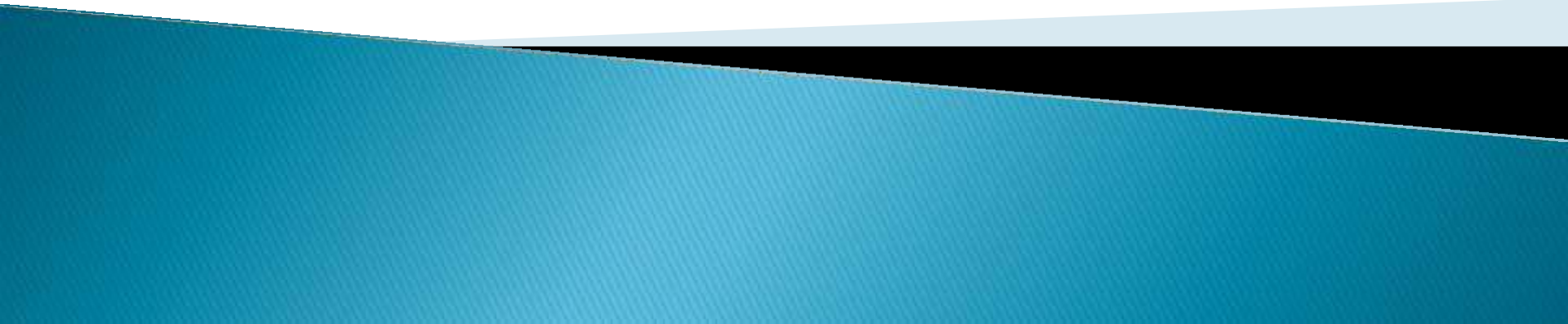


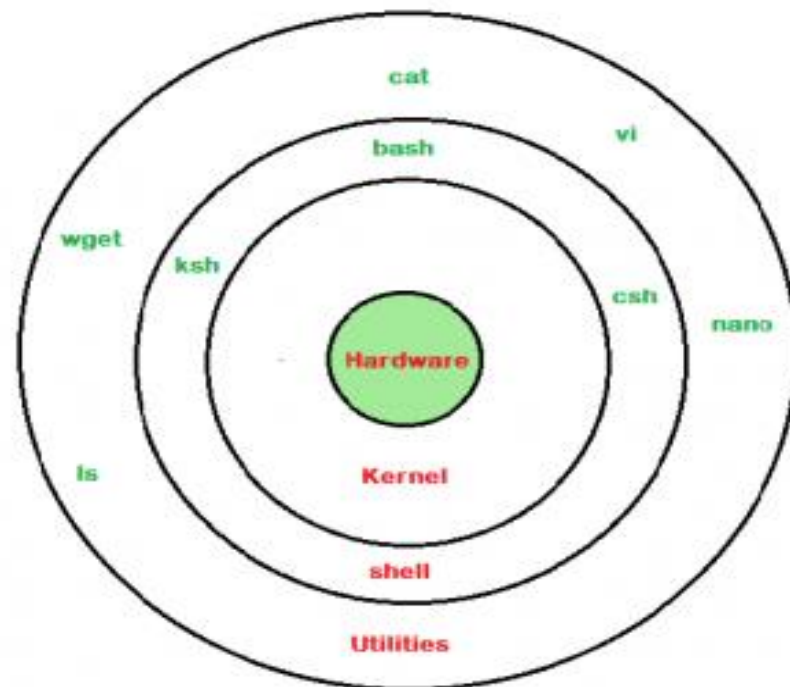
Introduction to LINUX SHELL and SHELL Scripting



What is Shell

A shell is **special user program which provide an interface to user to use operating system services**. Shell accept human readable commands from user and convert them into something which kernel can understand. It is a command language interpreter that execute commands read from input devices such as keyboards or from files. The shell gets started when the user logs in or start the terminal.

LINUX Shell



What is Shell

There are several shells are available for Linux systems like –

[BASH \(Bourne Again SHell\)](#) – It is most widely used shell in Linux systems. It is used as default login shell in Linux systems and in macOS. It can also be installed on Windows OS.

[CSH \(C SHell\)](#) – The C shell's syntax and usage are very similar to the C programming language.

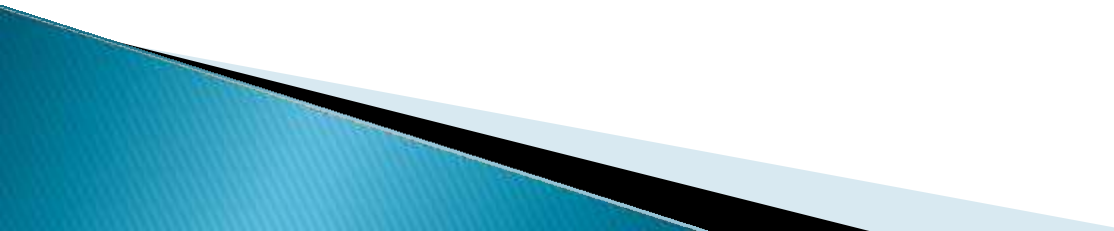
[KSH \(Korn SHell\)](#) – The Korn Shell also was the base for the POSIX Shell standard specifications etc.

Each shell does the same job but understand different commands and provide different built in functions.



What is Terminal

It's a program called a *terminal emulator*. This is a program that opens a window and lets you interact with the shell. There are a bunch of different terminal emulators we can use. Some Linux distributions install several. These might include gnome-terminal, konsole, xterm, rxvt, kvt, nxterm, and eterm.

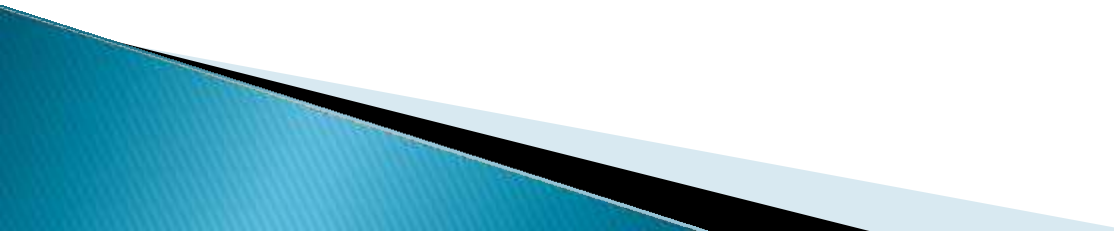


Shell Scripting

Usually shells are interactive that mean, they accept command as input from users and execute them.

However some time we want to execute a bunch of commands routinely, so we have type in all commands each time in terminal.

As shell can also take commands as input from file we can write these commands in a file and can execute them in shell to avoid this repetitive work. These files are called **Shell Scripts** or **Shell Programs**. Each shell script is saved with **.sh** file extension
eg. **myscript.sh**



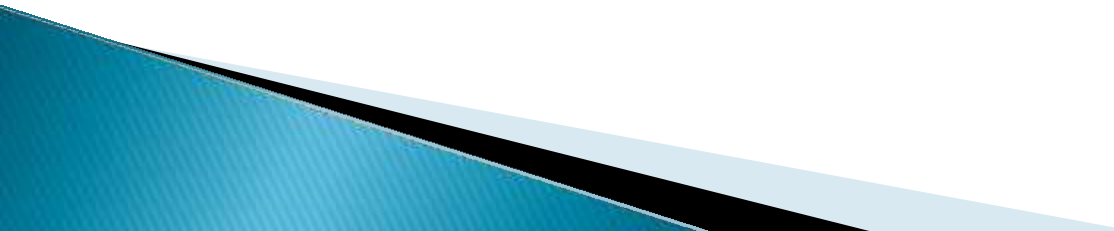
Shell Scripting

A shell script have syntax just like any other programming language. If you have any prior experience with any programming language like Python, C/C++ etc. it would be very easy to get started with it.

A shell script comprises following elements –

- Shell Keywords – if, else, break etc.
- Shell commands – cd, ls, echo, pwd, touch etc.
- Functions

Control flow – if..then..else, case and shell loops etc.

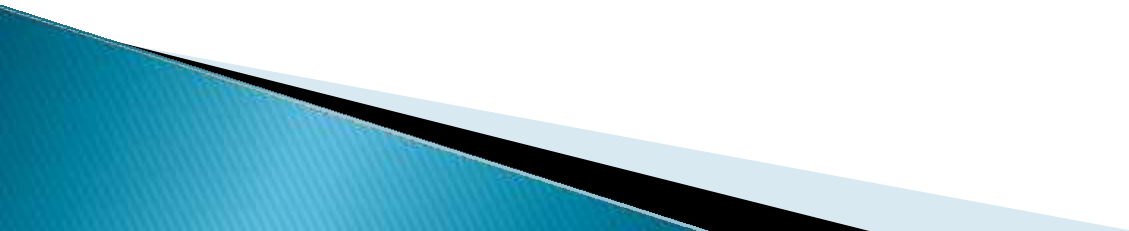


Why do we need shell scripts

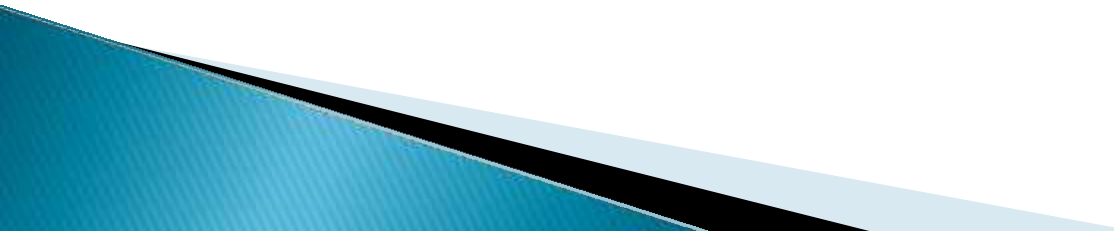
There are many reasons to write shell scripts:

To avoid repetitive work and automation

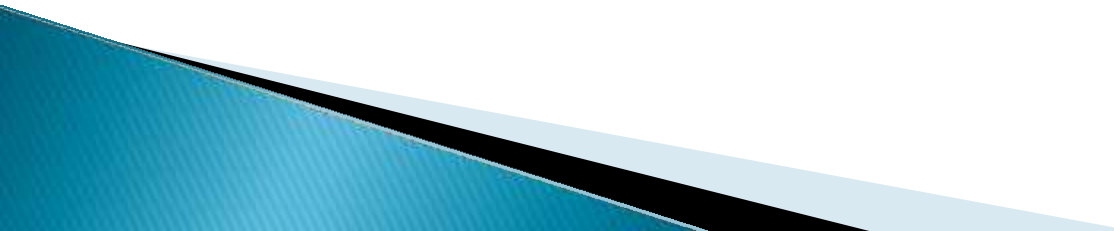
- System admins use shell scripting for routine backups
- System monitoring
- Adding new functionality to the shell etc.



Advantages shell scripts

- The command and syntax are exactly the same as those directly entered in command line, so programmer do not need to switch to entirely different syntax
 - Writing shell scripts are much quicker
 - Quick start
 - Interactive debugging etc.
- 

Disadvantages shell scripts

- Prone to costly errors, a single mistake can change the command which might be harmful
 - Slow execution speed
 - Design flaws within the language syntax or implementation
 - Not well suited for large and complex task
 - Provide minimal data structure unlike other scripting languages. etc
- 

Variables in Shell Scripting

Variables can be assigned in bash with the syntax: `NAME=value`. Note the lack of spaces between the assignment operator `=` and its operands.

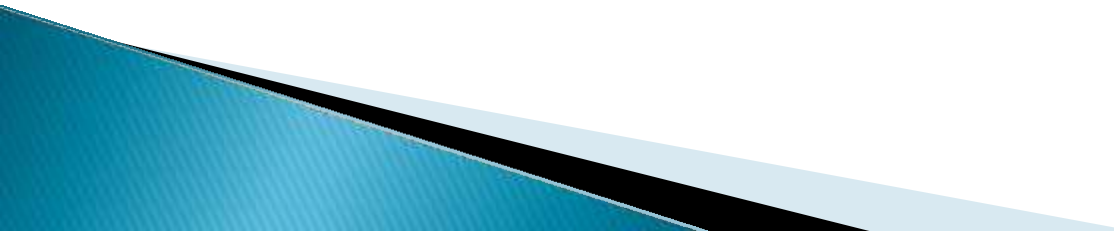
You can retrieve the value of a variable by prepending a `$` to it's name. Getting the value of `NAME` must be done with `$NAME`. This is called variable interpolation.

saima@ubuntu: ~

```
saima@ubuntu:~$ Name = "tux"
No command 'Name' found, did you mean:
  Command 'name' from package 'name' (multiverse)
  Command 'lame' from package 'lame' (universe)
Name: command not found
saima@ubuntu:~$ Name="tux"
saima@ubuntu:~$ echo Name
Name
saima@ubuntu:~$ echo $Name
tux
saima@ubuntu:~$
```

Basic Operators in Shell Scripting

There are **5** basic operators in bash/shell scripting:

- Arithmetic Operators
 - Relational Operators
 - Boolean Operators
 - Bitwise Operators
 - File Test Operators
- 

Arithmetic Operators

These operators are used to perform normal arithmetic/mathematical operations. There are 7 arithmetic operators:

Addition (+): Binary operation used to add two operands.

Subtraction (-): Binary operation used to subtract two operands.

Multiplication (*): Binary operation used to multiply two operands.

Division (/): Binary operation used to divide two operands.

Modulus (%): Binary operation used to find remainder of two operands.

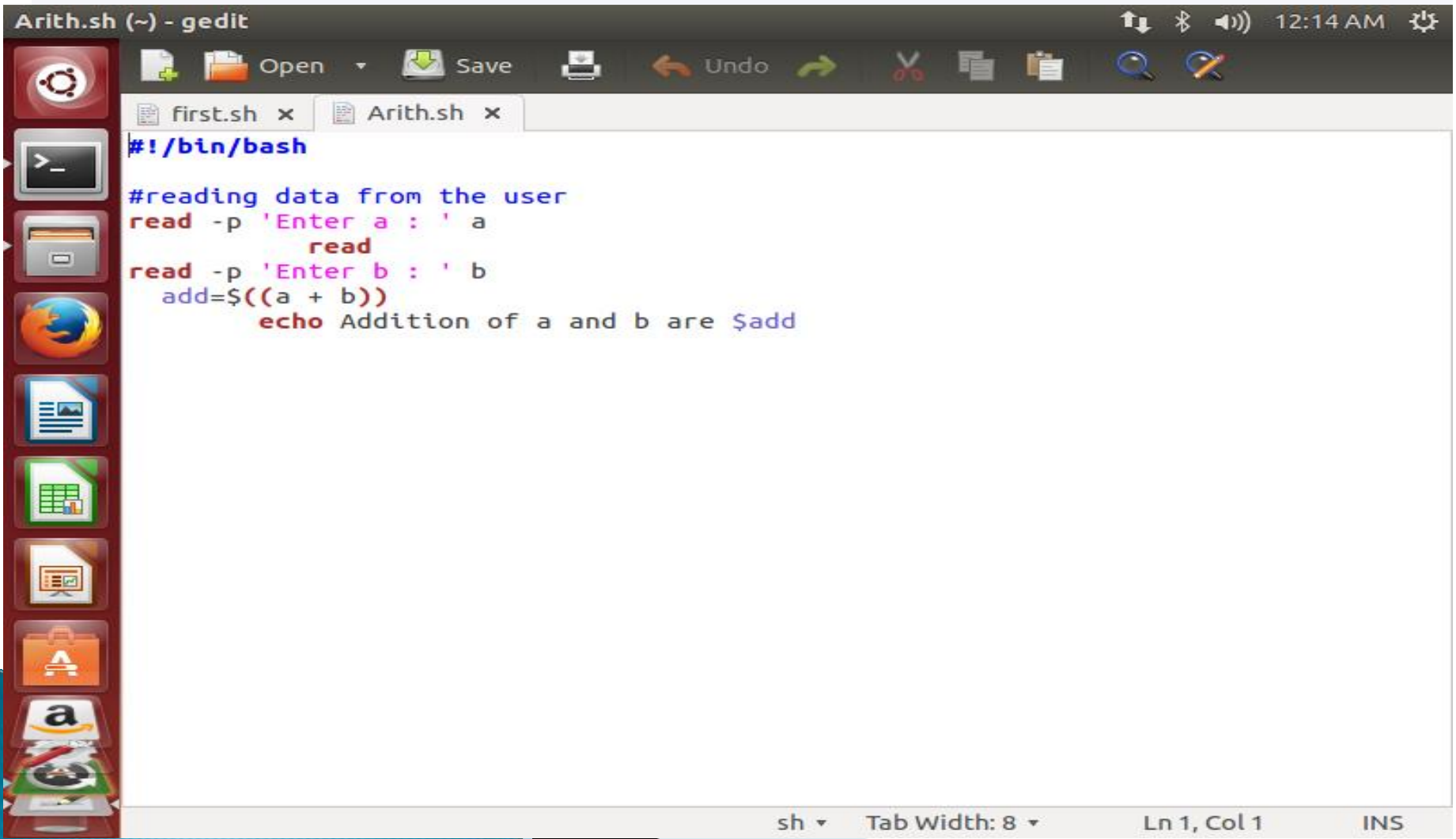
Increment Operator (++): Unary operator used to increase the value of operand by one.

Decrement Operator (--): Unary operator used to decrease the value of a operand by one

Shebang!

Shell scripts typically begin with the **shebang** line:

#!/path/to/interpreter -----> #!/bin/bash



```
Arith.sh (~) - gedit
#!/bin/bash
#reading data from the user
read -p 'Enter a : ' a
    read
read -p 'Enter b : ' b
    add=$((a + b))
    echo Addition of a and b are $add
```

sh Tab Width: 8 Ln 1, Col 1 INS

saima@ubuntu: ~

saima@ubuntu:~\$ chmod +x Arith.sh

saima@ubuntu:~\$./Arith.sh

Enter a : 5


Enter b : 8

Addition of a and b are 13

saima@ubuntu:~\$

Relational Operators

Relational operators are those operators which define the relation between two operands. They give either true or false depending upon the relation. They are of 6 types:

- **'==' Operator:** Double equal to operator compares the two operands. Its returns true is they are equal otherwise returns false.
 - **'!=' Operator:** Not Equal to operator return true if the two operands are not equal otherwise it returns false.
 - **'<' Operator:** Less than operator returns true if first operand is less than second operand otherwise returns false.
 - **'<=' Operator:** Less than or equal to operator returns true if first operand is less than or equal to second operand otherwise returns false
 - **'>' Operator:** Greater than operator return true if the first operand is greater than the second operand otherwise return false.
 - **'>=' Operator:** Greater than or equal to operator returns true if first operand is greater than or equal to second operand otherwise returns false C
- 



first.sh x Arith.sh x re.sh x

`#!/bin/bash``#reading data from the user``read -p 'Enter a : ' a``read -p 'Enter b : ' b``if(($a==$b))``then` `echo 'a is equal to b'``else` `echo 'a is not equal to b'``fi`

saima@ubuntu: ~

```
saima@ubuntu:~$ chmod +x re.sh
```

```
saima@ubuntu:~$ ./re.sh
```

```
Enter a : 4
```

```
Enter b : 5
```

```
a is not equal to b
```

```
saima@ubuntu:~$
```

Lab Assignment 1

Get user input and perform all arithmetic and relational operations.

