Bash is a popular tool that helps us unleash the full potential of the operating system on our computers. In this topic, we will learn to use bash to conduct mathematical operations. Though doing simple maths is not the *full potential*, it is a necessary stepping stone in fully utilizing the possibilities that bash provides. Here, we will take a look at commands we can use to make basic arithmetic operations and look at arithmetic operators that help us conduct mathematical calculations. We will also work with a command tool that provides extended functionality when working with mathematical operations.

Arithmetic operators

We can use multiple operators in bash to execute mathematical operations. But without using a command of some sort, it cannot determine what we want to do with the numbers and operators we provide. For this, bash has multiple commands. We will use double parentheses i.e.

`(())`, to help us achieve our goal. Using double parentheses is simple yet more flexible than other options. We can simply put our argument in between the double parentheses and use `$` in front of it to execute the operation.

Let's look at the operators available in bash and their respective examples:

Operator	Function	Description	Example	Result
+	Addition	Add two operands	$((10+3))	13

-	Subtraction	Subtract two operands	$((10-3))	7
*	Multiplication	Multiply two operands	$((10*3))	30
/	Division	Return quotient after division	$((10/3))	3
%	Modulo	Return remainder after division	$((10%3))	1
**	Exponential	Raise the first operand to the power of the second	$((10**3))	1000

Now let's try implementing these operators in bash and see the results for ourselves. For this example, we are going to create a script file named

`maths.sh`

to execute our bash commands.

```
#!usr/bin/env/bash

#Executing simple arithmetic operations
echo $(( 10*5+15 ))

echo $(( 40/6 ))
```

Running the script above using the command

`bash maths.sh`

, we get the following output:

```
65

6
```

Now, let's look at some other operators that we can use with variables.

Operations with a variable

We also have the ability to execute mathematical operations on an operand by a constant. Let's suppose the operand in consideration is

`x`

and it is equal to 10.

Operator	Function	Description	Example	Result

+=	Addition by constant	Add two operands	$((*x*+=3))	13
-=	Subtraction by constant	Subtract two operands	$((*x*-=3))	7
*=	Multiplication by constant	Multiply two operands	$((*x**=3))	30
/=	Division by constant	Return quotient after division	$((*x*/=3))	3
%=	Modulo by constant	Return remainder after division	$((*x*%=3))	1

At first glance, this might seem the same and pretty much useless when compared to the original operators, but these can be particularly useful when working with loops. Apart from these basic operators, we also have the ability to increment and decrement a single operand by 1. To do this, we use

`++`

and

`--`

. Using these before or after our operand is referred to as pre-increment/decrement or post-increment/decrement respectively. Be careful while using these operators as they can only be implemented to variables.

```
#!usr/bin/env/bash

#Assigning value to variables
val1=$(( 10*3-15 ))
echo $val1

#Arithmetic operations using constants

echo $(( val1 *= 3 ))
```

We can use both

`val=$(( 2 ))`

and `$(( val = 2 ))`

to assign values to variables.

Running the script above gives us the following output:`15 45`

Everything seems to work well, except for the ability to accurately perform divisions. Taking a closer look at the second command in our first example;

`echo $(( 40/6 ))`

gives 6 instead of 6.6666.....7. However, there is a workaround that we can use to overcome this problem, and in the process enable other helpful features.

The bc utility

As you might have noticed from the examples given above, bash isn't inherently good at maths. We are unable to produce floating-point numbers during division. And the basic operators described above are not enough to justify using bash. But, bash has a trump card up its sleeve. The bc utility; which stands for Basic Calculator, helps us turn our bash command line into a full-fledged calculator.

The

`bc`

utility provides us the ability to work with previously described operators as well as:

- Relational operators
- Logical or Boolean operators
- Math functions
- Conditional statements
- Iterative statements

We will use

`<<<`

i.e. *here-string* to make our commands simpler. The syntax we are going to use is

```
command argument <<< "input"
```

. Let's start by using some relational and logical operators.

Relational and logical operations

Let's create a bash script implementing some relational operators.

```
#!usr/bin/env/bash

#Less than operator
bc <<< "10 < 1"

#Is not equal to operator

bc <<< "10 != 11"
```

This script will give the following output:

```
0

1
```

Other relational operators are `>`, `>=`, `<=` and `==`. Now let's look at logical operators. The `bc` utility provides us with three of these: `&&`, `||` and `!`. The result is given as either `0` or

`1`

; which denotes *false* and *true* respectively.

The operators work in the following way:

Operator	Example	Evaluation logic	Result

&&	bc <<< "5&&6"	1; if all operands are non-zero	1
||	bc <<< "5||6"	1; if any one operand is non-zero	1
!	bc <<< "!5"	1; if the operand is 0	0

Out of all the other functionalities provided by the

`bc` utility, the one that actually helps justify its name is the support for math functions. Now, let's look at the math library.

Math library

To use this function we have to use the special option;

`-l` which enables the standard math library. This allows us to use features like sine and cosine functions, inverse tangents, natural logs and exponential functions, square roots, and base conversions. We can also control the number of digits after the decimal point in a floating-point number.

The following table provides necessary details about some important math functions.

Math function	Explanation	Example
sine	Takes radian value as operand and gives its sine value	bc -l <<< "s(30)"
cosine	Takes radian value as operand and gives its cosine value	bc -l <<< "c(30)"
tan-1	Gives the inverse tangent in radians	bc -l <<< "a(1)"
ln	Gives the natural logarithm	bc -l <<< "l(45)"

e	Raises e(euler's number i.e. 2.7182) to the power of the operand	bc -l <<< "e(1)"
√	Returns square root value	bc -l <<< "sqrt(9)"

Let's take a look at some examples that utilize the standard math library in bash.

```
#!usr/bin/env/bash

#Arithmetic operation that returns a floating-point number
bc -l <<< "(1 + 2)/7 * 2 ** 2"

#Using sine and cosine functions
bc -l <<< "s(1) + c(1)"

#Using logarithmic and exponential functions
bc -l <<< "e(1) + l(5)"

#Finding the square root of a number

bc -l <<< "sqrt(27)"
```

The script above will give the following output:

```
1.71428571428571428568
1.38177329067603622405
4.32771974089314560996

5.19615242270663188058
```

The bc utility also provides functionalities that help us control the number of digits after a decimal point and convert numbers into different bases. We need three arguments for base conversion; ibase(i.e. base of our input), obase(i.e. expected base for our output), and the operand itself. The default value for both ibase and obase is 10(i.e. decimal number system). To separate these arguments, we will use

`;`

.

```
#!usr/bin/env/bash

#Converting binary 10 to octal
bc -l <<< "ibase=2; obase=8; 10"

#Converting decimal 5 to binary
bc -l <<< "obase=2; 5"

#Using scale to control length of floating-point numbers

bc -l <<< "scale=3; a(30)"
```

We will get the following output:

```
2
101

1.537
```

Conclusion

In this topic, we looked at how we can use arithmetic operators and the bc utility in bash to conduct mathematical operations. You are now able to:

- Execute simple arithmetic operations in bash.
- Work with the `bc` utility.

- Utilize relational and logical operators provided by the `bc` utility.
- Use various mathematical functions to carry out complex mathematical operations.

Fill in the blanks with the correct command so that the result will contain 6 digits in total?
bc -l <<< "scale=5;22/7"

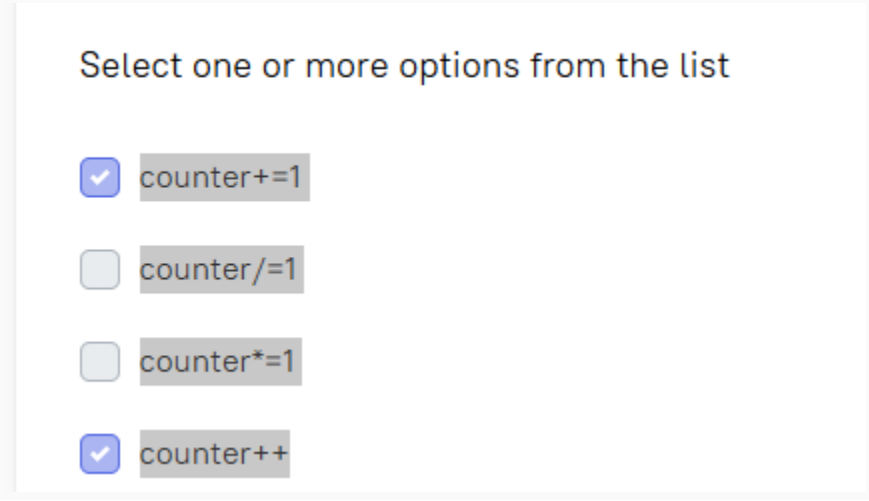
You are given a bash script that contains a loop.

```
#!usr/bin/env/bash

#Simple while loop

counter=1
while [ $counter -le 10 ]
do
    ((_____))
done


echo Complete
```



Which of the following commands can we use in order to convert 16 from octal to binary?

```
bc -l <<< "ibase=8; obase=2; 16"
bc -l <<< "obase=2; ibase=8; 16"
```