- Arithmetic operators are used to perform arithmetic operations on variables. Following arithmetic operators are available in bash shell

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Integer Division
%	Remainder
**	Exponentiation

Writing Arithmetic Expressions- We use arithmetic expansion to perform arithmetic operations. It can be done using the following syntax

```
$((Arithmetic Expression))
```

In the following example, we perform some basic arithmetic operations

```
#! /bin/bash
echo "Please Enter First Number"
read num1
echo "Please Enter the Second Number"
read num2
echo "The sum is $(($num1+$num2))"
echo "The multiplication is $(($num1*$num2))"
echo "Square of sum of number is $(($(($num1+$num2))**2))"
```

We can also use expr command to write arithmetic expressions. The syntax is as given below:

```
echo 'expr $num1 + $num2'
```

Note: There must be a space between operator and operands.

Assignment Operators

Assignment operators are used to assign values to variables. The following assignment operators are available in bash shell

Operator	Operation
=	Simple assignment. Assigns value to parameter. Example: a=$b Note: there should be no space between = operator and operands
parameter+=value	Addition. Equivalent to parameter = parameter + value.
parameter-=value	Subtraction. Equivalent to parameter = parameter - value.
parameter*=value	Multiplication. Equivalent to parameter = parameter * value.
parameter/=value	Integer Division. Equivalent to parameter = parameter / value.
parameter%=value	Remainder. Equivalent to parameter = parameter % value.
parameter++	Post increment
Parameter--	Post decrement
++parameter	Pre increment
-- parameter	Post increment

Comparison Operators

These operators are used to compare the values of variables. Assume variable a holds 10 and variable b holds 20 then:

Operator	Description	Example
-eq	Checks if the value of two operands are equal or not; if yes, then the condition becomes true.	[$a -eq $b] is not true.
-ne	Checks if the value of two operands are equal or not; if values are not equal, then the condition becomes true.	[$a -ne $b] is true.
-gt	Checks if the value of left operand is greater than the value of right operand; if yes, then the condition becomes true.	[$a -gt $b] is not true.
-lt	Checks if the value of left operand is less than the value of right operand; if yes, then the condition becomes true.	[$a -lt $b] is true.
-ge	Checks if the value of left operand is greater than or equal to the value of right operand; if yes, then the condition becomes true.	[$a -ge $b] is not true.
-le	Checks if the value of left operand is less than or equal to the value of right operand; if yes, then the condition becomes true.	[$a -le $b] is true.

Logical Operators

These operators are used to perform logical operations. Suppose the values of variable a and b are 10 and 20 respectively then:

Operator	Description	Example
!	This is logical negation. This inverts a true condition into false and vice versa.	[! false] is true.
-o	This is logical OR. If one of the operands is true, then the	[$a -lt 20 -o $b -gt 100]

	condition becomes true.	is true.
-a	This is logical AND. If both the operands are true, then the condition becomes true otherwise false.	[$a -lt 20 -a $b -gt 100] is false.

File Operators

We have several operators that can be used to test file properties. Assume a variable file holds an existing file name "test" the size of which is 100 bytes and has read, write and execute permission on.

Operator	Description	Example
-b file	Checks if file is a block special file; if yes, then the condition becomes true.	[-b $file] is false.
-c file	Checks if file is a character special file; if yes, then the condition becomes true.	[-c $file] is false.
-d file	Checks if file is a directory; if yes, then the condition becomes true.	[-d $file] is not true.
-f file	Checks if file is an ordinary file as opposed to a directory or special file; if yes, then the condition becomes true.	[-f $file] is true.
-g file	Checks if file has its set group ID (SGID) bit set; if yes, then the condition becomes true.	[-g $file] is false.
-k file	Checks if file has its sticky bit set; if yes, then the condition becomes true.	[-k $file] is false.
-p file	Checks if file is a named pipe; if yes, then the condition becomes true.	[-p $file] is false.
-t file	Checks if file descriptor is open and associated with a terminal; if yes, then the condition becomes true.	[-t $file] is false.
-u file	Checks if file has its Set User ID (SUID) bit set; if yes, then the condition becomes true.	[-u $file] is false.
-r file	Checks if file is readable; if yes, then the condition becomes true.	[-r $file] is true.
-w file	Checks if file is writable; if yes, then the condition becomes true.	[-w $file] is true.
-x file	Checks if file is executable; if yes, then the condition becomes true.	[-x $file] is true.
-s file	Checks if file has size greater than 0; if yes, then condition becomes true.	[-s $file] is true.
-e file	Checks if file exists; is true even if file is a directory but exists.	[-e $file] is true.

2) Solved Lab Activities

Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
1	***10***	***Low***	***CLO-6***
2	***10***	***Low***	***CLO-6***
3	***10***	***Low***	***CLO-6***
4	***10***	***Low***	***CLO-6***
5	***10***	***Low***	***CLO-6***

Activity 1:

Write a shell script that gets two numbers from user and perform basic arithmetic operations (+, -, *, /, %, **) on these numbers.

Solution:

Code

*scripts (/usr/bin) - gedit

Open

```
#!/bin/bash
echo "Please Enter the First Number"
read num1
echo "Please Enter the Second Number"
read num2
s=$(($num1+$num2))  #addition
echo "Sum of $num1 and $num2 is $s"
d=$(($num1-$num2))  #subtraction
echo "Dfiference of $num1 and $num2 is $d"
m=$(($num1*$num2))  #multiplication
echo "Product of $num1 and $num2 is $m"
div=$(($num1/$num2))  #division
echo "Division of $num1 and $num2 is $div"
r=$(($num1%$num2))  #remainder
echo "Remainder of $num1 and $num2 is $r"
e=$(($num1**$num2)) #exponentiation
echo "$num1 raise to power $num2 is $e"
```

Out-put

test_fa21 [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

Terminal File Edit View Search Terminal Help

```
ubuntu@ubuntu:~$ sudo chmod 777 scripts
ubuntu@ubuntu:~$ sudo mv scripts /usr/bin
ubuntu@ubuntu:~$ scripts
Please Enter the First Number
18
Please Enter the Second Number
6
Sum of 18 and 6 is 24
Dfiference of 18 and 6 is 12
Product of 18 and 6 is 108
Division of 18 and 6 is 3
Remainder of 18 and 6 is 0
18 raise to power 6 is 34012224
[2]+  Done                    gedit /usr/bin/scripts
ubuntu@ubuntu:~$
```

Activity 2:

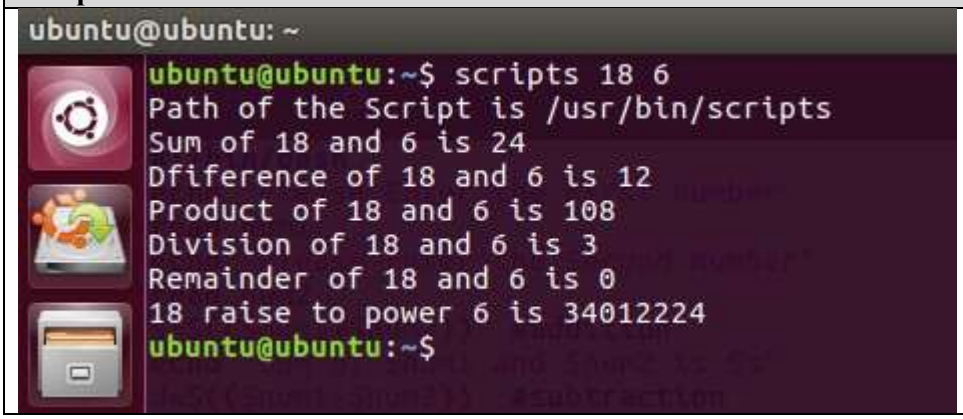
Write a shell script that gets two numbers at the command line. First it displays the path of the script and perform basic arithmetic operations (+, -, *, /, %, **) on these numbers.

Solution:

Code



Out-put

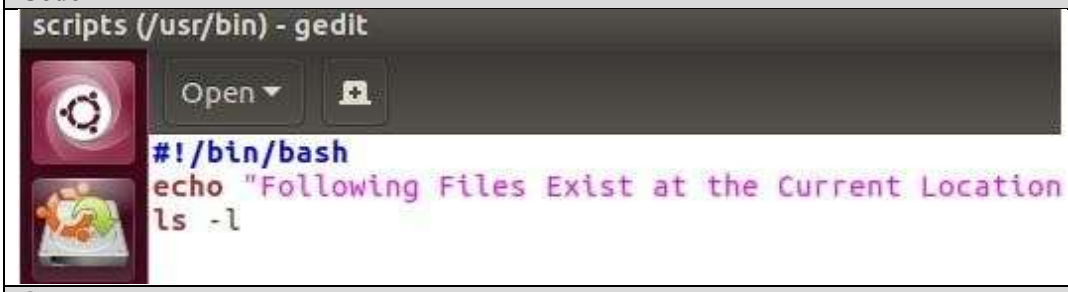


Activity 3:

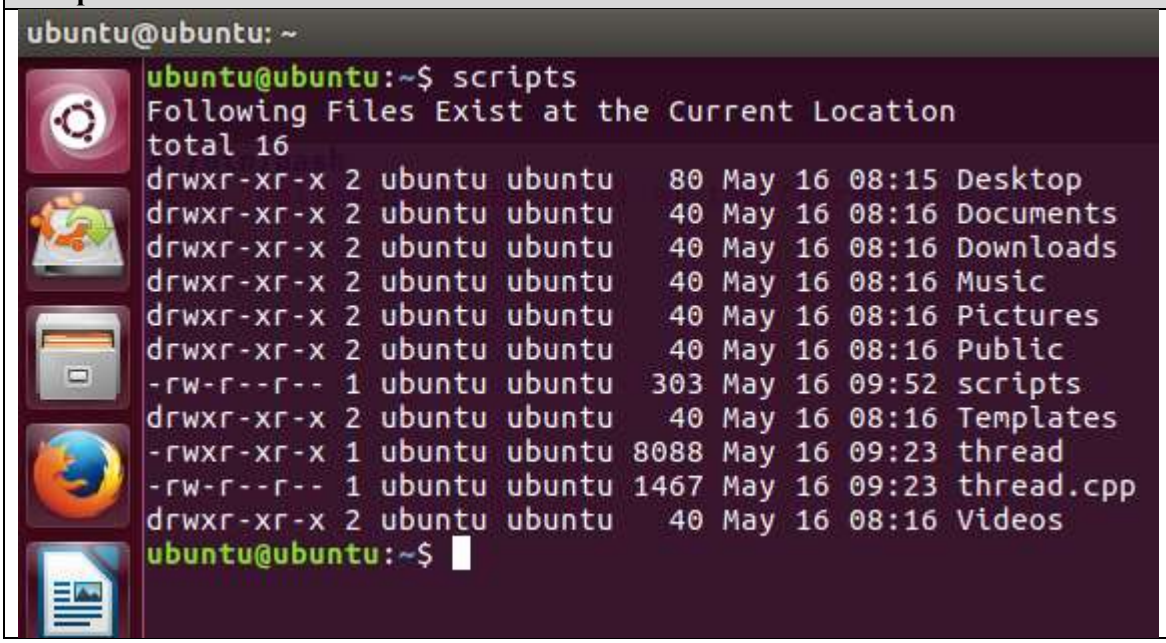
Write a shell script that creates a long list of all directories exists at the current location

Solution:

Code



Out-put

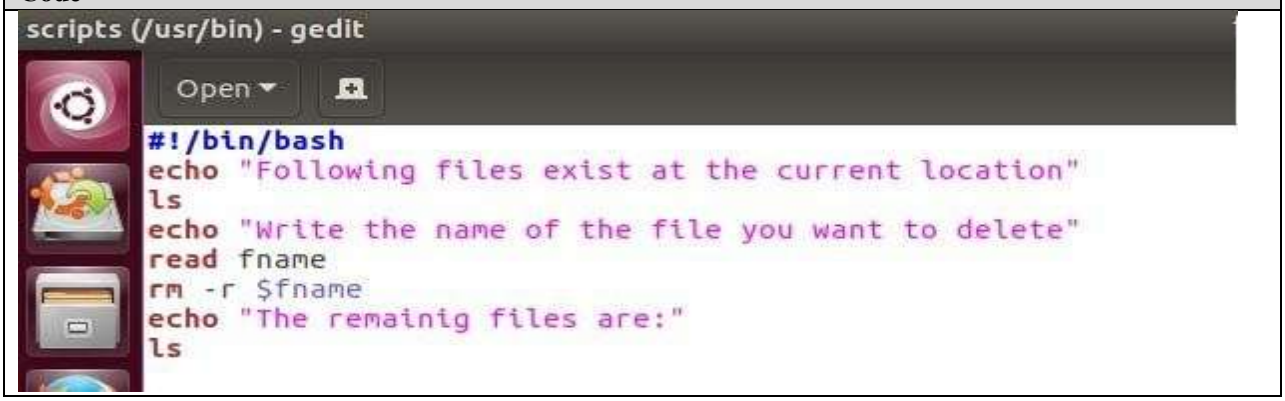


Activity 4:

Write a shell script that asks the user to enter the file name and deletes that file. After deleting the file it shows a success message and also displays the list of the remaining files

Solution:

Code



Out-put

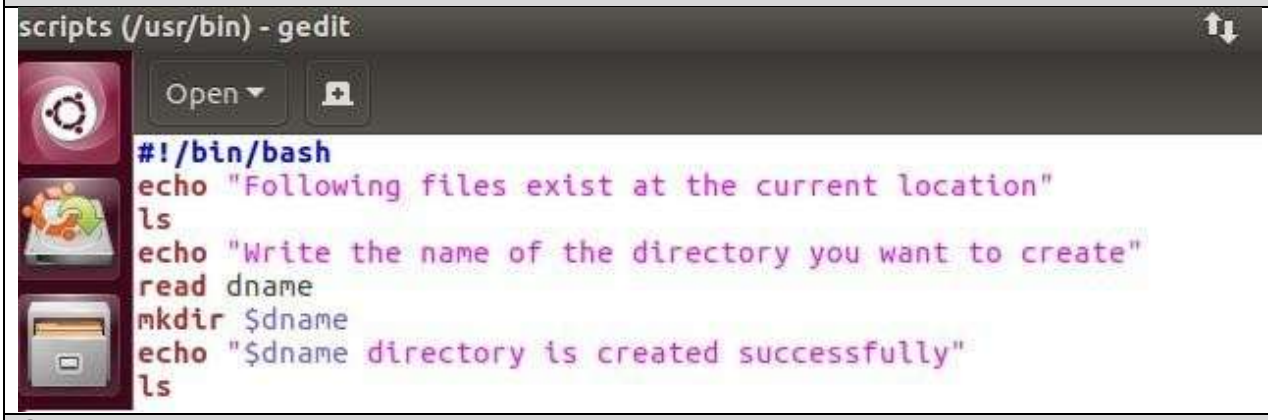


Activity 5:

Write a shell script that modifies the basic mkdir command as: first it show the list of existing files on the current location. Then it asks the users to enter the name of the directory; and then creates that directory and display a success message.

Solution:

Code



Out-put



3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficult and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Task 1:

Write a shell script that asks the user to enter the name of the directory and the destination; and then creates a directory at the given destination. Also display a success message along with the list of directories only.

Task 2:

In this task you have to re-write the cp command in such a way that your script asks the users to enter the path of files to be copied and the destination; and then copies all of the files at the destination. Also display a success message along with the list of directories only.

Task 3:

Write a shell script that takes a file path and a search string as input; and displays all of the lines that contain that string.

Task 4:

Write a shell script that takes a list of text files as input and merges all those files into a single file; and displays its contents in the less application.

Lab Exercise and Summary

Summary should cover Introduction, Procedure, Data Analysis and Evaluation

LABORATORY SKILLS ASSESSMENT (Psychomotor)
Total Marks: 100

(Max Marks)	Level 1 0% ≤ S < 50%	Level 2 50% ≤ S< 70%	Level 3 70% ≤ S< G0%	Level 4 G0%≤ S ≤100%	Score (S)
Procedural Awareness (20)	Selects inappropriate Linux commands, shell scripting techniques, or process management methods.	Selects and applies partially appropriate Linux commands and techniques	Selects and applies considerably appropriate Linux commands and techniques.	Selects and applies completely appropriate Linux commands and techniques	
Practical Implementation (20)	Makes major critical errors in executing Linux commands, scripting, and system processes.	Makes numerous critical errors in executing commands and process management.	Makes minor non-critical errors in executing Linux commands and system operations.	Executes Linux commands and manages processes correctly with no errors.	
Process Management and Shell Scripting (20)	Program logic contains major errors with incorrect or contradictory script flow.	Program logic has some errors with occasional contradictions in process execution.	Program logic is mostly correct but may contain occasional redundancy or minor errors.	Program logic is completely correct with no contradictions or redundant processes.	
Syntax Correctness and Results (20)	Program does not follow proper syntax for Linux commands and shell scripting, leading to incorrect outputs	Program partially follows proper syntax, producing correct results for few inputs.	Program adequately follows proper syntax, producing correct results for most inputs.	Program fully follows proper syntax, producing accurate results for all inputs.	
Use of OS Tools (10)	Uses OS tools (like terminal, process manager) with limited competence.	Uses OS tools with some competence.	Uses OS tools with considerable competence.	Uses OS tools proficiently with a high degree of competence.	
Safety (10)	Requires constant reminders to follow system safety procedures (e.g., file permissions, process handling).	Requires some reminders to follow system safety procedures.	Follows system safety procedures with minimal reminders.	Routinely follows system safety procedures.	
				Marks Obtained	

LABORATORY SKILLS ASSESSMENT (Affective)
Total Marks: 40

Marks)	Level 1 0% ≤ S < 50%	Level 2 50% ≤ S < 70%	Level 3 70% ≤ S < G0%	Level 4 G0% ≤ S ≤ 100%	Score (S)
Attitude s Engagement (5)	Shows little interest in lab activities; does not participate actively.	Participates occasionally but lacks enthusiasm and consistency.	Engages actively in most lab activities with interest.	Highly motivated, participates enthusiastically, and shows a proactive approach	
Responsibility s Punctuality (5)	Frequently misses deadlines and is often late to lab sessions.	Occasionally late or misses deadlines but tries to catch up.	Submits work on time and attends lab sessions regularly.	Always punctual, meets deadlines, and takes full responsibility for assigned tasks.	
Collaboration s Teamwork (10)	Rarely collaborates, struggles to work in a team, and does not contribute effectively.	Works with team members occasionally but struggles with communication.	Cooperates well, contributes effectively, and maintains professional interactions.	Actively engages in teamwork, supports peers, and demonstrates excellent collaboration.	
Communicatio n s Presentation Skills (10)	Struggles to explain concepts, unclear verbal/written communication.	Communicates ideas with some clarity but lacks confidence or coherence.	Presents ideas effectively with minor issues in clarity or structure.	Communicates clearly, confidently, and effectively in all aspects of lab work.	
Report Quality (10)	Report contains many errors.	Report is somewhat organized with some spelling or grammatical errors.	Report is well organized and cohesive but contains some grammatical errors.	Report is well organized and cohesive and contains no grammatical errors. Presentation seems polished.	
				Marks Obtained	