**System Utility Commands**

**Date-**

**Cal-**

**uname:-**

The command ‘uname‘ displays the information about the system.

Options :-

-v = current kernel version

-n = hostname

-s = kernel name

-r = release version

-o = os name

-i = hardware platform

-m = machine hardware name

-p = processor type



**logname :-**

logname -print user's login name

**Hostname :-**

to display or set the system's hostname

$hostname

myhostname

**hostname -i** -----> show ip address of hostname ( only primay network interface )

**hostname -I** -----> list all IP addresses associated with the system

**hostname -s** -----> short hostname

ex:-for myhost.example.com short hostname will be **myhost**

**hostname -f** -----> to display the fully qualified domain name (FQDN)

ex :-myhostname.example.com

**hostname -d** -----> to display the domain name of system

ex:-example.com

**hostname -a** -----> to display all the alias of current hostname

**which :-**

Refrence-

<https://www.geeksforgeeks.org/which-command-in-linux-with-examples/>

The which command allows users to search the list of paths in the $PATH

environment variable and outputs the full path of the command specified

as an argument. The command works by locating the executable file matching

the given command .

Exit status-:

0 if all specified commands are found and executable

1 if one or more specified commands is nonexistent or not executable

2 if an invalid option is specified



cat command is actually is in /usr/bin/cat

/bin/cat is a link to previous folder

**bc:-**

The "bc" command in Linux is actually a calculator. It stands for "basic calculator" and is used for performing arithmetic calculations within the terminal.

1. Basic Usage:

To start the calculator, open your terminal and type `bc`. You can then enter mathematical expressions, and the calculator will provide the result.

```bash

$ bc

```

2. Simple Arithmetic:

You can perform basic arithmetic operations like addition, subtraction, multiplication, and division.

```bash

5 + 3

10 - 4

6 2

8 / 2

```

3. Variables:

You can use variables to store values for later use.

```bash

x = 5

y = 3

x + y

```

4. Precision and Scale:

The "scale" variable controls the number of decimal places in the result.

```bash

scale = 2

10 / 3

```

5. Functions:

"bc" supports various mathematical functions like `sqrt()`, `sin()`, `cos()`, `atan()`, etc.

```bash

sqrt(25)

s = sin(30)

```

6. Interactive Mode:

You can use "bc" interactively by entering expressions directly into the terminal, or you can provide a file with expressions.

```bash

bc -q filename

```

7. Exiting "bc":

To exit the "bc" calculator, type `quit` or press `Ctrl + D`.

```bash

quit

```

```bash

Ctrl + D

```

8. Command-Line Usage:

You can use "bc" directly from the command line to evaluate expressions.

```bash

echo "5 + 3" | bc

```

```bash

bc <<< "6 2"

```

9. Scripting with "bc":

You can create scripts with "bc" for more complex calculations. Create a file, e.g., `myscript.bc`, and put your expressions in it.

```bash

# myscript.bc

x = 5

y = 3

x + y

```

Run the script with:

```bash

bc < myscript.bc

```

1. \*\*Conditionals:\*\*

`bc` supports basic conditionals using `if`, `else`, and `endif`. Here's a simple example:

```bash

x = 10

y = 20

if (x > y) {

result = x

} else {

result = y

}

result

```

The `result` variable will contain the maximum of `x` and `y`.

2. \*\*Loops (using Recursion):\*\*

`bc` doesn't have explicit loop constructs, but you can simulate loops using recursion. Here's an example of a simple loop that calculates the factorial of a number:

```bash

define factorial(x) {

if (x <= 1) {

return 1

} else {

return x \* factorial(x - 1)

}

}

factorial(5)

```

This defines a recursive function `factorial` that calculates the factorial of a number. Note that recursion in `bc` has limitations, and extremely large recursive calls may lead to a stack overflow.

3. \*\*Arrays (using Functions):\*\*

While `bc` itself doesn't have built-in support for arrays, you can use functions to simulate array-like behavior. Here's an example of a function that calculates the sum of elements in an array:

```bash

define array\_sum(array, size) {

if (size <= 0) {

return 0

} else {

return array[size] + array\_sum(array, size - 1)

}

}

my\_array[1] = 3

my\_array[2] = 7

my\_array[3] = 2

array\_sum(my\_array, 3)

```

This demonstrates a basic function `array\_sum` that recursively sums the elements of an "array" (simulated using indexed variables).

**uptime:-**

The `uptime` command in Linux is used to display the system's uptime and load average. It provides information about how long the system has been running, the current time, the number of users logged in, and the system load averages for the past 1, 5, and 15 minutes.

Here's the basic syntax:

```bash

uptime

```

The output typically looks like this:

```bash

11:24:35 up 2 days, 1:30, 2 users, load average: 0.05, 0.10, 0.08

```

Breaking down the output:

- `11:24:35`: The current time.

- `up 2 days, 1:30`: The system has been up for 2 days and 1 hour and 30 minutes.

- `2 users`: The number of users currently logged in.

- `load average: 0.05, 0.10, 0.08`: The system load averages for the past 1, 5, and 15 minutes, respectively. The load average represents the average number of processes in the run queue over the specified time periods.

The load averages are often interpreted as follows:

- If the 1-minute load average is significantly higher than the 5 or 15-minute averages, it might indicate a recent spike in activity.

- If the 1, 5, and 15-minute load averages are all high, it could indicate a more sustained period of heavy system usage.

This information can be useful for system administrators to get a quick overview of the system's current state and performance.