hitbullseye

Arithmetic in shell, shell script examples, interrupt processing

Arithmetic in the shell:

In this Lecture, we will learn about Arithmetic Operators in Shell Programming.

We will be covering the following math operations in this tutorial.

- Addition +
- Subtraction -
- Multiplication *
- Division /
- Modulus % to get remainder
- Each of these operators performs the operation on two integer variables or constants.

For Example, the below program illustrates each of these operations:

```
$ c=`expr $a + $b`
$ echo “the value of addition=$c”
$ d=`expr $a - $b`
$ echo “the value of subtraction=$d”
```

$ e= expr $a * $b`

$ echo “the value of multiplication=$e”

$ f=`expr $a / $b`

$ echo “the value of division=$f”

$ g= echo `expr $a % $b`

$ echo “the value of modulus=$c”

The Unix shell does not natively support floating-point operations. A separate command-line tool must be used for this. The ‘bc’ command is the most standard tool for this.

Example:

$ c = `echo “$a + $b” | bc`

$ d = `echo “$a + $b” | bc`

Note that each of the operators needs to be surrounded by a space on both sides, and the ‘*’ operators need to be escaped with a backslash ‘\’.

Example:

Here is an example that uses all the arithmetic operators –

```
#!/bin/sh

a=10
b=20
val=`expr $a + $b`
echo "a + b : $val"

val=`expr $a - $b`
echo "a - b : $val"
val=`expr $a \* $b`
echo "a * b : $val"

val=`expr $b / $a`
echo "b / a : $val"
val=`expr $b % $a`
echo "b % a : $val"
```

```
if [ $a == $b ]
then
   echo "a is equal to b"
fi
if [ $a != $b ]
then
   echo "a is not equal to b"
fi
```

The above script will produce the following result –

```
a + b : 30
a - b : -10
a * b : 200
b / a : 2
b % a : 0
a is not equal to b
```

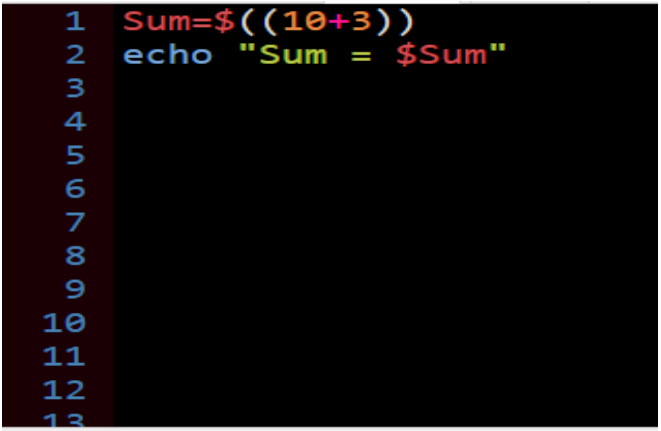
Let's see the examples for the uses of arithmetic operators:

Addition

Code:

```
Sum=$((10+3))
echo "Sum = $Sum"
```

Output:

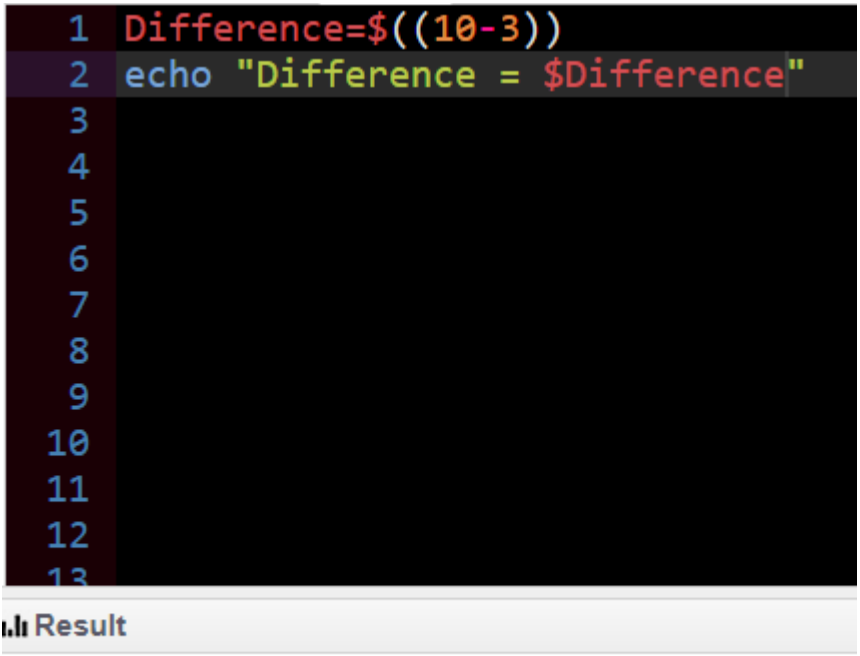


Subtraction

Code:

```
Difference=$((10-3))
echo "Difference = $Difference"
```

Output:



Multiplication

Code:

```
Product=$((10*3))
echo "Product = $Product"
```

Output:

Result

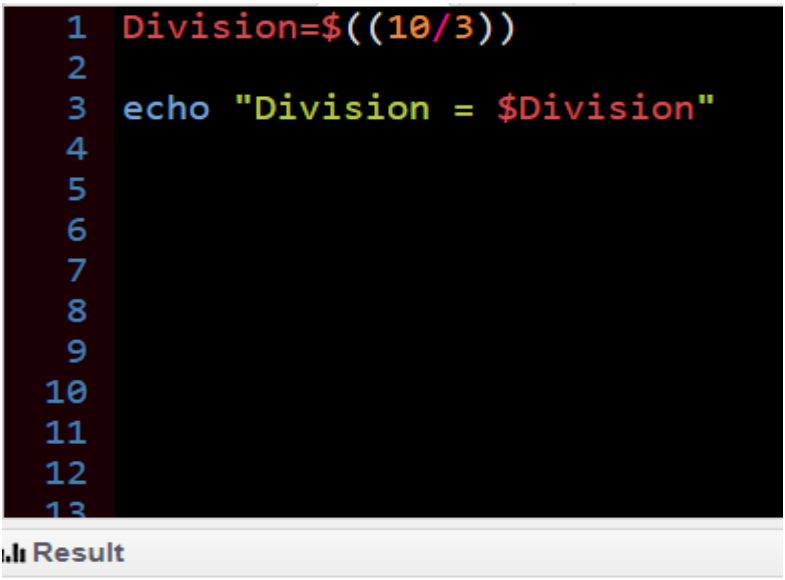
```
$bash -f main.sh
Product = 30
```

Division

Code:

```
Division=$((10/3))
echo "Division = $Division"
```

Output:



Modulo

Code:

```
Modulo=$((10%3))
echo "Modulo = $Modulo"
```

Output:

```
1  Modulo=$((10%3))
2
3  echo "Modulo = $Modulo"
4
5
6
7
8
9
10
11
12
13
```

Result

```
$bash -f main.sh
Modulo = 1
```

hitbullseye

Exponentiation

Code:

```
Exponent=$((10**2))
echo "Exponent = $Exponent"
```

Output:

Result

```
$bash -f main.sh
Exponent = 100
```

Example to show use of all the operators in a single code

Code:

```
x=10
y=20
echo "x=10, y=5"
echo "Addition of x and y"
echo $(( $x + $y ))
echo "Subtraction of x and y"
echo $(( $x - $y ))
echo "Multiplication of x and y"
echo $(( $x * $y ))
echo "Division of x by y"
echo $(( $x / $y ))
echo "Exponentiation of x,y"
echo $(( $x ** $y ))
echo "Modular Division of x,y"
echo $(( $x % $y ))
echo "Incrementing x by 10, then x= "
(( x += 10 ))
echo $x
echo "Decrementing x by 15, then x= "
(( x -= 15 ))
echo $x
```

```
echo "Multiply of x by 2, then x="
(( x *= 2 ))
echo $x
echo "Dividing x by 5, x= "
(( x /= 5 ))
echo $x
echo "Remainder of Dividing x by 5, x="
(( x %= 5 ))
echo $x
```

Output:

```
$bash -f main.sh
x=10, y=5
Addition of x and y
30
Subtraction of x and y
-10
Multiplication of x and y
200
Division of x by y
0
Exponentiation of x,y
77662796314522419 20
Modular Division of x,y
10
Incrementing x by 10, then x=
20
Decrementing x by 15, then x=
5
Multiply of x by 2, then x=
10
Dividing x by 5, x=
2
Remainder of Dividing x by 5, x=
2
```

Interrupt processing:

Imagine that you are watching TV or doing something. Suddenly you heard someone's voice which is like your Crush's voice. What will happen next? That's it, you are interrupted!! You will be very happy. Then stop your work whatever you are doing now and go outside to see him/her. Similar to us, Linux also stops its current work and is distracted because of interrupts, and then it will handle them. In Linux, interrupt signals are the distraction that diverts the processor to a new activity outside of the normal flow of execution. This new activity is called interrupt handler or interrupt service routine (ISR) and that distraction is Interrupts.

What will happen when the interruption comes?

An interrupt is produced by electronic signals from hardware devices and directed into input pins on an interrupt controller (a simple chip that multiplexes multiple interrupt lines into a single line to the processor). These are the process that will be done by the kernel.

1. Upon receiving an interrupt, the interrupt controller sends a signal to the processor.

2. The processor detects this signal and interrupts its current execution to handle the interrupt.
3. The processor can then notify the operating system that an interrupt has occurred, and the operating system can handle the interrupt appropriately.

Different devices are associated with different interrupts using a unique value associated with each interrupt. This enables the operating system to differentiate between interrupts and to know which hardware device caused such an interrupt. In turn, the operating system can service each interrupt with its corresponding handler. Interrupt handling is amongst the most sensitive tasks performed by the kernel and it must satisfy the following:

1. Hardware devices generate interrupts asynchronously (with respect to the processor clock). That means interrupts can come anytime.
2. Because interrupts can come at any time, the kernel might be handling one of them while another one (of a different type) occurs.
3. Some critical regions exist inside the kernel code where interrupts must be disabled. Such critical regions must be limited as much as possible.

Interrupts and Exceptions

Exceptions are often discussed at the same time as interrupts. Unlike interrupts, exceptions occur synchronously with respect to the processor clock; they are often called synchronous interrupts. Exceptions are produced by the processor while executing instructions either in response to a programming error (e.g. divide by zero) or abnormal conditions that must be handled by the kernel (e.g. a page fault). Because many processor architectures handle exceptions in a similar manner to interrupts, the kernel infrastructure for handling the two is similar.

Simple definitions of the two:

Interrupts – asynchronous interrupts generated by hardware.

System calls (one type of exception) on the x86 architecture are implemented by the issuance of a software interrupt, which traps the kernel and causes the execution of a special system call handler. Interrupts work in a similar way, except hardware (not software) issues interrupts.

There is a further classification of interrupts and exceptions.

Interrupts

Maskable – All Interrupt Requests (IRQs) issued by I/O devices give rise to maskable interrupts. A maskable interrupt can be in two states: masked or unmasked; a masked interrupt is ignored by the control unit as long as it remains masked.

Non-maskable – Only a few critical events (such as hardware failures) give rise to nonmaskable interrupts. Non-maskable interrupts are always recognized by the CPU.

Exceptions

Falts – Like Divide by zero, Page Fault, Segmentation Fault.

Traps – Reported immediately following the execution of the trapping instruction. Like Breakpoints.

Aborts – Aborts are used to report severe errors, such as hardware failures and invalid or inconsistent values in system tables.

For a device's each interrupt, its device driver must register an interrupt handler.

Interrupt handler

An interrupt handler or interrupt service routine (ISR) is the function that the kernel runs in response to a specific interrupt:

1. Each device that generates interrupts has an associated interrupt handler.
2. The interrupt handler for a device is part of the device's driver (the kernel code that manages the device).

In Linux, interrupt handlers are normal C functions, which match a specific prototype.