1→

Requirements:

When you run the script, display all file information from current working directory

|  |
| --- |
| #!/bin/bash |
|  | # |
|  | # Script for printing all the files related information in present working directory |
|  | # |
|  |  |
|  | ls -la |

2→

Pattern

1

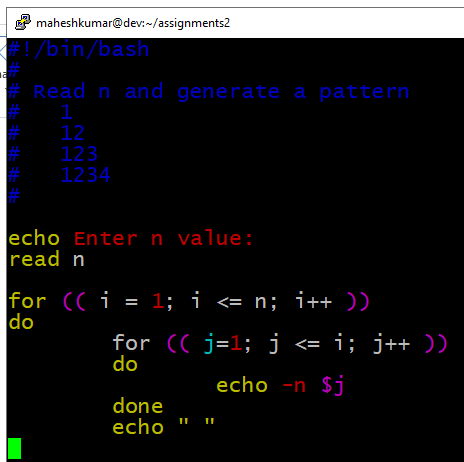
1 2

1 2 3

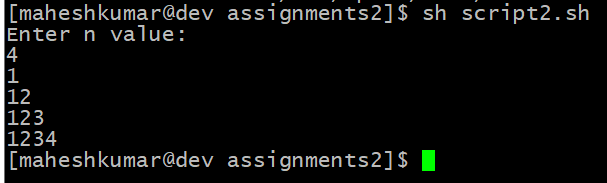
1 2 3 4

Create a pattern as mentioned above:

|  |
| --- |
| #!/bin/bash |
|  | # |
|  | # Read n and generate a pattern |
|  | # 1 |
|  | # 12 |
|  | # 123 |
|  | # 1234 |
|  | # |
|  |  |
|  | echo Enter n value: |
|  | read n |
|  |  |
|  | for (( i = 1; i <= n; i++ )) |
|  | do |
|  | for (( j=1; j <= i; j++ )) |
|  | do |
|  | echo -n $j |
|  | done |
|  | echo " " |
|  | done |



The output:



3→

Pattern

1

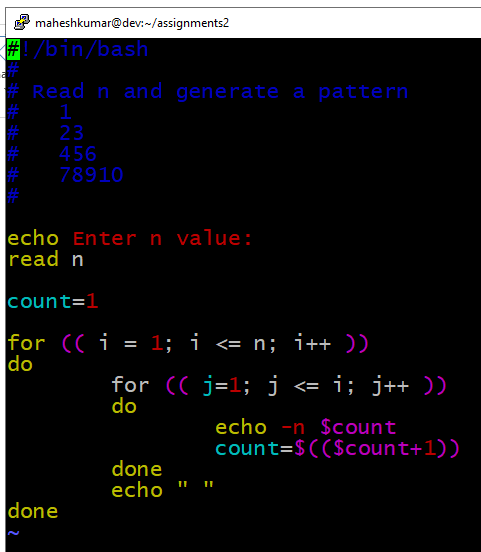
2 3

4 5 6

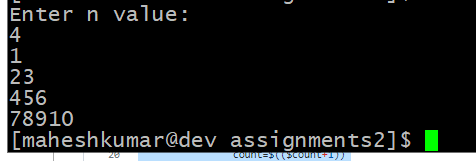
7 8 9 10

Create a pattern as mentioned above:

|  |
| --- |
| #!/bin/bash |
|  | # |
|  | # Read n and generate a pattern |
|  | # 1 |
|  | # 23 |
|  | # 456 |
|  | # 78910 |
|  | # |
|  |  |
|  | echo Enter n value: |
|  | read n |
|  |  |
|  | count=1 |
|  |  |
|  | for (( i = 1; i <= n; i++ )) |
|  | do |
|  | for (( j=1; j <= i; j++ )) |
|  | do |
|  | echo -n $count |
|  | count=$(($count+1)) |
|  | done |
|  | echo " " |
|  | done |



The output:



5

5→

Prerequisites:

How to add real numbers in script.

How to use piping in commands.

Objective:

To understand working of piping.

To learn arithmetic operations in shell script

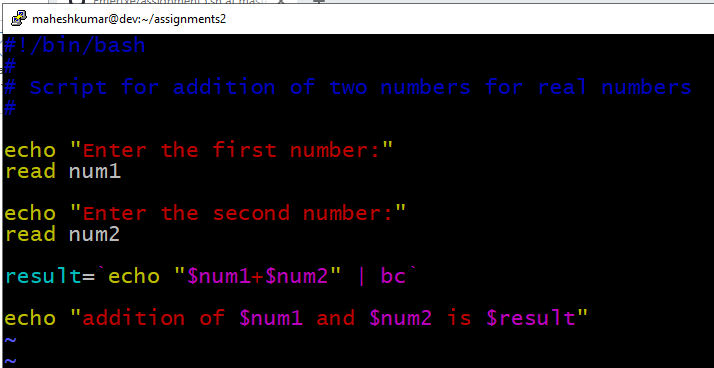
Requirements:

Ask user to enter two numbers

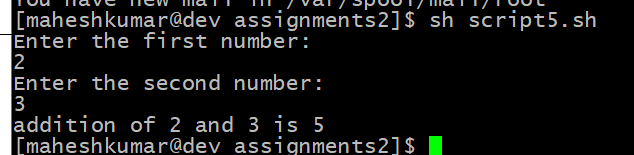
User can enter real numbers also

Use bc command and piping to do

|  |
| --- |
| #!/bin/bash |
|  | # |
|  | # Script for addition of two numbers for real numbers |
|  | # |
|  |  |
|  | echo "Enter the first number:" |
|  | read num1 |
|  |  |
|  | echo "Enter the second number:" |
|  | read num2 |
|  |  |
|  | result=`echo "$num1+$num2" | bc` |
|  |  |
|  | echo "addition of $num1 and $num2 is $result" |



The output:



6→

Pattern

1

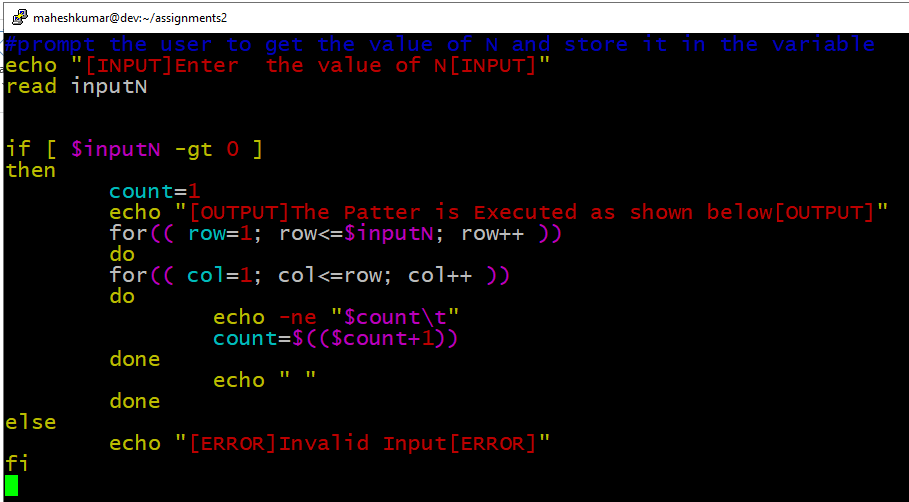
2 3

4 5 6

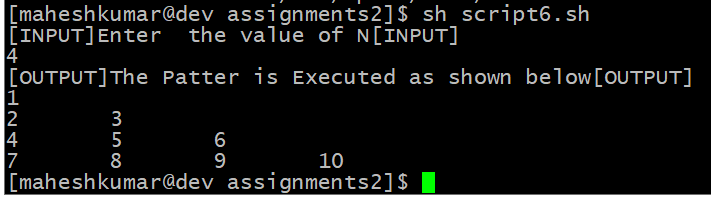
7 8 9 10

Create a pattern as mentioned above:

|  |
| --- |
| #prompt the user to get the value of N and store it in the variable |
|  | echo "[INPUT]Enter the value of N[INPUT]" |
|  | read inputN |
|  |  |
|  |  |
|  | if [ $inputN -gt 0 ] |
|  | then |
|  | count=1 |
|  | echo "[OUTPUT]The Patter is Executed as shown below[OUTPUT]" |
|  | for(( row=1; row<=$inputN; row++ )) |
|  | do |
|  | for(( col=1; col<=row; col++ )) |
|  | do |
|  | echo -ne "$count\t" |
|  | count=$(($count+1)) |
|  | done |
|  | echo " " |
|  | done |
|  | else |
|  | echo "[ERROR]Invalid Input[ERROR]" |
|  | fi |



The output:



7→

Prerequisites:

How to run a loops in shell scripts.

How to execute a bash script.

How to change execute permission of a file..

Requirements:

Using command-line pass n arguments.

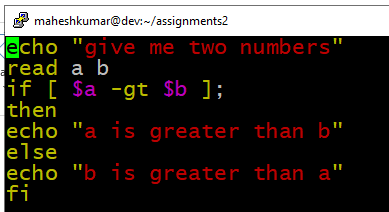
Compare all these arguments and print the largest

value

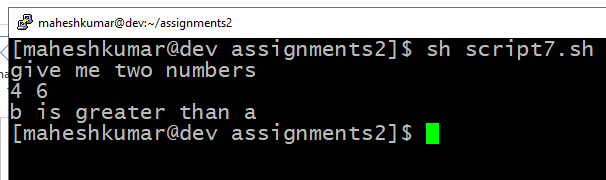
Print error in-case no arguments.

Number of arguments can vary every time.

|  |
| --- |
| Echo "give me two numbers" |
|  | read a b |
|  | if [ $a -gt $b ]; |
|  | then |
|  | echo "a is greater than b" |
|  | else |
|  | echo "b is greater than a" |
|  | fi |



The output:



8→

Requirements:

Read an multi-digit number from user and reverse

the number.

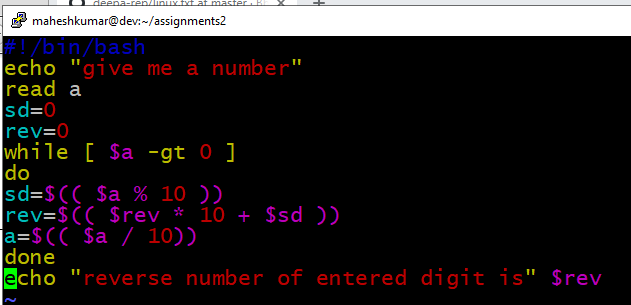
Its not just printing in reverse order

You have to extract each digit and convert to

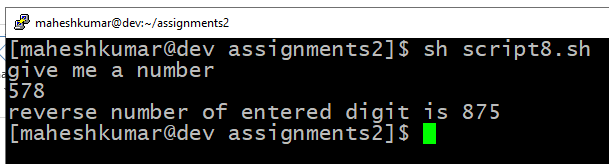
reverse.

When ‘0’ comes as last digit, discard while reversing

|  |
| --- |
| #!/bin/bash |
|  | echo "give me a number" |
|  | read a |
|  | sd=0 |
|  | rev=0 |
|  | while [ $a -gt 0 ] |
|  | do |
|  | sd=$(( $a % 10 )) |
|  | rev=$(( $rev \* 10 + $sd )) |
|  | a=$(( $a / 10)) |
|  | done |
|  | echo "reverse number of entered digit is" $rev |



The output:



10→

Requirements:

Read a string from user, must end with a operator

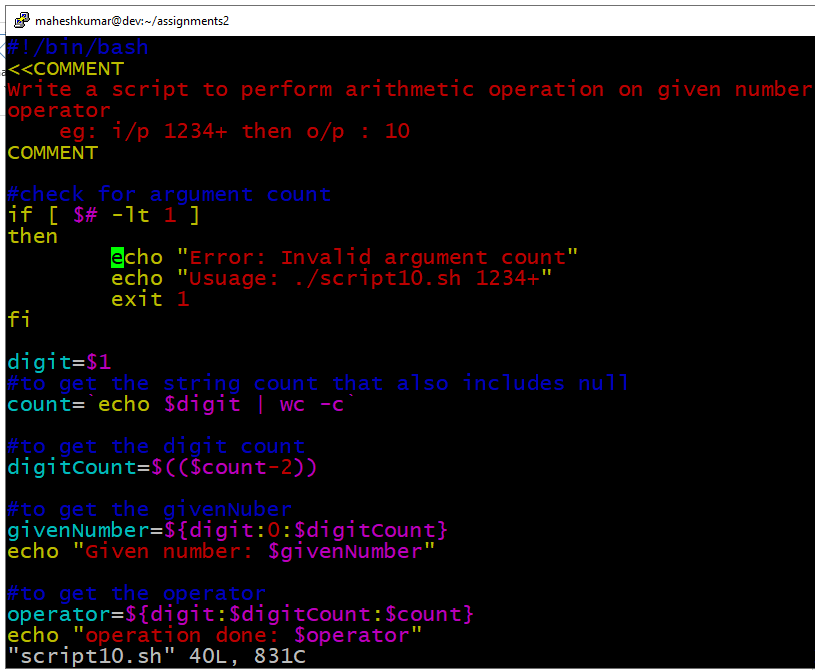
symbol.

Number can be any length but must end with an

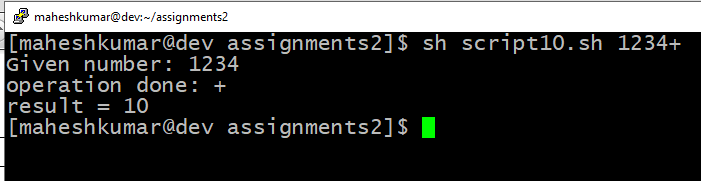
operator character

Always do left to right operations.

If 8312 – passed do 8-3-1-2 = 2



Output:



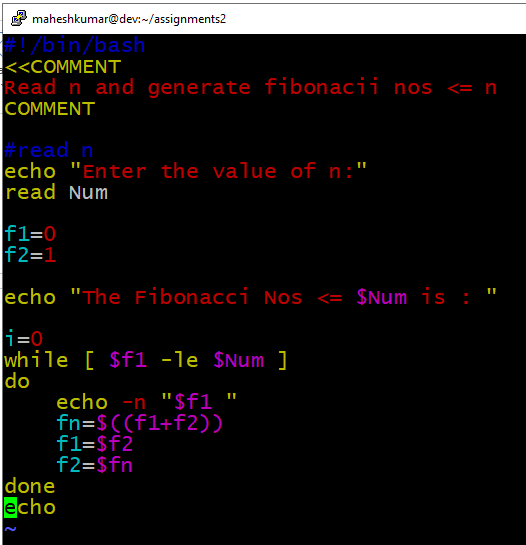
11→

Requirements:

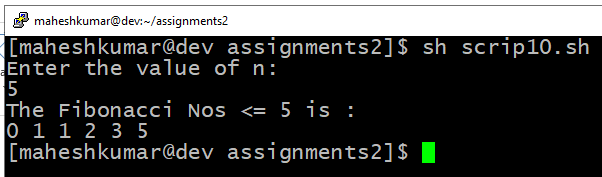
Knowledge about Fibonacci series.

Remember n is not number of elements to print

Its the boundary of elements to print.



The output:



12→

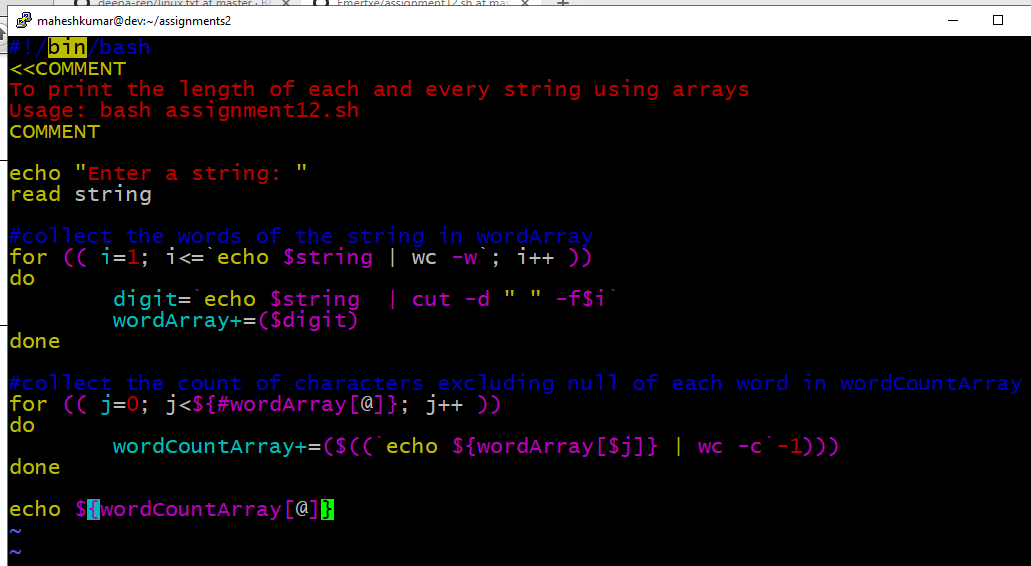
Requirements:

Pass some names or strings from command-line.

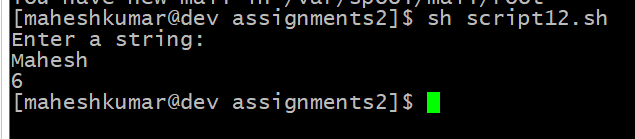
Print all the string lengths one-by-one.

Number of argument may vary.

|  |
| --- |
| #!/bin/bash |
|  | <<COMMENT |
|  | To print the length of each and every string using arrays |
|  | Usage: bash assignment12.sh |
|  | COMMENT |
|  |  |
|  | echo "Enter a string: " |
|  | read string |
|  |  |
|  | #collect the words of the string in wordArray |
|  | for (( i=1; i<=`echo $string | wc -w`; i++ )) |
|  | do |
|  | digit=`echo $string | cut -d " " -f$i` |
|  | wordArray+=($digit) |
|  | done |
|  |  |
|  | #collect the count of characters excluding null of each word in wordCountArray |
|  | for (( j=0; j<${#wordArray[@]}; j++ )) |
|  | do |
|  | wordCountArray+=($((`echo ${wordArray[$j]} | wc -c`-1))) |
|  | done |
|  |  |
|  | echo ${wordCountArray[@]} |



The output:



13→

Requirements:

To print a black box echo -e -n “\\\\e[40m” ” “

To print a white box echo -e -n “\\\\e[47m” ” “

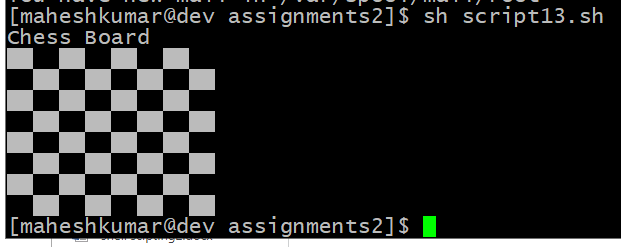
Call the commands in a loop.

After 8 columns make to normal color.

To make it normal echo -e -n “\\\\e[0m” ” “

|  |
| --- |
| #!/bin/bash |
|  | <<COMMENT |
|  | Chess board |
|  | COMMENT |
|  |  |
|  | echo "Chess Board" |
|  |  |
|  | for (( i=1; i<=8; i++)) |
|  | do |
|  | for (( j=1; j<=8; j++)) |
|  | do |
|  | total=$(($i+$j)) |
|  | temp=$(($total%2)) |
|  | # for alternative blocks |
|  | if [ $temp -eq 0 ] |
|  | then |
|  | echo -e -n "\033[47m" " " #white |
|  | else |
|  | echo -e -n "\033[40m" " " #black |
|  | fi |
|  | done |
|  | echo -e -n "\033[0m" " " |
|  | echo ' ' |
|  | done |





14→

Requirements:

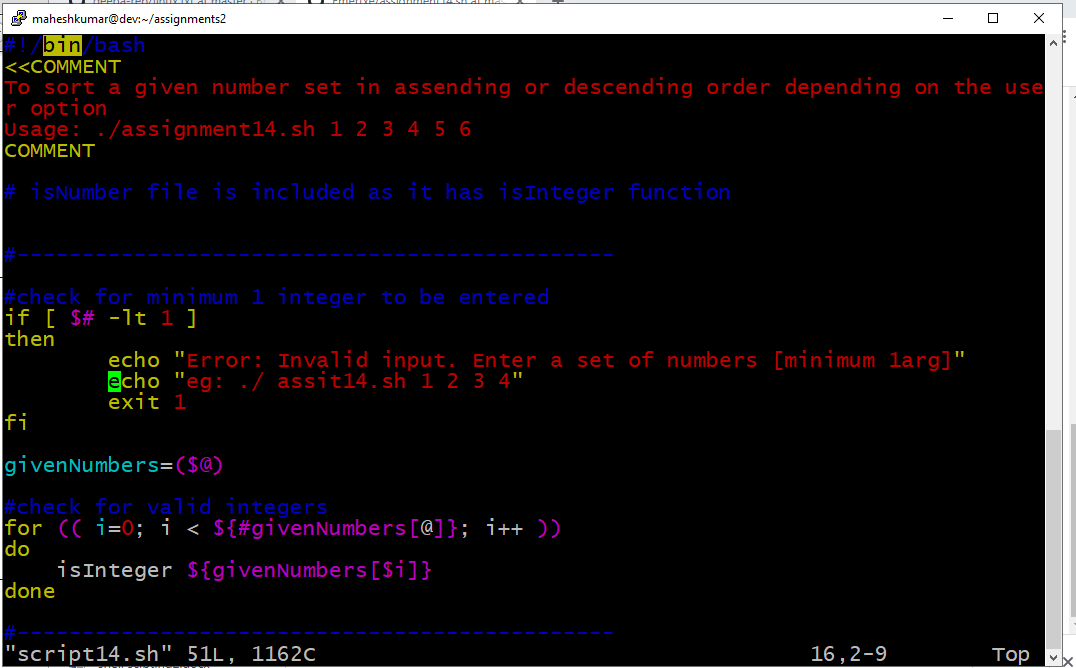
Pass numbers through command-line arguments.

Provide a menu for user to choose ascending or

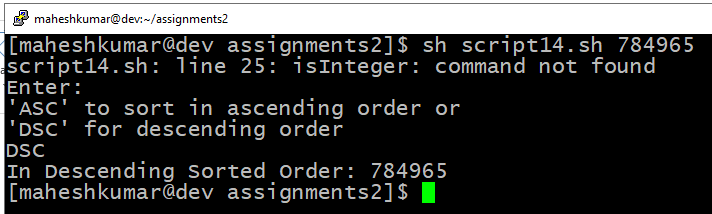
descending.

Show sorted array according to user choice.

|  |
| --- |
| #!/bin/bash |
|  | <<COMMENT |
|  | To sort a given number set in assending or descending order depending on the user option |
|  | Usage: ./assignment14.sh 1 2 3 4 5 6 |
|  |  |
|  | COMMENT |
|  | #---------------------------------------------- |
|  |  |
|  | #check for minimum 1 integer to be entered |
|  | if [ $# -lt 1 ] |
|  | then |
|  | echo "Error: Invalid input. Enter a set of numbers [minimum 1arg]" |
|  | echo "eg: ./ assignment14.sh 1 2 3 4" |
|  | exit 1 |
|  | fi |
|  |  |
|  | givenNumbers=($@) |
|  |  |
|  | #check for valid integers |
|  | for (( i=0; i < ${#givenNumbers[@]}; i++ )) |
|  | do |
|  | isInteger ${givenNumbers[$i]} |
|  | done |
|  |  |
|  | #---------------------------------------------- |
|  |  |
|  | echo "Enter: |
|  | 'ASC' to sort in ascending order or |
|  | 'DSC' for descending order" |
|  | read order |
|  |  |
|  | case $order in |
|  | ASC) #sort assendingly |
|  | sortedArray+=(`for (( i=0; i < ${#givenNumbers[@]}; i++ ));do |
|  | echo ${givenNumbers[$i]};done | sort -g`) |
|  | echo In Ascending Sorted Order: ${sortedArray[@]} |
|  | ;; |
|  | DSC) #sort descendingly |
|  | sortedArray+=(`for (( i=0; i < ${#givenNumbers[@]}; i++ ));do |
|  | echo ${givenNumbers[$i]};done | sort -rg`) |
|  | echo In Descending Sorted Order: ${sortedArray[@]} |
|  | ;; |
|  | \*) |
|  | echo "Error: Invalid order entry" |
|  | exit 1 |
|  | ;; |
|  | esac |

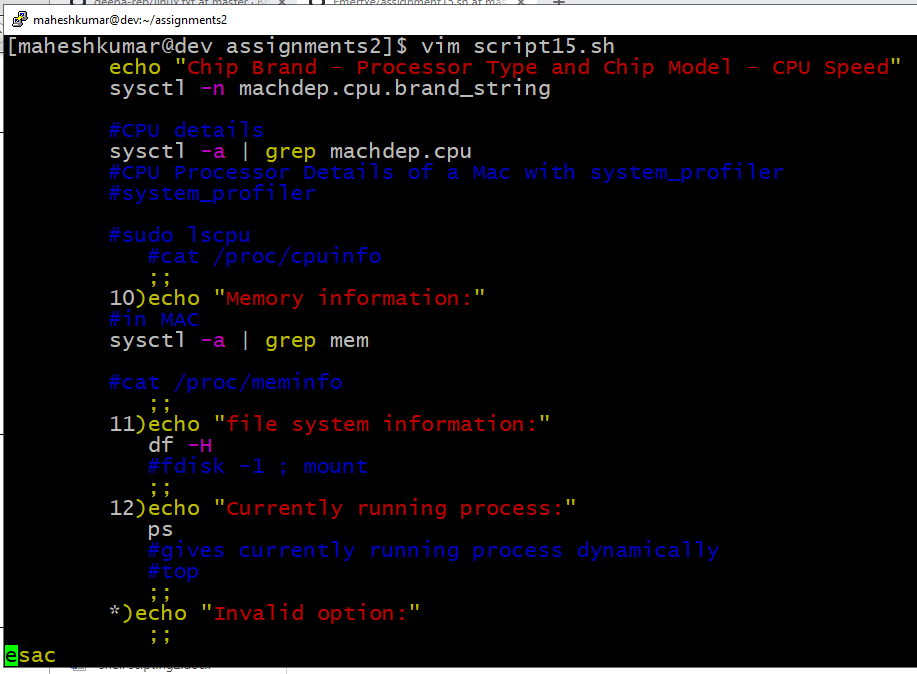


The output:

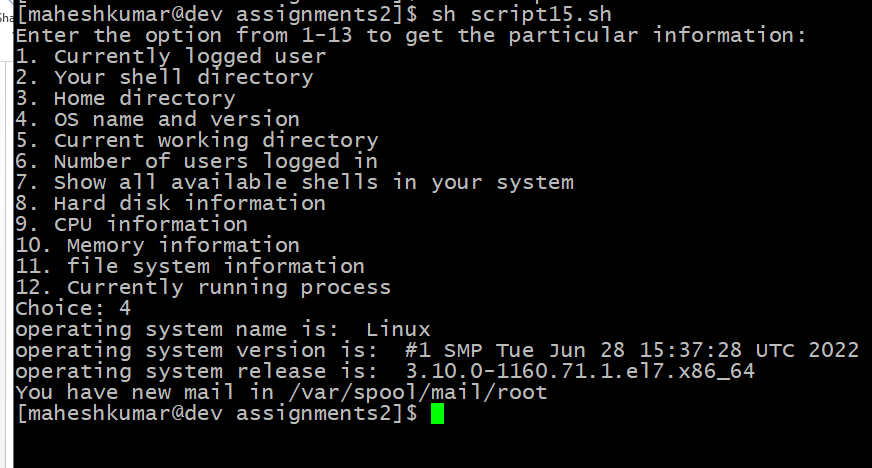


15→

|  |
| --- |
| #!/bin/bash |
|  | <<COMMENT |
|  | Print Informations: |
|  | 1. Currently logged user |
|  | 2. Your shell directory |
|  | 3. Home directory |
|  | 4. OS name and version |
|  | 5. Current working directory |
|  | 6. Number of users logged in |
|  | 7. Show all available shells in your system |
|  | 8. Hard disk information |
|  | 9. CPU information |
|  | 10. Memory information |
|  | 11. file system information |
|  | 12. Currently running process |
|  | COMMENT |
|  |  |
|  | echo "Enter the option from 1-13 to get the particular information: |
|  | 1. Currently logged user |
|  | 2. Your shell directory |
|  | 3. Home directory |
|  | 4. OS name and version |
|  | 5. Current working directory |
|  | 6. Number of users logged in |
|  | 7. Show all available shells in your system |
|  | 8. Hard disk information |
|  | 9. CPU information |
|  | 10. Memory information |
|  | 11. file system information |
|  | 12. Currently running process" |
|  |  |
|  | echo -n "Choice: " ; read option |
|  |  |
|  | case $option in |
|  | 1) # prints the names of the users currently logged in to the current host without showing any information about source, login time, or any other relevant information |
|  | echo "Currently logged user:"; users; |
|  | ;; |
|  | 2) #gives shell name relative to root |
|  | echo "Your shell directory: $SHELL" |
|  | ;; |
|  | 3) echo "Home directory: $HOME" |
|  | ;; |
|  | 4) echo -n "operating system name is: "; uname -s ; #uname |
|  | echo -n "operating system version is: "; uname -v |
|  | echo -n "operating system release is: "; uname -r |
|  | #echo -n network hostname: ; uname -n |
|  | #echo -n hardware name: ; uname -m |
|  | ;; |
|  | 5) echo "Current working directory: $PWD" |
|  | ;; |
|  | 6) echo -n "Number of users logged in is: " |
|  | who -u | wc -l |
|  | #who -q |
|  | ;; |
|  | 7) cat /etc/shells | grep -v "#" |
|  | ;; |
|  | 8) echo "Hard disk information:" |
|  | diskutil list #in MAC |
|  | #for further detailed information of each physical disk |
|  | diskutil info disk0 |
|  | diskutil info disk1 |
|  |  |
|  | #sudo lshw -short |
|  | #sudo hdparm -I /dev/sda |
|  | ;; |
|  | 9) echo "CPU information:" |
|  | #in MAC |
|  | echo "Chip Brand – Processor Type and Chip Model – CPU Speed" |
|  | sysctl -n machdep.cpu.brand\_string |
|  |  |
|  | #CPU details |
|  | sysctl -a | grep machdep.cpu |
|  | #CPU Processor Details of a Mac with system\_profiler |
|  | #system\_profiler |
|  |  |
|  | #sudo lscpu |
|  | #cat /proc/cpuinfo |
|  | ;; |
|  | 10)echo "Memory information:" |
|  | #in MAC |
|  | sysctl -a | grep mem |
|  |  |
|  | #cat /proc/meminfo |
|  | ;; |
|  | 11)echo "file system information:" |
|  | df -H |
|  | #fdisk -1 ; mount |
|  | ;; |
|  | 12)echo "Currently running process:" |
|  | ps |
|  | #gives currently running process dynamically |
|  | #top |
|  | ;; |
|  | \*)echo "Invalid option:" |
|  | ;; |
|  | esac |



The output:



16→

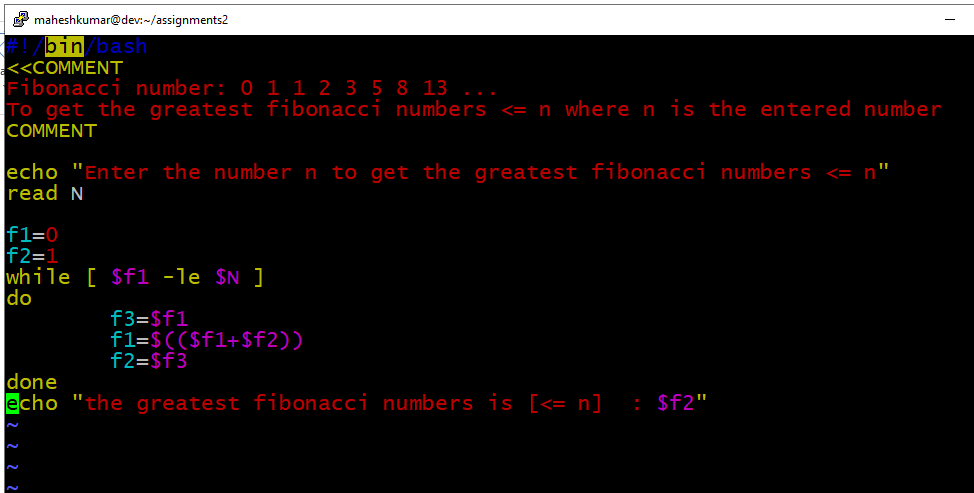
Knowledge about Fibonacci series.

Requirements:

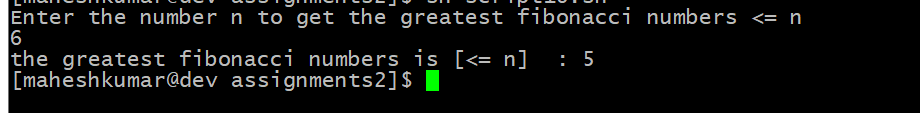
Remember n is not nth number of series.

Its the greatest element to print.

|  |
| --- |
| #!/bin/bash |
|  | <<COMMENT |
|  | Fibonacci number: 0 1 1 2 3 5 8 13 ... |
|  | To get the greatest fibonacci numbers <= n where n is the entered number |
|  | COMMENT |
|  |  |
|  | echo "Enter the number n to get the greatest fibonacci numbers <= n" |
|  | read N |
|  |  |
|  | f1=0 |
|  | f2=1 |
|  | while [ $f1 -le $N ] |
|  | do |
|  | f3=$f1 |
|  | f1=$(($f1+$f2)) |
|  | f2=$f3 |
|  | done |
|  | echo "the greatest fibonacci numbers is [<= n] : $f2" |



The output:



17→

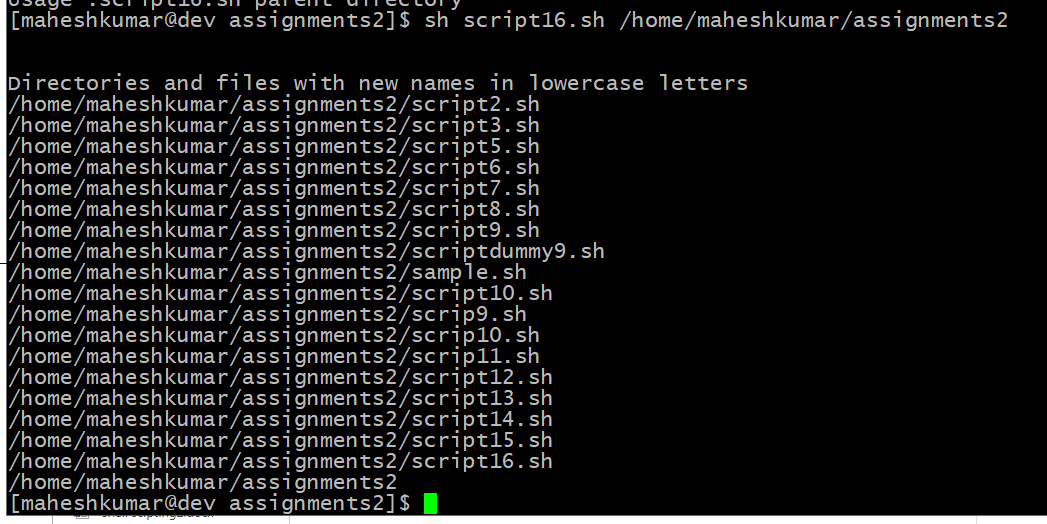
Rename all directories from current directories to

uppercase.

Digits and other symbols should remain same.

|  |
| --- |
|  |
| #!/bin/bash |
|  | #print usage |
|  | if [ -z $1 ];then |
|  | echo "Usage :$(basename $0) parent-directory" |
|  | exit 1 |
|  | fi |
|  |  |
|  | #process all subdirectories and files in parent directory |
|  | all="$(find $1 -depth)" |
|  |  |
|  |  |
|  |  |
|  | for name in ${all}; do |
|  | #set new name in lower case for files and directories |
|  | new\_name="$(dirname "${name}")/$(basename "${name}" | tr '[A-Z]' '[a-z]')" |
|  |  |
|  | #check if new name already exists |
|  | if [ "${name}" != "${new\_name}" ]; then |
|  | [ ! -e "${new\_name}" ] && mv -T "${name}" "${new\_name}"; echo "${name} was renamed to ${new\_name}" || echo "${name} wasn't renamed!" |
|  | fi |
|  | done |
|  |  |
|  | echo |
|  | echo |
|  | #list directories and file new names in lowercase |
|  | echo "Directories and files with new names in lowercase letters" |
|  | find $(echo $1 | tr 'A-Z' 'a-z') -depth |
|  |  |
|  | exit 0 |
|  |  |
|  |  |
|  |  |
|  |  |
|  | Orrrr |
|  |  |
|  |  |
|  |  |
|  |  |
|  | for f in `find`; do mv -v "$f" "`echo $f | tr '[A-Z]' '[a-z]'`"; done |
|  |  |

The output:



19→

Requirements:

Aim of this project is to rename all files in one

directory with a common name and indexing.

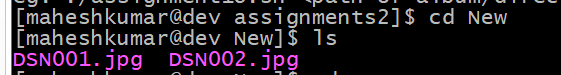
Usually when we takes pics in camera or mobile

default names are like DSN001.jpg, DSN002.jpg

These files need to be renamed by user given prefix

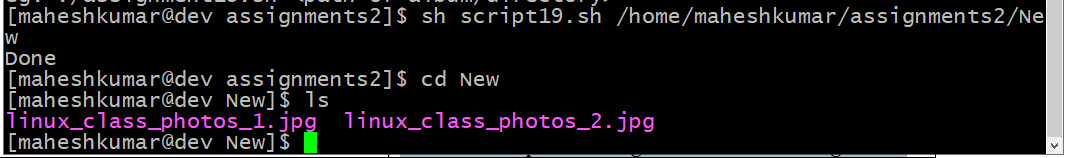
name

Prefix name pass through command-line argument



|  |
| --- |
| #!/bin/bash |
|  | <<COMMENT |
|  | Given the album name and corresponding directory, rename them properly by inserting index numbers |
|  | Usage: eg: given file numbers DSN001.jpg, DSN002.jpg as |
|  | linux\_class\_photos\_001.jpg, linux\_class\_photos\_002.jpg |
|  | COMMENT |
|  |  |
|  | #------------------------------------------ |
|  |  |
|  | #check for argument count |
|  | if [ $# -ne 1 ] |
|  | then |
|  | echo "Error: No directory/album name is given" |
|  | echo "Usage: Please provide the album/directory path:" |
|  | echo "eg: ./assignment18.sh <path of album/directory>" |
|  | exit 1 |
|  | fi |
|  |  |
|  | path=$1 |
|  |  |
|  | #check if the given file/directory path exists or not |
|  | if [ ! -e $path ] |
|  | then |
|  | echo "Error: No such file/directory exists" |
|  | exit 1 |
|  | fi |
|  |  |
|  | #---------------------------------------- |
|  | cd $1 |
|  |  |
|  |  |
|  | count=`ls | wc -w` |
|  |  |
|  | for (( j=1; j <= $count; j++ )) |
|  | do |
|  | newName=DSN00${j} |
|  | mv ${newName}.jpg linux\_class\_photos\_${j}.jpg |
|  | #mv linux\_class\_photos\_${j}.jpg DSN00${j}.jpg |
|  | done |
|  | echo Done |

The output:



20→

Requirements:

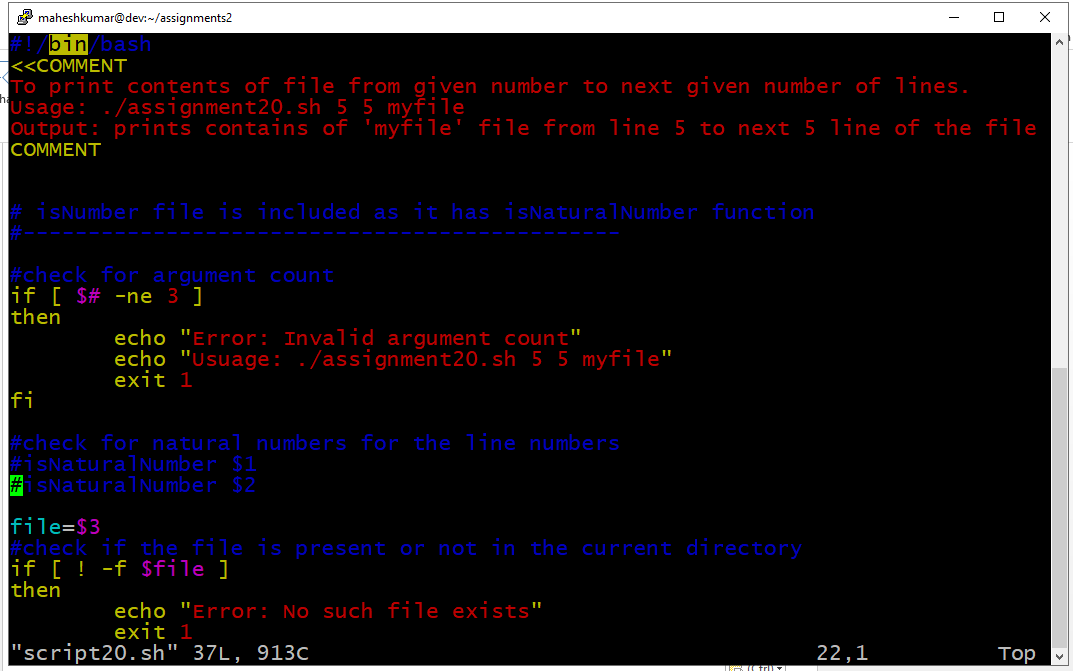
Pass three command-line arguments

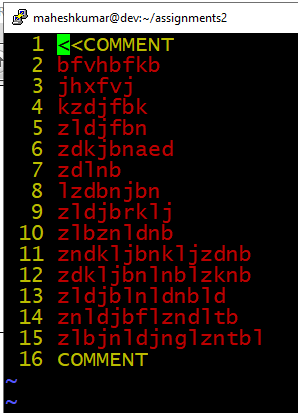
1- starting line number

2-number of lines and filename

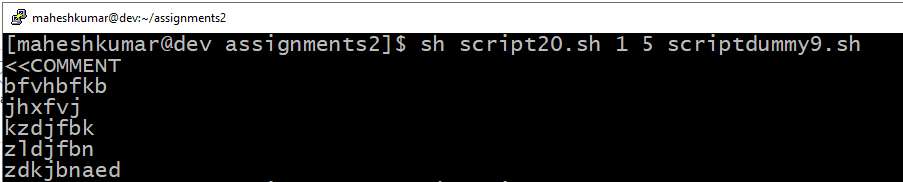
Script will print n lines from given starting line

|  |
| --- |
| #!/bin/bash |
|  | <<COMMENT |
|  | To print contents of file from given number to next given number of lines. |
|  | Usage: ./assignment20.sh 5 5 myfile |
|  | Output: prints contains of 'myfile' file from line 5 to next 5 line of the file |
|  | COMMENT |
|  |  |
|  |  |
|  | # isNumber file is included as it has isNaturalNumber function |
|  | . ./isNumber.sh |
|  |  |
|  | #---------------------------------------------- |
|  |  |
|  | #check for argument count |
|  | if [ $# -ne 3 ] |
|  | then |
|  | echo "Error: Invalid argument count" |
|  | echo "Usuage: ./assignment20.sh 5 5 myfile" |
|  | exit 1 |
|  | fi |
|  |  |
|  | #check for natural numbers for the line numbers |
|  | isNaturalNumber $1 |
|  | isNaturalNumber $2 |
|  |  |
|  | file=$3 |
|  | #check if the file is present or not in the current directory |
|  | if [ ! -f $file ] |
|  | then |
|  | echo "Error: No such file exists" |
|  | exit 1 |
|  | fi |
|  |  |
|  | #--------------------------------------------------------- |
|  |  |
|  | #to print contents of file($3) from given number ($1) to next given number of lines ($2) |
|  | var1=$1 |
|  | var2=$(($1+$2)) |
|  | sed -n "${var1},${var2}"p $file |





The output:



21→

Requirements:

The script should run as soon as you log-on to

system

Print greetings based on time as follows.

“Good morning” (5 AM – 12 PM)

“Good noon” (12 PM – 1 PM)

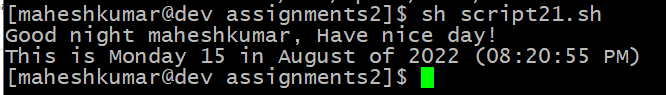
“Good afternoon” (2 PM – 5 PM)

“Good evening” (5PM – 9 PM)

“Good night” (9 PM – 5 AM)

|  |
| --- |
|  |
| #!/bin/bash |
|  |  |
|  | hour=`date +%c | tr -s " " | cut -d " " -f4 | cut -d ":" -f1` |
|  | day=`date +%A` |
|  | mon=`date +%B` |
|  | dte=`date +%d` |
|  | year=`date +%Y` |
|  | tf=`date +%r` |
|  | if [ $hour -ge 5 -a $hour -lt 12 ] |
|  | then |
|  | echo -e "Good morning `whoami`, Have nice day!\nThis is $day $dte in $mon of $year ($tf)" |
|  | elif [ $hour -ge 12 -a $hour -le 13 ] |
|  | then |
|  | echo -e "Good noon `whoami`, Have nice day!\nThis is $day $dte in $mon of $year ($tf)" |
|  | elif [ $hour -ge 14 -a $hour -lt 17 ] |
|  | then |
|  | echo -e "Good afternoon `whoami`, Have nice day!\nThis is $day $dte in $mon of $year ($tf)" |
|  | elif [ $hour -ge 17 -a $hour -lt 21 ] |
|  | then |
|  | echo -e "Good evening `whoami`, Have nice day!\nThis is $day $dte in $mon of $year ($tf)" |
|  | elif [ $hour -ge 21 -o $hour -lt 5 ] |
|  | then |
|  | echo -e "Good night `whoami`, Have nice day!\nThis is $day $dte in $mon of $year ($tf)" |
|  | Fi |
|  |  |

The output:



22→

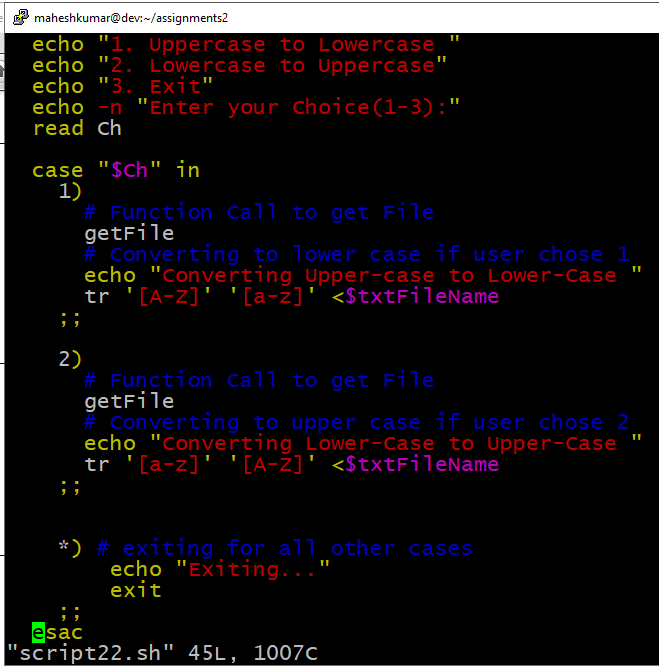
Requirements:

Provide a filename through command-line.

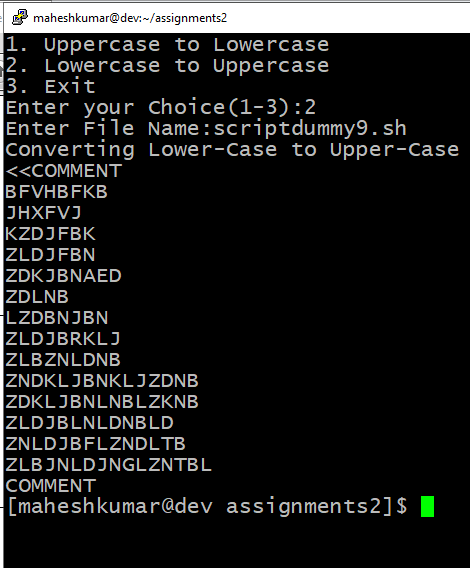
Ask user for conversion Lower to Upper / Upper to

Lower.

|  |
| --- |
| #!/bin/sh |
|  | # shebang to specify that this is an shell script |
|  |  |
|  | # Function to get File |
|  | getFile(){ |
|  | # Reading txtFileName to convert it's content |
|  | echo -n "Enter File Name:" |
|  | read txtFileName |
|  | # Checking if file exist |
|  | if [ ! -f $txtFileName ]; then |
|  | echo "File Name $txtFileName does not exists." |
|  | exit 1 |
|  | fi |
|  | } |
|  |  |
|  | clear |
|  | echo "1. Uppercase to Lowercase " |
|  | echo "2. Lowercase to Uppercase" |
|  | echo "3. Exit" |
|  | echo -n "Enter your Choice(1-3):" |
|  | read Ch |
|  |  |
|  | case "$Ch" in |
|  | 1) |
|  | # Function Call to get File |
|  | getFile |
|  | # Converting to lower case if user chose 1 |
|  | echo "Converting Upper-case to Lower-Case " |
|  | tr '[A-Z]' '[a-z]' <$txtFileName |
|  | ;; |
|  |  |
|  | 2) |
|  | # Function Call to get File |
|  | getFile |
|  | # Converting to upper case if user chose 2 |
|  | echo "Converting Lower-Case to Upper-Case " |
|  | tr '[a-z]' '[A-Z]' <$txtFileName |
|  | ;; |
|  |  |
|  |  |
|  | \*) # exiting for all other cases |
|  | echo "Exiting..." |
|  | exit |
|  | ;; |
|  | esac |



The output:



23→

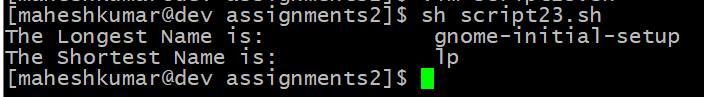
Fetch user-names from the first field in /etc/passwd

file.

Print longest and shortest name.

|  |
| --- |
| #!/bin/bash |
|  |  |
|  | allUsernames=(`cat /etc/passwd | cut -d ':' -f1`) #get all usernames |
|  |  |
|  | smallestName=${allUsernames[0]} #initialize the smallest and largest before looping and setting correct |
|  | largestName=${allUsernames[0]} |
|  |  |
|  | for username in ${allUsernames[@]} #logic to find largest and smallet names |
|  | do |
|  | if [ ${#username} -gt ${#largestName} ] |
|  | then |
|  | largestName=$username |
|  | fi |
|  | if [ ${#username} -lt ${#smallestName} ] |
|  | then |
|  | smallestName=$username |
|  | fi |
|  | done |
|  |  |
|  | echo "The Longest Name is: $largestName" |
|  | echo "The Shortest Name is: $smallestName" |

The output:



24→

Requirements:

Find and delete all .swp files (Temperory vi files).

If command-line directories are passed delete only

from that directories

If no arguments passed delete from entire ~/

directory

If no file present show a message.

#!/bin/bash

<<COMMENT

To find all the swp files from given directory

COMMENT

#check for argument count

if [ $# -ne 1 ]

then

echo "Error: Invalid argument count"

echo "Usuage: ./assignment23.sh <myDirectory>"

exit 1

fi

path=$1

#check if the path is directory or not

if [ ! -d $path ]

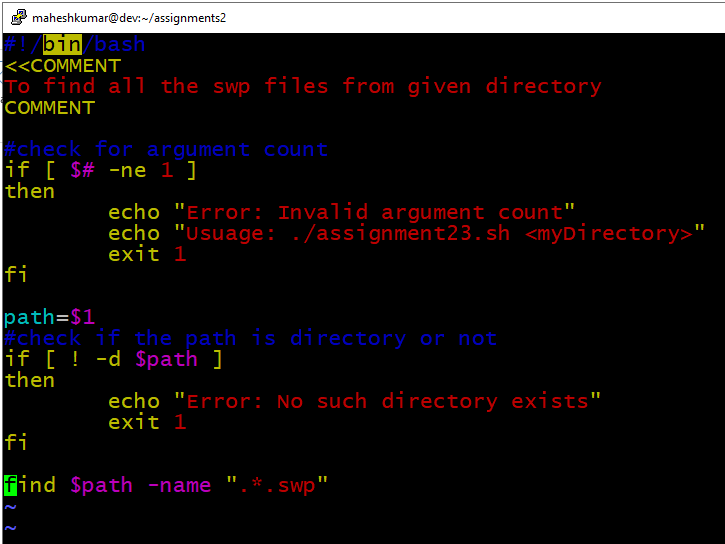
then

echo "Error: No such directory exists"

exit 1

fi

find $path -name ".\*.swp"



25→

Generate random values.

Requirements:

Every time a new password must created.

Password must contains a alpha-numeric and

special characters.

#!/bin/bash

<<COMMENT

To generate random 8-character passwords including alpha numeric characters

NOTE: <<In MAC>>

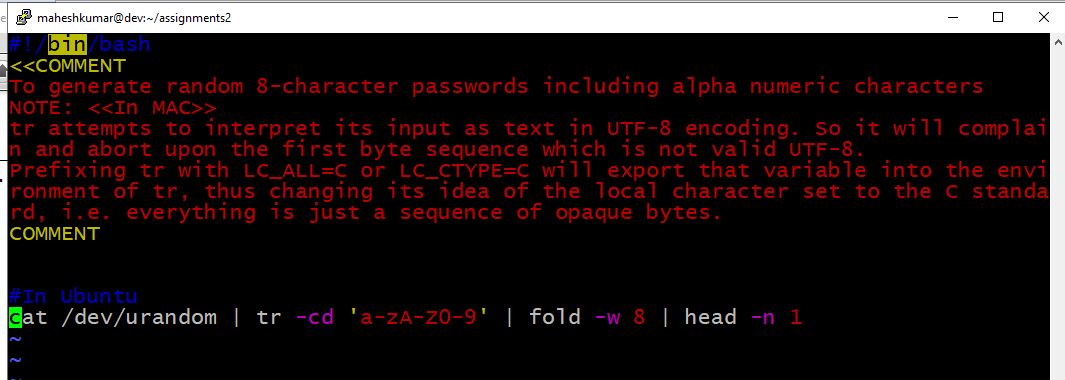
tr attempts to interpret its input as text in UTF-8 encoding. So it will complain and abort upon the first byte sequence which is not valid UTF-8.

Prefixing tr with LC\_ALL=C or LC\_CTYPE=C will export that variable into the environment of tr, thus changing its idea of the local character set to the C standard, i.e. everything is just a sequence of opaque bytes.

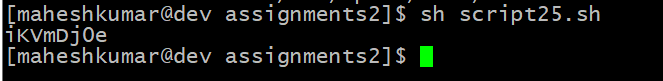
COMMENT

#In Ubuntu

cat /dev/urandom | tr -cd 'a-zA-Z0-9' | fold -w 8 | head -n 1



The output:



26→

Requirements:

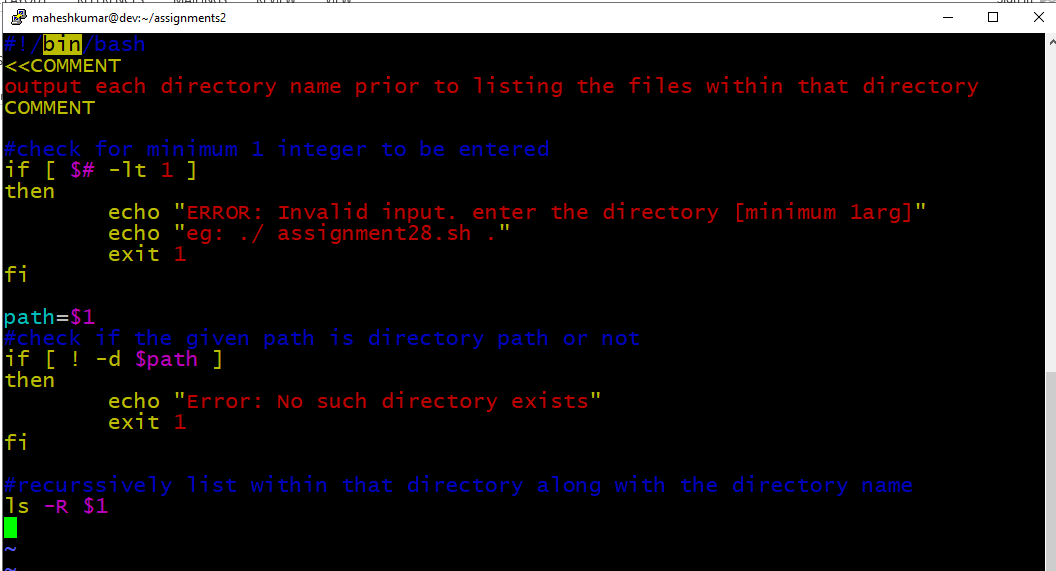
This script will work like a ls command.

Don’t use ls command.

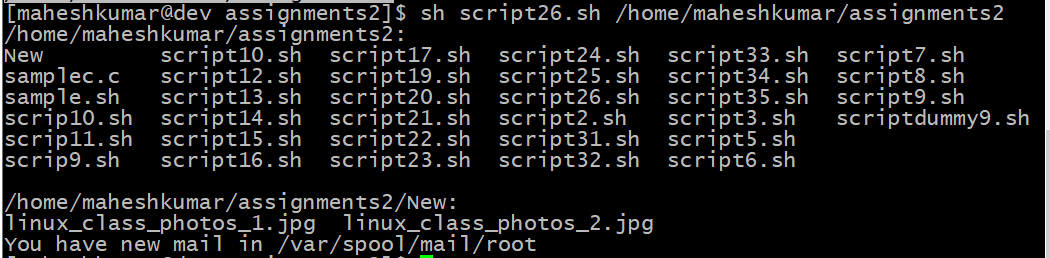
Pass any number of directories through commandline.

If no arguments passed, list current directory

|  |
| --- |
|  |
| #!/bin/bash |
|  | <<COMMENT |
|  | output each directory name prior to listing the files within that directory |
|  | COMMENT |
|  |  |
|  | #check for minimum 1 integer to be entered |
|  | if [ $# -lt 1 ] |
|  | then |
|  | echo "ERROR: Invalid input. enter the directory [minimum 1arg]" |
|  | echo "eg: ./ assignment28.sh ." |
|  | exit 1 |
|  | fi |
|  |  |
|  | path=$1 |
|  | #check if the given path is directory path or not |
|  | if [ ! -d $path ] |
|  | then |
|  | echo "Error: No such directory exists" |
|  | exit 1 |
|  | fi |
|  |  |
|  | #recurssively list within that directory along with the directory name |
|  | ls -R $1 |
|  |  |



Output:



32→

Requirements:

Fetch user-ids from the in /etc/passwd file.

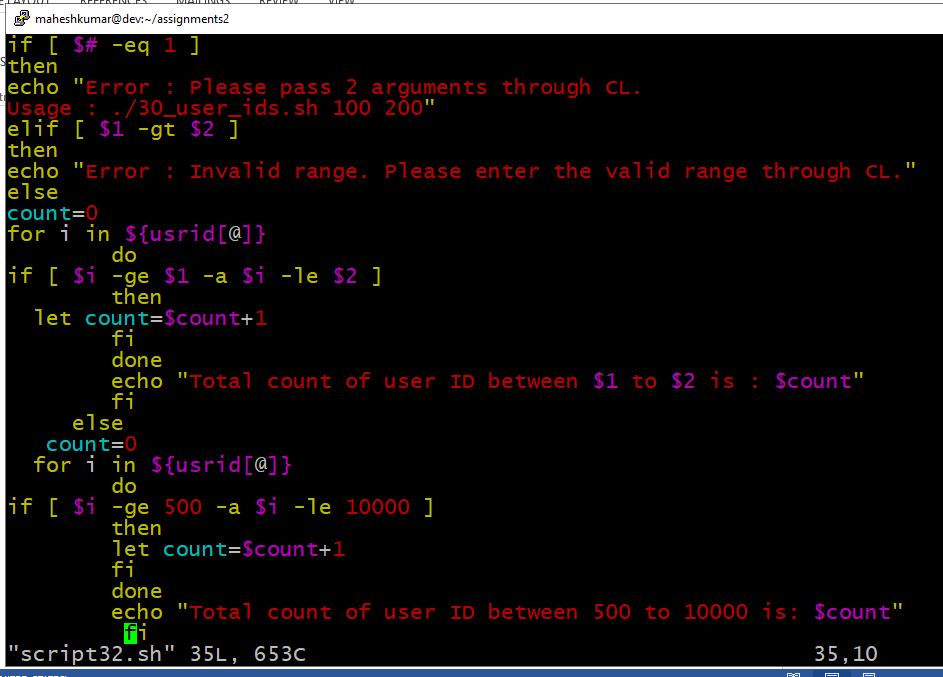
Display only usernames between the range.

User can change the range using command-line

arguments.

Default is 500 – 100000

|  |
| --- |
| #!/bin/bash |
|  |  |
|  |  |
|  | usrid=(`cut -d ":" -f3 /etc/passwd`) |
|  | if [ $# -gt 0 ] |
|  | then |
|  | if [ $# -eq 1 ] |
|  | then |
|  | echo "Error : Please pass 2 arguments through CL. |
|  | Usage : ./30\_user\_ids.sh 100 200" |
|  | elif [ $1 -gt $2 ] |
|  | then |
|  | echo "Error : Invalid range. Please enter the valid range through CL." |
|  | else |
|  | count=0 |
|  | for i in ${usrid[@]} |
|  | do |
|  | if [ $i -ge $1 -a $i -le $2 ] |
|  | then |
|  | let count=$count+1 |
|  | fi |
|  | done |
|  | echo "Total count of user ID between $1 to $2 is : $count" |
|  | fi |
|  | else |
|  | count=0 |
|  | for i in ${usrid[@]} |
|  | do |
|  | if [ $i -ge 500 -a $i -le 10000 ] |
|  | then |
|  | let count=$count+1 |
|  | fi |
|  | done |
|  | echo "Total count of user ID between 500 to 10000 is: $count" |
|  | fi |



The output:



33→

Requirements:

Fetch each directories from PATH variable.

Use -x option if if condition to check executable

permission.

Print directory and number of executable files oneby-

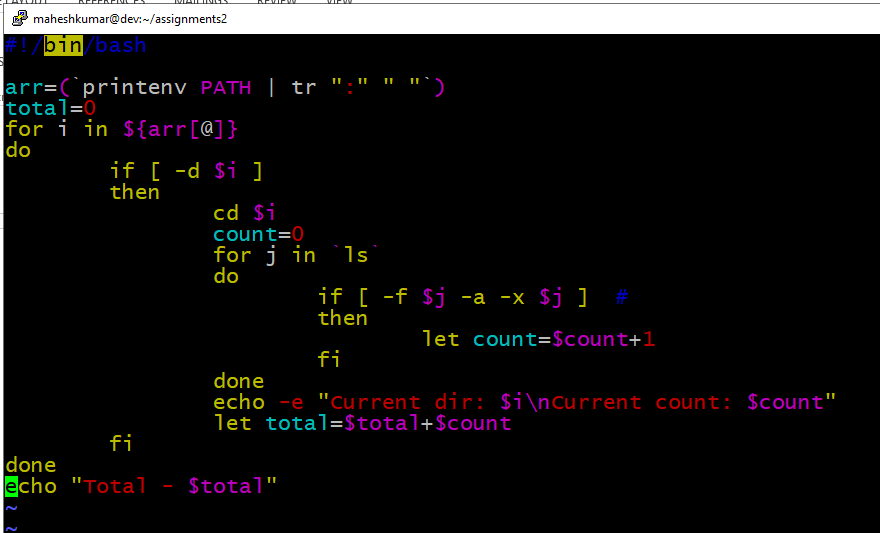
one.

Print the total number of executable files at last.

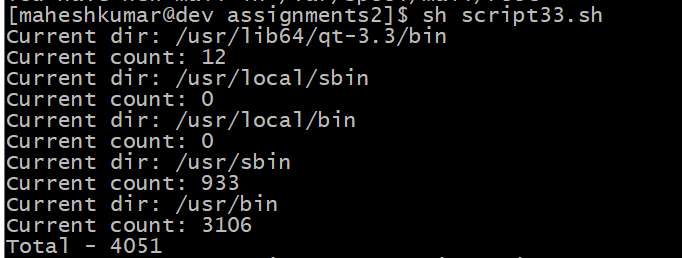
Count only files have executable permission.

Verify path is present every-time.

|  |
| --- |
| #!/bin/bash |
|  |  |
|  | arr=(`printenv PATH | tr ":" " "`) |
|  | total=0 |
|  | for i in ${arr[@]} |
|  | do |
|  | if [ -d $i ] |
|  | then |
|  | cd $i |
|  | count=0 |
|  | for j in `ls` |
|  | do |
|  | if [ -f $j -a -x $j ] # |
|  | then |
|  | let count=$count+1 |
|  | fi |
|  | done |
|  | echo -e "Current dir: $i\nCurrent count: $count" |
|  | let total=$total+$count |
|  | fi |
|  | done |
|  | echo "Total - $total" |



The output:



34→

Prerequisites:

Must know df, cut & tr commands.

Loops and arrays.

Objective:

Learn about etc configuration files.

Requirements:

Fetch user-names from the first field in /etc/passwd

file.

Search given name in the list.

|  |
| --- |
|  |
| awk -F: '{print $1}' /etc/passwd |
|  |

Output:

