# Linux

## Fundamentals of Pentesting

**Original Author:** *Karan Bhandari*

# Table of Contents

# Abstract

Linux is an open-source operating system known for its flexibility, security, and robustness. It has become the go-to choose for many cybersecurity professionals and enthusiasts due to its vast array of tools and its adaptability to various pentesting scenarios.

In this report, we'll explore the fundamentals of Linux, its relevance in the field of cybersecurity, and how it can be effectively used for conducting penetration tests. Whether you're new to Linux or an experienced user looking to enhance your penetration testing skills, this report aims to provide you with valuable insights and practical knowledge to navigate the world of Linux for pentesting effectively.

**Disclaimer: This report is provided for educational and informational purpose only (Penetration Testing). Penetration Testing refers to legal intrusion tests that aim to identify vulnerabilities and improve cybersecurity, rather than for malicious purposes.**

# Linux Basics

## Why use Linux for pentesting?

Linux offers a far higher level of control of the operating system, not to mention that it is **open source**. This also makes Linux **transparent** and easier to understand. Before we try to "hack" anything, it is a must to know how it works, this is why transparency in Linux is a huge plus.

Because Linux is very popular amongst the pen-testing community, most of the used **penetration testing tools and frameworks are also then built for Linux**.

**Maintenance** is also comparatively easy as the software can be easily installed up from its repository. It is also very stable when compared to traditional operating systems like Windows.

## Basic Linux Commands

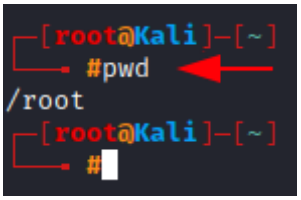
Just like how we use Windows on a daily basis, creating folders, moving files, copying things, we're going to learn these everyday operations for Linux.

We'll be spending most of our time in the terminal, which is the command-line interface of our operating system. This is where we type out commands to perform the operations we want.

### The "pwd" Command

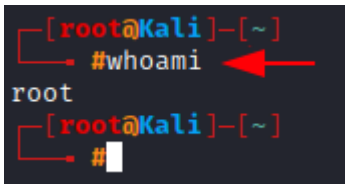
Before we begin, we should know which directory we are working in, and where are the files we create going to be stored. The pwd command is one way to identify the directory we're in.

So, as we did it in our case, we found that we're in the **/root** directory.



## The "whoami" Command

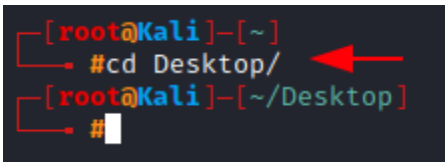
Using the whoami command we see which user we're logged in as. Here, we're logged in as **root** (which translates to an administrator in the windows terms)




## Cd: Changing directories

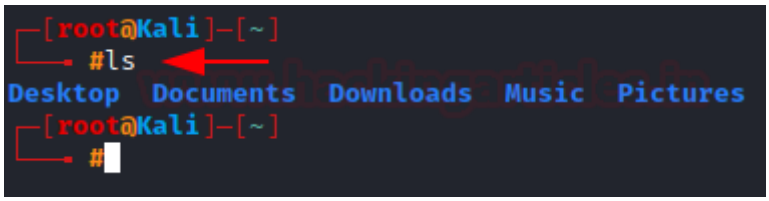
To change directories via the terminal, we use the **cd** command. Let's change our current directory to **Desktop.**

```
cd Desktop/
```




## Ls: Listing the Contents

To see the contents of a directory we use the "**ls"** command, (very similar to the dir command in windows)




## The "Help" Command

Nearly every command, application and or utility in Linux has a dedicated help file which guides its usage. If you want to learn more regarding a specific command or if you're stuck, help (-h, –help) will be your best friend.

Let's find out more about **volatility framework.**

```
volatility --help
```

```
┌──[root@Kali]─[~]
└─# #volatility --help
Volatility Foundation Volatility Framework 2.6
Usage: Volatility - A memory forensics analysis platform.

Options:
  -h, --help            list all available options and their default values.
                        Default values may be set in the configuration file
                        (/etc/volatilityrc)
```

## Man: The Manual Pages

In addition to the help file, most commands and applications also have a manual page, which can be accessed via typing **man** before the command.

As seen below, it provides a description and all the tags that can be used with the **ls command.**

```
man ls
```

```
LS(1)                                          User Commands

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List information about the FILEs (the current directory
       cally if none of -cftuvSUX nor --sort is specified.

       Mandatory arguments to long options are mandatory for s

       -a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..

       --author
              with -l, print the author of each file

       -b, --escape
              print C-style escapes for nongraphic characters

       --block-size=SIZE
              with -l, scale sizes by SIZE when printing them
              mat below
```

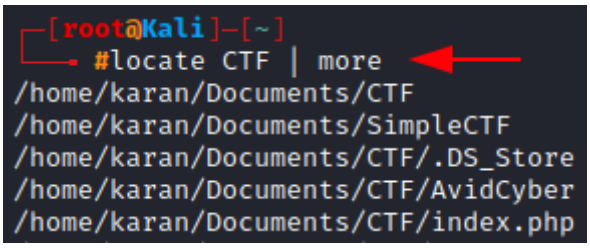
## Locate: Searching keywords

When searching for a specific keyword, one of the easiest ways to do so is using **locate**. Type locate and then the keyword on the terminal and it will search the entire file system for the occurrence of it.

Though a few drawbacks of using locate as it provides too much information and the database it uses is updated once a day, so you can't find files created minutes or hours ago.

Let's search for the keyword: **CTF with**

```
locate CTF | more
```




## Whereis: Finding binaries

*Let's begin this section with what are binaries?*

*Files that can be **executed**, similar to .exe's in Windows are referred to as binaries. These files generally reside in the **/usr/bin** or **/user/sbin** directories.*

Utilities like **ls**, **cd**, **cat**, **ps** (we'll cover some of these later in the article) are stored in these directories too.

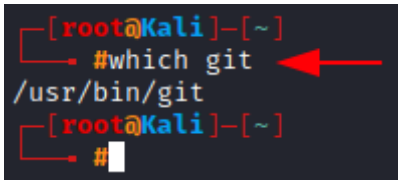
When looking for a binary file, we can use the whereis command. It returns the path of the binary as well it's man page. Finding the binary file: **git.**

```
whereis git
```

## Which: Finding binaries

The which command is more specific and only return the location of the binary in the PATH variable in Linux. Finding the binary file: **git.**

```
which git
```

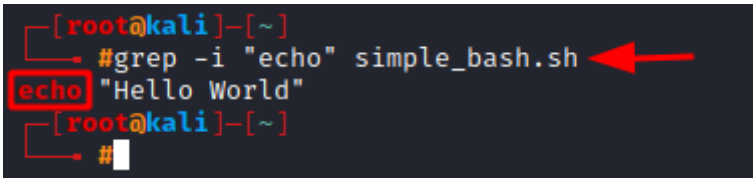



## Filtering with grep

Very often when using the command line, you'll want to search for a particular keyword, this is where grep comes in.

Let's search for the word: **echo,** in the **simple_bash.sh** file by typing

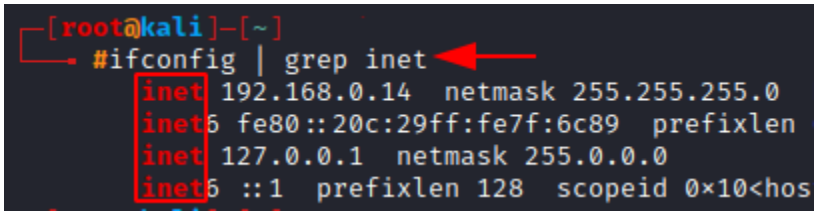
```
grep -I "echo" simple_bash.sh
```




Thought the most common use case of **grep** it to **pipe** the output into it with the keywords to filter the output.

Here we use grep just to get the IP address of our machine, instead of all the other information that comes when running the **ifconfig** command. (We'll touch on the ifconfig common in the later section)

```
ifconfig | grep inet
```

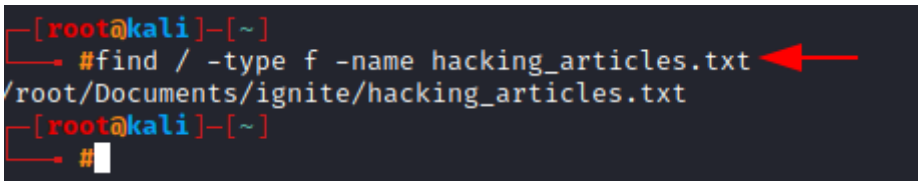



## Searching with the "find" command

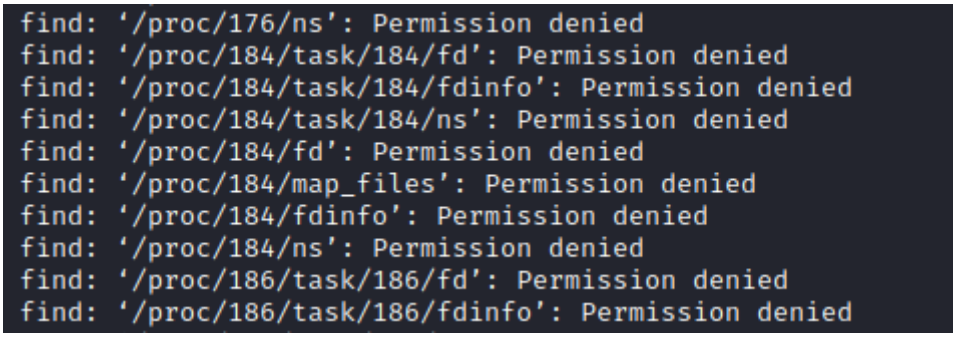
The find command is the most powerful and flexible of the searching utilities. It is capable of different parameters, including, the filename (obviously), date of creation and or modification, the owner, the group, permission and the size.

Here we use **-type** and **-name** tag which tells find the type of file we are looking for as well as its name. The backslash (/) indicates the root directory, which is where we want to search the file in.

```
find / -type f -name hacking_articles
```




If your result looks like this:




It is because the find command is also searching through directories your account doesn't have the permission to access to. Hence, for a cleaner result, we use **2>&1** which sends all the permission denied errors to /dev/null (into nothing) and then using **grep** filters them out of the output)

```
find / -type f -name hacking_articles 2>&1 | grep -v "Permission Denied"
```

```
┌──[root@Kali]─[~]
└─# #find / -type f -name hacking_articles 2>&1 | grep -v "Permission Denied"
/home/karan/Documents/ignite/hacking_articles
┌──[root@Kali]─[~]
└─# #
```

## The "cat" command

We use the cat command to output the contents of a file on the terminal. Let's use the cat command on "hacking-articles.txt".

```
cat hacking-articles.txt
```

```
┌──[root@Kali]─[~]
└─# #cat hacking-articles.txt
Contents of this file.
┌──[root@Kali]─[~]
└─# #
```

## Creating files with "touch"

The touch command allows you to create a new file. Simply specifying the filename after the touch command will result in the creation of that file.

Let's create a text file and name it "hacking-articles-2.txt"

```
touch hacking-artciles-2.txt
```

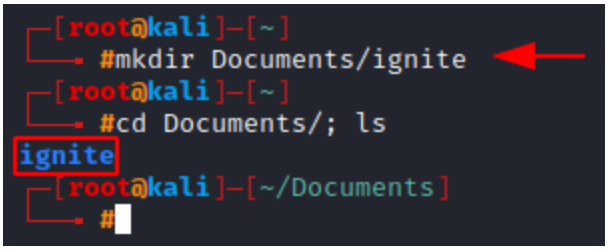
```
┌──[root@Kali]─[~]
└─# #touch hacking-articles-2.txt
┌──[root@Kali]─[~]
└─# #ls
Desktop    Downloads                hacking-articles.txt
Documents  hacking-articles-2.txt   Music
```

## Mkdir: Creating a directory

In order to make a directory or mkdir for short, we just need to specify the directory name after the mkdir command.

Let's create a directory: **ignite**

```
mkdir Documents/ignite
```

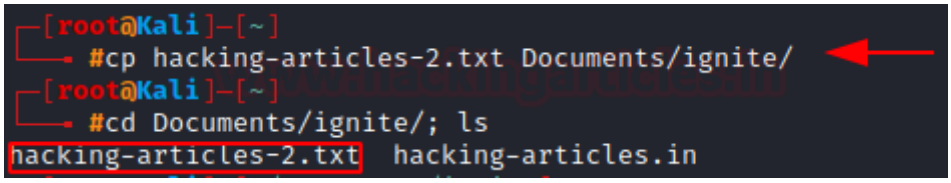



## Cp: Copying files

To copy files we use cp, which creates a duplicate of the file in the specified location. Let's copy the text file we created earlier into the directory we just created above. We then list the contents of the directory to ensure that the file has been copied.

To copy a file we type, **cp <the file we want to copy> <the destination of the "copied" file>**

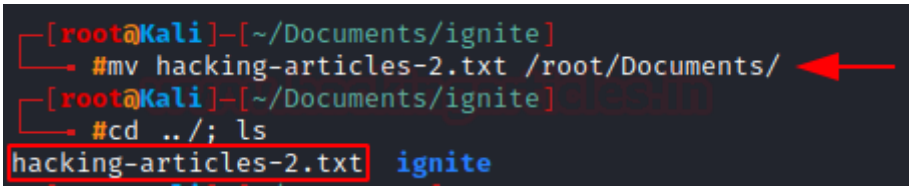
```
cp hacking-articles-2.txt Documents/ignite
```




## Mv: Moving/Renaming files

We can use the move command: **mv** not only to move files in the specified location but to also rename them. Now let's try to move the file we copied into the ignite folder, outside of it.

```
mv hacking-articles-2.txt /root/Documents/
```

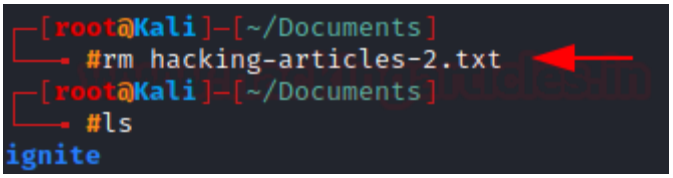



## Rm: Removing files

To remove a file, you can simply use the **rm command**. Let's remove the "**hacking-articles-2.txt**" file.

As you can see from ls, the file no longer exists.

```
rm hacking-artcles-2.txt
```

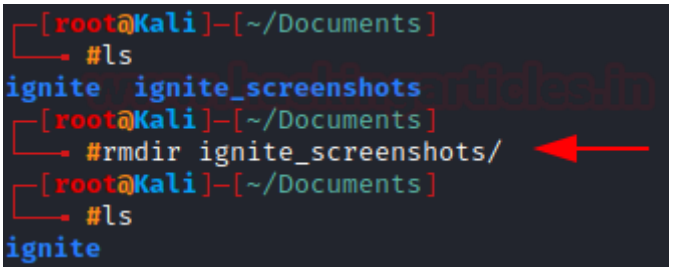



### Rmdir: Removing a directory

In order to remove a directory, we use the rmdir command which stands for "remove directory". Let's remove the "**ignite_screenshots**" directory.

(Use **rm -r** for directories with content inside them, r stands for recursive)

```
rmdir ignite_screenshots/
```




## Text manipulation

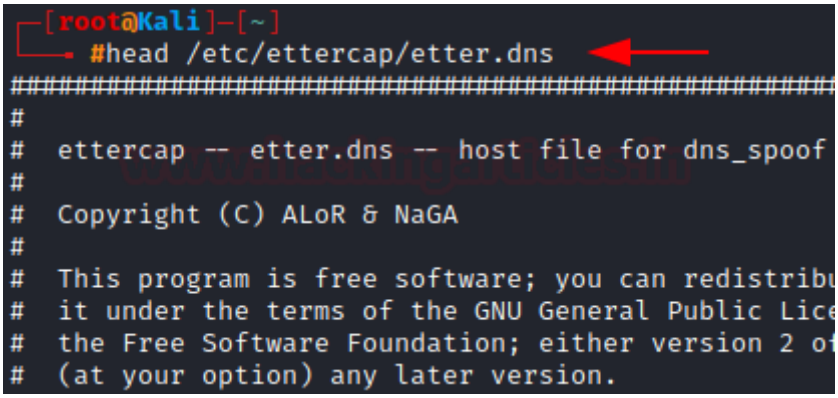
In Linux, almost everything you are going to deal with is going to be a **file**, more often a text file; for instances, configuration files. Hence, learning how to manipulate text becomes crucial while managing Linux and its applications.

### Grabbing the head of a file

When dealing with large files, we can use the head command, which by default displays the first 10 lines of a file. Let's view the first 10 lines of the **etter.dns** file.

(etter.dns is a file configuration of file of a tool called **Ettercap** which is used to in DNS spoofing and ARP attacks)

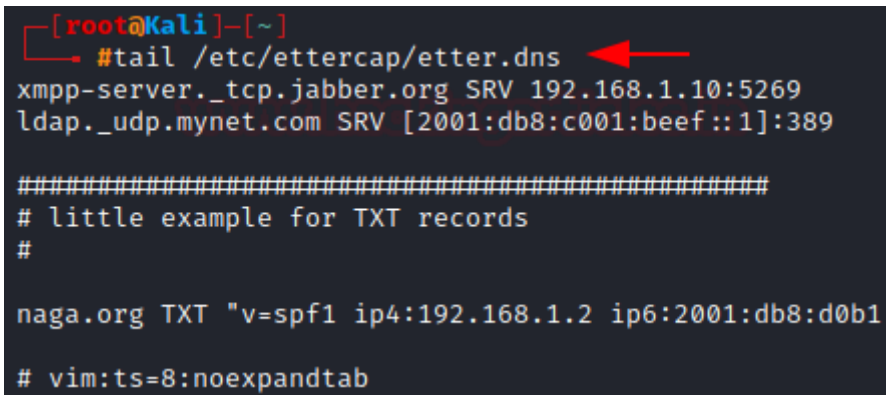
```
head /etc/Ettercap/etter.dns
```




## Grabbing the tail of a file

Similar to the head command, the tail command is used to view the last lines of file. Let's view the bottom lines of the **etter.dns** file.

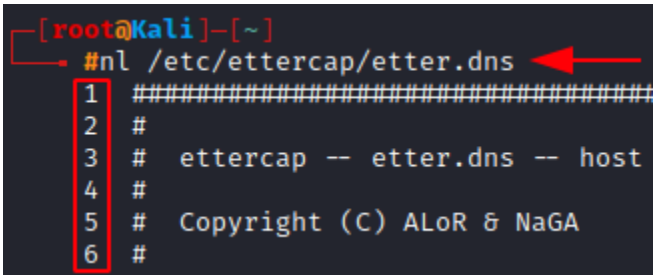
```
tail /etc/ettercap/etter.dns
```




## Nl: Numbering the lines

We can use the nl command to number the lines while it outputs them on the terminal window. Again, using the etter.dns let's number all of the lines this time.

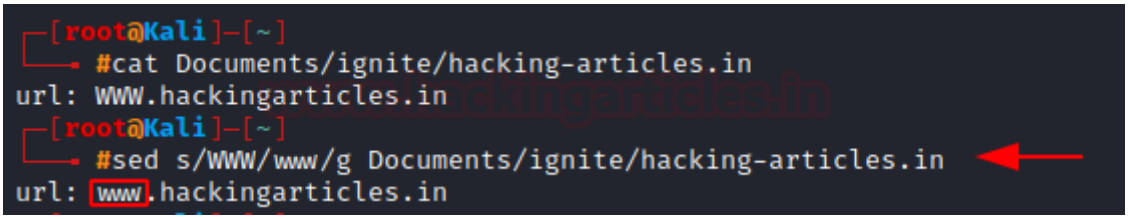
```
nl /etc/Ettercap/etter.dns
```




## Sed: To find & Replace the Text

The sed command lets you search for the occurrence of a word or a text pattern and then perform some action on it. Here we are going to use the **/s** tag to **search** for the occurrence of WWW and **/g** for **global replacement** with www.

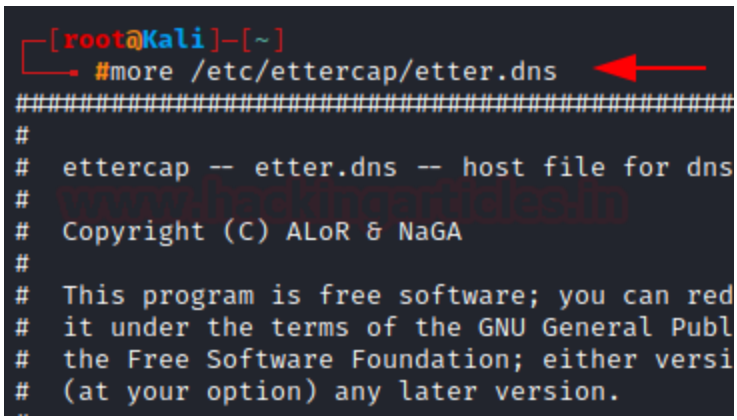
```
sed s/WWW/www/g hacking-artciles.in
```

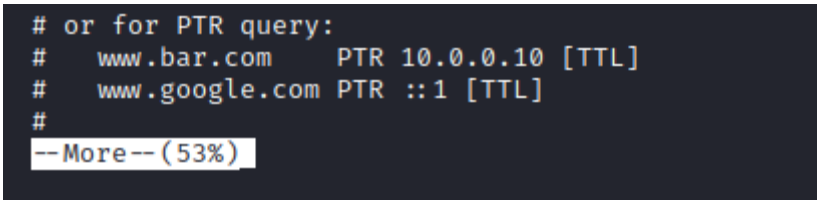



## More: Controlling the display of a file

The more command displays a page of a file at a time and lets you scroll down using the ENTER key. Opening the etter.dns file using more.

```
more /etc/ettercap/etter.dns
```






### Less: Displaying and filtering a file

The less command is very similar to more, but it comes with the added functionality of being able to filter keywords. Let's open the etter.dns file using less. We can further press the backward slash (/) on the keyboard and then enter the keyword we want to search for, here I've searched my own IP Address.

```
less /etc/ettercap/etter.dns
```


## Installing and Removing Softwares

We often need to install software that didn't come with your distribution of Linux or later down the lane, even remove the unwanted software.

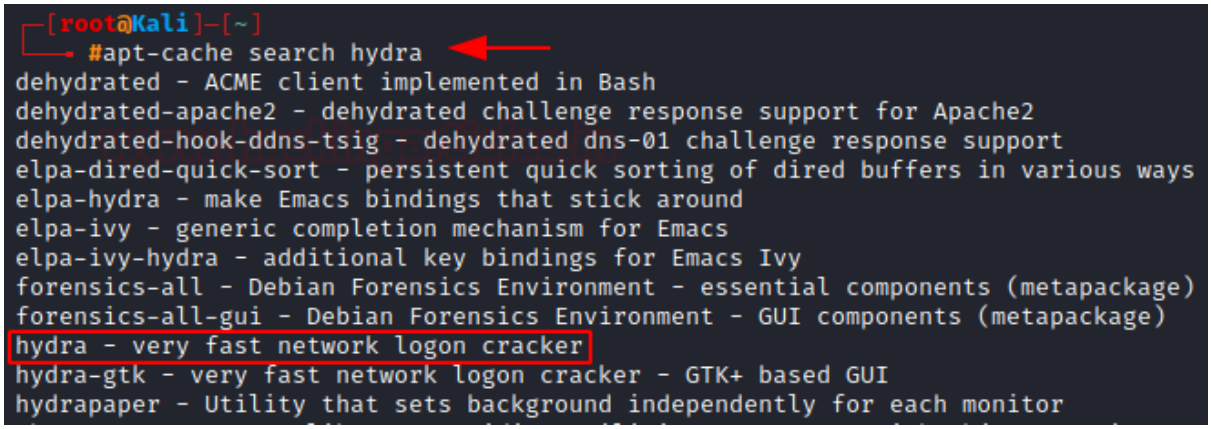
In Debian based Linux distributions, like Kali Linux (the one I am using), the default software manager is the Advance Packaging Tool or apt for short. Just how we would go to the Appstore to download an app, we have repositories in Linux. We'll learn how to access this repository, search in it and download from it.

### Searching for a package to install

Before we download any software package, let's check whether it is available in the repository, which is where our Linux operating stores information. We'll be using the apt tool.

Type **apt-cache search** and then the package that you want to search for, let's search for Hydra which is login cracking tool. Highlighted is the tool we are talking about.

```
apt-cache search hydra
```

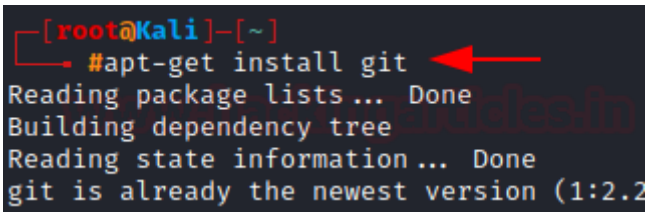



## Installing packages

Now let's install the packages we want. This time we'll be using the **apt-get** command followed by install and the package name.

Let's install git, which will later allow us to pull repositories from Github to install furthermore tools.

```
apt-get install git
```

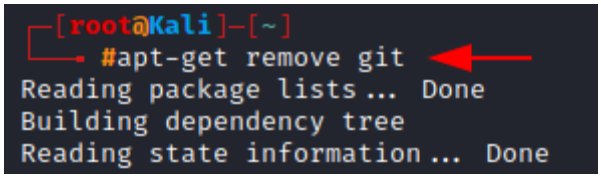



## Removing packages

To remove any package from your machine, simply type **remove** after **apt-get** with the package name.

Let's remove the git package. (I recommend to **Press n** to abort this step)

```
apt-get remove git
```

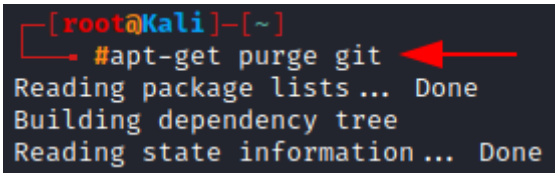



### Purging packages

Sometimes the package we just removed leaves residual files behind (an example would configuration files) In order to completely wipe out everything clean, we use the **purge** option with **apt-get**.

Let's try to purge git (again you can **press n** to abort)

```
apt-get purge git
```




## Updating the repository

It is good practice to update the repository as they are usually updated with new software or newer versions of existing software. These updates have to be requested and can be done by typing **update** after **apt-get.**

Let's update our repository. (Note: update doesn't apply these changes only downloads them)

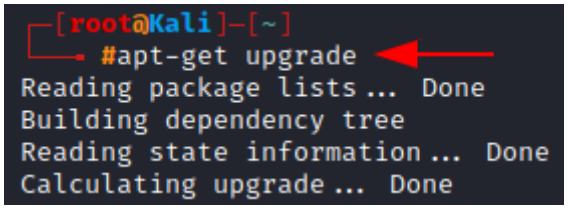
```
apt-get update
```

## Upgrading the repository

In order to apply the changes from the command we run above: update, we have to run the **apt-get** with the **upgrade** tag. This then installs or rather upgrades all the new updates that were downloaded to the system.

(Note: Upgrading can be time-consuming, so you might not be able to use your system for a while)

```
apt-get upgrade
```



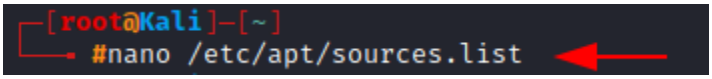

## Adding repositories to the sources.list file

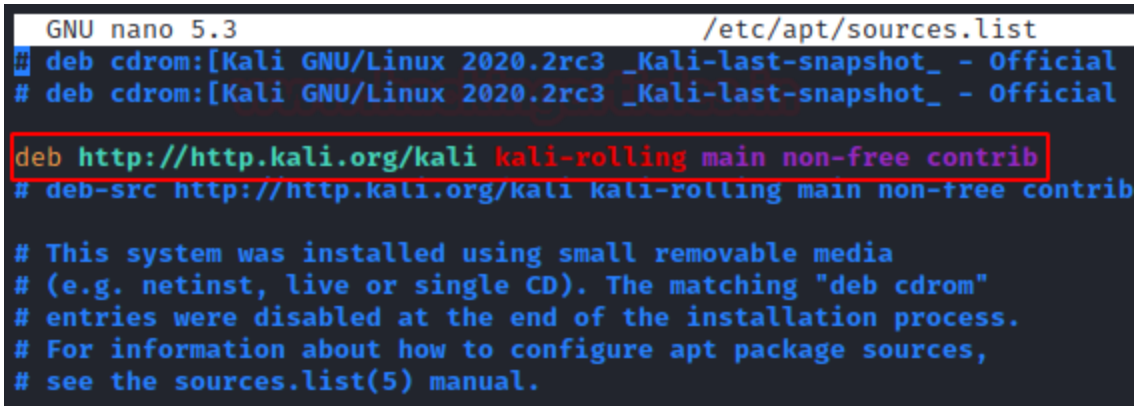
The server that holds the information of the software for particular distributions of Linux are known as repositories. We can **nano** into the file at **/etc/apt/sources.list** and add repositories here.

(I recommend **not** to add any experimental repositories in your sources.list because they can download problematic software and cause things to break. )

Highlighted is the repository my Kali Linux is using.

```
nano /etc/apt/sources.list
```






# Playing with Permissions

Before we start learning the Linux commands to play with permissions, let's learn about file/directory permission in Linux first.

As you know by now, in Linux the root user is all-powerful, the root user can do **anything** on the system. The other users have limited capabilities, and are usually collected into **groups** that generally share a similar function.

For example, a different group for the developer team, deployment team and administrators to initiate different levels of access and permission.

All the files and directories in Linux are allocated with three of levels of permission:

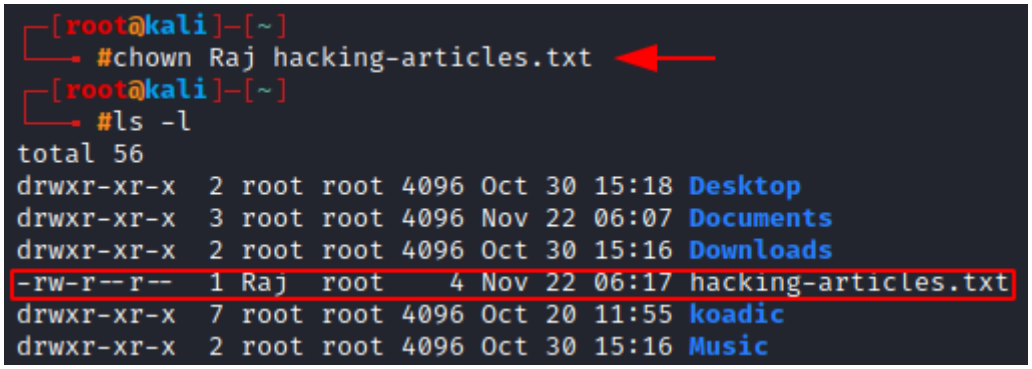
- **r permission:** This allows the user access to open and view a file
- **w permission:** This allows the user to view and edit the file
- **x permission:** This allows the user to execute the file (not necessarily view or edit it though)

## Granting ownership to an individual user

We change the ownership of the file so that the new user who owns can have the ability to control its permissions. Here we'll use the **chown** command to change the owner.

Let's change the owner of **hacking-artciles.txt** from **root** to **Raj**

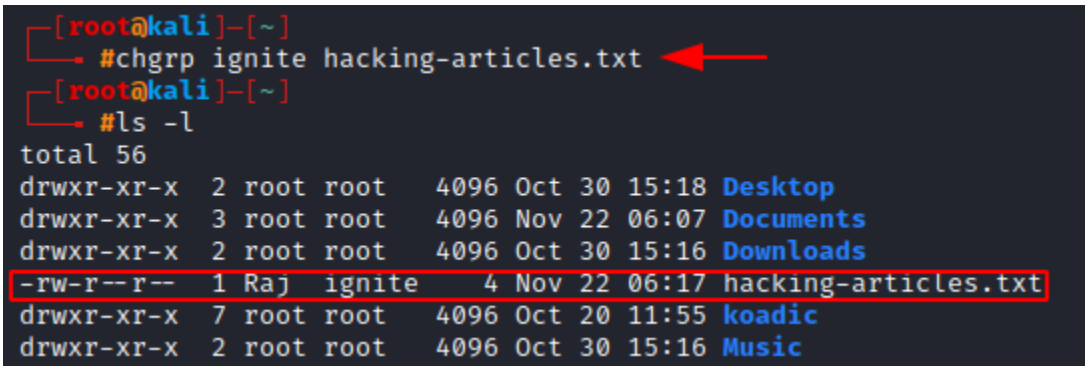
```
chown Raj hacking-articles.txt
```




## Granting ownership to a group

To transfer ownership of a file to a group we use the **chgrp** command. To ensure only the ignite team member can have the ownership, let's change the group to **ignite.**

```
chgrp ignite hacking-articles.txt
```




## Checking ownership

As you can see in the screenshots above, we are using the **ls** command with the **l** tag to view the permissions granted to the files and directories.

This out represent,

- The type of file (- representing a file, while d representing a directory)
- The permissions of the file for the owner, group and users, respectively
- The number of links
- The owner of the file, user and then group
- The size of the file in bytes
- When the file was last created or last modified
- The name of the file

Highlighted are the ownership section of the file.

```
┌──[root@kali]─[~]
└─# ls -l
total 52
drwxr-xr-x  2 root root  4096 Oct 30 15:18 Desktop
drwxr-xr-x  3 root root  4096 Nov 22 06:07 Documents
drwxr-xr-x  2 root root  4096 Oct 30 15:16 Downloads
-rw-r--r--  1 Raj  ignite    0 Nov 22 06:09 hacking-articles.txt
drwxr-xr-x  7 root root  4096 Oct 20 11:55 koadic
drwxr-xr-x  2 root root  4096 Oct 30 15:16 Music
drwxr-xr-x  2 root root  4096 Oct 30 15:16 Pictures
drwxr-xr-x  2 root root  4096 Oct 30 15:16 Public
-rwxr-xr-x  1 root root    19 Nov 20 08:41 simple_bash.sh
```

## Changing permissions

We use the **chmod** command to change the permissions of a file. This table will help you in deciding the permissions you want to give the file:

| | |
|---|---|
| 0 | — |
| 1 | –x |
| 2 | -w- |
| 3 | -wx |
| 4 | r– |
| 5 | r-x |
| 6 | rw- |
| 7 | rwx |

We could run, **chmod 777 $filename** to give the file ALL the permissions,

or simply **chmod 111 $filename** to give it executable permission.

Another way of doing so, is ch**mod +x $filename**, as seen below.

We can see the colour of the file change, indicating that it is executable.

```
chmod +x hacking-articles.txt
```

```
┌──[root@Kali]─[~]
└─# chmod +x hacking-articles.txt
┌──[root@Kali]─[~]
└─# ls
Desktop  Documents  Downloads  hacking-articles.txt  Music  Pictures
```

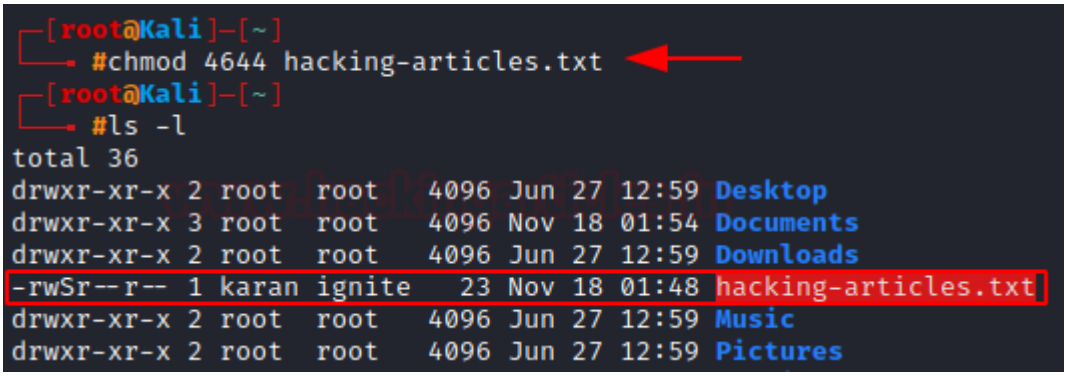
## Granting permissions with SUID

SUID bit says that any user can execute the file with the permissions of the owner but those permissions don't extend beyond the use of that particular file.

To set the SUID bit, we need to enter **4 before the regular permissions**, so the new resulting permission of 644 will become: **4644.**

Let's set the SUID bit for **"hacking-articles.txt".**

```
chmod 4644 hacking-articles.txt
```



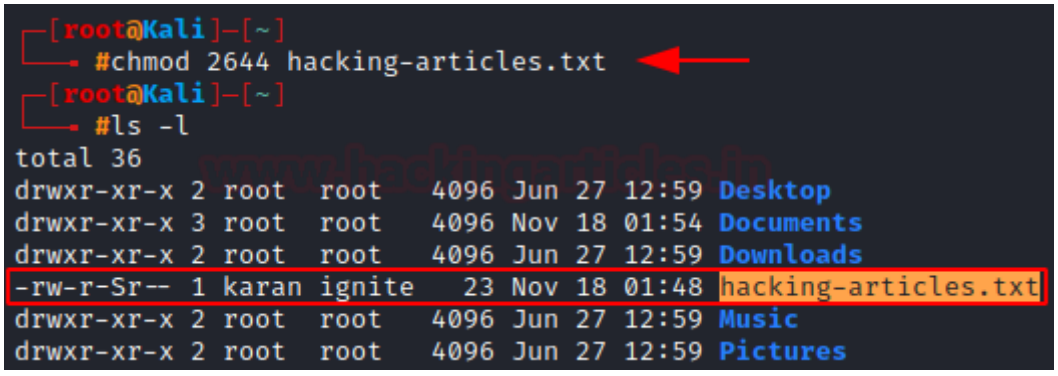

## Granting the root User's Group Permission SGID

Similar, SGID also grants temporary elevated permission but for the file owner's group.

To set SGID permission, we need to enter **2 before the regular permission**.

Let's set the SGID bit for "**hacking-artivcles.txt**".

```
chmod 2466 hacking-articles.txt
```



# Networks & Process Management

## Managing Networks

Networking is a crucial topic for any aspiring penetration tester. A lot of times you would be required to test a network or something over it. Hence, it becomes important to know you to connect and interact with all of your network devices.

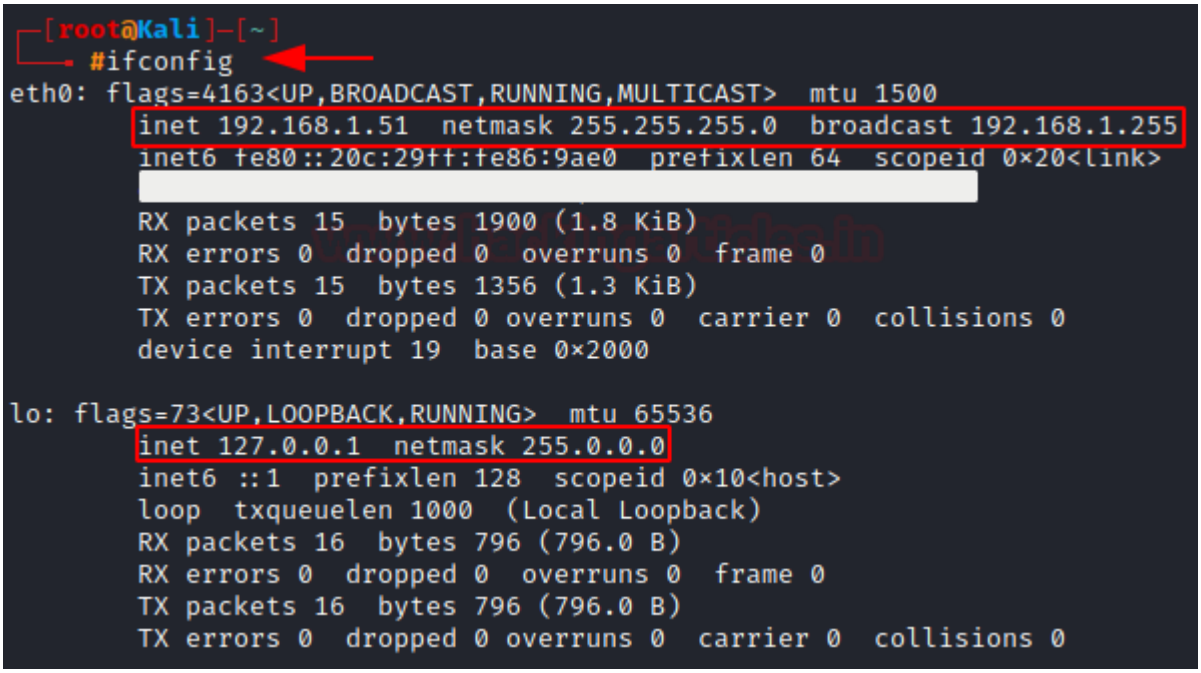
Let's get started with learning all the various tools and utilities to analyze and manage networks.

### Ifconfig: Analyzing networks

The ifconfig command is one of the most basic tools for interacting with active network interfaces. Here we run ifconfig and we can see the IP address mapped to our 2 network interfaces: **eth0** and **lo**.

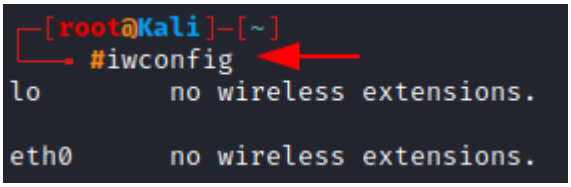
We can also see the **netmask** and a **broadcast address** of the network interface attached. As well as the **mac address** which I have blurred out.

(lo is localhost and is always mapped to 127.0.0.1)



## Iwconfig: Checking wireless network devices

If you have a wireless adapter, you can use the iwconfig command to gather crucial information such as its IP address, MAC address, which mode it is in and much more. Since I don't have a wireless adapter, my output is as such.

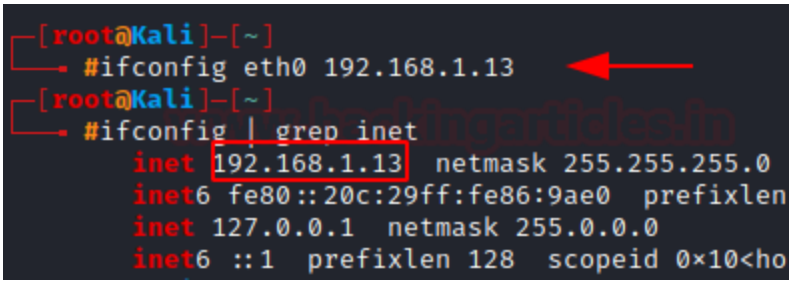



## Changing your IP Address

In order to change your IP address, enter ifconfig, the interface you want to change the address for and the new address you want to assign to it. Let's change the IP address to **192.168.1.13.**

Upon running ifconfig we see the change reflected.

```
ifconfig eth0 192.168.1.13
```

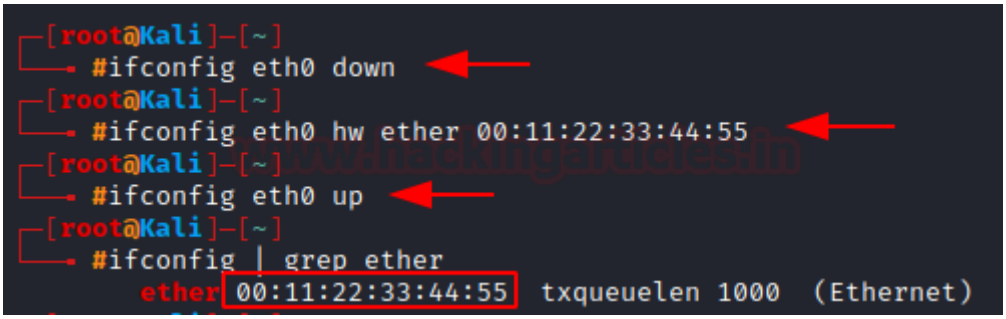



## Spoofing your MAC Address

You can also use ifconfig to change your MAC address. Since MAC address is globally unique and it often used as a security measure to keep the hackers out of networks or even to trace them, spoofing your MAC address is almost trivial in order to neutralize these security measures and maintain anonymity.

In order to change our MAC address to 00:11:22:33:44:55, we'll have to down the interface, change the MAC address and then up the interface again.

```
ifconfig eth0 down
ifconfig eth0 hw ether 00:11:22:33:44:55
ifconfig eth0 up
```

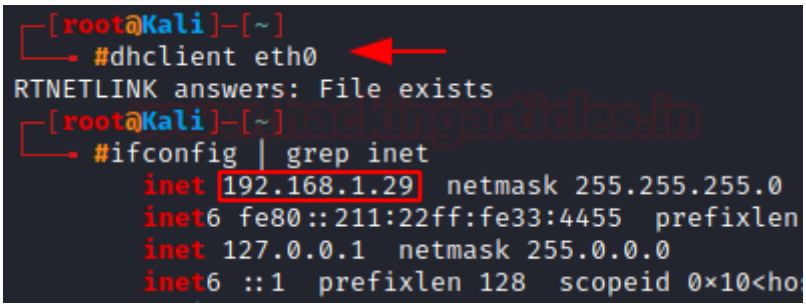



## Using DHCP Server to assign new IP Addresses

Linux has a Dynamic Host Configuration Protocol (DHCP) server that runs a daemon – a process that runs in the background called DHCP daemon. This DHCP server assigns IP addresses to all the systems on the subnet and it also keeps log files of such.

Let's request an IP Address from DHCP, by simply calling the DHCP server with the command **dhclient** and **network interface** you would want to change the IP Address of. We can see the IP Address has changed from what we had manually given it earlier.

```
dhclient eth0
```

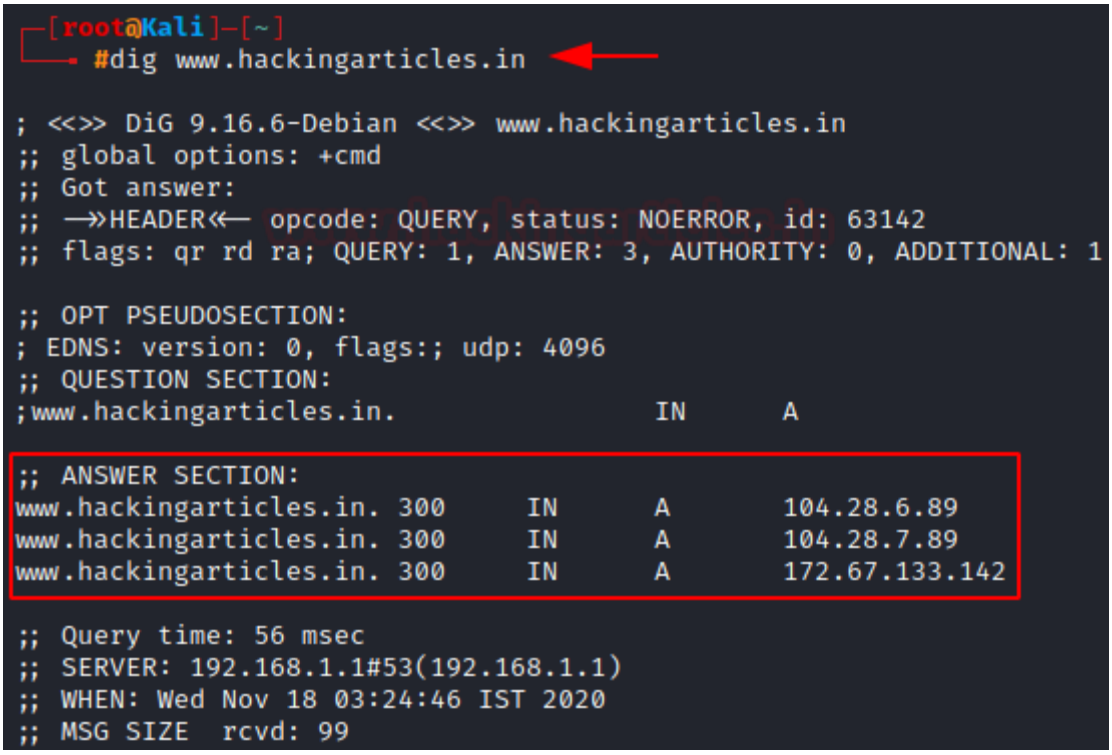



## Examining DNS with dig

DNS is a service that translates a domain name like **"hackingarticles.in"** to the appropriate IP address. We can use the dig command with added options such as mx (mail server), ns (name sever) to gather more information regarding the domain and its mail and name servers respectively.

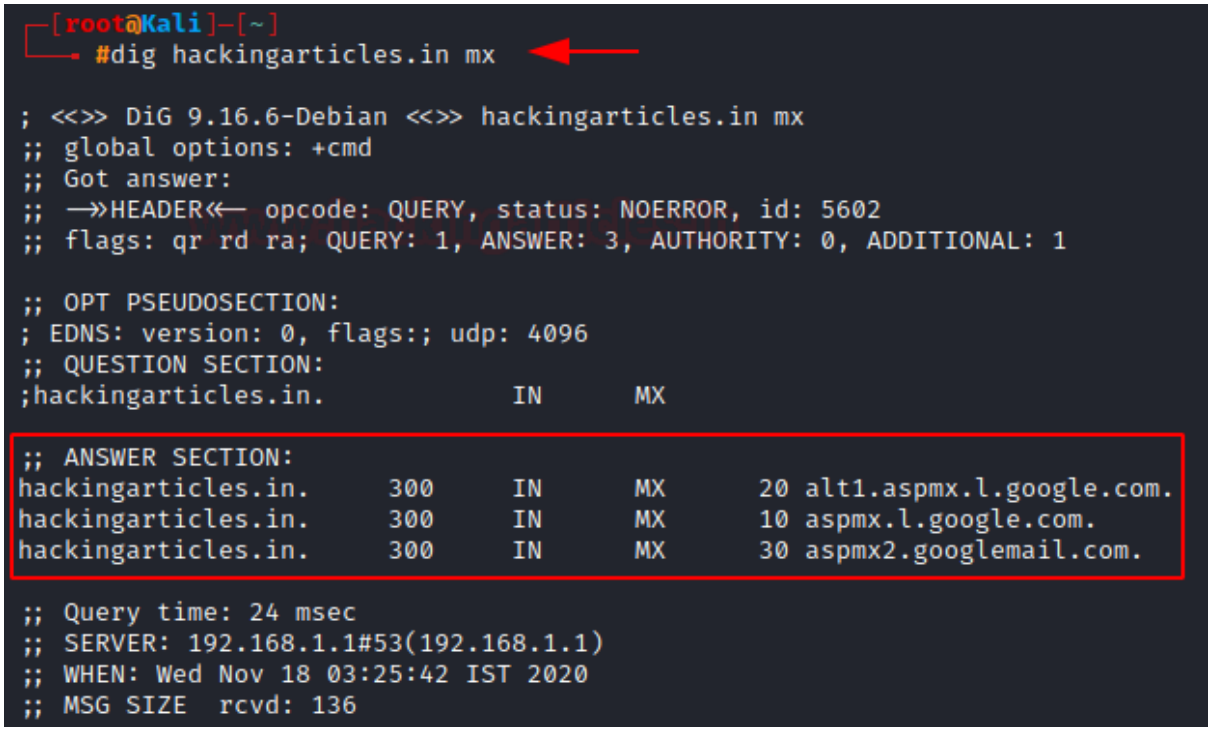
Let's use the dig command on **"www.hackingarticles.in"** here we can see the domain name resolve into IP Address.

```
dig www.hackingarticles.in
```




Further searching "**hackingatricles.in"** mail servers:

```
dig hackingarticles.in mx
```




Searching for the name servers:

```
dig hackingarticles.in ns
```

```
┌──[root@Kali]─[~]
└─── #dig hackingarticles.in ns

; <<>> DiG 9.16.6-Debian <<>> hackingarticles.in ns
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 59582
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;hackingarticles.in.            IN      NS

;; ANSWER SECTION:
hackingarticles.in.     86400   IN      NS      duke.ns.cloudflare.com.
hackingarticles.in.     86400   IN      NS      kay.ns.cloudflare.com.

;; Query time: 60 msec
;; SERVER: 192.168.1.1#53(192.168.1.1)
;; WHEN: Wed Nov 18 03:26:47 IST 2020
;; MSG SIZE  rcvd: 101
```

**Changing your DNS Server**

The DNS server information is stored in **/etc/resolv.conf**, in order to change the DNS server we need to edit this file. We can simply use **nano** or **vim** which are some of the common text editors Linux.

Here, we are going to use the **echo** command and > to overwrite the **resolve.conf** file. We can see the change reflect when reading using **cat.**

- is Cloudflare's public DNS server, you could also use Google's which is 8.8.8.8)

```
echo "nameserver 1.1.1.1" > /etc/resolv.conf
```

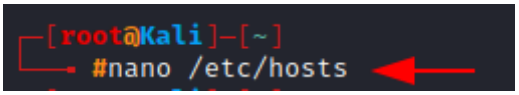
```
┌──[root@Kali]─[~]
└─── #echo "nameserver 1.1.1.1" > /etc/resolv.conf
┌──[root@Kali]─[~]
└─── #cat /etc/resolv.conf
nameserver 1.1.1.1
```

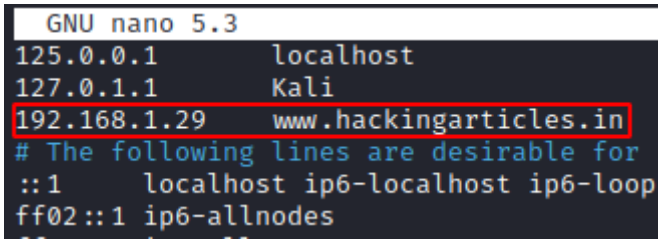
## Mapping the IP Addresses

There is a file in our system called **hosts** which also performs domain name – IP Address translation. The file is located in /**etc/hosts.** We can map any domain to the IP address of our choice, this can be useful as the hacker to direct traffic from network to a malicious web server (using dnspoof).

Let's nano into the file. Here we can see localhost and kali mapped to certain IP addresses. We can map www.hackingarticles .in to our IP address. Now if anyone on the network goes to this URL it will be re-directed to our IP address, we can further run an apache server and deploy a malicious website, tricking the users in the network.

```
nano /etc/hosts
```




# Process Management

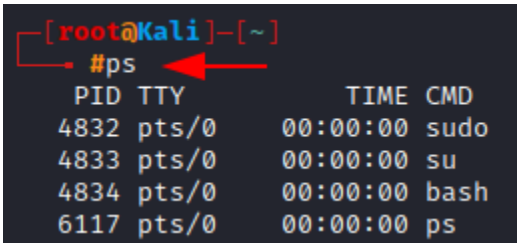
A process is just a program that's running on your system and consuming resources. There are times when a particular process has to be killed because it's malfunctioning or as a pen-tester, you would want to stop the anti-virus applications or firewalls. We'll learn how to discover and manage such processes in this section.

## Viewing process

In order to manage the process, we must be able to view them first. The primary tool to do so is **ps.**

Simple typing **ps** in the bash shell will list down all the **active processes**.

(PID stands for process ID and is unique for every invoked process.)

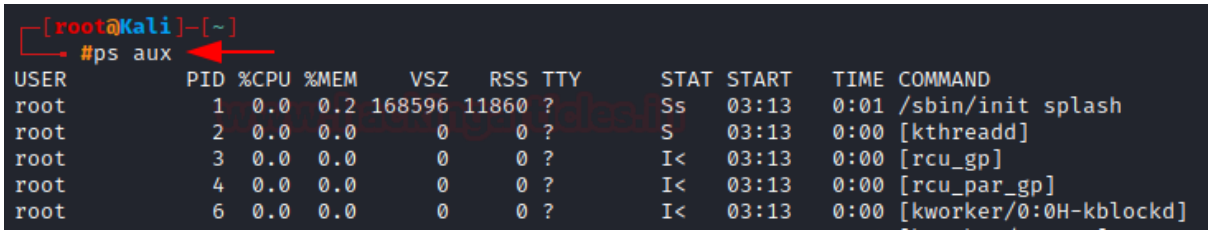



## Viewing process for all the users

Running **ps** command with **aux**, will display **all the running processes for all users**, so let's run:

```
ps aux
```

Here we can see PID, the user who invoked the process, %CPU the process is using, %MEM represent the percentage of memory being used and finally COMMAND which is the name of the command that has started the process




## Filtering Process with its name

As we learned earlier, we can pipe the output of **ps aux** into **grep** and filter out the specific information we want.

Let's search for **msfconsole** (A popular interface to use the Metasploit framework)

```
ps aux | grep msfconsole
```

## Top: Finding the greediest process

In some use cases when you want to know which process is using the most resources, we use the **top** command. It displays the process ordered by the resources used. Unlike ps, the top also refreshed dynamically – every 10 seconds.

```
┌──[root@Kali]─[~]
└─# top
```

```
top - 03:53:54 up 40 min,  1 user,  load average: 0.14, 0.18, 0.18
Tasks: 261 total,   2 running, 259 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.3 us,  0.3 sy,  0.0 ni, 99.4 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   3938.0 total,   2014.1 free,   1077.5 used,    846.4 buff/cache
MiB Swap:   6675.0 total,   6675.0 free,      0.0 used.   2579.0 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   4098 root      20   0  428268 138496  68452 R   2.3   3.4   0:57.85 Xorg
   4567 karan     20   0  514704  39116  31664 S   0.7   1.0   0:01.93 panel-17-pulsea
   1364 root      20   0  164036   9660   6576 S   0.3   0.2   0:04.38 vmtoolsd
   4332 karan     20   0  275100  25696  17488 S   0.3   0.6   0:00.49 xfce4-session
   4411 karan     20   0  400316  76704  58304 S   0.3   1.9   0:06.95 xfwm4
   4553 karan     20   0  450148 117944  49180 S   0.3   2.9   0:03.02 xfdesktop
```

## Changing Priority with the “nice” command

When you start a process, you can set its priority level with the **nice** command. Let’s increment the priority of **/usr/bin/ssh-agent** by **10** (increasing its priority) using the **n** tag.

```
nice -n -10 /usr/bin/ssh-agent
```

```
┌──[root@Kali]─[~]
└─# nice -n -10 /usr/bin/ssh-agent
SSH_AUTH_SOCK=/tmp/ssh-ePY6JdXO8FUY/agent.6241; export SSH_AUTH_SOCK;
SSH_AGENT_PID=6242; export SSH_AGENT_PID;
echo Agent pid 6242;
```

## The “renice” Command

The renice command takes an absolute value between -20 and 19 and sets the priority to that particular level. It also required the PID (process ID).

Let’s give a process of **PID 6242** a higher level of priority (increment it by 20).

```
renice 20 6242
```


## Kill: The deadliest Command

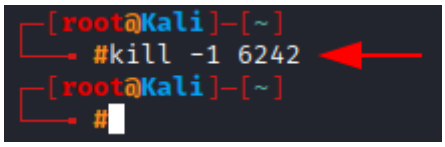
At times, when a process exhibits unusual behaviour or consumes too many system resources, they are called a **zombie process.** In order to stop these kinds of processes, we use the **kill** command.

The kill command has 64 different kill signals, each signifying something slightly different.

(**1** stands for Hangup and is designated to stop the process while **9** is the absolute kill, it forces the process to stop by sending its resources to /dev/null).

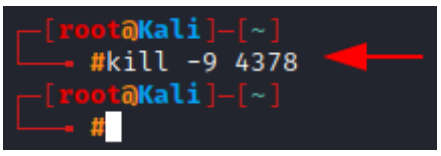
Let's stop the process 6242

```
kill -1 6242
```




And in order to force stop process 4378

```
kill -9 4378
```

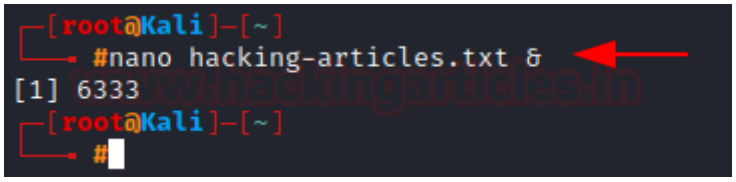



## Running processes in the background

At times, you may want a process to run in the background, and we can do so by simply adding **&** to the end of the command.

Let's run **nano** in the background. (You can see the PID that is generated)

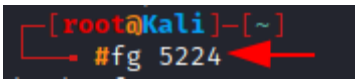
```
nano hacking-articles.txt &
```




### Moving a process to the foreground

If you want to move a process running in the background to the foreground, you can use the **fg** command. Simply type **fg** and then the **process ID**.

(In order to see the background processes in your system simply use the command **jobs**)

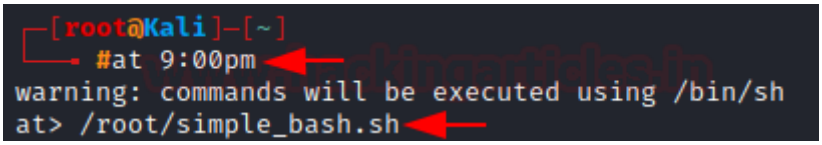



### Scheduling a process

Often one might need to schedule processes to run at a particular time of day. The **at** command is a daemon – a background process which is useful for scheduling a job to run once at some point in the future. While for jobs that occur every day, week, the **crond** is more suited.

Let's execute a **scanning_script.sh** at **9:30pm.**

```
at 9:00pm
/root/simple_bash.sh
```



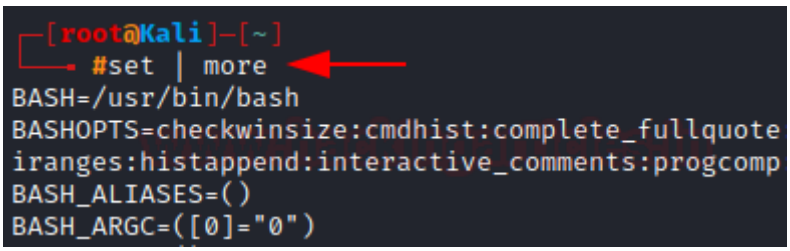

## User Environment Variables

Understanding environment variables is a must when trying to get the most from your Linux system, it is crucial to be able to manage them for optimal performance. Variables are just strings in key-value pairs. There are two types of variables, environment and shell, while the shell variables are only valid for the particular session, the environment variables are system-wide.

## Viewing all the Environment Variables

You can view all your default environment variables by entering **env** into your terminal from any directory, like so:

```
set | more
```



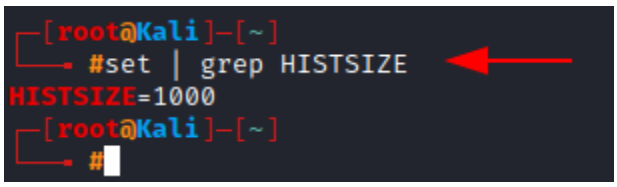

## Filtering for particular variables

Again, using **piping** the output to the **grep** command we can filter out the variables we want.

Let's filter out **HISTSIZE** (history size)

As we can see the history size is set to 1000.

```
set | grep HISTSIZE
```

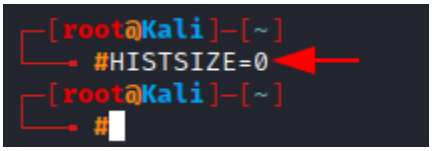



## Changing variable value temporarily

We can change the variable values simply by typing out the variable and equating it to a new value but this new value will only be changed for this particular session, if you open a new terminal window it will change back to its default.

*After running this, you'll see that when you press the up/down arrow keys to recall your previous commands, nothing happens since we changed to a number of commands being stored to 0.*

```
HISTSIZE = 0
```



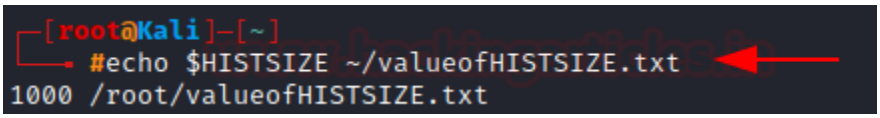

## Making the changes permanent

When changing the variables, it is always best practice to store the default value in say, a text. This way you can always undo your changes.

Let's **echo** the value into a text file name **valueofHISTSIZE** and save it in our working directory by

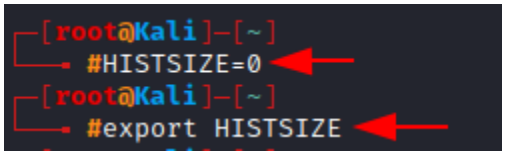
adding ~/

```
echo $HISTSIZE ~/valueofHISTSIZE.txt
```




Now, just like last time change the value of **HISTSIZE** but now we'll execute another command **export.** Which will make this change permanent.

```
HISTSIZE=0

export HISTSIZE
```

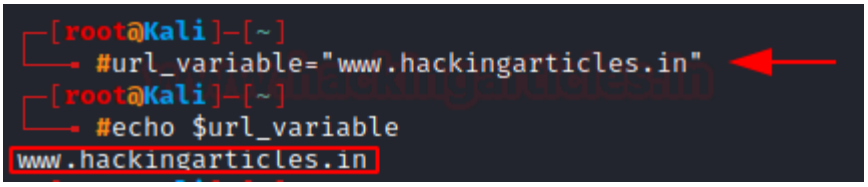



## Creating user-defined variables

You can also design your custom, user-defined variables just by assigning a value to a new value name of your choice.

Let's create a new variable called **URL** which has the value **www.hackingarticles.in.**

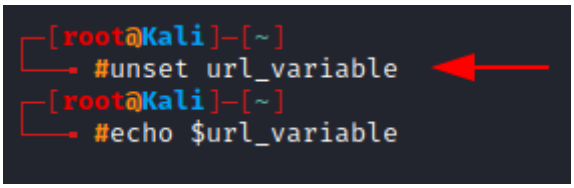
```
url_variable="www.hackingarticles.in"
```




We can also delete this variable by using the **unset** command. Simply typing unset and the name of the variable will do the trick.

As we can see, there is no result despite running the **echo** command.

```
unset url_variable
```




# Bash Scripting, automation and Linux Services

## Bash Scripting Basics

Hackers often have to automate certain commands, sometimes compile them from multiple tools, this can be achieved by writing small computer programs. We'll be learning how to write these programs or scripts in bash.

Going back to the basics, a shell is an interface between the user and the operating system that helps you interact with it, there are a number of different shells that are available for Linux, the one we're using is called **bash.**

The bash shell can run any system commands, utilities and applications. The only thing we'll need to get started is a text editor (like nano, vim). You can choose any as it would not make a difference regardless.

**Shebang: #!**

Let's create a new file: **first_script**. To tell our operating system we're using bash in order to write this script, we use shebang (#!) followed by /bin/bash as seen bellow. Open the file and type:

```
#! /bin/bash
```


## Echo

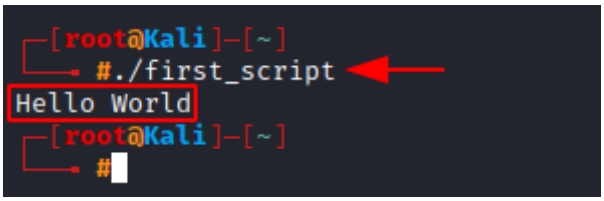
Like the name suggests, we use it to echo back a message or test we want. Let's echo back "Hello World".

```
#! /bin/bash
echo "Hello World"
```


## Running our bashscript

Before we can run our script, we need to give it permission to do so. As we learned earlier, using chmod with +x tag should give the file executable permission.

Adding "**./**" before the filename tells the system that we want to execute this script "first_script".



## Taking user input

To add more functionality to our bash script, we need to discuss variables.

*A Variable in like a bucket, it can hold some value inside the memory. This value can be any text (strings) or even numbers.*

Let's create another script where we learn how to take user input and declare variables.

```
echo "What is your name?"
read name
echo "Welcome, $name"
```

```
 GNU nano 5.3                                    second_script

#! /bin/bash

# This is a comment, the system doesn't read comments as
# part of the code. Simple writing anything after a hash(#)
# get's commented.

echo "What is your name?"

read name

# Here we are using the read command to get user input
# name is our variable which will sotre the value of User's name

echo "Welcome, $name"

# Use "$" in front of your varaiables to use the value that they store
```

Now we can finally see the magic variables, as we run this script. (Be sure to give the script executable permissions first).

```
┌──[root@Kali]─[~]
└─# ./second_script
What is your name?
Raj
Welcome, Raj
┌──[root@Kali]─[~]
└─#
```

## Creating a simple scanner

Let's create a script that would be more useful. We'll make our script scan the entire network for all the active hosts connected to it and find out their IP Addresses.

In order to do so, we'll be using **nmap.** It is simple at an essential tool when it comes to dealing with network penetration testing.

It used to discover the open ports of a system, the services it running and has the capability to detect the operating system as well.

The syntax of nmap is, **nmap <type of scan> <target IP>**.

We will be creating a script that allows us to scan all the device's IP addresses connected to our network. For this, we will be using the -sp tag of nmap. This allows for a simple ping scan, which checks for all the alive connections in your network.

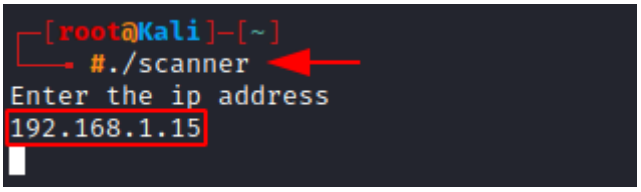
Create a new file: **scanner** and let's gets started.

```
echo "Enter the ip address"
read ip
nma -sp $ip/24 | grep scan | cut -d "" -f 5 | head -n -1
```

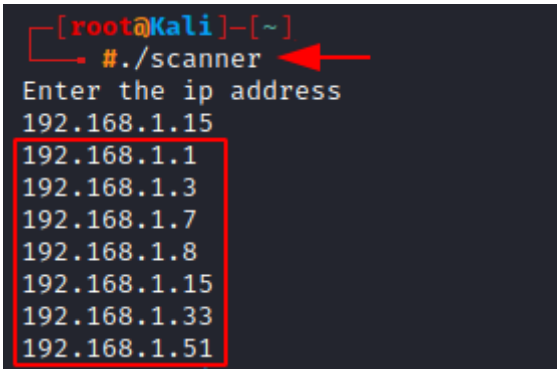
```
 GNU nano 5.3                                    scanner *
#! /bin/bash
echo "Enter the ip address"
read ip
# The value of the IP Address is now stored in IP
nmap -sP $ip/24 | grep scan | cut -d " " -f 5 | head -n -1

# -sP: used for a ping sweep (pinging all possible ip address to see if they up)
# using grep to pull out the line with the ip address
# cut -d " " -f 5: Gets rid of the first 4 words in front of the IP address
# head -n -1: Delete the last line- "Namp done"


# We are using the text manupilation concepts we learned about in the first article
# Play around with grep, cut and head command or go back to the pervoius article
# to get a good grip on them.
```

Let's give our new bash script executable permissions, and run it.

Enter your IP Address.




Now we can see, all the different devices and their IP Address's connected to your network.




## Scheduling Your Tasks

At times one is required to schedule tasks, such as a backup of your system. In Linux, we schedule jobs we want to run without having to do it manually or even think about it. Here, we'll learn about the **cron** daemon and **crontab** to run our scripts automatically.

The **crond** is a daemon that runs in the background, it checks for the cron table – **crontab** if there are any specific commands to run at times specified. Altering the crontab will allow us to execute our task.

The cron table file is located at /**etc/crontab**. It has a total of 7 fields, where the first 5 are used to specify the time for it to run, the 6$^{th}$ field is for specifying the user and the last one is for the path to the command you want to run.

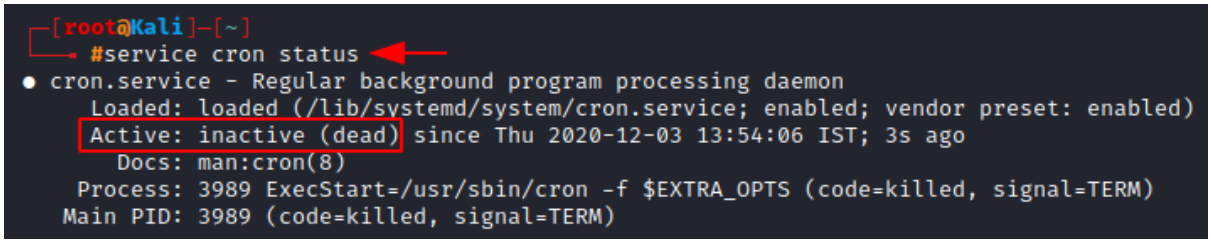
Here's a table to summarize the first 5 fields:

| Field | Unit it changes | Syntax to enter |
|---|---|---|
| 1. | Minute | 0-59 |
| 2. | Hour | 0-23 |
| 3. | Day of the month | 1-31 |
| 4. | Month | 1-12 |
| 5. | Day of the week | 0-7 |

## Scheduling our bash script- scanner

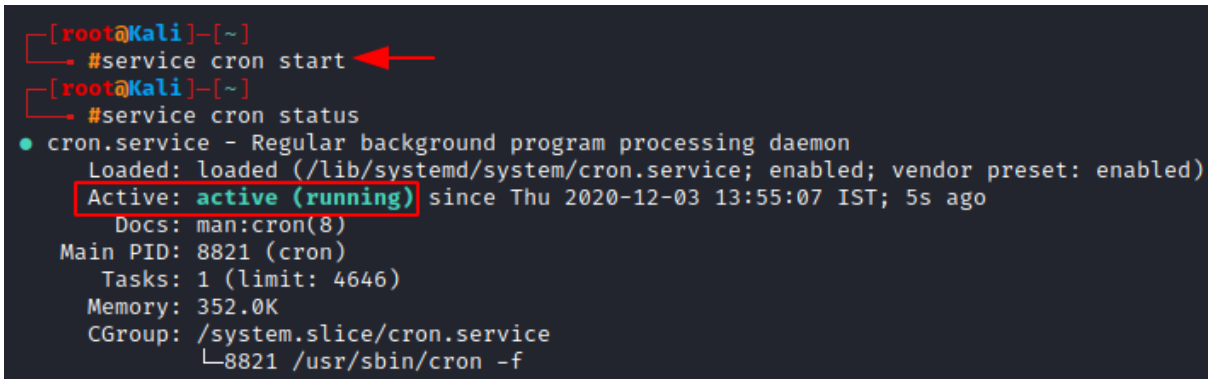
First, let's check whether the cron daemon is running or not by typing,

```
service cron status
```



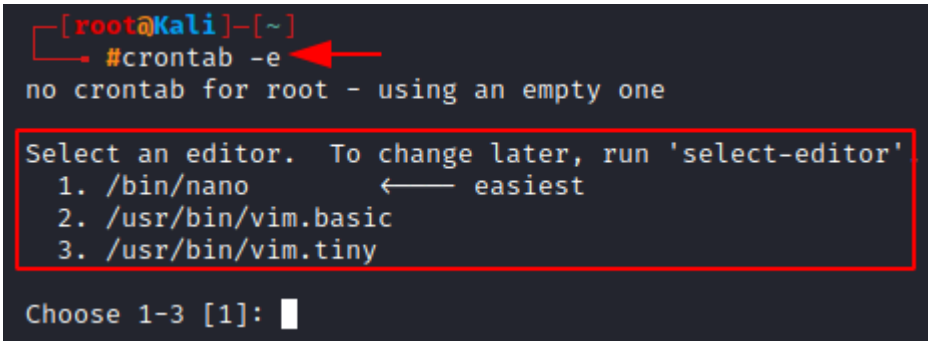

Since it shows inactive, we can start the service by typing

```
service cron start
```

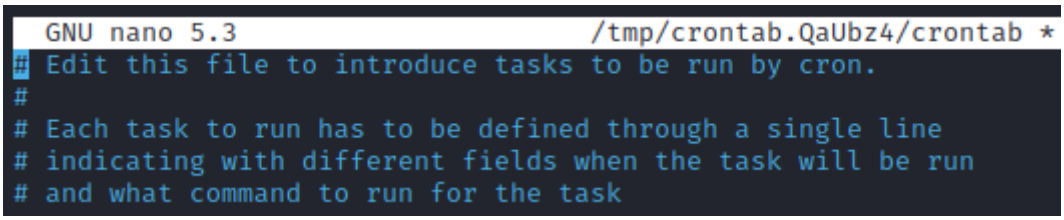



Now, open the cron table in order to edit it. Type crontab in the terminal, followed by the **"-e"** flag (e stands for edit).

```
crontab -e
```



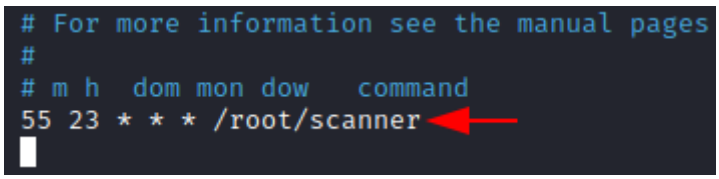

It gives you an option to select any text editor, we'll be choosing nano as we've been working with it so far. So, enter 1.




Now scroll down and simply enter all the 7 fields we learned about, to schedule the task.

Let's say we want to see all the devices connected to our network before we sleep, so we'll execute our scanner script every day at 11:55 PM automatically. Type the following,

```
55 23 * * * /root/scanner
```




## Initiate Jobs at startup using rc scripts

*Whenever you switch on your Linux machine, a number of process run which helps in setting up the environment that you'll use. The scripts that run are known as* ***rc scripts.***

*When booting up your machine, the kernel starts a daemon known as* ***init.d*** *which is responsible for running these scripts.*

*The next thing we should know about is,* ***Linux Runlevels.*** *Linux has multiple runlevels, which tell the system what services should be started at the bootup.*

*Here is a table indicating the above:*

| | |
|---|---|
| 0 | Halt the system |
| 1 | Single-user/minimal mode |
| 2-5 | Multiuser modes |
| 6 | Reboot the system |

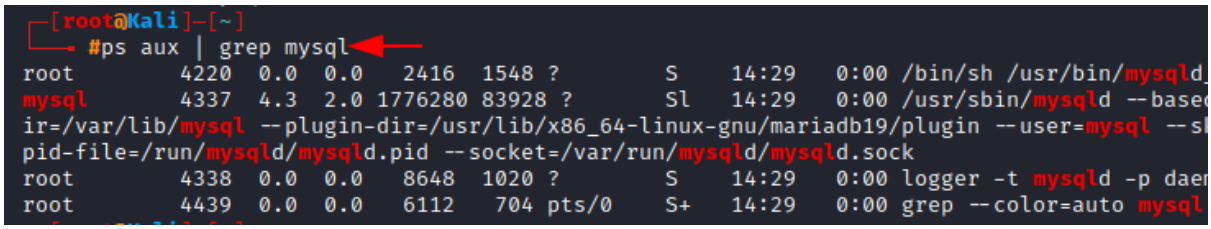
Let's add a service to the rc.d now. This can be done using **the update-rc.d** command. This enables you to add or remove services from the **rc script.**

We will enable MySQL to start every time we boot. Simply write MySQL after update-rc.d and follow it with defaults (**options: remove|defaults|disable|enable>)**

```
update-rc.d mysql defaults
```

Now, we restart the system, you'll see MYSQL has already been started.

We can check for it using the ps aux and grep command as we learned earlier.




# Using Services in Linux

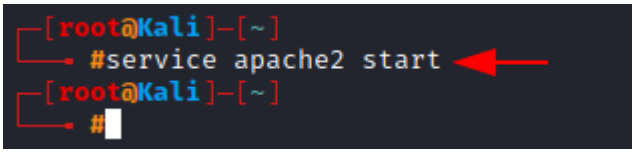
Services in Linux is a common way to denote an application that is running in the background for you to use. Multiple services come preinstalled in your Linux machine, one of the most common ones is Apache Web Server, which helps us creating and deploying Web Servers or OpenSSH which allows you to connect to another machine. Let's dig deeper into these services, to understand their inner function, which will help us in abusing them.

## Playing with services (start, stop, status, restart)

Before we begin, we should know how to manage these services. The basic syntax to do so is, service <service_name> <start|stop|restart|status>

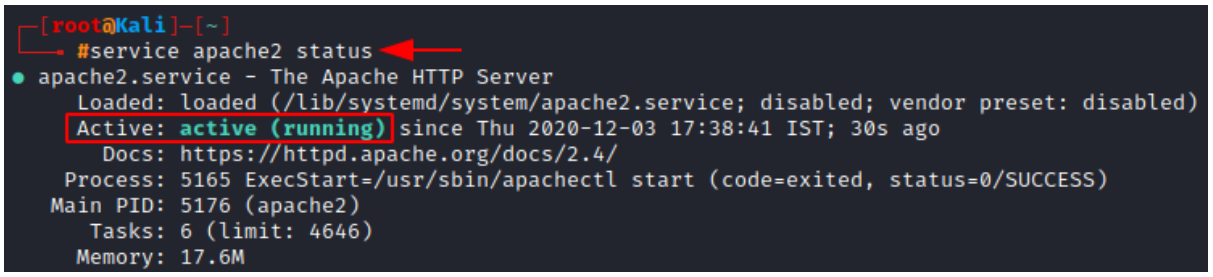
Let's start the apache2 server.

```
service apache2 start
```



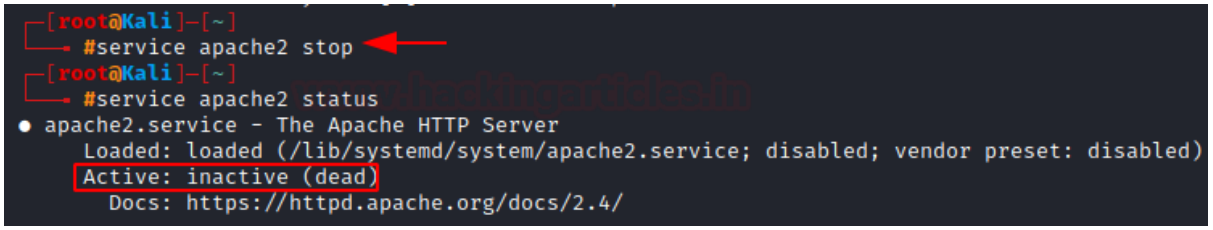

Now, we use the status tag to check whether the service is up or not

```
service apache2 status
```

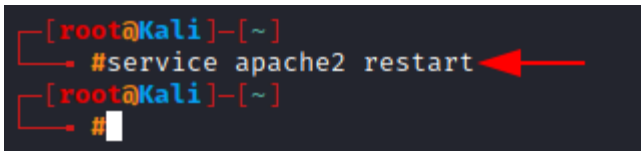



To stop this service, we type

```
service apache2 stop
```




At times when the service does a faulty start or you've changed a particular configuration, you might want to restart it, to reflect the changes. This can be done with the restart option.



## Creating an HTTP Web Server using Apache webserver

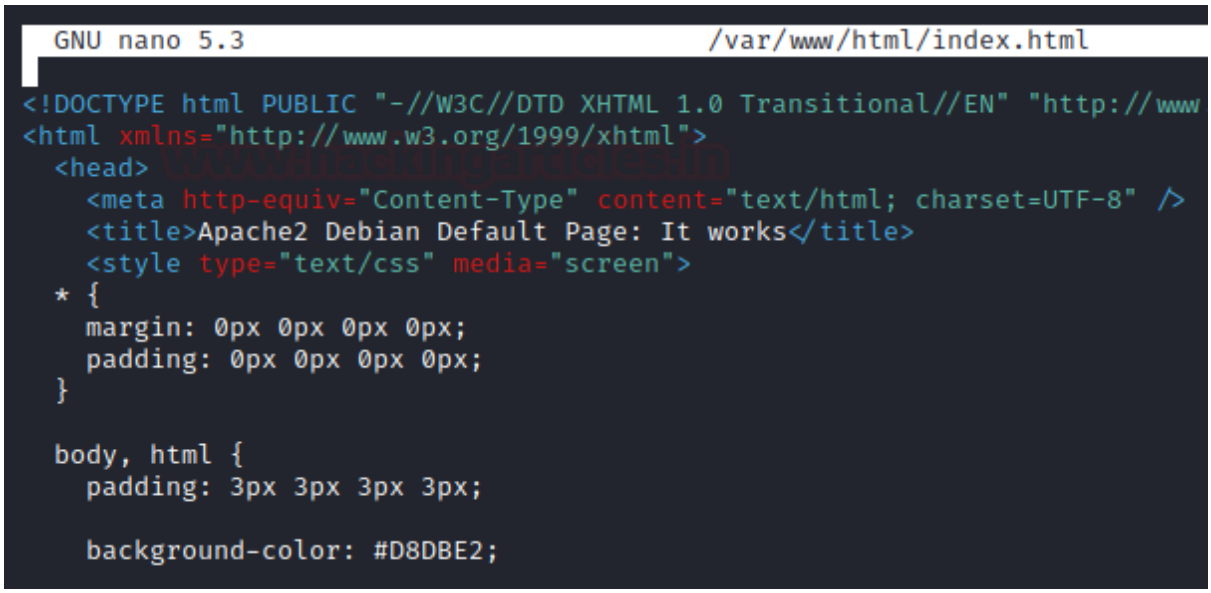
More than 60% of the world's web servers use Apache, it is one of the most commonly used services. As a pen-tester, it is critical to understand how apache works. So, let's deploy our own web server and get familiar with Apache.

Start the apache2 service (if you haven't already) and now we are going to the HTML file that will get displayed on the browser, apache's default web page is present at: **/var/www/html/index.html**

Let's open this with **nano** and write some of our HTML code.

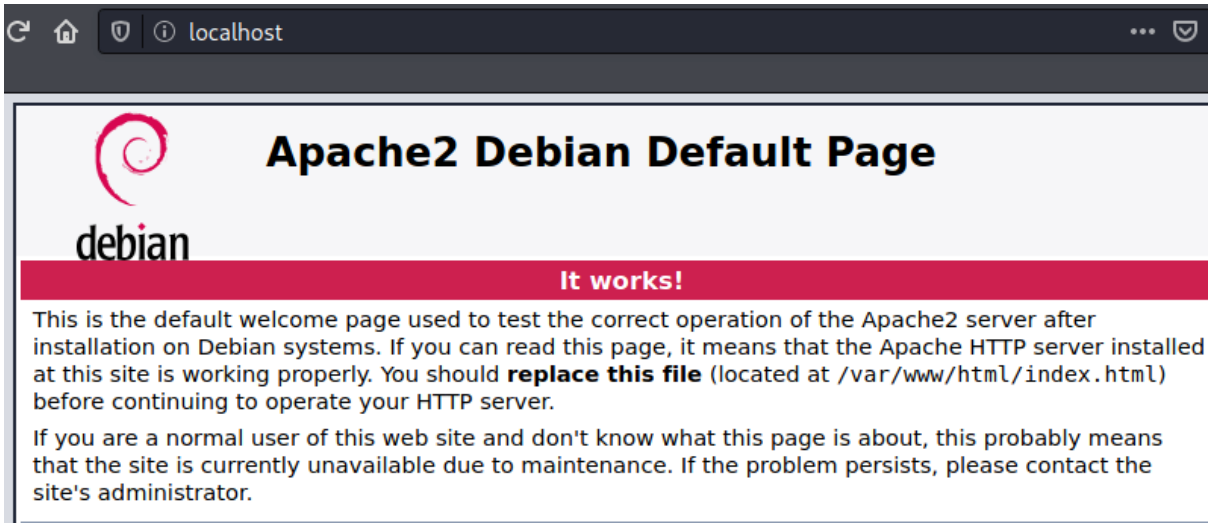
```
nano /var/www/html/index.html
```


We see the html code present by default




Save the file a now to see what the Apache server displays, we can go to the browser and type

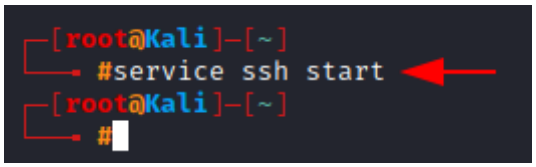
```
http://localhost
```




## Getting familiar with OpenSSH

Secure Shell or SSH is basically what enables us to connect to a terminal on a remote system, securely. Unlike its ancestor **telnet** which was used quite some years back, the channel SSH using for its communication is encrypted and hence more secure.

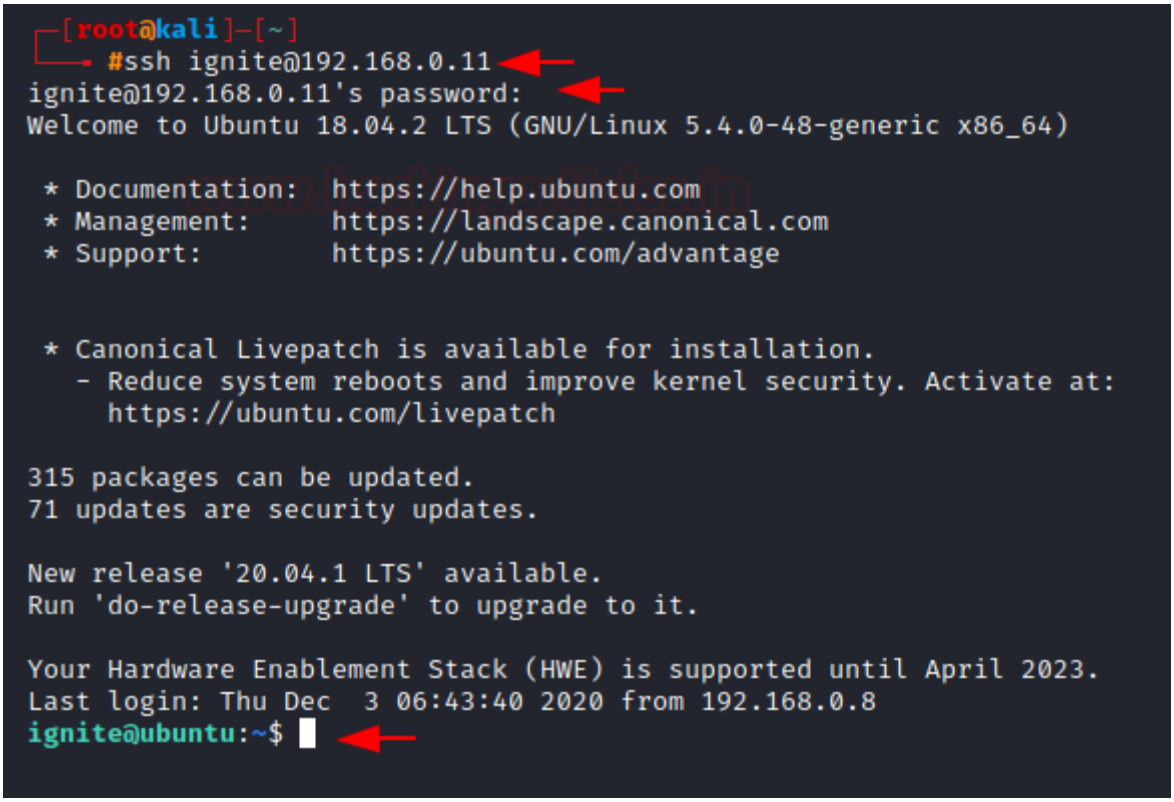
Again, before we start using the SSH service, we have to start it first.




Now to connect to a remote system and get access to its terminal, we type SSH followed the <username>@<ip address>. Let's connect to my host machine.

```
ssh ignite@192.168.0.11
```

We have successfully connected to another machine called **ubuntu** with the user **ignite**

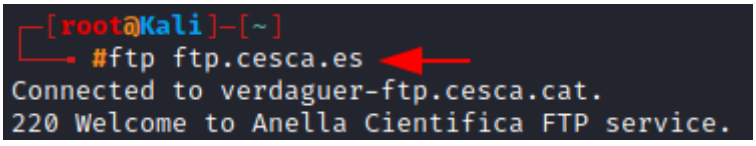



## Working with FTP

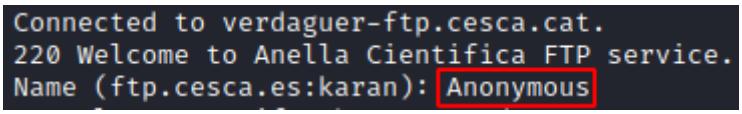
Let's talk about the **File Transfer Protocol** or FTP. This protocol is generally used, as the name suggests for transfer of files via the command line. Here we'll try connecting to an ftp server and download files from it, via the **ftp command.**

To access an ftp server, we type ftp followed by the domain name or the IP Address. Here's an example:

```
ftp ftp.cesca.es
```




Now it's going to ask you to enter a name, we can type **anonymous** here since this server allows it.




Now it's going to ask for the password, and we type **anonymous** there as well.

```
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

As we can see we've been logged in successfully. Now with the help of the basic navigation commands we learned in the first part of this article, we can ls to list the contents.

```
ftp> ls
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
drwxr-xr-x    5 0        0            4096 Aug 21  2014 anella
drwxrwxr-x   55 1005     1005         4096 Dec 02 12:47 centos
drwxr-xr-x    9 0        0            4096 Dec 03 09:56 debian
drwxr-xr-x    5 406      75           4096 Sep 27 02:29 debian-cd
drwxr-xr-x    4 0        0            4096 Dec 20  2012 scientific_linux
drwxr-xr-x    4 0        0            4096 Dec 18  2012 ubuntu
226 Directory send OK.
```

Navigate around for a file you want to download. Let's try download the file at

**ubuntu/release/favicon.ico,** simply type get and the file name

```
get favicon.ico
```

```
ftp> get favicon.ico
local: favicon.ico remote: favicon.ico
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for favicon.ico (1150 bytes).
226 Transfer complete.
1150 bytes received in 0.03 secs (38.9582 kB/s)
```

To exit the ftp session, type **bye.** Now we can ls and see the file we just downloaded.

```
ftp> bye
221 Goodbye.
┌─[root@Kali]─[~]
└──╼ #ls
Desktop    Downloads    first_script  Pictures
Documents  favicon.ico  Music         Public
```

# Conclusion

I hope this report, which covers basic and advanced Linux topics like managing networks, process management, scripting & automation, has helped you grasp the Linux operating system better.

Hence, one can make use of these commands as a cybersecurity professional to assess vulnerabilities on systems and keep these systems away from threat.

# References


- https://www.hackingarticles.in/linux-for-beginners-a-small-guide/
- https://www.hackingarticles.in/linux-for-beginners-a-small-guide-part-2/
- https://www.hackingarticles.in/linux-for-beginners-a-small-guide-part-3/