SHELL PROGRAMMING

Read-only Variables

- #!/bin/bash

  NAME="Unix"

  readonly NAME

  NAME="Linux"

  Output:

  ./Aritmatic.sh: line 4: NAME: readonly variable

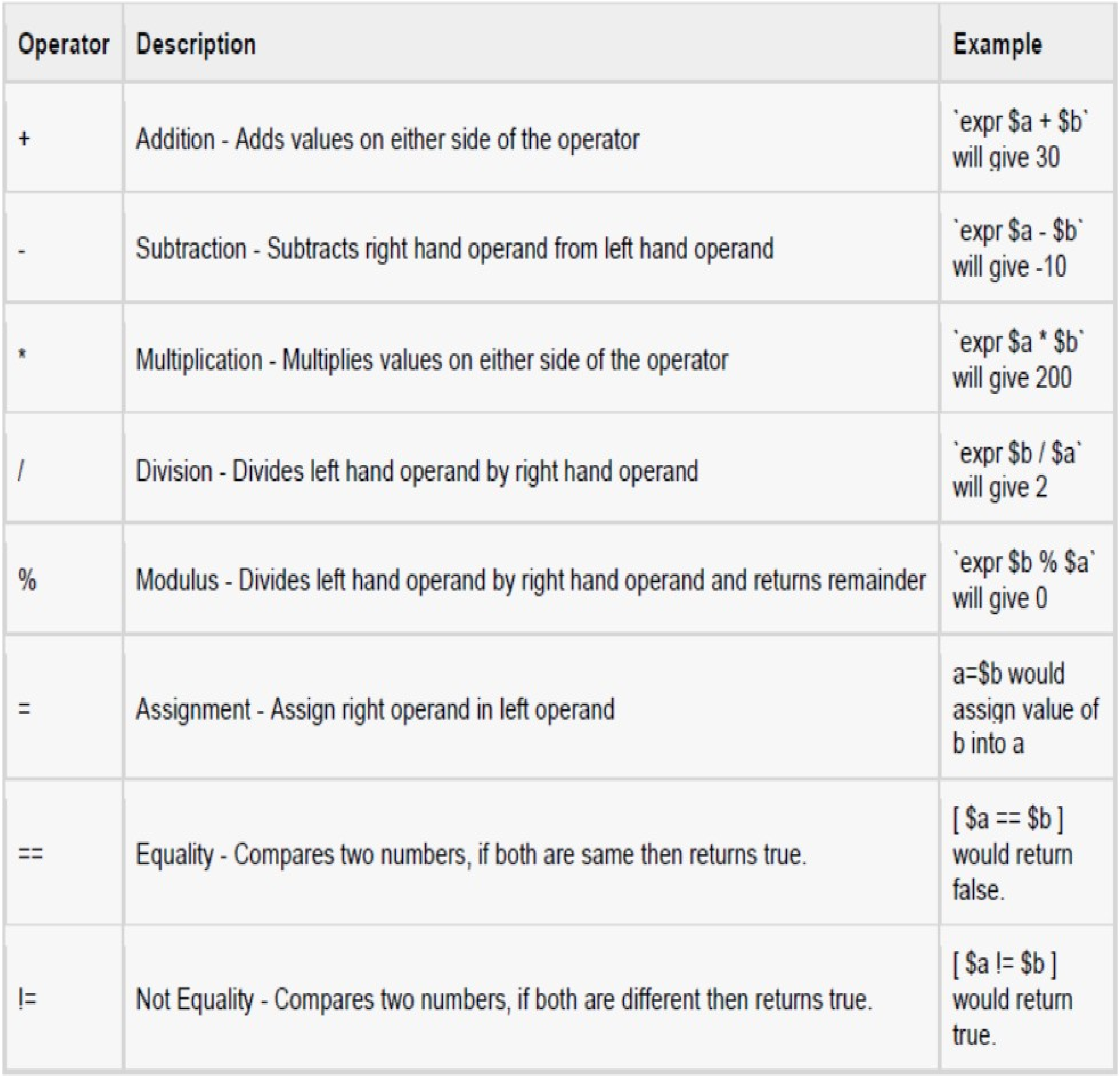
Unsetting Variables

- Unsetting or deleting a variable tells the shell to remove the variable from the list of variables that it tracks. Once you unset a variable, you would not be able to access stored value in the variable.
- the syntax to unset a defined variable using the unset command:

  ```
  unset variable_name
  ```

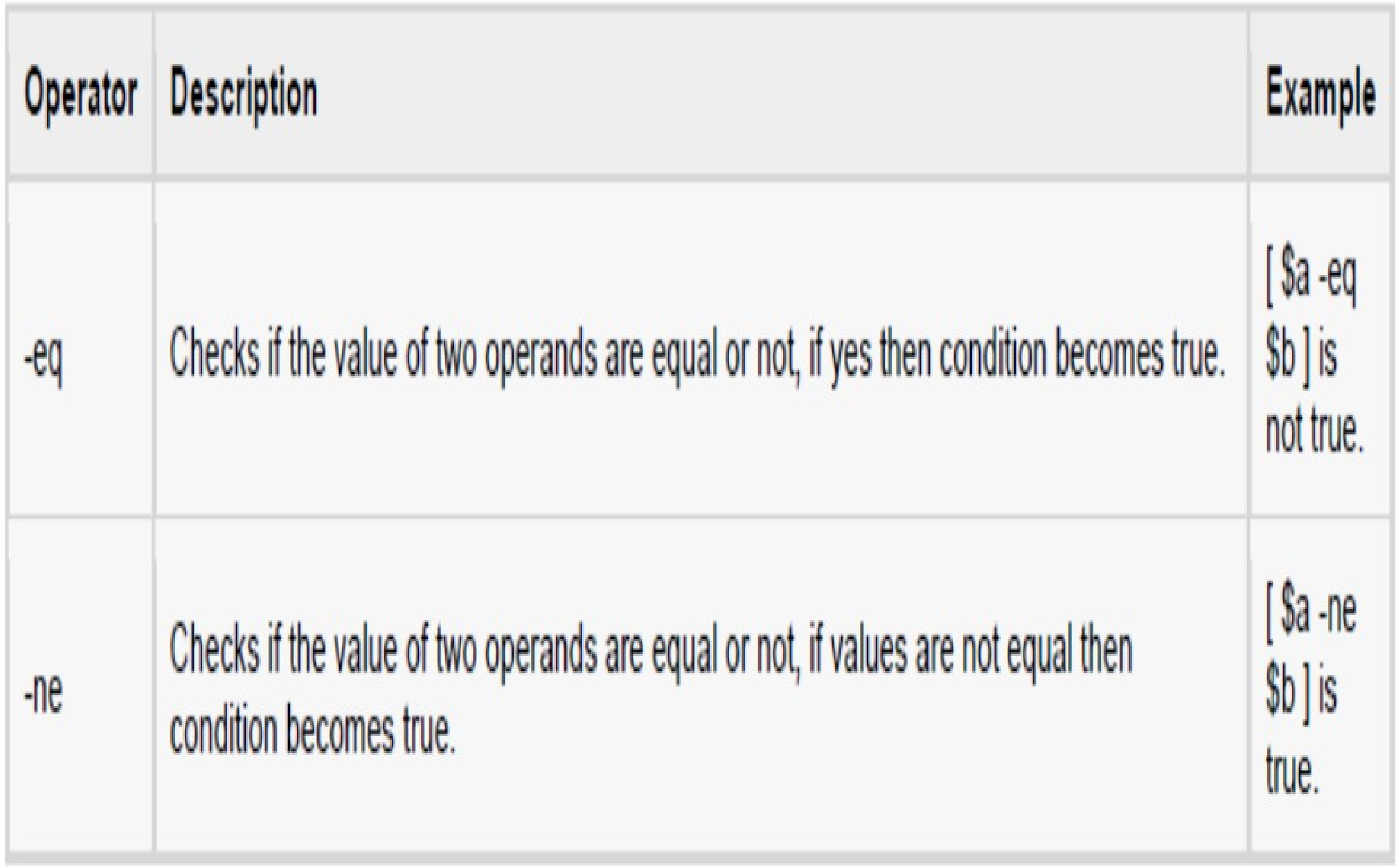
-
  ```
  #!/bin/bash
  NAME="Unix"
  unset NAME
  echo $NAME
  ```

- Arithmetic Operators.
- Relational Operators.
- Boolean Operators.
- String Operators.
- File Test Operators.



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 % $a` will give 0
=	Assignment - Assign right operand in left operand	a=$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.

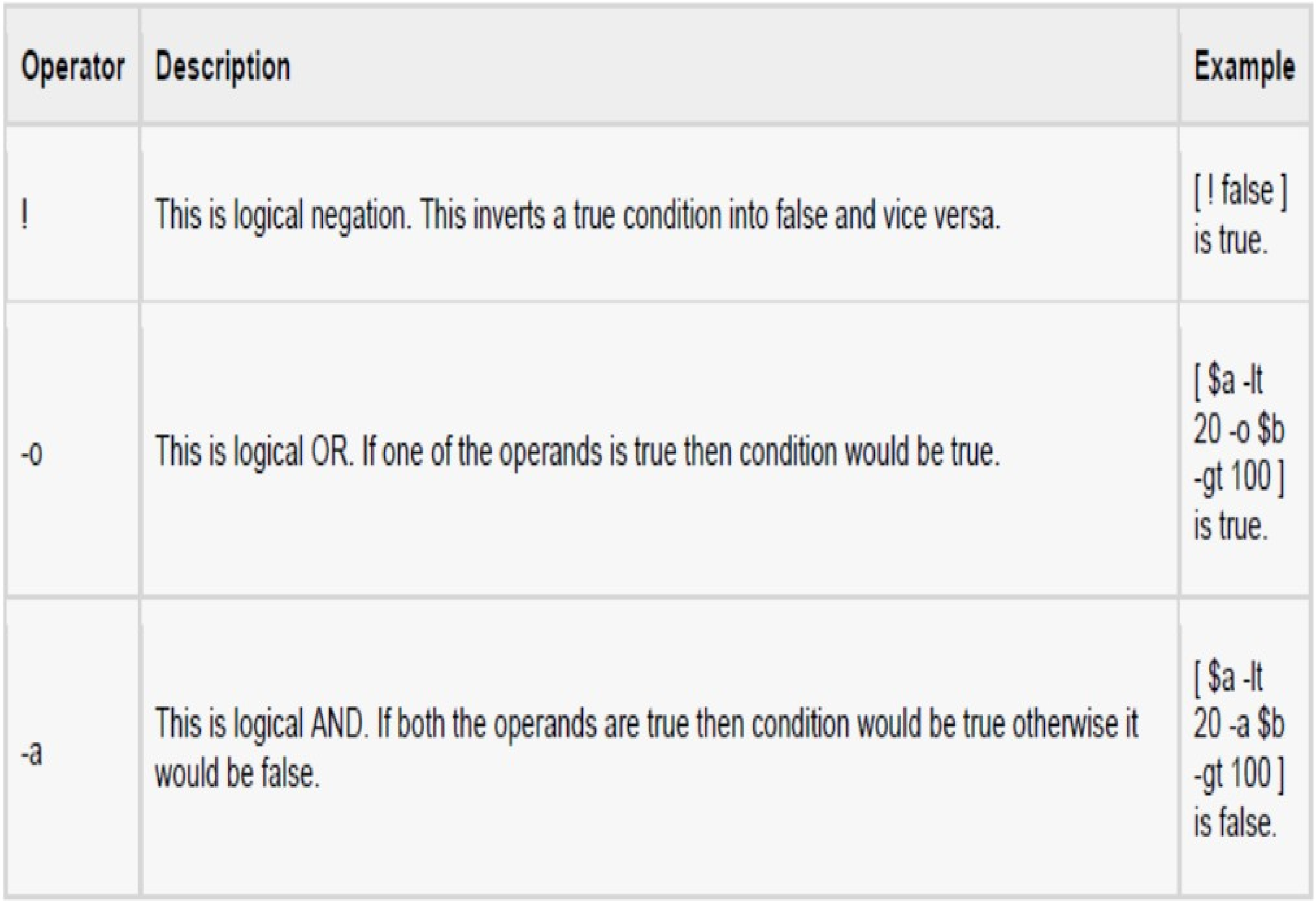
Relational Operators



Operator	Description	Example
-eq	Checks if the value of two operands are equal or not, if yes then condition becomes true.	[$a -eq $b] is not true.
-ne	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	[$a -ne $b] is true.

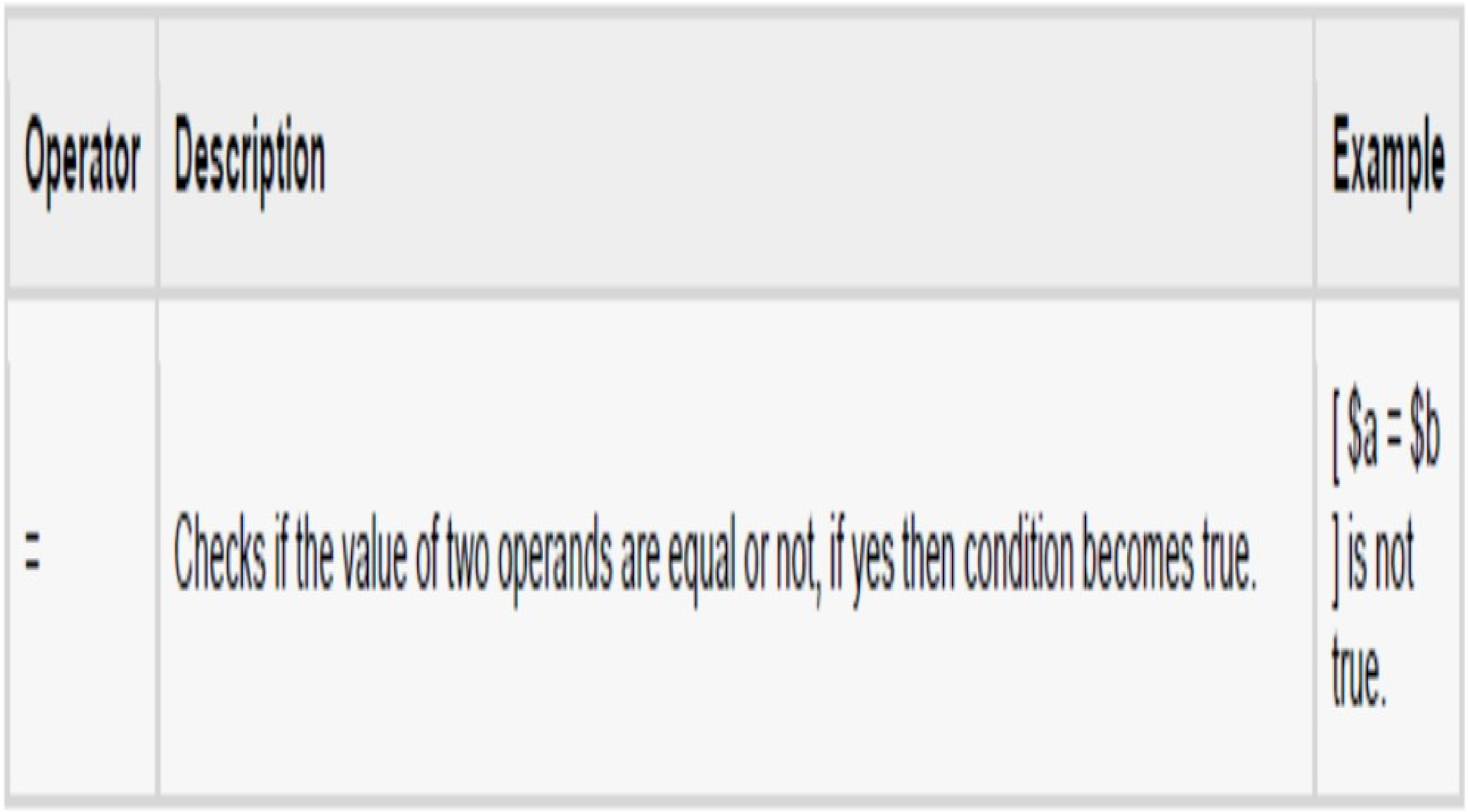
-gt	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	[$a -gt $b] is not true.
-lt	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	[$a -lt $b] is true.
-ge	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	[$a -ge $b] is not true.
-le	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	[$a -le $b] is true.

Boolean Operators



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 condition would be true.	[$a -lt 20 -o $b -gt 100] is true.
-a	This is logical AND. If both the operands are true then condition would be true otherwise it would be false.	[$a -lt 20 -a $b -gt 100] is false.

String Operators



Operator	Description	Example
=	Checks if the value of two operands are equal or not, if yes then condition becomes true.	[$a = $b] is not true.

!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	[$a != $b] is true.
-z	Checks if the given string operand size is zero. If it is zero length then it returns true.	[-z $a] is not true.
-n	Checks if the given string operand size is non-zero. If it is non-zero length then it returns true.	[-z $a] is not false.
Str	Check if str is not the empty string. If it is empty then it returns false.	[$a] is not false.

The if...else statements:

- If else statements are useful decision making statements which can be used to select an option from a given setof options.
- The if...fi statement is the fundamental control statement that allows Shell to make decisions and execute statements conditionally.
- Syntax:

```
if [ expression ]

then

Statement(s) to be executed if expression is true

fi
```

String Comparisons

=	equal
!=	not equal
<	less then
>	greater then
-n s1	string s1 is not empty
-z s1	string s1 is empty

-
  ```
  #!/bin/bash
  a=10
  b=20
  if [ $a == $b ]
  then
  echo "a is equal to b"
  fi
  if [ $a != $b ]
  then
  echo "a is not equal to b"
  fi
  ```

if...else...fi statement

- Syntax:

```
if [ expression ]
then
Statement(s) to be executed if
expression is true
else
Statement(s) to be executed if
expression is not true
fi
```

Bash Comparisons

```
#!/bin/bash
NUM1=2
NUM2=2
if [ $NUM1 -eq $NUM2 ]; then
echo "Both Values are equal"
else
echo "Values are NOT equal"
fi
```

if...elif...else...fi statement

- The if...elif...fi statement is the one level advance form of control statement that allows Shell to make correct decision out of several conditions.

- Syntax

```
if [ expression 1 ]
then
Statement(s) to be executed if expression 1 is true
elif [ expression 2 ]
then
Statement(s) to be executed if expression 2 is true
elif [ expression 3 ]
then
Statement(s) to be executed if expression 3 is true
else
Statement(s) to be executed if no expression is true
fi
```

-
  ```
  a=10
  b=20
  if [ $a == $b ]
  then
  echo "a is equal to b"
  elif [ $a -gt $b ]
  then
  echo "a is greater than b"
  elif [ $a -lt $b ]
  then
  echo "a is less than b"
  else
  echo "None of the condition met"
  fi
  ```

- bash shell : the syntax of array initialization:

  array_name=(value1 ... valuen)

- Accessing Array Values

  ${array_name[index]}

  Ex:

```
#!/bin/bash
NAME=(Unix POP Maths DS DDCO)
echo "First Subject: ${NAME[0]}"
echo "ALL SUBJECT: ${NAME[@]}"
```

For Loop

-
  ```
  for varname in list
  do
   command1
   command2
   ..
  done
  ```

-
  ```
  #!/bin/bash
  i=1
  for day in Mon Tue Wed Thu Fri
  do
  echo "Weekday $((i++)) : $day"
  done
  ```

- for ((expr1; expr2; expr3))

```
for (( expr1; expr2; expr3 ))
do
 command1
 command2
..
done
```

-
  ```
  #!/bin/bash
  echo "Enter the vlaue of n"
  read n
  for ((i=0; i<$n; i++))
  do
   echo "The value is $i"
  done
  ```

Infinite Bash for loop

-
  ```
  #!/bin/bash
  i=1
  for (( ; ; ))
  do
     sleep $i
     echo "Number: $((i++))"
  done
  ```

-
  ```
  #!/bin/bash
  for ((i=1, j=10; i <= 5 ; i++, j=j+5))
  do
   echo "Number $i: $j"
  done
  ```

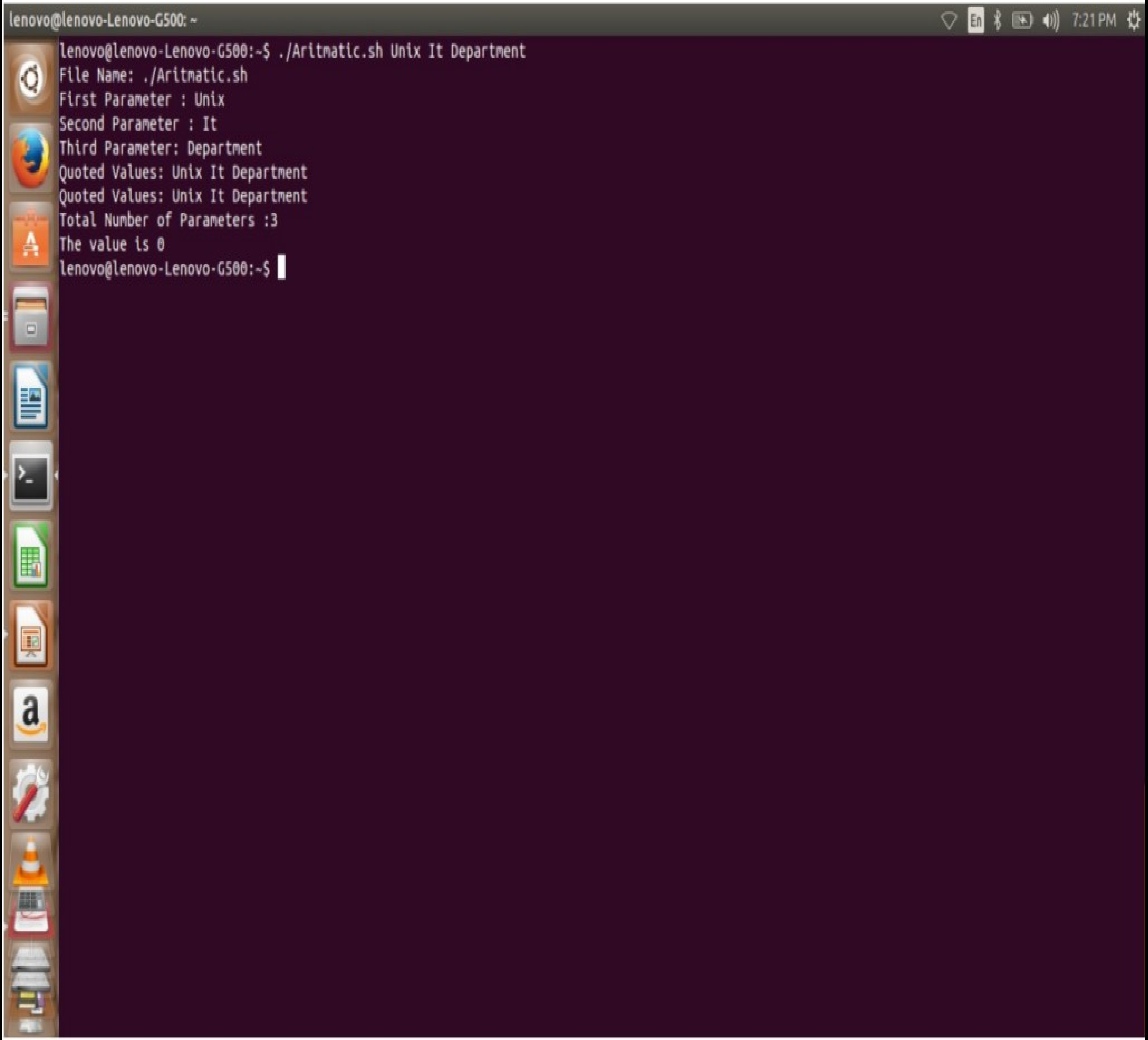
Reading User Input

```
#!/bin/bash
echo -e "Please enter two words? "
read word1 word2
echo "Here is your input: \"$word1\" \"$word2\""
echo -e "How do you feel about bash scripting? "
# read comd stores a reply into the default build-in variable
$REPLY
read
echo "You said $REPLY, I'm glad to hear that! "
echo -e "What are your favorite colours ? "
# -a makes read command to read into an array
read -a colours
echo "My favorite colours are also ${colours[0]}, ${colours[1]}
and ${colours[2]}:-)"
```

Variable	Description
$0	The filename of the current script.
$n	These variables correspond to the arguments with which a script was invoked. Here n is a positive decimal number corresponding to the position of an argument (the first argument is $1, the second argument is $2, and so on).
$#	The number of arguments supplied to a script.
$*	All the arguments are double quoted. If a script receives two arguments, $* is equivalent to $1 $2.
$@	All the arguments are individually double quoted. If a script receives two arguments, $@ is equivalent to $1 $2.
$?	The exit status of the last command executed.
$$	The process number of the current shell. For shell scripts, this is the process ID under which they are executing.
$!	The process number of the last background command.

Command-Line Arguments

-
  ```
  #!/bin/bash
  echo "File Name: $0"
  echo "First Parameter : $1"
  echo "Second Parameter : $2"
  echo "Third Parameter: $3"
  echo "Quoted Values: $@"
  echo "Quoted Values: $*"
  echo "Total Number of Parameters :$#"
  echo "The value is $?"
  ```



```
lenovo@lenovo-Lenovo-G500:~$ ./Aritmatic.sh Unix It Department
File Name: ./Aritmatic.sh
First Parameter : Unix
Second Parameter : It
Third Parameter: Department
Quoted Values: Unix It Department
Quoted Values: Unix It Department
Total Number of Parameters :3
The value is 0
lenovo@lenovo-Lenovo-G500:~$
```

FOR LOOP

-
  ```
  #!/bin/bash
  for f in $( ls /var/ )
  do
  echo $f
  done
  ```

The case...esac Statement

- Syntax:

```
case word in
pattern1)
Statement(s) to be executed if pattern1 matches
;;
pattern2)
Statement(s) to be executed if pattern2 matches
;;
pattern3)
Statement(s) to be executed if pattern3 matches
;;
esac
```

-
  ```
  #!/bin/bash
  echo "1.Addition"  echo "2.Subtraction" echo "3.Multiplication"
  echo "4.Division"
  echo "Enter your choice" read ch
  case $ch in
  1)echo "enter two numbers"
    read x; read y;
    z=`expr $x + $y`
    echo "Addition of two number is $z"
    ;;
  2)echo "enter two numbers"
    read x; read y;
    z=`expr $x - $y`
    echo "Subtraction of two number is $z"
    ;;
  ```

```
3)echo "enter two numbers"
  read x; read y;
  z=`expr $x \* $y`
  echo "Multiplicationof two number is $z"
  ;;
4)echo "enter two numbers"
  read x; read y;
  z=`expr $x / $y`
  echo "Division of two number is $z"
  ;;
esac
```

Bash File Testing

-b filename	Block special file
-c filename	Special character file
-d directoryname	Check for directory existence
-e filename	Check for file existence
-f filename	Check for regular file existence not a directory
-G filename	Check if file exists and is owned by effective group ID.
-g filename	true if file exists and is set-group-id.
-k filename	Sticky bit
-L filename	Symbolic link

-O filename	True if file exists and is owned by the effective user id.
-r filename	Check if file is a readable
-S filename	Check if file is socket
-s filename	Check if file is nonzero size
-u filename	Check if file set-ser-id bit is set
-w filename	Check if file is writable
-x filename	Check if file is executable

-
  ```
  #!/bin/bash
  directory="Scripting"
  # bash check if directory exists
  if [ -d $directory ]
  then
  echo "Directory exists"
  else
  echo "Directory does not exists"
  fi
  ```

-
  ```
  #!/bin/bash
  file="file1"
  if [ -e $file ]; then
  echo "File exists"
  else
  echo "File does not exists"
  fi
  ```

The while loop

- while command

  do

  Statement(s) to be executed if command is true

  do

-
  ```
  #!/bin/bash
  COUNT=6
  # bash while loop
  while [ $COUNT -gt 0 ]
  do
  echo "Value of count is: $COUNT"
  COUNT=`expr $COUNT - 1`
  done
  ```

Bash quoting with ANSI-C style

- \a alert (bell)
- \e an escape character
- \n newline
- \t horizontal tab
- \\ backslash
- \nnn octal value of characters
- \xnn hexadecimal value of character
- \b backspace
- \f form feed
- \r carriage return
- \v vertical tab
- \` single quote

- #!/bin/bash

  # as a example we have used \n as a new line,

  #\x40 is hex value for @

  #\56 is octal value for .

  echo $'web: www.linuxconfig.org\nemail: web\x40linuxconfig\56org'

until loop

- Syntax:

  until command

  do

  Statement(s) to be executed until command is true

  done

- Shell command is evaluated. If the resulting value is false, given statement(s) are executed

-
  ```
  #!/bin/bash
  a=0
  until [ ! $a -lt 10 ]
  do
  echo $a
  a=`expr $a + 1`
  done
  ```