Shell Programming

Shell Script

- A shell script is a list of commands in a computer program that is run by the Unix shell which is a command line interpreter
- .sh is extension for shell script
- Shell scripts are executed in a separate child shell process

Variables

- **Variables are symbolic names that represent values stored in memory**
- **Three different types of variables**
  - **Global Variables: Environment and configuration variables, capitalized, such as HOME, PATH, SHELL, USERNAME, and PWD.**

    When you login, there will be a large number of global System variables that are already defined. These can be freely referenced and used in your shell scripts.
  - **Local Variables**

    Within a shell script, you can create as many new variables as needed. Any variable created in this manner remains in existence only within that shell.
  - **Special Variables**

    Reversed for OS, shell programming, etc. such as positional parameters $0, $1 …

Referencing Variables

Variable contents are accessed using ‘$’:

e.g.
```
$ echo $HOME
$ echo $SHELL
```

Defining Local Variables

- As in any other programming language, variables can be defined and used in shell scripts.
- Unlike other programming languages, variables in Shell Scripts are not typed.

- Examples :

  a=10 ----- a is an Integer

  b=$a+1 -----will not perform arithmetic but be the string ‘1234+1’

  b=`expr $a + 1 ` -----will perform arithmetic so b is 1235 now.

  Note : +,-,/,*,**, % operators are available.

  b=abcde ------b is string

  b=‘abcde’ -----same as above but much safer.

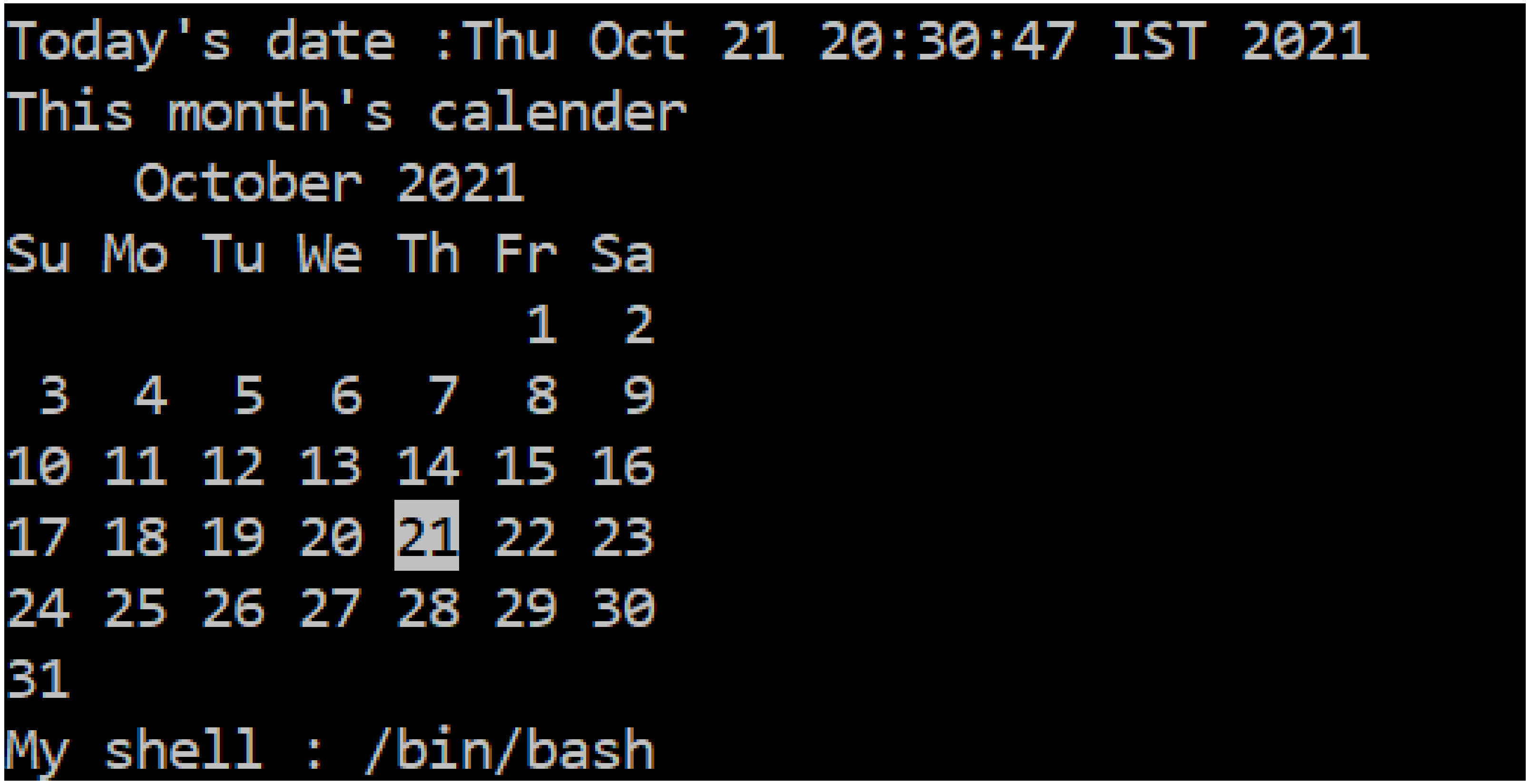
  IMPORTANT NOTE: DO NOT LEAVE SPACES AROUND THE =

Shell Script

```
#!/bin/sh
# script.sh : Sample shell script
echo “Today’s date : `date`”
echo “This month’s calender :”
cal `date “+%m 20%y”`
echo “My shell :$SHELL”
```

#! – Interpreter line begins with these characters followed by pathname of the shell to be used for running the script

Output



```
Today's date :Thu Oct 21 20:30:47 IST 2021
This month's calender
    October 2021
Su Mo Tu We Th Fr Sa
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
My shell : /bin/bash
```

read : Making scripts interactive

- The read command allows you to prompt for input and store it in a variable
- It is used with one or more variables
- Input supplied through the standard input is read into these variables

Example

read pname

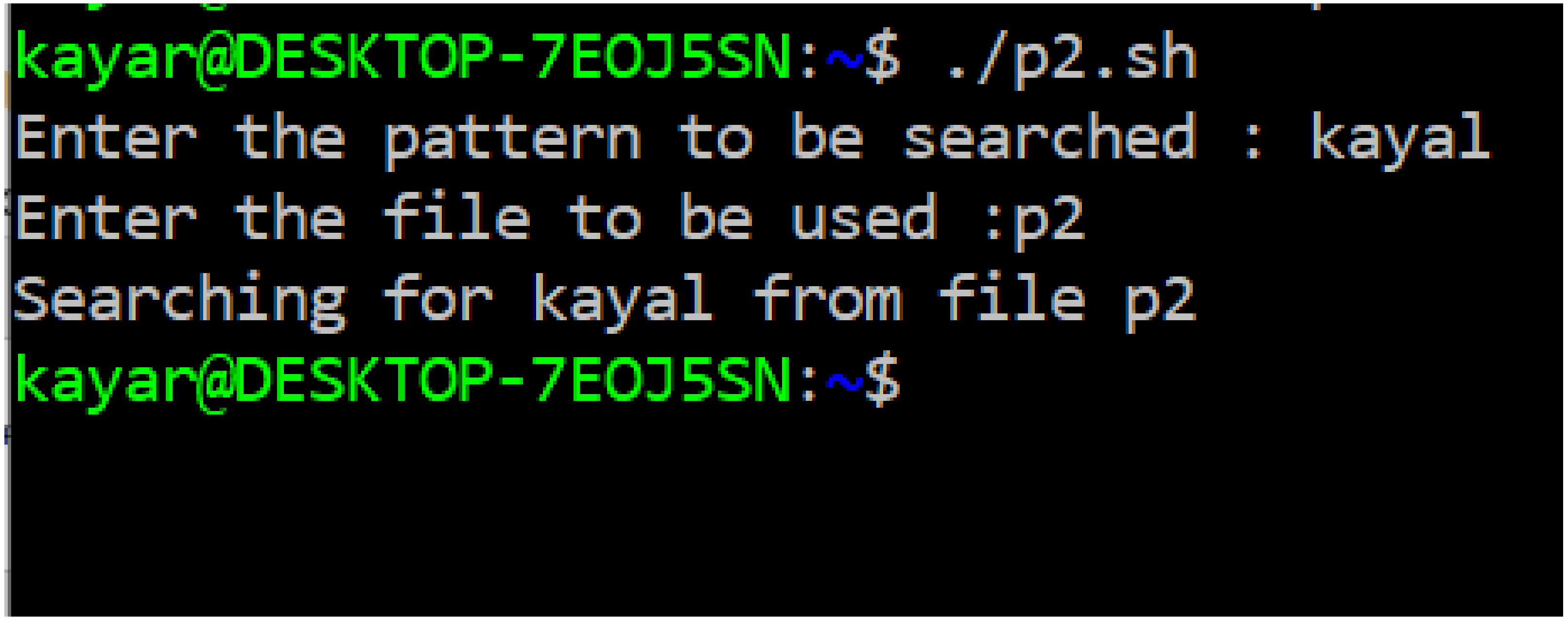
(or)

read pname flname

read : Making scripts interactive

```
#!/bin/sh
echo “Enter the pattern to be searched:\c”
read pname
echo “Enter the file to be used :\c”
read flname
echo “Searching for $pname from file $flname”
```

Output



```
kayar@DESKTOP-7EOJ5SN:~$ ./p2.sh
Enter the pattern to be searched : kayal
Enter the file to be used :p2
Searching for kayal from file p2
kayar@DESKTOP-7EOJ5SN:~$
```

grep Command

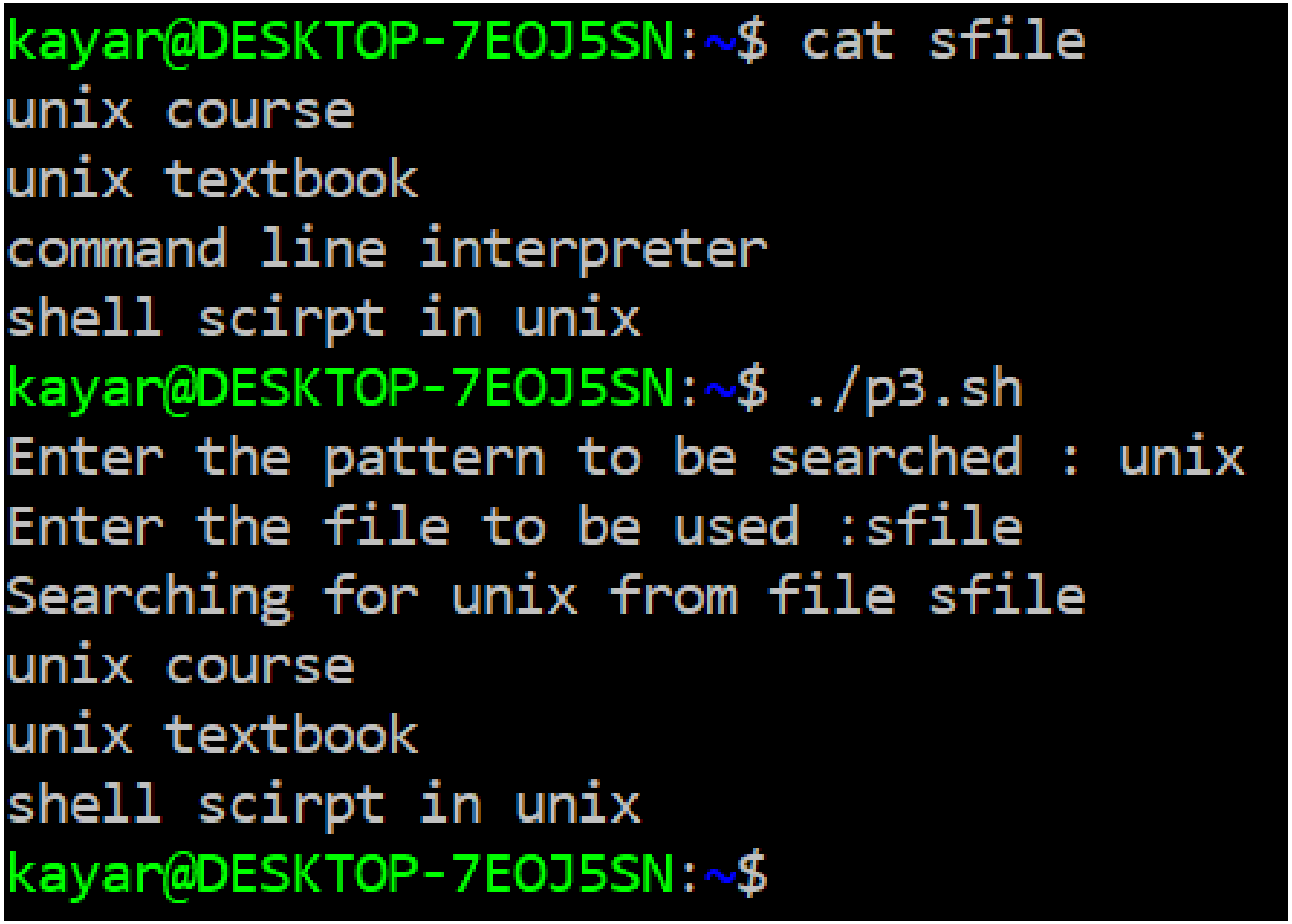
- The **grep filter searches a file for a particular pattern of characters**, and displays all lines that contain that pattern.
- The pattern that is searched in the file is referred to as the regular expression
- (grep stands for globally search for regular expression and print out)

```
grep [options] pattern [files]
```

read : Making scripts interactive

```
#!/bin/sh
echo “Enter the pattern to be searched:\c”
read pname
echo “Enter the file to be used :\c”
read flname
echo “Searching for $pname from file $flname”
grep “$pname” $flname
```

Output



```
kayar@DESKTOP-7EOJ5SN:~$ cat sfile
unix course
unix textbook
command line interpreter
shell scirpt in unix
kayar@DESKTOP-7EOJ5SN:~$ ./p3.sh
Enter the pattern to be searched : unix
Enter the file to be used :sfile
Searching for unix from file sfile
unix course
unix textbook
shell scirpt in unix
kayar@DESKTOP-7EOJ5SN:~$
```

Using Command Line Arguments

- Shell scripts accept arguments from command line
- When arguments are specified with a shell script, they are assigned to certain special “variable” – **Positional Parameters**

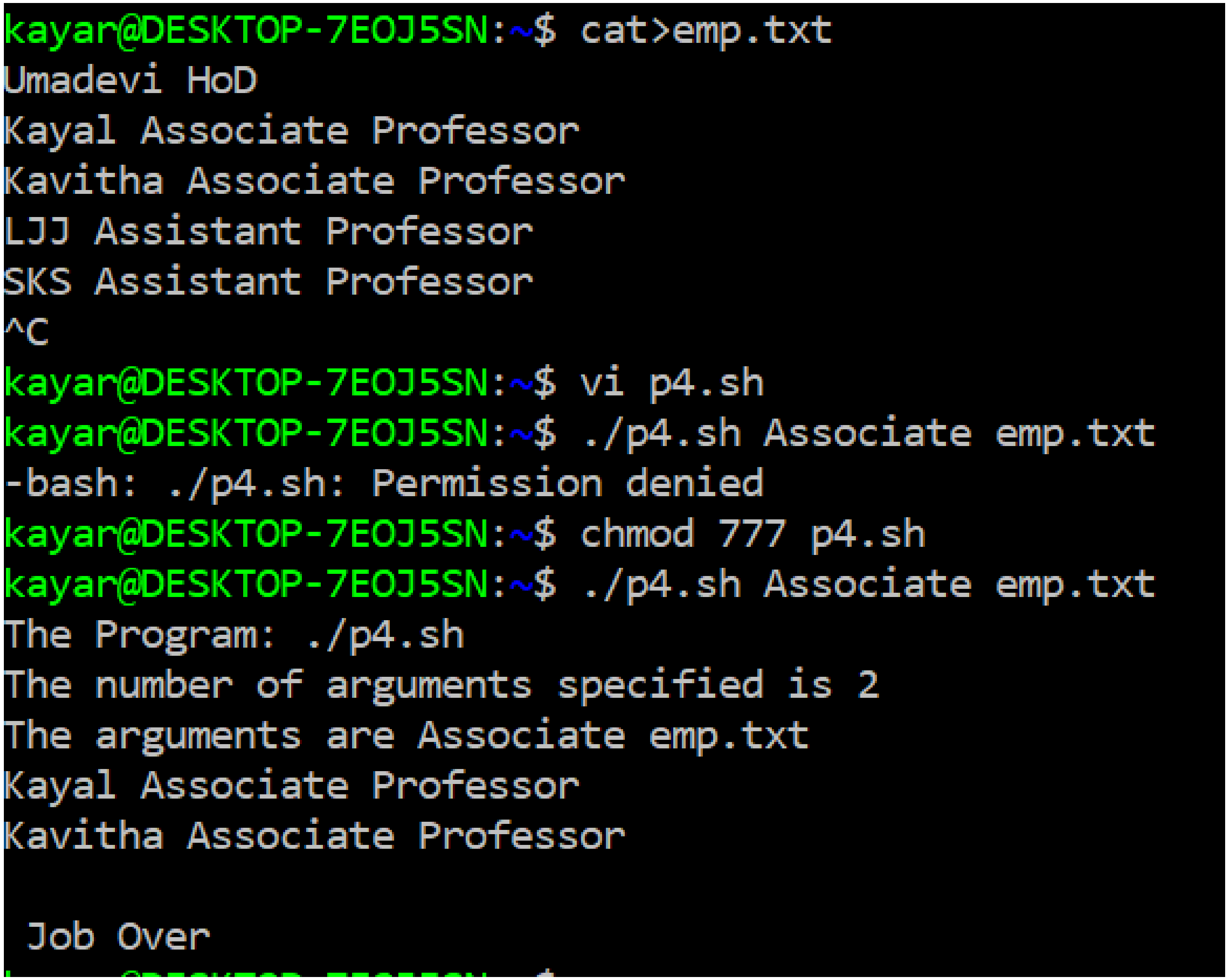
Special Parameters Used by Shell

Shell Parameter	Description
$1, $2	Positional parameters representing command line arguments
$#	Number of arguments specified in command line
$0	Name of executed Command
$*	Complete set of positional parameters as a single string
“$@”	Each quoted string treated as a separate argument (recommended over $*)
$?	Exit status of last command
$$	PID of the current shell
$!	PID of the last background

Example

```
#!/bin/sh
echo “Program : $0
The number of arguments specified is $#
The arguments are $*”
grep “$1” $2
echo “\n Job Over”
```

Output



```
kayar@DESKTOP-7EOJ5SN:~$ cat>emp.txt
Umadevi HoD
Kayal Associate Professor
Kavitha Associate Professor
LJJ Assistant Professor
SKS Assistant Professor
^C
kayar@DESKTOP-7EOJ5SN:~$ vi p4.sh
kayar@DESKTOP-7EOJ5SN:~$ ./p4.sh Associate emp.txt
-bash: ./p4.sh: Permission denied
kayar@DESKTOP-7EOJ5SN:~$ chmod 777 p4.sh
kayar@DESKTOP-7EOJ5SN:~$ ./p4.sh Associate emp.txt
The Program: ./p4.sh
The number of arguments specified is 2
The arguments are Associate emp.txt
Kayal Associate Professor
Kavitha Associate Professor

 Job Over
```

exit AND EXIT status of Command

- exit command is used to terminate the program
- The command is generally tun with a numeric arguments

  exit 0 ---- Used when everything went fine

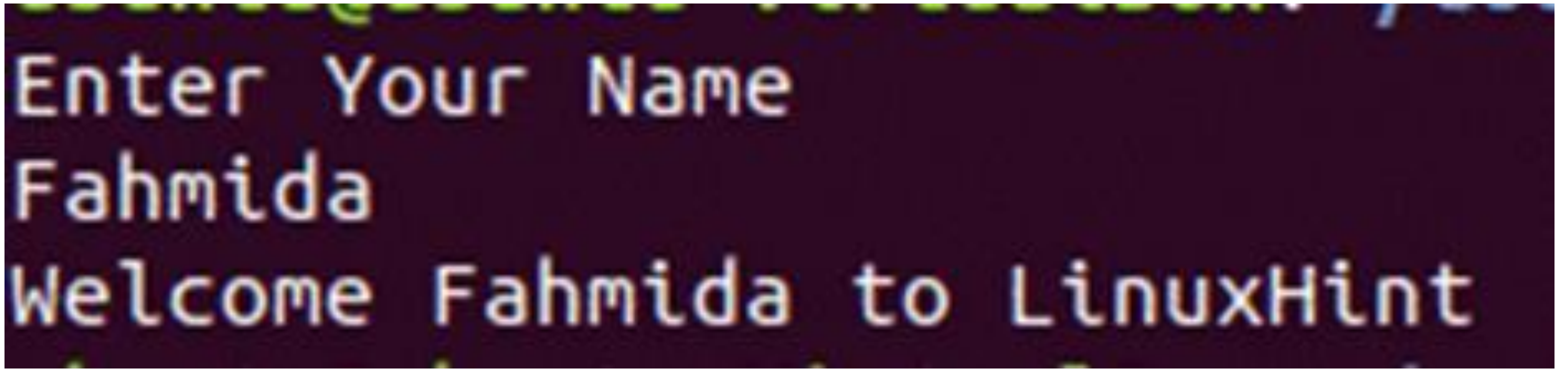
  exit 1 ---- Used when something went wrong
- Every command returns an exit status to the caller

exit AND EXIT status of Command

- The shell offers a variable ($?) and command (test) that evaluates a command's exit status
- The parameter $? stores the exit status of the last command
- It has the value 0 if the command succeeds and a non-zero value if its fails.

```
echo $?
```

Create a file named ‘**user_input.sh**’ and add the following script for taking input from the user. Here, one string value will be taken from the user and display the value by combining other string value.



```
Enter Your Name
Fahmida
Welcome Fahmida to LinuxHint
```

Read filename from user and delete it without warring message

Program

```
#!/bin/sh
  echo -n “Enter name of file to
  delete: ”
  read file
  echo “Type 'y' to remove it, 'n'
  to change your mind ... ”
  rm -i $file
  echo "That was YOUR decision!"
```

Arithmetic Operators

- The expr **command** in Unix evaluates a given expression and displays its corresponding output
- **expr supports the following operators:**
  - **arithmetic operators: +,-,*,/,%**
  - **comparison operators: <, <=, ==, !=, >=, >**
  - **boolean/logical operators: &, |**
  - **parentheses: (,)**
  - **precedence is the same as C, Java**

Example

```
#!/bin/sh
count=5
count=`expr $count + 1 `
echo $count
```

Write script to add two number