When writing computer programs it is often useful for your program to be able to make decisions based on inputs like arguments, files, and environmental variables. Bash provides mechanisms for creating logical expressions which resemble mathematical equations. These logical expressions can be evaluated until they are either true or false. In fact, true and false are both simple Bash commands! Let's try them both out now:

```
true
false
```

At first it doesn't look like they do much. In order to see how they work, we're going to need to look under the hood of Unix a little bit. Whenever you execute a program on the command line, in general one of two things will happen: either the command is executed successfully, or there's an error. In terms of errors there are many ways that a program can go wrong, and Unix can take different actions depending on what kind of error occurs. For example if I enter the name of a command that does not exist into the terminal, then I'll see an error:

```
this_command_does_not_exist

## Error in running command bash

```

Since that command does not exist, it creates a specific kind of error which is indicated by the program's exit status. The exit status of a program is an integer, the exit status of the last program run is stored in the question mark variable ($?). We can take a look at the exit status of the last program with echo:

```
echo $?

## 127

```

This particular exit status made an indication to the shell that it should print an error message to the console. What's the exit status of a program that runs successfully? Let's take a look:

```
echo I will succeed.
echo $?

## I will succeed.
## 0
```

So the exit status of a successful program is 0. Now let's take a look at the exit statuses of true and false:

```
true
echo $?
false
echo $?

## 0
## 1
```

As you can see true has an exit status of 0 and false has an exit status of 1. Since these programs don’t do much else, you could define true as a program that always has an exit status of 0 and false as a program that always has an exit status of 1.

Knowing the exit status of these programs is important when discussing the logical operators: the AND operator (&&) and the OR operator (||). The AND and OR operators can be used for conditional execution of programs on the command line. Conditional execution occurs when the execution of one program depends on the exit status of another program. For example in the case of the AND operator, the program on the right hand side of && will only be executed if the program on the left hand side of && has an exit status of 0. Let’s take a look at some small examples:

```
true && echo "Program 1 was executed."
false && echo "Program 2 was executed."

## Program 1 was executed.
```

Since false has an exit status of 1, the program echo "Program 2 was executed." is not executed, so nothing is printed to the console for that command. Several AND operators can be chained together like so:

```
false && true && echo Hello
echo 1 && false && echo 3
echo Athos && echo Porthos && echo Aramis

## 1
## Athos
## Porthos
## Aramis
```

In a series of programs joined together by AND operators, any programs to the right of a program that has a non-zero exit status is not executed.

The OR operator (||) follows a similar set of principles. Commands on the right hand side of || are only executed if the command on the left hand side fails and therefore has an exit status other than 0. Let’s take a look at how this works:

```
true || echo "Program 1 was executed."
false || echo "Program 2 was executed."

## Program 2 was executed.
```

Only echo "Program 2 was executed." runs because false has a non-zero exit status. You can combine multiple OR operators so that only the first program with an exit status of 0 is executed:

```
false || echo 1 || echo 2
echo 3 || false || echo 4
echo Athos || echo Porthos || echo Aramis

## 1
## 3
## Athos

```

You can combine AND and OR operators in commands, which are evaluated from left to right:

```
echo Athos || echo Porthos && echo Aramis
echo Gaspar && echo Balthasar || echo Melchior
## Athos
## Aramis
## Gaspar
## Balthasar

```

By combining AND and OR operators you can precisely control the conditions for when certain commands should be executed.

Mark as completed