Bash arithmetic

WE'LL COVER THE FOLLOWING

- Example 1:
- Example 2:
- Example 3:
- Code example:
- Bash floating point calculations
- Examples:

Bash arithmetic `expansion` provides a powerful tool for performing arithmetic operations in scripts. Translating a string into a numerical expression is relatively straightforward using backticks (`), double parentheses `(( ))`, or `let`.

Example 1:

```
i=1
i=`expr $i + 1`
echo $i
```

Where, `expr` is an all-purpose expression evaluator.

Example 2:

```
i=1
j=$(( i+1 ))
echo $j
```

An example consisting of `let`

Example 3:

```
let i=3+5
echo "3 + 5 =" $i
```

Order of Precedence operators are evaluated in order of precedence. The levels are listed in order of decreasing precedence:

```
       id++ id--
              variable post-increment and post-decrement
       ++id --id
              variable pre-increment and pre-decrement
       - +    unary minus and plus
       ! ~    logical and bitwise negation
       **     exponentiation
       * / %  multiplication, division, remainder
       + -    addition, subtraction
       << >>  left and right bitwise shifts
       <= >= < >
              comparison
       == !=  equality and inequality
       &      bitwise AND
       ^      bitwise exclusive OR
       |      bitwise OR
       &&     logical AND
       ||     logical OR
       expr?expr:expr
              conditional operator
       = *= /= %= += -= <<= >>= &= ^= |=
              assignment
       expr1 , expr2
              comma
```

Apart from the precedence, **operators that work with integers** are given below with some examples:

{title="Operator's order of Precedence in Bash"}

Operator	Description	Example	Output
`+`	Addition	`echo $(( 10 + 1 ))`	`11`
`-`	Subtraction	`echo $(( 11 - 1 ))`	`10`
`/`	Division	`echo $(( 10 / 2 ))`	`5`
`*`	Multiplication	`echo $(( 10 * 5 ))`	`50`
`%`	Modulus	`echo $(( 10 % 3 ))`	`1`
`++`	post-increment (add variable value by 1)	`x=5;echo $(( x++ ));echo $(( x++ ))`	`5 6`
`--`	post-decrement (subtract variable value by 1)	`x=5; echo $(( x-- ))`	`4`
`**`	Exponentiation	`x=3;y=3;echo $(( x ** y ))`	`9`

Code example:

```
#!/bin/bash

x=1
y=2
declare -i n
n=$x+$y
```

```
echo "Result is:$n "

# bash convert binary number input x

n=2 # $x
echo $n

# bash convert octal number input x
n=8 # $x
echo $n

# bash convert hex number input x
result=16 # $x
echo $n
```

Bash floating point calculations

You can perform floating point operation in Bash using the `bc` arbitrary precision calculator language. Note the need to escape the multiply operator `*` with a backslash (`\`) or enclose the arithmetic expression in single quotes (`' '`).

Examples:

```
$ x = 1.1
$ y = 2.2
$ echo x + y | bc -l
3.3

$ echo x - y | bc -l
-1.1

$ echo x \* y | bc -l
2.42

$ echo 'x * y' | bc -l
2.42

$ echo 'x / y' | bc -l
.5000

$ z=`echo '$x / $y' | bc -l`
$ echo $z
.5000

# Wrong use

$ echo x * y | bc -l
1.1
```

Note that there should be no space between the variable name and the equal sign (`=`) in the assignment, otherwise an error occurs.

[illegible]