Working with Linux

knowledge hut

LEARNING OBJECTIVES

- Learn about the wildcards in Linux
- What are aliases in Linux?
- Learn about some important Linux commands
- Learn about process management in Linux
- Understand what are background process
- Understand the Secure Shell in Linux
- Understand the shell script with some examples

What is a "Wildcard" in Linux?

knowledge hut

What is a "Wildcard" Linux?

- Sometimes users want to run a command on a group of files instead of a single file. For example: Delete all the old logs, list all audio files etc.
- For this purpose, "Wildcard" commands are used.
- Wildcards are patterns that can be used in place of file names and directory names that are used to apply the command on group of files/ directories that share something common in their name.
- Wildcards are used for file names and directory names only.
- Wildcards can be used in conjunction to most Linux commands such as cp, ls, rm,

Types of Wildcards:

There are three major types of wildcards used in Linux:

1. An asterisk (*) –More than one occurrences of any character, including space in between can be matched using this wildcard.
2. Question mark (?) –Single occurrence of any character is matched using this wildcard. So, it matches exactly one character.
3. Bracketed characters ([ ]) – Character to be matched is enclosed in the square brackets. Different types of characters (alphanumeric characters) can be used.

The "*" Wildcard:

'*' can replace any number of characters in a file/ directory name. It matches zero or more characters

For example:

1. `ls *.txt` : This command lists all the text files.
2. `ls X*` : This command lists all files starting with letter X.
3. `rm X*.txt` : This command removes all text files starting with X.
4. `rm *.js` : This command will remove all .js files and directories.

The "?" Wildcard:

Matches exactly one character

For example:

1. **`ls ?.txt:`** If you want exactly one letter before .txt then use this command.
2. **`rm X?:`** To match all the two letter words starting with the letter X.
3. **`rm X?.txt:`** To match all the two letter words starting with the letter X and ending with ".txt".

The "[]" Wildcard:

Matches only the characters present inside the brackets. This is used to provide more restrictions for matching than the "?". "[]" is basically a character class.

For example:

1. `ls results-[0-9][0-9].log` : This command lists the file named results-00.log to results-99.log.
2. `ls ca[nt]*` : This command will list the words starting with letter "ca" followed by either "n" or "t" and followed by any number of characters. For example, can, cannot, catch etc.

Alias/ Unalias commands on Linux

knowledge hut

Alias commands:

- Alias commands are simply a way to represent another command. It is used to assign name to specified command list.
- User can define a string for some frequently used commands with options and arguments.
- The general syntax of alias command is:

```
alias [alias-name[=string]...]
```

Some applications automatically create aliases while installation and to view them use the following command:

```
$ alias
```

```
Or
```

```
$ alias -p
```

Alias command examples

`$ alias kh='/home/sk/KH/mydoc.txt':` This command creates an alias for **`/home/sk/KH/mydoc.txt`** path. Now to access this path, you must just type "**kh**".

`$ alias lscol='ls --color=auto':` This command creates an alias for **`'ls --color=auto'`** command. Now to access this path, you must just type **`"lscol".`**

Unalias

The unalias is a command that removes the aliases from your system Unalias command syntax is given below:

```
unalias <alias-name>
```

For example:

```
$ unalias kh
$ unalias lscol
```

Now the path linked to **`"kh" and "lscol"`** will be removed.

Important Linux commands

knowledge hut

"echo" Command

- **echo** command in Linux is used to display line of text/string that are passed as an argument .

- The general syntax of echo command is **:**

```
echo [option] [string]
```

- Syntax for displaying a text/string :

```
echo [string]
```

- For example:

```
$ echo "KnowledgeHut"
```

- Output: `KnowledgeHut`

Commonly used options for "echo" command

echo command can be used with multiple options, such as:

1. echo -e: Enables interpretation of backslash escapes ("\").

- **\b :** This command is used to remove all the spaces in between the text for example :

```
$ echo -e "Knowledge \bHut"
```

Output: **KnowledgeHut**

- **\n :** This command creates new line from where it is used. Example **:**

```
$ echo -e "Knowledge \nHut \nFor \nLinux"
```

Output:

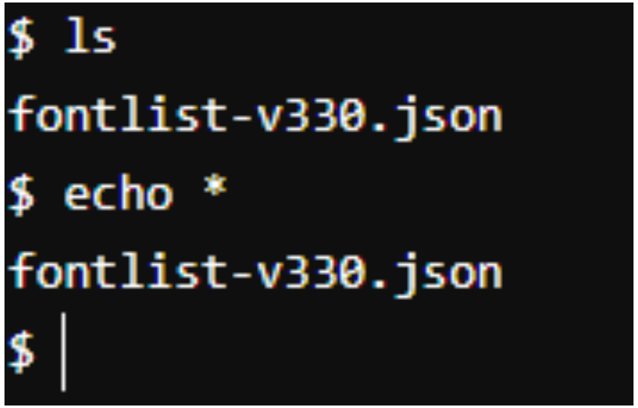
```
Knowledge
Hut
For
Linux
```

Commonly used options for "echo" command

echo *

This command will print all the files and directories as output. This is very similar to the **ls** command in Linux.

For example

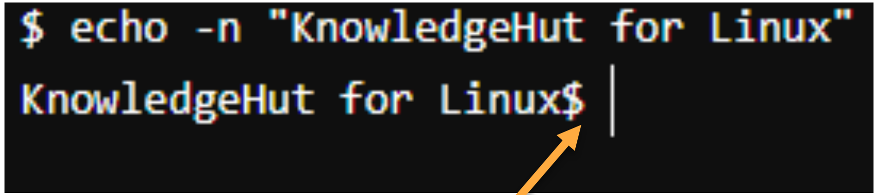


```
$ ls
fontlist-v330.json
$ echo *
fontlist-v330.json
$ |
```

Commonly used options for "echo" command

-n : this option is used to remove newline after one command . For example :

```
$ echo -n "KnowledgeHut for Linux"
```



```
$ echo -n "KnowledgeHut for Linux"
KnowledgeHut for Linux$
```

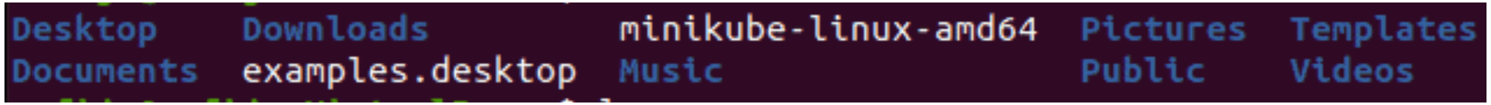
As seen in the above output, accepting next command will happen in the same line instead of new line.

"ls – a" Command

In Linux, hidden files starts with a dot. They are not visible in the regular folder or directory. To find all the files including hidden files in Linux, we use **`ls`** command with **`-a`** option.

`$ ls`

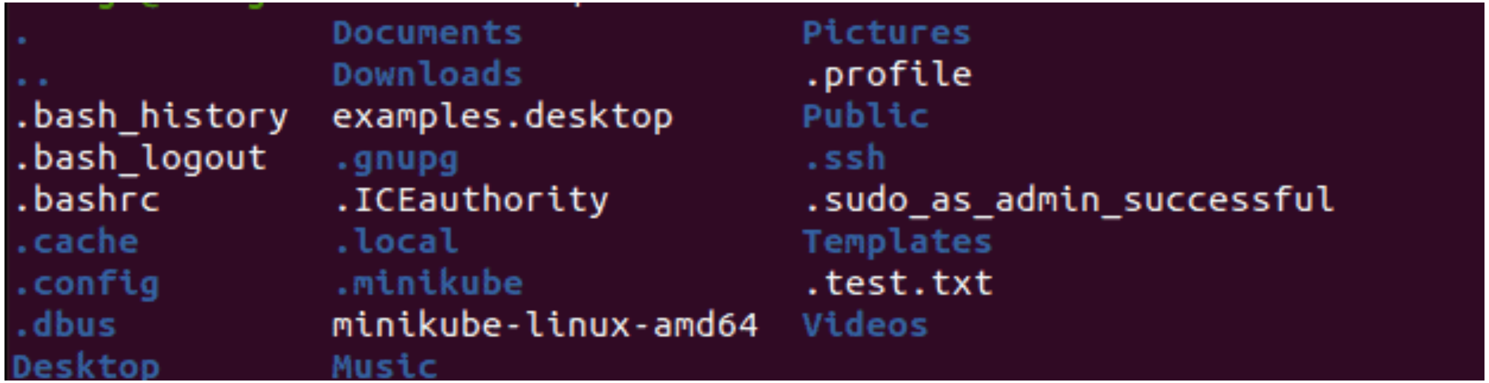
Output:



```
Desktop     Downloads         minikube-linux-amd64  Pictures  Templates
Documents   examples.desktop  Music                 Public    Videos
```

`$ ls -a`

Output:



```
.              Documents             Pictures
..             Downloads             .profile
.bash_history  examples.desktop      Public
.bash_logout   .gnupg                .ssh
.bashrc        .ICEauthority         .sudo_as_admin_successful
.cache         .local                Templates
.config        .minikube             .test.txt
.dbus          minikube-linux-amd64  Videos
Desktop        Music
```

"grep" command

- **Grep** is an acronym that stands for **G**lobal **R**egular **E**xpression **P**rint. This command is used when we want to search for a string of characters. The `grep` command is helps in searching through large log files.

For example, **to search a file**

```
$ grep knowledge samplefile
```

The above command will print **knowledge** in the file **samplefile. The "grep"** command will display lines where match for the word **knowledge**. You do not get exact matches. Instead, the system will print the lines with words containing the string of characters given with command.

- To Search Multiple Files, type names of the files as shown below

```
$ grep knowledge samplefile1 samplefile2 samplefile3
```

Links in Linux

- There are two types of links in Linux:

1. **Hard links**: Only links to files and not directories

2. **Symbolic links or Soft links**: Can link to directories

Process management in Linux

knowledge hut

Process in Linux

- Every time we run a shell command, a program is fetched and run this instance of the running program is called a **process**.

- Each process in Linux is identified with a unique number called a process id or **PID**. At any point in time, no two processes will have the same PID and anything and everything that is running is in execution will have a process ID.

- If we launch multiple shells, then each one of them will have its own process IDs. In computer terminology, processes are sometimes also known as tasks.

- A process in any operating systems can be a part of certain states, meaning a process can be in a state of running, sleeping, and stop etc.

1. Running : Actively using CPU
2. Sleeping : Wait state, process waits for a signal or another process to get over
3. Stop : Paused or sent to background

Process in Linux

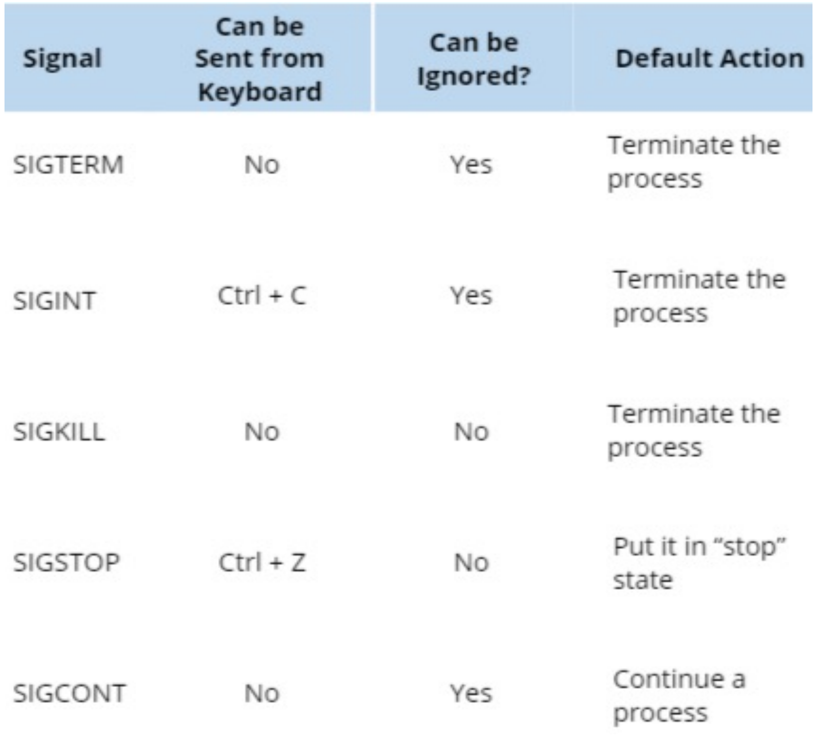
- The most common way to list processes currently running on our system is to use the command **`'ps'`**, which is short for process status. This command with its various options gives us a lot of information that comes in handy when troubleshooting our system.

'ps' command with options :

-a	hidden process in the shell
-A	all process in the system
aux	all process with a different set of columns
-ef	all process with a different set of columns
-ax	all process with only limited columns
-C	search for a process by name
-o	customising the output columns

Process in Linux

- Processes can be managed by sending certain signals to them by using **KILL** command.
- Signals are like software interrupts. A process responds to a signal in a predefined manner depending on the type of signal it receives.



Signal	Can be Sent from Keyboard	Can be Ignored?	Default Action
SIGTERM	No	Yes	Terminate the process
SIGINT	Ctrl + C	Yes	Terminate the process
SIGKILL	No	No	Terminate the process
SIGSTOP	Ctrl + Z	No	Put it in "stop" state
SIGCONT	No	Yes	Continue a process

Background Process

knowledge hut

Background Processes

- These are the processes that are running continually in the background of all other processes in the system all the time. For example, internal clock, some downloads in the background, etc.
- The command **"bg"** is used to resume a background process.
- These processes starts from a shell but runs independently from other tasks. It doesn’t interact with the users.

Terminating a process

- In Linux, if a process or background processes becomes unresponsive and blocks too many resources, it needs to be killed. Processes have their method of shutting down, some processes do not function properly and do not shut down as intended.

- To kill a process, we must first locate the process in the system. We can use **`ps`** with its options such as **`ps -aux`** for the detailed process list of all processes. The easiest way to view all the commands currently being run is to use the **`top`** command.

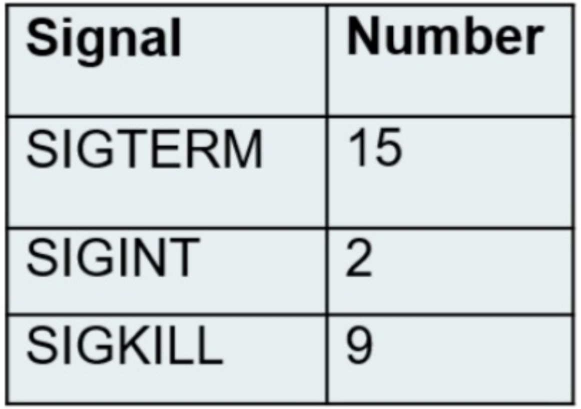
- After locating the process, we can give the following commands to kill the process:

1. **`killall <processname>:`** Kills processes by name.
2. **`pkill <process><option>:`** Kills process by name and some option.
3. **`kill <processid>:`** Kills single process by processID.
4. **`kill –9:`** sends SIGKILL signal.
5. **`xkill :`** closes connection (specific) of client to a server.

Terminating a process

- There can be multiple ways in which we can terminate a process:

1. **CTRL + C:** Simplest way of terminating the process from the terminal where it is running.
2. **KILL -9 4756 or SIGKILL 4756** is used to kill the process number 4756.
3. There are numbers associated with each signal, as shown below:



Signal	Number
SIGTERM	15
SIGINT	2
SIGKILL	9

Secure Shell and Shell Script

knowledge hut

Working with Secure Shell

SSH: to connect remote Linux servers. SSH creates and manages a secure and encrypted connection. Secure shell uses passwords, public/ private key pair, and host - based authentication.

SCP: SSH can also handle secure file copy. Syntax for copying files using SCP is as follows:

```
#scp source_file destination_file
```

For copying from current server to another server:

```
$ scp file1 root@101.127.27.23:/rootx
```

For copying from one remote server to another remote server:

```
$ scp root@101.127.27.25:/home/user/   root@103.127.35.26:/root/dir1/
```

Shell script

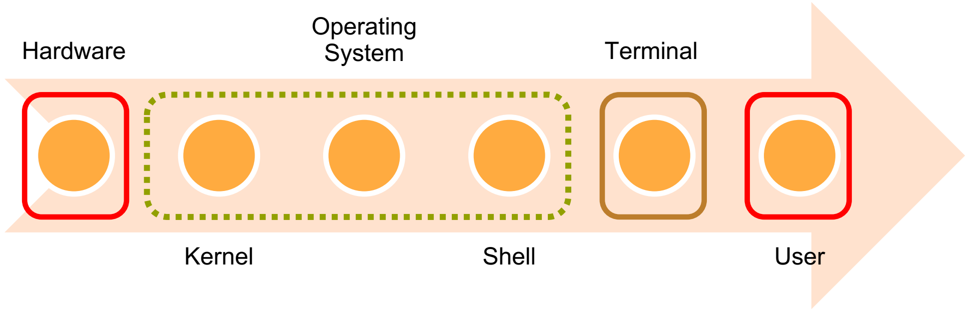
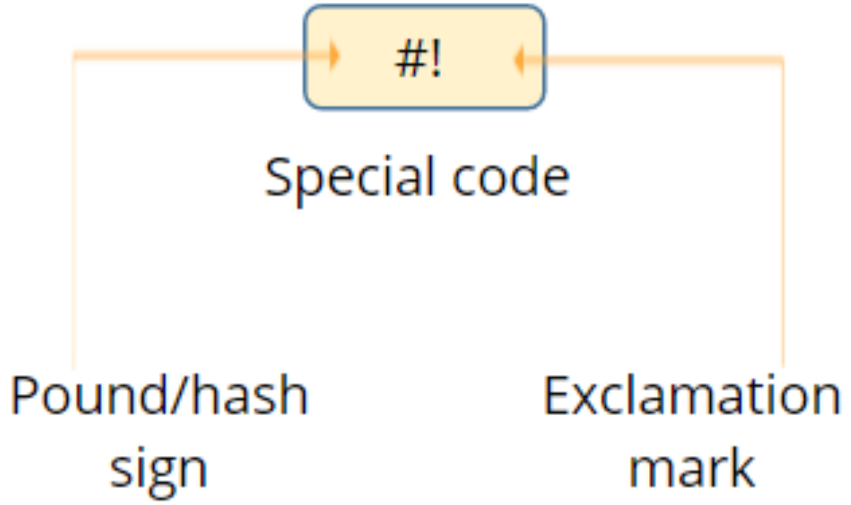


Fig: Components of Shell Program

A **computer script** is a list of commands executed by a certain program, or in our case, the shell. A Shell in the interface between the user and the operating system

In a **Linux environment** where we would typically be dealing with the bash shell, our scripts can be called **shell scripts** or **bash scripts**. The shell scripts are like plain text files having some special syntaxes, so we can use the **vi editor** or **nano editor** to create the scripts.

Writing a shell script



A shell script begins with a line that identifies the shell that is used to run it. Since we are using the bash shell our shell script must begin with the line of text '**#!**'.

Creating a shell script

Steps in creating a Shell Script:

1. **Create a file using** a **vi** editor with an extension **.sh**
2. The script should **start** with **#! /bin/sh**
3. Write the code.
4. Save the script as **filename.sh**
5. **Executing** the script by typing **bash filename.sh**

“#!” also known as shebang operator which directs the script to the location of the interpreter. If we use ”#! /bin/sh” the script is directed to the bourne-shell.

Variables

```
variable = "Hello"
echo $variable
```

Variables are the containers that holds a value or data. Every variable has a name depending on what data it stores.

Types of variables:

1. System defined variables: created by operating system (Linux) like HOME, SHELL, LOGNAME, etc.
2. User defined variables: created by users like age, name, password, username, etc.

Shell Script Example

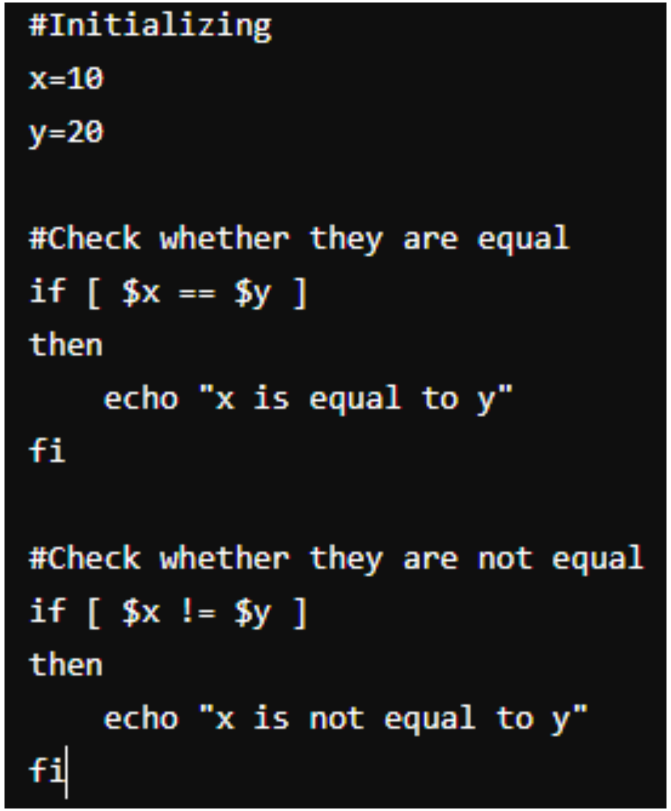
hello.sh

```
#!/bin/sh
echo "What is your name?"
read name
echo "How do you do, $name?"
read reply
echo "I'm $reply too!"
```

Output

```
(base) ➜  ~ sh hello.sh
What is your name?
James
How do you do, James?
Great
I'm Great too!
```

Conditional Statements



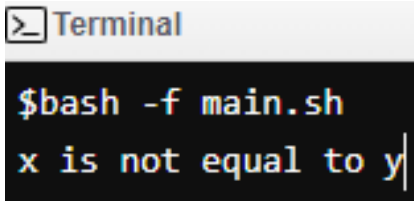
```
#Initializing
x=10
y=20

#Check whether they are equal
if [ $x == $y ]
then
    echo "x is equal to y"
fi

#Check whether they are not equal
if [ $x != $y ]
then
    echo "x is not equal to y"
fi
```

The basic structure of the '**if**' statement looks like this:

```
if [ conditional-expression ]
then
          commands
else
          other-commands
fi
```



Terminal

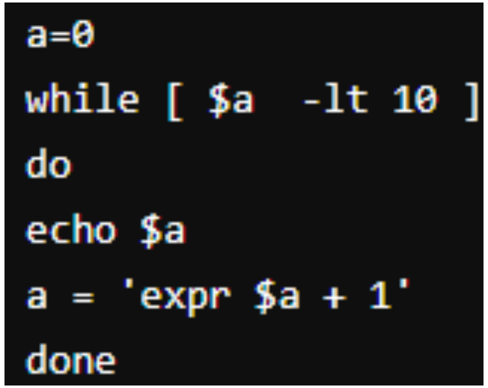
```
$bash -f main.sh
x is not equal to y
```

While Statement

The basic structure of the '**while**' statement looks like this:

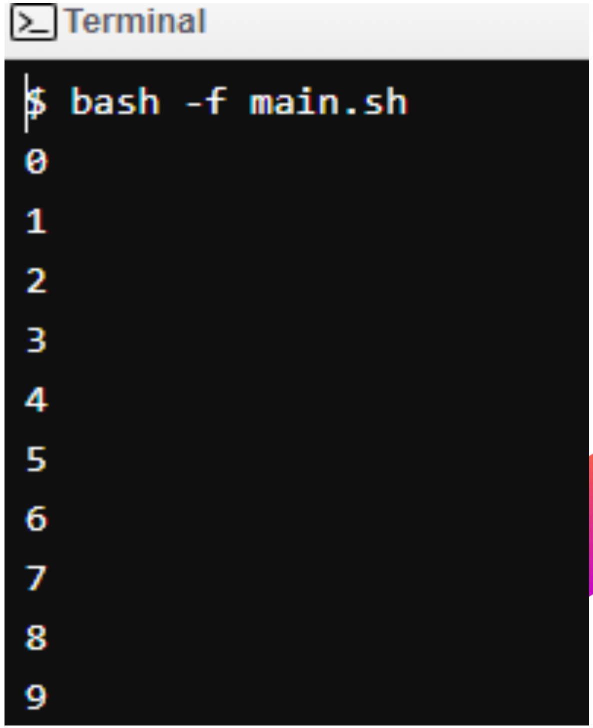
```
while [conditional-expression]
do
          commands
done
```

In the following code snippet, **`-lt 10`** refers to **less than 10.**



```
a=0
while [ $a  -lt 10 ]
do
echo $a
a = 'expr $a + 1'
done
```

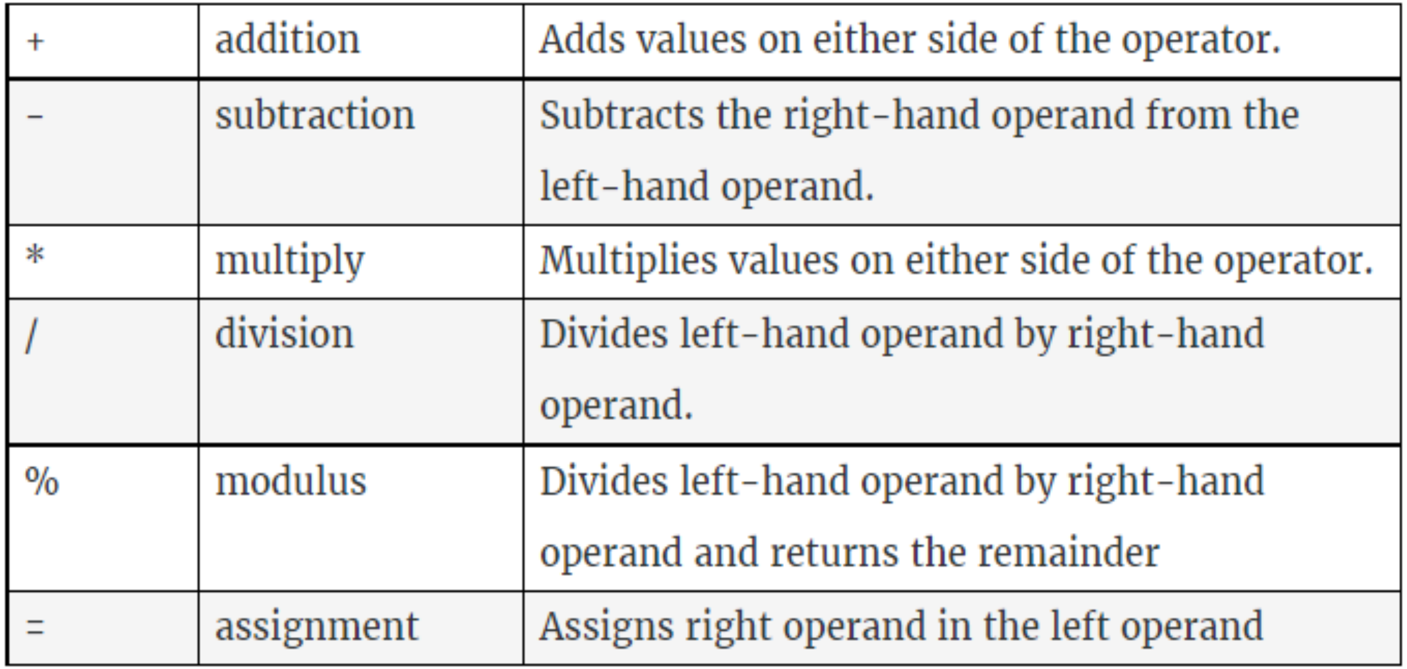
Output:



```
$ bash -f main.sh
0
1
2
3
4
5
6
7
8
9
```

Mathematical Operations

We can perform various mathematical operations in Linux and to do so, we would require to use the arithmetic operators shown below. To perform arithmetic operations, we use the **expr** command.

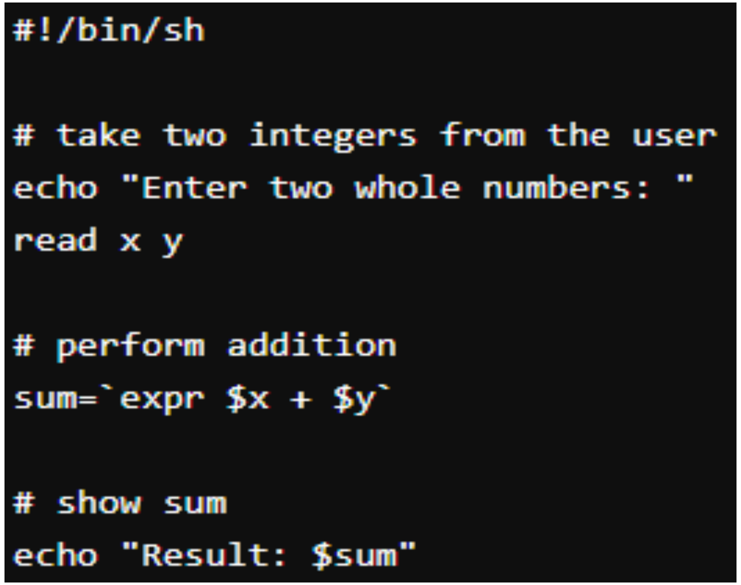


+	addition	Adds values on either side of the operator.
-	subtraction	Subtracts the right-hand operand from the left-hand operand.
*	multiply	Multiplies values on either side of the operator.
/	division	Divides left-hand operand by right-hand operand.
%	modulus	Divides left-hand operand by right-hand operand and returns the remainder
=	assignment	Assigns right operand in the left operand

Mathematical Operations

We can perform various mathematical operations in Linux and to do so, we would require to use the arithmetic operators shown below.

For example, to add any two integer, we can write the following shell script:



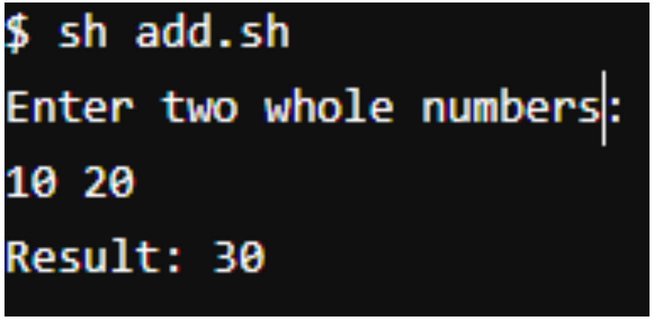
```
#!/bin/sh

# take two integers from the user
echo "Enter two whole numbers: "
read x y

# perform addition
sum=`expr $x + $y`

# show sum
echo "Result: $sum"
```

Output:



```
$ sh add.sh
Enter two whole numbers:
10 20
Result: 30
```

Thank you