**Variable naming conventions**

In Bash scripting, the following are the variable naming conventions:

* Variable names should start with a letter or an underscore (\_).
* Variable names can contain letters, numbers, and underscores (\_).
* Variable names are case-sensitive.
* Variable names should not contain spaces or special characters.
* Use descriptive names that reflect the purpose of the variable.
* Avoid using reserved keywords, such as if, then, else, fi, and so on as variable names.

**Accessing value of variable:**

We can access value of a variable as below-

**$var\_name**

**Reading input :**

For reading the input from user we can use read command. Read command can be used as below-

read [ option ] hint\_text [var\_name]

hint\_text : ----- Text that want to print on screen while reading the the input

var\_name ---- variable in which want to store the value

[ option ]

* -p ---- to print the hint test on screen
* -s ---- to capture the sensitive information, like password
* -n <number> --- used to capture number of characters as input from terminal
* -r -- Specifies that the read command treat a \ (backslash) character as part of the input line, not as a control character.

**Example:**

|  |  |
| --- | --- |
| read -p "enter your name: " name  echo "enter name is: $name" | read -p "enter your name: " name  echo "enter name is: $name" |

**Command line argument**

This is used to read the line while passing the value on script call.

In shell script each value is stored in variable - $1, $2 , $3 ….

* $1 ---- store value of first argument
* $2 ---- stores value of second argument
* $3 --- store value of third argument.

Greeting.sh file

|  |  |
| --- | --- |
| #!/bin/bash  echo "Hello, $1!" | ./greeting.sh rakesh  Output: -" Hello Rakesh" |

**Reading data from file:**

For reading data from file line by line we can use -r option with read command.

**Method 1:** Using while loop

#!usr/bin/env bash

file="temp.txt"

while read -r line; do

echo -e "$line\n"

done <$file

**Method 2:** Using for loop

#!usr/bin/env bash

file=$(cat temp.txt)

for line in $file

do

echo -e "$line\n"

done

**Writing and Appending to file:**

For writing and appending the some value to a file we have below two operator-

1. > ----- to write in file
2. >> ----- To append at end of file

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

# Basic Operators in Shell Scripting #

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

There are 5 basic operators in bash/shell scripting:

* Arithmetic Operators
* Relational Operators
* Boolean Operators
* Bitwise Operators
* File Test Operators

1. **Arithmetic Operators:** These operators are used to perform normal arithmetics/mathematical operations. There are 7 arithmetic operators:

* Addition (+): Binary operation used to add two operands.
* Subtraction (-): Binary operation used to subtract two operands.
* Multiplication (\*): Binary operation used to multiply two operands.
* Division (/): Binary operation used to divide two operands.
* Modulus (%): Binary operation used to find remainder of two operands.
* Increment Operator (++): Unary operator used to increase the value of operand by one.
* Decrement Operator (- -): Unary operator used to decrease the value of a operand by one

2. **Relational Operators:** Relational operators are those operators which define the relation between two operands. They give either true or false depending upon the relation. They are of 6 types:

* ‘==’ Operator / **-eq** : Double equal to operator compares the two operands. Its returns true is they are equal otherwise returns false.
* ‘!=’ Operator / **-ne**: Not Equal to operator return true if the two operands are not equal otherwise it returns false.
* ‘<‘ Operator/**-lt** : Less than operator returns true if first operand is less than second operand otherwise returns false.
* ‘<=’ Operator / **-le**: Less than or equal to operator returns true if first operand is less than or equal to second operand otherwise returns false
* ‘>’ Operator / **-gt**: Greater than operator return true if the first operand is greater than the second operand otherwise return false.
* ‘>=’ Operator/ **-ge** : Greater than or equal to operator returns true if first operand is greater than or equal to second operand otherwise returns false

**3. Logical Operators :** They are also known as boolean operators. These are used to perform logical operations. They are of 3 types:

* Logical AND (&&): This is a binary operator, which returns true if both the operands are true otherwise returns false.
* Logical OR (||): This is a binary operator, which returns true is either of the operand is true or both the operands are true and return false if none of then is false.
* Not Equal to (!): This is a unary operator which returns true if the operand is false and returns false if the operand is true.

**4. Bitwise Operators:** A bitwise operator is an operator used to perform bitwise operations on bit patterns. They are of 6 types:

* Bitwise And (&): Bitwise & operator performs binary AND operation bit by bit on the operands.
* Bitwise OR (|): Bitwise | operator performs binary OR operation bit by bit on the operands.
* Bitwise XOR (^): Bitwise ^ operator performs binary XOR operation bit by bit on the operands.
* Bitwise complement (~): Bitwise ~ operator performs binary NOT operation bit by bit on the operand.
* Left Shift (<<): This operator shifts the bits of the left operand to left by number of times specified by right operand.
* Right Shift (>>): This operator shifts the bits of the left operand to right by number of times specified by right operand.

**5. File Test Operator:** These operators are used to test a particular property of a file.

* -b operator: This operator check whether a file is a block special file or not. It returns true if the file is a block special file otherwise false.
* -c operator: This operator checks whether a file is a character special file or not. It returns true if it is a character special file otherwise false.
* -d operator: This operator checks if the given directory exists or not. If it exists then operators returns true otherwise false.
* -e operator: This operator checks whether the given file exists or not. If it exits this operator returns true otherwise false.
* -r operator: This operator checks whether the given file has read access or not. If it has read access then it returns true otherwise false.
* -w operator: This operator check whether the given file has write access or not. If it has write then it returns true otherwise false.
* -x operator: This operator check whether the given file has execute access or not. If it has execute access then it returns true otherwise false.
* -s operator: This operator checks the size of the given file. If the size of given file is greater than 0 then it returns true otherwise it is false.

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

**# Conditional statement: #**

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

**If / else statement:**

We have below syntax of if else statement in shell scripting.

if [[ condition ]];

then

statement

elif [[ condition ]]; then

statement

else

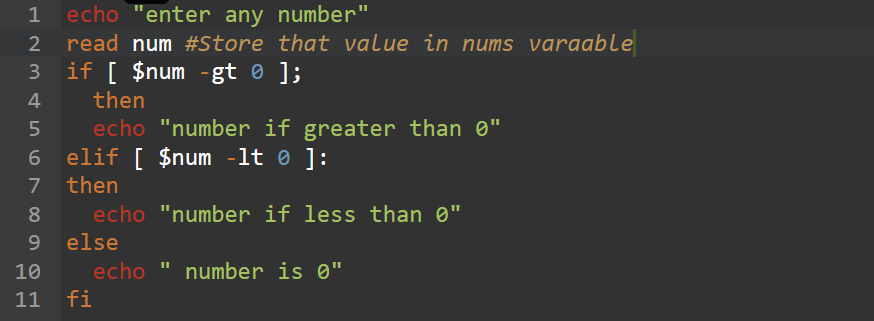
do this by default

fi

**Question:**

WAS to get number from user and check number is +ve, -ve or 0.

**Solution:**



**Nested if/else in shell**

This is same as nested if/else in python or java.

Example:

#!/bin/bash

number=9

if [[ ${number} -gt 10 ]]

then

    if [[ $number -gt 50 ]]

    then

        if [[ ${number} -gt 100 ]]

        then

            echo "number is grater then 100"

        fi

    else

        echo "number is in between 11 to 50"

    fi

else

    echo "number is less then or equal to 10"

fi

**#################**

**# Case statements #**

**#################**

Case statements are used to compare a given value against a list of patterns and execute a block of code based on the first pattern that matches. It’s syntax is-

case expression in

pattern1)

# code to execute if expression matches pattern1

;;

pattern2)

# code to execute if expression matches pattern2

;;

pattern3)

# code to execute if expression matches pattern3

;;

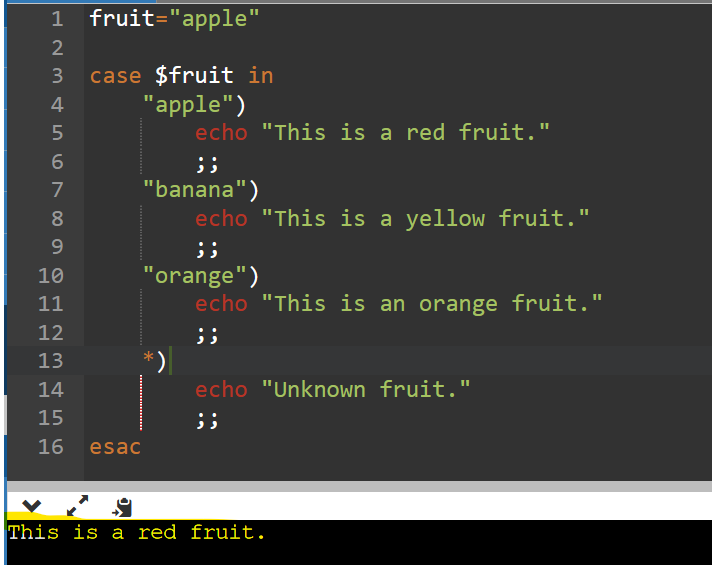
\*)

# code to execute if none of the above patterns match expression

;;

esac

Example:



**################################################################################**

**# Looping and Branching #**

**################################################################################**

**# While loop #**

A while loop is a statement that iterates over a block of code till the condition specified is evaluated to false. We can use this statement or loop in our program when do not know how many times the condition is going to evaluate to true before evaluating to false.

while [ condition ];

do

# statements

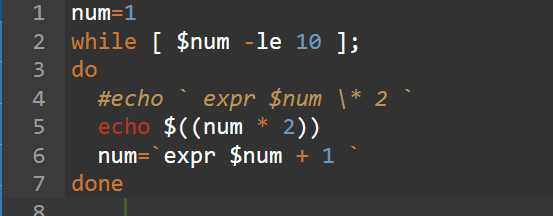
# commands

done

Question:

WAS to print he table of 2.

Solution:



**##########**

**# For Loop #**

**##########**

The for loop operate on lists of items. It repeats a set of commands for every item in a list.

Here var is the name of a variable and word1 to wordN are sequences of characters separated by spaces (words). Each time the for loop executes, the value of the variable var is set to the next word in the list of words, word1 to wordN.

**For loop with list of values**

for var in value1 value2 ... valueN

do

<command 1>

<command 2>

<etc>

done

Example:

|  |  |
| --- | --- |
| #Start of for loop  for a in 1 2 3 4 5 6 7 8 9 10  do  # if a is equal to 5 break the loop  if [ $a == 5 ]  then  break  fi  # Print the value  echo "Iteration no $a"  done |  |

**For loop with initialization, condition and increments /decrement**

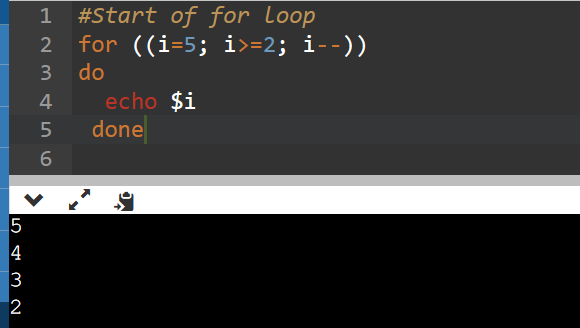
for (( initialization; condition; increament/decrement ))

do

#statement

done

Example:



**################**

**# until statement #**

**################**

The until loop is executed as many as times the condition/command evaluates to false. The loop terminates when the condition/command becomes true.

until <condition>

do

<command 1>

<command 2>

<etc>

done

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

# Function in shell script #

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

Function is block of code which is used to perform some specific task, we can declare it as-

function\_name () {

list of commands

}

Example:

#!/bin/sh

# Define your function here

Hello () {

echo "Hello World"

}

**Passing values to function**

If we want to pass the value to function then we should pass it during function call.

In called function we can access using command line argument ($1, $2, $3..)

Example:

# Define your function here

Hello () {

echo "Hello World $1 $2"

}

# Invoke your function

Hello Zara Ali

Solution:

$./test.sh

Hello World Zara Ali

**Return value from function**

We can return values from function using return keyword.

Returned values can be accessed using $? Parameter.

Example:

