a) Write a shell script that takes a command –line argument and reports on whether it is directory, a file, or something else.

b) Write a shell script that accepts one or more file name as arguments and converts all of them to uppercase, provided they exist in the current directory.

c) Write a shell script that determines the period for which a specified user is working on.

a. Write a shell script that takes a command –line argument and reports on whether it is directory, a file, or something else.

```
echo " enter file"
read str
if test -f $str
then echo "file exists n it is an ordinary file"
elif test -d $str
then echo "directory file"
else
echo "not exists"
fi
```

(or)

```
$ vi filetype.sh
   echo  "Enter the file name: "
    read file
   if [  -f  $file ]
   then
echo $file "---> It is a ORDINARY FILE."
  elif [  -d  $file  ]
   then
      echo  $file "---> It is a DIRCTORY."
else
      echo  $file "---> It is something else."
 fi
```

If condition:

The **if...elif...fi** statement is the one level advance form of control statement that allows Shell to make correct decision out of several conditions.

Syntax:

```
if [ expression 1 ]
then
   Statement(s) to be executed if expression 1 is true
elif [ expression 2 ]
then
   Statement(s) to be executed if expression 2 is true
elif [ expression 3 ]
then
   Statement(s) to be executed if expression 3 is true
else
   Statement(s) to be executed if no expression is true
fi
```

There is nothing special about this code. It is just a series of *if* statements, where each *if* is part of the*else* clause of the previous statement. Here statement(s) are executed based on the true condition, if non of the condition is true then *else* block is executed.

Test:

Checks file types and compares values.

Syntax

test *EXPRESSION* [*EXPRESSION*]

Description

test is used as part of the conditional execution of shell commands.

test exits with the status determined by EXPRESSION. Placing the EXPRESSION between square brackets (**[** and **]**) is the same as testing the EXPRESSION with **test**. To see the exit status at the command prompt, echo the value "**$?**" A value of 0 means the expression evaluated as true, and a value of 1 means the expression evaluated as false.

Expressions

Expressions take the following forms:

(*EXPRESSION* **)**	*EXPRESSION* is true
! *EXPRESSION*	*EXPRESSION* is false
EXPRESSION1 **-a** *EXPRESSION2*	both *EXPRESSION1* and *EXPRESSION2* are true
EXPRESSION1 **-o** *EXPRESSION2*	either *EXPRESSION1* or *EXPRESSION2* is true
-n *STRING*	the length of *STRING* is nonzero
STRING	equivalent to **-n** *STRING*
-z *STRING*	the length of *STRING* is zero
STRING1 = *STRING2*	the strings are equal
STRING1 **!=** *STRING2*	the strings are not equal
INTEGER1 **-eq** *INTEGER2*	*INTEGER1* is equal to *INTEGER2*
INTEGER1 **-ge** *INTEGER2*	*INTEGER1* is greater than or equal to *INTEGER2*
INTEGER1 **-gt** *INTEGER2*	*INTEGER1* is greater than *INTEGER2*

INTEGER1 **-le** *INTEGER2*	*INTEGER1* is less than or equal to *INTEGER2*
INTEGER1 **-lt** *INTEGER2*	*INTEGER1* is less than *INTEGER2*
INTEGER1 **-ne** *INTEGER2*	*INTEGER1* is not equal to *INTEGER2*
FILE1 **-ef** *FILE2*	*FILE1* and *FILE2* have the same device and inode numbers
FILE1 **-nt** *FILE2*	*FILE1* is newer (modification date) than *FILE2*
FILE1 **-ot** *FILE2*	*FILE1* is older than *FILE2*
-b *FILE*	*FILE* exists and is block special
-c *FILE*	*FILE* exists and is character special
-d *FILE*	*FILE* exists and is a directory
-e *FILE*	*FILE* exists
-f *FILE*	*FILE* exists and is a regular file
-g *FILE*	*FILE* exists and is set-group-ID
-G *FILE*	*FILE* exists and is owned by the effective group ID
-h *FILE*	*FILE* exists and is a symbolic link (same as **-L**)
-k *FILE*	*FILE* exists and has its sticky bit set
-L *FILE*	*FILE* exists and is a symbolic link (same as **-h**)
-O *FILE*	*FILE* exists and is owned by the effective user ID

-p *FILE*	*FILE* exists and is a named pipe
-r *FILE*	*FILE* exists and read permission is granted
-s *FILE*	*FILE* exists and has a size greater than zero
-S *FILE*	*FILE* exists and is a socket
-t *FD*	file descriptor *FD* is opened on a terminal
-u *FILE*	*FILE* exists and its set-user-ID bit is set
-w *FILE*	*FILE* exists and write permission is granted
-x *FILE*	*FILE* exists and execute (or search) permission is granted

Except for **-h** and **-L**, all FILE-related tests dereference symbolic links. Beware that parentheses need to be escaped (e.g., by backslashes) for shells. **INTEGER** may also be**-l STRING**, which evaluates to the length of **STRING**.

NOTE: your shell may have its own version of **test**, which usually supersedes the version described here. Please refer to your shell's documentation for details about the options it supports.

Examples

```
test 100 -gt 99 && echo "Yes, that's true." || echo "No, that's false."
```

This command will print the text "**Yes, that's true.**" because **100** is greater than **99**.

```
test 100 -lt 99 && echo "Yes." || echo "No."
```

This command will print the text "**No.**" because **100** is not less than **99**.

```
[ "awesome" = "awesome" ]; echo $?
```

This command will print "**0**" because the expression is true; the two strings are identical.

```
[ 5 -eq 6 ]; echo $?
```

This command will print "**1**" because the expression is false; 5 does not equal 6.

b. Write a shell script that accepts one or more file name as arguments and converts all of them to uppercase, provided they exist in the current directory.

```
$vi upper.sh
for file in *
do
  if [ -f $file ]
  then
      echo $file | tr '[a-z]' '[A-Z]'
  fi
done
```

```
# get filename
echo -n "Enter File Name : "
read fileName
# make sure file exits for reading
if [ ! -f $fileName ]
then
echo "Filename $fileName does not exists"
exit 1
fi
# convert uppercase to lowercase using tr command
tr '[A-Z]' '[a-z]' < $fileName
```

Tr command:

tr - translate or delete characters

Syntax

The syntax of tr command is:

```
$ tr [OPTION] SET1 [SET2]
```

Translation

If both the SET1 and SET2 are specified and '-d' OPTION is not specified, then tr command will replace each characters in SET1 with each character in same position in SET2.

1. **Convert lower case to upper case**

The following tr command is used to convert the lower case to upper case

```
$ tr abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
thegeekstuff
THEGEEKSTUFF
```

The following command will also convert lower case to upper case

```
$ tr [:lower:] [:upper:]
thegeekstuff
THEGEEKSTUFF
```

You can also use ranges in tr. The following command uses ranges to convert lower to upper case.

```
$ tr a-z A-Z
thegeekstuff
THEGEEKSTUFF
```

2. **Translate braces into parenthesis**

You can also translate from and to a file. In this example we will translate braces in a file with parenthesis.

```
$ tr '{}' '()' < inputfile > outputfile
```

The above command will read each character from “inputfile”, translate if it is a brace, and write the output in “outputfile”.

3. **Translate white-space to tabs**

The following command will translate all the white-space to tabs

```
$ echo "This is for testing" | tr [:space:] '\t'
This	is	for	testing
```

4. **Squeeze repetition of characters using -s**

In Example 3, we see how to translate space with tabs. But if there are two are more spaces present continuously, then the previous command will translate each spaces to a tab as follows.

```
$ echo "This   is   for testing" | tr [:space:] '\t'
This		is		for	testing
```

We can use -s option to squeeze the repetition of characters.

```
$ echo "This   is   for testing" | tr -s [:space:] '\t'
This	is	for	testing
```

Similarly you can convert multiple continuous spaces with a single space

```
$ echo "This  is  for testing" | tr -s [:space:] ' '
This is for testing
```

5. **Delete specified characters using -d option**

tr can also be used to remove particular characters using -d option.

```
$ echo "the geek stuff" | tr -d 't'
he geek suff
```

To remove all the digits from the string, use

```
$ echo "my username is 432234" | tr -d [:digit:]
my username is
```

Also, if you like to delete lines from file, you can use sed d command.

6. **Complement the sets using -c option**

You can complement the SET1 using -c option. For example, to remove all characters except digits, you can use the following.

```
$ echo "my username is 432234" | tr -cd [:digit:]
```

```
432234
```

7. **Remove all non-printable character from a file**

The following command can be used to remove all non-printable characters from a file.

```
$ tr -cd [:print:] < file.txt
```

8. **Join all the lines in a file into a single line**

The below command will translate all newlines into spaces and make the result as a single line.

```
$ tr -s '\n' ' ' < file.txt
```

c. Write a shell script that determines the period for which a specified user is working on.

```
echo  "Enter the USER NAME : "
read  user
last  $user
```

Output :

```
$ sh logtime.sh
Enter the USER NAME :
cse123
cse123    tty7        :0              Fri Sep 26
13:27   still logged in
cse123    pts/1       :0.0            Thu Sep 25
15:08 - 15:45  (00:37)
cse123    tty7        :0              Thu Sep 25
14:53 - 16:32  (01:39)
cse123    tty7        :0              Thu Sep 25
14:13 - 14:25  (00:11)
cse123    tty7        :0              Tue Sep 23
13:54 - 15:30  (01:36)
cse123    pts/2       :20.0           Mon Sep 22
17:02 - 17:23  (00:21)

(or)
# w -h root | awk '{print $1,"\t",$3}'
root 25Sep0810days
root 25Sep0810days
root 25Sep0810days
```

Last command:

The last command reads listing of last logged in users from the system file called /var/log/wtmp or the file designated by the -f options.

Purpose

To find out when a **particular user last logged in** to the Linux or Unix server.

Syntax

The basic syntax is:

last

last [userNameHere]

last [tty]

last [options] [userNameHere]

If no options provided last command displays a list of all users logged in (and out) since /var/log/wtmp file was created. You can filter out results by supplying names of users and tty's to show only those entries matching the username/tty.

last command examples

To find out who has recently logged in and out on your server, type:

```
$ last
```

Sample outputs:

```
oot     pts/1        10.1.6.120      Tue Jan 28 05:59   still logged in
root     pts/0        10.1.6.120      Tue Jan 28 04:08   still logged in
root     pts/0        10.1.6.120      Sat Jan 25 06:33 - 08:55  (02:22)
root     pts/1        10.1.6.120      Thu Jan 23 14:47 - 14:51  (00:03)
root     pts/0        10.1.6.120      Thu Jan 23 13:02 - 14:51  (01:48)
root     pts/0        10.1.6.120      Tue Jan  7 12:02 - 12:38  (00:35)

wtmp begins Tue Jan  7 12:02:54 2014
```

You can specifies a file to search other than /var/log/wtmp using -f option. For example, search /nas/server/webserver/.log/wtmp:

```
$ last -f /nas/server/webserver/.log/wtmp
```

```
last -f /nas/server/webserver/.log/wtmp userNameHere
```

List all users last logged in/out time

last command searches back through the file /var/log/wtmp file and the output may go back to several months. Just use the less command or more command as follows to display output one screen at a time:

```
$ last | more
```

```
last | less
```

List a particular user last logged in

To find out when user vivek last logged in, type:

```
$ last vivek
```

```
$ last vivek | less
```

```
$ last vivek | grep 'Thu Jan 23'
```

For loop

Syntax: for

```
for var in word1 word2 ... wordN
do
   Statement(s) to be executed for every word.
done
```

Here *var* is the name of a variable and word1 to wordN are sequences of characters separated by spaces (words). Each time the for loop executes, the value of the variable var is set to the next word in the list of words, word1 to wordN.

Example:

Here is a simple example that uses for loop to span through the given list of numbers:

```
#!/bin/sh

for var in 0 1 2 3 4 5 6 7 8 9
do
   echo $var
done
```

This will produce following result:

```
0
1
2
3
4
5
6
7
8
9
```

Following is the example to display all the files starting with **.bash** and available in your home. I'm executing this script from my root:

```
#!/bin/sh

for FILE in $HOME/.bash*
do
   echo $FILE
done
```

This will produce following result:

```
/root/.bash_history
/root/.bash_logout
/root/.bash_profile
/root/.bashrc
```