Birla Institute of Technology & Science, Pilani, K. K Birla Goa Campus
Computer Programming (CS F111)
Second Semester 2017-2018
Lab-03 (UNIX- Shell scripting)

Objectives

1. Introduction
2. More Decision Making Constructs in Shell
3. Looping Constructs in Shell
4. Exercises with solution
5. Additional Exercises

---

1. Introduction

In the last lab (lab 2) we learned how to write an interactive shell script, how to use system defined variables and how to define and use user defined variables. We also learned the usage of one of the decision making constructs. In this lab we are going to focus on more powerful decision making constructs available in Linux shell and how to use them to write a powerful shell scripts.

2. More Decision Making Constructs in Shell

2.1. Nested if-else-fi

You can write the entire if-else construct within either the body of if statement of the body of else statement. This is called the nesting of ifs. Consider a file named **isPos_nestedif** with the following content.

```
if [ $# -eq 0 ]
then
     echo "You must supply one integer"
else
     if test $1 -ge 0
     then
         echo "$1 number is positive"
     else
         echo "$1 number is negative"
     fi
fi
```

Run the above shell script as follows:

chmod +x isPos_nestedif
sh isPosnestedif 20
20 number is positive

Note that Second *if-else* construct is nested in the first *else* statement. If condition in the first '*if*' statement is false the condition in the second '*if*' statement is checked. If it is false as well the final *else* statement is executed.

You can use the nested *if*s as follows also:
Syntax:

```
if condition
then
     if condition
     then
          do this
     else
          do this
     fi
else

          do this
fi
```

2.2. Multilevel if-then-else

Syntax:

```
if condition
then
    condition is zero (true - 0)
    execute all commands up to elif statement
elif condition1
then
    condition1 is zero (true - 0)
    execute all commands up to elif statement
else
    None of the above condition, condtion1, condtion2 are
    true (i.e. all of the above nonzero or false) execute
    all commands up to fi
fi
```

For multilevel if-then-else statement try the following script in a file named **isPos_Multilevelif**:

```
if [ $# -eq 0 ]
then
     echo "You must supply one integer"
elif test $1 -gt 0
then
    echo "$1 number is positive"
elif test $1 -lt 0
then
    echo "$1 number is negative"
elif test $1 -eq 0
then
    echo "$1 number is zero"
else
    echo "Opps! $1 is Not a Number, supply a Number"
fi
```

Try above script as follows:
chmod +x isPos_Multilevelif

sh isPos_Multilevelif 20

sh isPos_Multilevelif 0

sh isPos_Multilevelif -20

sh isPos_Multilevelif CP

2.3. The case Statement

The case statement is a good alternative to multilevel if-then-else-fi statement. It enables you to match several values against one variable. It's easier to read and write.

Syntax:

```
case  $variable-name  in
     pattern1)      command
                    command;;
     pattern2)      command
                    command;;
     patternN)      command
                    command;;
     *)             command
                    command;;
 esac
```

The *$variable-name* is compared against the patterns until a match is found. The shell then executes all the statements up to the two semicolons that are next to each other. The default is *) and it's executed if no match is found. For example write script named **vehicle** as follows:

```
rental=$1
case $rental in
"car") echo "For $rental Rs.400 per day";;
"van") echo "For $rental Rs.500 per day";;
"jeep") echo "For $rental Rs.300 per day";;
"bicycle") echo "For $rental Rs.20 per day";;
*) echo "Sorry, I cannot get a $rental for you";;
esac
```

Save it and run it as follows:
chmod +x vehicle
sh vehicle van
sh vehicle car
sh vehicle Maruti-800

Note that esac is always required to indicate end of case statement.

3. Looping Constructs in Shell

Loop is defined as:
"A Program *can repeat particular set of instructions again and again, until particular condition satisfies. A group of instructions that is executed repeatedly is called a loop.*"

Bash supports:

- for loop
- while loop

Note that in each and every loop,

(a) First, the variable used in loop condition must be initialized, then execution of the loop begins.
(b) A test (condition) is made at the beginning of each iteration.
(c) The body of loop ends with a statement that modifies the value of the test (condition) variable.

3.1. for Loop

Syntax :

```
for { variable name } in { list }
do
     execute one for each item in the list until
     the list is finished (And repeat all
     statement between do and done)
done
```

For example write script named **display_num** as follows:

```
for i in 1 2 3 4 5
do
    echo "Welcome $i times"
done
```

Run the above script as follows:
chmod +x display_num
sh display_num

The for loop first creates i variable and assignes a number to i from the list of numbers [from 1 to 5]. The shell executes echo statement for each assignment of i. (This is usually know as iteration). This process will continue until all the items in the list are done. So it will repeat 5 echo statements.
You can use following syntax as well:

```
Syntax:
    for (( expr1; expr2; expr3 ))
    do
         repeat all statements between do and
         done until expr2 is TRUE
    done
```

In the above syntax, before the first iteration, ***expr1*** is evaluated. This is usually used to initialize variables for the loop.
All the statements between do and done is executed repeatedly UNTIL the value of ***expr2*** is TRUE.
After each iteration of the loop, ***expr3*** is evaluated. This is usually used to increment a loop counter.

For example write script named **display_num1** as follows:

```
for (( i=0 ; i<=5; i++ ))
do
   echo "Welcome $i times"
done
```

Run the above script as follows:
chmod +x display_num1
bash display_num1

In the above example, first expression (i=0) is used to set the variable **i**'s value to zero. Second expression is a condition i.e. all statements between do and done executed as long as expression 2 is TRUE. (i.e. continue the execution as long as the value of variable **i** is less than or equal to 5). The last expression, **i++** increments the value of **i** by 1 i.e. it's equivalent to i=i+1 statement.

3.2 Nesting of for Loop

Loop statement can also be nested similar to the if statements. You can nest the for loop. To understand the nesting of for loop, see the following shell script named **display_num_nested**.

```
for (( i=1; i<=5; i++ ))          # Outer for loop
 do
     for (( j=1; j<=5; j++ ))     # Inner for loop
     do
           echo  "$i,$j   "
     done
   echo ""                        #print the new line
 done
```

Run the above script as follows:
chmod +x display_num_nested
bash display_num_nested

Here, for each value of **i,** the inner loop is cycled through 5 times, with the variable **j** taking values from 1 to 5. The inner for loop terminates when the value of **j** exceeds 5, and the outer loop terminates when the value of **i** exceeds 5.

Infinite loop

Infinite for loop can be created with empty expressions, such as

```
for (( ; ; ))
do
   echo "infinite loops [ hit CTRL+C to stop]"
done
```

Conditional exit with break

You can do early exit from a loop with break statement inside the for loop. You can exit from within a FOR or WHILE loop using break. General break statement inside the for loop:

```
for I in 1 2 3 4 5
do
     statements1      #Executed for all values of ''I'', up
                      #to distater condition
     if (disaster-condition)
     then
         break               #Abandon the loop.
     fi
     statements3    #While good and, no disaster-condition.
done
```

Early continuation with continue statement

To resume the next iteration of the enclosing FOR, WHILE or UNTIL loop, use continue statement.

```
for I in 1 2 3 4 5
do
     statements1  #Executed for all values of ''I'', up
                  #to a disaster-condition
     statements2
     if (condition)
     then
           continue   #Go to next iteration of I in the
                      #loop and skip statements3
     fi
     statements3
done
```

3.3. while loop

Syntax:

```
while [ condition ]
do
         command1
         command2
         command3
done
```

The loop is executed as long as given condition is true.

For example write script named **mul_Table** as follows:

```
if [ $# -eq 0 ]
then
       echo "Error - Number missing form command line argument"
      exit 1
fi
n=$1
i=1
while [ $i -le 10 ]
do
       echo "$n * $i = `expr $i \* $n`"
       i=`expr $i + 1`
done
```

Run the above script as follows:

chmod +x mul_Table

sh mul_Table 7

Above loop can be explained as follows:

n=$1	Set the value of command line argument to variable n. (Here it's set to 7)
i=1	Set variable i to 1
while [$i -le 10]	This is our loop condition, here if value of i is less than 10 then, shell execute all statements between do and done
do	Start loop
echo "$n * $i = `expr $i \* $n`"	Print multiplication table as 7 * 1 = 7 7 * 2 = 14 .... 7 * 10 = 70, Here each time value of variable n is multiply by i.
i=`expr $i + 1`	Increment i by 1 and store result to i. (i.e. i=i+1) **Caution:** If you ignore (remove) this statement than our loop become infinite loop because value of variable i always remain less than 10 and program will only output 7 * 1 = 7 ... ... E (infinite times)
done	Loop stops here if i is not less than 10 i.e. condition of loop is not true. Hence loop is terminated

Infinite loops

Infinite for while can be created with empty expressions, such as

```
while :
do
     echo "infinite loops [ hit CTRL+C to stop]"
done
```

Conditional while loop exit with break statement

You can do early exit with the break statement inside the whil loop. You can exit from within a WHILE using break.

In this example, the break statement will skip the while loop when user enters -1, otherwise it will keep adding two numbers:

```
while :
do
   read -p "Enter two numnbers ( - 1 to quit ) : " a b
   if [ $a -eq -1 ]
   then
       break
   fi
   ans=$(( a + b ))
   echo $ans
done
```

Early continuation with the continue statement

To resume the next iteration of the enclosing WHILE loop use the continue statement as follows:

```
while [ condition ]
do
  statements1  #Executed as long as condition is true
               #and/or, up to a disaster condition if any
  statements2
  if (condition)
  then
        continue   #Go to next iteration of the loop and
                   #skip statements3
  fi
  statements3
done
```

4. Exercises with Solution

1. Script to reverse a given positive integer

```
if [ $# -ne 1 ]
then
    echo "Usage: $0   number"
    echo "I will find reverse of given positive integer"
    echo "For eg. $0 123, I will print 321"
    exit 1
fi

n=$1
rev=0; sd=0

while [ $n -gt 0 ]
do
    sd=`expr $n % 10`
    rev=`expr $rev \* 10  + $sd`
    n=`expr $n / 10`
done
echo  "Reverse number is $rev"
```

2. Write a shell script to find the GCD for the 2 given numbers.

```
echo Enter two numbers with space in between
read a b
m=$a
if [ $b -lt $m ]
then
     m=$b
fi
while [ $m -ne 0 ]
do
     x=`expr $a % $m`
     y=`expr $b % $m`
     if [ $x -eq 0 -a $y -eq 0 ]
     then
          echo gcd of $a and $b is $m
          break
     fi
     m=`expr $m - 1`
done
```

3. Script to find the sum of first N numbers.

```
echo "Enter a number: "
read num
i=1
sum=0

while [ $i -le $num ]
do
     sum=`expr $sum + $i`
     i=`expr $i + 1`
done
echo "The sum of first $num numbers is: $sum"
```

4. Write a shell script to take the input number from the user.(Numbers are between 10 and 99) If the number is even find all divisors of the number else find sum of digits.

```
while read n
do
     rem=$(( $n % 2 ))
     if [ $rem -eq 0 ]
     then
          i=2
          sd=0
          echo "Factors are:"
          while [ $i -lt $n ]
          do
               sd=`expr $n % $i`
               if [ $sd -eq 0 ]
                    then echo $i
               fi
               i=`expr $i + 1`
          done
     else
          sum=0
          sd=0
          while [ $n -gt 0 ]
          do
               sd=`expr $n % 10`
               sum=`expr $sum + $sd`
               n=`expr $n / 10`
          done
          echo  "Sum of digit is: $sum"
     fi
```

5. Write a shell program to find the perfect number between 1 and 99.

Perfect numbers, a positive integer that is equal to the sum of its proper divisors. The smallest **perfect number** is 6, which is the sum of 1, 2, and 3.

```
echo Enter a number
read no
i=1
ans=0
while [ $i -le 'expr $no / 2' ]
do
       if [ 'expr $no % $i' -eq 0 ]
       then
          ans='expr $ans + $i'
       fi
       i='expr $i + 1'
done
if [ $no -eq $ans ]
then
       echo $no is perfect
else
       echo $no is NOT perfect
fi
```

6. Write a shell script program to display the following pattern up to N lines (take N from the user). Figure below shows the output when N = 5.

0

1 0

2 1 0

3 2 1 0

4 3 2 1 0

```
a=0
while [ "$a" -lt 10 ]    # this is loop1
do
   b="$a"
   while [ "$b" -ge 0 ]  # this is loop2
   do
      echo -n "$b "
      b=`expr $b - 1`
   done
   echo
   a=`expr $a + 1`
done
```

7. Write a Shell Script to make a menu driven calculator using case statements.

```
sum=0
i="y"
echo " Enter one no."
read n1
echo "Enter second no."
read n2
while [ $i = "y" ]
do
     echo "1.Addition"
     echo "2.Subtraction"
     echo "3.Multiplication"
     echo "4.Division"
     echo "Enter your choice"
     read ch
     case $ch in
          1)sum=`expr $n1 + $n2`
            echo "Sum ="$sum;;
          2)sum=`expr $n1 - $n2`
            echo "Sub = "$sum;;
          3)sum=`expr $n1 \* $n2`
            echo "Mul = "$sum;;
          4)sum=`expr $n1 / $n2`
            echo "Div = "$sum;;
          *)echo "Invalid choice";;
     esac
     echo "Do u want to continue ?"
     read i
     if [ $i != "y" ]
     then
          exit
     fi
done
```

5. Additional Exercises

1. Write a script to find the given number is Armstrong number [also known as narcissistic numbers] of 3 digits. Armstrong numbers are the numbers whose sum of digits to the power of the number of digits is equal to the number itself.
Example: $153 = 1^3 + 5^3 + 3^3$

2. Write a shell script program to display the following pattern up to N lines (take N from the user. Make sure N is always an odd number). Fig. below shows the output when N = 5.

```
                    *
               *    *    *
          *    *    *    *    *
               *    *    *
                    *
```

3. Write a shell script program to read a number from the user and check whether the number is Abundant / Deficient / Perfect number. Make sure the number is between 1 and 10000. Print the number as Abundant / Deficient / Perfect.

Definition of Abundant Number: The number n is abundant if the sum of all its positive divisors except itself is greater than n.

Definition of Perfect Number: The number n is perfect if the sum of all its positive divisors except itself is equal to n.

Definition of Deficient Number: The number n is deficient if the sum of all its positive divisors except itself is less than n.

4.Write a shell script program to read a number N from the user [the program should make sure that the number is in between 1 and 1000] and find all the EVIL numbers between 1 and N.

Definition of EVIL Number:

The Number is an evil number if it has even number of 1's in its binary expansion.

Examples:

Example 1: 40

Binary expansion=101000
Number of 1's=2
40 is an Evil Number

Example 2: 666

Binary expansion=1010011010
Number of 1's= 5
666 is not an Evil Number

5. Write a shell script program to read a number and find the number is magic number or not.

Definition of Magic Number:

The Number is Magic number if the sum of the digits of a number MULTIPLIED BY reverse of the sum is same as the number

Example: Is 1729 a Magic Number?
sum of digits: 19
Reverse of 19 : 91
91*19=1729
1729 is a Magic Number