Enabling your Bash script to make decisions is extremely useful. Conditional execution allows you to control the circumstances where certain programs are executed based on whether those programs succeed or fail, but you can also construct conditional expressions which are logical statements that are either equivalent to true or false. Conditional expressions either compare two values, or they ask a question about one value. Conditional expressions are always between double brackets ([[ ]]), and they either use logical flags or logical operators. For example, there are several logical flags you could use for comparing two integers. If we wanted to see if one integer was greater than another we could use -gt, the greater than flag. Enter this simple conditional expression into the command line:

```
[[ 4 -gt 3 ]]
```

The logical expression above is asking: Is 4 greater than 3? No result is printed to the console so let's check the exit status of that expression.

```
echo $?

## 0

```

It looks like the exit status of this program is 0, the same exit status as true. This conditional expression is saying that [[4 -gt 3]] is equivalent to true, which of course we know is logically consistent, 4 is in fact greater than 3! Let's see what happens if we flip the expression around so we're asking if 3 is greater than 4:

```
[[ 3 -gt 4 ]]

```

Again, nothing is printed to the console so we'll look at the exit status:

```
echo $?

## 1

```

Ah-ha! Obviously 3 is not greater than 4, so this false logical expression resulted in an exit status of 1, which is the same exit status as false! Because they have the same exit status [[3 -gt 4]] and false are essentially equivalent. To quickly test the logical value of a conditional expression, we can use the AND and OR operators so that an expression will print "t" if it's true and "f" if its false:

```
[[ 4 -gt 3 ]] && echo t || echo f
[[ 3 -gt 4 ]] && echo t || echo f

## t
## f

```

This is a little trick you can use to quickly look at the resulting value of a logical expression.

These **binary** logical expressions compare two values, but there are also **unary**logical expressions that only look at one value. For example, you can test whether or not a file exists using the -e logical flag. Let's take a look at this flag in action:

```
cd ~/Code
[[ -e math.sh ]] && echo t || echo f

## t
```

As you can see the file math.sh exists! Most of the time when you're writing bash scripts you won't be comparing two raw values or trying to find something out about one raw value, instead you'll want to create a logical statement about a value contained in a variable. Variables behave just like raw values in logical expressions. Let's take a look at a few examples:

```
number=7
[[ $number -gt 3 ]] && echo t || echo f
[[ $number -gt 10 ]] && echo t || echo f
[[ -e $number ]] && echo t || echo f

## t
## f
## f

```

As you can see 7 is greater than 3 though it is not greater than 10, and there is not file in this directory called 7. There are several other varieties of logical flags, and you can find a table of several of these flags below.

Logical Flag	**Meaning**	**Usage**
-gt	**G**reater **T**han	[[$planets -gt 8]]
-ge	**G**reater Than or **E**qual To	[[$votes -ge 270]]
-eq	**Eq**ual	[[$fingers -eq 10]]
-ne	**N**ot **E**qual	[[$pages -ne 0]]
-le	**L**ess Than or **E**qual To	[[$candles -le 9]]
-lt	**L**ess **T**han	[[$wives -lt 2]]
-e	A File **E**xists	[[-e $taxes_2016]]
-d	A **D**irectory Exists	[[-d $photos]]
-z	Length of String is **Z**ero	[[-z $name]]
-n	Length of String is **N**on-Zero	[[-n $name]]

Try using each of these flags on the command line before moving on to the next section.

In addition to logical flags there are also logical operators. One of the most useful logical operators is the regex match operator =~. The regex match operator compares a string to a regular expression and if the string is a match for the regex then the expression is equivalent to true, otherwise it's equivalent to false. Let's test this operator a couple different ways:

```
[[ rhythms =~ [aeiou] ]] && echo t || echo f
my_name=sean
[[ $my_name =~ ^s.+n$ ]] && echo t || echo f

## f
## t
```

There's also the NOT operator !, which inverts the value of any conditional expression. The NOT operator turns true expressions into false expressions and vice-versa. Let's take a look at a few examples using the NOT operator:

```
[[ 7 -gt 2 ]] && echo t || echo f
[[ ! 7 -gt 2 ]] && echo t || echo f
[[ 6 -ne 3 ]] && echo t || echo f
[[ ! 6 -ne 3 ]] && echo t || echo f

## t
## f
## t
## f

```

Here's a table of some of the useful logical operators in case you need to reference how they're used later:

Logical Operator	Meaning	Usage
=~	Matches Regular Expression	[[$consonants =~ [aeiou]]]
=	String Equal To	[[$password = "pegasus"]]
!=	String Not Equal To	[[$fruit != "banana"]]
!	Not	[[! "apple" =~ ^b]]

Mark as completed