A Shell script is a plain text file. This file contains different commands for step-by-step execution. These commands can be written directly into the command line but from a re-usability perceptive it is useful to store all of the inter-related commands for a specific task in a single file. We can use that file for executing the set of commands one or more times as per our requirements.

Here in this article, we are going to discuss decision-making within Shell Scripting.

In programming, Decision making is one of the important concepts. The programmer provides one or more conditions for the execution of a block of code. If the conditions are satisfied then those block of codes only gets executed. Two types of decision-making statements are used within shell scripting. They are –

* If-else statement
* case-sac statement

Now let’s talk about them one by one.

**1. If-else statement**

If else statement is a conditional statement. It can be used to execute two different codes based on whether the given condition is satisfied or not. There are a couple of varieties present within the if-else statement. They are –

* if-fi
* if-else-fi
* if-elif-else-fi
* nested if-else

The syntax will be –

**if-fi**

if [ expression ]; then

statements

fi

**if-else-fi**

if [ expression ]

then

statement1

else

statement2

fi

**if-elif-else-fi**

if [ expression1 ]

then

statement1

statement2

.

.

elif [ expression2 ]

then

statement3

statement4

.

.

else

statement5

fi

**nested if-else**

if [ expression ]

then

statement1

if [ expression ]

then

statement

else

statement

fi

else

statement2

fi

Now understand these concepts using examples.

**Example of if-fi**

Name="Satyajit"

if [ "$Name" = "Satyajit" ]; then

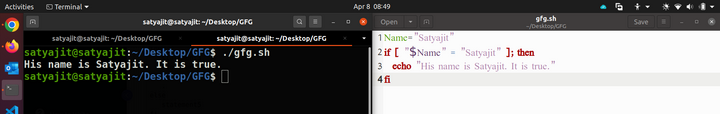
echo "His name is Satyajit. It is true."

fi

Output

His name is Satyajit. It is true.

In the above example, during the condition checking the name matches and the condition becomes true. Hence, the block of code present within the if block gets executed. In case the name doesn’t match then we will not have an output. Below is the terminal shell pictorial depiction after executing the following script –



**Example of if-else-fi**

Age=17

if [ "$Age" -ge 18 ]; then

echo "You can vote"

else

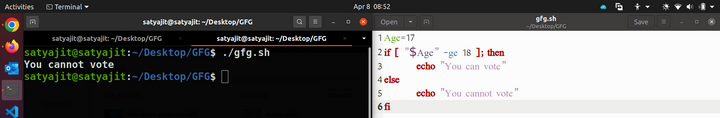
echo "You cannot vote"

fi

**Output**

You cannot vote

In the above example, during the condition checking the Age is 17, so it doesn’t satisfy the condition of if statement that is the age must be greater than or equal to 18. Hence the code inside the if block will not get executed and the code written inside the else block will get executed. Below is the terminal shell pictorial depiction after executing the following script –



**Example of if-elif-else-fi**

Age=17

if [ "$Age" -ge 18 ]; then

echo "You can vote"

elif [ "$Age" -eq 17 ]; then

echo "You can vote after one year"

else

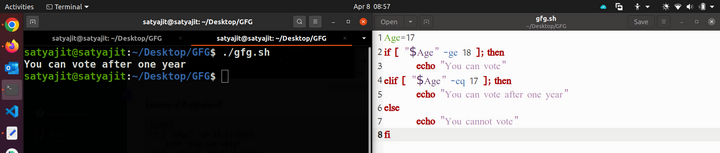
echo "You cannot vote"

fi

**Output**

You can vote after one year

In the above example, during the condition checking the Age is 17, so it doesn’t satisfy the condition of if statement that is the age must be greater than or equal to 18 but it has satisfied the elif condition. So, it has executed the code written within the elif block only.



**Example of Nested if-else**

echo "Enter subject"

read subject

if [ $subject == 'Linux' ]

then

echo "Enter Marks"

read marks

if [ $marks -ge 30 ]

then

echo "You passed"

else

echo "You failed"

fi

else

echo "Wrong Subject"

fi

**Output 1**

Enter subject

Linux

Enter Marks

97

You passed

**Output 2**

Enter subject

Linux

Enter Marks

29

You failed

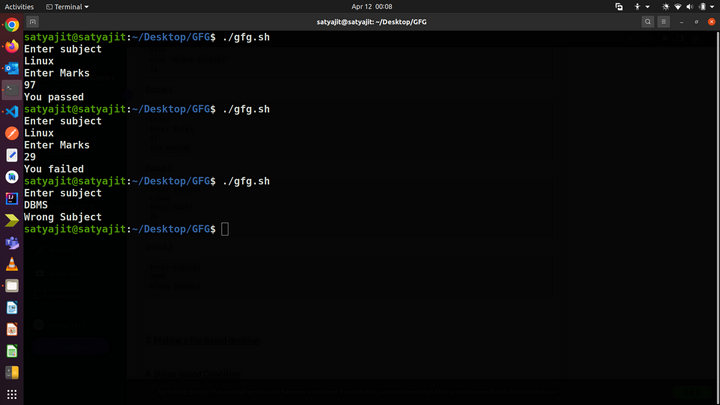
**Output 3**

Enter subject

DBMS

Wrong Subject

In the above example, the if statements are nested, which means one if statement is written inside another one. The following script first checks for the Subject is Linux or not. If the subject is Linux then it goes to the another if statement and that if statement checks for the mark are above to consider as a pass or not.  Below is the terminal shell pictorial depiction after executing the following script –



**1.1. Making a file-based decision**

The file-based decision is basically a type of decision-making based on whether the file exists or not. A use case of this will be to check for available file permission or to create a file if it is not available etc. A minimal structure of such script can be written as –

if [ -e gfg.sh ]

then

echo "file exists"

else

echo "file does not exist"

fi

This can be used to check whether a file exists in the home directory or not.

**Example of file-based decision**

echo "Enter filename"

read filename

if [ -e $filename ]

then

echo "$filename is exits on the directory"

cat $filename

else

cat > $filename

echo "File created"

fi

**Output :**

**First time:**

Enter filename

geeks.txt

Hello Geek

File created

**Second time:**

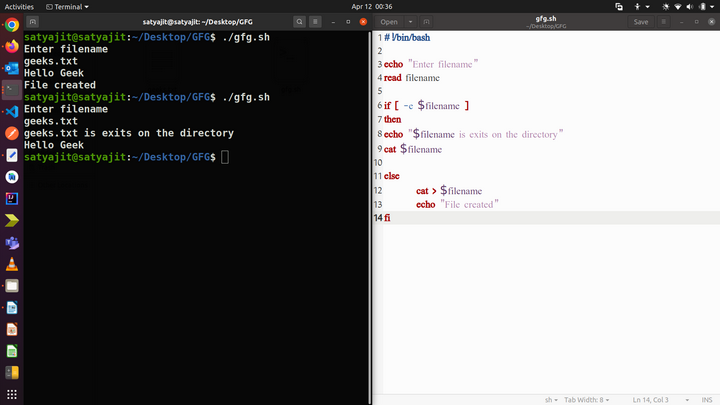
Enter filename

geeks.txt

geeks.txt is exits on the directory

Hello Geek

So, in this above example the first time, the script could not find any file with that file name, and the else block gets executed. It created the file and put some data into that file. When we run it a second time with the same file name, then it finds the file. So, is the if block gets executed and that displays the contents of the file.  Below is the terminal shell pictorial depiction after executing the following script –



**1.2. String-based Condition**

The string-based condition means in the shell scripting we can take decisions by doing comparisons within strings as well. Here is a descriptive table with all the operators –

| **Operator** | **Description** |
| --- | --- |
| == | Returns true if the strings are equal |
| != | Returns true if the strings are not equal |
| -n | Returns true if the string to be tested is not null |
| -z | Returns true if the string to be tested is null |

Below is the usage of all of them  –

# ==

if [ 'Geeks' == 'Geeks' ];

then

echo "same" #output

else

echo "not same"

fi

# !=

if [ 'Geeks' != 'Apple' ];

then

echo "not same" #output

else

echo "same"

fi

# -n

if [ -n "Geeks" ];

then

echo "not null" #output

else

echo "null"

fi

# -z

if [ -z "Geeks" ];

then

echo "null"

else

echo "not null" #output

fi

**Output:**

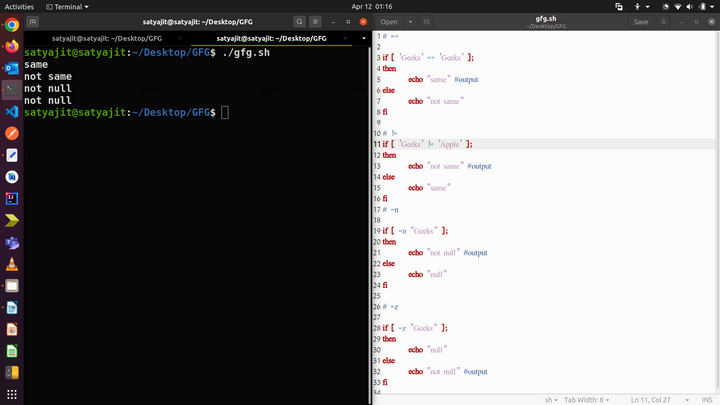
same

not same

not null

not null

Below is the terminal shell pictorial depiction after executing the following script –



**1.3. Arithmetic-based Condition**

Arithmetic operators are used for checking the arithmetic-based conditions. Like less than, greater than, equals to, etc. Here is a descriptive table with all the operators –

| **Operator** | **Description** |
| --- | --- |
| -eq | Equal |
| -ge | Greater Than or Equal |
| -gt | Greater Than |
| -le | Less Than or Equal |
| -lt | Less Than |
| -ne | Not Equal |

Below is the usage of all of them  –

# -eq

if [ 10 -eq 10 ];then

echo "Equal"

fi

# -ge

if [ 10 -ge 9 ];then

echo "Greater or equal"

fi

# -gt

if [ 10 -gt 8 ];then

echo "Greater"

fi

# -le

if [ 10 -le 12 ];then

echo "Less than or equal"

fi

# -lt

if [ 10 -lt 13 ];then

echo "Less than"

fi

# -ne

if [ 10 -ne 13 ];then

echo "Not Equal"

fi

**Output:**

Equal

Greater or equal

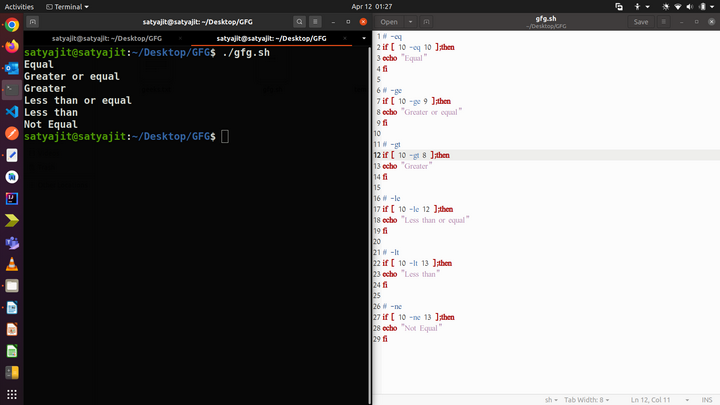
Greater

Less than or equal

Less than

Not Equal

Below is the terminal shell pictorial depiction after executing the following script –



**2. The case-sac statement**

case-sac is basically working the same as switch statement in programming. Sometimes if we have to check multiple conditions, then it may get complicated using if statements. At those moments we can use a case-sac statement. The syntax will be –

case $var in

Pattern 1) Statement 1;;

Pattern n) Statement n;;

esac

**Example of case-sac statement**

Name="Satyajit"

case "$Name" in

#case 1

"Rajib") echo "Profession : Software Engineer" ;;

#case 2

"Vikas") echo "Profession : Web Developer" ;;

#case 3

"Satyajit") echo "Profession : Technical Content Writer" ;;

esac

**Output**

Profession : Technical Content Writer

In the above example, the case-sac statement executed the statement which is a part of the matched pattern here the ‘Name’. Below is the terminal shell pictorial depiction after executing the following script –

