Introduction to Shell Scripting

A Shell provides you with an interface to the Linux system. A shell script is a simple executable document that contains a sequence of commands that we might want to perform on a regular basis. By putting them into a shell script, we are reducing the job to just single command execution. Let’s look at some sample codes.

Code:

```
1.#!/bin/bash
2.# Hello World Bash Shell Script
  # ----------------------------
  # print
3.echo "Hello World”
#End of the script
```

Code Interpretation:

Line1 is called a hashbang or shebang, it tells Linux that this script should be run through the /bin/bash shell. Linux flags line2 as a comment, and the rest of the line is completely ignored. In line3, we are printing “Hello World” on the terminal with the help of an echo command.
After saving the above file, we need to give it execute permission to make it runnable. If the filename is hello.sh then you can set the execute permission as follows:

```
chmod +x  hello.sh
```

Now, execute the file using one of the following commands:

```
bash hello.sh
```

OR

```
./hello.sh
```

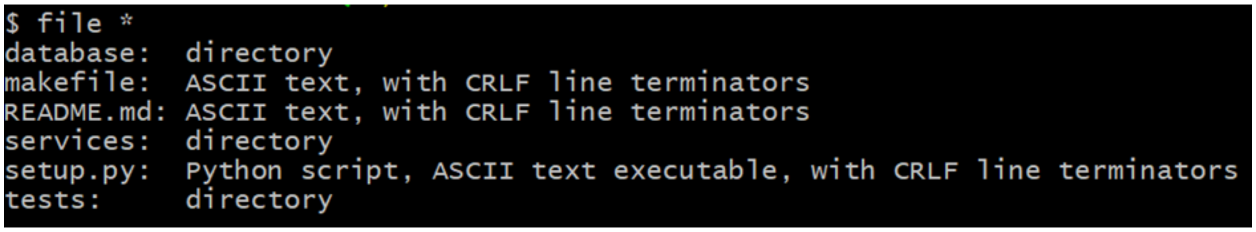
Wildcards

A wildcard is a character that can be used as a substitute for any of a class of characters in a search, thereby greatly increasing the flexibility and efficiency of searches.

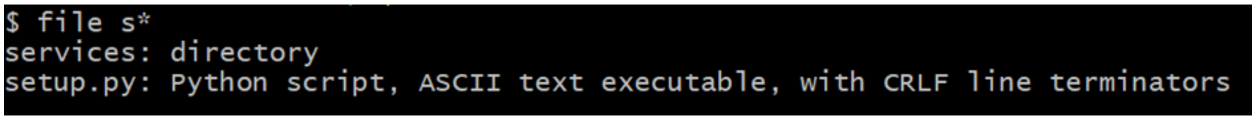
Wildcard	**Meaning**	**Example**
*	It can represent zero or more number of characters	ls *.html to list all the files in the current directory that have .html as extension
?	Represents exactly one character	ls ?? to list all the files in the current directory that have exactly two characters
[]	To represent any of the characters enclosed in the brackets	ls stat[0-9] to display all the filenames in the current directory with name stat followed by a digit

[!]	Match any single character not in the bracketed set	ls intro[!a-z] to display all the filenames in the current directory with the name intro followed by any character other than characters a to z

Observe and understand the output of the following commands file *, file s*:



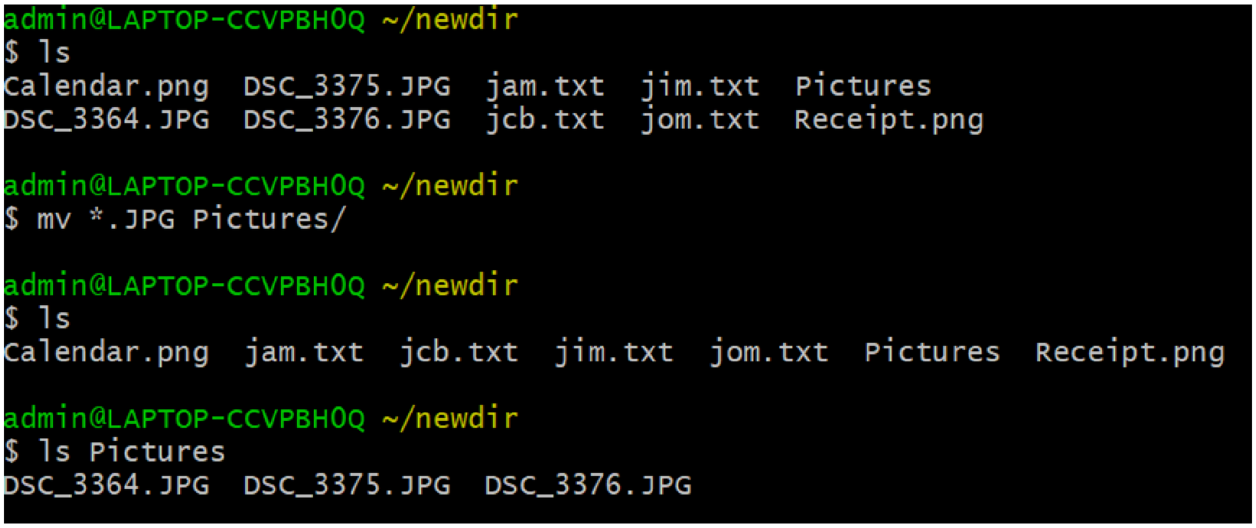
```
$ file *
database:  directory
makefile:  ASCII text, with CRLF line terminators
README.md: ASCII text, with CRLF line terminators
services:  directory
setup.py:  Python script, ASCII text executable, with CRLF line terminators
tests:     directory
```



```
$ file s*
services: directory
setup.py: Python script, ASCII text executable, with CRLF line terminators
```

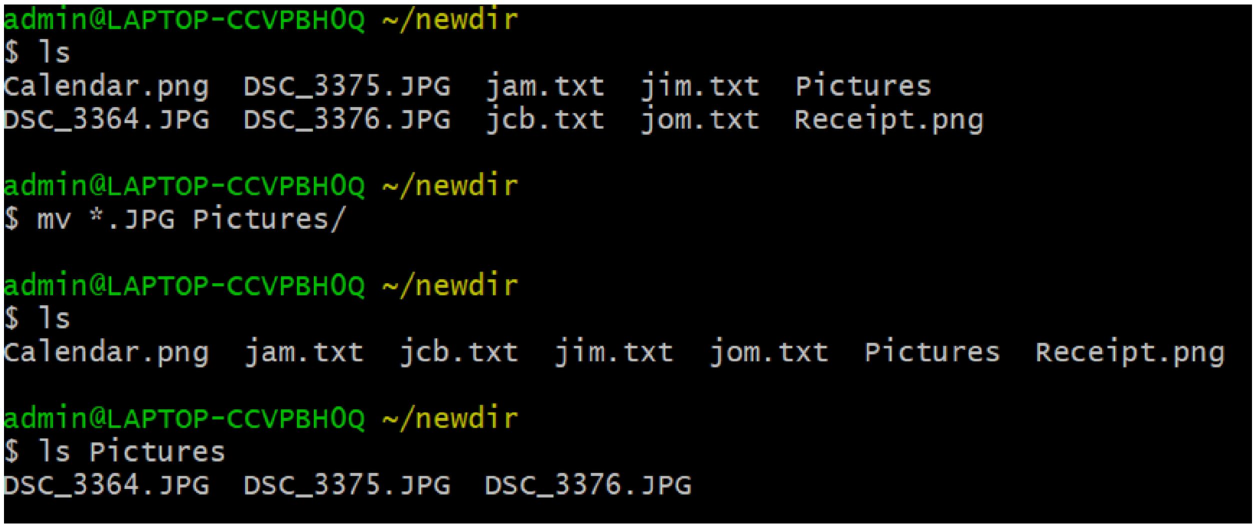
Observe and understand the outputs of the following commands:

a. mv *.jpg Pictures/



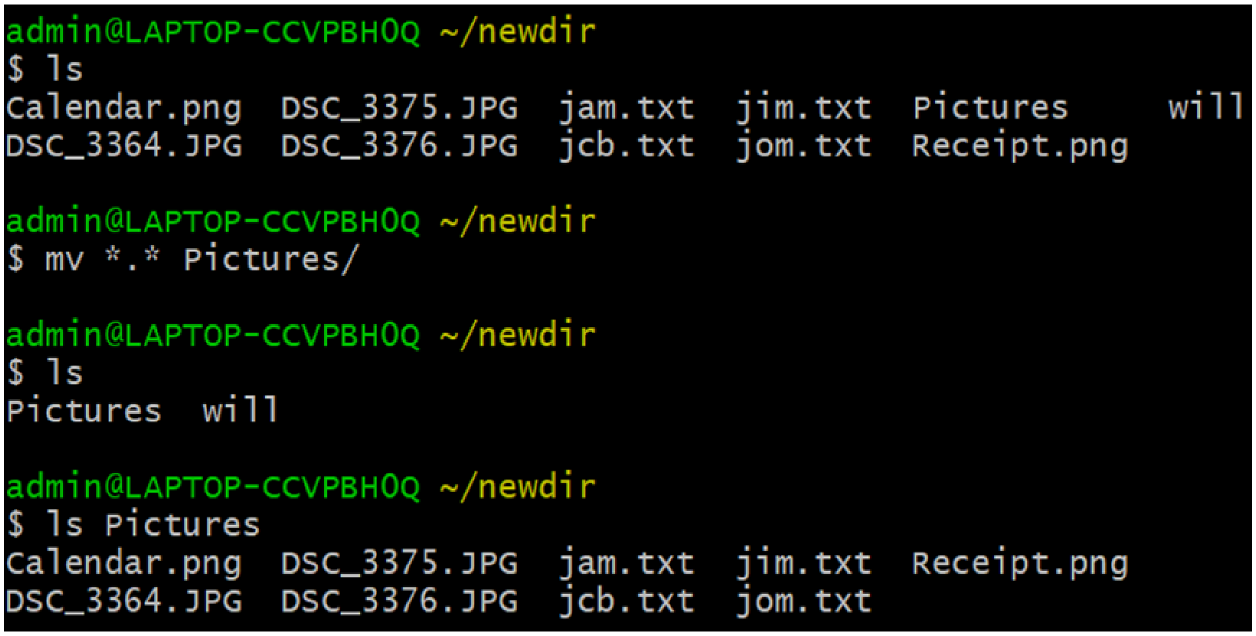
```
admin@LAPTOP-CCVPBH0Q ~/newdir
$ ls
Calendar.png  DSC_3375.JPG  jam.txt  jim.txt  Pictures
DSC_3364.JPG  DSC_3376.JPG  jcb.txt  jom.txt  Receipt.png

admin@LAPTOP-CCVPBH0Q ~/newdir
$ mv *.JPG Pictures/

admin@LAPTOP-CCVPBH0Q ~/newdir
$ ls
Calendar.png  jam.txt  jcb.txt  jim.txt  jom.txt  Pictures  Receipt.png

admin@LAPTOP-CCVPBH0Q ~/newdir
$ ls Pictures
DSC_3364.JPG  DSC_3375.JPG  DSC_3376.JPG
```

b. mv *.??? Pictures/



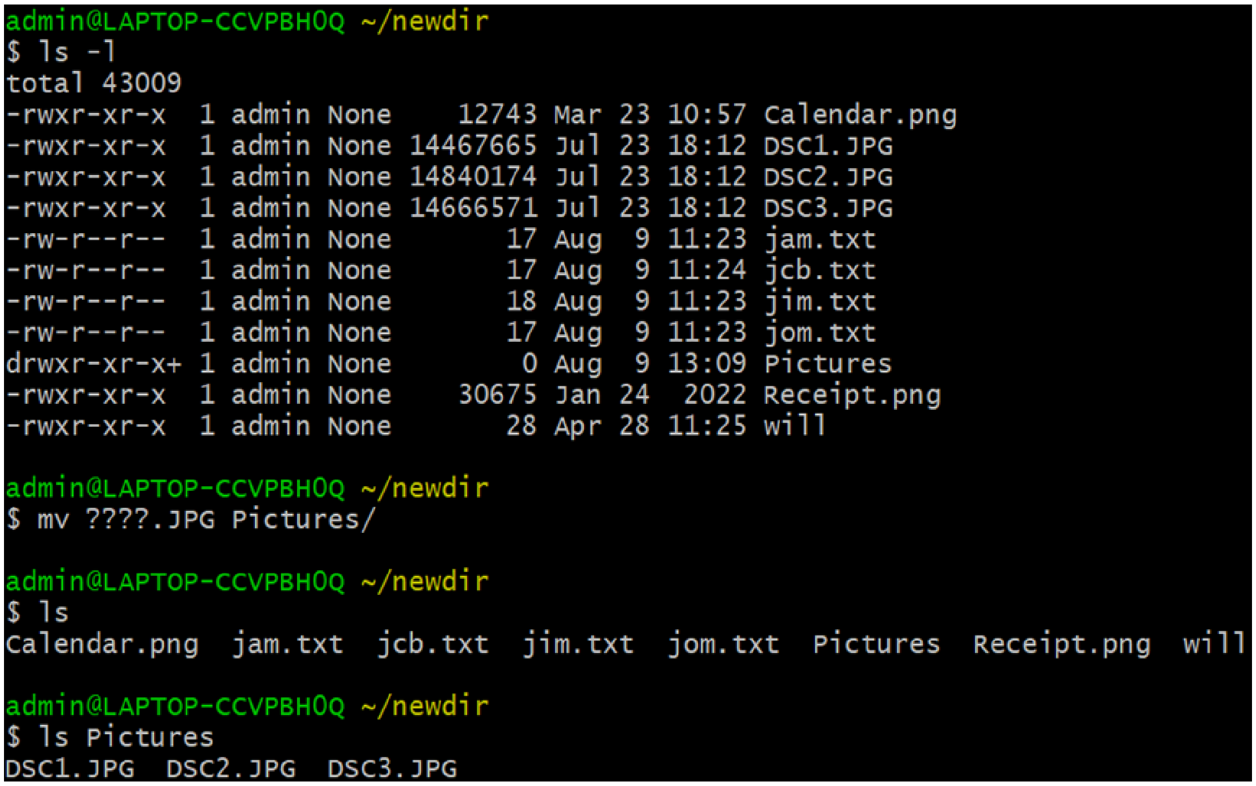
```
admin@LAPTOP-CCVPBH0Q ~/newdir
$ ls
Calendar.png  DSC_3375.JPG  jam.txt  jim.txt  Pictures
DSC_3364.JPG  DSC_3376.JPG  jcb.txt  jom.txt  Receipt.png

admin@LAPTOP-CCVPBH0Q ~/newdir
$ mv *.JPG Pictures/

admin@LAPTOP-CCVPBH0Q ~/newdir
$ ls
Calendar.png  jam.txt  jcb.txt  jim.txt  jom.txt  Pictures  Receipt.png

admin@LAPTOP-CCVPBH0Q ~/newdir
$ ls Pictures
DSC_3364.JPG  DSC_3375.JPG  DSC_3376.JPG
```

c. mv *.* Pictures/



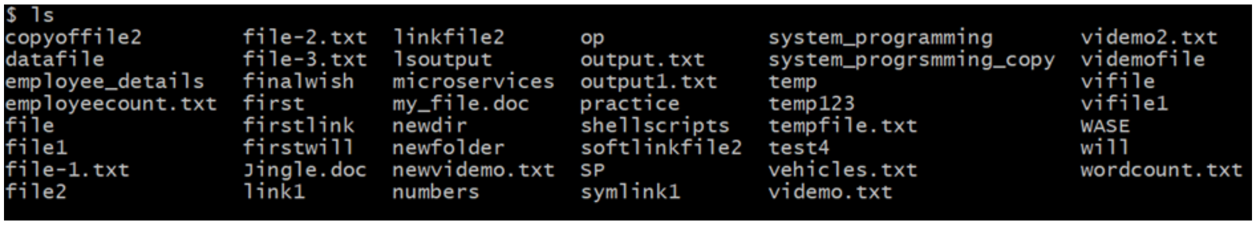
```
admin@LAPTOP-CCVPBH0Q ~/newdir
$ ls
Calendar.png  DSC_3375.JPG  jam.txt  jim.txt  Pictures     will
DSC_3364.JPG  DSC_3376.JPG  jcb.txt  jom.txt  Receipt.png

admin@LAPTOP-CCVPBH0Q ~/newdir
$ mv *.* Pictures/

admin@LAPTOP-CCVPBH0Q ~/newdir
$ ls
Pictures  will

admin@LAPTOP-CCVPBH0Q ~/newdir
$ ls Pictures
Calendar.png  DSC_3375.JPG  jam.txt  jim.txt  Receipt.png
DSC_3364.JPG  DSC_3376.JPG  jcb.txt  jom.txt
```

d. mv ????.jpg Pictures/



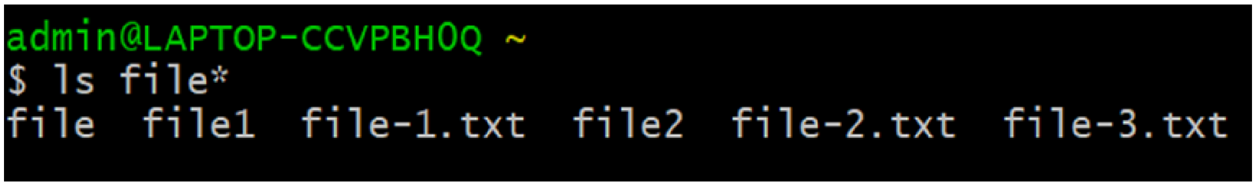
```
admin@LAPTOP-CCVPBH0Q ~/newdir
$ ls -l
total 43009
-rwxr-xr-x  1 admin None    12743 Mar 23 10:57 Calendar.png
-rwxr-xr-x  1 admin None 14467665 Jul 23 18:12 DSC1.JPG
-rwxr-xr-x  1 admin None 14840174 Jul 23 18:12 DSC2.JPG
-rwxr-xr-x  1 admin None 14666571 Jul 23 18:12 DSC3.JPG
-rw-r--r--  1 admin None       17 Aug  9 11:23 jam.txt
-rw-r--r--  1 admin None       17 Aug  9 11:24 jcb.txt
-rw-r--r--  1 admin None       18 Aug  9 11:23 jim.txt
-rw-r--r--  1 admin None       17 Aug  9 11:23 jom.txt
drwxr-xr-x+ 1 admin None        0 Aug  9 13:09 Pictures
-rwxr-xr-x  1 admin None    30675 Jan 24  2022 Receipt.png
-rwxr-xr-x  1 admin None       28 Apr 28 11:25 will

admin@LAPTOP-CCVPBH0Q ~/newdir
$ mv ????.JPG Pictures/

admin@LAPTOP-CCVPBH0Q ~/newdir
$ ls
Calendar.png  jam.txt  jcb.txt  jim.txt  jom.txt  Pictures  Receipt.png  will

admin@LAPTOP-CCVPBH0Q ~/newdir
$ ls Pictures
DSC1.JPG  DSC2.JPG  DSC3.JPG
```

Observe and understand the output of the following commands:



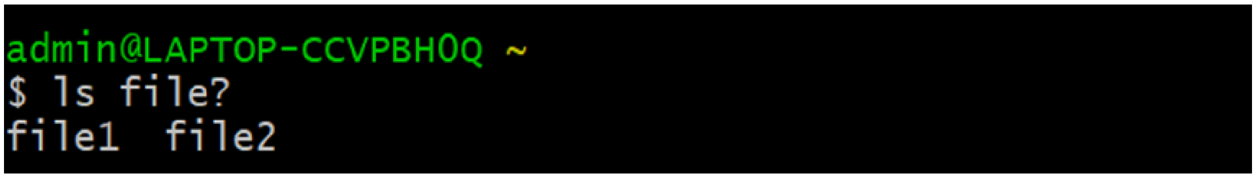
```
$ ls
copyoffile2        file-2.txt  linkfile2      op             system_programming        videmo2.txt
datafile           file-3.txt  lsoutput       output.txt     system_progrsmming_copy   videmofile
employee_details   finalwish   microservices  output1.txt    temp                      vifile
employeecount.txt  first       my_file.doc    practice       temp123                   vifile1
file               firstlink   newdir         shellscripts   tempfile.txt              WASE
file1              firstwill   newfolder      softlinkfile2  test4                     will
file-1.txt         Jingle.doc  newvidemo.txt  SP             vehicles.txt              wordcount.txt
file2              link1       numbers        symlink1       videmo.txt
```

a. ls file*



```
admin@LAPTOP-CCVPBH0Q ~
$ ls file*
file  file1  file-1.txt  file2  file-2.txt  file-3.txt
```

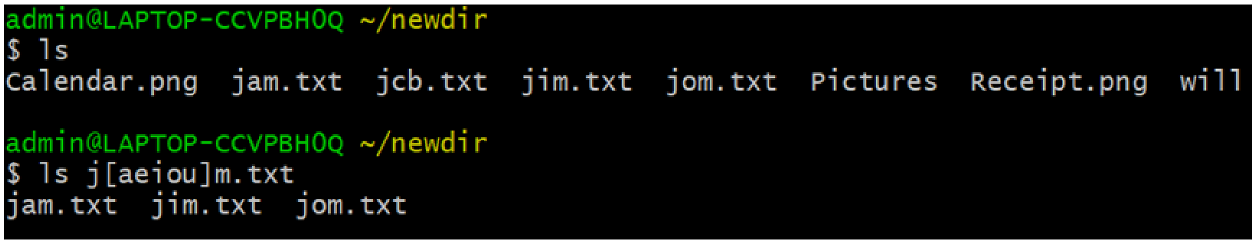
b. ls file?



```
admin@LAPTOP-CCVPBH0Q ~
$ ls file?
file1  file2
```

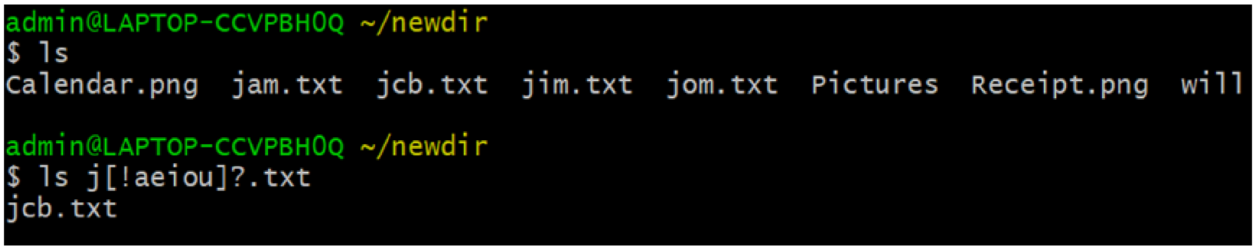
Observe and understand the output of the following commands:

a. ls j[aeiou]m.txt



```
admin@LAPTOP-CCVPBH0Q ~/newdir
$ ls
Calendar.png  jam.txt  jcb.txt  jim.txt  jom.txt  Pictures  Receipt.png  will

admin@LAPTOP-CCVPBH0Q ~/newdir
$ ls j[aeiou]m.txt
jam.txt  jim.txt  jom.txt
```

b. ls j[!aeiou]?.txt



```
admin@LAPTOP-CCVPBH0Q ~/newdir
$ ls
Calendar.png  jam.txt  jcb.txt  jim.txt  jom.txt  Pictures  Receipt.png  will

admin@LAPTOP-CCVPBH0Q ~/newdir
$ ls j[!aeiou]?.txt
jcb.txt
```

Meta-characters

The characters that have a special meaning attached to them.

Meta-character	Meaning
>	Output redirection (overwrite)
>>	Output redirection (append)

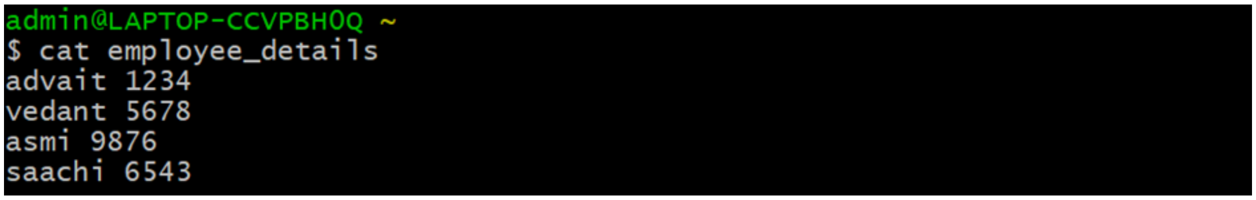
<	Input redirection
*	Substitution wildcard; zero or more characters
?	Substitution wildcard; one character
[]	Substitution wildcard; any character between brackets
|	Pipe
||	OR conditional execution
&&	AND conditional execution
#	Comment
$	Expand the value of a variable
\	Escape interpretation of the next character
;	Command terminator

Sometimes we might need to pass meta-characters to a Linux command and do not want the shell to interpret them. There are following two options to avoid shell interpretation of meta-characters:

a. Escape the meta-character with a backslash (\). Example: echo \$800 to print $800.

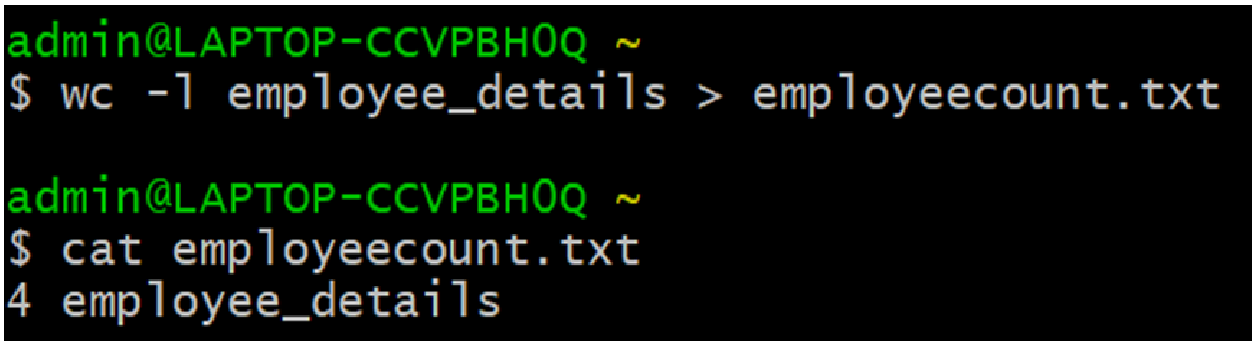
b. Use single quotes (' ') around a string. Example: echo $800" to print '$800 '.

Read the employee details file carefully. Observe and analyze the outputs.



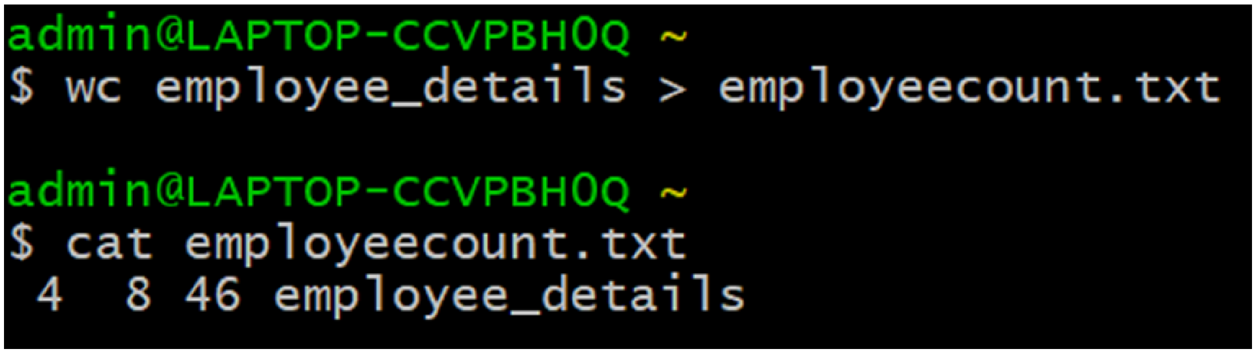
```
admin@LAPTOP-CCVPBHOQ ~
$ cat employee_details
advait 1234
vedant 5678
asmi 9876
saachi 6543
```

a. By using output redirection, we can send the output to a file instead of showing it on the terminal.



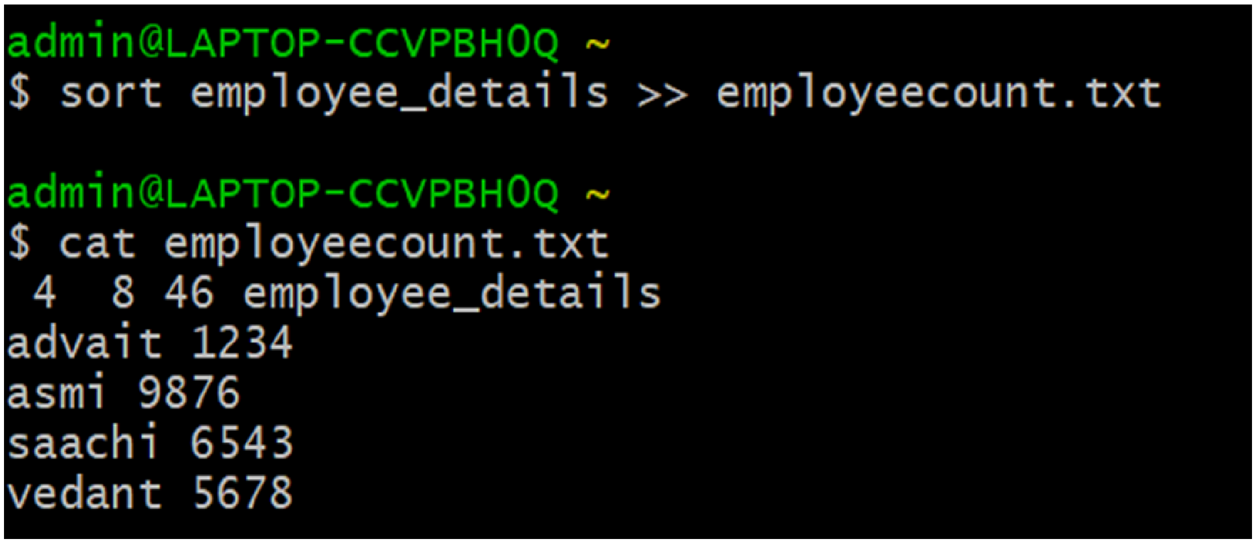
```
admin@LAPTOP-CCVPBHOQ ~
$ wc -l employee_details > employeecount.txt

admin@LAPTOP-CCVPBHOQ ~
$ cat employeecount.txt
4 employee_details
```

b. After part (a), if we run wc employee_details > employeecount.txt, then the contents of the file get overwritten.



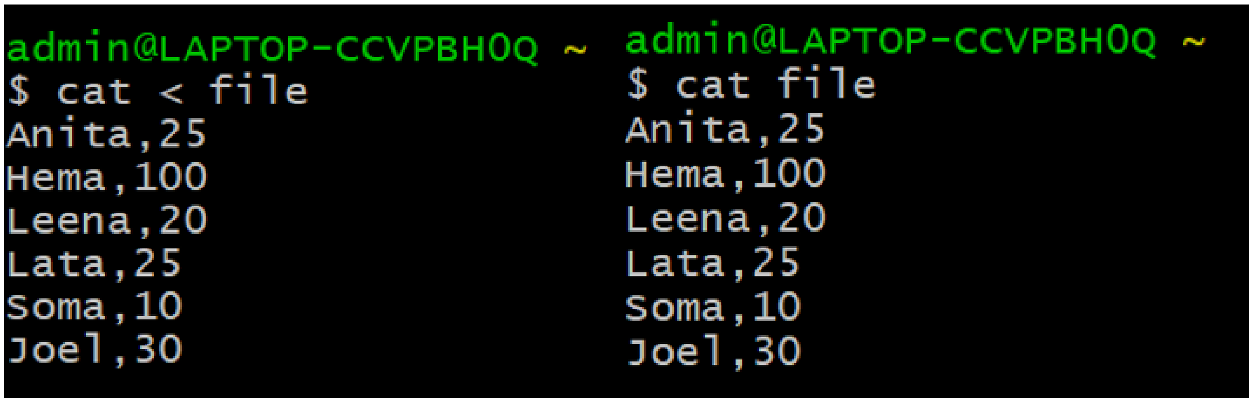
```
admin@LAPTOP-CCVPBHOQ ~
$ wc employee_details > employeecount.txt

admin@LAPTOP-CCVPBHOQ ~
$ cat employeecount.txt
 4  8 46 employee_details
```

c. After part (b), if we run sort employee_details >> employeecount.txt, the output of this command gets appended to the already existing contents of the file.



```
admin@LAPTOP-CCVPBHOQ ~
$ sort employee_details >> employeecount.txt

admin@LAPTOP-CCVPBHOQ ~
$ cat employeecount.txt
 4  8 46 employee_details
advait 1234
asmi 9876
saachi 6543
vedant 5678
```

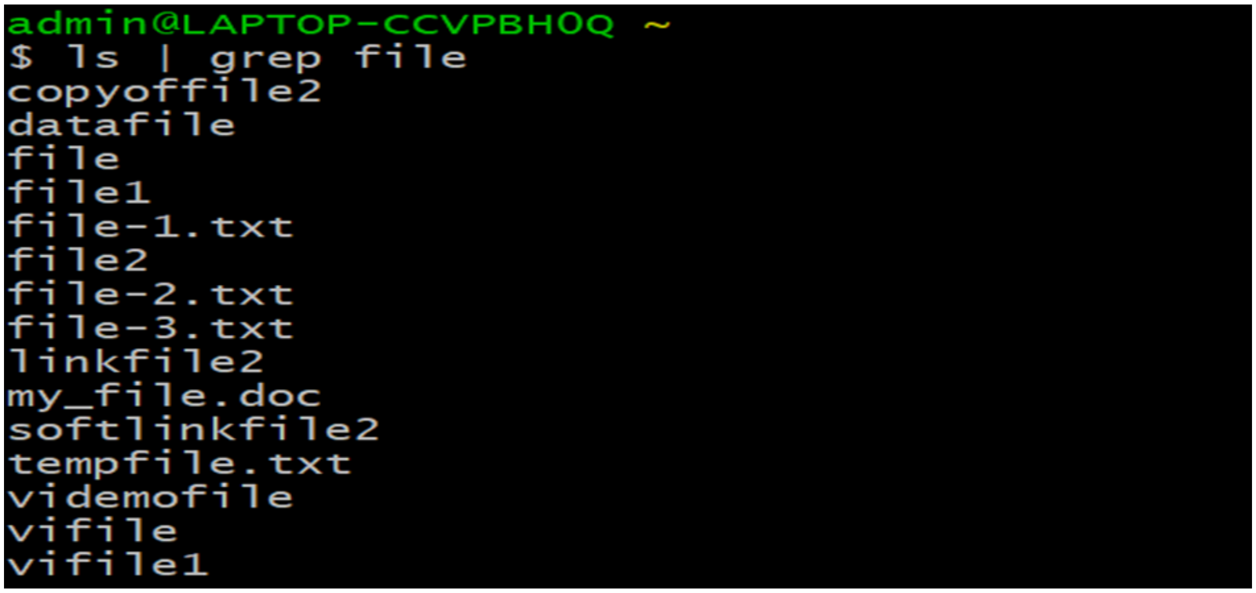
Observe and understand if there is any difference between cat file1 and cat < file1.



```
admin@LAPTOP-CCVPBHOQ ~
$ cat < file
Anita,25
Hema,100
Leena,20
Lata,25
Soma,10
Joel,30
```

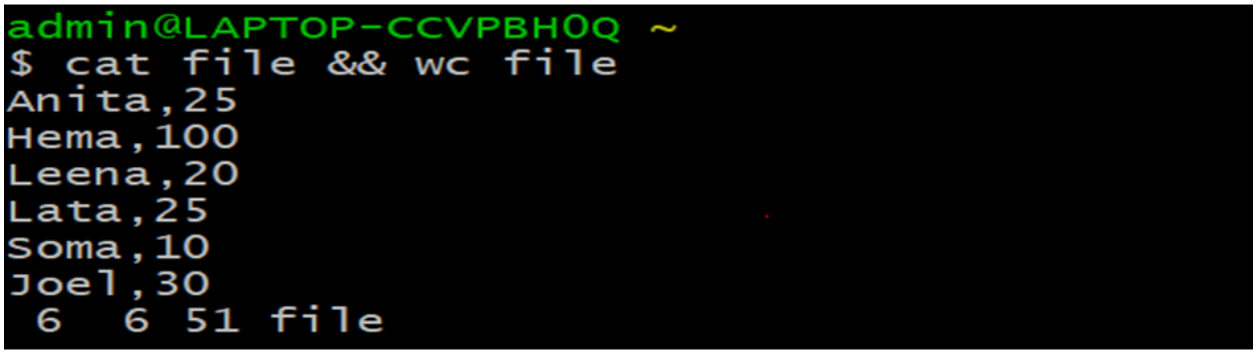
```
admin@LAPTOP-CCVPBHOQ ~
$ cat file
Anita,25
Hema,100
Leena,20
Lata,25
Soma,10
Joel,30
```

List all files and directories and give them as input to `grep` command using piping in Linux. Then, grep is listing those lines that contain the word “file.”



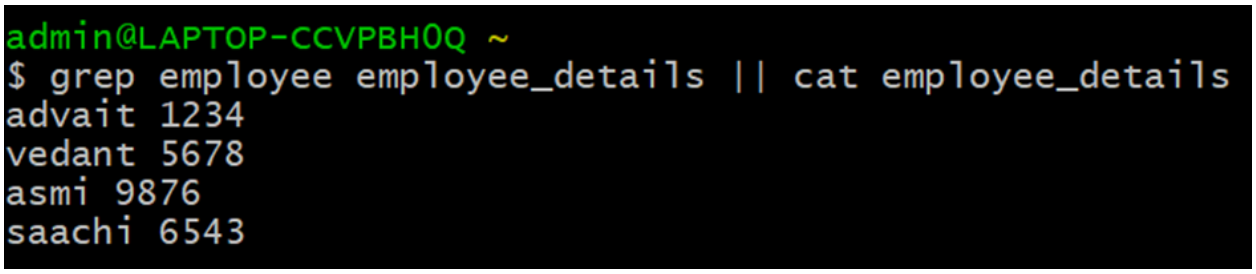
```
admin@LAPTOP-CCVPBHOQ ~
$ ls | grep file
copyoffile2
datafile
file
file1
file-1.txt
file2
file-2.txt
file-3.txt
linkfile2
my_file.doc
softlinkfile2
tempfile.txt
videmofile
vifile
vifile1
```

The command shell interprets the && as the logical AND. When using this command, the second command will be executed only when the first one has been successfully executed.



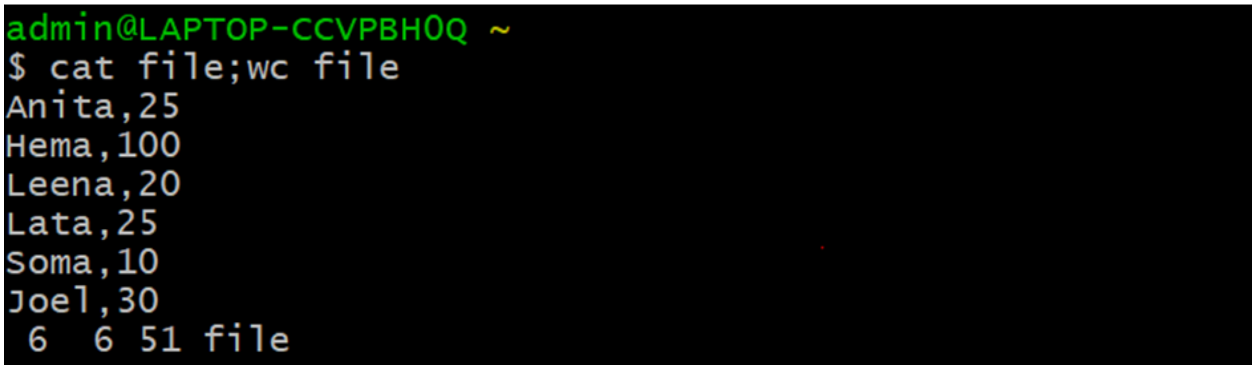
```
admin@LAPTOP-CCVPBHOQ ~
$ cat file && wc file
Anita,25
Hema,100
Leena,20
Lata,25
Soma,10
Joel,30
 6  6 51 file
```

The || represents a logical OR. The second command is executed only when the first command fails.



```
admin@LAPTOP-CCVPBHOQ ~
$ grep employee employee_details || cat employee_details
advait 1234
vedant 5678
asmi 9876
saachi 6543
```

We can execute multiple commands by separating them using “;”



```
admin@LAPTOP-CCVPBH0Q ~
$ cat file;wc file
Anita,25
Hema,100
Leena,20
Lata,25
Soma,10
Joel,30
 6  6 51 file
```

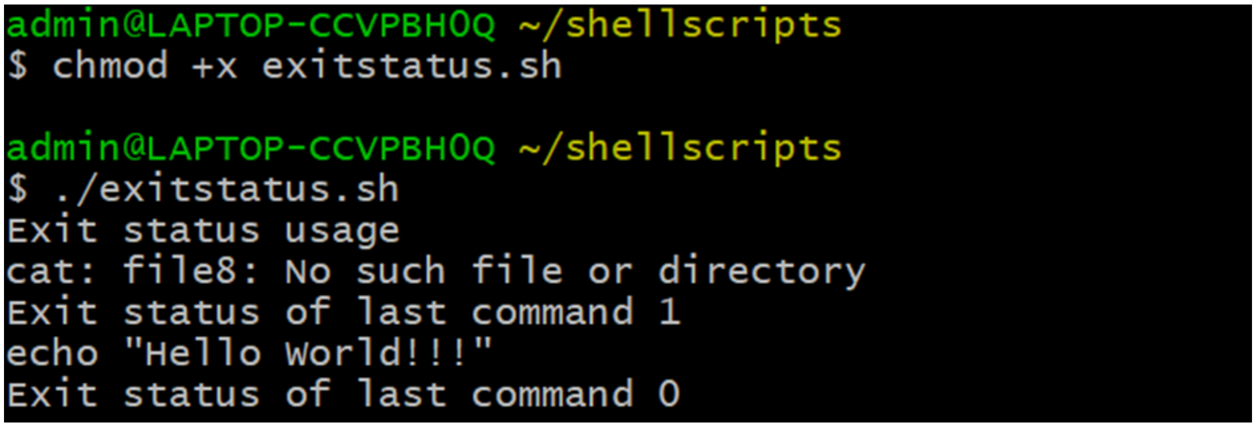
Special Variables:

By using variables, we can make a script generic and apply it to various situations. Special variables are the ones that are reserved for some specific functions.

Special Variable	Description
$0	The filename of the current script.
$n	"n" refers to a positive number that represents the nth argument passed to the script. For example, $1 represents the first argument. We can have nine such arguments in bash shell $0, $1, $2, ... $9.
$#	Represents the number of arguments passed to the script
$*	Represents all the arguments passed to the script.
$?	Returns the exit status of the last command that was executed.
$!	Holds the process ID of the last background command.
$$	Represents the process ID of the current shell. For shell scripts, this is the process ID under which the scripts run.
$@	Represents all the arguments passed to the script.

Example Code:

```
#!/bin/bash
echo "Exit status usage"
cat file8
echo "Exit status of last command $?"
cat hello.sh
echo "Exit status of last command $?"
```

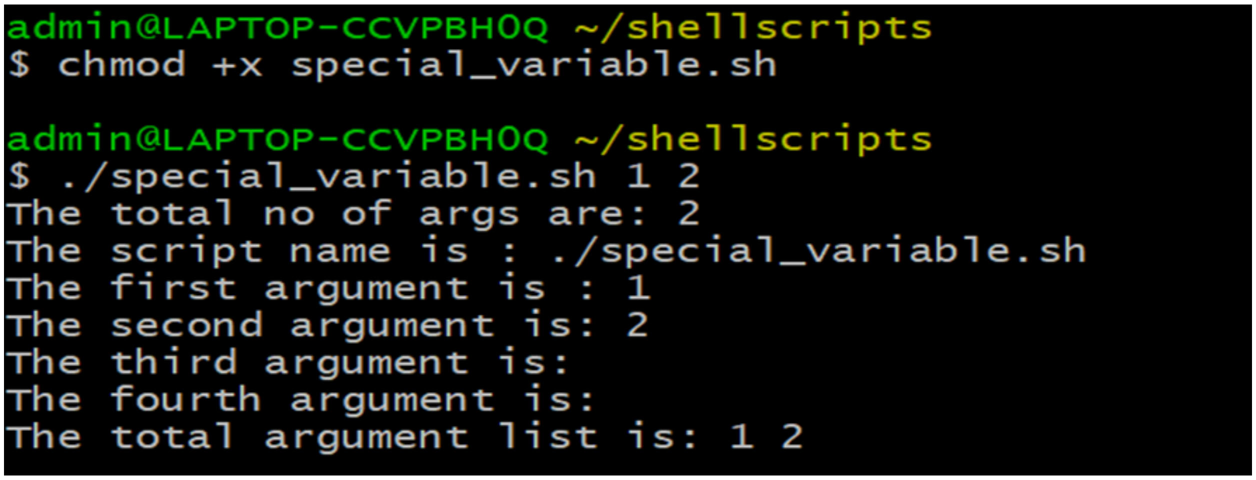


```
admin@LAPTOP-CCVPBHOQ ~/shellscripts
$ chmod +x exitstatus.sh

admin@LAPTOP-CCVPBHOQ ~/shellscripts
$ ./exitstatus.sh
Exit status usage
cat: file8: No such file or directory
Exit status of last command 1
echo "Hello World!!!"
Exit status of last command 0
```

Observe and understand how the shell script responds to different options:

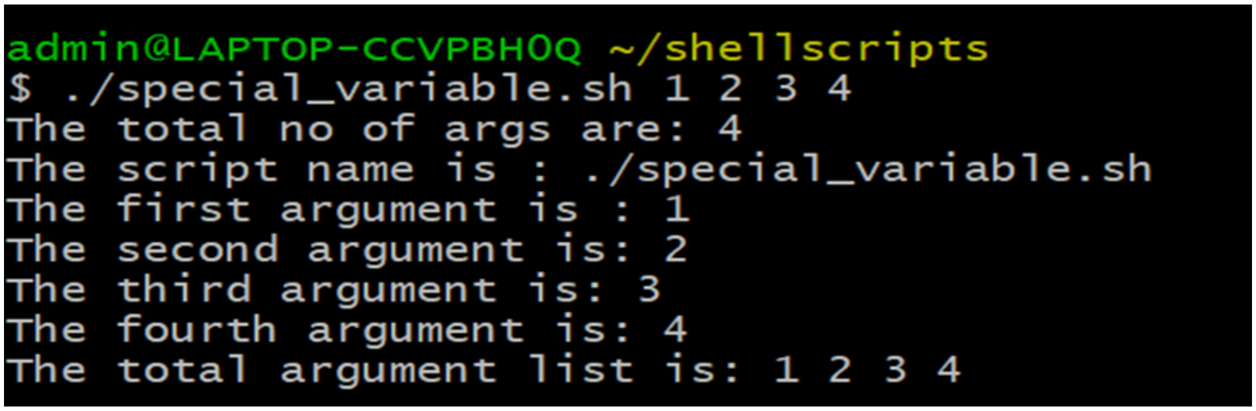
```
cat> special_variable.sh
#!/usr/bin/bash
echo "The total no of args are: $#"
echo "The script name is : $0"
echo "The first argument is : $1"
echo "The second argument is: $2"
echo "The third argument is: $3"
echo "The fourth argument is: $4"
echo "The total argument list is: $*"
```

a. ./special_variable.sh 1 2



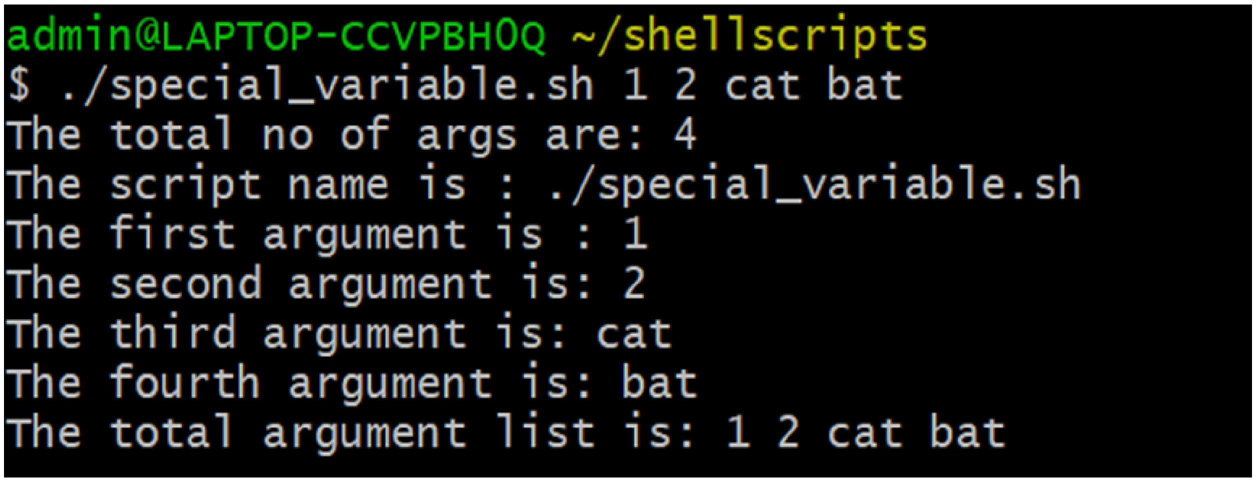
```
admin@LAPTOP-CCVPBHOQ ~/shellscripts
$ chmod +x special_variable.sh

admin@LAPTOP-CCVPBHOQ ~/shellscripts
$ ./special_variable.sh 1 2
The total no of args are: 2
The script name is : ./special_variable.sh
The first argument is : 1
The second argument is: 2
The third argument is:
The fourth argument is:
The total argument list is: 1 2
```

b. ./special_variable.sh 1 2 3 4



```
admin@LAPTOP-CCVPBHOQ ~/shellscripts
$ ./special_variable.sh 1 2 3 4
The total no of args are: 4
The script name is : ./special_variable.sh
The first argument is : 1
The second argument is: 2
The third argument is: 3
The fourth argument is: 4
The total argument list is: 1 2 3 4
```

c. ./special_variable.sh 1 2 cat bat

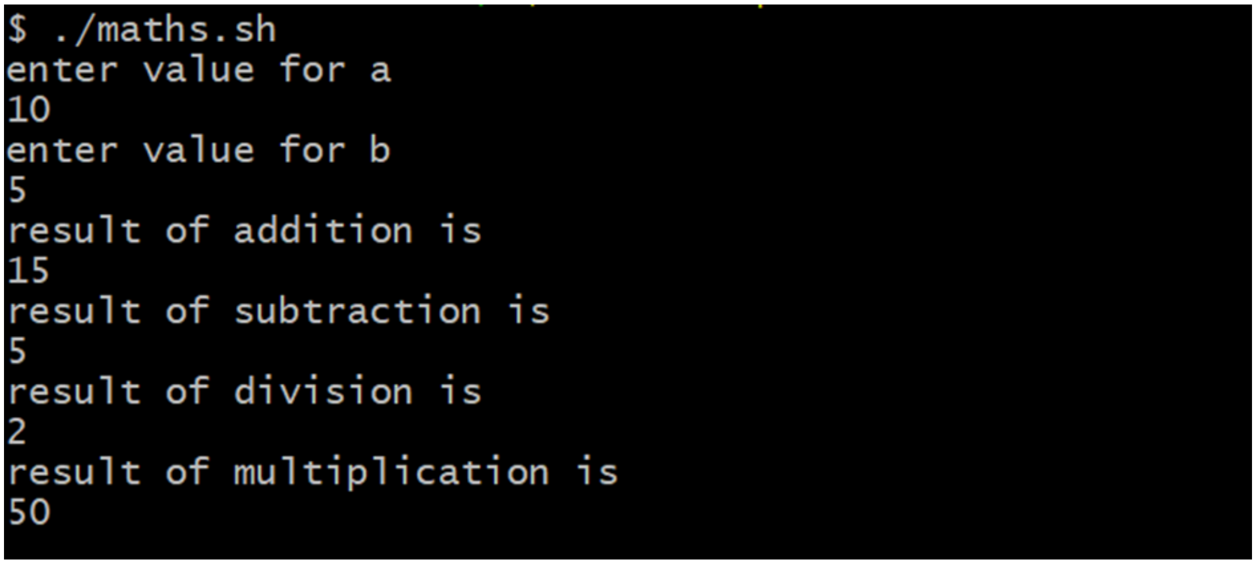


```
admin@LAPTOP-CCVPBHOQ ~/shellscripts
$ ./special_variable.sh 1 2 cat bat
The total no of args are: 4
The script name is : ./special_variable.sh
The first argument is : 1
The second argument is: 2
The third argument is: cat
The fourth argument is: bat
The total argument list is: 1 2 cat bat
```

Example Codes

Example1: Read two integer values as input from the user and perform basic mathematical operations on them.

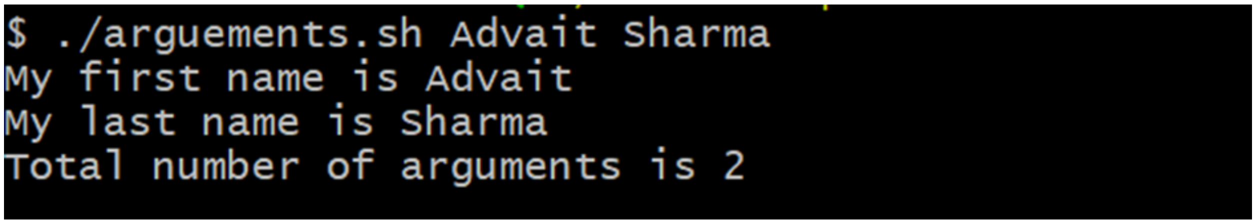
```
#!/bin/bash
echo "enter value for a"
read a
echo "enter value for b"
read b
echo result of addition is
c=`expr $a + $b`
echo $c
echo result of subtraction is
c=`expr $a - $b`
echo $c
echo result of division is
c=`expr $a / $b`
echo $c
echo result of multiplication is
c=`expr $a \* $b`
echo $c
```



```
$ ./maths.sh
enter value for a
10
enter value for b
5
result of addition is
15
result of subtraction is
5
result of division is
2
result of multiplication is
50
```

Example2: Using arguments in shell script

```
#!/bin/bash
# using arguments in a shell script
echo "My first name is $1"
echo "My last name is $2"
echo "Total number of arguments is $#"
```



```
$ ./arguements.sh Advait Sharma
My first name is Advait
My last name is Sharma
Total number of arguments is 2
```