Flie 1 = hello.sh

#! /bin/bash

# this is a comment

echo "Hello world this is a shell scripting class" #this is also a comment

# below are the system defined varibles examples:

echo my shell name is $BASH #this variable gives the bash or shell

echo my shell version is $BASH\_VERSION #this gives the version

echo my home is $HOME #gives the home

echo my present working place $PWD #give the present working directory

# below the user defined varibles

name=Vikas

echo My Name is $name

10val=10 #variable name should not start with number it will not be considered

echo value $10val

val=10

echo value of number is $val

# Read user input

echo "Enter name: "

read name

echo "enterd name : $name"

# adding multiple names

echo "Enter 3 names : "

read name1 name2 name3 # adding multiple names

echo "Names : $name1 , $name2 , $name3 "

# same line

read -p 'username : ' user\_var

read -sp 'password : ' pass\_var

echo

echo "username : $user\_var"

echo "password : $pass\_var"

# read an array (-a)

echo "Enter array names: "

read -a names

echo "Names : ${names[0]}, ${names[1]}"

# trick

echo "Enter name: "

read # if u dont read anything here it will go to the default space and can be used.

echo "Name : $REPLY"

**File2 = passArgs.sh**

#! /bin/bash

#pass arguments to the bash script

echo $1 $2 $3 '>echo $s1 $2 $3'

# ./passArgs.sh tom jerry cartoon = use this cmd

# output = tom jerry cartoon >echo $s1 $2 $3

echo $0 $1 $2 $3 '>echo $s1 $2 $3'

# ./passArgs.sh tom jerry cartoon = use this cmd

# output = ./passArgs.sh tom jerry cartoon >echo $s1 $2 $3

#pass args as an array

echo $0 $1 $2 $3

args=("$@")

echo ${args[0]} ${args[1]} ${args[2]} ${args[3]}

# The array 0 is gng to take " tom " that is the first argument

# The array 1 is gng to take " jerry " that is the second argument

# The array 2 is gng to take " cartoon " that is the third argument

# even if the ${args[3]} is present its gng to work bcoz for an array its gng to take value 1

echo ${args[0]} ${args[1]} ${args[2]}

# trick pass args as an array

echo $0 $1 $2 $3

args=("$@")

echo $@ #defalut variable

# print number of args

echo $#

File3: ifCondition.sh

#! /bin/bash

#if condition

#syntax:

# if [condtion]

# then

# statement

# fi = to end the if statement

#integer comparison = (), (())

# -eq = equal, -ne = not equal, -gt = greater than, -lt = less than, -le = less than or equal to

# < = less than , <= ,> ,>= etc..

#string comparison = [], [[]]

# == - equal, != - not equal.

# [[]] == < = less than , <= ,> ,>= etc..

# -z string is null, that is, has zero length

###############################

# integer comparsion

count=10

if [ $count -eq 9 ]

then

echo "condition is true"

fi

# when you run it does not give any output becoz the condition is not ture

count=10

if [ $count -eq 10 ]

then

echo "condition is true for equal"

fi

# when you run it does gives any output of "condition is true for equal"

count=10

if [ $count -ne 100 ]

then

echo "condition is true for not equal"

fi

count=10

if [ $count -gt 5 ]

then

echo "condition is true for greater for 5"

fi

# we can use double closes (()) for >, >= <, <=

count=10

if (( $count > 7 ))

then

echo "condition is true for greater for 7"

fi

######################

# String comparison

word=abc

if [ $word == "abc" ]

then

echo "its true for abc"

fi

word=abc

if [ $word != "abcwdwd" ]

then

echo "its true not for abc"

fi

### [[]]

word1=a

if [[ $word1 < "b" ]]

then

echo "condtion is true <"

fi

### ELSE

word1=h

if [[ $word1 < "b" ]]

then

echo "condtion is true <"

else

echo "condition is false <"

fi

### multiple condtion

word2=a

if [[ $word2 = "b" ]]

then

echo "condtion is true <"

elif [[ $word2 == "a" ]]

then

echo "condition a is true in multiple <"

else

echo "condition is false <"

fi

File4: FileTestOperators.sh

#! /bin/bash

echo -e "Enter the file name : \c" #use -e and \c for the same line input.

read file\_name

if [ -e $file\_name ] # -e is find the file which is built in operator.

then

echo " $file\_name found "

else

echo " $file\_name not found "

fi

# -f = wheather file exist or not and its a regular file or not ?

echo -e "Enter the file name : \c" #use -e and \c for the same line input.

read file\_name

if [ -f $file\_name ] # -e is find the file which is built in operator.

then

echo " $file\_name found "

else

echo " $file\_name not found "

fi

# -d = check for directory

echo -e "Enter the file name : \c" #use -e and \c for the same line input.

read file\_name

if [ -d $file\_name ] # -e is find the file which is built in operator.

then

echo " $file\_name found "

else

echo " $file\_name not found "

fi

# -c = character file

# -b = block file (binary file)

# -s = checks wheather the file is empty or not

# -r = read

# -w = write

# -x = execute

# ls -l = gives info about the files.

File5: appendData.sh

#! /bin/bash

echo -e "Enter the name of the file : \c"

read file\_name

if [ -f $file\_name ]

then

if [ -w $file\_name ] # checking the write permission to the file.

then

echo " Type some txt data. To quit press crtl + d"

cat >> $file\_name # >> means it will append

else

echo "The file do not have write permissions."

fi

else

echo "$file\_name not exists"

fi

file6: logicalOperators.sh (AND)

#! /bin/bash

#normal way of AND operator

age=25

if [ "$age" -gt 18 ] && [ "$age" -lt 30 ]

then

echo "vaild age"

else

echo "age not vaild"

fi

# AND can also be used " -a " below is the code example

age=25

if [ "$age" -gt 18 -a "$age" -lt 30 ]

then

echo "vaild age"

else

echo "age not vaild"

fi

# other way also for AND operator

age=25

if [[ "$age" -gt 18 && "$age" -lt 30 ]]

then

echo "vaild age"

else

echo "age not vaild"

fi

File7: logicalOperators.sh (OR)

#! /bin/bash

# OR operator ||

age=60

if [ "$age" -eq 18 ] || [ "$age" -eq 30 ]

then

echo "vaild age"

else

echo "age not vaild"

fi

# OR = -o

age=60

if [ "$age" -eq 18 -o "$age" -eq 30 ]

then

echo "vaild age"

else

echo "age not vaild"

fi

# double brackets

age=60

if [[ "$age" -eq 18 || "$age" -eq 30 ]]

then

echo "vaild age"

else

echo "age not vaild"

fi

File8: arithmeticOperations.sh

#! /bin/bash

# arithmetic operations

num1=20 # remember no spaces like num1 = 20, num1= 20 will not be vaild.

num2=5

echo $(( num1 + num2 )) # dollar $ symbol and double brackets are must (( ))

echo $(( num1 - num2 ))

echo $(( num1 \* num2 ))

echo $(( num1 / num2 ))

echo $(( num1 % num2 ))

# other way is the "expr" method

echo "expr method way"

echo $( expr $num1 + $num2 )

echo $( expr $num1 - $num2 )

echo $( expr $num1 \\* $num2 ) # for multiple in expr we need to use \\* to make it work.

echo $( expr $num1 / $num2 )

echo $( expr $num1 % $num2 )

File9: floatOperations.sh (decimal operations)

#! /bin/bash

# floating operations using "BC" - its a lang for calculations.

num1=24.5 # remember no spaces like num1 = 20, num1= 20 will not be vaild.

num2=5

echo "20.5+4" | bc

echo "20.5-4" | bc

echo "20.5\*4" | bc

echo "scale=6;20.5/4" | bc #scale will give the decimal points of the value.

echo "20.5%4" | bc

# calculations via varibale values.

echo "below are the calculations via varibale values."

echo "$num1+$num2" | bc

echo "$num1-$num2" | bc

echo "$num1\*$num2" | bc

echo "$num1/$num2" | bc

echo "$num1%$num2" | bc

echo "below is the sqrt example"

num3=27

echo "scale=2;sqrt($num3)" | bc -l #-l is the math library, sqrt = square root

echo "for the power example "

echo "scale=2;4^4" | bc -l

File10: caseStatement.sh

vehicle=$1

case $vehicle in

"car" )

echo "Rent of $vehicle is 100 doller" ;;

"van" )

echo "Rent of $vehicle is 80 doller" ;;

"bicycle" )

echo "Rent of $vehicle is 5 dollar" ;;

"truck" )

echo "Rent of $vehicle is 150 dollar" ;;

\* )

echo "Unknown vehicle" ;;

esac

# example output:

# vv3004@vv3004:~/Desktop$ ./caseStatement.sh bike

# Unknown vehicle

# vv3004@vv3004:~/Desktop$ ./caseStatement.sh van

# Rent of van is 80 doller

echo -e "Enter some character : \c"

read value

case $value in

[A-Z] )

echo " user entered $value which is in A to Z " ;;

[a-z] )

echo " user entered $value which is in a to z " ;;

[0-9] )

echo " user entered $value which is in 0 to 9 " ;;

? )

echo " user entered $value which is in special chracter " ;;

\* )

echo " Unknown value " ;;

esac

File11: Arrays.sh

#! /bin/bash

os=('ubuntu' 'windows' 'macos')

os[3]='kali'

unset os[2] # to remove the array

echo "${os[@]}"

echo "${os[0]}"

echo "${!os[@]}"

echo "${#os[0]}" # length of the array

string=cwiubewufbu

echo "${string[@]}"

echo "${string[0]}" # finds the zero index

echo "${string[1]}" # finds the first index

echo "${#string[@]}"

File12: whileLoops.sh

#! /bin/bash

# while loops

n=1

while [ $n -le 10 ]

do

echo "$n"

n=$(( n+1 ))

done

# other way through the post increment

echo "other way through the post increment"

n=1

while [ $n -le 10 ]

do

echo "$n"

(( n++ ))

done

# the double brackets and <= operator way

echo "the double brackets and <= operator way"

n=1

while (( $n <= 10 ))

do

echo "$n"

(( n++ ))

done

File13: whileLoops2.sh

n=1

while [ $n -le 10 ]

do

echo "$n"

(( n++ ))

sleep 1 # does the loops after every one second

done

# continous loop

# echo "here comes the continous loop "

# n=1

# while [ $n -le 10 ]

# do

# echo "$n"

# sleep 1 # does the loops after every one second

# done

# open terminal gnome

n=1

while [ $n -le 3 ]

do

echo "$n"

(( n++ ))

gnome-terminal &

done

File14: readFileContent.sh

#!/bin/bash

# while loops

# read file content

# 1st way = input redirection

while read p # here p is a variable

do

echo $p

done < readFileContent.sh # < is to read the direction the file is redirected to while loop

# 2nd way read in a single varible and then print it.

cat readFileContent.sh | while read p # cat is the output to while input.

do

echo $p

done

# 3rd way is IFS way

while IFS=' ' read -r p

do

echo $line

done < readFileContent.sh

File15: untillLoop.sh

#! /bin/bash

#untill loops

# In the while loop the if the condition is true then it excutes the program

# In the Untill loop if the condition is false then it excutes the program

# syntax

#until [ condition ]

#do

# command1

# command2

# ..

# ...

# commandN

#done

n=1

until [ $n -gt 10 ]

do

echo $n

n=$(( n+1 ))

done

echo "below is the 2nd way."

n=1

until (( $n > 10 ))

do

echo $n

n=$(( n+1 ))

done

file16: forLoop.sh

#!/bin/bash

#for loops

# different ways

#for VARIABLE in 1 2 3 4 5 .. N

#do

# command1

# command2

# commandN

#done

#OR--------------------------------------------------------------------------------------

#for VARIBALE in file1 file2 file3

#do

# command1

# command2

# commandN

#done

#OR--------------------------------------------------------------------------------------

#for OUTPUT in $(Linux-Or-Unix-Command-Here)

#do

# command1 on $OUTPUT

# command2 on $OUTPUT

# commandN

#done

#OR--------------------------------------------------------------------------------------

#for (( EXP1; EXP2; EXP3 ))

#do

# command1

# command2

# command3

#done

# Examples

for i in 1 2 3 4 5 6 7

do

echo $i

done

# if you have bash version greater than 3

echo "if you have bash version greater than 3"

for k in {1..10}

do

echo $k

done

# {start..end..increment}

echo "{start..end..increment}"

for j in {1..10..2}

do

echo $j

done

# one more example

echo "{start..end..increment}"

for j in {0..10..2}

do

echo $j

done

# other way (( ))

echo ${BASH\_VERSION}

for (( i=0; i<5; i++ ))

do

echo $i

done

# for loops to excute

for command in ls pwd date

do

echo "----------------$command-----------------"

$command

done

#

for item in \*

do

if [ -d $item ]

then

echo $item

fi

done

File17: selectLoop.sh

#!/bin/bash

# select loop

#syntax

#select varName in list

#do

# command1

# command2

# ....

# .....

# commandN

#done

#Example:

#select name in mark john tom ben

#do

# echo "$name selected"

#done

#example 2 using case and select loop

select names in mark john tom ben

do

case $names in

mark )

echo mark selected

;;

john )

echo john selected

;;

tom )

echo tom selected

;;

ben )

echo ben selected

;;

\*)

echo " plz provide the no btw 1..4 "

esac

done

file18: breakAndContinue.sh

#!/bin/bash

# Break and Continue

echo " below this is the break example example "

for (( i=1; i<=10; i++ ))

do

if [ $i -gt 5 ]

then

break

fi

echo "$i"

done

#continue

echo " below this is the continue example "

for (( k=1 ; k<=10 ; k++ ))

do

if [ $k -eq 3 -o $k -eq 6 ]

then

continue #f the condition is true it will skip and continue the loop accordingly.

fi

echo "$k"

done

file19: functions.sh

#!/bin/bash

#functions = sub routine a code block that has set of rules which can be used once or multiple times.

#syntax

# 2 ways:

# type 1:

#function name(){

# commands

# }

# type 2:

#name () {

# commands

# }

# examples:

function Hello(){

echo "Hello whats u r name"

}

quit () {

exit

}

Hello #calling the function hello

echo "foo"

quit #calling the fn quit

# pass args to the function

echo " pass args to the function "

function print(){

echo $1 $2 $3

}

quit1 () {

exit

}

print hello world again # passing args to the function and calling it multiple times

echo "fooo"

quit1

file20: localVaribles.sh

#!/bin/bash

# all varibales are global in shell script

# local varibles

function print(){

local name=$1 # by adding local it becomes the local varible which can be used within the function.

echo " the name is $name"

}

name="Tom"

echo " The name is $name : Before"

print Max

echo " The name is $name : After" # comment local key word ans see the change.

File21: functionExample.sh  
#!/bin/bash

# example for the funtions checks for the file is present or not.

usage() {

echo " you need to provide an arg : "

echo " usage : $0 file\_name "

}

is\_file\_exist() {

local file="$1" # this $1 is provided to the function

[[ -f "$file" ]] && return 0 || return 1 #ternary operation

}

[[ $# -eq 0 ]] && usage # gives the number of args

if ( is\_file\_exist "$1" ) # this $1 is provided to the args for calling it.

then

echo " file found "

else

echo " file not found "

fi

file22: readOnly.sh

#!/bin/bash

# read only commands , ( we canot overide or change when its readonly )

# -f for the funtions

# -e for the variables

var=31

readonly var # using key word " readonly " to make it.

var=50

echo "var => $var"

hello() {

echo "hello world"

}

readonly -f hello # using key word " readonly " to make it for funtions u need to add -f also.

hello() {

echo " hello world again "

}

readonly # makes whole file read only if u add it at the add of u r file.

File23: signalsAndTraps.sh

#!/bin/bash

# signals and traps

# $$ = gives the pid number.

# ctrl+Z = signal

# kill process by = " kill -9 (pid number) "

# signals

#Example : 01

trap "echo exit cmd is detected " SIGINT # if signal is 0 then it is a success symbol

echo "pid is $$"

while (( count < 10 ))

do

sleep 10

(( count ++ ))

echo $count

done

exit 0

#Traps

#SIGKILL, #SIGSTOP trap will not echo anything if these two are used

#Example : 01

#trap "echo exit cmd is detected " 0 # if signal is 0 then it is a success symbol

#echo " hello world "

#exit 0

# use cmd " man 7 signal " to get the whole list of it.

File24: debug.sh

#!/bin/bash

#debug

#This command can be used debug the file content and procedure " bash -x ./<filename> "

# example = " bash -x ./debug.sh "

# " set -x " is the starts exact point where you want to debug

# " set +x " is the end exact point of the debug.

echo "vikas is here "