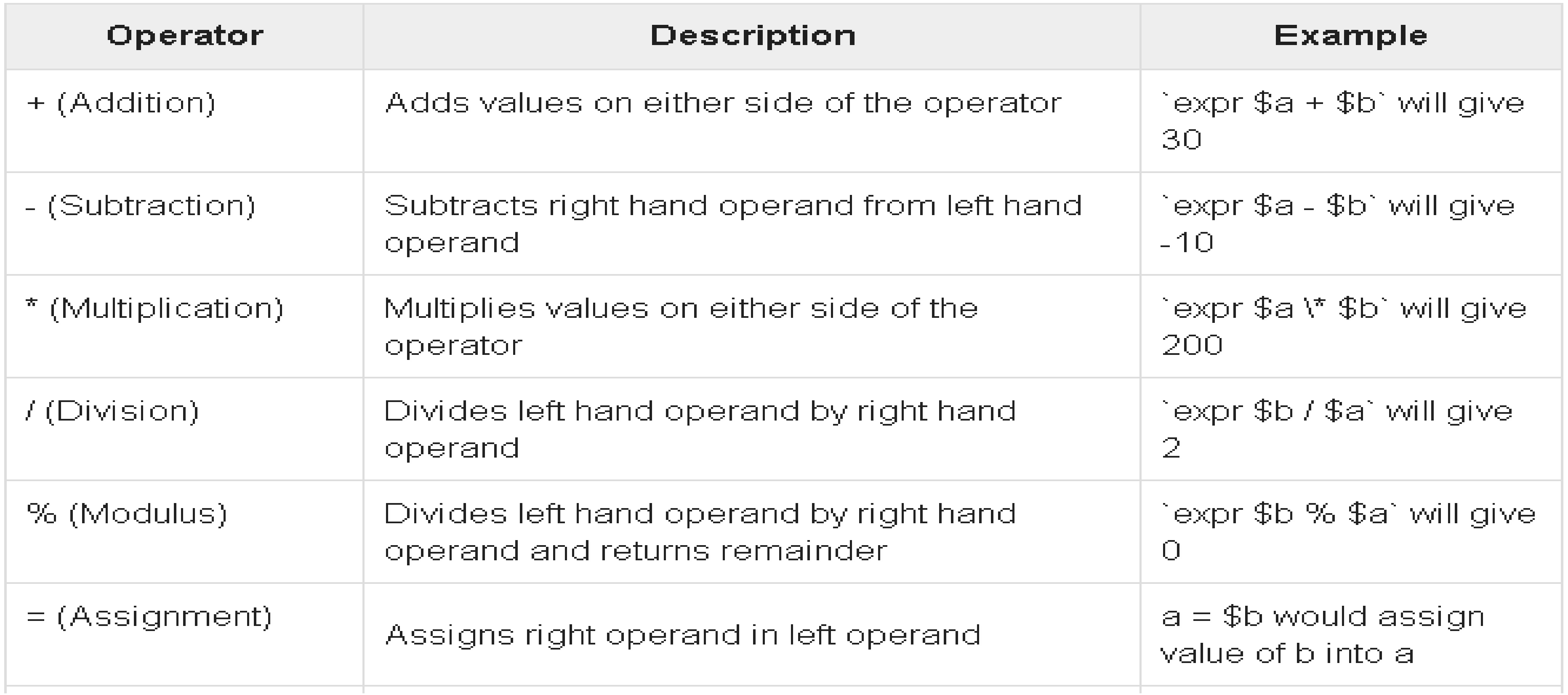
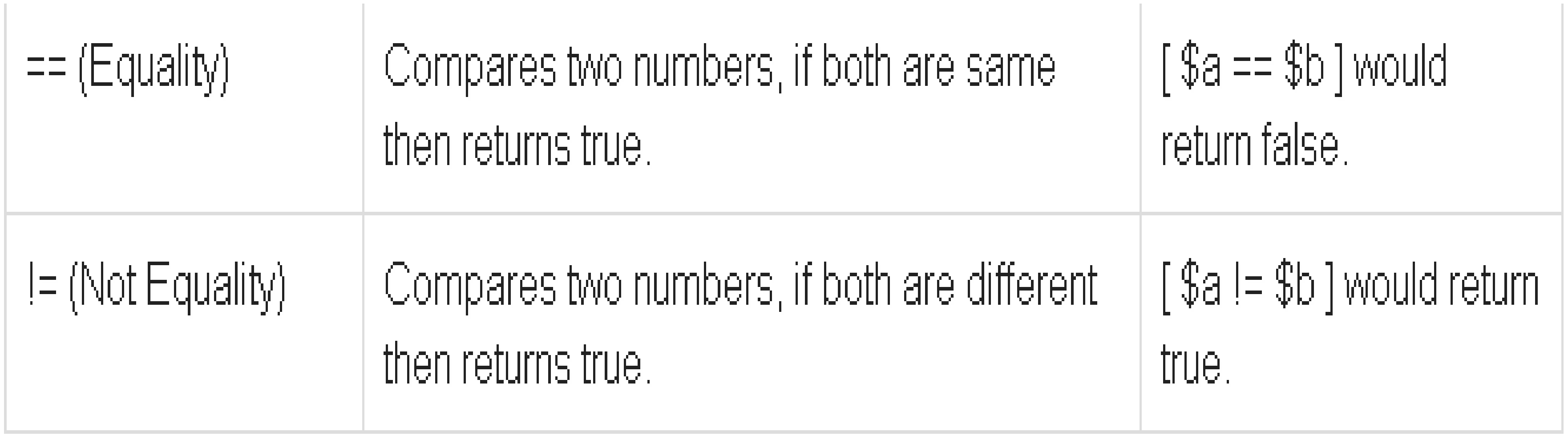
Arithmetic

- Assume variable **a** holds 10 and variable **b** holds 20 then



Operator	Description	Example
+ (Addition)	Adds values on either side of the operator	\`expr $a + $b\` will give 30
- (Subtraction)	Subtracts right hand operand from left hand operand	\`expr $a - $b\` will give -10
* (Multiplication)	Multiplies values on either side of the operator	\`expr $a \* $b\` will give 200
/ (Division)	Divides left hand operand by right hand operand	\`expr $b / $a\` will give 2
% (Modulus)	Divides left hand operand by right hand operand and returns remainder	\`expr $b % $a\` will give 0
= (Assignment)	Assigns right operand in left operand	a = $b would assign value of b into a



== (Equality)	Compares two numbers, if both are same then returns true.	[$a == $b] would return false.
!= (Not Equality)	Compares two numbers, if both are different then returns true.	[$a != $b] would return true.

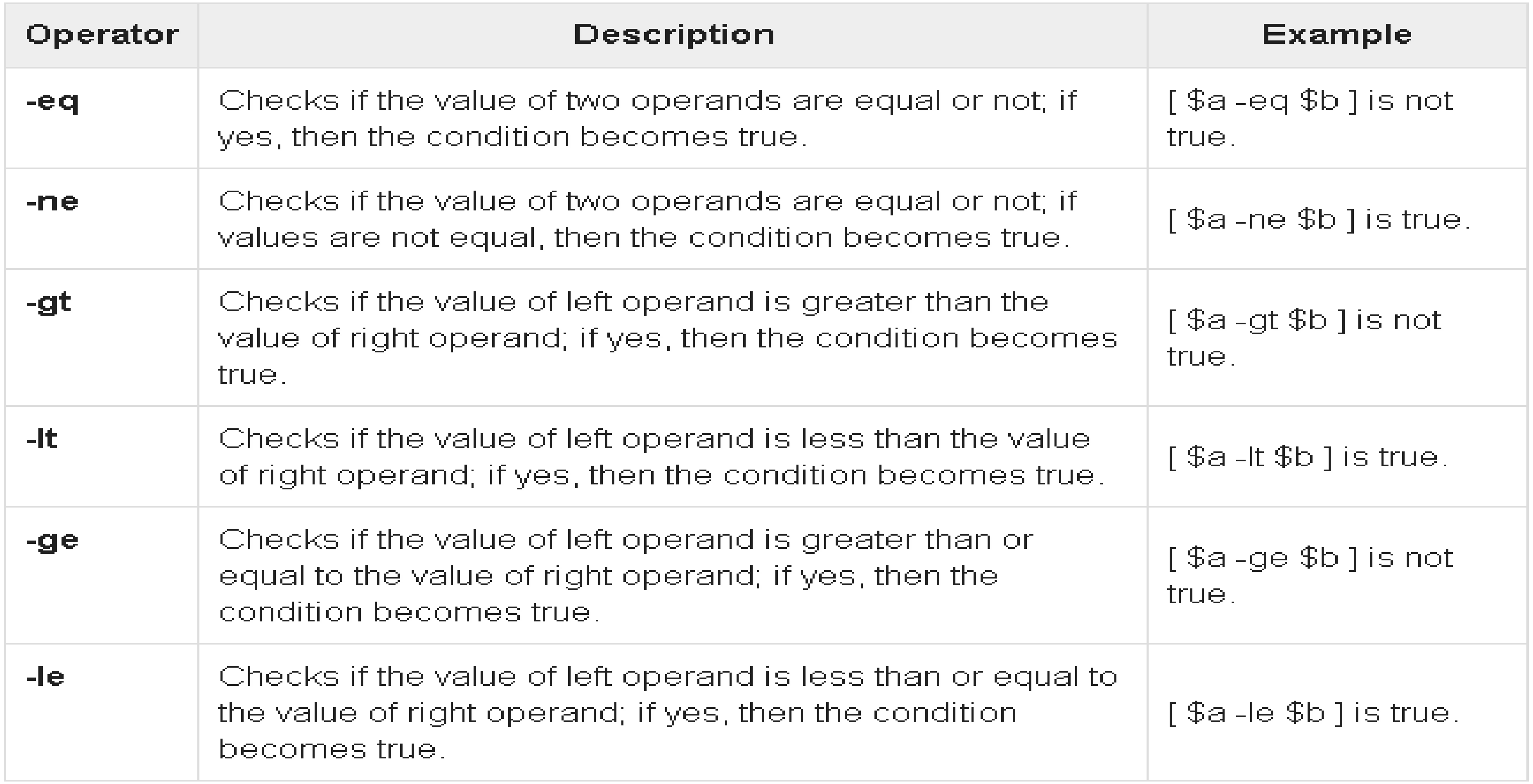
It is very important to understand that all the conditional expressions should be inside square braces with spaces around them, for example [$a == $b] is correct whereas, [$a==$b] is incorrect.

All the arithmetical calculations are done using long integers.

```
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"
```

Relational Operators

- Assume variable **a** holds 10 and variable **b** holds 20 then



Operator	Description	Example
-eq	Checks if the value of two operands are equal or not; if yes, then the condition becomes true.	[$a -eq $b] is not true.
-ne	Checks if the value of two operands are equal or not; if values are not equal, then the condition becomes true.	[$a -ne $b] is true.
-gt	Checks if the value of left operand is greater than the value of right operand; if yes, then the condition becomes true.	[$a -gt $b] is not true.
-lt	Checks if the value of left operand is less than the value of right operand; if yes, then the condition becomes true.	[$a -lt $b] is true.
-ge	Checks if the value of left operand is greater than or equal to the value of right operand; if yes, then the condition becomes true.	[$a -ge $b] is not true.
-le	Checks if the value of left operand is less than or equal to the value of right operand; if yes, then the condition becomes true.	[$a -le $b] is true.

- It is very important to understand that all the conditional expressions should be placed inside square braces with spaces around them

- **Using the *Bash Arithmetic Expansion***
- to evaluate arithmetic expressions with integers in Bash is to use the Arithmetic Expansion capability of the shell. The builtin shell expansion allows you to use the parentheses ((...)) to do math calculations.

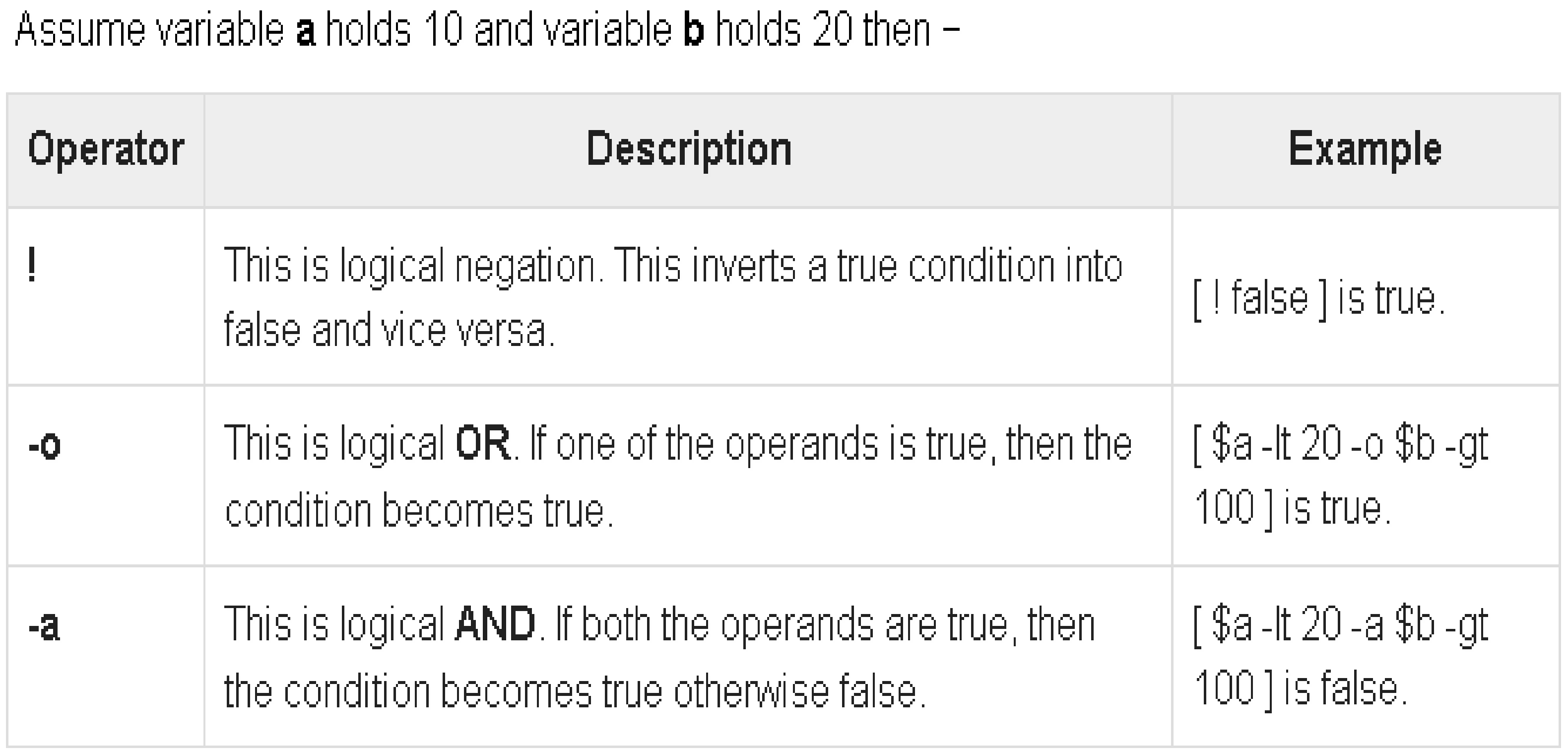
- The format for the Bash arithmetic expansion is $((arithmetic expression)). The shell expansion will return the result of the latest expression given.

Control Structures

```
if [ expression ]
then
    command1
else
    command2
fi
```

-
  ```
  if [ expression ]
  then
        command1
  elif [  expression ]
  then
        command2
   elif [  expression ]
   then
        command3
    else
        command4
   fi
  ```

Logical Operator



Assume variable **a** holds 10 and variable **b** holds 20 then −

Operator	Description	Example
!	This is logical negation. This inverts a true condition into false and vice versa.	[! false] is true.
-o	This is logical **OR**. If one of the operands is true, then the condition becomes true.	[$a -lt 20 -o $b -gt 100] is true.
-a	This is logical **AND**. If both the operands are true, then the condition becomes true otherwise false.	[$a -lt 20 -a $b -gt 100] is false.

```
while [ expression  ]
do
  Statement(s) to be executed if expression is true
done
```

```
i=0
while [ $i –le 5 ]
do
echo “$i”
i=`expr$i + 1`
done
```

```
i=0
While(($i < 5 ))
do
echo “$i”
i=`expr$i + 1`
done
```

Nesting Loops

- All the loops support nesting concept which means you can put one loop inside another similar one or different loops. This nesting can go up to unlimited number of times based on your requirement.

- Here is an example of nesting while loop. The other loops can be nested based on the programming requirement in a similar way −

Nesting while Loops

It is possible to use a while loop as part of the body of another while loop

.Syntax

```
while command1 ; # this is loop1, the outer loop
do
  Statement(s) to be executed if command1 is true

  while command2 ; # this is loop2, the inner loop
  do
    Statement(s) to be executed if command2 is true
  done

  Statement(s) to be executed if command1 is true
done
```