Decision-Making in Shell Scripts

Relational operators allow us to work with numerical values and with strings that are numbers in various useful ways. We can use them to compare numbers, thus helping us in statements that decision-making. The value that we receive after evaluation is 1 if the expression is true and 0 if false. Comparison expressions match lines where if the condition is true, then a particular action is performed.

Relational Operators in shell script

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

Boolean operators are used to perform various logical operations like AND, OR, and NOT. They are used as conjunctions to combine or exclude expressions for the purpose of decision-making.

Boolean Operators in shell script

Operator	Description	Usage
!	This is logical negation. This inverts a true condition into false and vice versa.	[! false]
-o	This is logical OR. If one of the operands is true, then the condition becomes true.	[$a -lt 10 -o $b -gt 50]
-a	This is logical AND. If both the operands are true, then the condition becomes true; otherwise false.	[$a -lt 40 -a $b -gt 90]

String operators allow us to manipulate the values of variables in various useful ways without having to write complex programs or use some external UNIX utilities.

String Operators in shell script

Operator	Description	Usage

=	Checks if the values of two operands are equal or not; if yes, then the condition becomes true.	[$a = $b]
!=	Checks if the values of two operands are equal or not; if values are not equal, then the condition becomes true.	[$a != $b]
-z	Checks if the given string operand size is zero; if it is of zero length, then it returns true.	[-z $a]
-n	Checks if the given string operand size is non-zero; if it is of non-zero length, then it returns true.	[-n $a]
str	Checks if str is not the empty string; if it is empty, then it returns false.	[$a]

File test operators help us test many features of a file, such as whether it is readable or not, whether the user has write permissions on a file, the file size, whether the file exists or not, etc.
File test operators in shell script

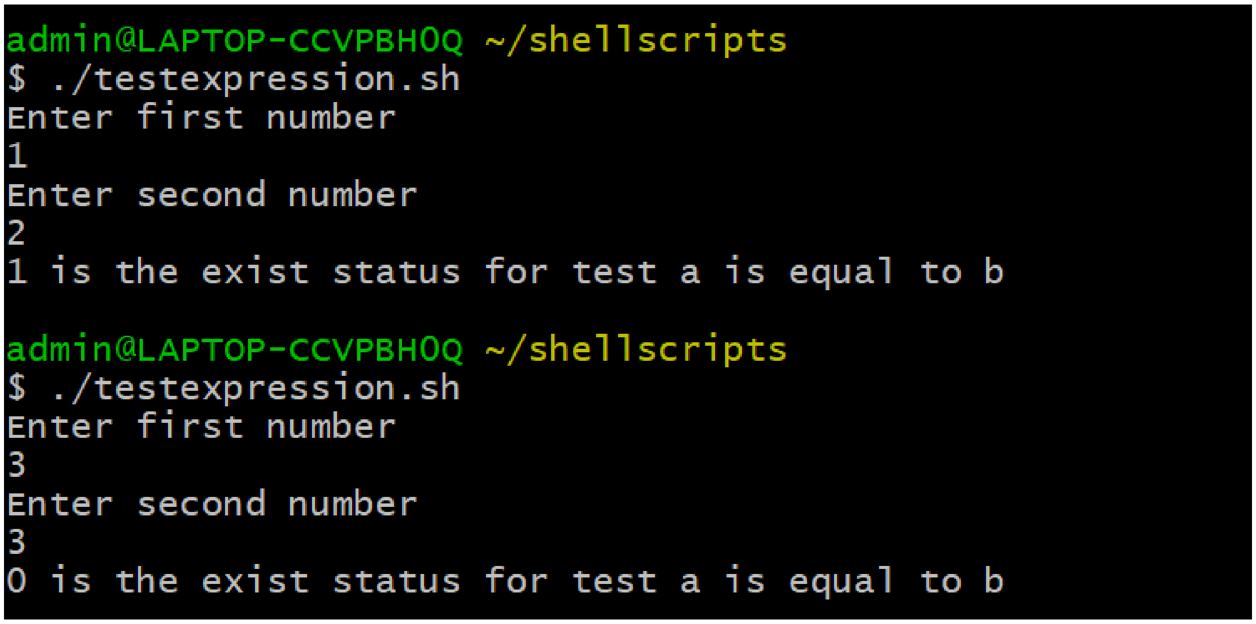
Operator	**Description**	**Usage**
-d	Checks if the file is a directory; if yes, then the condition becomes true.	[-d $file]
-f	Checks if the file is an ordinary file; if yes, then the condition becomes true.	[-f $file]
-r	Checks if the file is readable; if yes, then the condition becomes true.	[-r $file]
-w	Checks if the file is writable; if yes, then the condition becomes true.	[-w $file]
-x	Checks if the file is executable; if yes, then the condition becomes true.	[-x $file]
-e	Checks if file exists; is true even if file is a directory but exists.	[-e $file]

Test statement is used to handle the true or false value returned by expressions. It uses certain operators to evaluate the condition on its right. It returns either a true or false exit status. If the given expression is true, test exits with a status of zero; otherwise, it exits with a status of 1. This can be used for making decisions.
Example: Numeric comparison using test

```
#!/bin/bash
# usage of test expression on numeric values
echo Enter first number
read a
echo Enter second number
read b
# using test command to check if numbers are equal
```

test $a -eq $b

echo "$? is the exist status for test a is equal to b"



```
admin@LAPTOP-CCVPBHOQ ~/shellscripts
$ ./testexpression.sh
Enter first number
1
Enter second number
2
1 is the exist status for test a is equal to b

admin@LAPTOP-CCVPBHOQ ~/shellscripts
$ ./testexpression.sh
Enter first number
3
Enter second number
3
0 is the exist status for test a is equal to b
```

Example: String comparison using test

#!/bin/bash

# Example to check if two strings are equal or not

echo Enter the first string

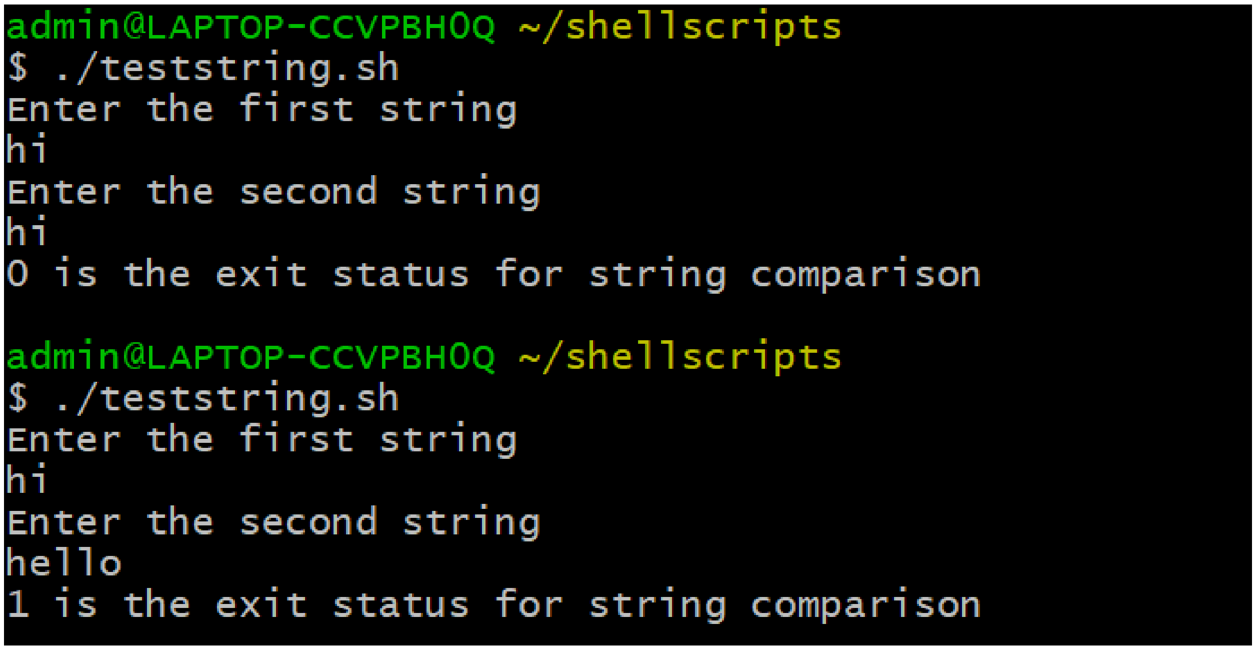
read a

echo Enter the second string

read b

test $a = $b

echo $? is the exit status for string comparison



```
admin@LAPTOP-CCVPBHOQ ~/shellscripts
$ ./teststring.sh
Enter the first string
hi
Enter the second string
hi
0 is the exit status for string comparison

admin@LAPTOP-CCVPBHOQ ~/shellscripts
$ ./teststring.sh
Enter the first string
hi
Enter the second string
hello
1 is the exit status for string comparison
```

Decision-making in shell script: One of the most important parts of any programming language is the if-else statements. An if-else statement allows us to execute the conditional statements. We can use if-else in shell scripts when we want to check a particular condition, and then decide to execute a particular set of statements based on the result of this condition.
Let us have a look at the basic form of if statement.

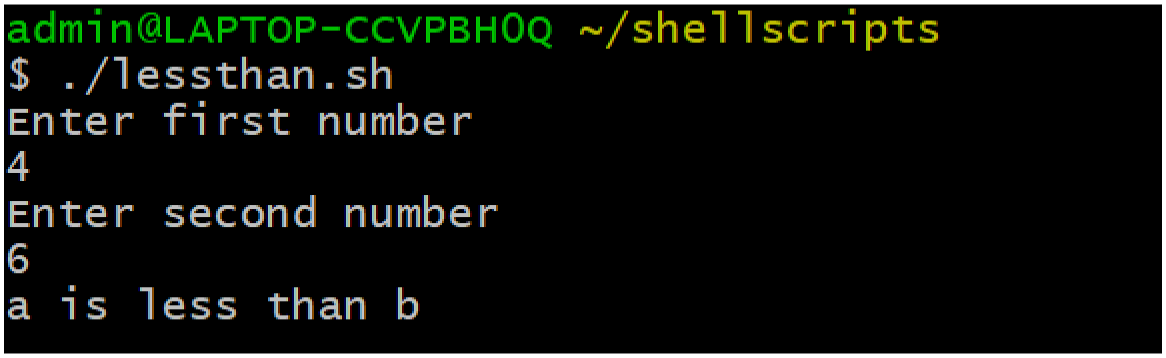
Basic Form

```
if [condition]
then
    statement1
fi
```

In the simple form, if the condition succeeds, the statements within the if block are executed; otherwise, this code block will not be executed.

Example: Shell script to check if the first number is less than the second number

```
#!/bin/bash
echo "Enter first number"
read a
echo "Enter second number"
read b
#check if variable a is less than b
if [ $a -lt $b ]
then
        echo "a is less than b"
fi
```

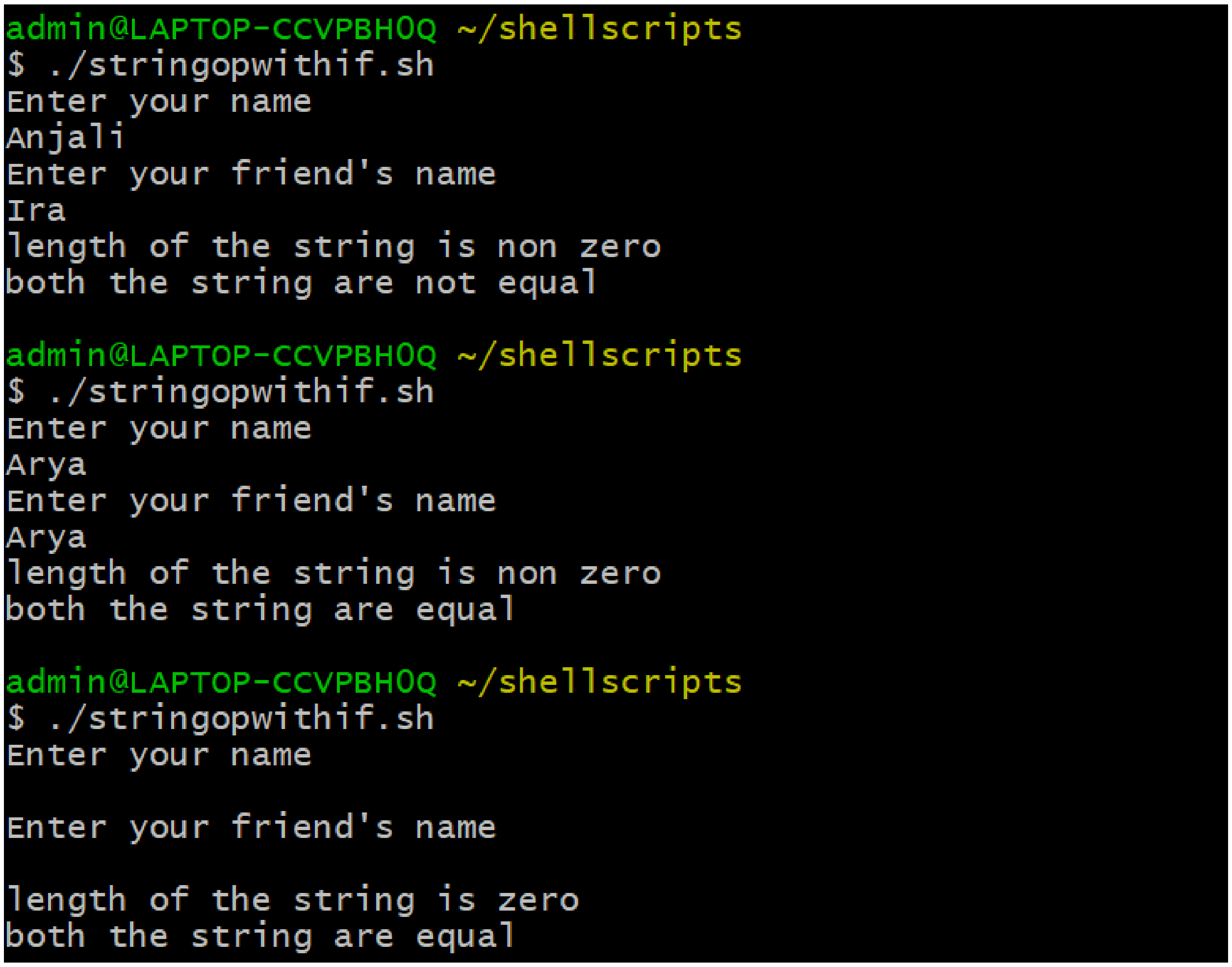


```
admin@LAPTOP-CCVPBHOQ ~/shellscripts
$ ./lessthan.sh
Enter first number
4
Enter second number
6
a is less than b
```

Example: Shell script to show basic string operations

```
#!/bin/bash
echo "Enter your name"
read name
echo "Enter your friend's name"
read othername
# to check if the string length is non-zero
if [[ -n ${name} ]]
then
      echo "length of the string is non-zero"
fi
# to check if the string length is zero
if [[ -z ${name} ]]
then
```

```
        echo "length of the string is zero"
fi
# check two if the two string are equal
if [[ ${name} = ${othername} ]]
then
        echo "both the string are equal"
fi
# check if the two string are not equal
if [[ ${name} != ${othername} ]]
then
        echo "both the string are not equal"
fi
```

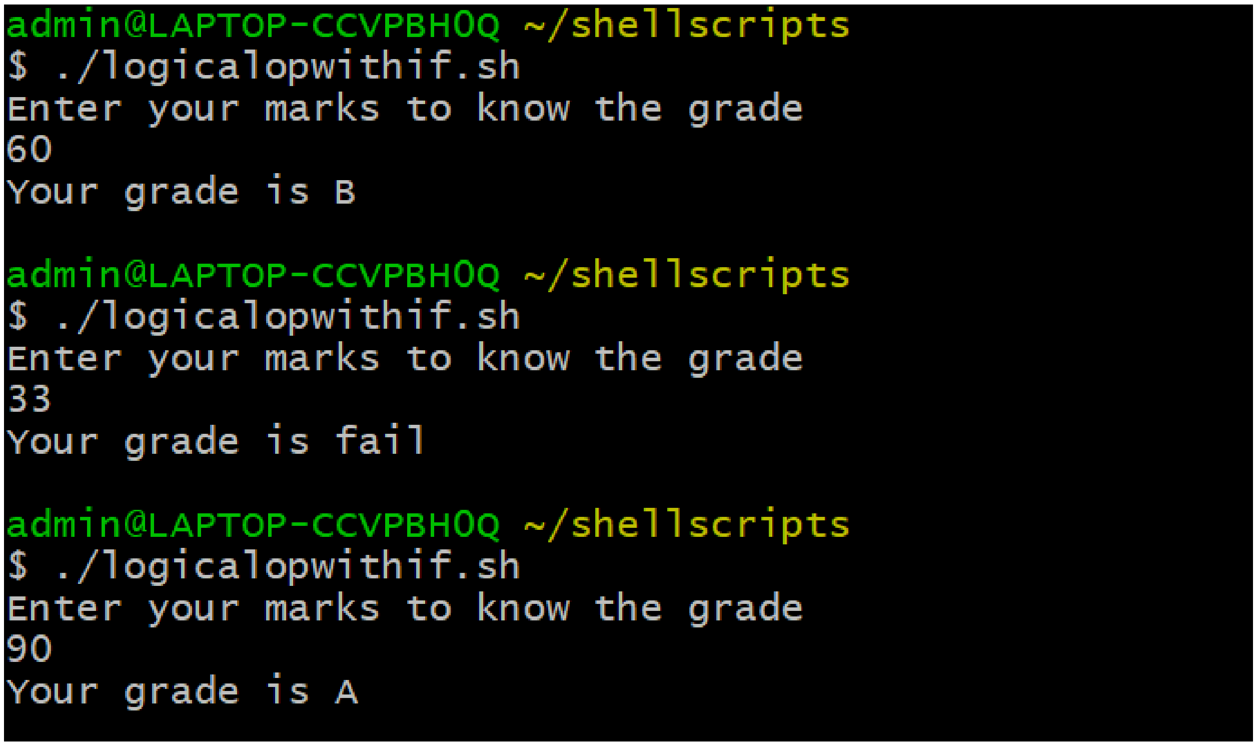


```
admin@LAPTOP-CCVPBHOQ ~/shellscripts
$ ./stringopwithif.sh
Enter your name
Anjali
Enter your friend's name
Ira
length of the string is non zero
both the string are not equal

admin@LAPTOP-CCVPBHOQ ~/shellscripts
$ ./stringopwithif.sh
Enter your name
Arya
Enter your friend's name
Arya
length of the string is non zero
both the string are equal

admin@LAPTOP-CCVPBHOQ ~/shellscripts
$ ./stringopwithif.sh
Enter your name

Enter your friend's name

length of the string is zero
both the string are equal
```

Example: Shell script to demonstrate the usage of logical operators

```
#!/bin/bash
echo "Enter your marks to know the grade"
read a
#check if variable a is less than or equal to 100 AND greater than and equal to 80
#logical operator –a(AND) is used here
if [ $a -le 100 -a $a -ge 80 ]
then
```

```
        echo "Your grade is A"
fi
#check if variable a is less than 80 AND greater than and equal to 60
#logical operator –a(AND) is used here
if [ $a -lt 80 -a $a -ge 60 ]
then
         echo "Your grade is B"
fi
#check if variable a is less than 60 AND greater than and equal to 40
#logical operator –a(AND) is used here
if [ $a -lt 60 -a $a -ge 40 ]
then
        echo "Your grade is C"
fi
#check if variable is equal to 0 OR variable a is less than 40
#logical operator –o(OR) is used here
if [ $a -eq 0 -o $a -lt 40 ]
then
        echo "Your grade is fail"
fi
```

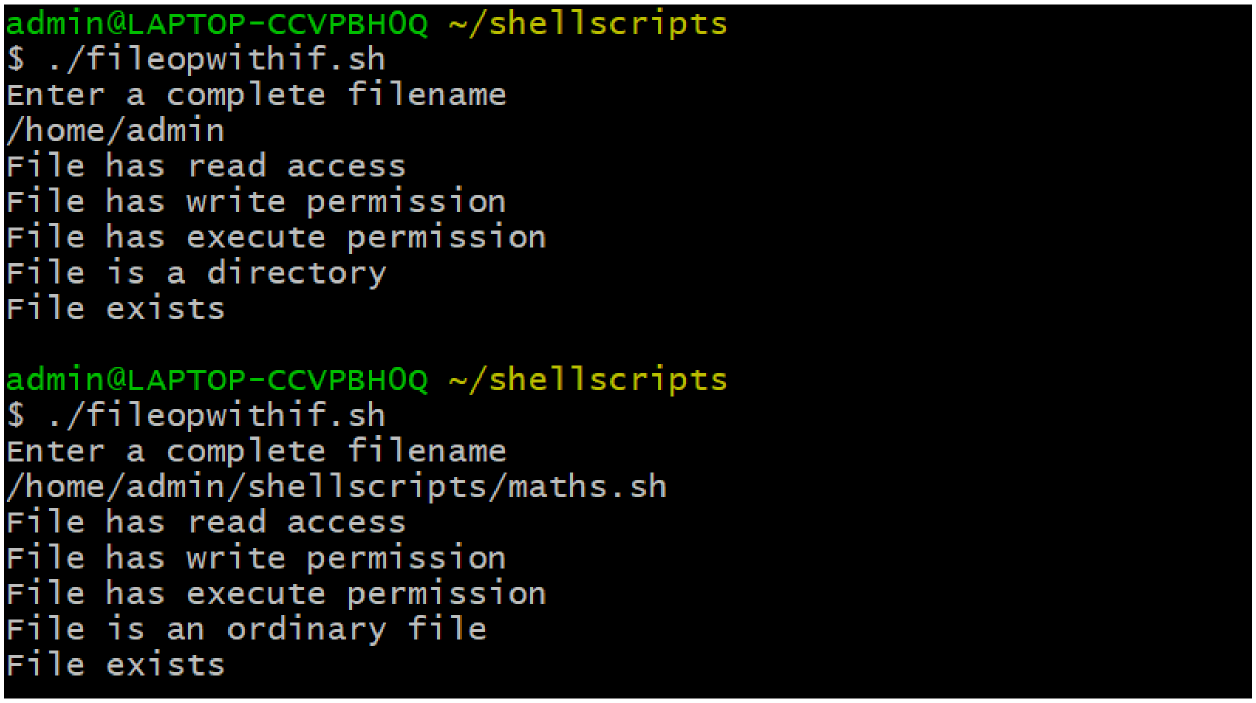


```
admin@LAPTOP-CCVPBH0Q ~/shellscripts
$ ./logicalopwithif.sh
Enter your marks to know the grade
60
Your grade is B

admin@LAPTOP-CCVPBH0Q ~/shellscripts
$ ./logicalopwithif.sh
Enter your marks to know the grade
33
Your grade is fail

admin@LAPTOP-CCVPBH0Q ~/shellscripts
$ ./logicalopwithif.sh
Enter your marks to know the grade
90
Your grade is A
```

Example: Shell script to show usage of file test operators

```
#!/bin/bash
echo "Enter a complete filename"
read file
#check read permission of file
if [ -r $file ]
```

```
then
          echo "File has read access"
fi
#check write permission of file
if [ -w $file ]
then
         echo "File has write permission"
fi
#check execute permission of file
if [ -x $file ]
then
          echo "File has execute permission"
fi
#check if file is a normal file
if [ -f $file ]
then
         echo "File is an ordinary file"
fi
#check if file is a directory
if [ -d $file ]
then
        echo "File is a directory"
fi
#check if the file exists
if [ -e $file ]
then
        echo "File exists"
fi
```



```
admin@LAPTOP-CCVPBHOQ ~/shellscripts
$ ./fileopwithif.sh
Enter a complete filename
/home/admin
File has read access
File has write permission
File has execute permission
File is a directory
File exists

admin@LAPTOP-CCVPBHOQ ~/shellscripts
$ ./fileopwithif.sh
Enter a complete filename
/home/admin/shellscripts/maths.sh
File has read access
File has write permission
File has execute permission
File is an ordinary file
File exists
```

Looping Constructs

Loops are used to repeatedly execute the statements following the test expression if a condition is true. Loops are often used to iterate through the fields within a record.
for loop: It requires three expressions within the parentheses: the initialization expression, the test expression, and the expression to update the variables within the test expression.
Let us look at different ways in which we can use for loop

1

```
foreach var (wordlist)
            commands
end
```

2

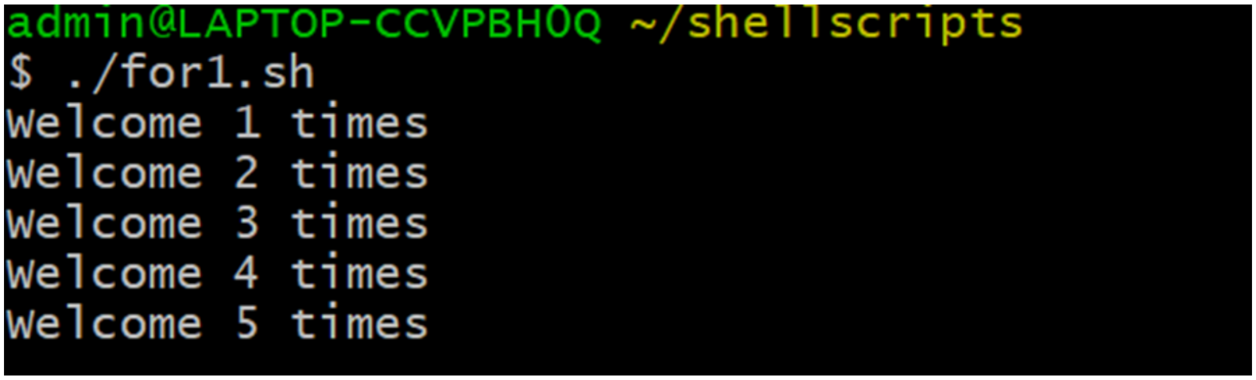
```
for var in wordlist
do
            commands
done
```

3

```
for ( x = 3; x <= n; x++ )
            commands
```

Example: Let us look at a very simple implementation of for loop

```
#!/bin/bash
#simple for loop iteration
for i in 1 2 3 4 5
do
echo "Welcome $i times"
done
```

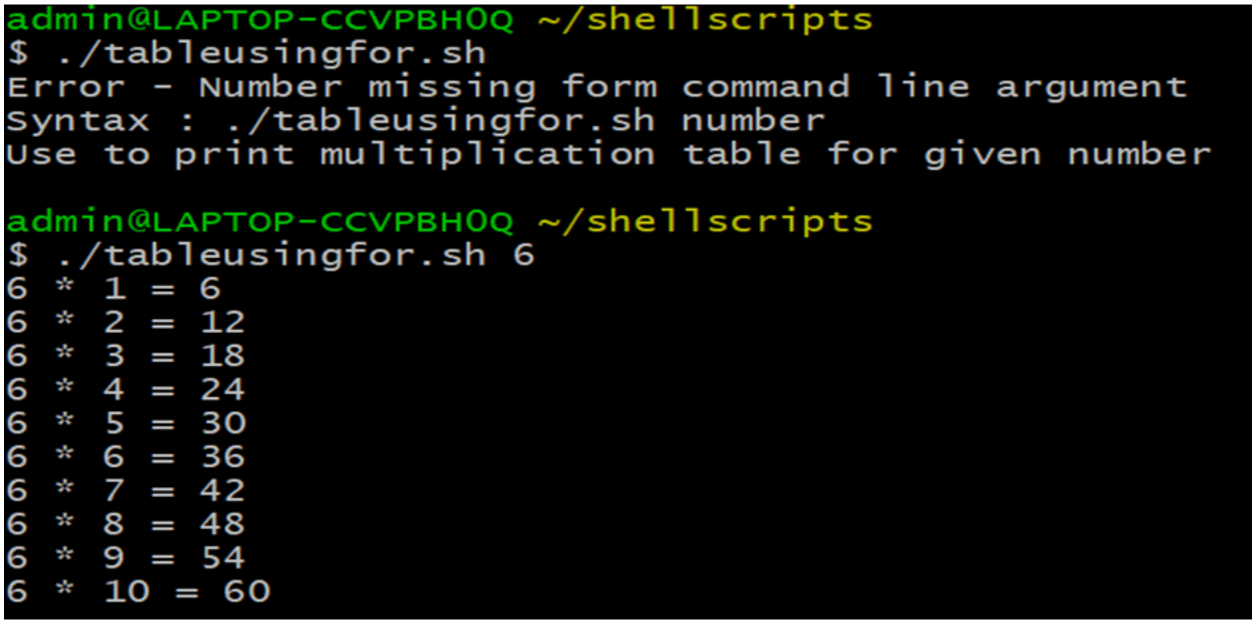


```
admin@LAPTOP-CCVPBHOQ ~/shellscripts
$ ./for1.sh
Welcome 1 times
Welcome 2 times
Welcome 3 times
Welcome 4 times
Welcome 5 times
```

Example: Let us print a table of a given number using for loop

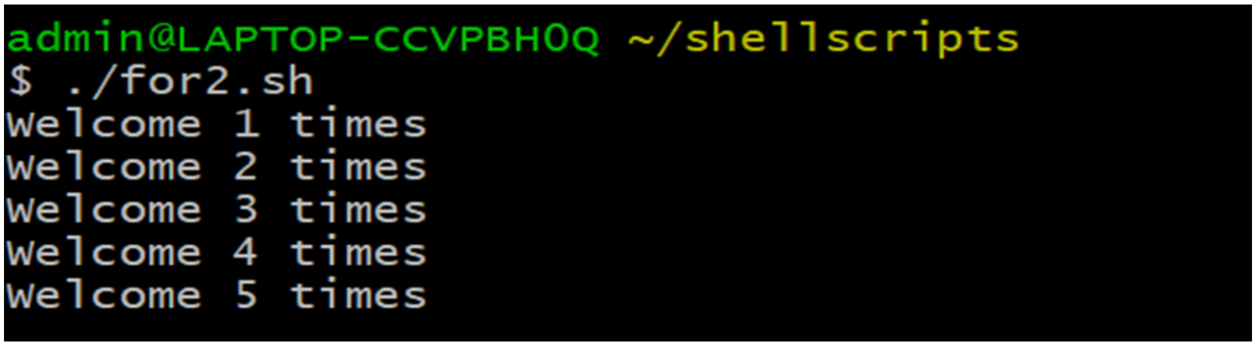
```
#!/bin/bash
#Script to print table of a number
#check if argument is provided if not print number missing and then exit
if [ $# -eq 0 ]
then
echo "Error - Number missing form command line argument"
echo "Syntax : $0 number"
echo "Use to print multiplication table for given number"
exit 1
fi
```

#print the table for the given number
n=$1
***for** i **in** 1 2 3 4 5 6 7 8 9 10*
do
*echo "$n * $i = `expr $i \* $n`"*
done



```
admin@LAPTOP-CCVPBHOQ ~/shellscripts
$ ./tableusingfor.sh
Error - Number missing form command line argument
Syntax : ./tableusingfor.sh number
Use to print multiplication table for given number

admin@LAPTOP-CCVPBHOQ ~/shellscripts
$ ./tableusingfor.sh 6
6 * 1 = 6
6 * 2 = 12
6 * 3 = 18
6 * 4 = 24
6 * 5 = 30
6 * 6 = 36
6 * 7 = 42
6 * 8 = 48
6 * 9 = 54
6 * 10 = 60
```

Example: Script using for loop
#!/bin/bash/
# set counter 'c' to 1 and check condition if c is less than or equal to 5
# the loop will keep on repeating c becomes greater than 5
***for**(c=1; c<=5; c++)*
do
  echo "Welcome $c times"
done



```
admin@LAPTOP-CCVPBHOQ ~/shellscripts
$ ./for2.sh
Welcome 1 times
Welcome 2 times
Welcome 3 times
Welcome 4 times
Welcome 5 times
```

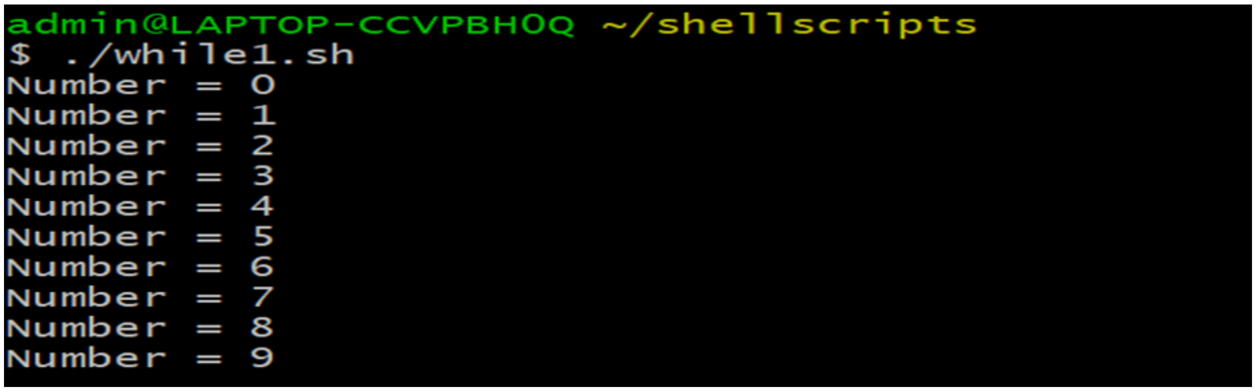
While loop: The first step in using a while loop is to set a variable to an initial value. The value is then tested in the while expression. If the expression evaluates to true (nonzero), the body of the loop is entered and the statements within that body are executed.
Let us look at different ways in which we can use while loop

1	2

***while** (expression)*
commands
end
	***while** command*
	do
	command
	done

Example: Let us look at a very simple example of while loop

```
#!/bin/bash
#simple usage of while loop
number=0
while [ $number -lt 10 ]
do
    echo "Number = $number"
    number=`expr $number + 1`
done
```

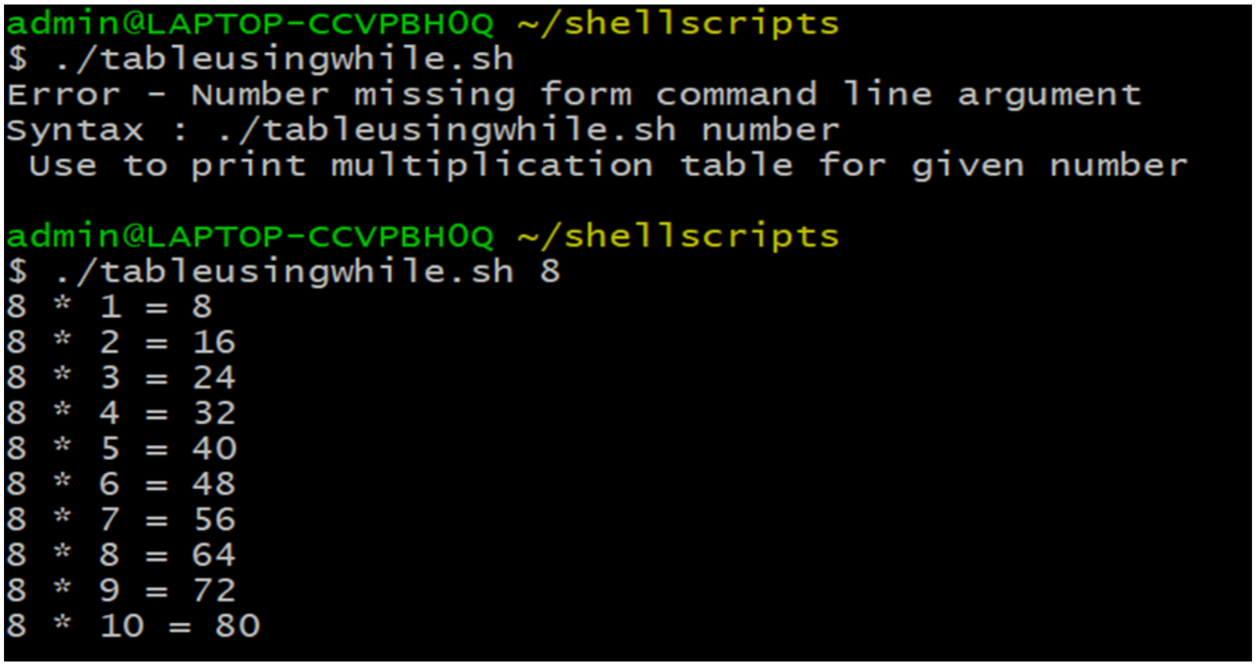


```
admin@LAPTOP-CCVPBHOQ ~/shellscripts
$ ./while1.sh
Number = 0
Number = 1
Number = 2
Number = 3
Number = 4
Number = 5
Number = 6
Number = 7
Number = 8
Number = 9
```

Example: This shell script prints the multiplication table for a given number using while loop

```
#!/bin/sh
#Script to print table for a given number
#check if input number is provided or not. If not then print a message and exit from the program
if [ $# -eq 0 ]
then
  echo "Error - Number missing form command line argument"
  echo "Syntax : $0 number"
  echo " Use to print multiplication table for given number"
exit 1
fi
#print the table
n=$1
i=1
while [ $i -le 10 ]
do
    echo "$n * $i = `expr $i \* $n`"
```

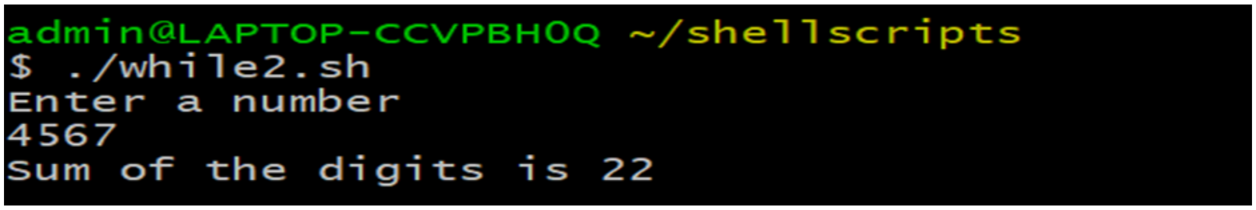
    i=`expr $i + 1`
done



```
admin@LAPTOP-CCVPBH0Q ~/shellscripts
$ ./tableusingwhile.sh
Error - Number missing form command line argument
Syntax : ./tableusingwhile.sh number
 Use to print multiplication table for given number

admin@LAPTOP-CCVPBH0Q ~/shellscripts
$ ./tableusingwhile.sh 8
8 * 1 = 8
8 * 2 = 16
8 * 3 = 24
8 * 4 = 32
8 * 5 = 40
8 * 6 = 48
8 * 7 = 56
8 * 8 = 64
8 * 9 = 72
8 * 10 = 80
```

Example: Script to print the sum of the digits of a given number using while loop

```
#!/bin/bash
#Script to print the sum of the digits of a number
echo "Enter a number"
read num
sum=0
while [ $num -gt 0 ]
do
  mod=$((num % 10))    #It will split each digits
  sum=$((sum + mod))   #Add each digit to sum
  num=$((num / 10))    #divide num by 10.
done
echo Sum of the digits is $sum
```



```
admin@LAPTOP-CCVPBH0Q ~/shellscripts
$ ./while2.sh
Enter a number
4567
Sum of the digits is 22
```

Until loop: It is used to iterate over a block of commands until the required condition is false. First, the condition is checked if the condition is false, then executes the statements and keeps on repeating this. But once the condition becomes true, the program control moves to the next command in the script.

***until** [condition];*
do

```
   block-of-statements
done
```

Example: Script to print the value of variable i until it becomes equal to 1. We start with value i=10 and then keep on decreasing it by 1.

```
#!/bin/bash
#script to print the value of i until it becomes 1
echo "until loop"
i=10
until [ $i == 1 ]
do
  echo "$i is not equal to 1";
  i=$((i-1))
done
echo "i value is $i"
echo "loop terminated"
```

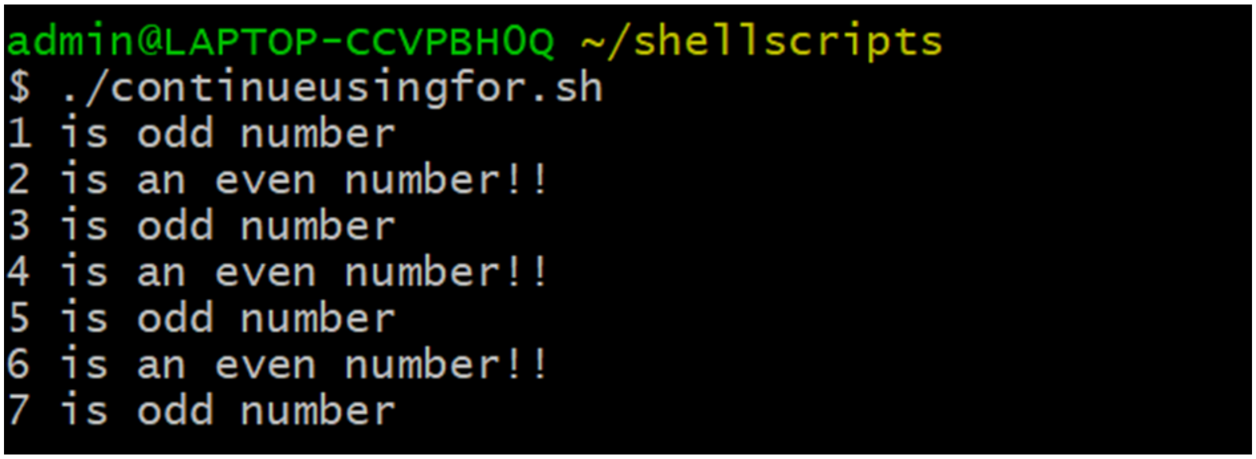
Break and Continue Statements:

The break statement is used to terminate the execution of the entire loop after completing the execution of all of the lines of code up to the break statement. It then steps down to the code following the end of the loop.

The continue statement is similar to the break command, except that it causes the current iteration of the loop to exit, rather than the entire loop. This statement is useful when an error has occurred but you want to try to execute the next iteration of the loop.

Example: Script to identify even and odd numbers using for loop and continue statement

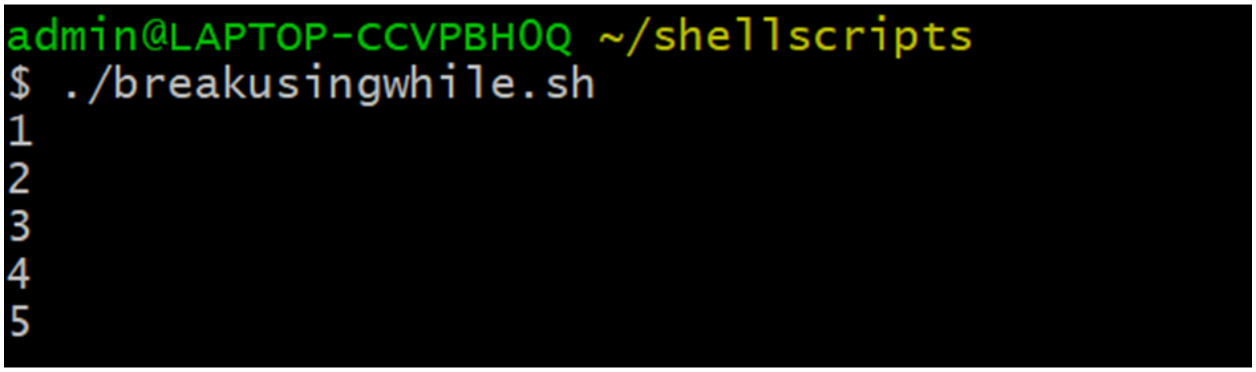
```
NUMS="1 2 3 4 5 6 7"
for Num in $NUMS
do
Q=`expr $Num % 2`
if [ $Q -eq 0 ]
 then
   echo "$Num is an even number!!"
 continue
fi
echo "$Num is odd number"
done
```



```
admin@LAPTOP-CCVPBHOQ ~/shellscripts
$ ./continueusingfor.sh
1 is odd number
2 is an even number!!
3 is odd number
4 is an even number!!
5 is odd number
6 is an even number!!
7 is odd number
```

Example: We are using for loop to iterate over values from 1 to 10. Inside the loop, we are using the echo command to print the value of variable i. We also have an if statement that checks if the value of i is equal to 5. If it is, then we use break statement to exit the loop. As a result, when the program is run, it will print values 1 through 5 and then terminate the loop.

```
#!/bin/bash/
#usage of break statement
i=1
while [ $i -le 10 ]
do
  echo $i
  if [ $i -eq 5 ]
  then
     break
  fi
# (( ... )) construct permits arithmetic expansion and evaluation.
  i=$((i+1))
done
```



```
admin@LAPTOP-CCVPBHOQ ~/shellscripts
$ ./breakusingwhile.sh
1
2
3
4
5
```

File Handling

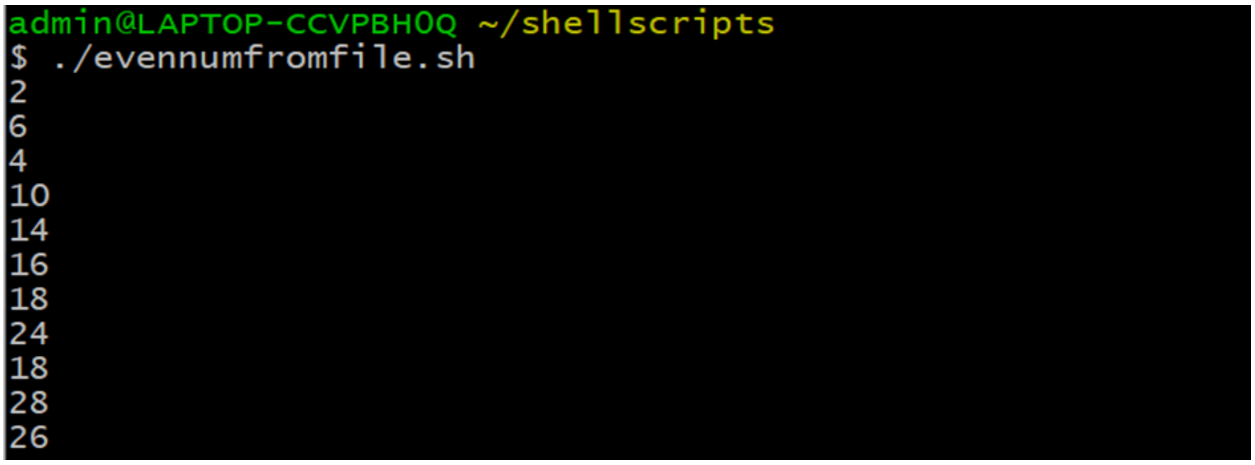
A lot of time, we use shell scripting to interact with the files. Shell scripting offers some operators and commands to check different properties associated with the file. Now, let us focus on performing different functions on file using shell scripts.
If you need to read each line from a file and perform some action with it, then you can use “while” loop. In the syntax given below, replace “[commands]” section with whatever actions you want to perform on the input file and replace “[INPUT_FILE]” with the appropriate input file name.

Syntax:

```
while read line
do
        [COMMAND]
done < [INPUT_FILE]
```

Example: Create a text file called integernos.txt and store some random numbers in it (one number in one line). Write a shell script program to display all the even numbers from this file.

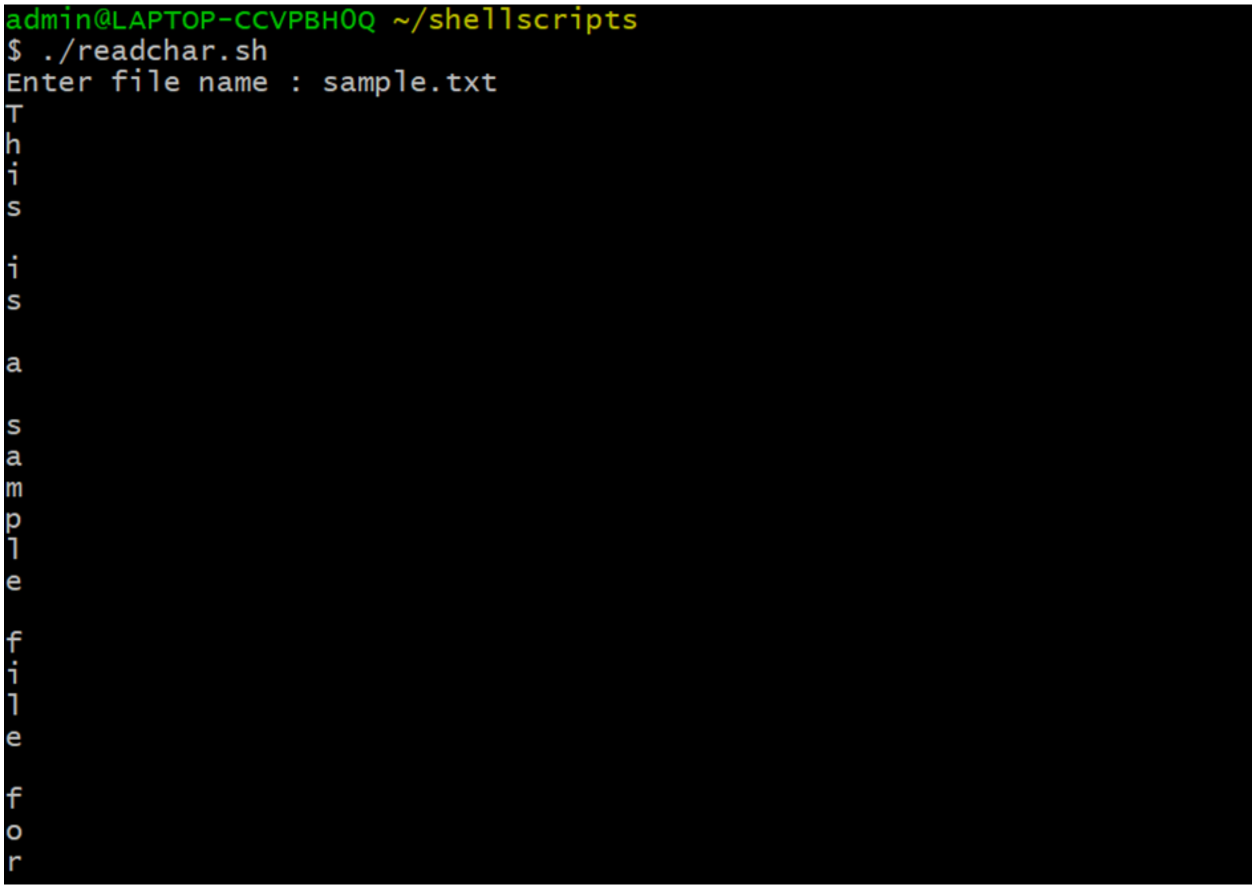
```
#!/bin/bash/
while read line
do
     x=`expr $line % 2` #test for even number
        if [$x -eq 0]
        then
                echo $line #display even numbers
        fi
done < integernos.txt #input file
```



```
admin@LAPTOP-CCVPBH0Q ~/shellscripts
$ ./evennumfromfile.sh
2
6
4
10
14
16
18
24
18
28
26
```

Example: Script to read file character by character.

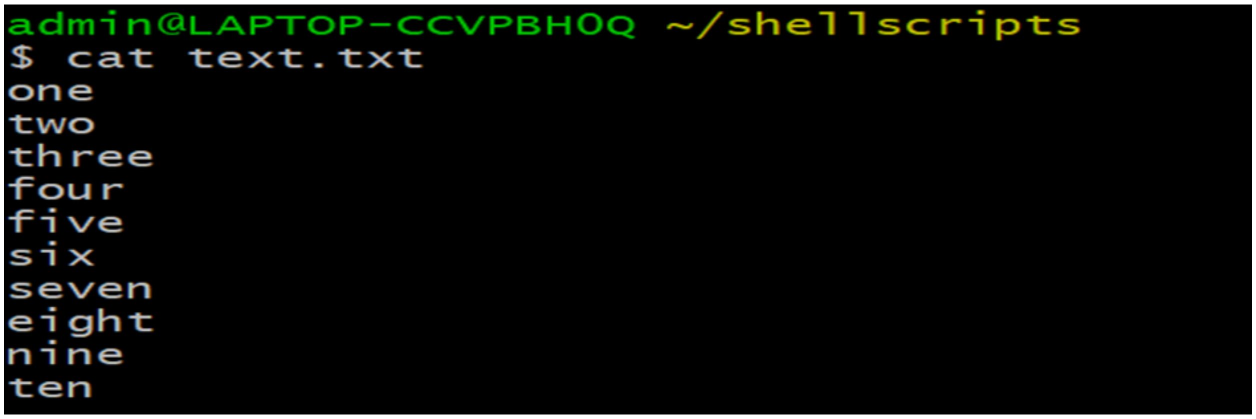
```
#!/bin/bash
read -p "Enter file name : " filename
while read -n1 character
do
echo $character
done < $filename
```



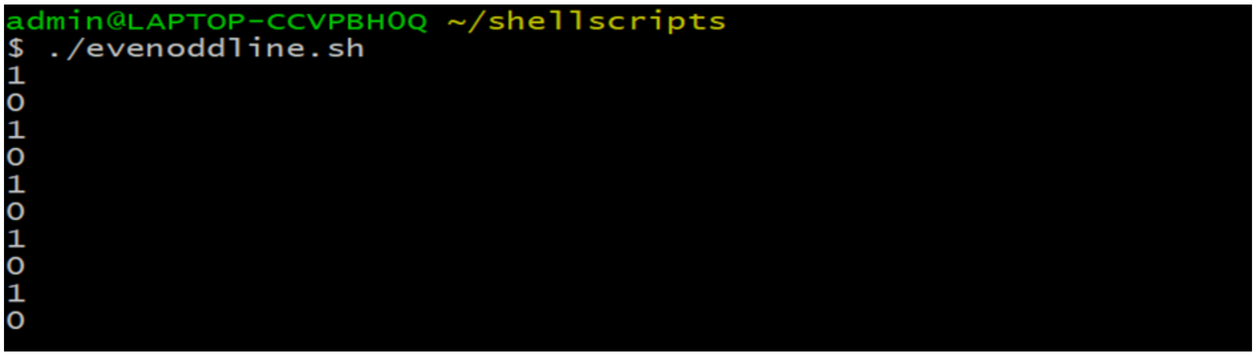
```
admin@LAPTOP-CCVPBH0Q ~/shellscripts
$ ./readchar.sh
Enter file name : sample.txt
T
h
i
s

i
s

a

s
a
m
p
l
e

f
i
l
e

f
o
r
```

Example: Copy the odd lines of the file named data.txt to a file named odd.txt and copy the even lines of this file to a file named even.txt

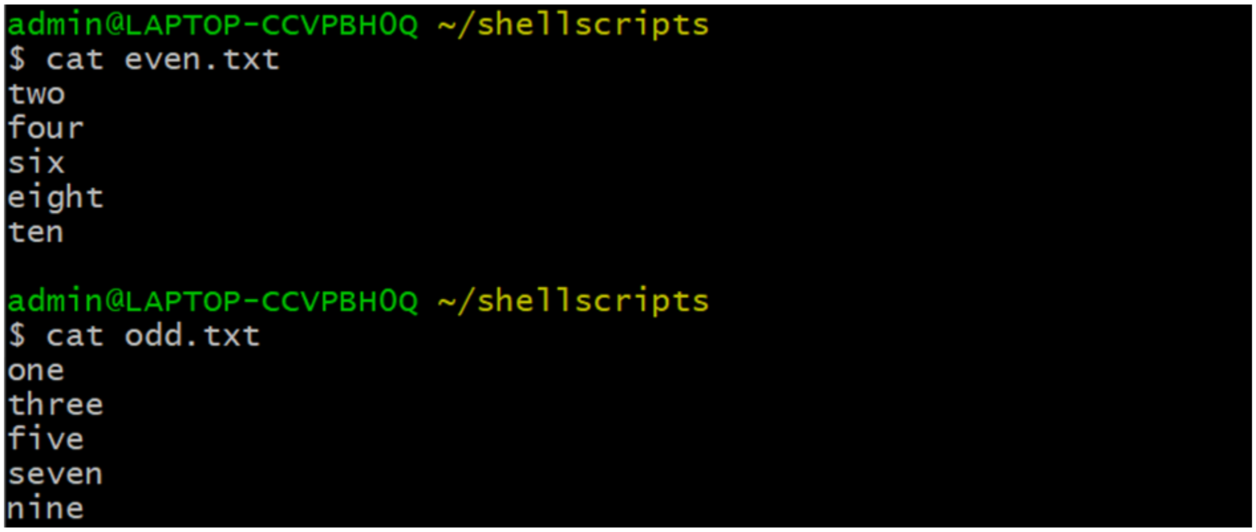
```
#!/bin/bash/
i=1
while read line
do
        x=`expr $i % 2` #test for even number
        echo $x
        if [ $x -eq 0 ]
        then
                echo $line >> even.txt
        else
                echo $line >> odd.txt
        fi
        i=`expr $i + 1`
done < text.txt #input file
```



```
admin@LAPTOP-CCVPBHOQ ~/shellscripts
$ cat text.txt
one
two
three
four
five
six
seven
eight
nine
ten
```



```
admin@LAPTOP-CCVPBHOQ ~/shellscripts
$ ./evenoddline.sh
1
0
1
0
1
0
1
0
1
0
```



```
admin@LAPTOP-CCVPBHOQ ~/shellscripts
$ cat even.txt
two
four
six
eight
ten

admin@LAPTOP-CCVPBHOQ ~/shellscripts
$ cat odd.txt
one
three
five
seven
nine
```