

Linux for Beginners: A Small Guide (Part 3)

December 6, 2020 By Raj Chandel

Let's cover more advanced concepts and pick up where we left in [part 2 of this article](#) where we learned somehow to interact and manage network devices, discover the different processes running on your system and getting a grip of the usage of environment variables.

In this article, we'll be focusing on some advanced Linux fundamentals, like scripting, automation and Linux services that help us with getting more out of our Operating system, making our workflow optimal. Let's dig right into it.

Table of Content

- Bash scripting concepts
- Scheduling your tasks
- Using services in Linux
 - Apache webserver
 - OpenSSH
 - FTP
- Conclusion

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 below. Open the file and type:

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

```
GNU nano 5.3
#!/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"
```

```
GNU nano 5.3
#!/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".

```
[root@Kali]~# ./first_script
Hello World
[root@Kali]~#
```

Taking user input

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

A Variable is 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 store the value of User's name

echo "Welcome, $name"

# Use "$" in front of your variables 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 get started.

```
echo "Enter the ip address"
read ip
nmap -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- "Nmap done"

# We are using the text manipulation 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.

```
[root@Kali]~# ./scanner
Enter the ip address
192.168.1.15
```

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

```
[root@Kali]~# ./scanner
Enter the ip address
192.168.1.15
192.168.1.1
192.168.1.3
192.168.1.7
192.168.1.8
192.168.1.15
192.168.1.33
192.168.1.51
```

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 6th 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
```

```
[root@Kali]~# service cron status
● cron.service - Regular background program processing daemon
   Loaded: loaded (/lib/systemd/system/cron.service; enabled; vendor preset: enabled)
   Active: inactive (dead) since Thu 2020-12-03 13:54:06 IST; 3s ago
     Docs: man:cron(8)
   Process: 3989 ExecStart=/usr/sbin/cron -f $EXTRA_OPTS (code=killed, signal=TERM)
  Main PID: 3989 (code=killed, signal=TERM)
```

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

```
service cron start
```

```

[root@Kali]~# service cron start
[root@Kali]~# service cron status
● cron.service - Regular background program processing daemon
   Loaded: loaded (/lib/systemd/system/cron.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2020-12-03 13:55:07 IST; 5s ago
     Docs: man:cron(8)
    Main PID: 8821 (cron)
      Tasks: 1 (limit: 4646)
     Memory: 352.0K
    CGroup: /system.slice/cron.service
            └─8821 /usr/sbin/cron -f

```

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
```

```

[root@Kali]~# crontab -e
no crontab for root - using an empty one

Select an editor. To change later, run 'select-editor'
 1. /bin/nano          ← easiest
 2. /usr/bin/vim.basic
 3. /usr/bin/vim.tiny

Choose 1-3 [1]:

```

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.

```

GNU nano 5.3 /tmp/crontab.QaUbz4/crontab *
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task

```

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
```

```

# For more information see the manual pages
#
# m h dom mon dow  command
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.

```
[root@Kali]~# ps aux | grep mysql
root      4220  0.0  0.0   2416  1548 ?        S      14:29   0:00 /bin/sh /usr/bin/mysqld
mysql     4337  4.3  2.0 1776280 83928 ?        Sl     14:29   0:00 /usr/sbin/mysqld --based
ir=/var/lib/mysql --plugin-dir=/usr/lib/x86_64-linux-gnu/mariadb19/plugin --user=mysql --s
pid-file=/run/mysqld/mysqld.pid --socket=/var/run/mysqld/mysqld.sock
root      4338  0.0  0.0   8648  1020 ?        S      14:29   0:00 logger -t mysqld -p dae
root      4439  0.0  0.0   6112   704 pts/0    S+     14:29   0:00 grep --color=auto mysql
```

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.

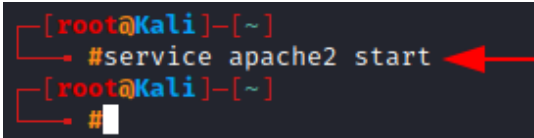
Multiple services come preinstalled in your Linux machine, one of the most common ones is Apache Web Server, which helps us creating and deploying Web Servers or OpenSSH which allows you to connect to another machine. Let's dig deeper into these services, to understand their inner function, which will 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
```

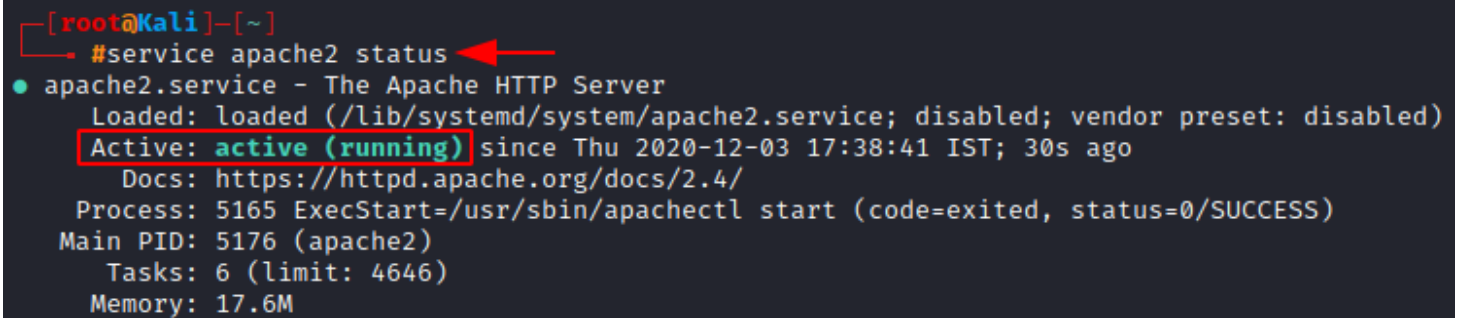


```
[root@Kali]~#  
#service apache2 start  
[root@Kali]~#
```

A red arrow points to the command `#service apache2 start`.

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

```
service apache2 status
```

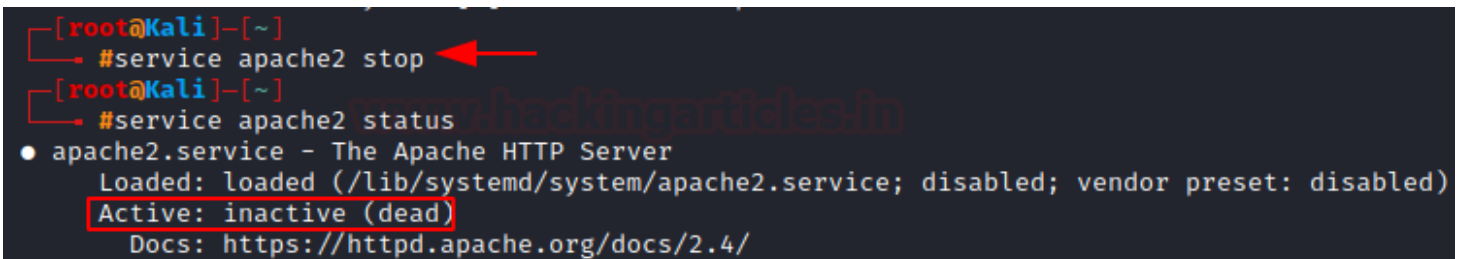


```
[root@Kali]~#  
#service apache2 status  
● apache2.service - The Apache HTTP Server  
   Loaded: loaded (/lib/systemd/system/apache2.service; disabled; vendor preset: disabled)  
   Active: active (running) since Thu 2020-12-03 17:38:41 IST; 30s ago  
     Docs: https://httpd.apache.org/docs/2.4/  
   Process: 5165 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUCCESS)  
   Main PID: 5176 (apache2)  
     Tasks: 6 (limit: 4646)  
    Memory: 17.6M
```

A red arrow points to the command `#service apache2 status`. The output shows the service is active (running).

To stop this service, we type

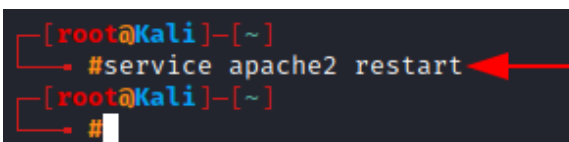
```
service apache2 stop
```



```
[root@Kali]~#  
#service apache2 stop  
[root@Kali]~#  
#service apache2 status  
● apache2.service - The Apache HTTP Server  
   Loaded: loaded (/lib/systemd/system/apache2.service; disabled; vendor preset: disabled)  
   Active: inactive (dead)  
     Docs: https://httpd.apache.org/docs/2.4/
```

A red arrow points to the command `#service apache2 stop`. The subsequent status check shows the service is inactive (dead).

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.



```
[root@Kali]~#  
#service apache2 restart  
[root@Kali]~#
```

A red arrow points to the command `#service apache2 restart`.

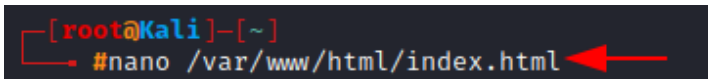
Creating an HTTP Web Server using Apache webserver

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.

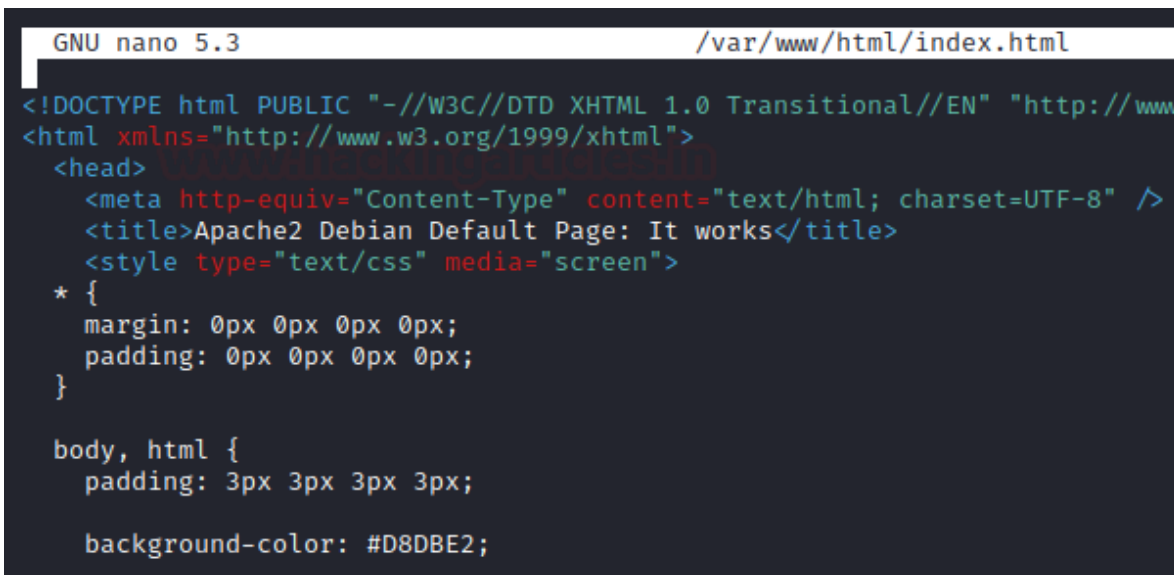
Start the apache2 service (if you haven't already) and now we are going to the HTML file that will get displayed on the browser, apache's default web page is present at: **/var/www/html/index.html**

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

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

A terminal window with a dark background. The prompt is [root@Kali]~. The command #nano /var/www/html/index.html has been entered. A red arrow points to the end of the command.

We see the html code present by default

A terminal window showing the GNU nano 5.3 editor. The file being edited is /var/www/html/index.html. The code is a default Apache2 Debian default page. It includes a DOCTYPE declaration, an XML namespace, a head section with a meta tag for content type, a title "Apache2 Debian Default Page: It works", and a style tag for CSS. The body and html elements have padding and a background color of #D8DBE2.

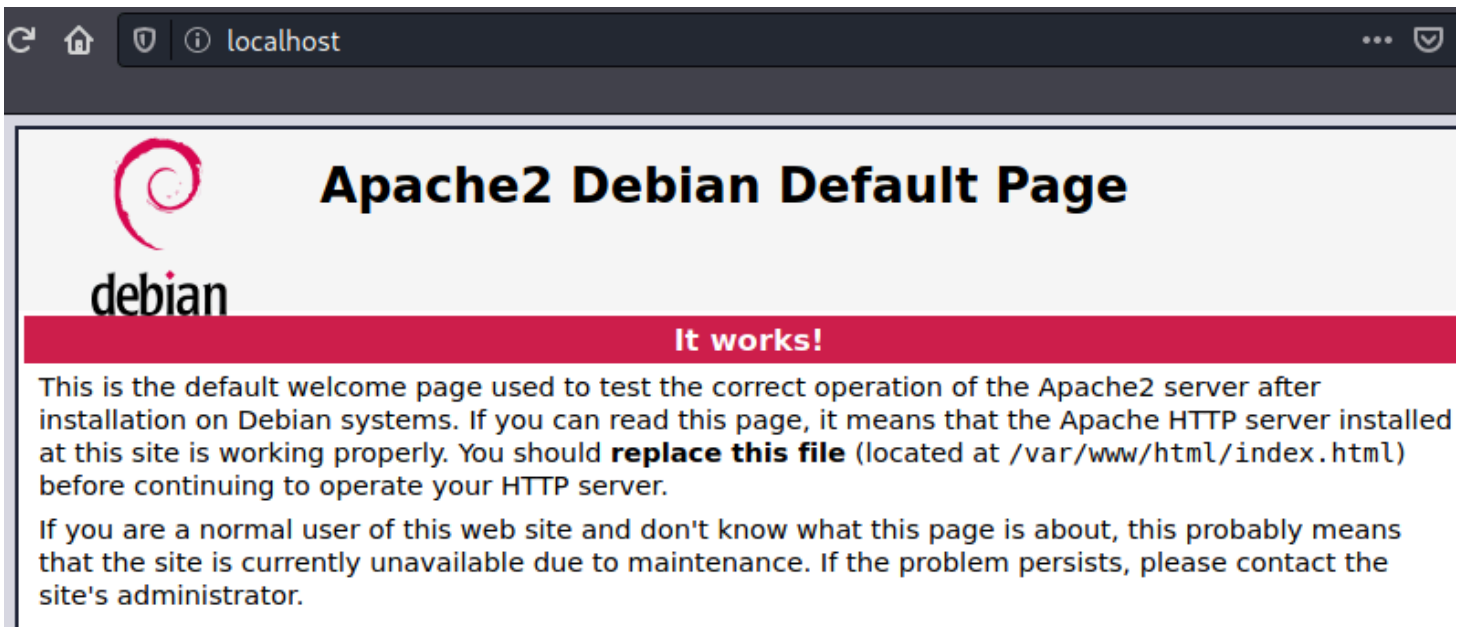
```
GNU nano 5.3 /var/www/html/index.html
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Apache2 Debian Default Page: It works</title>
    <style type="text/css" media="screen">
      * {
        margin: 0px 0px 0px 0px;
        padding: 0px 0px 0px 0px;
      }

      body, html {
        padding: 3px 3px 3px 3px;

        background-color: #D8DBE2;
```

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.

```
[root@Kali]~#  
#service ssh start  
[root@Kali]~#
```

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**

```
[root@kali]~# ssh ignite@192.168.0.11
ignite@192.168.0.11's password:
Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 5.4.0-48-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

315 packages can be updated.
71 updates are security updates.

New release '20.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Your Hardware Enablement Stack (HWE) is supported until April 2023.
Last login: Thu Dec  3 06:43:40 2020 from 192.168.0.8
ignite@ubuntu:~$
```

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
```

```
[root@kali]~# ftp ftp.cesca.es
Connected to verdaguer-ftp.cesca.cat.
220 Welcome to Anella Cientifica FTP service.
```

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

```
Connected to verdaguer-ftp.cesca.cat.
220 Welcome to Anella Cientifica FTP service.
Name (ftp.cesca.es:karan): Anonymous
```

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

Hope this article helped you understand some more advanced concepts of the Linux operating system. Do practice these commands and get familiar with the services we used. Continue your learning journey, search for new things, try them yourself, there is a giant Linux community to help you at any step, cheers!