Experiment No. 6

Title: Keyboard Input and Shell Arithmetic in Linux

Batch: B2 **Roll No: 16010420117** **Experiment No: 6**

Aim: Implementation of Keyboard Input and Shell Arithmetic.

Resources needed: Any open source OS / Cocalc online editor

Theory:
Pre lab/Prior concepts:

Up to now, our scripts have not been interactive. That is, they did not require any input from the user. In this lesson, we will see how your scripts can ask questions, and get and use responses.

read

To get input from the keyboard, you use the read command. The read command takes input from the keyboard and assigns it to a variable. Here is an example:

```
#!/bin/bash

echo -n "Enter some text > "
read text
echo "You entered: $text"
```

As you can see, we displayed a prompt on line 3. Note that "-n" given to the echo command causes it to keep the cursor on the same line; i.e., it does not output a carriage return at the end of the prompt.

Next, we invoke the read command with "text" as its argument. What this does is wait for the user to type something followed by a carriage return (the Enter key) and then assign whatever was typed to the variable text.

Here is the script in action:

```
[me@linuxbox me]$ read_demo.bash
Enter some text > this is some text
You entered: this is some text
```

If you don't give the read command the name of a variable to assign its input, it will use the environment variable REPLY.

The read command also takes some command line options. The two most interesting ones are -t and -s. The -t option followed by a number of seconds provides an automatic

Time out for the read command. This means that the read command will give up after the specified number of seconds if no response has been received from the user. This option could be used in the case of a script that must continue (perhaps resorting to a default response) even if the user does not answer the prompts.

The -s option causes the user's typing not to be displayed. This is useful when you are asking the user to type in a password or other security related information.

Arithmetic

Since we are working on a computer, it is natural to expect that it can perform some simple arithmetic. The shell provides features for integer arithmetic.

What's an integer? That means whole numbers like 1, 2, 458, -2859. It does not mean fractional numbers like 0.5, .333, or 3.1415. If you must deal with fractional numbers, there is a separate program called bc which provides an arbitrary precision calculator language.

Variations

Arithmetic expansion with backticks (often used in conjunction with expr)

```
z=`expr $z + 3`                        # The 'expr' command performs the
expansion.
```

Arithmetic expansion with double parentheses, and using ~~let~~ .The use of backticks (backquotes) in arithmetic expansion has been superseded by double parentheses -- ((...)) and $((...)) -- and also by the very convenient let construction.

```
z=$(($z+3))

z=$((z+3))                                    #   Also correct.

                                              #   Within double
parentheses,

                                              #+ parameter
dereferencing

                                              #+ is optional.


# $((EXPRESSION)) is arithmetic expansion.   #   Not to be confusedwith

#+ command
substitution.
```

```
# You may also use operations within double parentheses without
assignment.

    n=0
    echo "n = $n"                                            # n = 0

    (( n += 1 ))                                             # Increment.
# (( $n += 1 )) is incorrect!
echo "n = $n"                                                # n = 1

let z=z+3
let "z += 3"   #   Quotes permit the use of spaces in variableassignment.
#   The 'let' operator actually performs arithmetic
evaluation,
#+ rather than expansion.
```

```
#!/bin/bash

# Demonstrating some of the uses of 'expr'
# =======================================

echo

# Arithmetic Operators
# ----------------------------

echo "Arithmetic Operators"
echo
a=`expr 5 + 3`
echo "5 + 3 = $a"

a=`expr $a + 1`
echo
```

```
echo "a + 1 = $a"
echo "(incrementing a variable)"

a=`expr 5 % 3`# modulo
echo
echo "5 mod 3 = $a"

echo echo
```

Activities:

1. Write shell can perform a variety of common (like +,-,*, /,% etc) arithmetic operations.
2. Write shell program that formats an arbitrary number of seconds into hours and minutes

---

Results: Perform the activity.
The assignment submitted should be e- media saved as <Roll No_Batch No_Date>

This file must contain on the top:
Name:
Roll No.
Exp No.s
Batch:
Date:

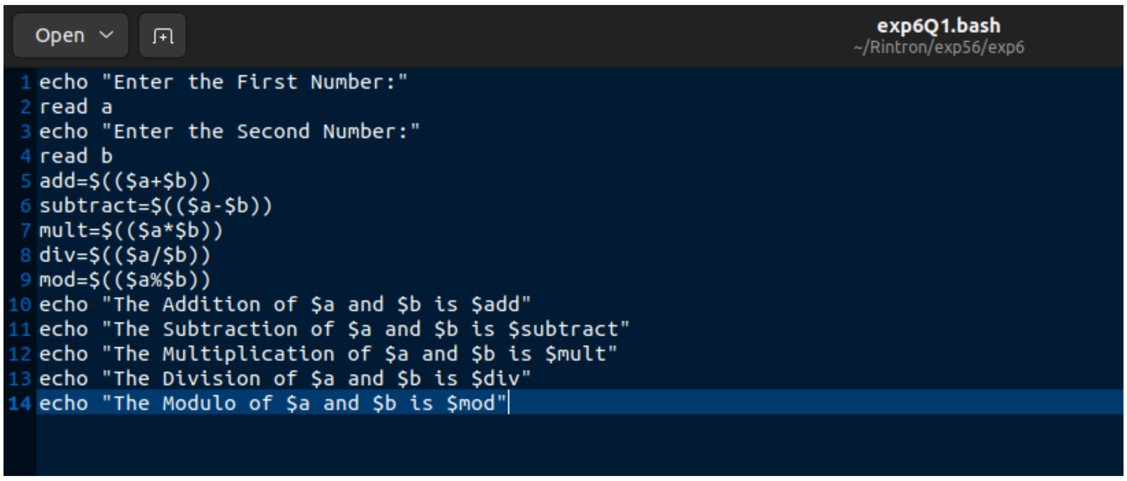
And Students have to upload this document electronically.

---

1. Write shell can perform a variety of common (like +,-,*, /,% etc) arithmetic operations.

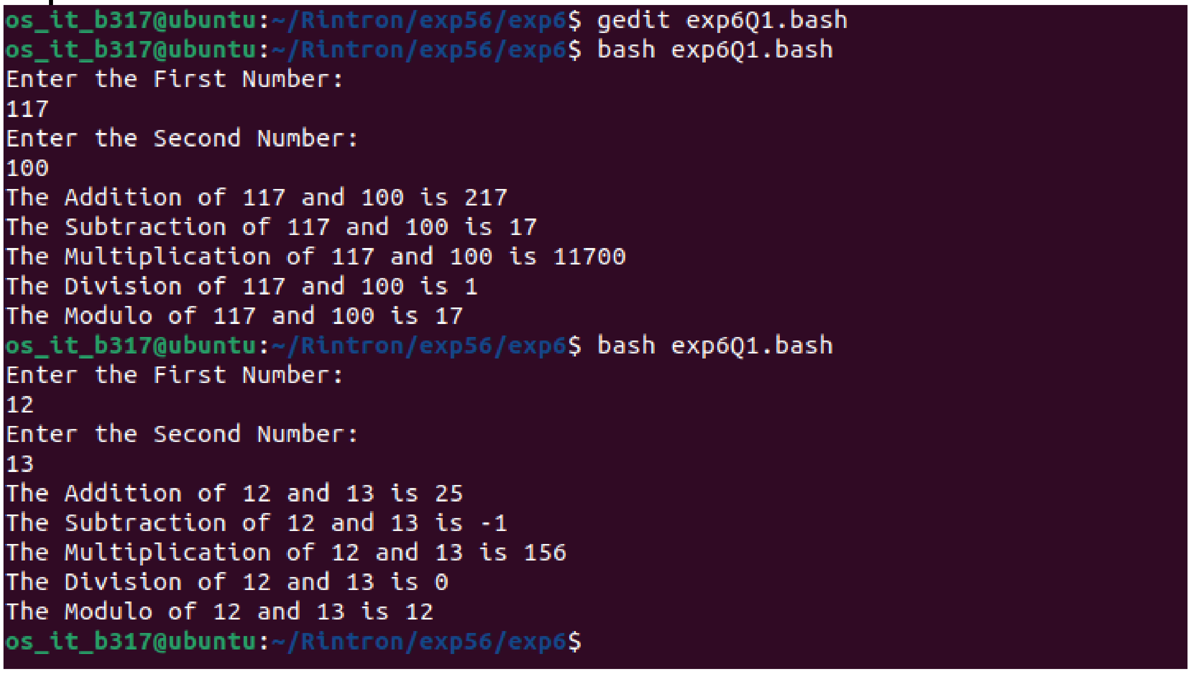
Ans:

Code:



```
echo "Enter the First Number:"
read a
echo "Enter the Second Number:"
read b
add=$(($a+$b))
subtract=$(($a-$b))
mult=$(($a*$b))
div=$(($a/$b))
mod=$(($a%$b))
echo "The Addition of $a and $b is $add"
echo "The Subtraction of $a and $b is $subtract"
echo "The Multiplication of $a and $b is $mult"
echo "The Division of $a and $b is $div"
echo "The Modulo of $a and $b is $mod"
```

Output:



```
os_it_b317@ubuntu:~/Rintron/exp56/exp6$ gedit exp6Q1.bash
os_it_b317@ubuntu:~/Rintron/exp56/exp6$ bash exp6Q1.bash
Enter the First Number:
117
Enter the Second Number:
100
The Addition of 117 and 100 is 217
The Subtraction of 117 and 100 is 17
The Multiplication of 117 and 100 is 11700
The Division of 117 and 100 is 1
The Modulo of 117 and 100 is 17
os_it_b317@ubuntu:~/Rintron/exp56/exp6$ bash exp6Q1.bash
Enter the First Number:
12
Enter the Second Number:
13
The Addition of 12 and 13 is 25
The Subtraction of 12 and 13 is -1
The Multiplication of 12 and 13 is 156
The Division of 12 and 13 is 0
The Modulo of 12 and 13 is 12
os_it_b317@ubuntu:~/Rintron/exp56/exp6$
```

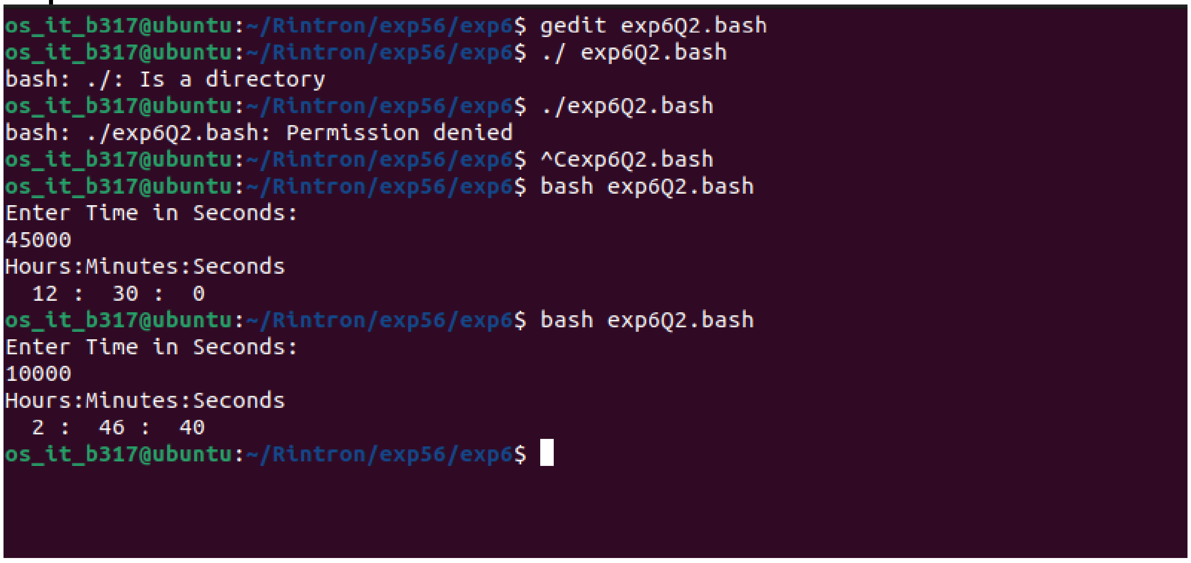
2. Write shell program that formats an arbitrary number of seconds into hours and minutes

Ans:

Code:

```
os_it_b317@ubuntu:~/Rintron/exp56/exp6$ bash exp6Q2.bash
Enter Time in Seconds:
45000
Hours:Minutes:Seconds
  12 :  30 :  0
os_it_b317@ubuntu:~/Rintron/exp56/exp6$ bash exp6Q2.bash
Enter Time in Seconds:
10000
Hours:Minutes:Seconds
  2 :  46 :  40
os_it_b317@ubuntu:~/Rintron/exp56/exp6$
```

Output:



```
os_it_b317@ubuntu:~/Rintron/exp56/exp6$ gedit exp6Q2.bash
os_it_b317@ubuntu:~/Rintron/exp56/exp6$ ./ exp6Q2.bash
bash: ./: Is a directory
os_it_b317@ubuntu:~/Rintron/exp56/exp6$ ./exp6Q2.bash
bash: ./exp6Q2.bash: Permission denied
os_it_b317@ubuntu:~/Rintron/exp56/exp6$ ^Cexp6Q2.bash
os_it_b317@ubuntu:~/Rintron/exp56/exp6$ bash exp6Q2.bash
Enter Time in Seconds:
45000
Hours:Minutes:Seconds
  12 :   30 :   0
os_it_b317@ubuntu:~/Rintron/exp56/exp6$ bash exp6Q2.bash
Enter Time in Seconds:
10000
Hours:Minutes:Seconds
  2 :   46 :   40
os_it_b317@ubuntu:~/Rintron/exp56/exp6$
```

Outcomes: CO4: Demonstrate open source standards usage.

Conclusion:

<Students should write about the concluding remarks of the experiment themselves>.

Grade: AA/AB/BB/BC/CC/CD/DD

Signature of faculty in-charge with date

References:

Books/ Journals/ Websites:

1. Richard Blum and Christine Bresnahan, “Linux Command Line & Shell Scripting”, IInd Edition edition, Wiley, 2012.