Manual and Assignment for Operating System Lab (CS693)

Lab #4

Objectives

Essentials of shell programming:

1. **Use of arithmetic expression.**

Assume variable **a** holds 10 and variable **b** holds 20 then −

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator	`expr $a + $b` will give 30
- (Subtraction)	Subtracts right hand operand from left hand operand	`expr $a - $b` will give -10
* (Multiplication)	Multiplies values on either side of the operator	`expr $a \* $b` will give 200
/ (Division)	Divides left hand operand by right hand operand	`expr $b / $a` will give 2
% (Modulus)	Divides left hand operand by right hand operand and returns remainder	`expr $b % $a` will give 0
= (Assignment)	Assigns right operand in left operand	a = $b would assign value of b into a
== (Equality)	Compares two numbers, if both are same then returns true.	[$a == $b] would return false.
!= (Not Equality)	Compares two numbers, if both are different then returns true.	[$a != $b] would return true.

It is very important to understand that all the conditional expressions should be inside square braces with spaces around them, for example **[$a == $b]** is correct whereas, **[$a==$b]** is incorrect.

Example:

```
a=10

b=20

val=`expr $a + $b`
```

```
echo "a + b : $val"

val=`expr $a - $b`
echo "a - b : $val"

val=`expr $a \* $b`
echo "a * b : $val"

val=`expr $b / $a`
echo "b / a : $val"

val=`expr $b % $a`
echo "b % a : $val"

if [ $a == $b ]
then
   echo "a is equal to b"
fi

if [ $a != $b ]
then
   echo "a is not equal to b”

fi
```

Output:

```
a + b : 30

a - b : -10
a * b : 200
b / a : 2
b % a : 0
a is not equal to b
```

2. **Relational Operators.**

The following operators do not work for string values unless their value is numeric.

For example, following operators will work to check a relation between 10 and 20 as well as in between "10" and "20" but not in between "ten" and "twenty".

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.

Example

```
a=10

b=20

if [ $a -eq $b ]
then
   echo "$a -eq $b : a is equal to b"
else
   echo "$a -eq $b: a is not equal to b"
fi

if [ $a -ne $b ]
then
   echo "$a -ne $b: a is not equal to b"
else
   echo "$a -ne $b : a is equal to b"
fi

if [ $a -gt $b ]
then
   echo "$a -gt $b: a is greater than b"
else
   echo "$a -gt $b: a is not greater than b"
fi

if [ $a -lt $b ]
then
   echo "$a -lt $b: a is less than b"
else
   echo "$a -lt $b: a is not less than b"
fi

if [ $a -ge $b ]
then
   echo "$a -ge $b: a is greater or  equal to b"
else
   echo "$a -ge $b: a is not greater or equal to b"
fi

if [ $a -le $b ]
then
   echo "$a -le $b: a is less or  equal to b"
else
   echo "$a -le $b: a is not less or equal to b"
fi
```

Output

```
10 -eq 20: a is not equal to b

10 -ne 20: a is not equal to b
```

```
10 -gt 20: a is not greater than b
10 -lt 20: a is less than b
10 -ge 20: a is not greater or equal to b
10 -le 20: a is less or  equal to b
```

3. Boolean Operators

The following Boolean operators are supported by the Bourne Shell.

Assume variable **a** holds 10 and variable **b** holds 20 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 condition becomes true.	[$a -lt 20 -o $b -gt 100] is true.
-a	This is logical **AND**. If both the operands are true, then the condition becomes true otherwise false.	[$a -lt 20 -a $b -gt 100] is false.

Example

```
a=10

b=20

if [ $a != $b ]
then
   echo "$a != $b : a is not equal to b"
else
   echo "$a != $b: a is equal to b"
fi

if [ $a -lt 100 -a $b -gt 15 ]
then
   echo "$a -lt 100 -a $b -gt 15 : returns true"
else
   echo "$a -lt 100 -a $b -gt 15 : returns false"
fi

if [ $a -lt 100 -o $b -gt 100 ]
then
   echo "$a -lt 100 -o $b -gt 100 : returns true"
else
   echo "$a -lt 100 -o $b -gt 100 : returns false"
fi

if [ $a -lt 5 -o $b -gt 100 ]
then
   echo "$a -lt 100 -o $b -gt 100 : returns true"
else
   echo "$a -lt 100 -o $b -gt 100 : returns false"
fi
```

Output

```
10 != 20 : a is not equal to b

10 -lt 100 -a 20 -gt 15 : returns true
10 -lt 100 -o 20 -gt 100 : returns true
10 -lt 5 -o 20 -gt 100 : returns false
```

4. **Use of for loop.**

The **for** loop operates on lists of items. It repeats a set of commands for every item in a list.

Syntax:

```
for var in 0 1 2 3 4 5

do
    echo $var
done
```

Output:

```
0

1

2

3

4

5
```

The above code can be written like this-

```
for var in `seq 0 5`

do
      echo $var
done
```

The above code can also be written like this-

```
j=”0 1 2 3 4 5”

for var in $j

do
      echo $var
done
```

5. **Use of while loop.**

The **while** loop enables you to execute a set of commands repeatedly until some condition occurs. Each while loop consists of a set of commands and a condition. The general syntax as follows for bash while loop:

Syntax

```
while [ condition ]

do
  command1
  command2
  commandN
done
```

Example

```
a=0

while [ $a -lt 5 ]
do
    echo $a
   a=`expr $a + 1`
done
```

Output:

0

1

2

3

4

5

6. **Use of switch case.**

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

Syntax

```
        case  $variable-name  in

          pattern1)
              command1
              ...
              ....
              commandN
              ;;
          pattern2)
```

```
        command1
        ...
        ....
        commandN
        ;;
    patternN)
        command1
        ...
        ....
        commandN
       ;;
    *)
esac
```

Or

```
case  $variable-name  in

  pattern1|pattern2|pattern3)
      command1
      ...
      ....
      commandN
     ;;
  pattern4|pattern5|pattern6)
      command1
      ...
      ....
      commandN
     ;;
  pattern7|pattern8|patternN)
      command1
      ...
      ....
      commandN
     ;;
  *)
 esac
```

Example

```
FRUIT="kiwi"


case "$FRUIT" in
   "apple") echo "Apple pie is quite tasty."
   ;;
   "banana") echo "I like banana nut bread."
   ;;
   "kiwi") echo "New Zealand is famous for kiwi."
   ;;
esac
```

Output

```
New Zealand is famous for kiwi.
```

Example

echo "Please talk to me ..."

```
while :
do
  read INPUT_STRING
  case $INPUT_STRING in
        hello)
                echo "Hello yourself!"
                ;;
        bye)
                echo "See you again!"
                break
                ;;
        *)
                echo "Sorry, I don't understand"
                ;;
  esac
done
echo
echo "That's all folks!"
```

Execution

```
Please talk to me ...

hello
Hello yourself!
What do you think of politics?
Sorry, I don't understand
bye
See you again!

That's all folks!
```

Assignment

1. Write a shell program to calculate the factorial of a number.

2. Write a shell menu driven program to do the following:

   a. Display the current working directory.

   b. Check whether an input number is even or odd.

   c. Display the number of counts of all the files in the directory.

   d. Print the long listing of all the files.

3. Write a shell program to display all the prime numbers between 1 to 100 using while loop.

4. Write a menu program to find out whether a given letter is vowel or not.

5. Write a shell script which will generate the output as follows:

```
*

* *

* * *

* * * *
```