```
#! /bin/bash

#Tutorial 1
#Hello
echo "Hello World"

#Variable types - system (Capital always) and user defined
#Use of system variables
echo Our shell is $BASH
echo Our Shell version is $BASH_VERSION
echo Our home directory is $HOME
echo Our PWD is $PWD

#Define user define variables
name=Mark

#Use of user defined variable
echo The name is $name

#To clear the contents on screen - clear

#To get input from Keyboard and assign to variable
echo Enter the name
#read name
echo Entered name $name

#Tutorial 2
#To enter multiple names using read command
echo Enter the names
#read name1 name2 name3
echo Entered name $name1, $name2, $name3

#Want to read the name in same line after a message
#read -p 'username :' user_var
echo Username is $user_var

#Want to read password but dont want to show what is entering on console
#read -sp 'password :' pass_var
echo Password is $pass_var

#read command with no variable - Automatically goes into builtin variable REPLY
echo Entername:
#read
echo Name is $REPLY

#To save multiple inputs in array
echo Enter names
#read -a names
echo Names are ${names[0]} ${names[1]}  # To echo some indexes of array

#Tutorial 3
#Pass arguments From a bash script- Automatically goes into default arguments $1
$2 etc
# On console we write ./hello.sh Shivani Ahana Mayank with ./hello.sh being $0
argument
echo $1 $2 $3 $4 '> echo $1 $2 $3'

#For Passing arguments into array
argvar=("$@")
# To echo some indexes of argument array
echo ${argvar[0]} ${argvar[1]}
# To print all the arguments passed in array
echo $@
# To print the number of variables that has been passed
echo $#
```

```
#Tutorial 4
# If - else :
#Integer Comparison- equality
count=10
if [ $count -eq 9 ]
then
echo "Condition is true"
else
echo "Condition is false"
fi

#Non equality
if [ $count -ne 9 ]
then
echo "Condition is true"
else
echo "Condition is false"
fi


#Greater than
if [ $count -gt 9 ]
then
echo "Condition is true"
else
echo "Condition is false"
fi


#Greater or equal
if [ $count -ge 9 ]
then
echo "Condition is true"
else
echo "Condition is false"
fi


#less than
if [ $count -lt 9 ]
then
echo "Condition is true"
else
echo "Condition is false"
fi


#lesser or equal
if [ $count -le 9 ]
then
echo "Condition is true"
else
echo "Condition is false"
fi


#less than - <
if (($count < 9))
then
echo "Condition is true"
else
echo "Condition is false"
fi

#less or equal - <=
if (($count <= 9))
```

```
then
echo "Condition is true"
else
echo "Condition is false"
fi

#greater than - >
if (($count > 9))
then
echo "Condition is true"
else
echo "Condition is false"
fi

#greater or equal - >=
if (($count >= 9))
then
echo "Condition is true"
else
echo "Condition is false"
fi

#String Comparison - Equality
word=abcde
if [ $word = abcdef ]
then
echo "Condition is true"
else
echo "Condition is false"
fi

#or

if [ $word == abcdef ]
then
echo "Condition is true"
else
echo "Condition is false"
fi

#Not equal to
if [ $word != abcdef ]
then
echo "Condition is true"
else
echo "Condition is false"
fi

#Less than in ASCII alphabetical order
if [[ $word < abcdef ]]
then
echo "Condition is true"
else
echo "Condition is false"
fi

#Greater than in ASCII alphabetical order
if [[ $word > abcdef ]]
then
echo "Condition is true"
else
echo "Condition is false"
fi

#String is null
if [[ -z $word ]]
```

```
then
echo "Condition is true"
else
echo "Condition is false"
fi

#Tutorial 5
#To check information related to files- File test operators
# \c - to keep cursor on same line
# -e flag enable interpreter of \c
echo -e "Enter name of file: \c "
#read file_name

#To check if file exist -e filename
if [ -e $file_name ]
then
echo "$file_name found"
else
echo "$file_name not found"
fi

#To check if file is a reagular file or not -f flag
if [ -f $file_name ]
then
echo "$file_name found"
else
echo "$file_name not found"
fi

#To check if directory exist -d flag
echo -e "Enter name of directory: \c "
#read dir_name
if [ -d $dir_name ]
then
echo "$dir_name found"
else
echo "$dir_name not found"
fi

#To check if file is block special -b and for character special -c
if [ -b $file_name ]
then
echo "$file_name found"
else
echo "$file_name not found"
fi

if [ -c $file_name ]
then
echo "$file_name found"
else
echo "$file_name not found"
fi


#To check if file is empty or not -s
if [ -s $file_name ]
then
echo "$file_name not empty"
else
echo "$file_name empty"
fi

#To check if file has read permission -r
if [ -r $file_name ]
then
```

```
echo "$file_name read allowed"
else
echo "$file_name read not allowed"
fi

#To check if file has write permission -w
#To check if file has execute permission -x
#To write something in a file: cat > filename -> Press enter -> Enter text like
hello -> Ctrl+d to come out of cat

#Tutorial 6
# To append output to the end of text file
echo -e "Enter the file name: \c"
#read file_name
if [ -f $file_name ] #file exist or not
then
if [ -w $file_name ] #Have write permission then cat text into the file
then
echo Type some Text and end using ctrl+d
cat >> $file_name # >> append into existing text whereas > overwrite the file
else
echo "$file_name has no write permissioms"
fi
else
echo $file_name not exists
fi

#Tutorial 7
#Logical AND operator
#Way 1 - Use of &&
age=25
#if [ $age -gt 18 ] && [ $age -lt 30 ] or
if [[ $age -gt 18 && $age -lt 30 ]]
then
echo Success
else
echo Fail
fi

#Way 2 - Use of -a flag / and flag
if [ $age -gt 18 -a $age -lt 30 ]
then
echo Success
else
echo Fail
fi

#Tutorial 8
#Logical OR operator
#Way 1 - Use of ||
age=25
#if [ $age -gt 18 ] || [ $age -lt 30 ] or
if [[ $age -gt 18 || $age -lt 30 ]]
then
echo Success
else
echo Fail
fi

#Way 2 - Use of -o flag / or flag
if [ $age -gt 18 -o $age -lt 30 ]
then
echo Success
else
echo Fail
fi
```

```
#Tutorial 9
#Perform Arithmetic operations on Integers

echo 1+1 #Not performing anything as echo take string

#Way 1
num1=20
num2=5
echo $(( num1 + num2 ))
echo $(( num1 - num2 ))
echo $(( num1 * num2 ))
echo $(( num1 / num2 ))
echo $(( num1 % num2 ))

#Way 2 - using expr command
echo $( expr $num1 + $num2 )
echo $( expr $num1 - $num2 )

echo $( expr $num1 * $num2 )
#Give syntax error - as with expr the * is not escaped so add \escape character

echo $( expr $num1 \* $num2 )
echo $( expr $num1 / $num2 )
echo $( expr $num1 % $num2 )

#Tutorial 10
#Perform Arithmetic operations on Floating point
#We have to use special tool "BC - Basic Calculator" because if we perform the
operations using above methods normally then we have Invalid arithmetic operator
error or non integer argument error

#Usually come with linux distribution - On command prompt write: man bc to see
desciption of the calculator

num1=20.5
num2=5

echo 20.5+5 | bc
#the expression got piped into bc as input
echo $num1+$num2 | bc

echo 20.5-5 | bc
echo 20.5*5 | bc

echo 20.5/5 | bc
#not giving proper answer - to solve it we need to set a variable scale ie upto
how many decimal places like 2

echo "scale=2;20.5/5" | bc
echo 20.5%5 | bc

#Square root - function sqrt - Present in math library so we have to use math
library along with bc command: -l for calling the math library
num=27
echo "scale=2;sqrt($num)" | bc -l

#Power
num=3
echo "scale=2;3^3" | bc -l

#Tutorial 11
#Case statement

#Syntax : case expression in pattern1 ) statements ;; pattern2 ) statements ;; * )
statement ;; esac
```

```
#Where *) is default situation - Wildcard for any other case

#Example1
vehicle=$1 #Passing argument from console
case $vehicle in
"Car" )
echo Rent of Car is 7000 ;;
"Scooty" )
echo Rent of Scooty is 500 ;;
* )
echo Unknown vehicle
esac

#Example2 - Enter chracter, evaluate chracter and then see what kind of chracter
is entered - REGULAR EXPRESSIONS

echo -e "Enter some character : \c"
#read value
case $value in
[a-z] ) # This is a pattern
echo Enter chracter is lowercase $value ;;
#Even if K is entered it goes in [a-z] pattern to resolve this we have to set LANG
environment variable to C language as LANG=C
[A-Z] )
echo Enter chracter is uppercase $value ;;
[0-9] )
echo Enter chracter is number $value ;;
? ) #? is a pattern which accepts one letter special chracter
echo Enter character is special $value ;;
* )
echo Unknown input ;;
esac

#Tutorial 12
#Array variables - BASH Supports only 1 dimensional array

#os is array variable
os=('ubuntu' 'windows' 'kali')
os[3]='mac' #To add an element on 3rd index
os[0]='mac' #Updation
unset os[1]
#to remove the element at 1st index - also 1 index will also removed o/p as 0 2 3
as gaps are okay in BASH array
echo "${os[@]}" #To print all
echo "${os[0]}" #To print index 0
echo "${!os[@]}" #To print indexes
echo "${#os[@]}" #To print length of array

#BASH permits array operations on array as well even if the variable are not
explicitly declared as array

#Example
string=shivani
echo "${string[@]}" #Print shivani as it will consider string variable as array
with shivani present at index 0

#Tutorial 13
#While loop - syntax : while [ condition ] do comand 1 command 2 ... done

#Example
n=1
while [ $n -le 10 ]
#while (( $n <= 10 ))
do
echo $n
#n=$(( n+1 ))
```

```
(( n++ ))
done

#Example - Using sleep with while loop
n=1
while [ $n -le 10 ]
do
echo $n
(( n++ ))
#sleep 1 #Pause execution for 1 second
done

#To come out of an infinite loop press Ctrl + C on command prompt

#Example - To open terminals
n=1
while [ $n -le 3 ] #To open 3 terminals
do
echo $n
(( n++ ))
#gnome-terminal &
xterm & # to open xterm terminal
done

#Example - To read file using while loop
#By using input redirecton
#Way 1
while read p #p variable in which we want to save the content line by line
do
echo $p
done < hello.sh # a < b this indicates that the contents of b is redirected to a
as pointing edge is towards a

#Way 2
cat hello.sh | while read p
do
echo $p
done

#Way 2 - The above 2 methods create problem if some special character is present
in the file so we use IFS - Internal field seperator used by shell to determine
how to word splitting ie to recognize word  boundaries

while IFS= read -r p
#while IFS='' read -r p
#we are not providing read command to IFS but in actual the space before read
command , also the -r flag prevents backslash escapes from being interpreted
do
echo $p
done < hello.sh #can give path here as well where file is located

#Tutorial 14
#Until loop syntax: until [ condition ] do command 1 command 2 done
#In this loop commands will execute only if the condition is false

n=1
until [ $n -gt 10 ]
#while (( $n > 10 ))
do
echo $n
#n=$(( n+1 ))
(( n++ ))
done

#Tutorial 15
#for loop syntax1: for VARIABLE in 1 2 3 4 5 .. N do commands done
```

```
#for loop syntax2: for VARIABLE in file1 file2 file3 do command1 on $VARIABLE
command2 done
#for loop syntax3: for OUTPUT in $(Linux-Or-Unix-Command-Here) do command1 on
$OUTPUT command2 on $OUTPUT done
#for loop syntax4: for (( EXP1; EXP2; EXP3 )) do command1 command2 done

#Example 1 -Print and iterate over the numbers
for VARIABLE in 1 2 3 4 5
do
echo $VARIABLE
done
#Not good as this method as we have to write each and every number to iterate over
it, to solve if you have bash version greater than 4.0 then you can write as

#Way 1
for VARIABLE in {1..10} #Indicate want to iterate in 1 - 10
do
echo $VARIABLE
done

#Way 2
for VARIABLE in {1..10..2} #Indicate want to iterate in 1 - 10 with increment by 2
ie 1,5,7,9
do
echo $VARIABLE
done

#Example 2
for (( i=1; i<=5; i++ ))
do
echo $i
done

#Example 3 - Use for loop to execute command
for command in ls pwd date
do
echo --------$command------ #Print command name
$command #execute command
done

for item in *
#* denotes we want to iterate over each and every directory in which we are
currently in
do
if [ -d $item ] #check if item is directory or not
# -f $item to choose all the files
then
echo $item
fi
done

#Tutorial 16
#select loop - allows us to generate easy menues
#Syntax : select VARIABLE in LIST do command 1 command 2 done

#select name in mark john tom ben ross rachel
#do
#echo $name selected
#Give list of name as menus with number provided to each name and the prompt will
ask to give any number for selection and o/p is john selected if press 2
#done

#Select statemet can often be used with case statement
#select name in mark john tom
#do
#case $name in
```

```
#mark)
#echo Mark selected ;;
#john)
#echo john selected ;;
#tom)
#echo tom selected ;;
#*)
#echo Please enter number from list only ;;
#esac
#done

#Tutorial 17
#Break and continue
for (( i=1; i<=10; i++ ))
do
if [ $i -gt 5 ]
then
break
fi
echo $i
done

for (( i=1; i<=10; i++ ))
do
if [ $i -eq 5 ]
then
continue
fi
echo $i
done

#Tutorial 18
#Functions
#Syntax 1 : function name() { commands }
#syntax 2 : name() { commands }

function Hello() { #declaration
echo HelloWOrld
}

print () {
echo $1 $2 #receive arguments
}

quit () {
exit
}

Hello  #Calling
print shivani abhishek #passing arguments while calling

#Local variables
#By default every variable declared in the script is a global variable

function print(){
local name=$1
echo Name is $name
}

print shivani

#Example
function file_exist() {
local file_name=$1
[[ -f $file_name ]] && return 0 || return 1
#If file exist will get 1 then directly jumps to or and if doesnt vice versa will
```

```
happen
}

usage() {
echo Provide argument
}

[[ $# -eq 0 ]] && usage #No of arguments given to script

if ( file_exist $1 )
then
echo found
else
echo not found
fi

#Tutorial 19
#Read only commands

var=31
readonly var
var=21 #It doesnt allow to redefine the new value
echo $var

#On function
hello() {
echo Helloworld
}

readonly -f hello
#-f compulsory to make function readonly

hello() {
echo Hello world dead
}

readonly #It will print all the readonly built in variables or we made in our
script or with flag p
readonly -p

readonly -f #It will print all the readonly built in functions or we made in our
script or with flag p

#Tutorial 20
#Signals and Traps

echo pid is $$ #PID of the script itself
count=1
while (( count < 10 ))
do
#sleep 10
echo $count
(( count ++ ))
done
#exit 0

#Ctrl + c is an interrupt signal and in signal command terms it is a sigint
command
#Ctrl + z can also be used called suspend signal , signal command sigtstp
#We can also kill the running script by opening another terminal and writing kill
-9 pidno where -9 is another signal sig kill

#All these signals and their description can be seen using "man 7 signal" command
with each signal having value greater than 0 so in trap command if 0 will received
by it then commands after tap is executed
#To avoid such interruptions trap command is used to capture this interruption
```

```
signals and clean it up within the script.
#Trap SIGKILL or SIGSTOP it is unable to catch them - dont use them
#To determine the trap commands used in the scripts or in any running sessions
write trap on command prompt

#Example 1
trap " echo Exit command detected" 0
echo Hello world
#exit 0

#Example 2
trap " echo Exit command detected" SIGINT
echo pid is $$ #PID of the script itself
count=1
while (( count < 10 ))
do
#sleep 10
echo $count
(( count ++ ))
done
#exit 0

#Example 3
file=demo.txt
trap "rm -f $file && echo file detected; exit" 0 SIGINT #; combines two commands
echo pid is $$ #PID of the script itself
count=1
while (( count < 10 ))
do
#sleep 10
echo $count
(( count ++ ))
done
#exit 0

#To remove traps : write trap -name of signal eg trap - 0 SIGINT

#Tutorial 21
#DEBUGGING
#Bash provides extensive debugging features - the most common way is to start the
subshell using -x option by writing as bash -x ./hello.sh so it will print all the
execution steps and we can capable of determining at which point debugging is
required

#In script we have to use set option
#Way 1
#! /bin/bash -x
file=demo.txt
trap "rm -f $file && echo file detected; exit" 0 SIGINT #; combines two commands
echo pid is $$ #PID of the script itself
count=1
while (( count < 10 ))
do
#sleep 10
echo $count
(( count ++ ))

#Way2
set -x #Starts debugging from where it is written
set +x #Stops debugging after from this point
file=demo.txt
trap "rm -f $file && echo file detected; exit" 0 SIGINT #; combines two commands
echo pid is $$ #PID of the script itself
count=1
while (( count < 10 ))
do
```

```
#sleep 10
echo $count
(( count ++ ))
done
#exit 0
```