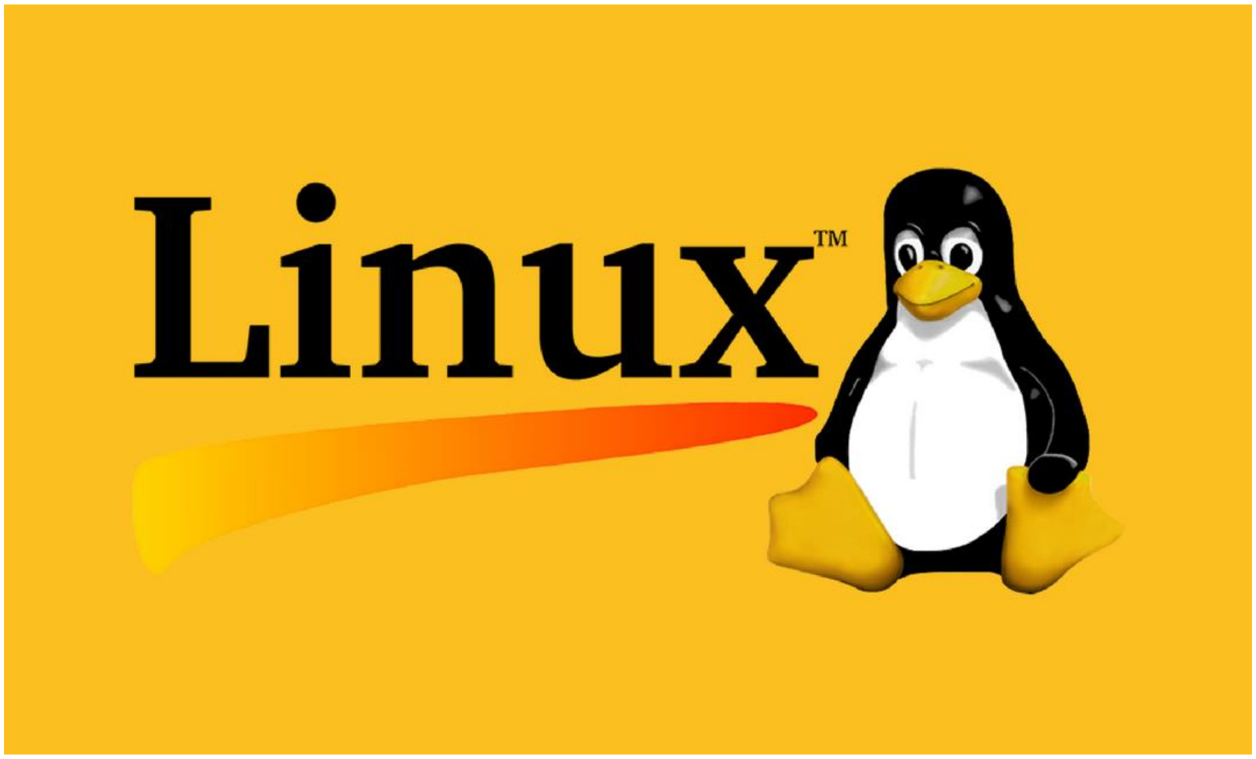
Linux Ultimate Guide



Author & Credits:

Occupytheweb

https://creator.wonderhowto.com/occupythewebotw/

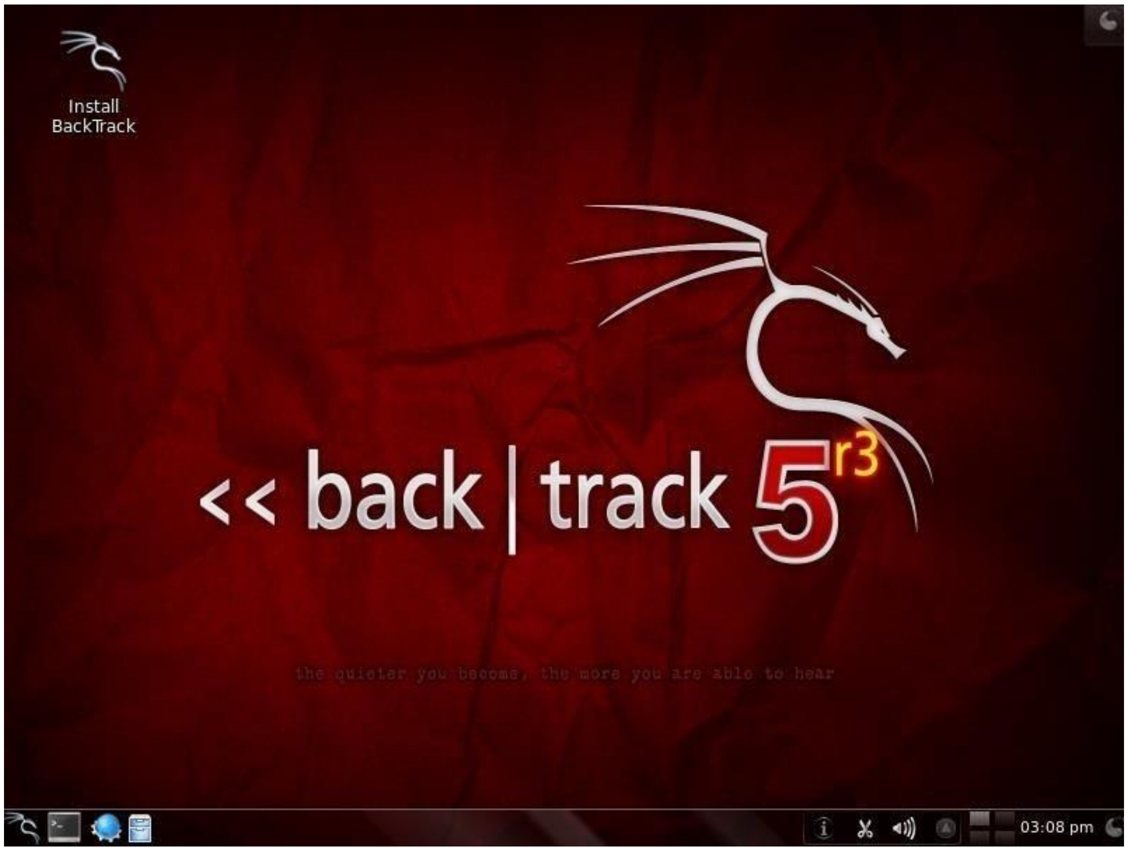
Getting Started

Step 1 Boot up Linux

Once you've booted up BackTrack, logged in as "root" and then type:

- **bt > startx**

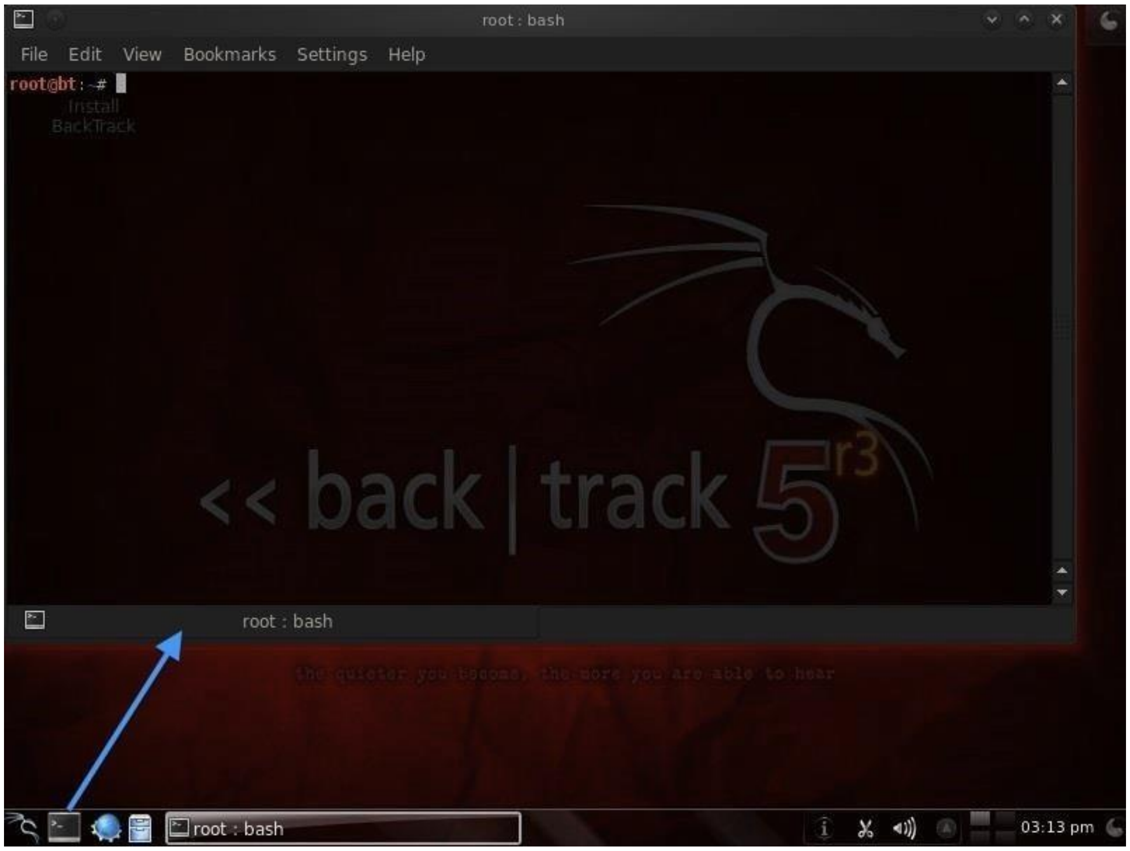
You should have a screen that looks similar to this.



Step 2 Open a Terminal

To become proficient in Linux, you MUST master the terminal. Many things can be done now in the various Linux distributions by simply pointing and clicking, similar to Windows or Mac OS, but the expert hacker must know how to use the terminal to run most of the hacking tools.

So, let's open a terminal by clicking on the terminal icon on the bottom bar. That should give us a screen that looks similar to this.


root : bash
File Edit View Bookmarks Settings Help
Install BackTrack
<< back | track 5 r3
root : bash
the quieter you become, the more you are able to hear
root : bash
03:13 pm

```
root@bt:~#
```

If you've ever used the command prompt in Windows, the Linux terminal is similar, but far more powerful. Unlike the Windows command prompt, you can do EVERYTHING in Linux from the terminal and control it more precisely than in Windows.

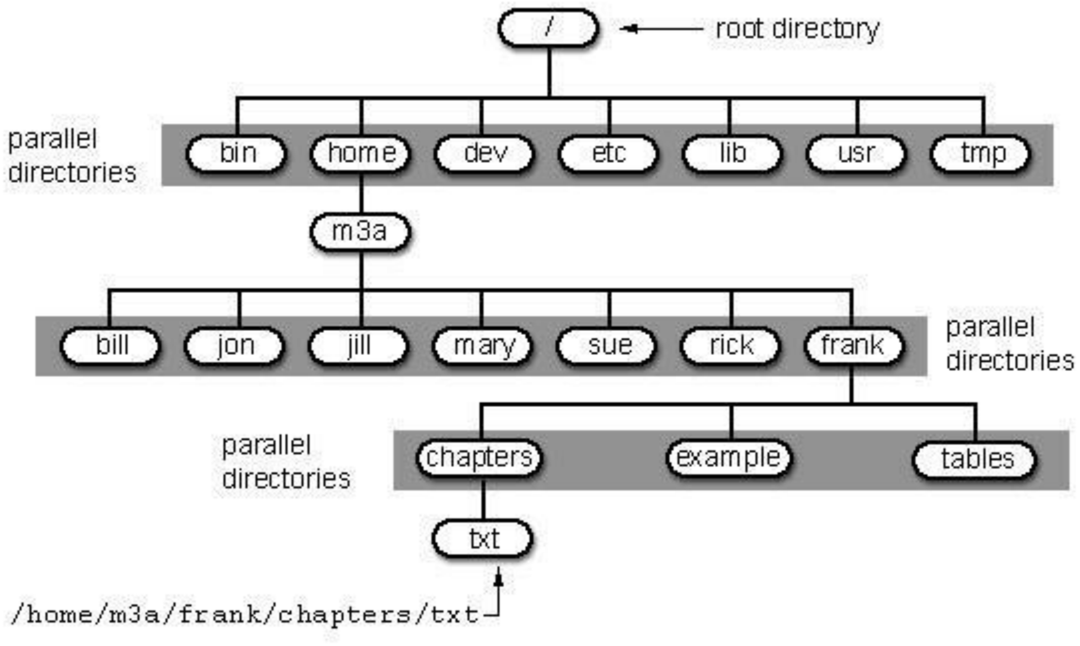
It's important to keep in mind that unlike Windows, Linux is case-sensitive. This means that "Desktop" is different from "desktop" which is different from "DeskTop". Those who are new to Linux often find this challenging, so try to keep this in mind.

Step 3 Examine the Directory Structure

Let's start with some basic Linux. Many beginners get tripped up by the structure of the file system in Linux. Unlike Windows, Linux's file system is not linked to a physical drive like in Windows, so we don't have a **c:\** at the beginning of our Linux file system, but rather **a /**.

The forward slash (**/**) represents the "root" of the file system or the very top of the file system. All other directories (folders) are beneath this directory just like folders and sub-folders are beneath the c:\ drive.

To visualize the file system, let's take a look at this diagram below.



It's important to have a basic understanding of this file structure because often we need to navigate through it from the terminal without the use of a graphical tool like Windows Explorer.

A couple key things to note in this graphical representation:

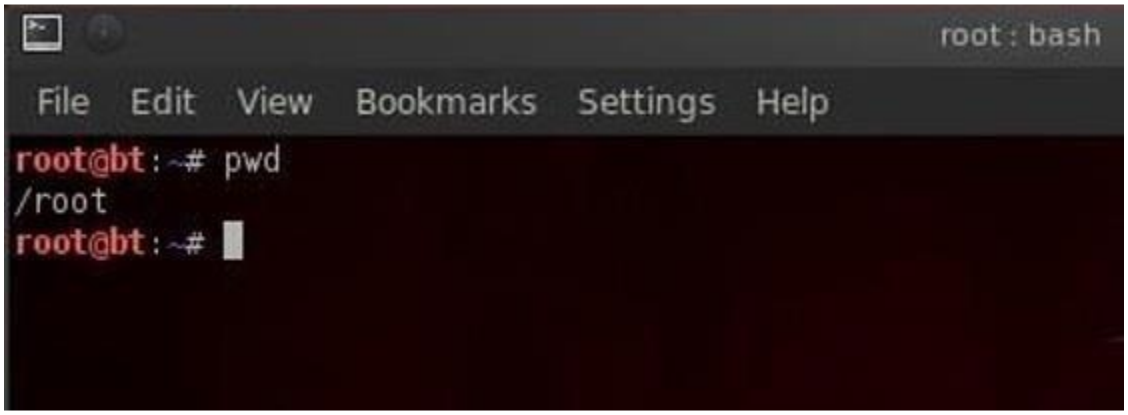
- The **/bin** directory is where binaries are stored. These are the programs that make Linux run.
- **/etc** is generally where the configuration files are stored. In Linux, nearly everything is configured with a text file that is stored under **/etc**.
- **/dev** directory holds device files, similar to Windows device drivers.
- **/var** is generally where log files, among other files, are stored.

Step 4 Using Pwd

When we open a terminal in BackTrack, the default directory we're in is our "home" directory. As you can see from the graphic above, it's to the right of the "root" directory or one level "below" root. We can confirm what directory we are in by typing:

- **bt > pwd**

pwd stands for "present working directory" and as you can see, it returns "**/root**" meaning we're in the root users directory (don't confuse this with the top of the directory tree "root." This is the root **users** directory).



```
root@bt:~# pwd
/root
root@bt:~# 
```

pwd is a handy command to remember as we can use it any time to tell us where we are in the directory tree.

Step 5 Using Cd Command

We can change the directory we're working in by using the **cd** (change directory) command. In this case, let's navigate "up" to the top of the directory structure by typing:

- **bt > cd ..**

The **cd** command followed by the double dots (**..**) says, "move me up one level in the directory tree." Notice that our command prompt has changed and when we type **pwd** we see that Linux responds by telling us we are in the "**/**" or the top of the directory tree (or the root directory).

- **bt > pwd**

Creating Directories & Files

Let's open up BackTrack and getting started learning more Linux for the aspiring hacker.

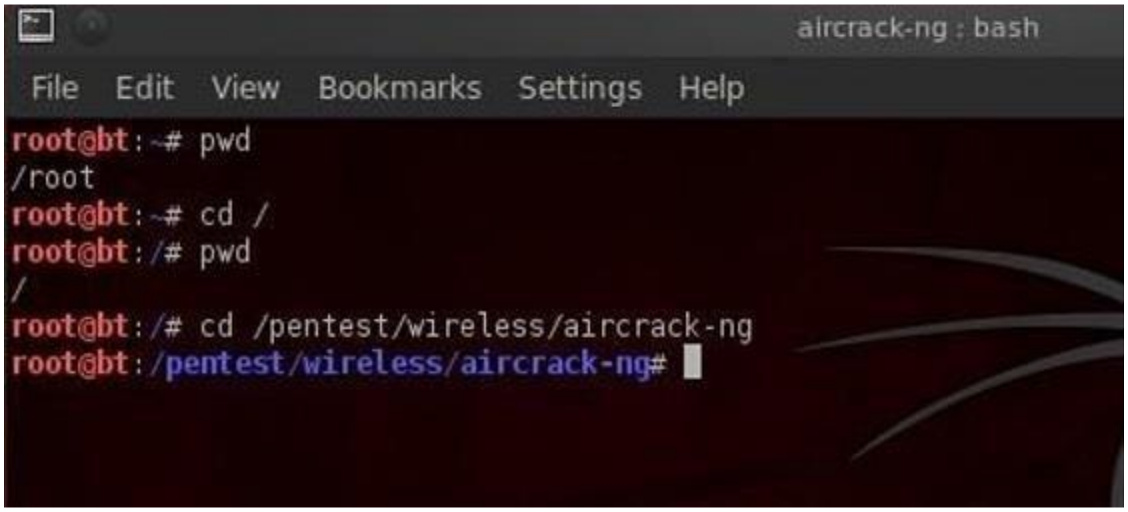
Step 1 Change Directory (Cd)

We can change directories in multiple ways with **cd**..we can use **cd ..** to move up one level in the directory tree. We can also move directly to the root directory by typing **cd /** or move to our home directory by **cd ~**.

More often, we will use **cd** to move to a directory by using the absolute path of the directory. This mean that we write out the entire path of the directory we want to move to after **cd**. We can also move to the directory by using the relative path of the directory. This means that we don't need to write the entire path, but simply use the path that we're currently in and append to it. Let's look at some examples.

Let's say we're in our root user directory in **BackTrack** and we want to move to the **aircrack-ng** directory (we'll be doing some aircrack tutorials soon). We can simply type:

- **bt > cd /pentest/wireless/aircrack-ng**

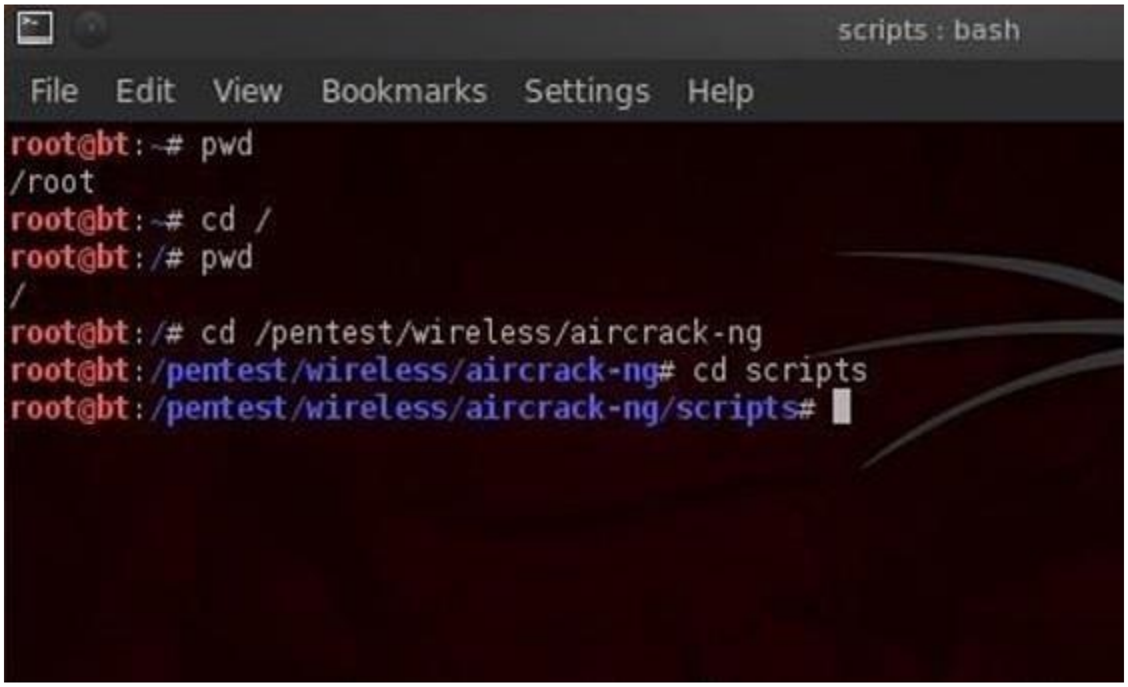


```
root@bt:~# pwd
/root
root@bt:~# cd /
root@bt:/# pwd
/
root@bt:/# cd /pentest/wireless/aircrack-ng
root@bt:/pentest/wireless/aircrack-ng#
```

This will take us directly to the **aircrack-ng** directory.

Now let's say we want to go to the **scripts sub-directory** within **aircrack-ng**. We could type out the full path to the sub-directory, but it's much simpler to type the relative path from where we are. We know we are **/pentest/wireless/aircrack-ng**, so type:

- **bt > cd scripts**



```
root@bt:~# pwd
/root
root@bt:~# cd /
root@bt:/# pwd
/
root@bt:/# cd /pentest/wireless/aircrack-ng
root@bt:/pentest/wireless/aircrack-ng# cd scripts
root@bt:/pentest/wireless/aircrack-ng/scripts#
```

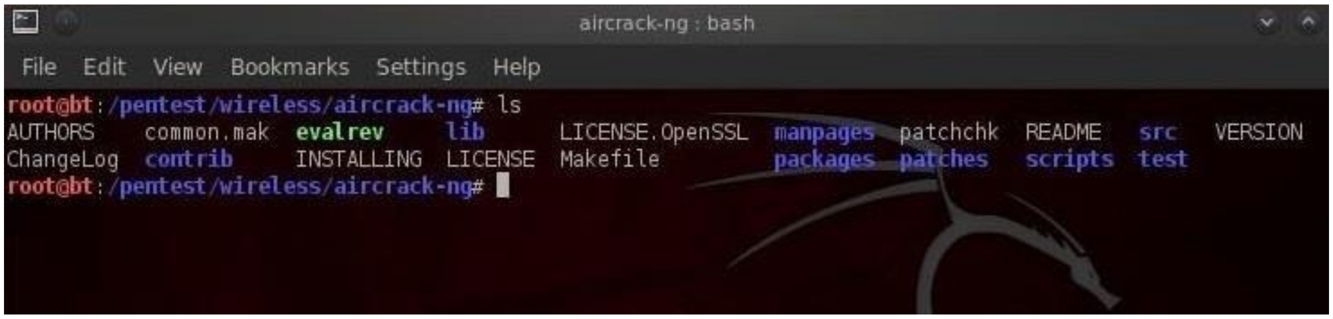
And that takes us to the **scripts sub-directory** within **aircrack-ng** or **/pentest/wireless/aircrack-ng/scripts**.

Once again, it's critical to emphasize that Linux is case sensitive, so typing the directory without the proper case will result in the error message, "no such file or directory".

Step 2 Listing Command (Ls)

Once of most used and important commands in Linux is **ls** or **list**. This command is used to list the contents of a directory or sub-directory so that we can see the contents. It's very similar to the **dir** command in Windows. So let's use it in the **aircrack-ng** directory;

- **bt > ls**



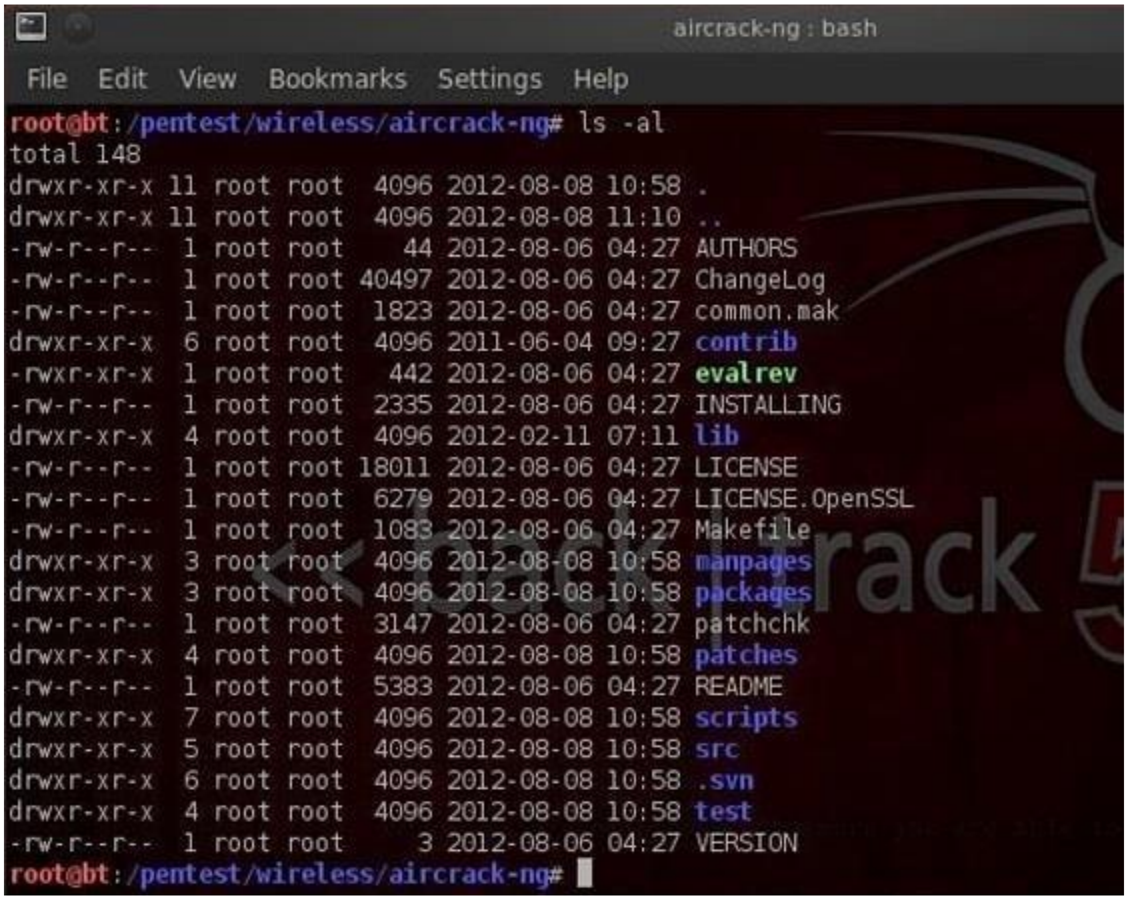
```
root@bt:/pentest/wireless/aircrack-ng# ls
AUTHORS    common.mak  evalrev     lib      LICENSE.OpenSSL  manpages  patchchk  README   src   VERSION
ChangeLog  contrib     INSTALLING  LICENSE  Makefile         packages  patches   scripts  test
root@bt:/pentest/wireless/aircrack-ng#
```

We can see that Linux listed all the files and directories within the **aircrack-ng** directory. Linux allows us to modify its commands by using switches; these are usually letters preceded by the dash **(-)**. With **ls**, it's helpful to use two of theses switches, **-a** and **-l**.

The **-a** switch means all, so when we use it, Linux will list all files and directories, even those that are hidden. When we use the **-l** switch, it gives us a long listing, meaning it gives us info on the security permissions, the size, the owner, the group of the file or directory, when it was created, etc.

Let's type:

- **bt > ls -la**



```
root@bt:/pentest/wireless/aircrack-ng# ls -al
total 148
drwxr-xr-x 11 root root  4096 2012-08-08 10:58 .
drwxr-xr-x 11 root root  4096 2012-08-08 11:10 ..
-rw-r--r--  1 root root    44 2012-08-06 04:27 AUTHORS
-rw-r--r--  1 root root 40497 2012-08-06 04:27 ChangeLog
-rw-r--r--  1 root root  1823 2012-08-06 04:27 common.mak
drwxr-xr-x  6 root root  4096 2011-06-04 09:27 contrib
-rwxr-xr-x  1 root root   442 2012-08-06 04:27 evalrev
-rw-r--r--  1 root root  2335 2012-08-06 04:27 INSTALLING
drwxr-xr-x  4 root root  4096 2012-02-11 07:11 lib
-rw-r--r--  1 root root 18011 2012-08-06 04:27 LICENSE
-rw-r--r--  1 root root  6279 2012-08-06 04:27 LICENSE.OpenSSL
-rw-r--r--  1 root root  1083 2012-08-06 04:27 Makefile
drwxr-xr-x  3 root root  4096 2012-08-08 10:58 manpages
drwxr-xr-x  3 root root  4096 2012-08-08 10:58 packages
-rw-r--r--  1 root root  3147 2012-08-06 04:27 patchchk
drwxr-xr-x  4 root root  4096 2012-08-08 10:58 patches
-rw-r--r--  1 root root  5383 2012-08-06 04:27 README
drwxr-xr-x  7 root root  4096 2012-08-08 10:58 scripts
drwxr-xr-x  5 root root  4096 2012-08-08 10:58 src
drwxr-xr-x  6 root root  4096 2012-08-08 10:58 .svn
drwxr-xr-x  4 root root  4096 2012-08-08 10:58 test
-rw-r--r--  1 root root     3 2012-08-06 04:27 VERSION
root@bt:/pentest/wireless/aircrack-ng#
```

We'll examine more closely the security permissions in a later tutorial, but you must know that you need execute **(x)** permission on any file you want to execute. So, if you download a new tool, you must make certain that you have execute permission on it.

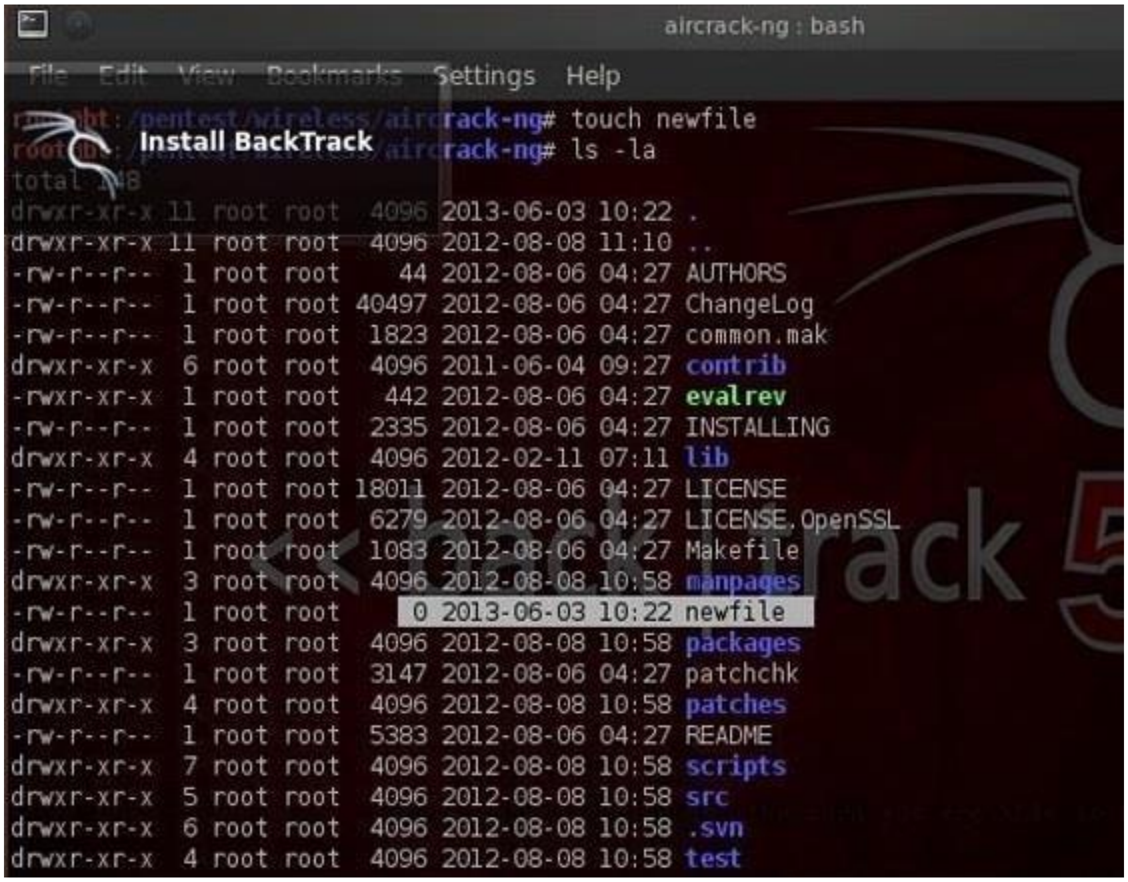
Step 3 Create a File (Touch)

To create a file in Linux, it's a bit different from Windows. In Linux, we use the **touch** command. So, let's create a new file called **newfile**:

- **bt > touch newfile**

Now we can check to see if that file exists by doing a directory listing:

- **bt > ls -la**



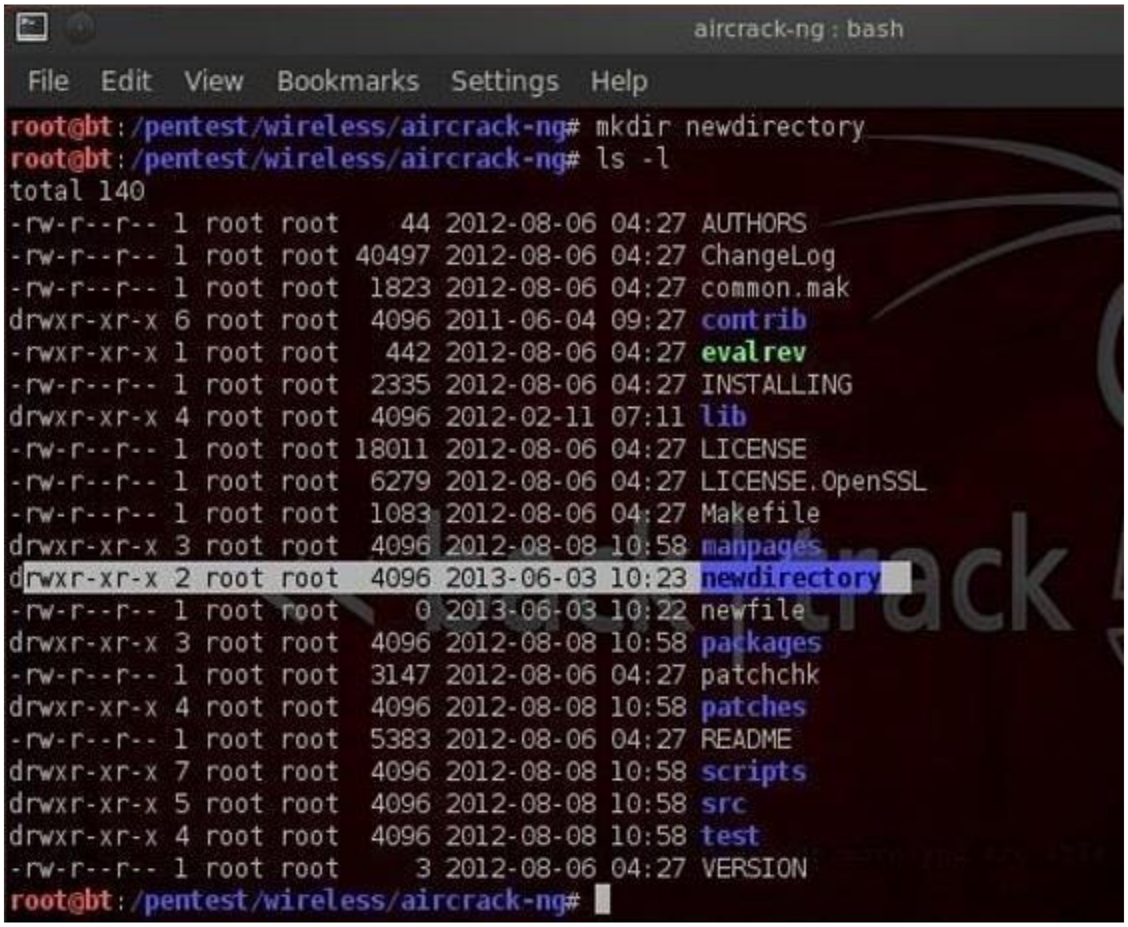
```
root@bt:/pentest/wireless/aircrack-ng# touch newfile
root@bt:/pentest/wireless/aircrack-ng# ls -la
total 148
drwxr-xr-x 11 root root  4096 2013-06-03 10:22 .
drwxr-xr-x 11 root root  4096 2012-08-08 11:10 ..
-rw-r--r--  1 root root    44 2012-08-06 04:27 AUTHORS
-rw-r--r--  1 root root 40497 2012-08-06 04:27 ChangeLog
-rw-r--r--  1 root root  1823 2012-08-06 04:27 common.mak
drwxr-xr-x  6 root root  4096 2011-06-04 09:27 contrib
-rwxr-xr-x  1 root root   442 2012-08-06 04:27 evalrev
-rw-r--r--  1 root root  2335 2012-08-06 04:27 INSTALLING
drwxr-xr-x  4 root root  4096 2012-02-11 07:11 lib
-rw-r--r--  1 root root 18011 2012-08-06 04:27 LICENSE
-rw-r--r--  1 root root  6279 2012-08-06 04:27 LICENSE.OpenSSL
-rw-r--r--  1 root root  1083 2012-08-06 04:27 Makefile
drwxr-xr-x  3 root root  4096 2012-08-08 10:58 manpages
-rw-r--r--  1 root root     0 2013-06-03 10:22 newfile
drwxr-xr-x  3 root root  4096 2012-08-08 10:58 packages
-rw-r--r--  1 root root  3147 2012-08-06 04:27 patchchk
drwxr-xr-x  4 root root  4096 2012-08-08 10:58 patches
-rw-r--r--  1 root root  5383 2012-08-06 04:27 README
drwxr-xr-x  7 root root  4096 2012-08-08 10:58 scripts
drwxr-xr-x  5 root root  4096 2012-08-08 10:58 src
drwxr-xr-x  6 root root  4096 2012-08-08 10:58 .svn
drwxr-xr-x  4 root root  4096 2012-08-08 10:58 test
```

We can see that new file has been created!

Step 4 Create a Directory (Mkdir)

Similar to Windows, we can create a directory by using the **make directory command (mkdir)**. Let's now make a new directory.

- **bt > mkdir newdirectory**



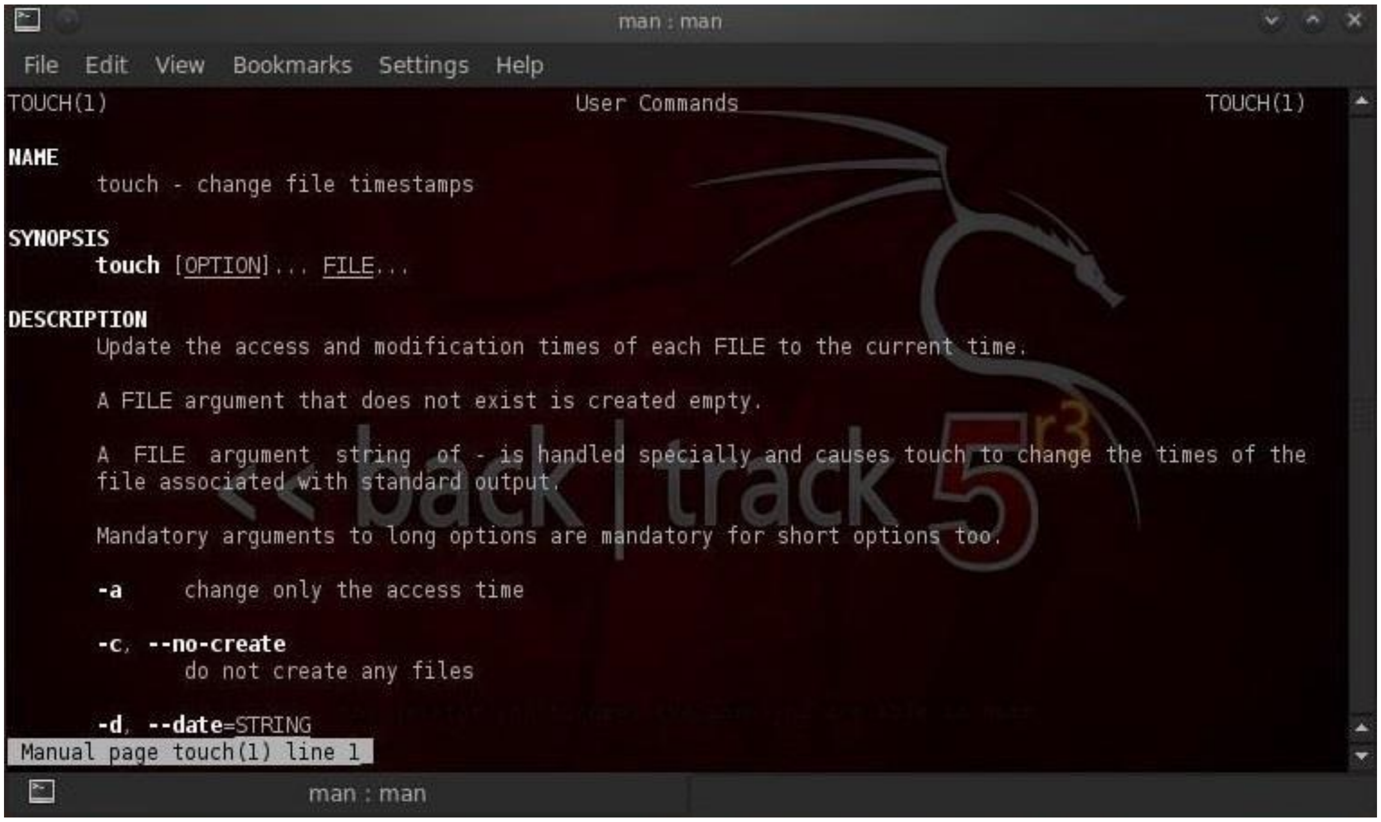
```
root@bt:/pentest/wireless/aircrack-ng# mkdir newdirectory
root@bt:/pentest/wireless/aircrack-ng# ls -l
total 140
-rw-r--r-- 1 root root    44 2012-08-06 04:27 AUTHORS
-rw-r--r-- 1 root root 40497 2012-08-06 04:27 ChangeLog
-rw-r--r-- 1 root root  1823 2012-08-06 04:27 common.mak
drwxr-xr-x 6 root root  4096 2011-06-04 09:27 contrib
-rwxr-xr-x 1 root root   442 2012-08-06 04:27 evalrev
-rw-r--r-- 1 root root  2335 2012-08-06 04:27 INSTALLING
drwxr-xr-x 4 root root  4096 2012-02-11 07:11 lib
-rw-r--r-- 1 root root 18011 2012-08-06 04:27 LICENSE
-rw-r--r-- 1 root root  6279 2012-08-06 04:27 LICENSE.OpenSSL
-rw-r--r-- 1 root root  1083 2012-08-06 04:27 Makefile
drwxr-xr-x 3 root root  4096 2012-08-08 10:58 manpages
drwxr-xr-x 2 root root  4096 2013-06-03 10:23 newdirectory
-rw-r--r-- 1 root root     0 2013-06-03 10:22 newfile
drwxr-xr-x 3 root root  4096 2012-08-08 10:58 packages
-rw-r--r-- 1 root root  3147 2012-08-06 04:27 patchchk
drwxr-xr-x 4 root root  4096 2012-08-08 10:58 patches
-rw-r--r-- 1 root root  5383 2012-08-06 04:27 README
drwxr-xr-x 7 root root  4096 2012-08-08 10:58 scripts
drwxr-xr-x 5 root root  4096 2012-08-08 10:58 src
drwxr-xr-x 4 root root  4096 2012-08-08 10:58 test
-rw-r--r-- 1 root root     3 2012-08-06 04:27 VERSION
root@bt:/pentest/wireless/aircrack-ng# 
```

Now type **ls** and we can see that a new directory has been created.

Step 5 Getting Help (Man)

Linux has a very useful utility called **man**. **Man** is the manual for nearly every command. If you should forget what a command does, simply type **man** and the name of the command and it will display the manual with all the info you need about that command, its switches, and arguments. For instance, type:

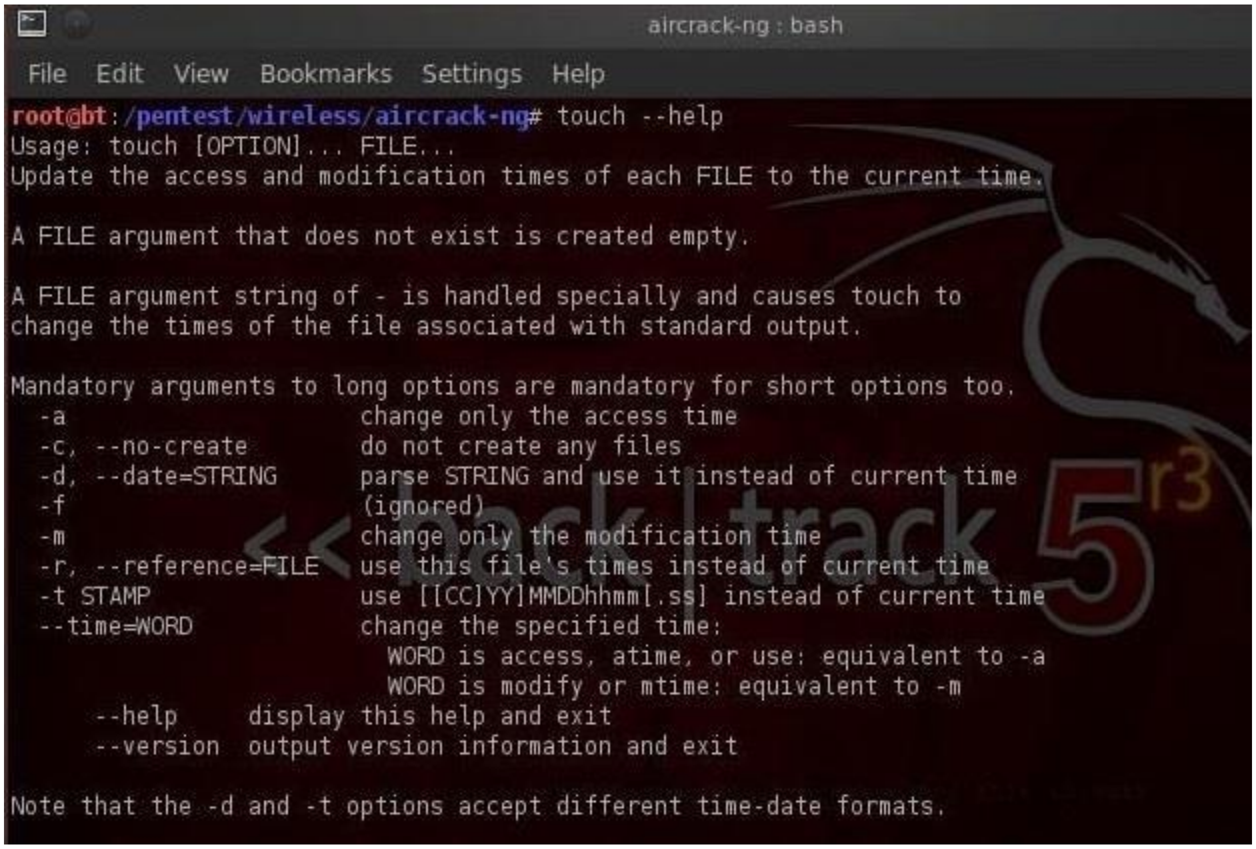
- **bt > man touch**



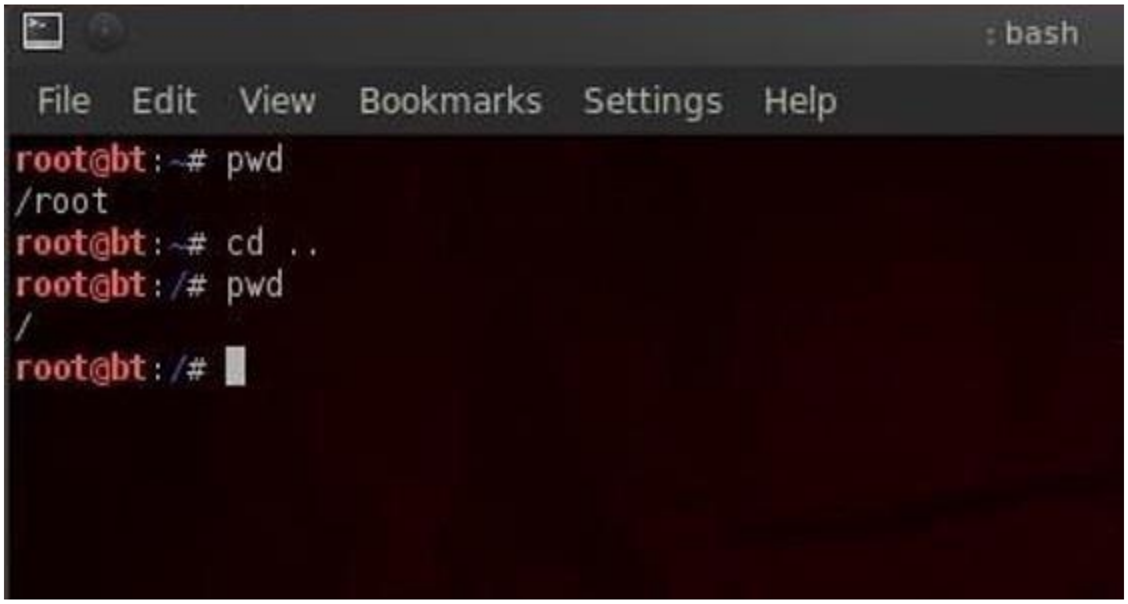
```
TOUCH(1)                          User Commands                         TOUCH(1)

NAME
       touch - change file timestamps

SYNOPSIS
       touch [OPTION]... FILE...

DESCRIPTION
       Update the access and modification times of each FILE to the current time.

       A FILE argument that does not exist is created empty.

       A  FILE  argument  string  of - is handled specially and causes touch to change the times of the
       file associated with standard output.

       Mandatory arguments to long options are mandatory for short options too.

       -a     change only the access time

       -c, --no-create
              do not create any files

       -d, --date=STRING
 Manual page touch(1) line 1
```

With most commands, you can also use either the **-h** switch or the **--help** switch after the command to obtain "help" about a particular command. In the case of "**touch**", we must use the **--help** to obtain help on the **touch** command.

- **bt > touch --help**



```
root@bt:/pentest/wireless/aircrack-ng# touch --help
Usage: touch [OPTION]... FILE...
Update the access and modification times of each FILE to the current time.

A FILE argument that does not exist is created empty.

A FILE argument string of - is handled specially and causes touch to
change the times of the file associated with standard output.

Mandatory arguments to long options are mandatory for short options too.
  -a                     change only the access time
  -c, --no-create        do not create any files
  -d, --date=STRING      parse STRING and use it instead of current time
  -f                     (ignored)
  -m                     change only the modification time
  -r, --reference=FILE   use this file's times instead of current time
  -t STAMP               use [[CC]YY]MMDDhhmm[.ss] instead of current time
  --time=WORD            change the specified time:
                           WORD is access, atime, or use: equivalent to -a
                           WORD is modify or mtime: equivalent to -m
      --help     display this help and exit
      --version  output version information and exit

Note that the -d and -t options accept different time-date formats.
```

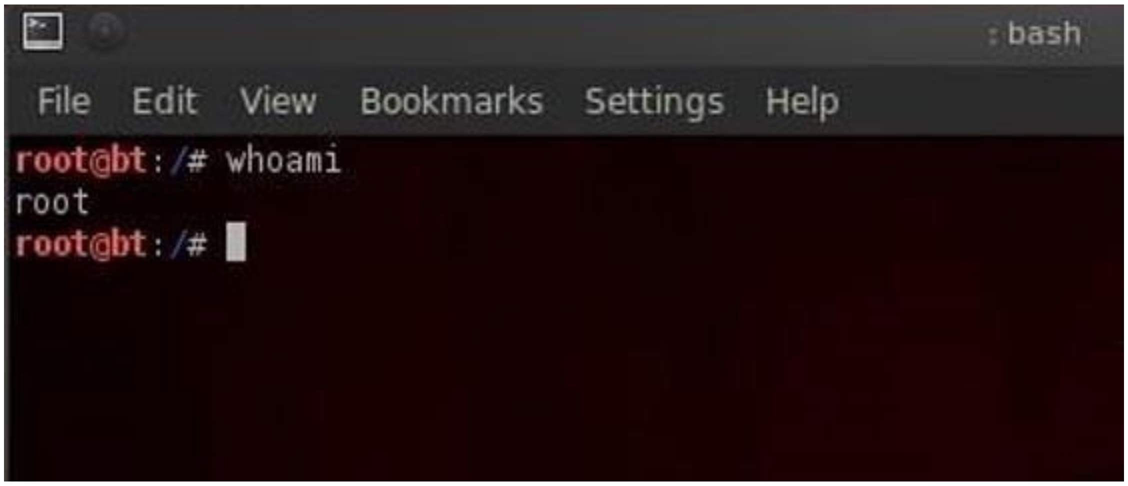


```
root@bt:~# pwd
/root
root@bt:~# cd ..
root@bt:/# pwd
/
root@bt:/# 
```

Step 6 Using the Whoami Command

In our last lesson of this tutorial, we'll use the **whoami** command. This command will return the name of the user we're logged in as. Since we're the root user, we can log in to any user account and that user's name would be displayed here.

- **bt > whoami**

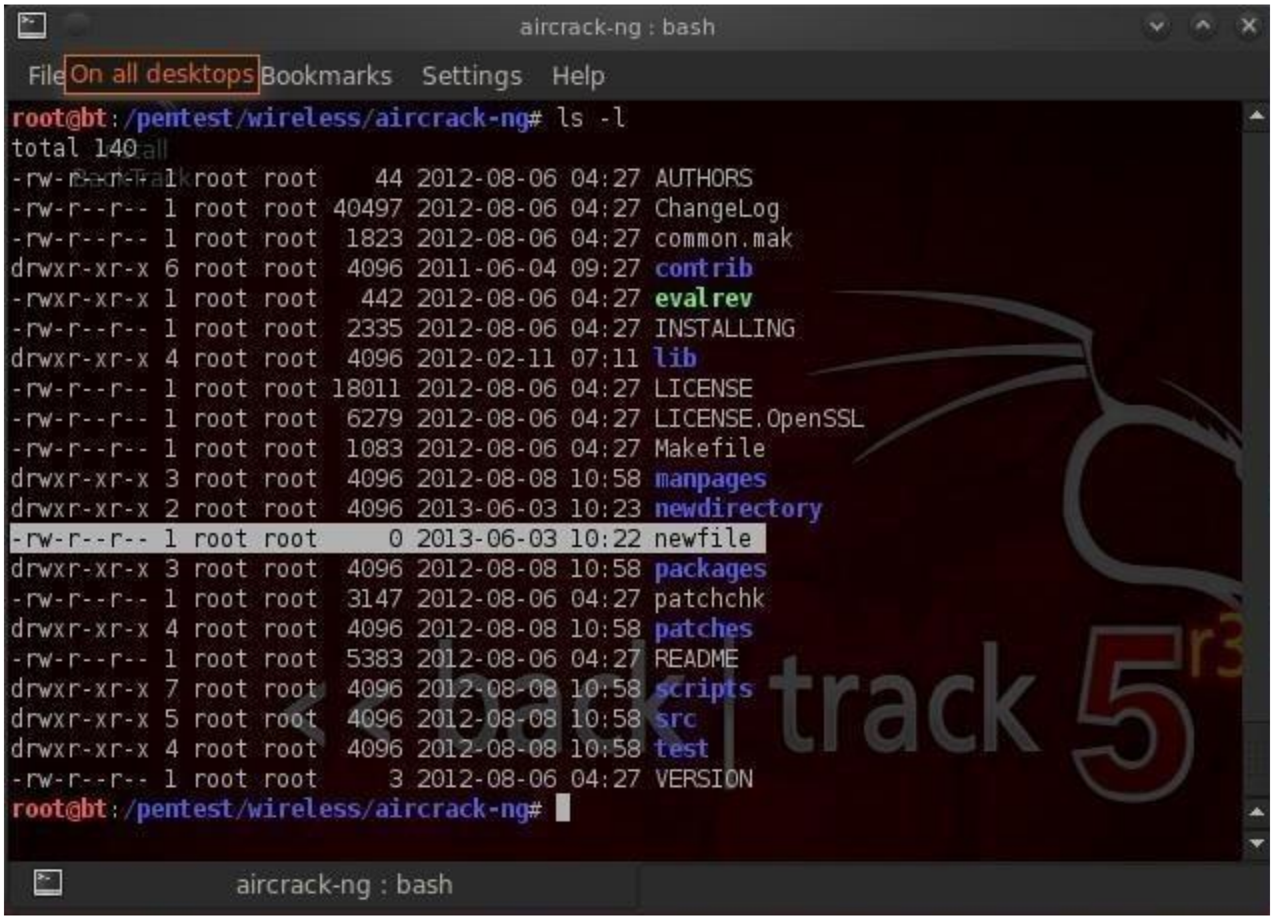


```
root@bt:/# whoami
root
root@bt:/#
```

Managing Directories & Files

In this installment, we'll look at how to manage files and directories in Linux, namely copying, renaming, moving, and viewing. Then we'll look a bit at networking and the ifconfig command.

Step 1 Copying Files (Cp)

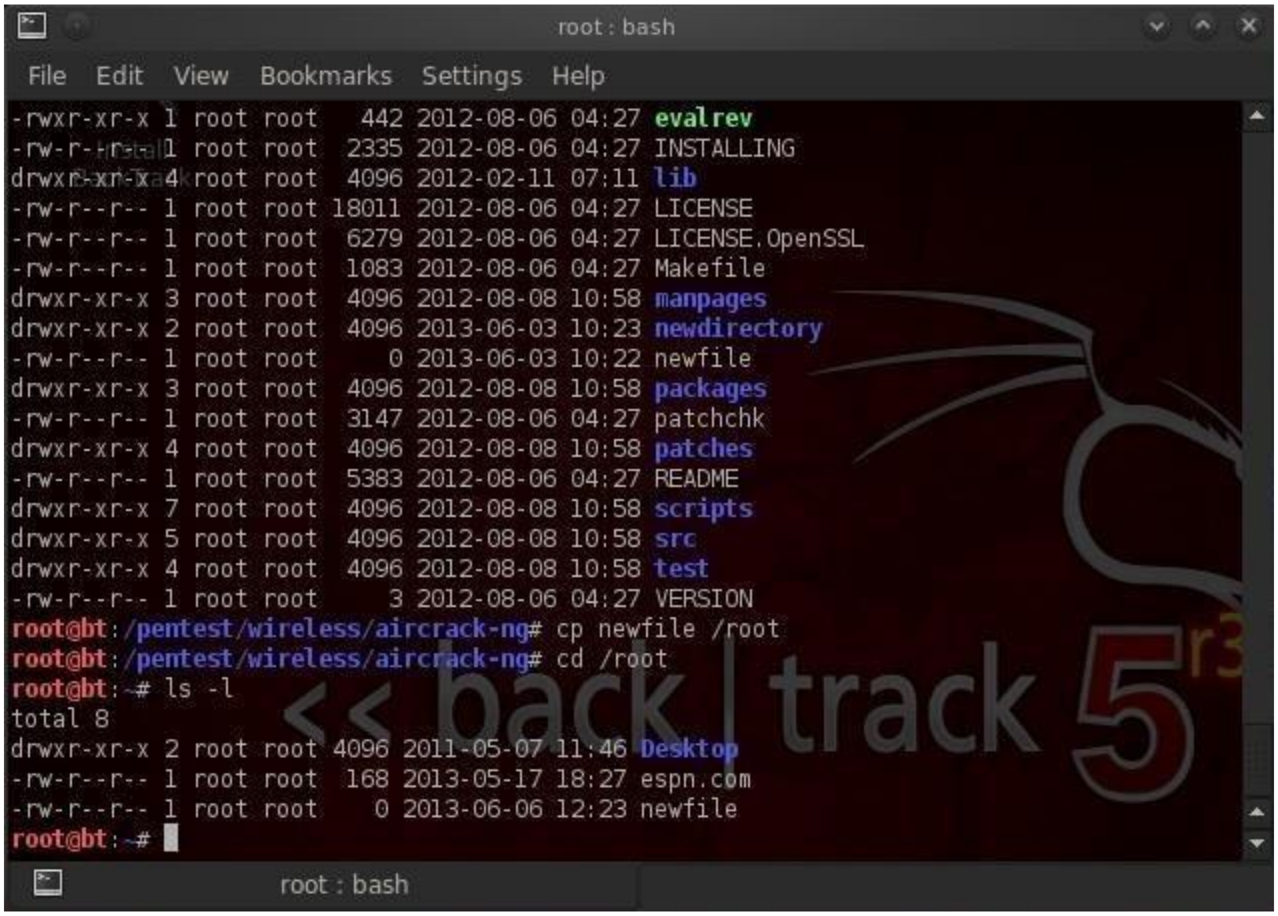


```
root@bt:/pentest/wireless/aircrack-ng# ls -l
total 140
-rw-r--r-- 1 root root    44 2012-08-06 04:27 AUTHORS
-rw-r--r-- 1 root root 40497 2012-08-06 04:27 ChangeLog
-rw-r--r-- 1 root root  1823 2012-08-06 04:27 common.mak
drwxr-xr-x 6 root root  4096 2011-06-04 09:27 contrib
-rwxr-xr-x 1 root root   442 2012-08-06 04:27 evalrev
-rw-r--r-- 1 root root  2335 2012-08-06 04:27 INSTALLING
drwxr-xr-x 4 root root  4096 2012-02-11 07:11 lib
-rw-r--r-- 1 root root 18011 2012-08-06 04:27 LICENSE
-rw-r--r-- 1 root root  6279 2012-08-06 04:27 LICENSE.OpenSSL
-rw-r--r-- 1 root root  1083 2012-08-06 04:27 Makefile
drwxr-xr-x 3 root root  4096 2012-08-08 10:58 manpages
drwxr-xr-x 2 root root  4096 2013-06-03 10:23 newdirectory
-rw-r--r-- 1 root root     0 2013-06-03 10:22 newfile
drwxr-xr-x 3 root root  4096 2012-08-08 10:58 packages
-rw-r--r-- 1 root root  3147 2012-08-06 04:27 patchchk
drwxr-xr-x 4 root root  4096 2012-08-08 10:58 patches
-rw-r--r-- 1 root root  5383 2012-08-06 04:27 README
drwxr-xr-x 7 root root  4096 2012-08-08 10:58 scripts
drwxr-xr-x 5 root root  4096 2012-08-08 10:58 src
drwxr-xr-x 4 root root  4096 2012-08-08 10:58 test
-rw-r--r-- 1 root root     3 2012-08-06 04:27 VERSION
root@bt:/pentest/wireless/aircrack-ng# 
```

Let's imagine that we need a copy of the file in our home directory, **user root**. We can do that by:

- **bt > cp newfile /root**

We simply tell Linux copy (**cp**) the **newfile** (in our current directory) to the directory of the **root user** (once again, don't confuse this with the **/** directory). We don't need to specify the directory that newfile is in, if it's in our current working directory. The copy command makes a copy of the file specified and places it in the specified directory leaving the original untouched and unchanged, so we now have two copies of the original file.



```
-rwxr-xr-x 1 root root   442 2012-08-06 04:27 evalrev
-rw-r--r-- 1 root root  2335 2012-08-06 04:27 INSTALLING
drwxr-xr-x 4 root root  4096 2012-02-11 07:11 lib
-rw-r--r-- 1 root root 18011 2012-08-06 04:27 LICENSE
-rw-r--r-- 1 root root  6279 2012-08-06 04:27 LICENSE.OpenSSL
-rw-r--r-- 1 root root  1083 2012-08-06 04:27 Makefile
drwxr-xr-x 3 root root  4096 2012-08-08 10:58 manpages
drwxr-xr-x 2 root root  4096 2013-06-03 10:23 newdirectory
-rw-r--r-- 1 root root     0 2013-06-03 10:22 newfile
drwxr-xr-x 3 root root  4096 2012-08-08 10:58 packages
-rw-r--r-- 1 root root  3147 2012-08-06 04:27 patchchk
drwxr-xr-x 4 root root  4096 2012-08-08 10:58 patches
-rw-r--r-- 1 root root  5383 2012-08-06 04:27 README
drwxr-xr-x 7 root root  4096 2012-08-08 10:58 scripts
drwxr-xr-x 5 root root  4096 2012-08-08 10:58 src
drwxr-xr-x 4 root root  4096 2012-08-08 10:58 test
-rw-r--r-- 1 root root     3 2012-08-06 04:27 VERSION
root@bt:/pentest/wireless/aircrack-ng# cp newfile /root
root@bt:/pentest/wireless/aircrack-ng# cd /root
root@bt:~# ls -l
total 8
drwxr-xr-x 2 root root 4096 2011-05-07 11:46 Desktop
-rw-r--r-- 1 root root  168 2013-05-17 18:27 espn.com
-rw-r--r-- 1 root root    0 2013-06-06 12:23 newfile
root@bt:~#
```

You can see in the screenshot above that when we change directory (**cd**) to the **root user** and list the files (**ls**) that now a **newfile copy** appears in that directory.

What if we wanted to copy a file from a directory that wasn't in our current working directory? In that case, we would need to specify a path to the directory, such as:

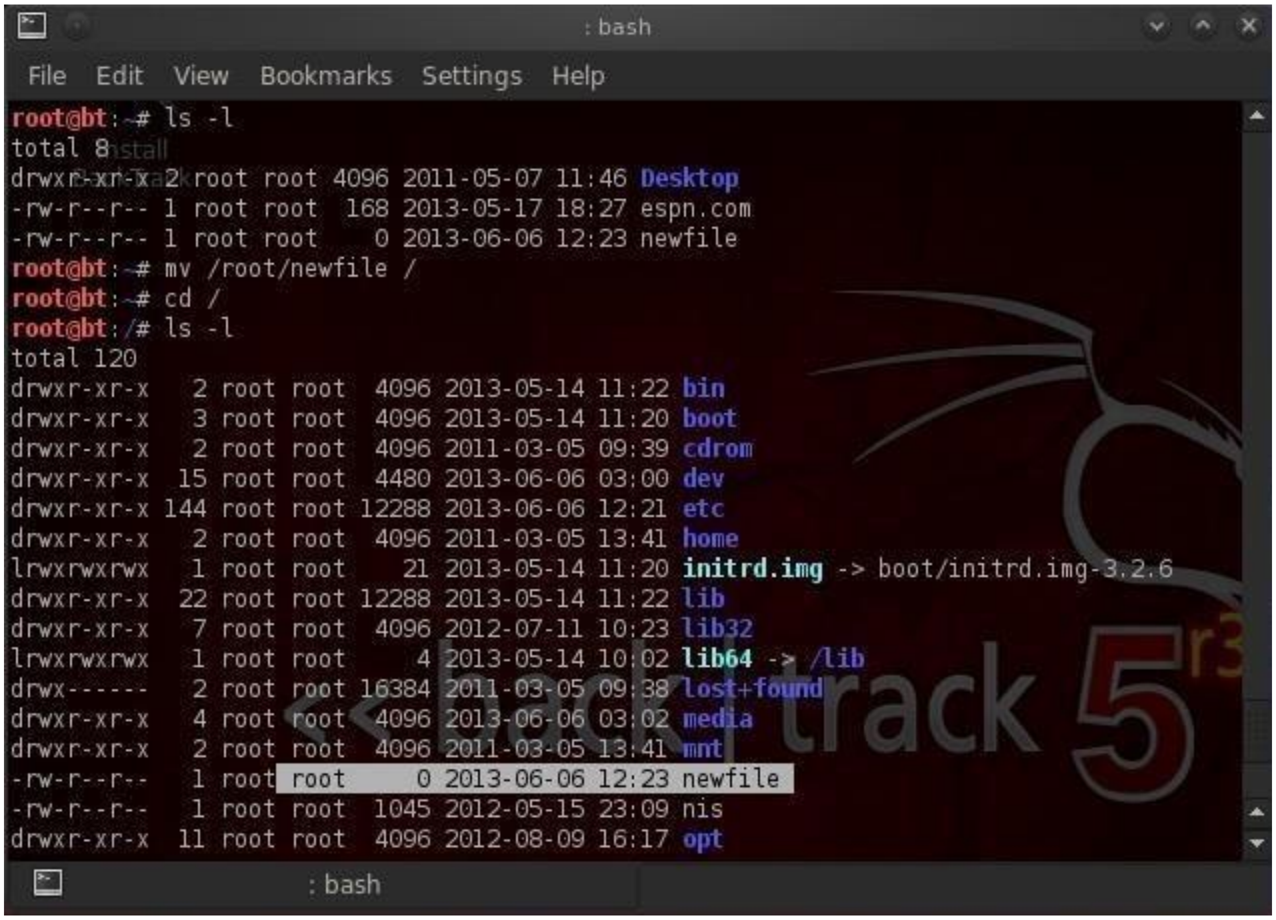
- **bt > cp /etc/newfile /root**

Also, note that we don't need to specify the file name we're copying it to. It simply makes a copy and gives it the same name as the original "newfile."

Step 2 Moving Files (Mv)

Unfortunately, Linux doesn't have a rename command for renaming files, so most users use the move (**mv**) command to both move files and rename them. Let's imagine now that we placed that **newfile** in the wrong directory and we really wanted it in the **root (/)** directory. We can use the move command to do so.

- **bt > mv /root/newfile /**

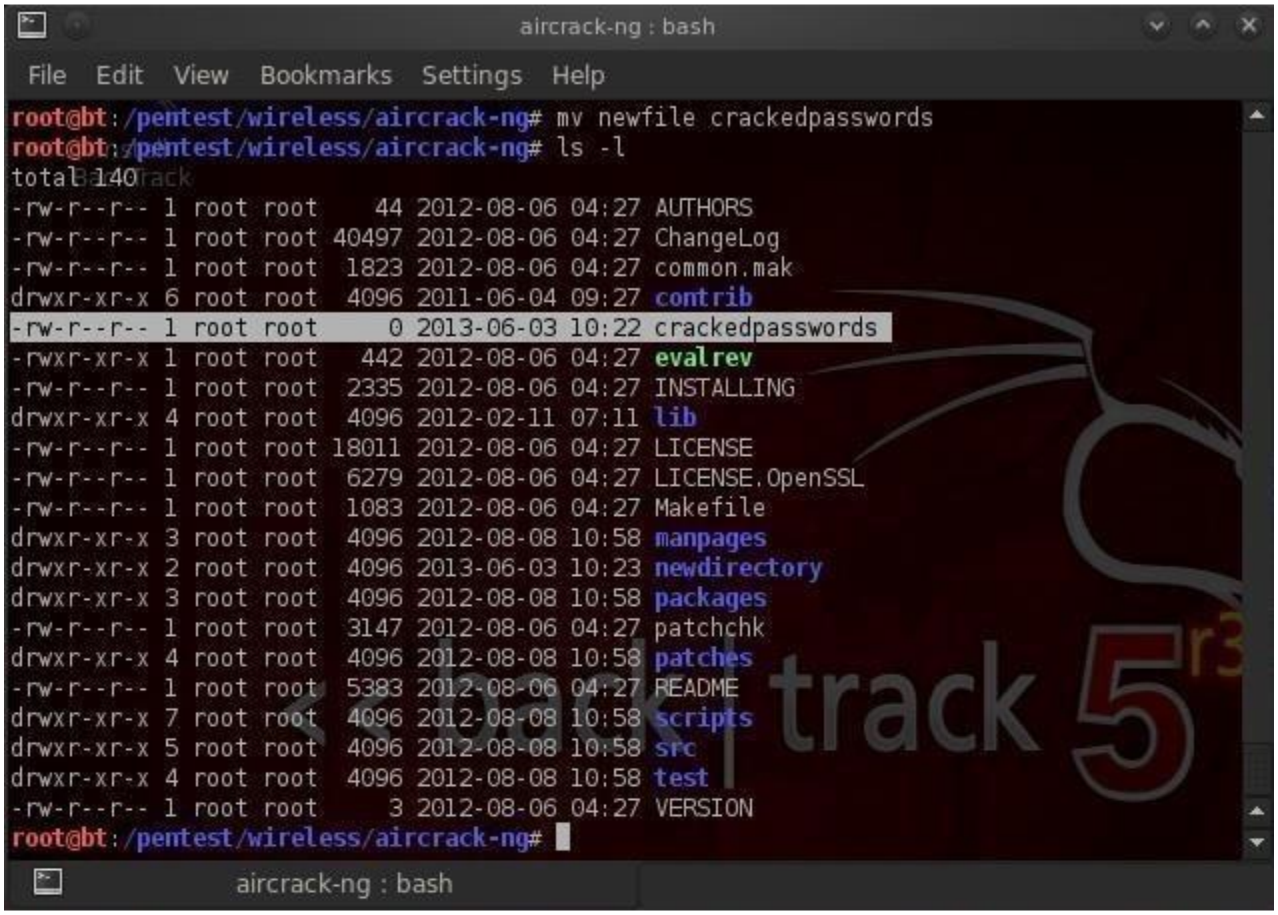


```
root@bt:~# ls -l
total 8
drwxr-xr-x  2 root root 4096 2011-05-07 11:46 Desktop
-rw-r--r--  1 root root  168 2013-05-17 18:27 espn.com
-rw-r--r--  1 root root    0 2013-06-06 12:23 newfile
root@bt:~# mv /root/newfile /
root@bt:~# cd /
root@bt:/# ls -l
total 120
drwxr-xr-x   2 root root  4096 2013-05-14 11:22 bin
drwxr-xr-x   3 root root  4096 2013-05-14 11:20 boot
drwxr-xr-x   2 root root  4096 2011-03-05 09:39 cdrom
drwxr-xr-x  15 root root  4480 2013-06-06 03:00 dev
drwxr-xr-x 144 root root 12288 2013-06-06 12:21 etc
drwxr-xr-x   2 root root  4096 2011-03-05 13:41 home
lrwxrwxrwx   1 root root    21 2013-05-14 11:20 initrd.img -> boot/initrd.img-3.2.6
drwxr-xr-x  22 root root 12288 2013-05-14 11:22 lib
drwxr-xr-x   7 root root  4096 2012-07-11 10:23 lib32
lrwxrwxrwx   1 root root     4 2013-05-14 10:02 lib64 -> /lib
drwx------   2 root root 16384 2011-03-05 09:38 lost+found
drwxr-xr-x   4 root root  4096 2013-06-06 03:02 media
drwxr-xr-x   2 root root  4096 2011-03-05 13:41 mnt
-rw-r--r--   1 root root     0 2013-06-06 12:23 newfile
-rw-r--r--   1 root root  1045 2012-05-15 23:09 nis
drwxr-xr-x  11 root root  4096 2012-08-09 16:17 opt
```

This command says, move the **newfile** from the **root user** directory to the **root (/)** directory. The move command literally moves the file and does not leave a copy where the old one existed. Note that the **newfile** has moved to the **root directory**.

Sometimes, we want change the name of the file and not actually move it to a different location. The move command can be used for that also. We simply tell Linux to move the original file to a new file with a new name. Take for instance our **newfile** in the **aircrack-ng directory**. Let's say that we want to rename that file to "crackedpasswords**. We can simply type:**

- bt > mv newfile crackedpasswords



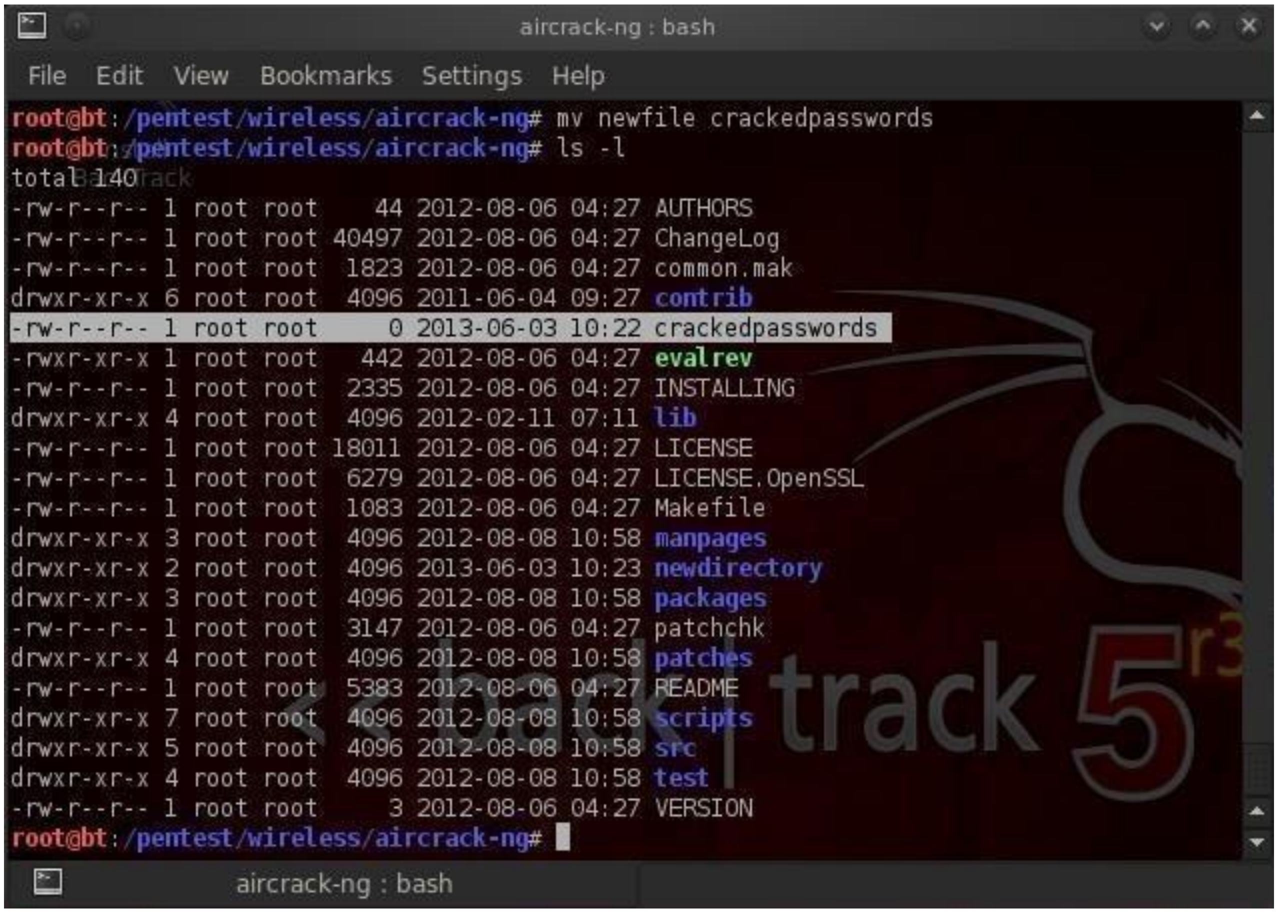
```
root@bt:/pentest/wireless/aircrack-ng# mv newfile crackedpasswords
root@bt:/pentest/wireless/aircrack-ng# ls -l
total 140
-rw-r--r-- 1 root root    44 2012-08-06 04:27 AUTHORS
-rw-r--r-- 1 root root 40497 2012-08-06 04:27 ChangeLog
-rw-r--r-- 1 root root  1823 2012-08-06 04:27 common.mak
drwxr-xr-x 6 root root  4096 2011-06-04 09:27 contrib
-rw-r--r-- 1 root root     0 2013-06-03 10:22 crackedpasswords
-rwxr-xr-x 1 root root   442 2012-08-06 04:27 evalrev
-rw-r--r-- 1 root root  2335 2012-08-06 04:27 INSTALLING
drwxr-xr-x 4 root root  4096 2012-02-11 07:11 lib
-rw-r--r-- 1 root root 18011 2012-08-06 04:27 LICENSE
-rw-r--r-- 1 root root  6279 2012-08-06 04:27 LICENSE.OpenSSL
-rw-r--r-- 1 root root  1083 2012-08-06 04:27 Makefile
drwxr-xr-x 3 root root  4096 2012-08-08 10:58 manpages
drwxr-xr-x 2 root root  4096 2013-06-03 10:23 newdirectory
drwxr-xr-x 3 root root  4096 2012-08-08 10:58 packages
-rw-r--r-- 1 root root  3147 2012-08-06 04:27 patchchk
drwxr-xr-x 4 root root  4096 2012-08-08 10:58 patches
-rw-r--r-- 1 root root  5383 2012-08-06 04:27 README
drwxr-xr-x 7 root root  4096 2012-08-08 10:58 scripts
drwxr-xr-x 5 root root  4096 2012-08-08 10:58 src
drwxr-xr-x 4 root root  4096 2012-08-08 10:58 test
-rw-r--r-- 1 root root     3 2012-08-06 04:27 VERSION
root@bt:/pentest/wireless/aircrack-ng# 
```

Notice here that I did not use any directory paths because I was moving a file in my current working directory and to a file in my current working directory. If we run a directory listing now, we can see that **newfile** is gone and **crackedpasswords** now exists in the **aircrack-ng directory**.

Step 3 Viewing Files (Cat, More, Less)

From the command line in the terminal, we can view the contents of files by using the **cat** command. **cat** is short for concatenate, which is a $20 word for putting together a bunch of pieces (we are putting together the words for display on the screen). Concatenate is a fancy word, but is used throughout computer science and information technology, so add it to your vocabulary.

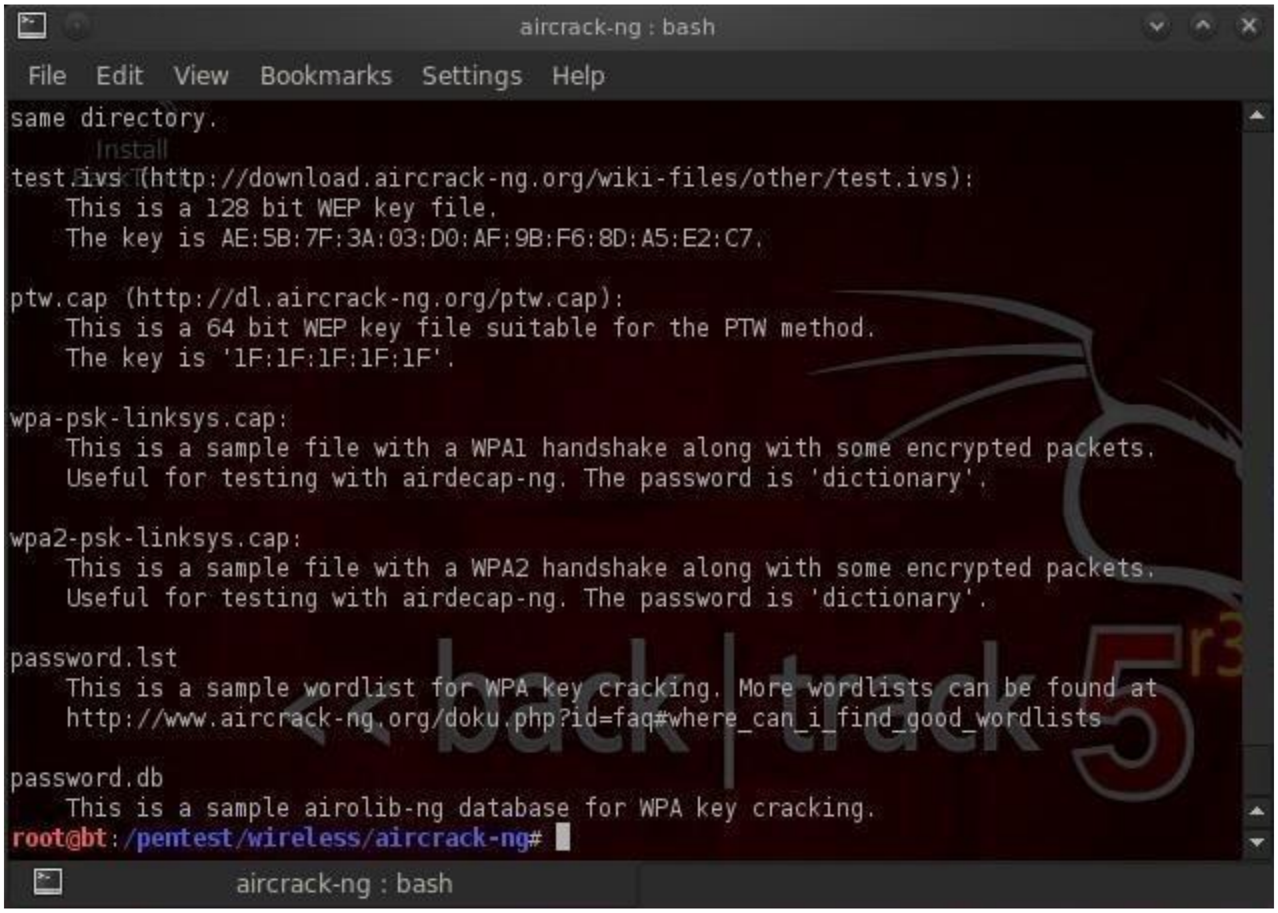
Staying in the **/pentest/wireless/aircrack-ng directory**, let's **cat** some files. First, let's get a listing of files in this directory.



```
root@bt:/pentest/wireless/aircrack-ng# mv newfile crackedpasswords
root@bt:/pentest/wireless/aircrack-ng# ls -l
total 140
-rw-r--r-- 1 root root    44 2012-08-06 04:27 AUTHORS
-rw-r--r-- 1 root root 40497 2012-08-06 04:27 ChangeLog
-rw-r--r-- 1 root root  1823 2012-08-06 04:27 common.mak
drwxr-xr-x 6 root root  4096 2011-06-04 09:27 contrib
-rw-r--r-- 1 root root     0 2013-06-03 10:22 crackedpasswords
-rwxr-xr-x 1 root root   442 2012-08-06 04:27 evalrev
-rw-r--r-- 1 root root  2335 2012-08-06 04:27 INSTALLING
drwxr-xr-x 4 root root  4096 2012-02-11 07:11 lib
-rw-r--r-- 1 root root 18011 2012-08-06 04:27 LICENSE
-rw-r--r-- 1 root root  6279 2012-08-06 04:27 LICENSE.OpenSSL
-rw-r--r-- 1 root root  1083 2012-08-06 04:27 Makefile
drwxr-xr-x 3 root root  4096 2012-08-08 10:58 manpages
drwxr-xr-x 2 root root  4096 2013-06-03 10:23 newdirectory
drwxr-xr-x 3 root root  4096 2012-08-08 10:58 packages
-rw-r--r-- 1 root root  3147 2012-08-06 04:27 patchchk
drwxr-xr-x 4 root root  4096 2012-08-08 10:58 patches
-rw-r--r-- 1 root root  5383 2012-08-06 04:27 README
drwxr-xr-x 7 root root  4096 2012-08-08 10:58 scripts
drwxr-xr-x 5 root root  4096 2012-08-08 10:58 src
drwxr-xr-x 4 root root  4096 2012-08-08 10:58 test
-rw-r--r-- 1 root root     3 2012-08-06 04:27 VERSION
root@bt:/pentest/wireless/aircrack-ng#
```

Notice in the screenshot above, there is a file called **README**. Often, software developers use this file to provide important notes to their users. This file can be critical, especially with hacking tools because most are open source and seldom have manuals. Let's take a look at the contents of this file.

- **bt > cat README**

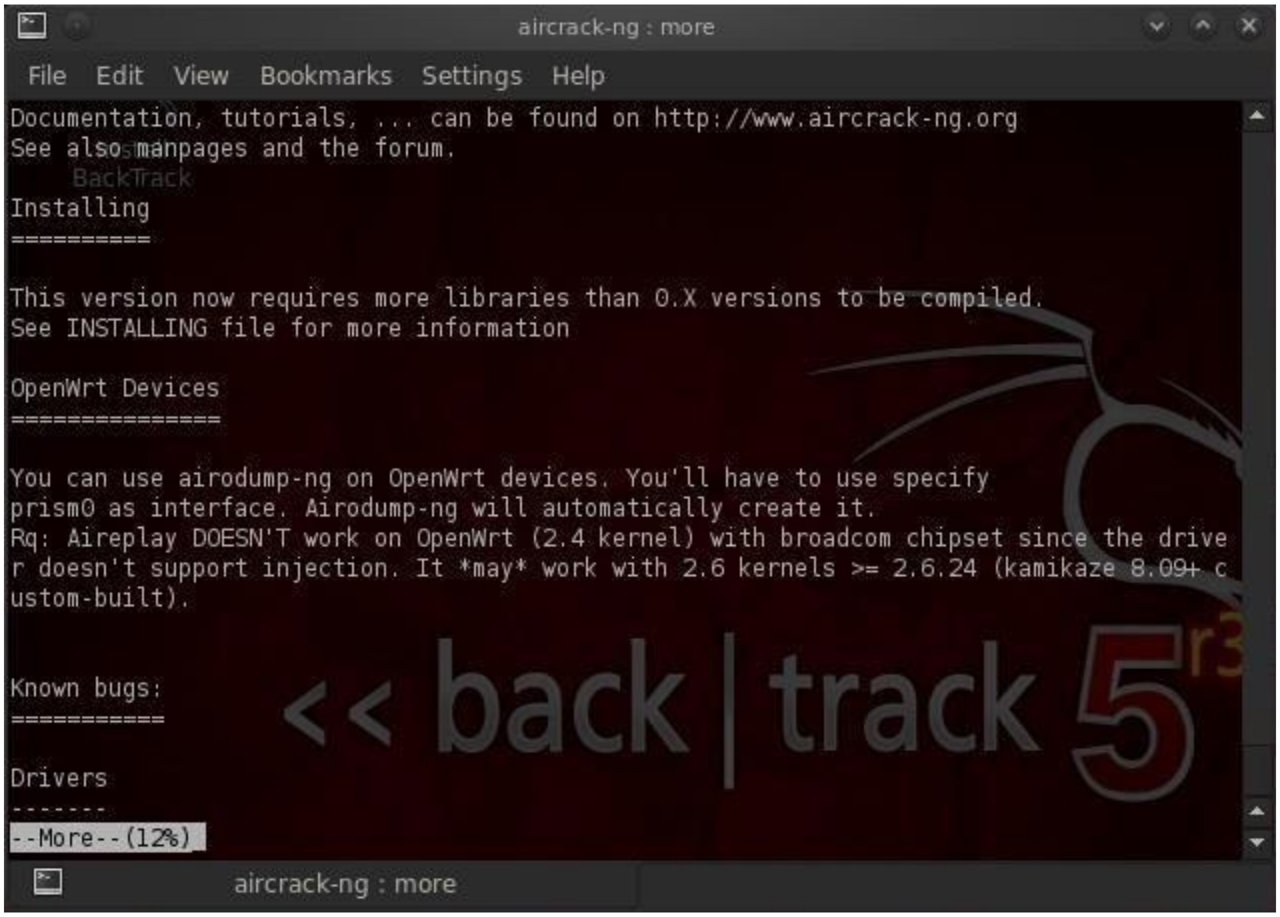


```
same directory.

test.ivs (http://download.aircrack-ng.org/wiki-files/other/test.ivs):
    This is a 128 bit WEP key file.
    The key is AE:5B:7F:3A:03:D0:AF:9B:F6:8D:A5:E2:C7.

ptw.cap (http://dl.aircrack-ng.org/ptw.cap):
    This is a 64 bit WEP key file suitable for the PTW method.
    The key is '1F:1F:1F:1F:1F'.

wpa-psk-linksys.cap:
    This is a sample file with a WPA1 handshake along with some encrypted packets.
    Useful for testing with airdecap-ng. The password is 'dictionary'.

wpa2-psk-linksys.cap:
    This is a sample file with a WPA2 handshake along with some encrypted packets.
    Useful for testing with airdecap-ng. The password is 'dictionary'.

password.lst
    This is a sample wordlist for WPA key cracking. More wordlists can be found at
    http://www.aircrack-ng.org/doku.php?id=faq#where_can_i_find_good_wordlists

password.db
    This is a sample airolib-ng database for WPA key cracking.
root@bt:/pentest/wireless/aircrack-ng#
```

When you run this command, you'll see lots of text running across your screen. Obviously, it goes by way too fast to read, but when its done, we could use the scroll button on the terminal to scroll up to read all the text. There is another way, though, that might be easier.

There are two commands that work similar to **cat** but don't simply run the text across the screen until it hits the end of file. These are **more** and **less**. They are very similar, each only displaying one page of information on your screen until you prompt it to scroll down. Let's try **more** first.

- **bt > more README**

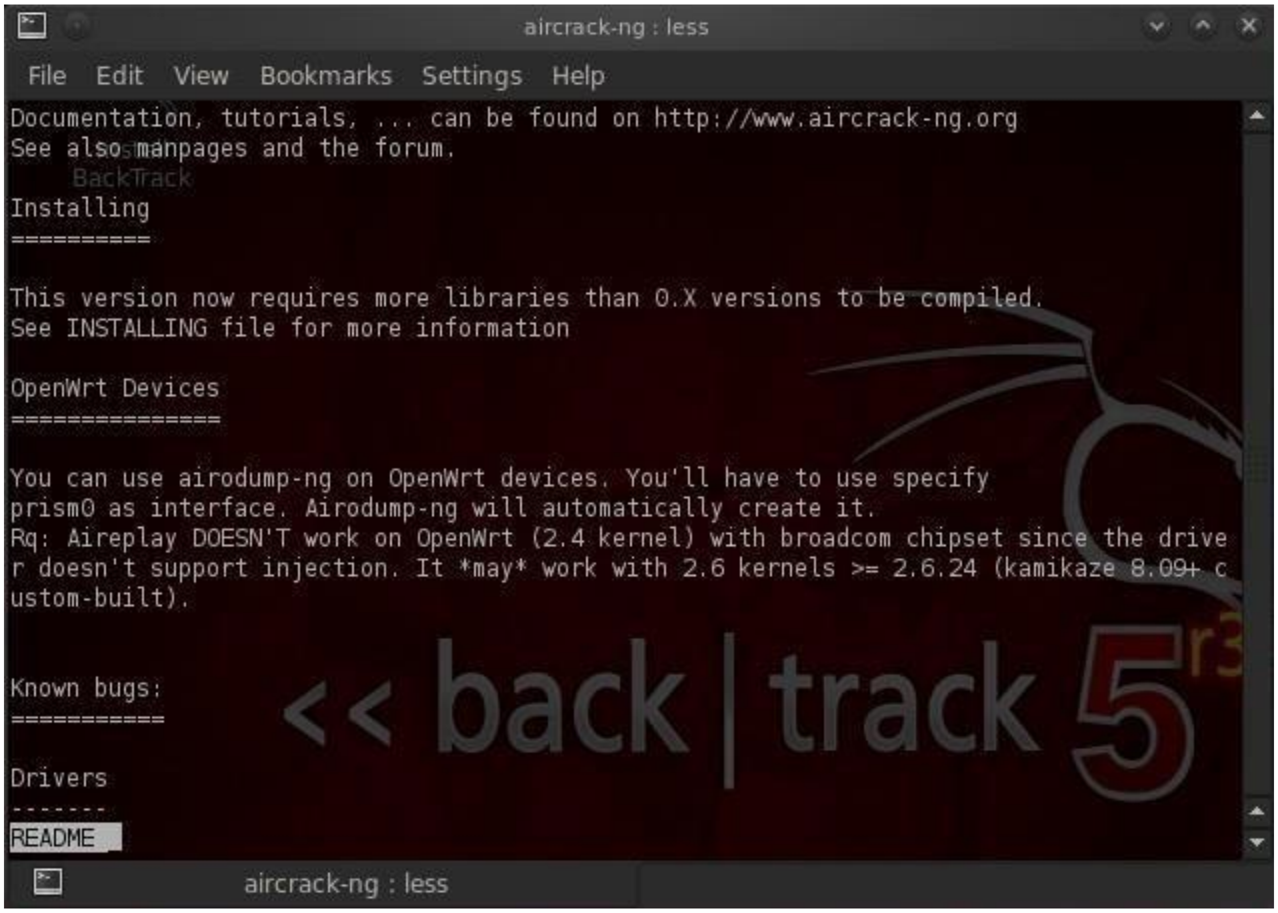


```
Documentation, tutorials, ... can be found on http://www.aircrack-ng.org
See also manpages and the forum.

Installing
==========

This version now requires more libraries than 0.X versions to be compiled.
See INSTALLING file for more information

OpenWrt Devices
===============

You can use airodump-ng on OpenWrt devices. You'll have to use specify
prism0 as interface. Airodump-ng will automatically create it.
Rq: Aireplay DOESN'T work on OpenWrt (2.4 kernel) with broadcom chipset since the drive
r doesn't support injection. It *may* work with 2.6 kernels >= 2.6.24 (kamikaze 8.09+ c
ustom-built).


Known bugs:
===========

Drivers
-------
--More--(12%)
```

As you can see, when I use **more** and the **filename**, it displays the file until the screen fills and waits for further instructions from me. If I hit **enter**, it will scroll down one line at a time, while if I hit the **spacebar**, it will scroll one page at a time.

Now let's try the more powerful **less** (in some Linux circles, there is a saying "less is more", meaning that less is more powerful than more).

- **bt > less README**



```
Documentation, tutorials, ... can be found on http://www.aircrack-ng.org
See also manpages and the forum.

Installing
==========

This version now requires more libraries than 0.X versions to be compiled.
See INSTALLING file for more information

OpenWrt Devices
===============

You can use airodump-ng on OpenWrt devices. You'll have to use specify
prism0 as interface. Airodump-ng will automatically create it.
Rq: Aireplay DOESN'T work on OpenWrt (2.4 kernel) with broadcom chipset since the drive
r doesn't support injection. It *may* work with 2.6 kernels >= 2.6.24 (kamikaze 8.09+ c
ustom-built).


Known bugs:
===========

Drivers
-------
README
```

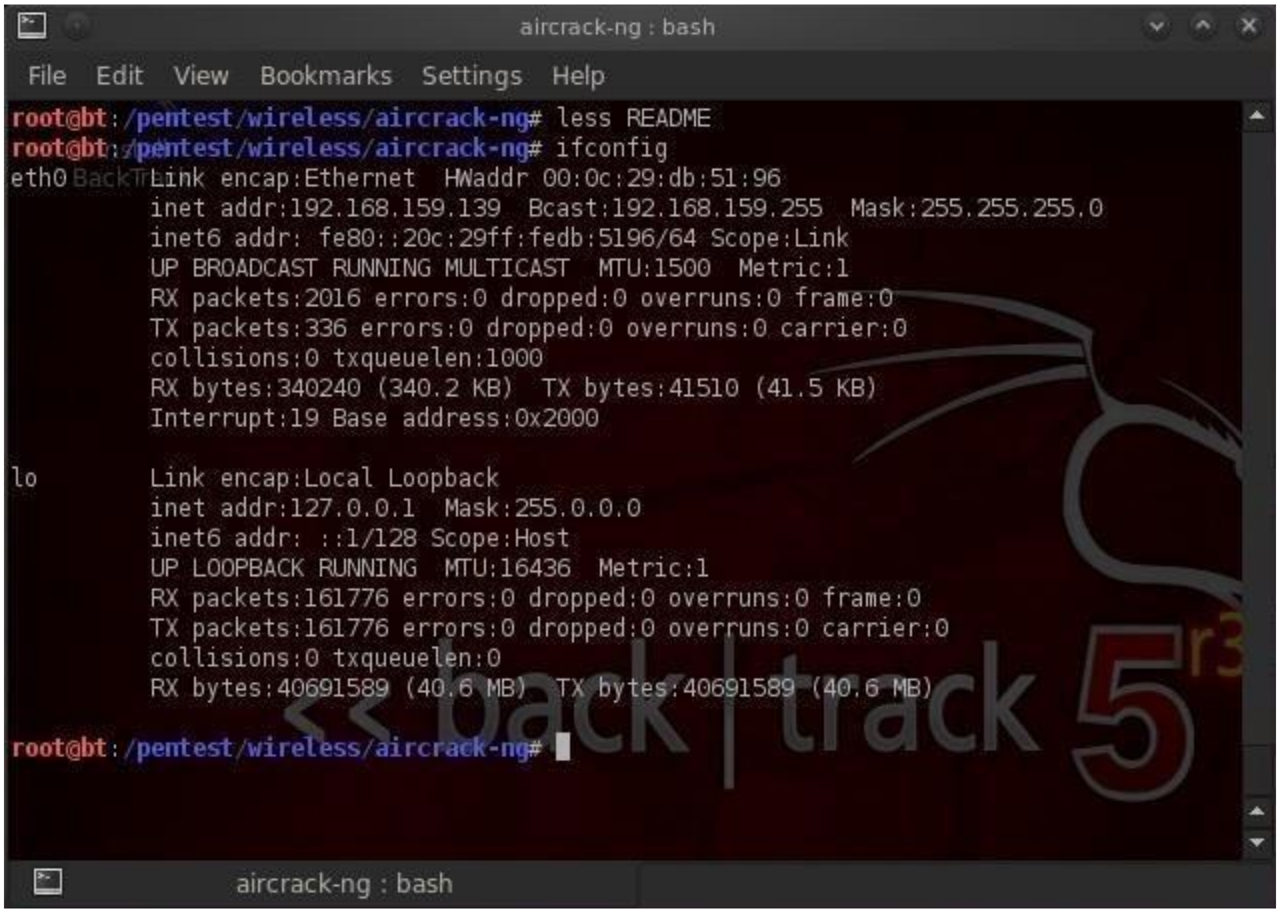
You can see that **less** followed by the **filename**, once again displays the **README** file until it fills up my terminal just like **more**. Though, note that **less** displays the name of the file that I'm viewing in the lower left-hand corner. Probably more importantly, **less** has powerful text searching capabilities that are missing from **more**. I can search for text within this file by typing the **forward slash** followed by what I'm searching for and **less** will find it and highlight it for me.

That's one of the primary reasons I prefer **less**.

Step 4 Networking (Ifconfig)

Before I finish this tutorial, I want to show you one last simple networking command, **ifconfig**. Those of you comfortable with Windows networking, know that you can use the **ipconfig** command in Windows to display key information on your networking configuration. **ifconfig** in Linux is very similar, with only one letter different. Let's run **ifconfig** see what it tells us.

- **bt >ifconfig**



```
root@bt:/pentest/wireless/aircrack-ng# less README
root@bt:/pentest/wireless/aircrack-ng# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:db:51:96
          inet addr:192.168.159.139  Bcast:192.168.159.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fedb:5196/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2016 errors:0 dropped:0 overruns:0 frame:0
          TX packets:336 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:340240 (340.2 KB)  TX bytes:41510 (41.5 KB)
          Interrupt:19 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:161776 errors:0 dropped:0 overruns:0 frame:0
          TX packets:161776 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:40691589 (40.6 MB)  TX bytes:40691589 (40.6 MB)

root@bt:/pentest/wireless/aircrack-ng# 
```

As you can see, it displays much of the key info I need to know about the network configuration of my system including IP address, netmask, broadcast address, interfaces, MAC address of my interface, etc.

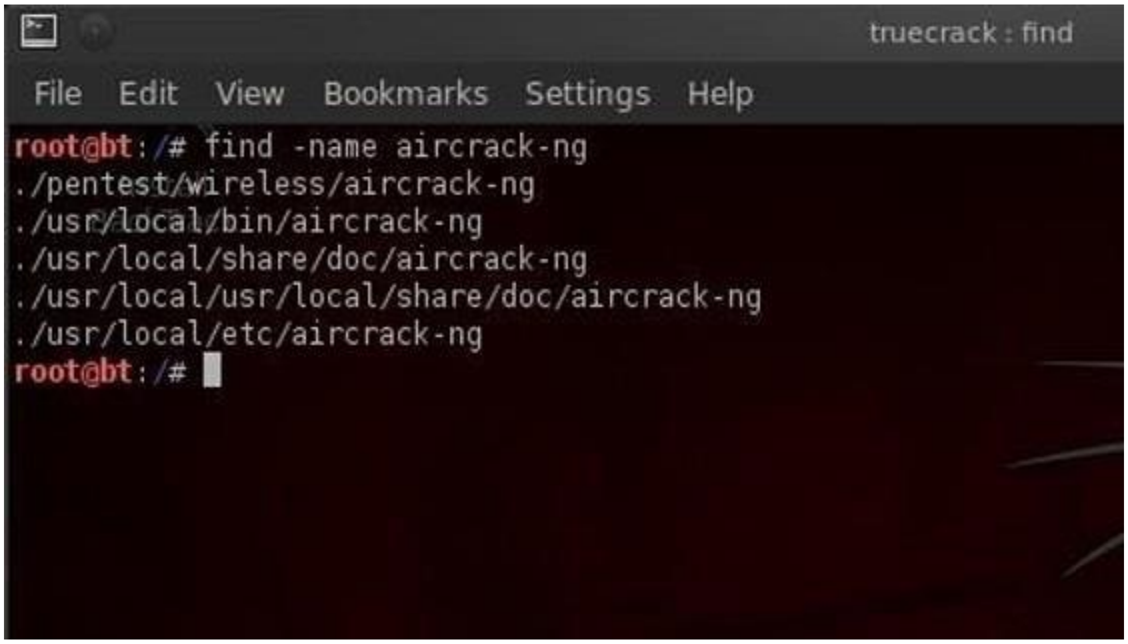
Finding Files

Linux beginners are often faced with the issue of how to find files and programs, especially considering the radically different directory structure as compared to Mac OS or Windows. Beginners sometimes get frustrated trying to find the necessary files or binaries, so I'm dedicating this tutorial to finding stuff in Linux.

Step 1 Finding Files in a Directory (Find)

The first command I want to show you is **find**. As you probably guessed, **find** is able to find stuff by looking in a directory for the file you're hunting for. By default, its recursive, which means it will look in all sub-directories and display a list of everywhere it finds the file. For instance, if we are looking for **aircrack-ng**, we could type:

- **bt > find -name aircarck-ng**

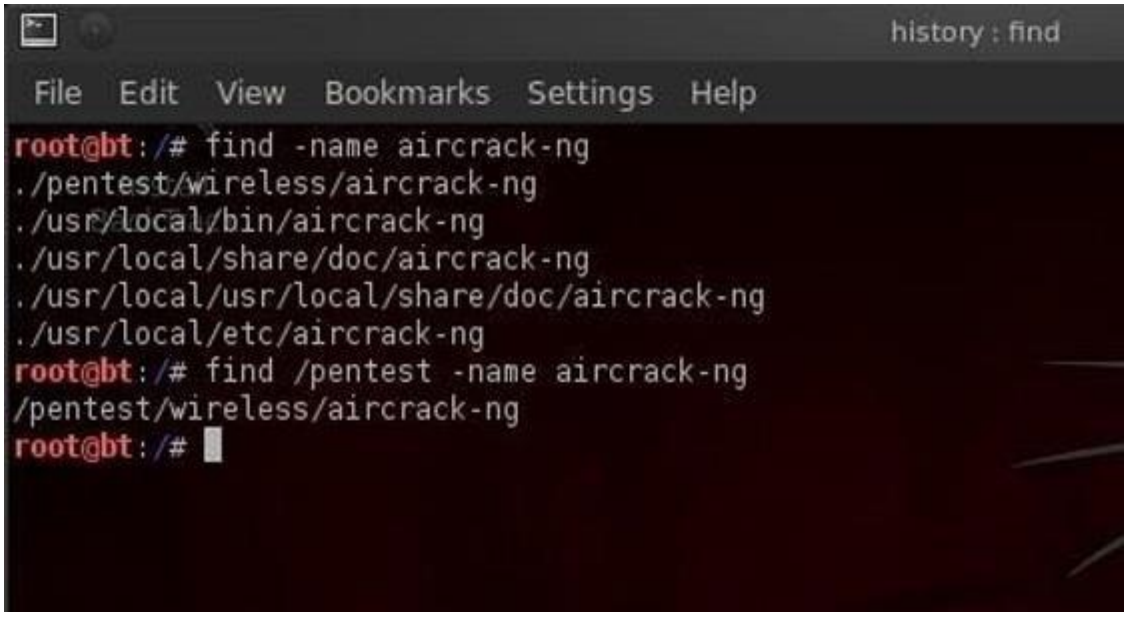


```
root@bt:/# find -name aircrack-ng
./pentest/wireless/aircrack-ng
./usr/local/bin/aircrack-ng
./usr/local/share/doc/aircrack-ng
./usr/local/usr/local/share/doc/aircrack-ng
./usr/local/etc/aircrack-ng
root@bt:/# 
```

Note that we need to tell Linux that we want to search by name (**-name**) and then the name of the file we're searching for.

It then returns the full path of every place where it finds **aircrack-ng**. We can be more specific and ask Linux to only tell us where it finds **aircrack-ng** in the **/pentest** directory. We can do this by typing:

- **bt > find /pentest -name aircrack-ng**



```
root@bt:/# find -name aircrack-ng
./pentest/wireless/aircrack-ng
./usr/local/bin/aircrack-ng
./usr/local/share/doc/aircrack-ng
./usr/local/usr/local/share/doc/aircrack-ng
./usr/local/etc/aircrack-ng
root@bt:/# find /pentest -name aircrack-ng
/pentest/wireless/aircrack-ng
root@bt:/#
```

This command says, "look in the pentest directory and all its sub-directories and tell me where you find something called aircrack-ng".

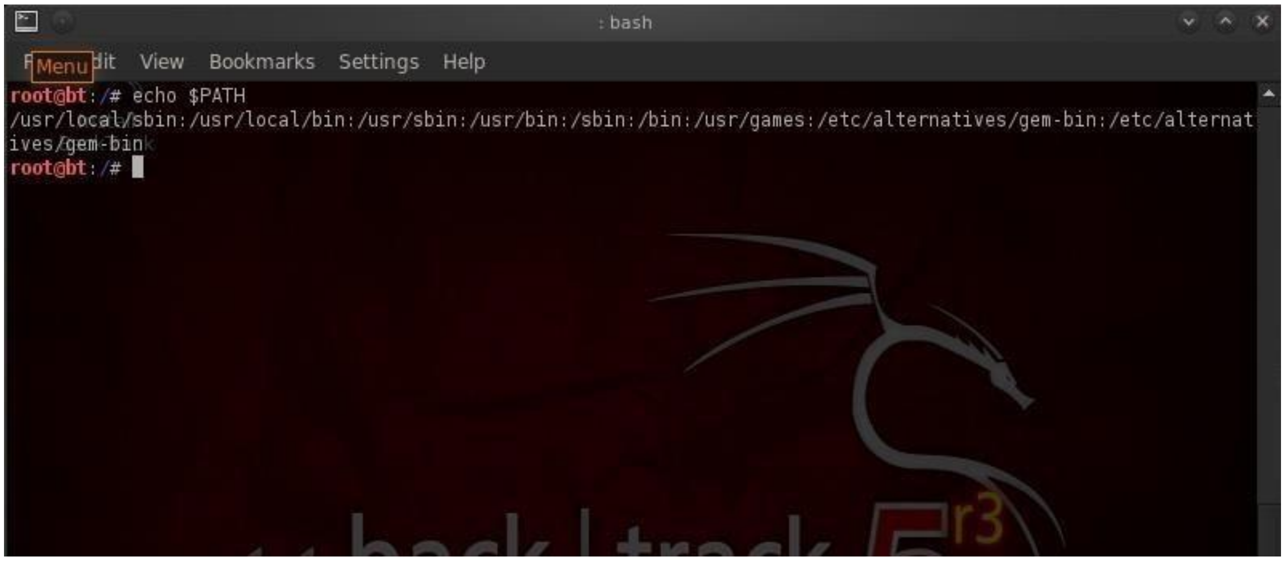
Now, Linux only returns those paths to files that are in the directory **/pentest** or its sub-directories, such as **/pentest/wireless/aircrack-ng** and the others.

Step 2 Finding Binaries in Path Variables (Which)

The next searching command we want to look at is **which**. This command allows us to search for binaries that are in our path variable. Hmm...even I think that's a lot of techo-googlygoop. Let's try to make some sense of it.

Binaries are the files that are the equivalent of executables in Windows. These are files that do something like **echo**, **ls**, **cd**, **mv**, etc. Our path variable is the variable that keeps the directory path to our binaries. Usually, our binaries are in the **/bin** (bin is short for binaries) or **/sbin** directory and that's reflected in our path variable. Our path variable setting can be checked by asking Linux to **echo** the value in the variable. We do this by typing:

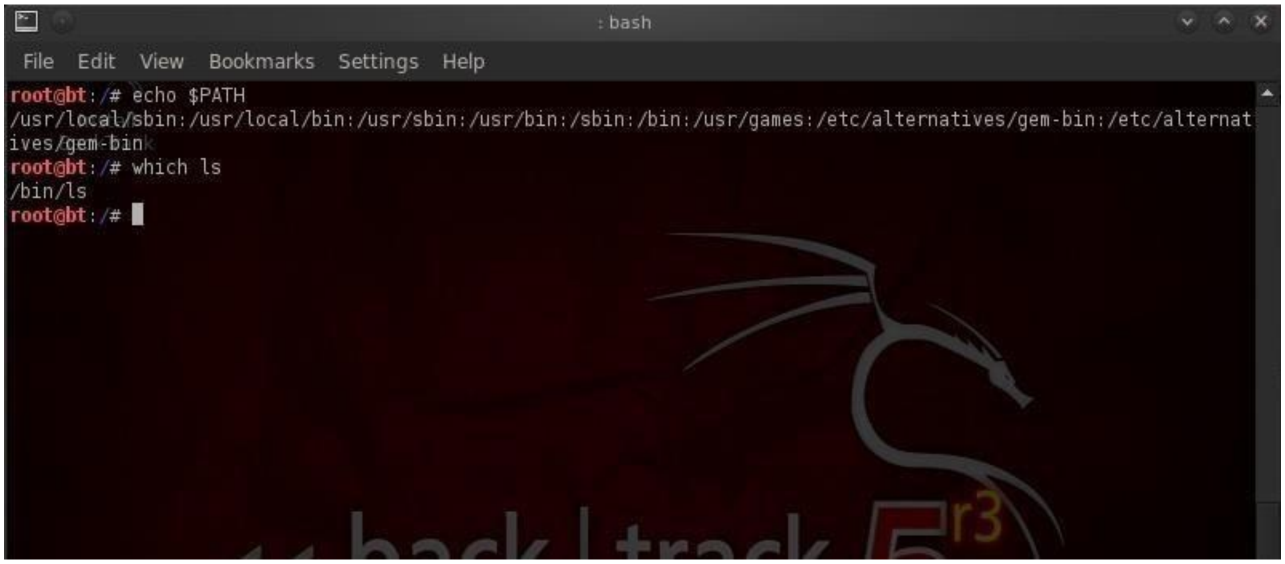
- **bt > echo $PATH**



```
root@bt:/# echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/etc/alternatives/gem-bin:/etc/alternat
ives/gem-bin
root@bt:/# 
```

Linux responds with the value in our path variable. These are the places that **which** will search for binaries. So when we type:

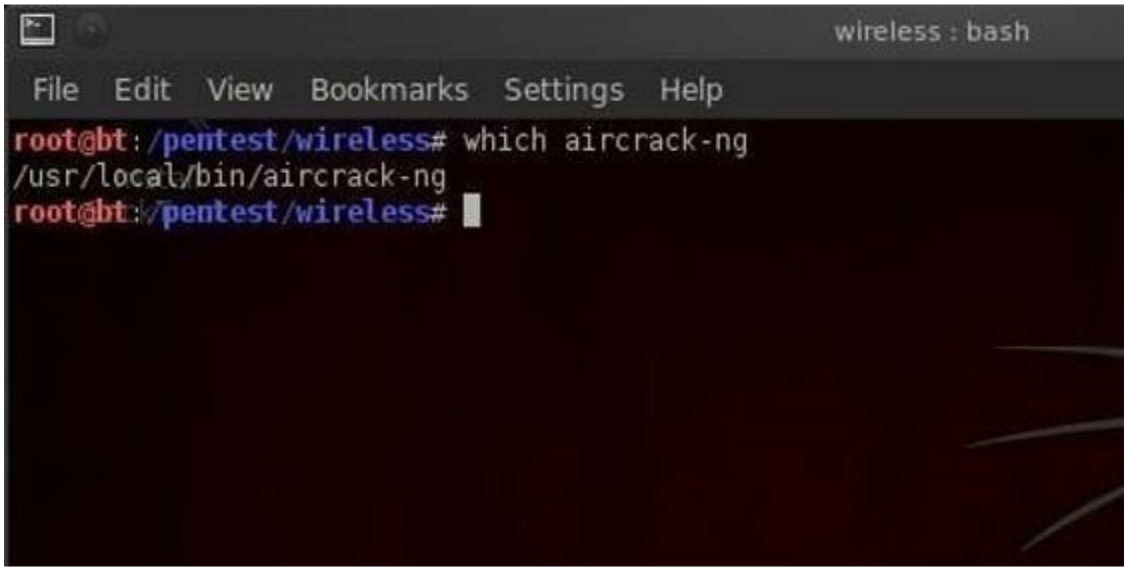
- **bt > which ls**



```
root@bt:/# echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/etc/alternatives/gem-bin:/etc/alternat
ives/gem-bin
root@bt:/# which ls
/bin/ls
root@bt:/# 
```

It returns the path to that binary. If we use **which** to search for **aircrack-ng**:

- **bt > which aircrack-ng**



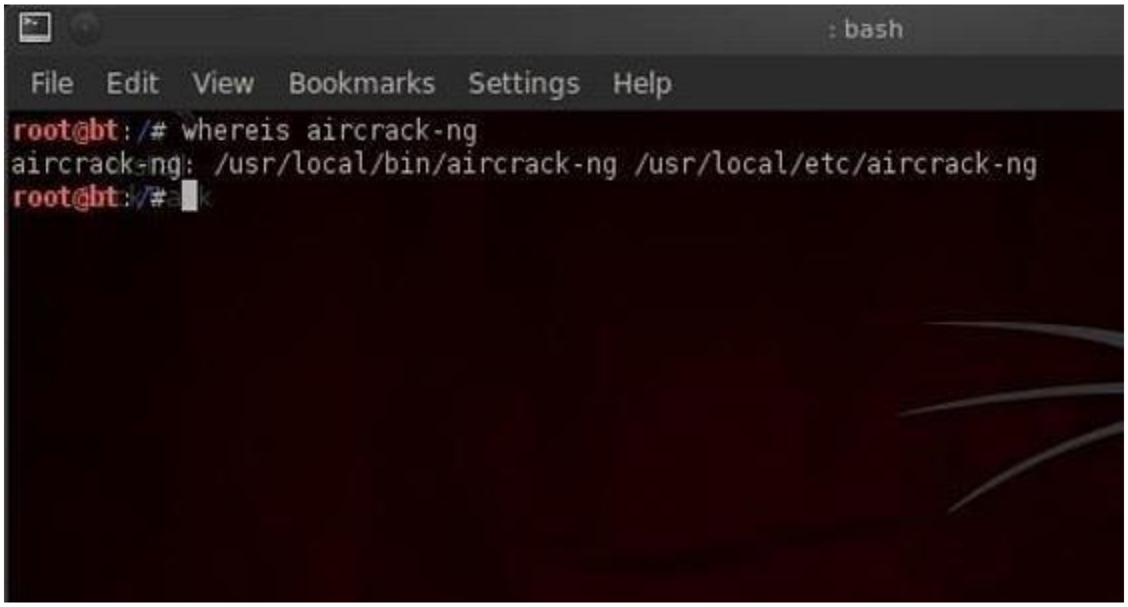
```
root@bt:/pentest/wireless# which aircrack-ng
/usr/local/bin/aircrack-ng
root@bt:/pentest/wireless#
```

Then we can see that Linux returns **/usr/local/bin/aircrack-ng**. If **aircrack-ng** were not in a directory that was in our path, it would not be able to help us.

Step 3 Finding Any File in Any Directory (Whereis)

Unlike **which**, **whereis** is not limited to finding binaries in our path. It can locate files in any directory, and in addition, it also locates the files manual or **man** pages. So, when we type:

- **bt > whereis aircrack-ng**



```
root@bt:/# whereis aircrack-ng
aircrack-ng: /usr/local/bin/aircrack-ng /usr/local/etc/aircrack-ng
root@bt:/#
```

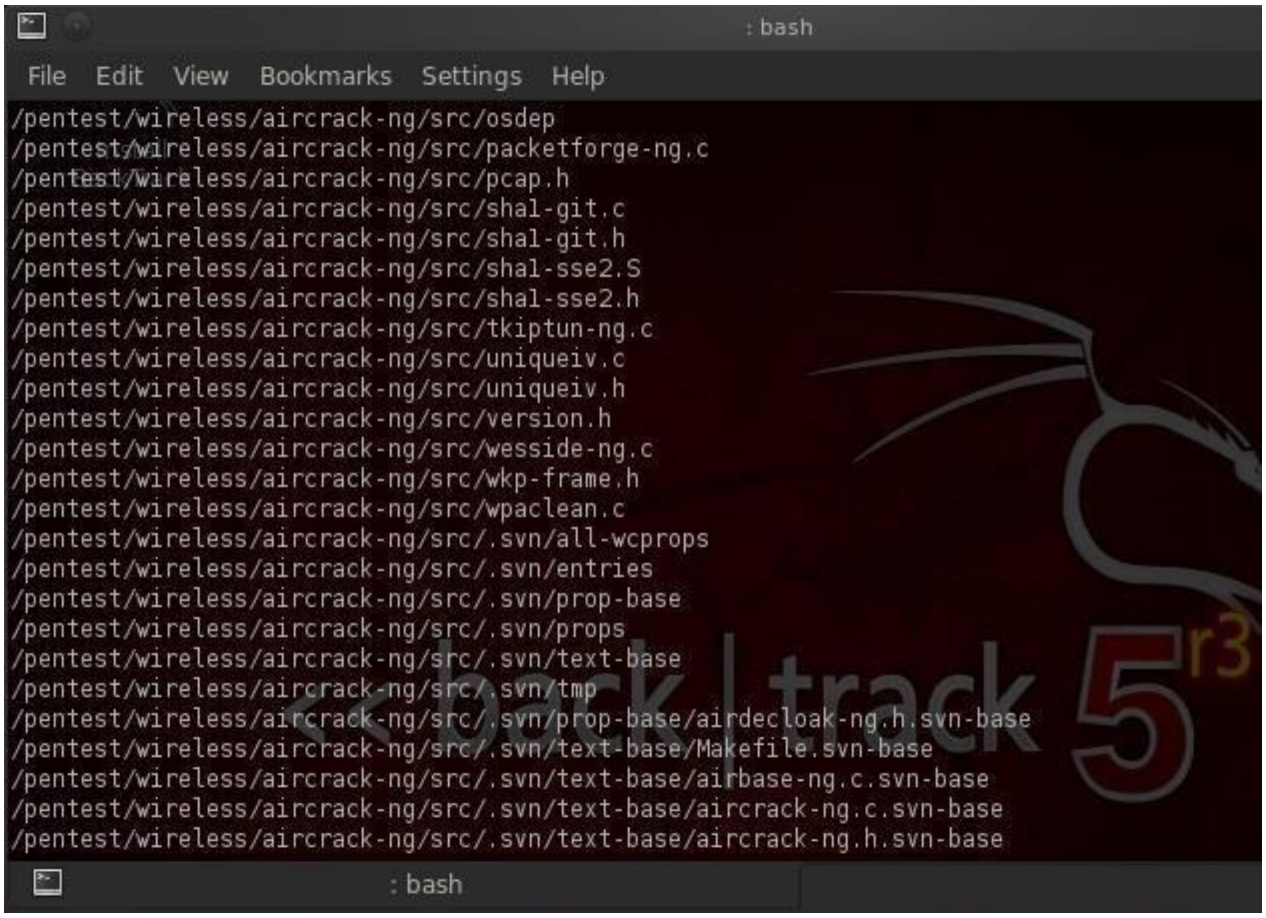
We can see that **whereis** returns the path to multiple locations of **aircrack-ng** including the **man** pages.

Step 4 Finding Files Using the Database (Locate)

The **locate** command can also be used to find files and usually is much faster than either **which** or **whereis**. The difference is that **locate** uses a database of all the files in the file system and searches therefore take place much faster.

The drawback to locate is that *new* files will NOT be found by **locate** as the database is typically only updated daily, usually scheduled in the middle of the night when activity on the system is light as updating this database can be CPU intensive.

- **locate aircrack-ng**



```
/pentest/wireless/aircrack-ng/src/osdep
/pentest/wireless/aircrack-ng/src/packetforge-ng.c
/pentest/wireless/aircrack-ng/src/pcap.h
/pentest/wireless/aircrack-ng/src/sha1-git.c
/pentest/wireless/aircrack-ng/src/sha1-git.h
/pentest/wireless/aircrack-ng/src/sha1-sse2.S
/pentest/wireless/aircrack-ng/src/sha1-sse2.h
/pentest/wireless/aircrack-ng/src/tkiptun-ng.c
/pentest/wireless/aircrack-ng/src/uniqueiv.c
/pentest/wireless/aircrack-ng/src/uniqueiv.h
/pentest/wireless/aircrack-ng/src/version.h
/pentest/wireless/aircrack-ng/src/wesside-ng.c
/pentest/wireless/aircrack-ng/src/wkp-frame.h
/pentest/wireless/aircrack-ng/src/wpaclean.c
/pentest/wireless/aircrack-ng/src/.svn/all-wcprops
/pentest/wireless/aircrack-ng/src/.svn/entries
/pentest/wireless/aircrack-ng/src/.svn/prop-base
/pentest/wireless/aircrack-ng/src/.svn/props
/pentest/wireless/aircrack-ng/src/.svn/text-base
/pentest/wireless/aircrack-ng/src/.svn/tmp
/pentest/wireless/aircrack-ng/src/.svn/prop-base/airdecloak-ng.h.svn-base
/pentest/wireless/aircrack-ng/src/.svn/text-base/Makefile.svn-base
/pentest/wireless/aircrack-ng/src/.svn/text-base/airbase-ng.c.svn-base
/pentest/wireless/aircrack-ng/src/.svn/text-base/aircrack-ng.c.svn-base
/pentest/wireless/aircrack-ng/src/.svn/text-base/aircrack-ng.h.svn-base
```

You can see in the screenshot above that locate returns a path every time it encounters any file with **aircrack-ng** in it, binary or not.

Installing New Software

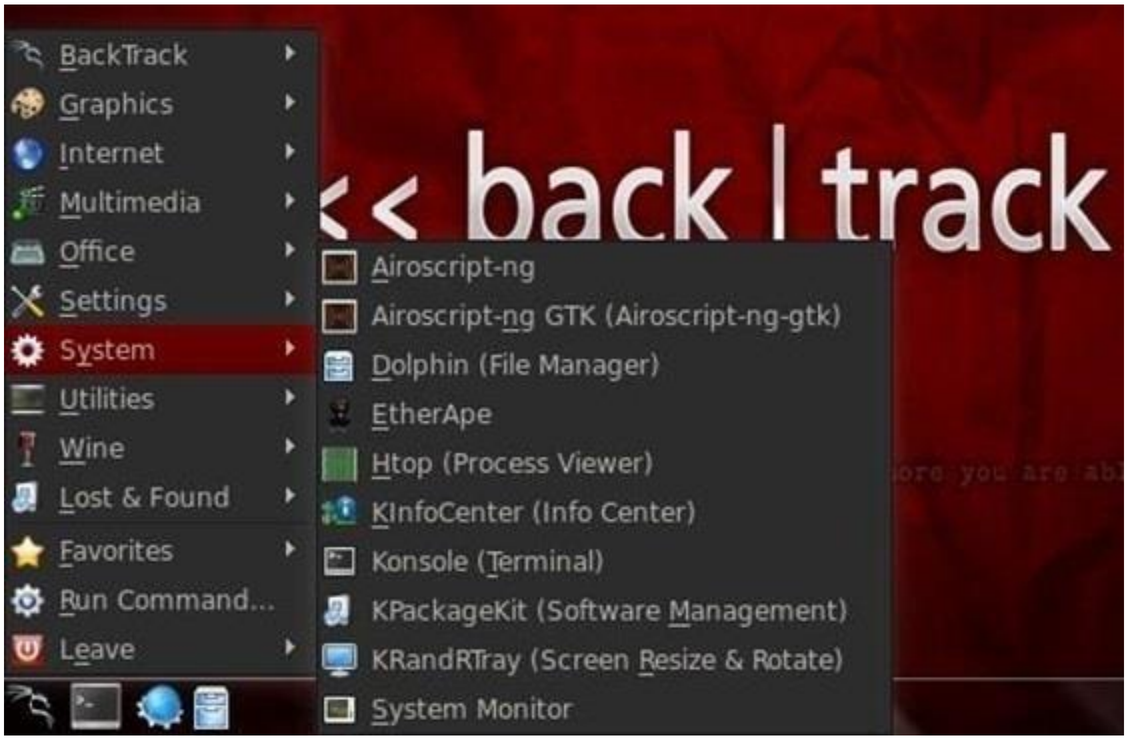
We've looked at numerous basic commands in the first few tutorials, but here I want to focus on installing new software in Linux, and especially in BackTrack.

BackTrack v5r3 was built on Ubuntu, which is a type of Debian Linux. That's important because different Linux systems use different methods for package management (package management means downloading and installing new software packages).

Step 1 Using the GUI Package Manager

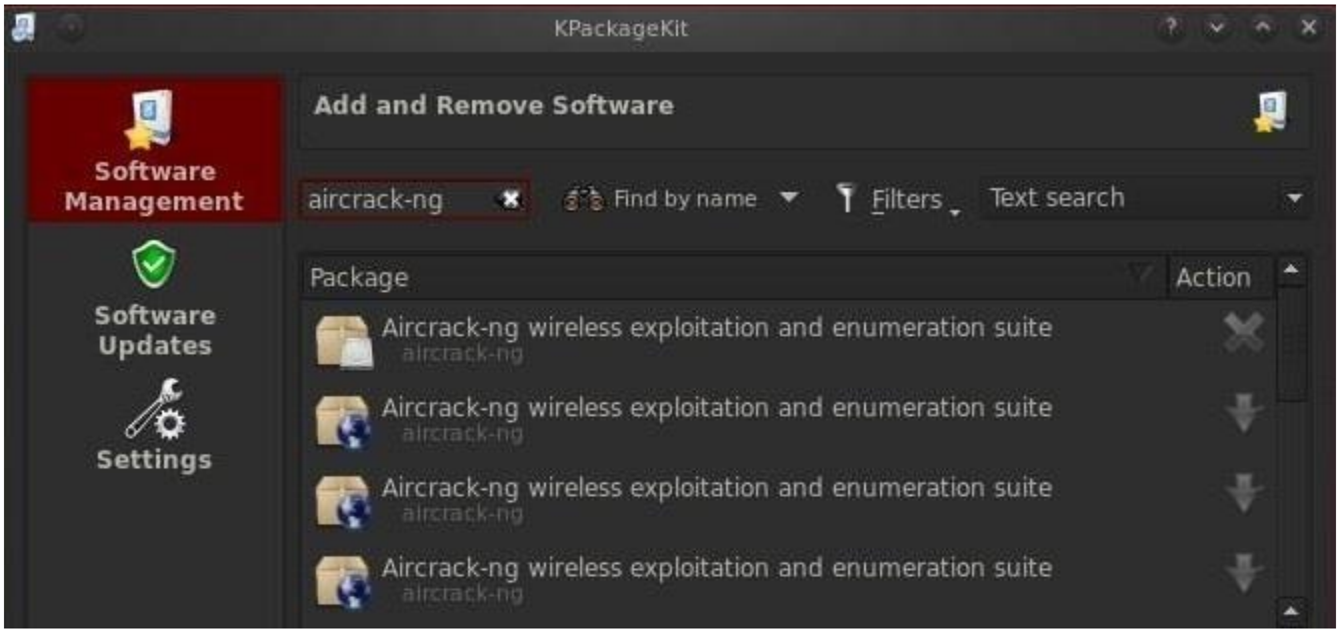
The simplest way to install software on BackTrack is to use the GUI package manager. In my KDE-based BackTrack 5, the GUI package manager is called **KPackageKit** (some of you may have **Synaptic**).

These package managers enable us find packages, download them, and install them on our system. We can open KPackageKit by navigating to **System** and then **KPackageKit** as shown in the screenshot below.



When open, you simply put the name into search field. It will then retrieve all the options fulfilling the criteria of your search, then just click on the icon next to the package you want to download.

In this example, we will be looking for the wireless hacking software, **aircrack-ng**.



Aircrack-ng wireless exploitation and enumeration suite
aircrack-ng

Aircrack-ng wireless exploitation and enumeration suite
aircrack-ng

Aircrack-ng wireless exploitation and enumeration suite
aircrack-ng

Aircrack-ng wireless exploitation and enumeration suite
aircrack-ng

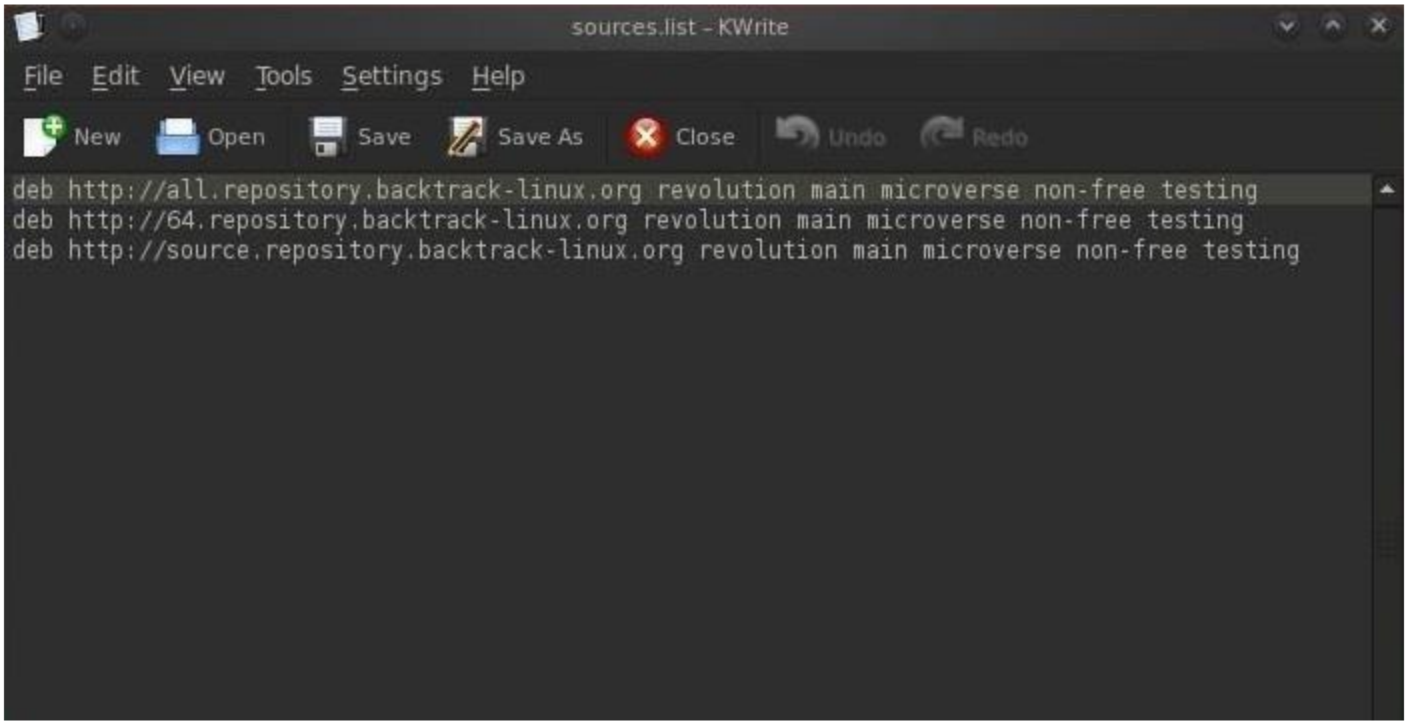
Note that if the package is already installed, there will be an **X** next to it. If not, there will be a downward-pointing arrow. Click on the arrow and then click on the **APPLY** button below.

Step 2 Updating Your Repositories

Package managers search in specified repositories (websites housing packages) for the package you are seeking. If you get a message that the package was not found, it doesn't necessarily mean that it doesn't exist, but simply that it's not in the repositories your OS is searching.

BackTrack defaults to searching in **backtrack-linux.org** where many hacking tools are available. Unfortunately, if you are looking for something that is not a hacking tool or a new hacking tool that BackTrack hasn't yet placed in its repository, you may have to revise where your operating system searching for packages.

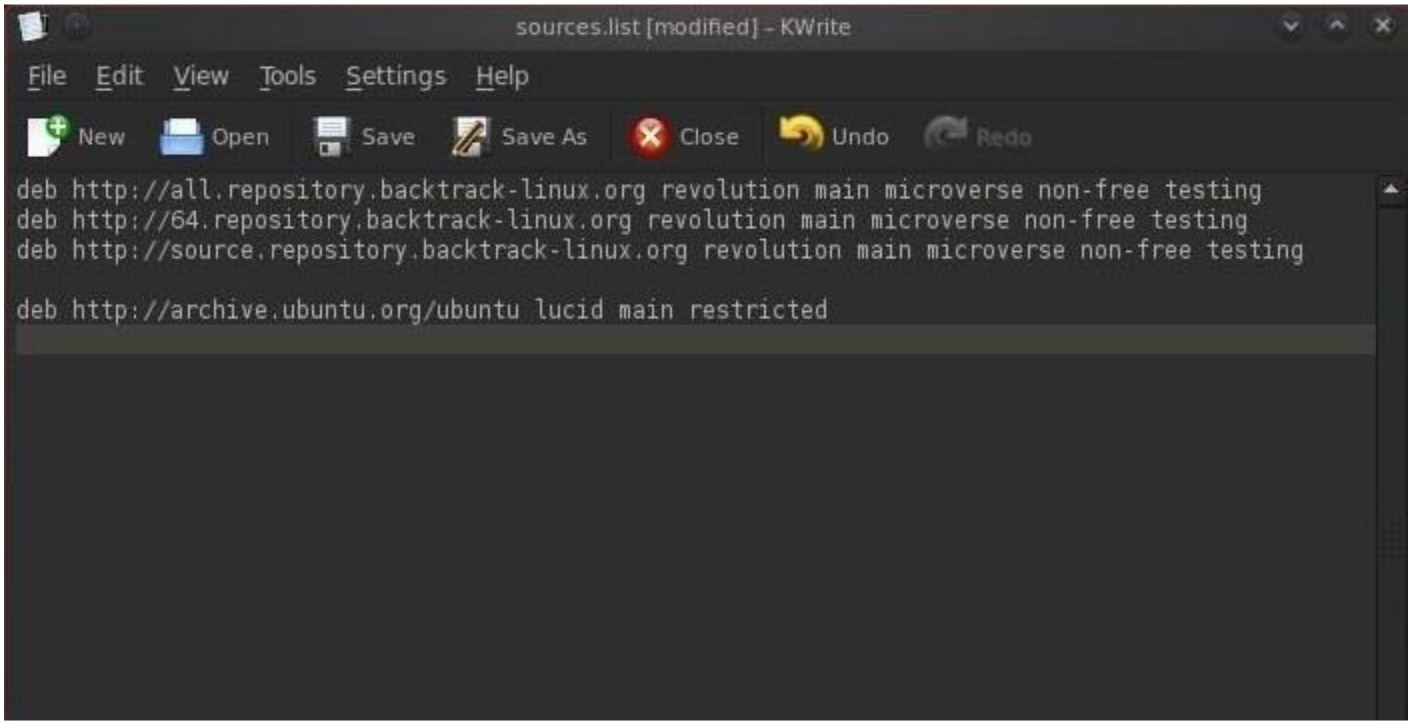
This can be done by editing the **/etc/apt/sources.list** file. Let's open it with **KWrite** and take a look.


sources.list - KWrite
File Edit View Tools Settings Help
New Open Save Save As Close Undo Redo

```
deb http://all.repository.backtrack-linux.org revolution main microverse non-free testing
deb http://64.repository.backtrack-linux.org revolution main microverse non-free testing
deb http://source.repository.backtrack-linux.org revolution main microverse non-free testing
```

As you can see, BackTrack has three default sources on its **sources.list**, all pointing to BackTrack repositories. We can add any repository with Linux software to this list, but since BackTrack is a Ubuntu distribution, we might want to add an Ubuntu repository to this list to download and install Ubuntu software. We can do this by adding a single line to this file:

- **deb http://archive.ubuntu.org/ubuntu lucid main restricted**


sources.list [modified] – KWrite
File Edit View Tools Settings Help
New Open Save Save As Close Undo Redo

```
deb http://all.repository.backtrack-linux.org revolution main microverse non-free testing
deb http://64.repository.backtrack-linux.org revolution main microverse non-free testing
deb http://source.repository.backtrack-linux.org revolution main microverse non-free testing

deb http://archive.ubuntu.org/ubuntu lucid main restricted
```

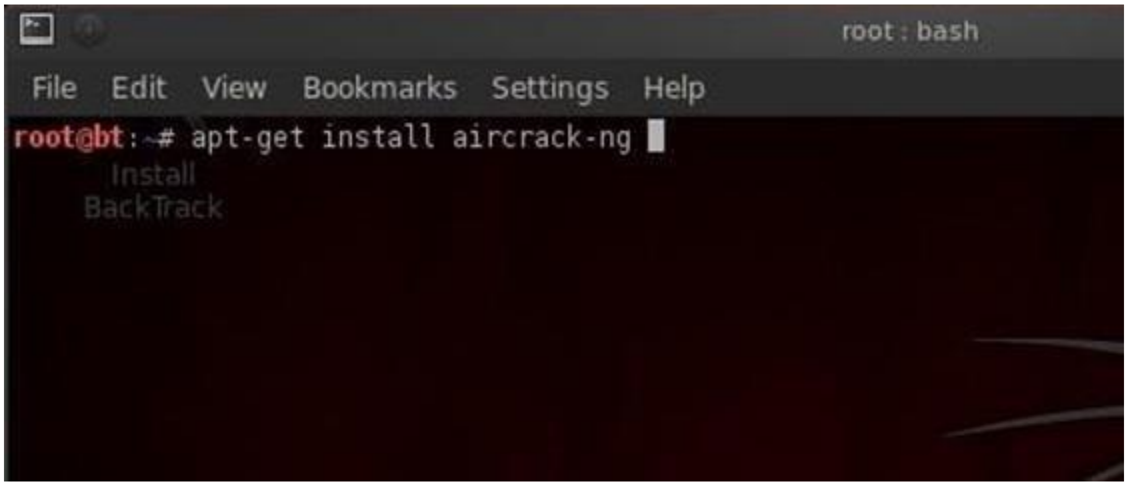
Now when I use my package manager, it will search the three BackTrack repositories first, and if it fails to find the package in any of those places, it will then search for it in the Ubuntu repository.

Step 3 Command Line Package Management

Ubuntu also has a command line package manager called **apt**. The basic syntax for using apt to download packages is:

- **apt-get install aircrack-ng**

So, let's open a terminal and type the above command to install **aircrack-ng** (of course, we just need to replace the name of the package to install other software).



```
root@bt:~# apt-get install aircrack-ng
```

If the package is in one of our repositories, it will download it and any of the necessary dependencies (files that the package need to run properly), and install it on your system automatically.

Step 4 Installing from Source

Finally, sometimes you will need to download software that is neither in a repository, nor in a package. Most often these are archived as **tar** or **tarballs**. These are files that are "tarred" together into a single file and often compressed (similar to zipping files with WinZip and then putting them together into a .zip file).

Let's say that **aircrack-ng** was not in our repository (some software never finds its way into a repository) and we had to download it from aircrack-ng.org website. We could download the file **aircrack-ng-1.2-beta1.tar**.

Once we've downlaoded it, then we need to untar it using the **tar** command:

- **tar xvf aircrack-ng-1.2-beta1.tar**

This will untar and uncompress it, if it's compressed. Next we need to compile it with the GNU compiler. Compiling from source code will give us binaries (the program files) that are optimized for our hardware and operating system, meaning they will often run faster and more efficiently. We can compile this source code by typing:

- **gcc aircrack-ng**

Finally, we can now run this file from within the directory where we unzipped it:

- **./aircrack-ng**

Note that to run the file, we preceded it with the **./**, which tells Linux to execute this file from the directory we are presently in, so make certain you run this command in the same directory that you compiled the source code in.

That should cover all the major ways of installing software and I hope it wasn't too confusing. In most cases, we can simply use the GUI based package manager to install software, but like all things in life, there are exceptions.

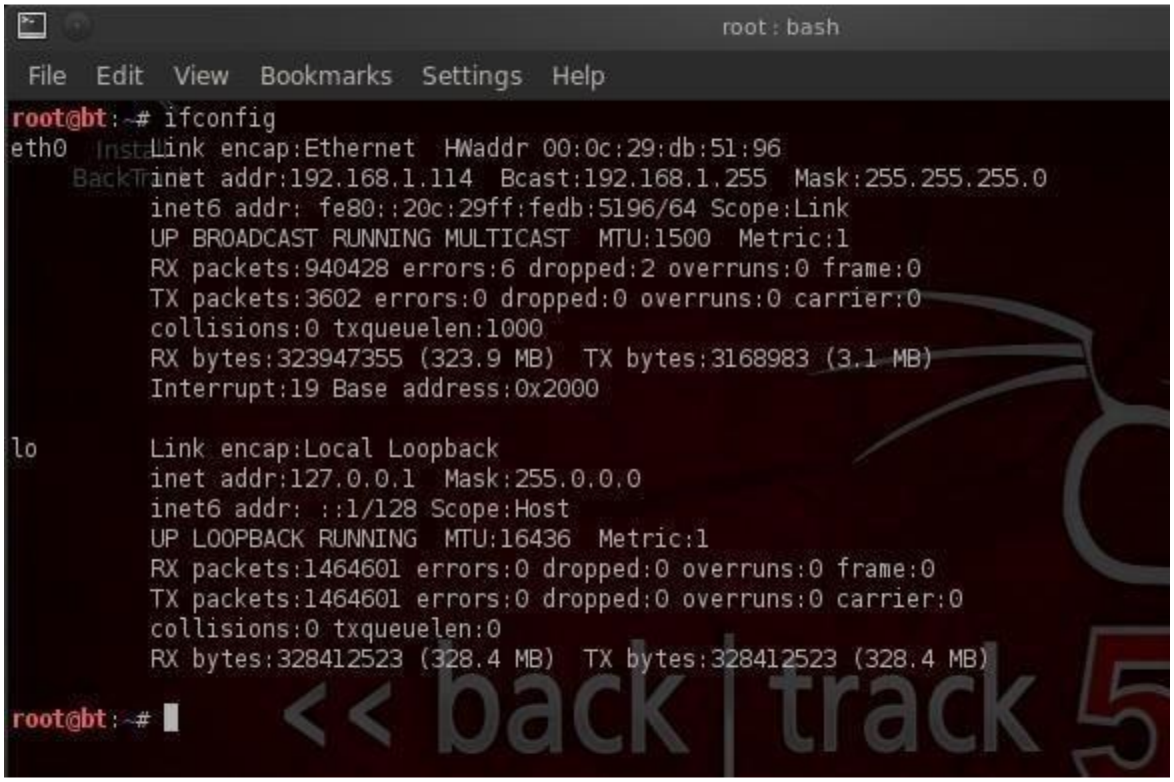
Networking Basics

I assume that you understand a small amount of networking concepts, things like IP addresses, MAC addresses, DNS, DHCP, etc.

Step 1 Analyzing Networks

The most basic linux command for analyzing networks is **ifconfig**. It's very similar to the Windows command **ipconfig**. Let's take a look at it.

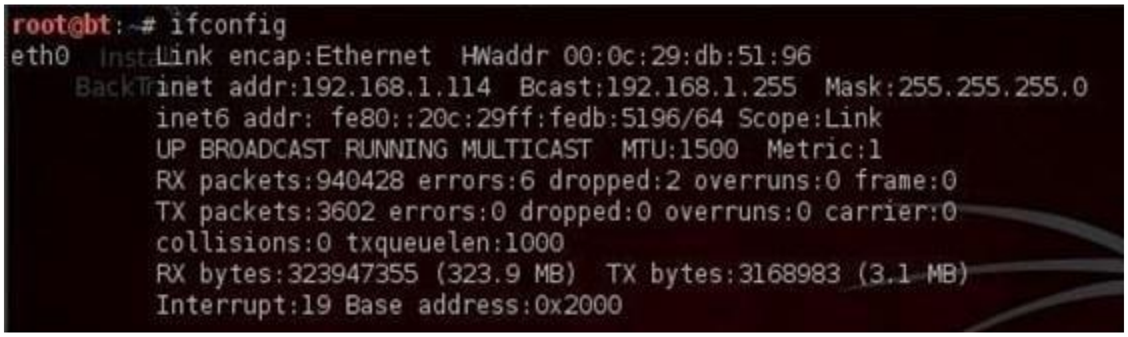
- **ifconfig**



```
root@bt:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:db:51:96
          inet addr:192.168.1.114  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fedb:5196/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:940428 errors:6 dropped:2 overruns:0 frame:0
          TX packets:3602 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:323947355 (323.9 MB)  TX bytes:3168983 (3.1 MB)
          Interrupt:19 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:1464601 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1464601 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:328412523 (328.4 MB)  TX bytes:328412523 (328.4 MB)

root@bt:~# 
```

As you can see in this screenshot, **ifconfig** conveys a significant amount of information to the user. In the very first line, we see to the far left **eth0**. This is the first **wired** network connection, **ether**net 0 (Linux usually starts counting at 0).

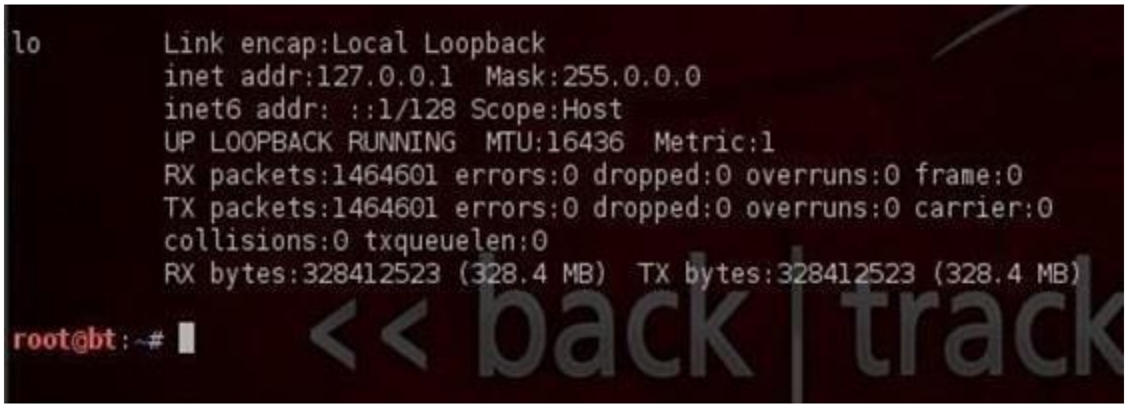
Following this, we see the type of **network** being used (Ethernet) and the **hardware address** (this is the globally unique address stamped on every piece of network hardware, in this case the NIC).



```
root@bt:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:db:51:96
          inet addr:192.168.1.114  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fedb:5196/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:940428 errors:6 dropped:2 overruns:0 frame:0
          TX packets:3602 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:323947355 (323.9 MB)  TX bytes:3168983 (3.1 MB)
          Interrupt:19 Base address:0x2000
```

The second line then contains information of the **IP address**, in this case, 192.168.1.114, the **broadcast address** (the address to send out information to all IPs on the subnet), and finally the **network mask** (this is the info on what part of the IP address is network and which part is hosts). There is a lot more technical info there, but it's beyond the scope of a Linux basics tutorial.

If we look down below to what appears to be a second paragraph, we see the start of another paragraph with **lo** to the far left.



```
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:1464601 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1464601 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:328412523 (328.4 MB)  TX bytes:328412523 (328.4 MB)

root@bt:~# 
```

This is the **loopback address** or **localhost**. This is the address of the machine you're working on if you simply wanted to test something like a website. It generally is represented with the IP address 127.0.0.1.

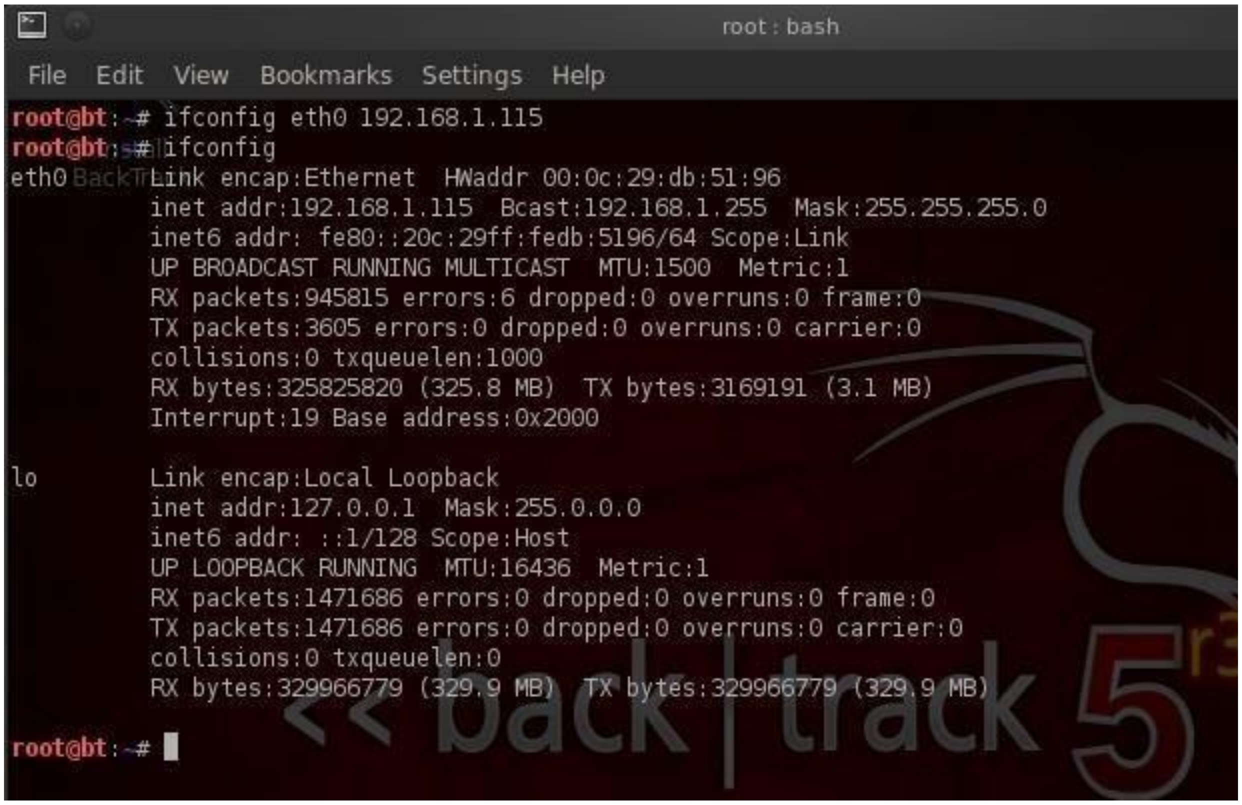
Step 2 Changing IP Addresses

Changing IP addresses can be fairly simple in Linux. Remember that in most cases, you're going to have a dynamically assigned address from a DHCP server. In some cases, you may need to reassign the address, especially if you're hacking. This can be useful in spoofing your IP address, making network forensics more challenging, but certainly not impossible.

We can do this by using the **ifconfig** command with the interface we want to assign the IP to and the IP address we want. Such as:

- **ifconfig eth0 192.168.1.115**

Now, when we type **ifconfig**, we can see that our IP address has changed to the new IP address.



```
root@bt:~# ifconfig eth0 192.168.1.115
root@bt:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:db:51:96
          inet addr:192.168.1.115  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fedb:5196/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:945815 errors:6 dropped:0 overruns:0 frame:0
          TX packets:3605 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:325825820 (325.8 MB)  TX bytes:3169191 (3.1 MB)
          Interrupt:19 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:1471686 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1471686 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:329966779 (329.9 MB)  TX bytes:329966779 (329.9 MB)

root@bt:~# 
```

We can also change the **netmask** and **broadcast address**, if necessary, such as:

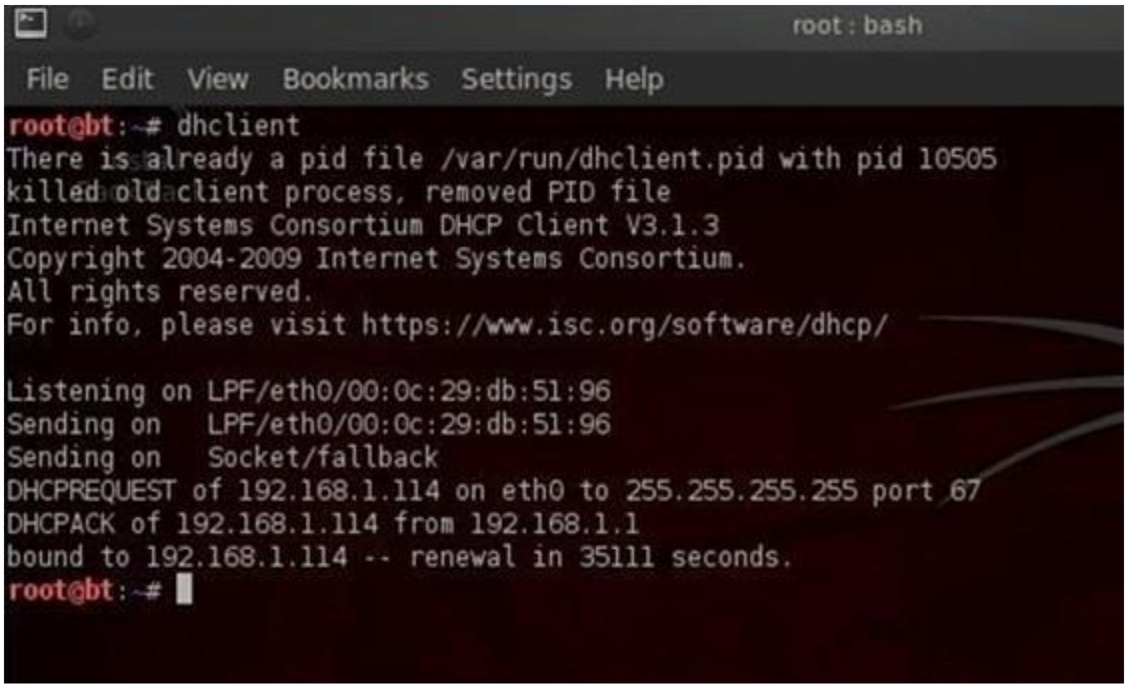
- **ifconfig eth0 192.168.1.115 netmask 255.255.255.0 broadcast 192.168.1.255**

Step 3 DHCP (Dynamic Host Configuration Server)

Linux has a DHCP server that runs a daeman called **dhcpd**. It's this DHCP server that assigns IP addresses to all the systems on the subnet. It also keeps logs files of which machines had which IP addresses at which time. It's this log that is often used to trace hackers in a forensic analysis after an attack.

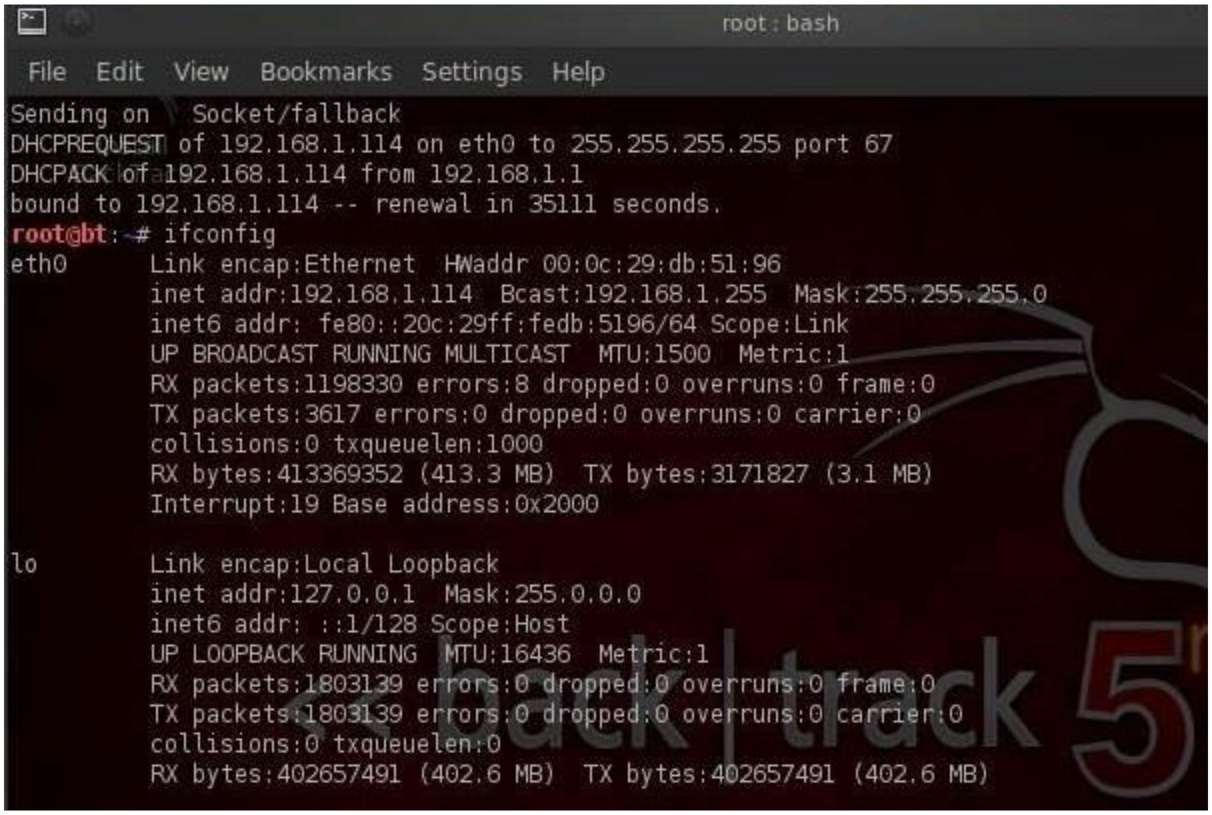
When I want to be assigned a new address from the DHCP server, I can simply call the server with the command **dhclient** (different Linux distros use different DHCP clients, but BackTrack is built on Ubuntu which uses dhclient), like this:

- **dhclient**



```
root@bt:~# dhclient
There is already a pid file /var/run/dhclient.pid with pid 10505
killed old client process, removed PID file
Internet Systems Consortium DHCP Client V3.1.3
Copyright 2004-2009 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/eth0/00:0c:29:db:51:96
Sending on   LPF/eth0/00:0c:29:db:51:96
Sending on   Socket/fallback
DHCPREQUEST of 192.168.1.114 on eth0 to 255.255.255.255 port 67
DHCPACK of 192.168.1.114 from 192.168.1.1
bound to 192.168.1.114 -- renewal in 35111 seconds.
root@bt:~# 
```

As you can see, the **dhclient** command sends out **DHCPDISCOVER** request from the default NIC. It then gets an offer (**DHCPOFFER**) of 192.168.1.114 from the DHCP server, then confirms the IP assignment to the DHCP server. Now, if we type **ifconfig**, we can see that the DHCP server has assigned a new IP address.



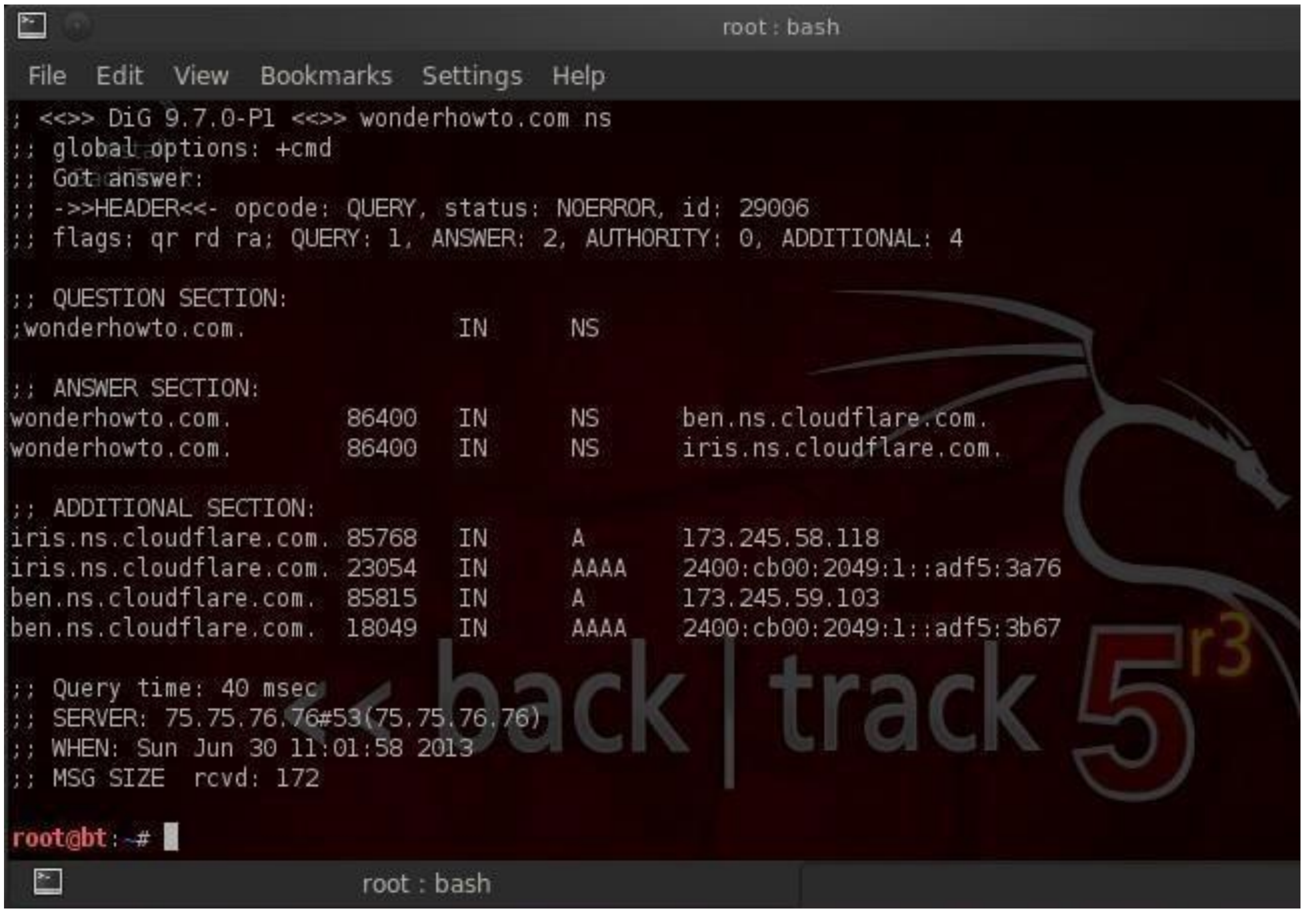
```
Sending on   Socket/fallback
DHCPREQUEST of 192.168.1.114 on eth0 to 255.255.255.255 port 67
DHCPACK of 192.168.1.114 from 192.168.1.1
bound to 192.168.1.114 -- renewal in 35111 seconds.
root@bt:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:db:51:96
          inet addr:192.168.1.114  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fedb:5196/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1198330 errors:8 dropped:0 overruns:0 frame:0
          TX packets:3617 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:413369352 (413.3 MB)  TX bytes:3171827 (3.1 MB)
          Interrupt:19 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:1803139 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1803139 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:402657491 (402.6 MB)  TX bytes:402657491 (402.6 MB)
```

Step 4 DNS (Domain Name Service)

DNS, or Domain Name Services, is the service that enables us to type in a domain name like www.wonderhowto.com, which it then translates to the appropriate IP address. Without it, we would all have to remember thousands of IP addresses of our favorite websites (no small task even for a savant).

One of the most useful commands for the aspiring hacker is **dig**, which is the equivalent of **nslookup** in Windows, but offers us much more information on the domain. For instance, we **dig** wonderhowto.com and by adding the **ns** option, it will display the name server for wonderhowto.com.

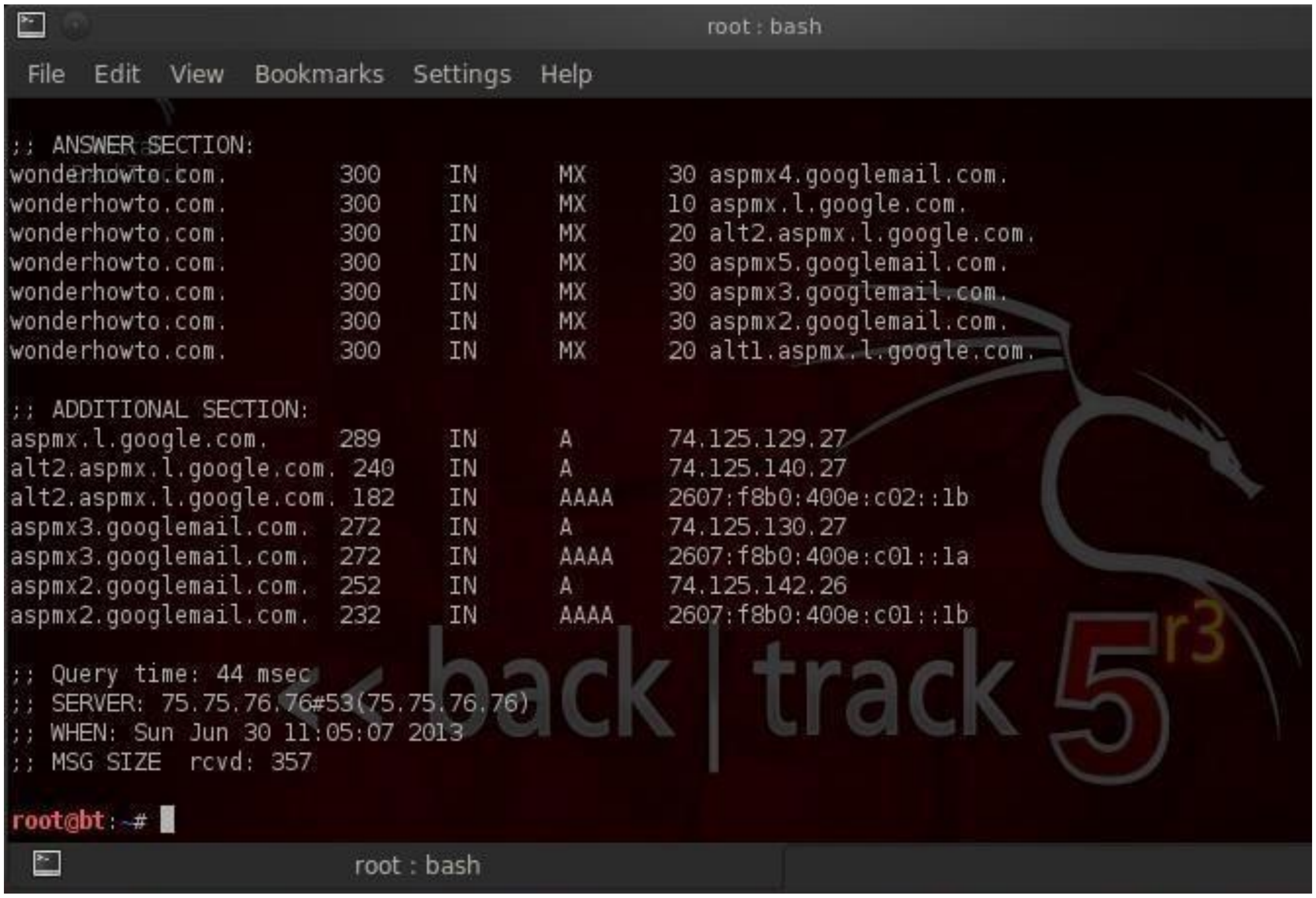
- **dig wonderhowto.com ns**



```
; <<>> DiG 9.7.0-P1 <<>> wonderhowto.com ns
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 29006
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 4

;; QUESTION SECTION:
;wonderhowto.com.               IN      NS

;; ANSWER SECTION:
wonderhowto.com.        86400   IN      NS      ben.ns.cloudflare.com.
wonderhowto.com.        86400   IN      NS      iris.ns.cloudflare.com.

;; ADDITIONAL SECTION:
iris.ns.cloudflare.com. 85768   IN      A       173.245.58.118
iris.ns.cloudflare.com. 23054   IN      AAAA    2400:cb00:2049:1::adf5:3a76
ben.ns.cloudflare.com.  85815   IN      A       173.245.59.103
ben.ns.cloudflare.com.  18049   IN      AAAA    2400:cb00:2049:1::adf5:3b67

;; Query time: 40 msec
;; SERVER: 75.75.76.76#53(75.75.76.76)
;; WHEN: Sun Jun 30 11:01:58 2013
;; MSG SIZE  rcvd: 172

root@bt:~#
```

By using the dig command with the **mx** option, we can get info on WonderHowTo's email servers.

- **dig wonderhowto.com mx**

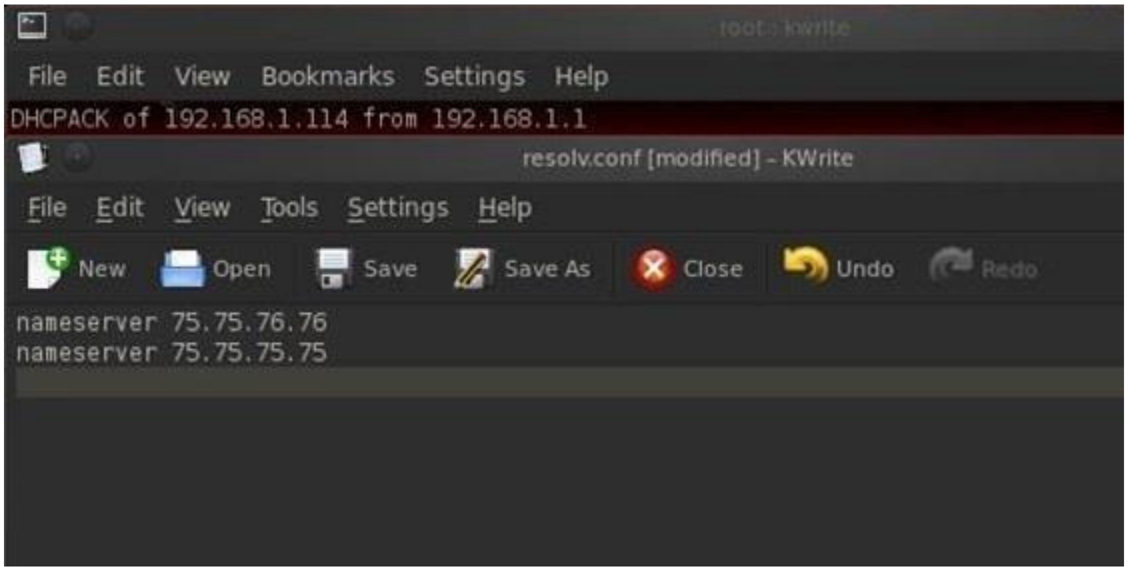


```
;; ANSWER SECTION:
wonderhowto.com.        300     IN      MX      30 aspmx4.googlemail.com.
wonderhowto.com.        300     IN      MX      10 aspmx.l.google.com.
wonderhowto.com.        300     IN      MX      20 alt2.aspmx.l.google.com.
wonderhowto.com.        300     IN      MX      30 aspmx5.googlemail.com.
wonderhowto.com.        300     IN      MX      30 aspmx3.googlemail.com.
wonderhowto.com.        300     IN      MX      30 aspmx2.googlemail.com.
wonderhowto.com.        300     IN      MX      20 alt1.aspmx.l.google.com.

;; ADDITIONAL SECTION:
aspmx.l.google.com.     289     IN      A       74.125.129.27
alt2.aspmx.l.google.com. 240    IN      A       74.125.140.27
alt2.aspmx.l.google.com. 182    IN      AAAA    2607:f8b0:400e:c02::1b
aspmx3.googlemail.com.  272     IN      A       74.125.130.27
aspmx3.googlemail.com.  272     IN      AAAA    2607:f8b0:400e:c01::1a
aspmx2.googlemail.com.  252     IN      A       74.125.142.26
aspmx2.googlemail.com.  232     IN      AAAA    2607:f8b0:400e:c01::1b

;; Query time: 44 msec
;; SERVER: 75.75.76.76#53(75.75.76.76)
;; WHEN: Sun Jun 30 11:05:07 2013
;; MSG SIZE  rcvd: 357

root@bt:~#
```

The most common Linux DNS server is the Berkeley Internet Name Domain, or **BIND**. In some cases, Linux users will often refer to DNS as BIND, so don't be confused. DNS or BIND simply maps individual domain names to IP addresses.

On our BackTrack system, we can point out DNS services to a local DNS server or a public DNS server. This pointing takes place in the a plain text tile named **/etc/resolv.conf** file. Let's open it with **kwrite**:

- **kwrite /etc/resolv.conf**


root : kwrite
File Edit View Bookmarks Settings Help

```
DHCPACK of 192.168.1.114 from 192.168.1.1
```

resolv.conf [modified] – KWrite
File Edit View Tools Settings Help
New Open Save Save As Close Undo Redo

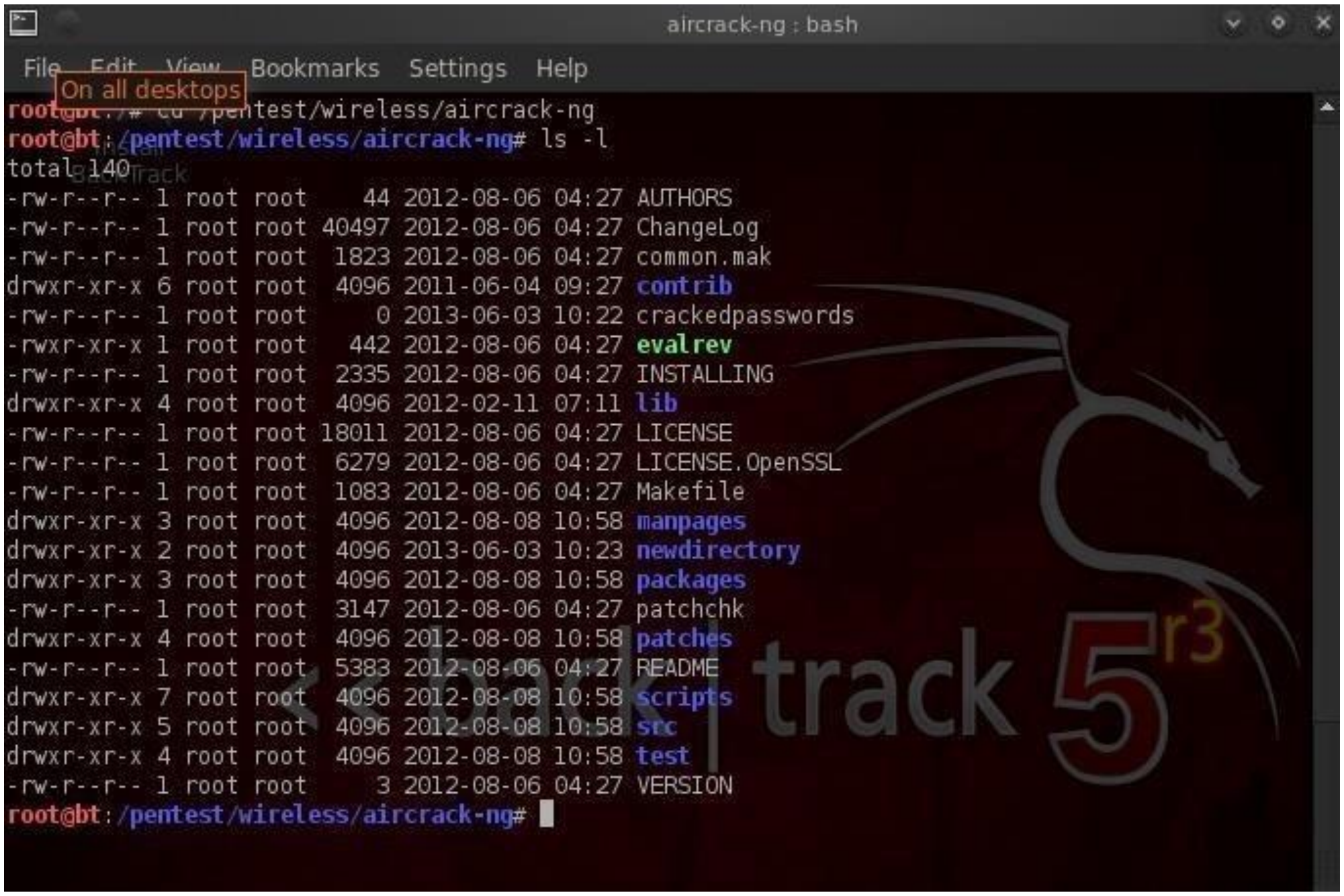
```
nameserver 75.75.76.76
nameserver 75.75.75.75
```

As you can see, we are pointing to two public DNS servers to provide us with DNS services. If we want to change our DNS servers or add another server, we can simply add another line to this text file and save it. The next time DNS services are required, the Linux operating system will look to the new DNS server designated in this file.

Managing Permissions

Step 1 Checking Permissions

When we want to find the permissions on a file, we can simply use the **ls** command with the **-l** or **long switch**. Let's use that command in the **pentest/wireless/aircrack-ng** directory and see what it tells us about the files there.

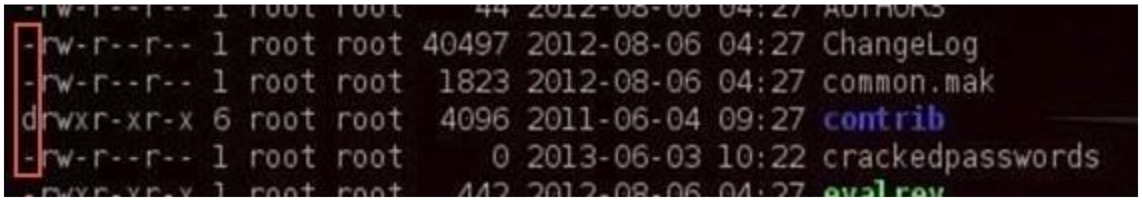


```
root@bt:/# cd /pentest/wireless/aircrack-ng
root@bt:/pentest/wireless/aircrack-ng# ls -l
total 140
-rw-r--r-- 1 root root    44 2012-08-06 04:27 AUTHORS
-rw-r--r-- 1 root root 40497 2012-08-06 04:27 ChangeLog
-rw-r--r-- 1 root root  1823 2012-08-06 04:27 common.mak
drwxr-xr-x 6 root root  4096 2011-06-04 09:27 contrib
-rw-r--r-- 1 root root     0 2013-06-03 10:22 crackedpasswords
-rwxr-xr-x 1 root root   442 2012-08-06 04:27 evalrev
-rw-r--r-- 1 root root  2335 2012-08-06 04:27 INSTALLING
drwxr-xr-x 4 root root  4096 2012-02-11 07:11 lib
-rw-r--r-- 1 root root 18011 2012-08-06 04:27 LICENSE
-rw-r--r-- 1 root root  6279 2012-08-06 04:27 LICENSE.OpenSSL
-rw-r--r-- 1 root root  1083 2012-08-06 04:27 Makefile
drwxr-xr-x 3 root root  4096 2012-08-08 10:58 manpages
drwxr-xr-x 2 root root  4096 2013-06-03 10:23 newdirectory
drwxr-xr-x 3 root root  4096 2012-08-08 10:58 packages
-rw-r--r-- 1 root root  3147 2012-08-06 04:27 patchchk
drwxr-xr-x 4 root root  4096 2012-08-08 10:58 patches
-rw-r--r-- 1 root root  5383 2012-08-06 04:27 README
drwxr-xr-x 7 root root  4096 2012-08-08 10:58 scripts
drwxr-xr-x 5 root root  4096 2012-08-08 10:58 src
drwxr-xr-x 4 root root  4096 2012-08-08 10:58 test
-rw-r--r-- 1 root root     3 2012-08-06 04:27 VERSION
root@bt:/pentest/wireless/aircrack-ng# 
```

If we look at each line, we can see quite a bit of info on the file including whether it's a file or directory, the permissions on the file, the number of links, the owner of the file, the group owner of the file, the size of the file, when it was created or modified, and finally, the name of the file. Let's examine each of these.

Identifying a File or Directory

The very first character of the line tells us whether it's a file or directory. If the line begins with a **d**, it's a directory. If it begins with a **-**, it's a file.

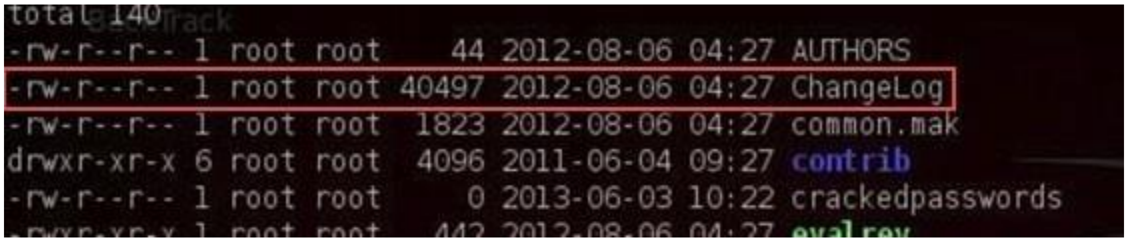


```
-rw-r--r-- 1 root root 40497 2012-08-06 04:27 ChangeLog
-rw-r--r-- 1 root root  1823 2012-08-06 04:27 common.mak
drwxr-xr-x 6 root root  4096 2011-06-04 09:27 contrib
-rw-r--r-- 1 root root     0 2013-06-03 10:22 crackedpasswords
```

Identifying the Permissions

The next section of characters defines the permissions on the file. There are three sets of **rwx** that stands for **read**, **write** and **execute**. This determines whether there is the permission to read the file, write to the file, or execute the file. Each set of **rwx** represents the permissions of the owner, group, and then all others.

So, if we look at the second line for the ChangeLog file...

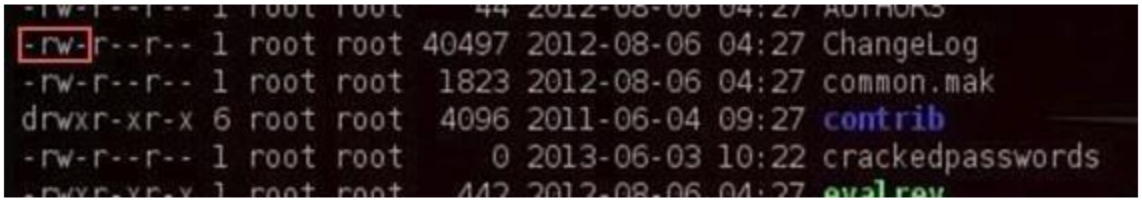


```
total 140
-rw-r--r-- 1 root root    44 2012-08-06 04:27 AUTHORS
-rw-r--r-- 1 root root 40497 2012-08-06 04:27 ChangeLog
-rw-r--r-- 1 root root  1823 2012-08-06 04:27 common.mak
drwxr-xr-x 6 root root  4096 2011-06-04 09:27 contrib
-rw-r--r-- 1 root root     0 2013-06-03 10:22 crackedpasswords
```

We can see that it begins with:

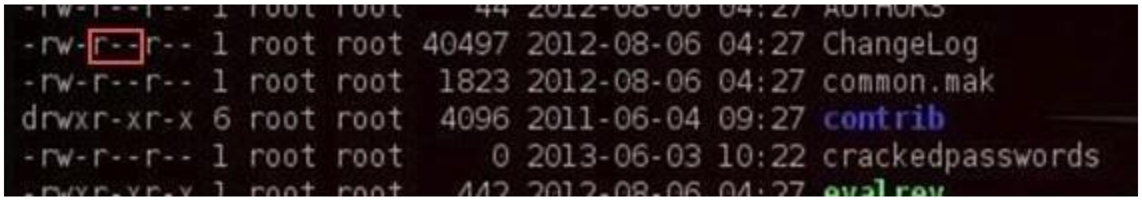
- **-rw-r--r--**

This means that it's a file (**-**) where the owner has read (**r**) and write (**w**) permissions, but not execute permission (**-**).



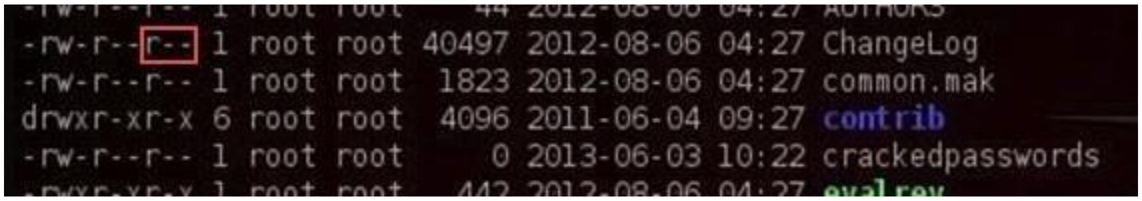
```
-rw-r--r-- 1 root root 40497 2012-08-06 04:27 ChangeLog
-rw-r--r-- 1 root root  1823 2012-08-06 04:27 common.mak
drwxr-xr-x 6 root root  4096 2011-06-04 09:27 contrib
-rw-r--r-- 1 root root     0 2013-06-03 10:22 crackedpasswords
```

The next set of permissions represents those of the group. Here we can see that the group has read permissions (**r**), but not write (**-**) or execute permission (**-**).



```
-rw-r--r-- 1 root root 40497 2012-08-06 04:27 ChangeLog
-rw-r--r-- 1 root root  1823 2012-08-06 04:27 common.mak
drwxr-xr-x 6 root root  4096 2011-06-04 09:27 contrib
-rw-r--r-- 1 root root     0 2013-06-03 10:22 crackedpasswords
```

Finally, the last set of permissions are for all others. We can see that all others have only the read (**r**) permission on the ChangeLog file.



```
-rw-r--r-- 1 root root 40497 2012-08-06 04:27 ChangeLog
-rw-r--r-- 1 root root  1823 2012-08-06 04:27 common.mak
drwxr-xr-x 6 root root  4096 2011-06-04 09:27 contrib
-rw-r--r-- 1 root root     0 2013-06-03 10:22 crackedpasswords
```

Step 2 Changing Permissions

Let's imagine a case where we wanted the group to be able to both write and execute the ChangeLog file. Linux has a command called **chmod** that allows us to change the permissions on a file as long as we're root or the owner of the file. These permissions are represented by their binary equivalents in the operating system.

The Numbers

Remember that everything is simply zeros and ones in the underlying operating system, and these permissions are represented by on and off switches in the system. So, if we could imagine the permissions as three on/off switches and these switches are in the base two-number system, the far right switch represents 1 when it's on, the middle switch represents 2 when it's on, and finally, the far left switch represents 4 when on.

So, the three permissions look like this when they are all on:

- **r w x**
- **4 2 1 = 7**

If you sum these three, you get seven, right? In Linux, when all the permission switches are on, we can represent it with the decimal numerical equivalent of 7. So, if we wanted to represent that the owner (**7**) and the group (**7**) and all users (**7**) had all permissions, we could represent it as:

- **777**

Now, lets go back to our ChangeLog file. Remember its permissions? They were **rw-r--r--**, so we could represent that numerically like:

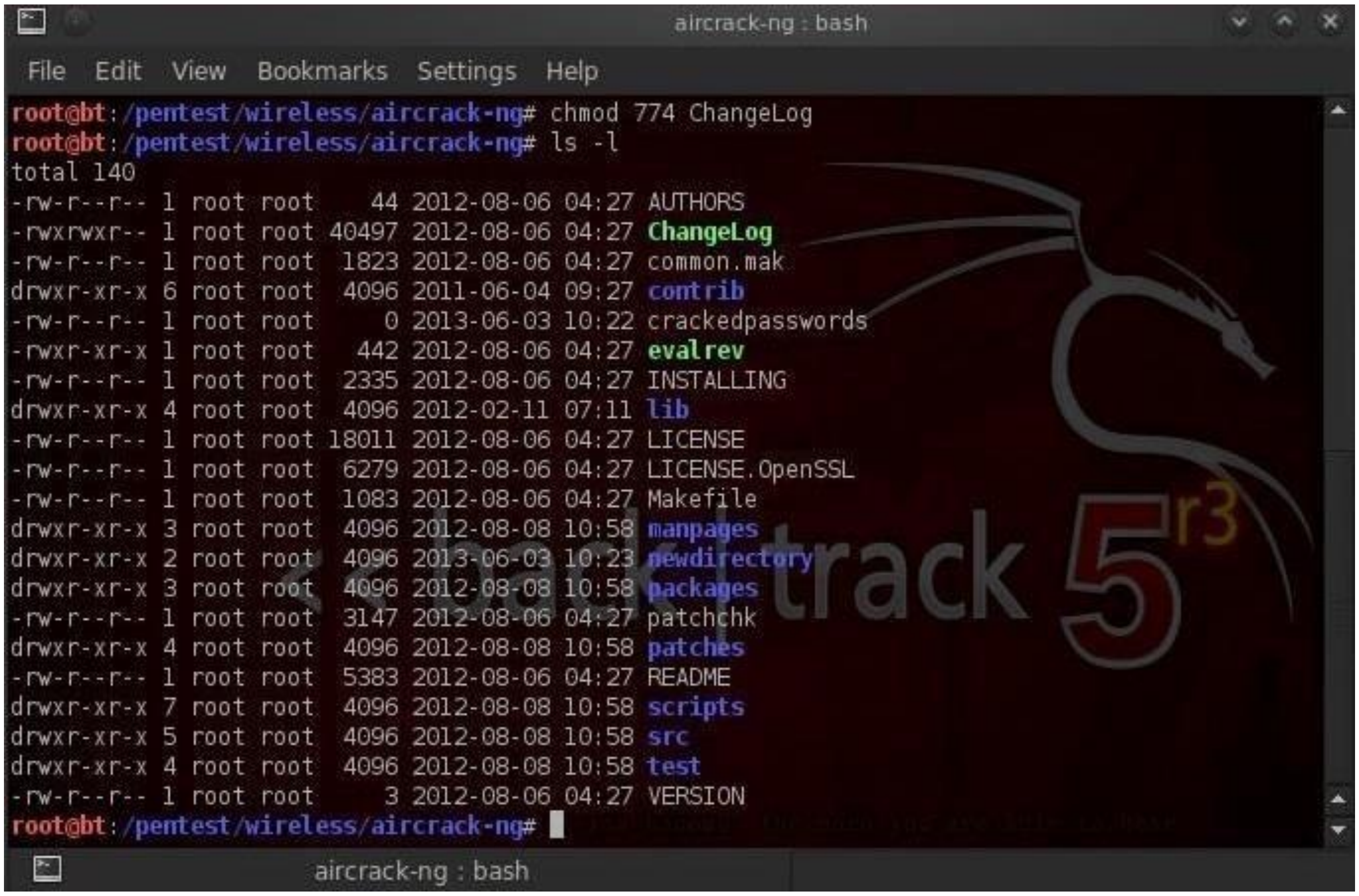
- **r w - r - - r - -**
- **4 2 0 4 0 0 4 0 0**

This can be represented by **644**.

Changing the Actual Permissions of ChangeLog

Now, if we wanted to give the group write (**2**) and execute (**1**) privilege, we can use the **chmod** command to do it. We need to add the write (**2**) privilege and the execute (**1**) privilege to the ChangeLog file. We do that by:

- **chmod 7 7 4 ChangeLog**



```
root@bt:/pentest/wireless/aircrack-ng# chmod 774 ChangeLog
root@bt:/pentest/wireless/aircrack-ng# ls -l
total 140
-rw-r--r-- 1 root root    44 2012-08-06 04:27 AUTHORS
-rwxrwxr-- 1 root root 40497 2012-08-06 04:27 ChangeLog
-rw-r--r-- 1 root root  1823 2012-08-06 04:27 common.mak
drwxr-xr-x 6 root root  4096 2011-06-04 09:27 contrib
-rw-r--r-- 1 root root     0 2013-06-03 10:22 crackedpasswords
-rwxr-xr-x 1 root root   442 2012-08-06 04:27 evalrev
-rw-r--r-- 1 root root  2335 2012-08-06 04:27 INSTALLING
drwxr-xr-x 4 root root  4096 2012-02-11 07:11 lib
-rw-r--r-- 1 root root 18011 2012-08-06 04:27 LICENSE
-rw-r--r-- 1 root root  6279 2012-08-06 04:27 LICENSE.OpenSSL
-rw-r--r-- 1 root root  1083 2012-08-06 04:27 Makefile
drwxr-xr-x 3 root root  4096 2012-08-08 10:58 manpages
drwxr-xr-x 2 root root  4096 2013-06-03 10:23 newdirectory
drwxr-xr-x 3 root root  4096 2012-08-08 10:58 packages
-rw-r--r-- 1 root root  3147 2012-08-06 04:27 patchchk
drwxr-xr-x 4 root root  4096 2012-08-08 10:58 patches
-rw-r--r-- 1 root root  5383 2012-08-06 04:27 README
drwxr-xr-x 7 root root  4096 2012-08-08 10:58 scripts
drwxr-xr-x 5 root root  4096 2012-08-08 10:58 src
drwxr-xr-x 4 root root  4096 2012-08-08 10:58 test
-rw-r--r-- 1 root root     3 2012-08-06 04:27 VERSION
root@bt:/pentest/wireless/aircrack-ng#
```

This statements says give the owner all permissions (**4+2+1=7**), the group the same (**4+2+1=7**). and give everyone else simply read permission (**4+0+0=4**). When we now do a **ls -l**, we can see that the permissions for ChangeLog are now:

- **r w x r w x r - -**

Simple, right?

Step 3 Changing Permissions with UGO

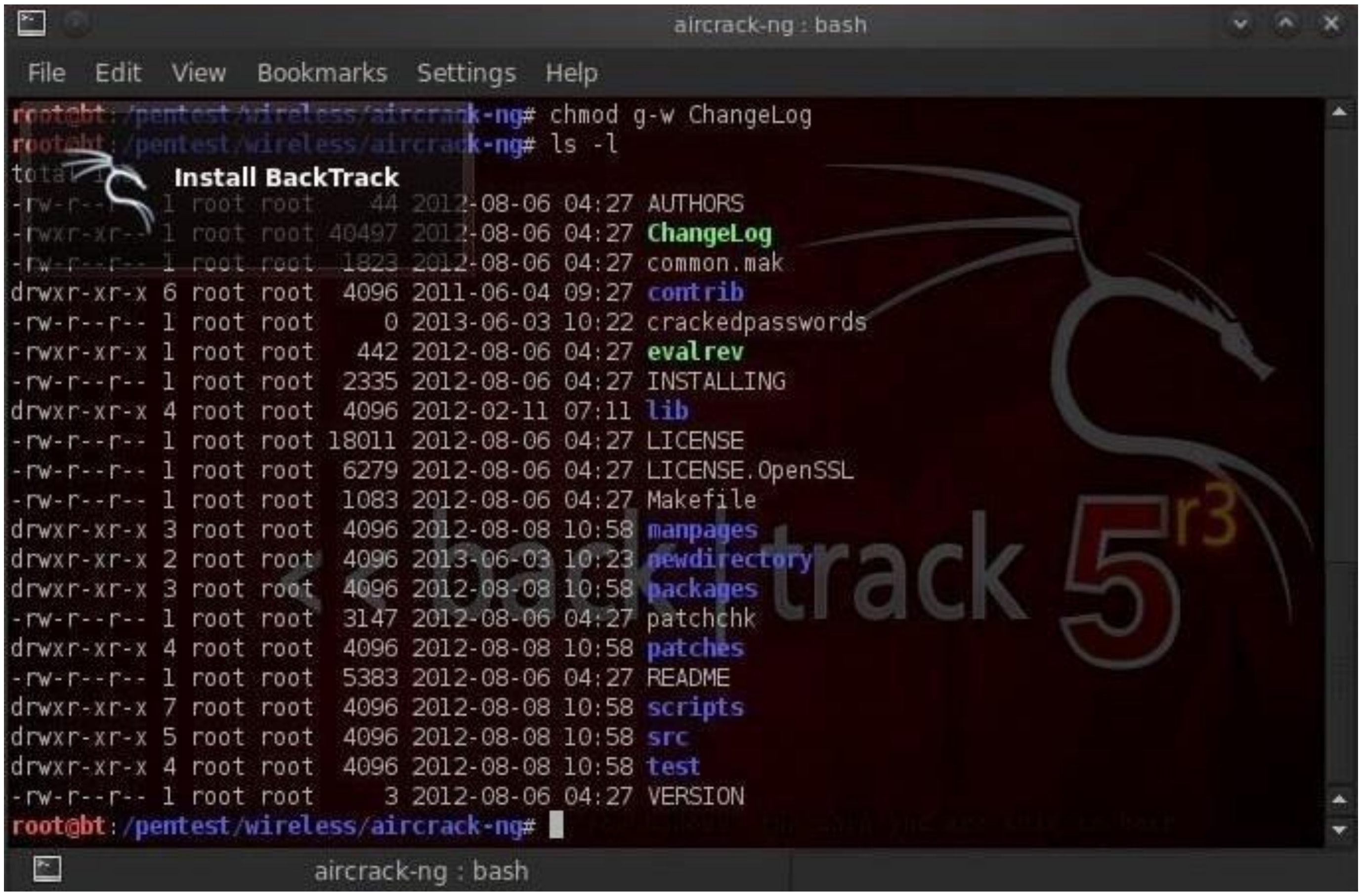
Although the numeric method is probably the most common method for changing permissions in Linux (every self-respecting Linux guru can use it), there's another method that some people are more comfortable with. It's often referred to as the **UGO syntax**. UGO stands for **U**=user or owner, **G**=group and **O**=others. UGO has three operators:

- **+** for **add** a permission
- **-** for **subtract** a permission
- **=** to **set** a permission

So, if I wanted to subtract the write permission to the group that ChangeLog belongs to, I could write:

- **chmod g-w ChangeLog**

This command says "**for the group (g) subtract (-) the write (w) permission to ChangeLog**."



```
root@bt:/pentest/wireless/aircrack-ng# chmod g-w ChangeLog
root@bt:/pentest/wireless/aircrack-ng# ls -l
total
-rw-r--r-- 1 root root    44 2012-08-06 04:27 AUTHORS
-rwxr-xr-x 1 root root 40497 2012-08-06 04:27 ChangeLog
-rw-r--r-- 1 root root  1823 2012-08-06 04:27 common.mak
drwxr-xr-x 6 root root  4096 2011-06-04 09:27 contrib
-rw-r--r-- 1 root root     0 2013-06-03 10:22 crackedpasswords
-rwxr-xr-x 1 root root   442 2012-08-06 04:27 evalrev
-rw-r--r-- 1 root root  2335 2012-08-06 04:27 INSTALLING
drwxr-xr-x 4 root root  4096 2012-02-11 07:11 lib
-rw-r--r-- 1 root root 18011 2012-08-06 04:27 LICENSE
-rw-r--r-- 1 root root  6279 2012-08-06 04:27 LICENSE.OpenSSL
-rw-r--r-- 1 root root  1083 2012-08-06 04:27 Makefile
drwxr-xr-x 3 root root  4096 2012-08-08 10:58 manpages
drwxr-xr-x 2 root root  4096 2013-06-03 10:23 newdirectory
drwxr-xr-x 3 root root  4096 2012-08-08 10:58 packages
-rw-r--r-- 1 root root  3147 2012-08-06 04:27 patchchk
drwxr-xr-x 4 root root  4096 2012-08-08 10:58 patches
-rw-r--r-- 1 root root  5383 2012-08-06 04:27 README
drwxr-xr-x 7 root root  4096 2012-08-08 10:58 scripts
drwxr-xr-x 5 root root  4096 2012-08-08 10:58 src
drwxr-xr-x 4 root root  4096 2012-08-08 10:58 test
-rw-r--r-- 1 root root     3 2012-08-06 04:27 VERSION
root@bt:/pentest/wireless/aircrack-ng#
```

You can see that when I now check file permissions by typing **ls -l**, that the ChangeLog file no longer has write permission for the group.

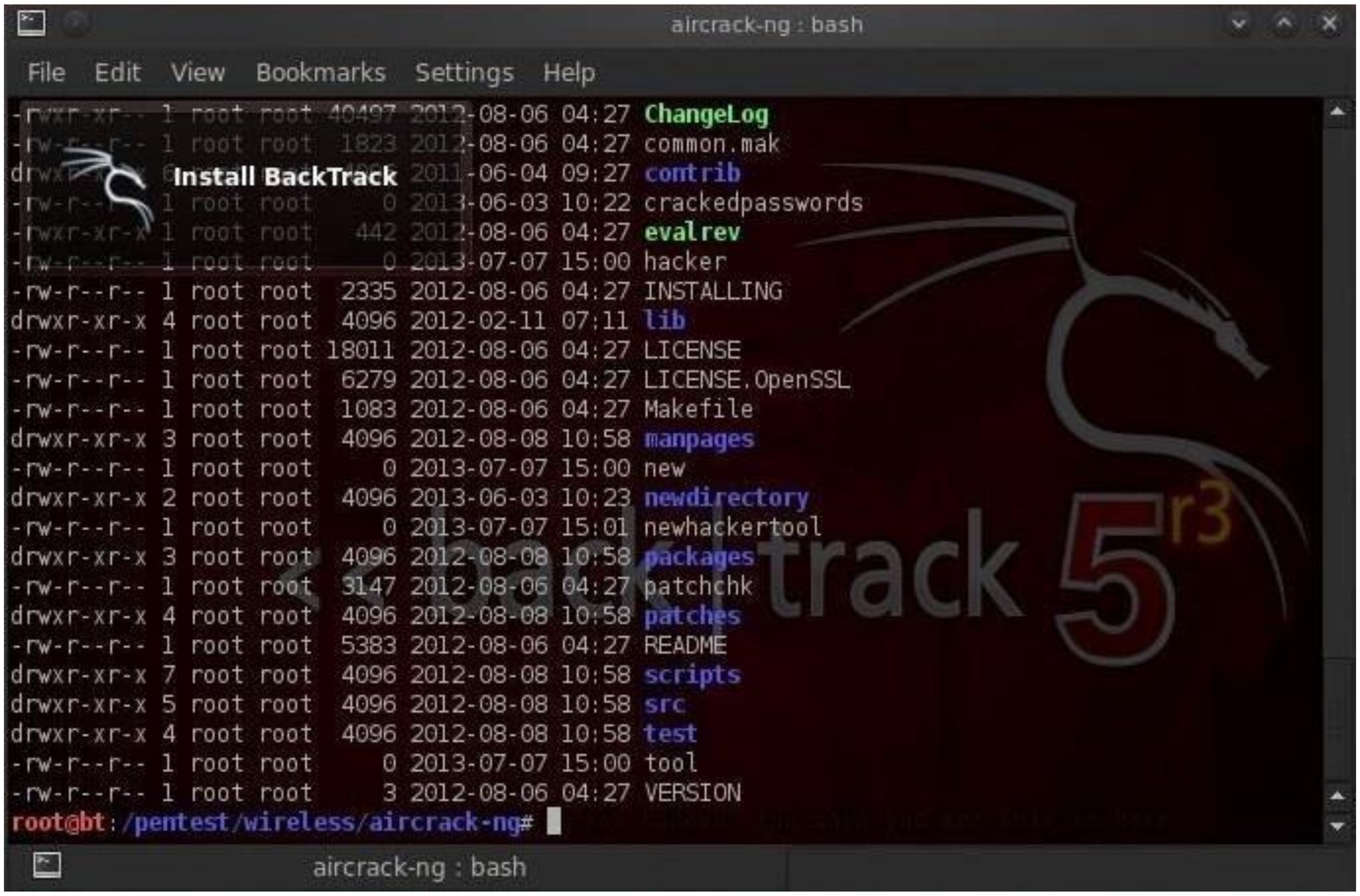
If I wanted to give both the user and group execute permission, I could type:

- **chmod u+x, g+x ChangeLog**

This command says "**for the user add the execute permission, for the group add the execute permission to the file ChangeLog**."

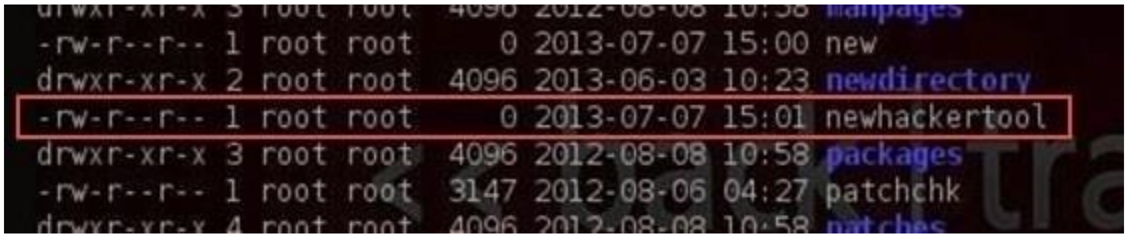
Step 4 Giving Ourselves Execute Permission on a New Hacking Tool

Very often as a hacker, we'll need to download new hacking tools. After we download, extract, unzip, make, and install them, we'll very often need to give ourselves permission to execute it. If we don't, we will usually get a message that we don't have adequate permission to execute.



```
-rwxr-xr-- 1 root root 40497 2012-08-06 04:27 ChangeLog
-rw-r--r-- 1 root root  1823 2012-08-06 04:27 common.mak
drwx     6               2011-06-04 09:27 contrib
-rw-r--  1 root root     0 2013-06-03 10:22 crackedpasswords
-rwxr-xr-x 1 root root   442 2012-08-06 04:27 evalrev
-rw-r--r-- 1 root root     0 2013-07-07 15:00 hacker
-rw-r--r-- 1 root root  2335 2012-08-06 04:27 INSTALLING
drwxr-xr-x 4 root root  4096 2012-02-11 07:11 lib
-rw-r--r-- 1 root root 18011 2012-08-06 04:27 LICENSE
-rw-r--r-- 1 root root  6279 2012-08-06 04:27 LICENSE.OpenSSL
-rw-r--r-- 1 root root  1083 2012-08-06 04:27 Makefile
drwxr-xr-x 3 root root  4096 2012-08-08 10:58 manpages
-rw-r--r-- 1 root root     0 2013-07-07 15:00 new
drwxr-xr-x 2 root root  4096 2013-06-03 10:23 newdirectory
-rw-r--r-- 1 root root     0 2013-07-07 15:01 newhackertool
drwxr-xr-x 3 root root  4096 2012-08-08 10:58 packages
-rw-r--r-- 1 root root  3147 2012-08-06 04:27 patchchk
drwxr-xr-x 4 root root  4096 2012-08-08 10:58 patches
-rw-r--r-- 1 root root  5383 2012-08-06 04:27 README
drwxr-xr-x 7 root root  4096 2012-08-08 10:58 scripts
drwxr-xr-x 5 root root  4096 2012-08-08 10:58 src
drwxr-xr-x 4 root root  4096 2012-08-08 10:58 test
-rw-r--r-- 1 root root     0 2013-07-07 15:00 tool
-rw-r--r-- 1 root root     3 2012-08-06 04:27 VERSION
root@bt:/pentest/wireless/aircrack-ng# 
```

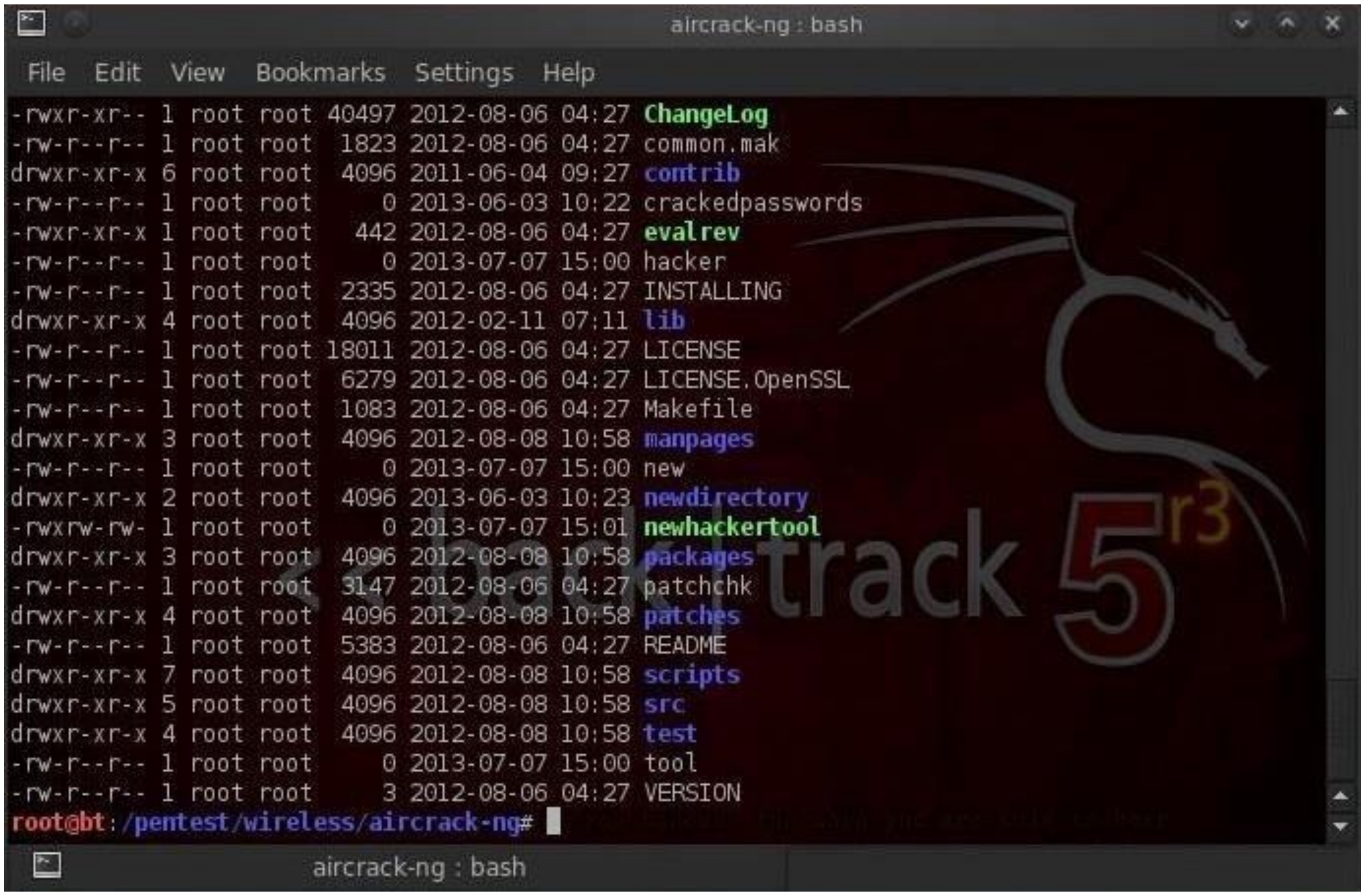
We can see in the screenshot above that our **newhackertool** does not have execute permission for anyone.



```
drwxr-xr-x 3 root root  4096 2012-08-08 10:58 manpages
-rw-r--r-- 1 root root     0 2013-07-07 15:00 new
drwxr-xr-x 2 root root  4096 2013-06-03 10:23 newdirectory
-rw-r--r-- 1 root root     0 2013-07-07 15:01 newhackertool
drwxr-xr-x 3 root root  4096 2012-08-08 10:58 packages
-rw-r--r-- 1 root root  3147 2012-08-06 04:27 patchchk
drwxr-xr-x 4 root root  4096 2012-08-08 10:58 patches
```

We can give ourselves permission to execute on a newhackertool by writing:

- **chmod 766 newhackertool**



```
-rwxr-xr-- 1 root root 40497 2012-08-06 04:27 ChangeLog
-rw-r--r-- 1 root root  1823 2012-08-06 04:27 common.mak
drwxr-xr-x 6 root root  4096 2011-06-04 09:27 contrib
-rw-r--r-- 1 root root     0 2013-06-03 10:22 crackedpasswords
-rwxr-xr-x 1 root root   442 2012-08-06 04:27 evalrev
-rw-r--r-- 1 root root     0 2013-07-07 15:00 hacker
-rw-r--r-- 1 root root  2335 2012-08-06 04:27 INSTALLING
drwxr-xr-x 4 root root  4096 2012-02-11 07:11 lib
-rw-r--r-- 1 root root 18011 2012-08-06 04:27 LICENSE
-rw-r--r-- 1 root root  6279 2012-08-06 04:27 LICENSE.OpenSSL
-rw-r--r-- 1 root root  1083 2012-08-06 04:27 Makefile
drwxr-xr-x 3 root root  4096 2012-08-08 10:58 manpages
-rw-r--r-- 1 root root     0 2013-07-07 15:00 new
drwxr-xr-x 2 root root  4096 2013-06-03 10:23 newdirectory
-rwxrw-rw- 1 root root     0 2013-07-07 15:01 newhackertool
drwxr-xr-x 3 root root  4096 2012-08-08 10:58 packages
-rw-r--r-- 1 root root  3147 2012-08-06 04:27 patchchk
drwxr-xr-x 4 root root  4096 2012-08-08 10:58 patches
-rw-r--r-- 1 root root  5383 2012-08-06 04:27 README
drwxr-xr-x 7 root root  4096 2012-08-08 10:58 scripts
drwxr-xr-x 5 root root  4096 2012-08-08 10:58 src
drwxr-xr-x 4 root root  4096 2012-08-08 10:58 test
-rw-r--r-- 1 root root     0 2013-07-07 15:00 tool
-rw-r--r-- 1 root root     3 2012-08-06 04:27 VERSION
root@bt:/pentest/wireless/aircrack-ng#
```

As you now know, this would give us, the owner, all permissions including execute, and the group and everyone else just read and write permissions (**4+2=6**). You can see in the screenshot above that after running the **chmod** command, that's exactly what we get!

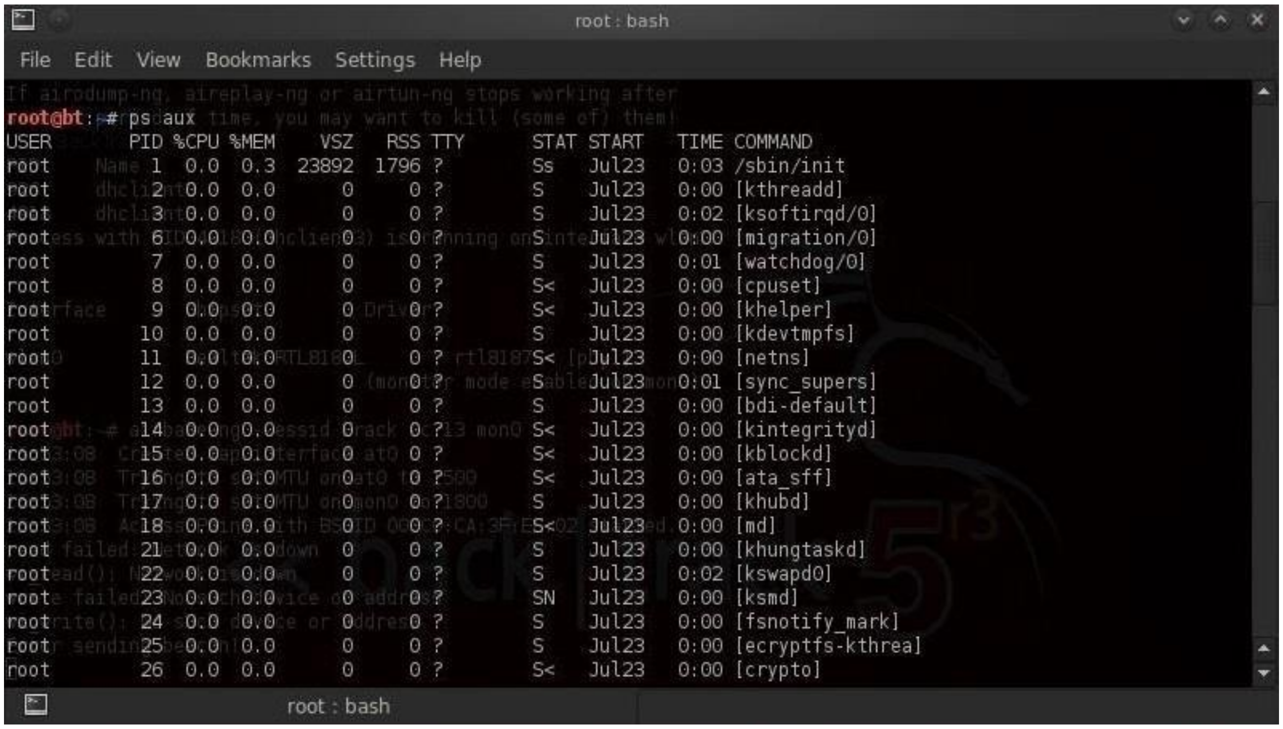
Managing Processes

In Linux, a process is a program running in memory. Typically, your computer is running hundreds of processes simultaneously. If they're system processes, Linux folks refer to them as **daemons** or **demons**. You will often see the process name ending with a "**d**" such **httpd**, the process or daemon responsible for the http service.

Step 1 See What Processes Are Running

We can see all the processes running on your system by typing:

- **ps aux**



```
root@bt:~# ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.3  23892  1796 ?        Ss   Jul23   0:03 /sbin/init
root         2  0.0  0.0      0     0 ?        S    Jul23   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S    Jul23   0:02 [ksoftirqd/0]
root         6  0.0  0.0      0     0 ?        S    Jul23   0:00 [migration/0]
root         7  0.0  0.0      0     0 ?        S    Jul23   0:01 [watchdog/0]
root         8  0.0  0.0      0     0 ?        S<   Jul23   0:00 [cpuset]
root         9  0.0  0.0      0     0 ?        S<   Jul23   0:00 [khelper]
root        10  0.0  0.0      0     0 ?        S    Jul23   0:00 [kdevtmpfs]
root        11  0.0  0.0      0     0 ?        S<   Jul23   0:00 [netns]
root        12  0.0  0.0      0     0 ?        S    Jul23   0:01 [sync_supers]
root        13  0.0  0.0      0     0 ?        S    Jul23   0:00 [bdi-default]
root        14  0.0  0.0      0     0 ?        S<   Jul23   0:00 [kintegrityd]
root        15  0.0  0.0      0     0 ?        S<   Jul23   0:00 [kblockd]
root        16  0.0  0.0      0     0 ?        S<   Jul23   0:00 [ata_sff]
root        17  0.0  0.0      0     0 ?        S    Jul23   0:00 [khubd]
root        18  0.0  0.0      0     0 ?        S<   Jul23   0:00 [md]
root        21  0.0  0.0      0     0 ?        S    Jul23   0:00 [khungtaskd]
root        22  0.0  0.0      0     0 ?        S    Jul23   0:02 [kswapd0]
root        23  0.0  0.0      0     0 ?        SN   Jul23   0:00 [ksmd]
root        24  0.0  0.0      0     0 ?        S    Jul23   0:00 [fsnotify_mark]
root        25  0.0  0.0      0     0 ?        S    Jul23   0:00 [ecryptfs-kthrea]
root        26  0.0  0.0      0     0 ?        S<   Jul23   0:00 [crypto]
```

These switches will provide all processes (**a**), the user (**u**) ,and processes not associated with a terminal (**x**). This is my favorite set of switches for using **ps** as it enables me to see which user initiated the process and how much in resources it's using.

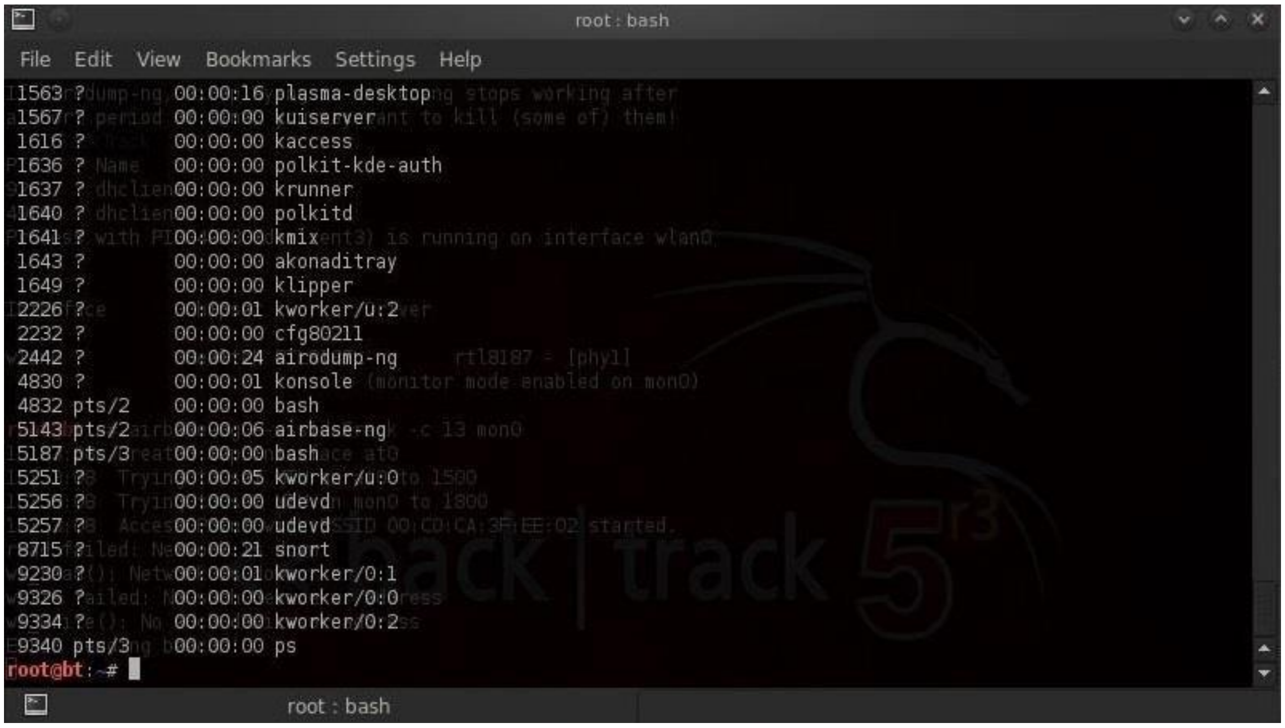
Note that each process listed shows us among many things.

- user
- PID (process identifier)
- %CPU

- %MEM (memory)

If we just wanted to see the all the processes with limited information, we can type:

- **ps -A**



```
1563 ?        00:00:16 plasma-desktop
1567 ?        00:00:00 kuiserver
1616 ?        00:00:00 kaccess
1636 ?        00:00:00 polkit-kde-auth
1637 ?        00:00:00 krunner
1640 ?        00:00:00 polkitd
1641 ?        00:00:00 kmix
1643 ?        00:00:00 akonaditray
1649 ?        00:00:00 klipper
2226 ?        00:00:01 kworker/u:2
2232 ?        00:00:00 cfg80211
2442 ?        00:00:24 airodump-ng
4830 ?        00:00:01 konsole
4832 pts/2    00:00:00 bash
5143 pts/2    00:00:06 airbase-ng
5187 pts/3    00:00:00 bash
5251 ?        00:00:05 kworker/u:0
5256 ?        00:00:00 udevd
5257 ?        00:00:00 udevd
8715 ?        00:00:21 snort
9230 ?        00:00:01 kworker/0:1
9326 ?        00:00:00 kworker/0:0
9334 ?        00:00:00 kworker/0:2
9340 pts/3    00:00:00 ps
root@bt:~#
```

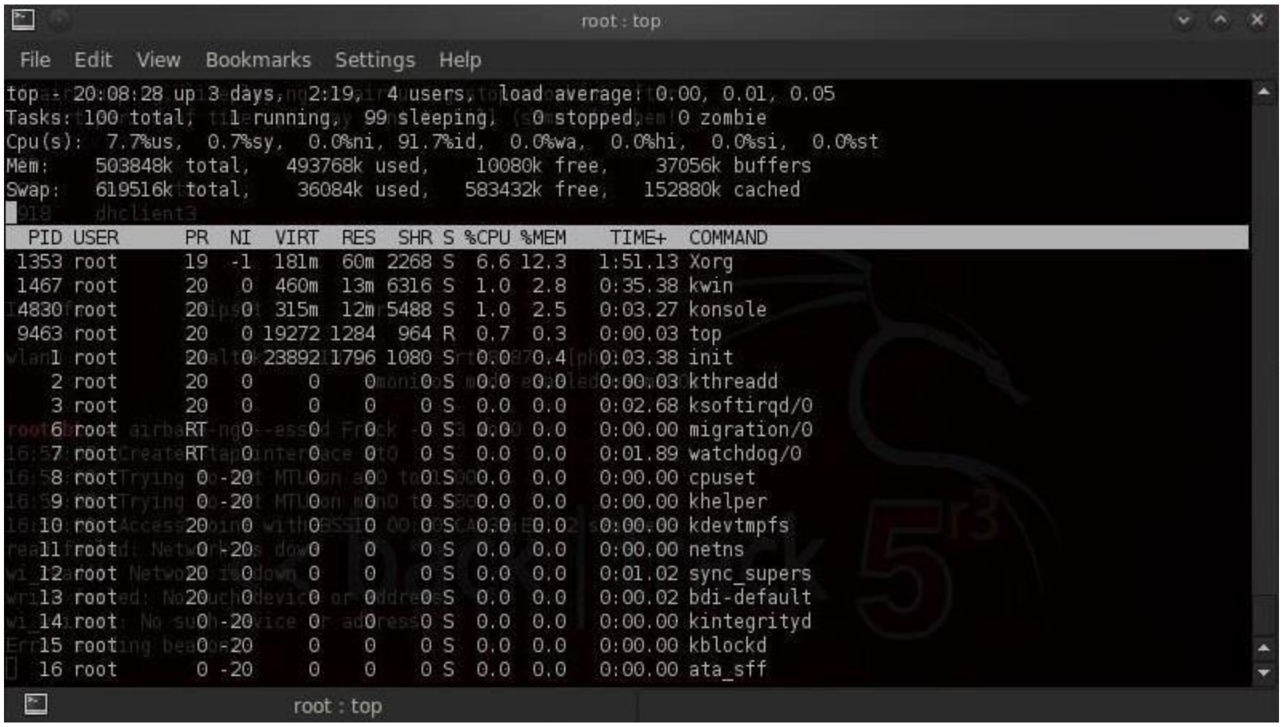
You can see all the processes running, but without such information as CPU percentage and memory percentage. Note that airbase-ng is listed with **PID 5143** and the last process is the **ps** command.

Process numbers, or PIDs, are critical for working in Linux, as you often need the PID to manage a process. As you might have seen in some of my Metasploit tutorials, the PID often becomes critical in hacking the victim systems.

Step 2 The Top Command

Similar to the **ps** command is the **top** command, except that top shows us only the top processes. In other words, it only shows us the processes using the most resources and it's dynamic, meaning that it is gives us a real-time look at our processes. Simply type:

- **top**



```
top - 20:08:28 up 3 days,  2:19,  4 users,  load average: 0.00, 0.01, 0.05
Tasks: 100 total,   1 running,  99 sleeping,   0 stopped,   0 zombie
Cpu(s):  7.7%us,  0.7%sy,  0.0%ni, 91.7%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:    503848k total,   493768k used,    10080k free,    37056k buffers
Swap:   619516k total,    36084k used,   583432k free,   152880k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 1353 root      19  -1  181m  60m 2268 S  6.6 12.3   1:51.13 Xorg
 1467 root      20   0  460m  13m 6316 S  1.0  2.8   0:35.38 kwin
 4830 root      20   0  315m  12m 5488 S  1.0  2.5   0:03.27 konsole
 9463 root      20   0 19272 1284  964 R  0.7  0.3   0:00.03 top
    1 root      20   0 23892 1796 1080 S  0.0  0.4   0:03.38 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.03 kthreadd
    3 root      20   0     0    0    0 S  0.0  0.0   0:02.68 ksoftirqd/0
    6 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/0
    7 root      RT   0     0    0    0 S  0.0  0.0   0:01.89 watchdog/0
    8 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 cpuset
    9 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 khelper
   10 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kdevtmpfs
   11 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 netns
   12 root      20   0     0    0    0 S  0.0  0.0   0:01.02 sync_supers
   13 root      20   0     0    0    0 S  0.0  0.0   0:00.02 bdi-default
   14 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 kintegrityd
   15 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 kblockd
   16 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 ata_sff
```

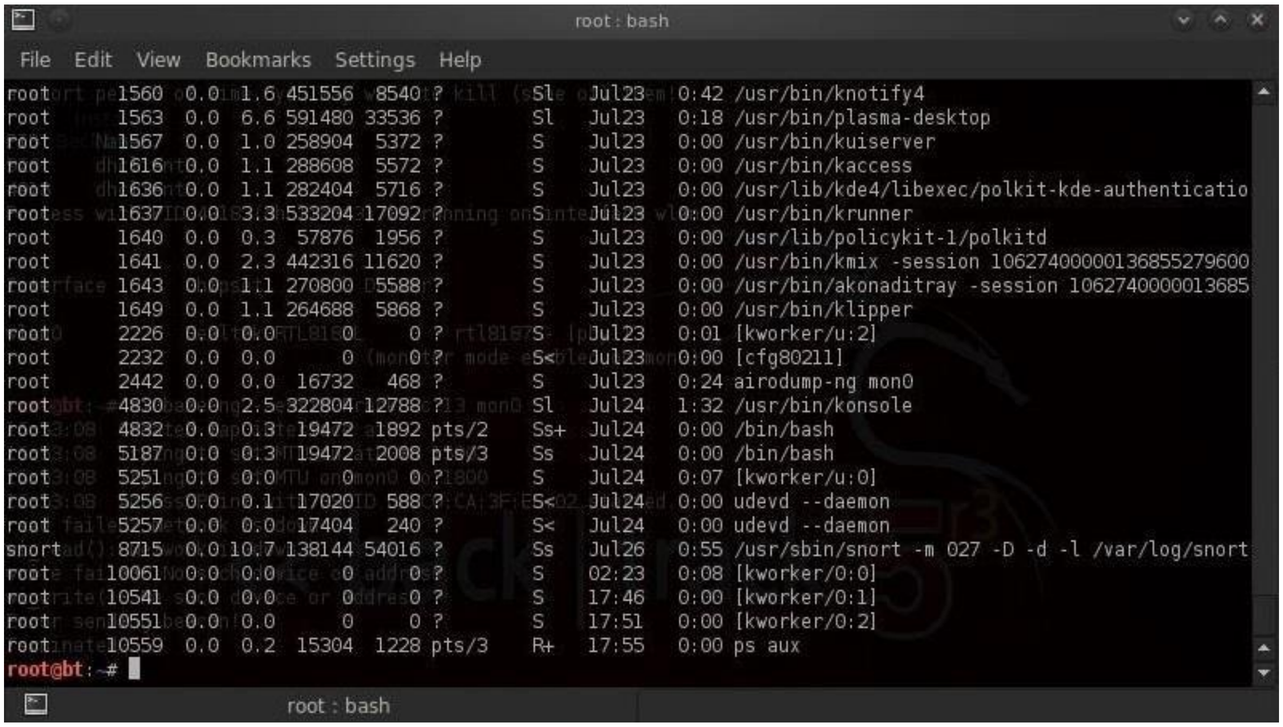
As you can see, the processes are listed in the order by how much system resources they are using, and the list is constantly changing as the processes use more or less resources.

Step 3 Killing Processes

Sometimes we will need to stop processes in Linux. The command we use is **kill**. Don't worry, it sounds more violent than it actually is. This command is particularly important if we have a process that continues to run and use system resources, even after we have tried to stop it. These processes are often referred to as "zombie" processes.

We can kill a process by simply typing kill and the process ID or PID. So to kill my airbase-ng process, I can simply type:

- **kill 5143**



```
root      1560  0.0  1.6 451556  8540 ?        Sl   Jul23   0:42 /usr/bin/knotify4
root      1563  0.0  6.6 591480 33536 ?        Sl   Jul23   0:18 /usr/bin/plasma-desktop
root      1567  0.0  1.0 258904  5372 ?        S    Jul23   0:00 /usr/bin/kuiserver
root      1616  0.0  1.1 288608  5572 ?        S    Jul23   0:00 /usr/bin/kaccess
root      1636  0.0  1.1 282404  5716 ?        S    Jul23   0:00 /usr/lib/kde4/libexec/polkit-kde-authenticatio
root      1637  0.0  3.3 533204 17092 ?        S    Jul23   0:00 /usr/bin/krunner
root      1640  0.0  0.3  57876  1956 ?        S    Jul23   0:00 /usr/lib/policykit-1/polkitd
root      1641  0.0  2.3 442316 11620 ?        S    Jul23   0:00 /usr/bin/kmix -session 1062740000013685527960
root      1643  0.0  1.1 270800  5588 ?        S    Jul23   0:00 /usr/bin/akonaditray -session 106274000001368
root      1649  0.0  1.1 264688  5868 ?        S    Jul23   0:00 /usr/bin/klipper
root      2226  0.0  0.0      0     0 ?        S    Jul23   0:01 [kworker/u:2]
root      2232  0.0  0.0      0     0 ?        S<   Jul23   0:00 [cfg80211]
root      2442  0.0  0.0  16732   468 ?        S    Jul23   0:24 airodump-ng mon0
root      4830  0.0  2.5 322804 12788 ?        Sl   Jul24   1:32 /usr/bin/konsole
root      4832  0.0  0.3  19472  1892 pts/2    Ss+  Jul24   0:00 /bin/bash
root      5187  0.0  0.3  19472  2008 pts/3    Ss   Jul24   0:00 /bin/bash
root      5251  0.0  0.0      0     0 ?        S    Jul24   0:07 [kworker/u:0]
root      5256  0.0  0.1  17020   588 ?        S<   Jul24   0:00 udevd --daemon
root      5257  0.0  0.0  17404   240 ?        S<   Jul24   0:00 udevd --daemon
snort     8715  0.0 10.7 138144 54016 ?        Ss   Jul26   0:55 /usr/sbin/snort -m 027 -D -d -l /var/log/snort
root     10061  0.0  0.0      0     0 ?        S    02:23   0:08 [kworker/0:0]
root     10541  0.0  0.0      0     0 ?        S    17:46   0:00 [kworker/0:1]
root     10551  0.0  0.0      0     0 ?        S    17:51   0:00 [kworker/0:2]
root     10559  0.0  0.2  15304  1228 pts/3    R+   17:55   0:00 ps aux
root@bt:~#
```

We can see in the screenshot above that my airbase-ng process is no longer running.

There are many types of "kills". The default kill (when we use the kill command without any switches) is **kill 15** or the termination signal. It allows the process to cleanup and gently terminate its process.

Sometimes, processes still refuse to terminate even when sent the default kill command. In that case, we have to get more serious and use the absolute terminator to do the job. This is **kill -9**, which takes no prisoners and ends the job without allowing it to say its goodbyes and forces the kernel to terminate it immediately.

Step 4 Change Process Priority

Every process in Linux is given a priority number. As you probably guessed, this priority number determines how important the process is and where it stands in line in terms of using system resources. These priority numbers range from 0 to 127 with 0 being the highest priority and 127 being the lowest.

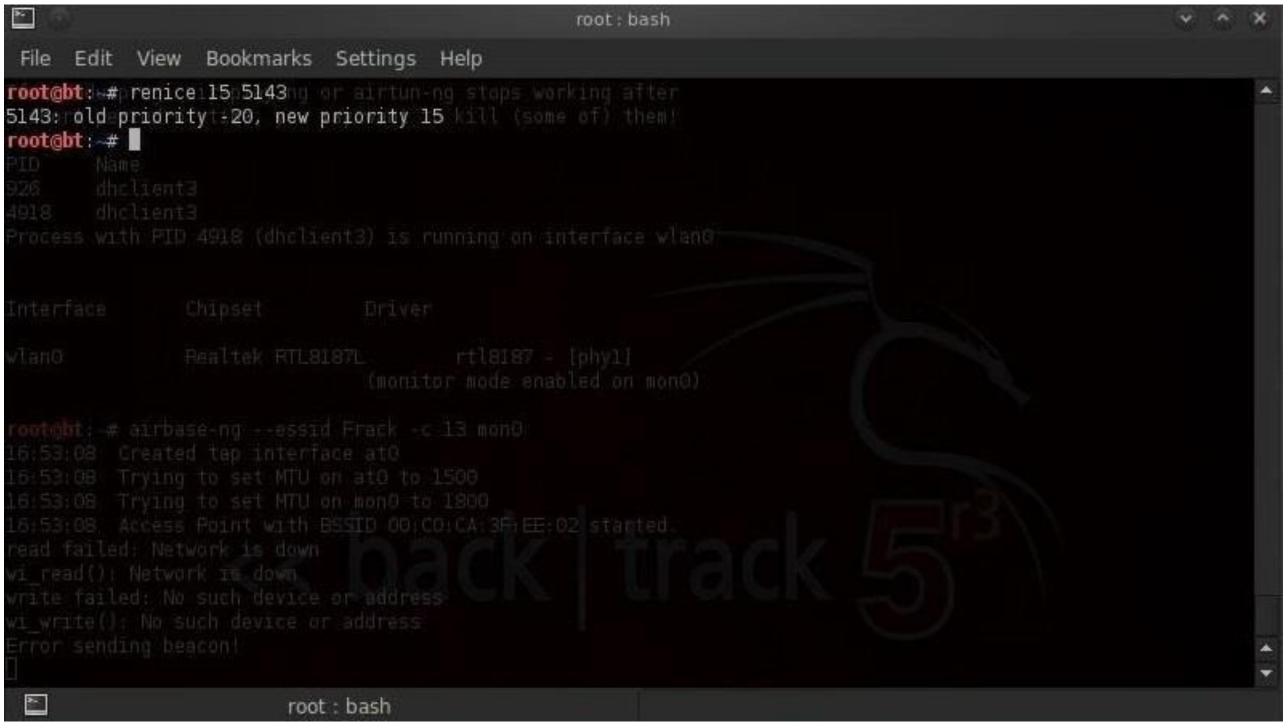
As the root user or system admin, we can't directly determine the priority of a process—that is the job of the kernel—but we can hint to the kernel that we would like a process to run with a higher priority. We can do this through the **nice** command. Nice values range from **-20** to **+19** with the lower values indicating a higher priority.

We can set a processes' nice value by using the **nice** command, the **-n** switch, the **value** of the nice, and then the **command** we want to run. So, if if we wanted to start our airbase-ng process from our Evil Twin tutorial with the highest priority, we could type:

- **nice -n -20 airbase-ng -a 00:09:5B:6F:64:1E --essid "Elroy" -c 11 mon0**

Later on, if we felt that we wanted to reduce the priority of the airbase-ng command, we could **renice** it. The renice command requires simply the **renice** command, the **priority level**, and unlike the nice command, it only takes the process **PID**, such as:

- **renice 15 5143**



```
root@bt:~# renice 15 5143
5143: old priority -20, new priority 15
root@bt:~#
```

We can see that by renice-ing the airbase-ng command, we have reduced its priority from -20 (highest) to 15 (relatively low).

Step 5 Push a Process into the Background

When we run a command from the shell terminal, the process will take control of that shell until it is complete. If it's an ongoing process, similar to airbase-ng, it will maintain control of that terminal until we stop it. Until that time, we can't use that shell.

If we want to still use that shell, we can send that process into the background and then get control of the shell again. To start a command in the background, we simply need to end the command with the **&** or ampersand. So, to get airbase-ng to run in the background, we simply type:

- **airbase-ng -a 00:09:5B:6F:64:1E --essid "Elroy" -c 11 mon0 &**

If we want to bring a background job to the foreground, we simply type **fg**. To send a foreground processes to the background, we can type **Control Z** to stop it and then and using the **bg** command with the PID to send it to the background.

Managing Environmental Variables

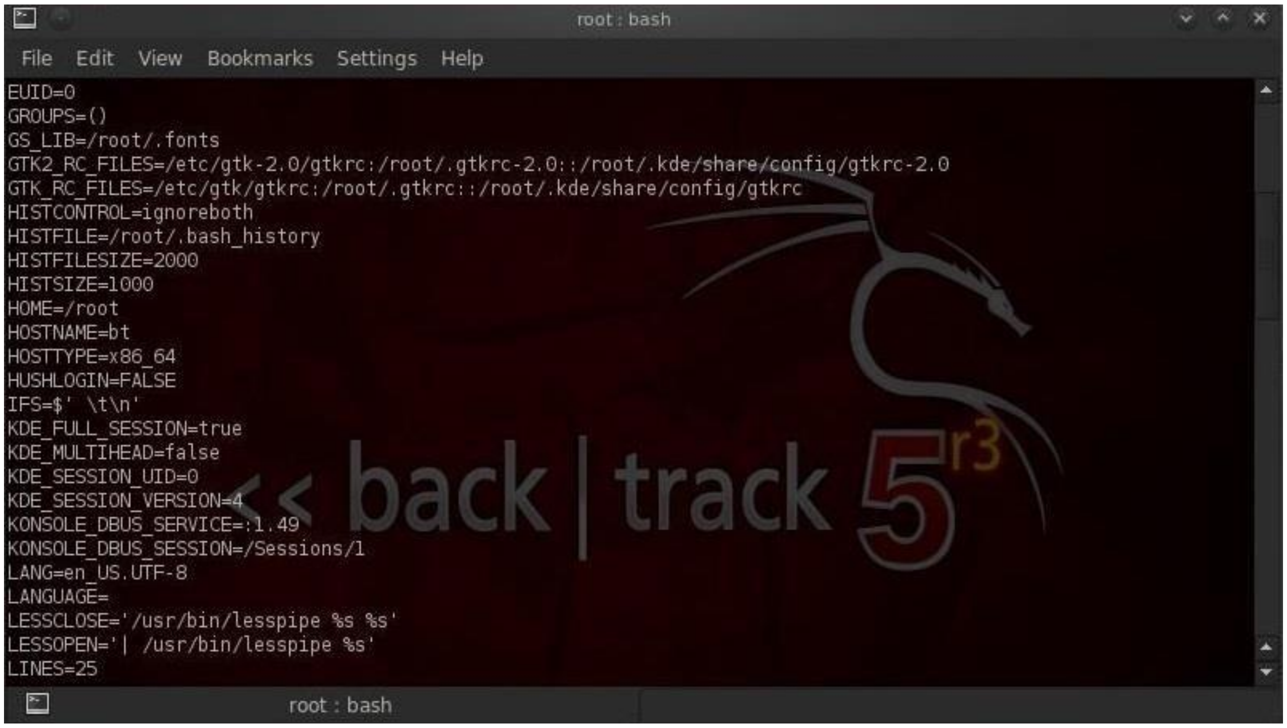
One of the areas that often gives Linux newcomers problems are the environment variables. Although Windows systems have environment variables, most users, and for that matter, most administrators, never manage their environment variables.

Environment variables are the variables that are used by our particular user environment. In most cases, this will be our BASH shell. Each user, including root, has a set of environment variables that are set at default values unless they're changed. We can change these values to make our system work more efficiently and tailor our work environment to best meet our individual needs.

Step 1 View Our Environment Variables

We can view our environment variables by typing:

- **set**

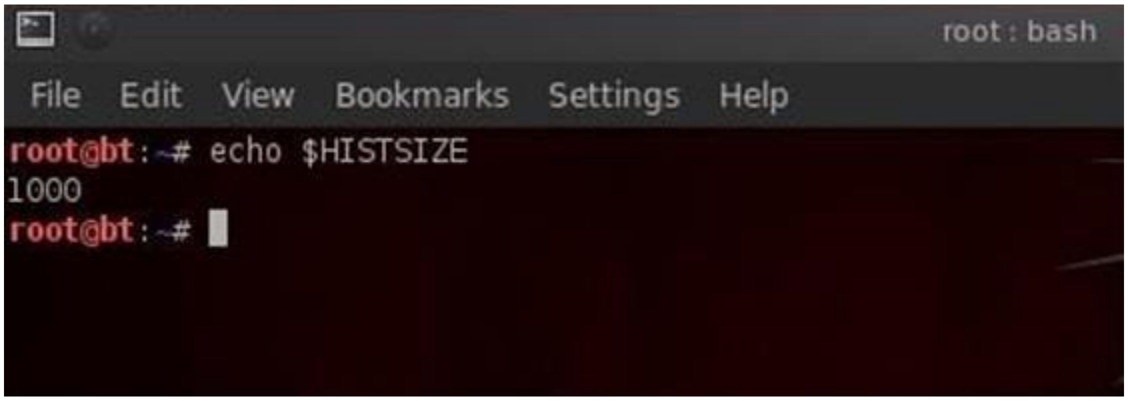


```
EUID=0
GROUPS=()
GS_LIB=/root/.fonts
GTK2_RC_FILES=/etc/gtk-2.0/gtkrc:/root/.gtkrc-2.0::/root/.kde/share/config/gtkrc-2.0
GTK_RC_FILES=/etc/gtk/gtkrc:/root/.gtkrc::/root/.kde/share/config/gtkrc
HISTCONTROL=ignoreboth
HISTFILE=/root/.bash_history
HISTFILESIZE=2000
HISTSIZE=1000
HOME=/root
HOSTNAME=bt
HOSTTYPE=x86_64
HUSHLOGIN=FALSE
IFS=$' \t\n'
KDE_FULL_SESSION=true
KDE_MULTIHEAD=false
KDE_SESSION_UID=0
KDE_SESSION_VERSION=4
KONSOLE_DBUS_SERVICE=:1.49
KONSOLE_DBUS_SESSION=/Sessions/1
LANG=en_US.UTF-8
LANGUAGE=
LESSCLOSE='/usr/bin/lesspipe %s %s'
LESSOPEN='| /usr/bin/lesspipe %s'
LINES=25
```

Notice that **set** lists for us all of the environment variables, user defined functions, and aliases. Also, make note that our environment variables are always UPPER CASE names such as **HOME**, **PATH**, **HISTSIZE**, etc.

If want to see the value inside the variable, we can type:

- **echo $HISTSIZE**



```
root@bt:~# echo $HISTSIZE
1000
root@bt:~# 
```

It's important to notice that when we want to use the value inside the variable, such as here, we need to put a **$** before the variable name.

The **HISTSIZE** variable contains the value of the number of commands that are stored in our history file. As you can see in this screenshot, the HISTSIZE variable is set to 1000. In some cases, we may not want our commands stored in the history file, such as **when we are covering our tracks**, then we can set our HISTSIZE variable to zero.

- **HISTSIZE=0**

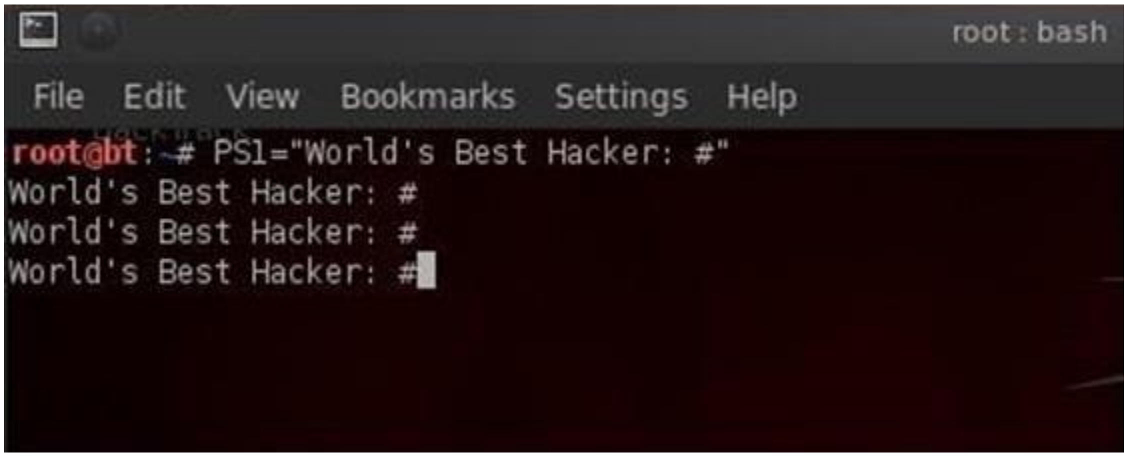
When we change an environment variable, it's only for that environment. This means that once we close that terminal, any changes that we made to these variables is lost or set back to the default value. If we want the value to remain for our next terminal session and other terminal sessions, we need to export the variable. We can do this by simply typing:

- **export HISTSIZE**

Step 2 Changing Our Terminal Prompt

Let's have a little fun and change the prompt in our terminal. The environment variable that contains our prompt for the first terminal is **PS1**. We can change it by typing:

- **PS1= "World's Best Hacker: #"**



```
root@bt:~# PS1="World's Best Hacker: #"
World's Best Hacker: #
World's Best Hacker: #
World's Best Hacker: #
```

Remember that our prompt will now be "World's Best Hacker" whenever we open the first terminal (PS1), but the second terminal will still be the default BackTrack command prompt. In addition, if we really like this command prompt and want to keep it, we need to export it so that each time we open this terminal, the prompt will be "World's Best Hacker."

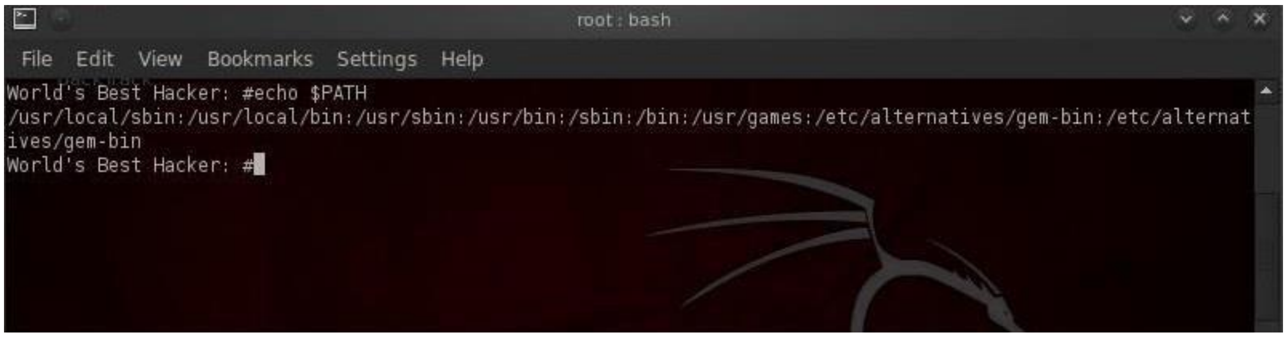
- **export PS1**

Step 3 Changing Our Path Variable

Probably the most important variable in our environment is our **PATH** variable. This is what controls where our shell looks for the commands we type, such as cd, ls, echo, etc. If it doesn't find the command in one of the directories in our path, it returns an error "command not found," even if it DOES exist in another directory not in our PATH.

Let's take a look at our path variable:

- **echo =$PATH**



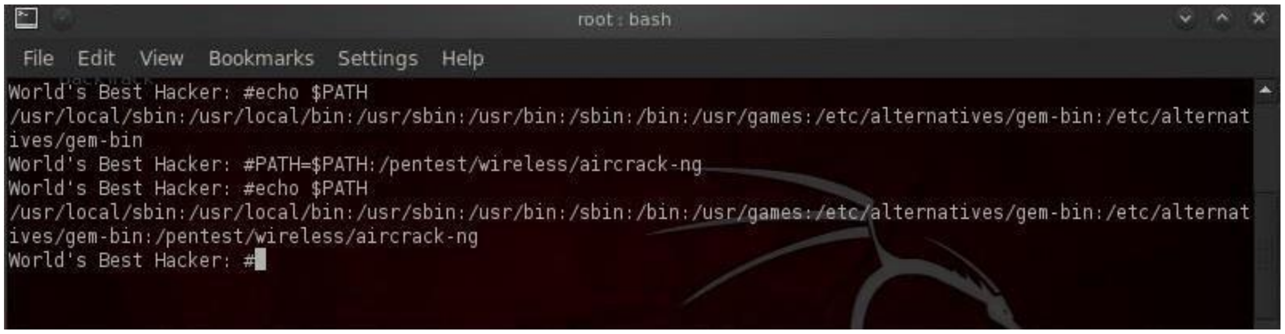
```
World's Best Hacker: #echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/etc/alternatives/gem-bin:/etc/alternat
ives/gem-bin
World's Best Hacker: #
```

Notice the directories included in our PATH. These are usually the various **/bin** and **/sbin** directories where our system variables are found. When we type **ls**, the system knows to look in each of these directories for the **ls** command.

Whenever we want to use aircrack-ng or another hacking application in this PATH variable, we have to first navigate to that directory. In the case of aircrack-ng, that would be /pentest/wireless/aircrack-ng.

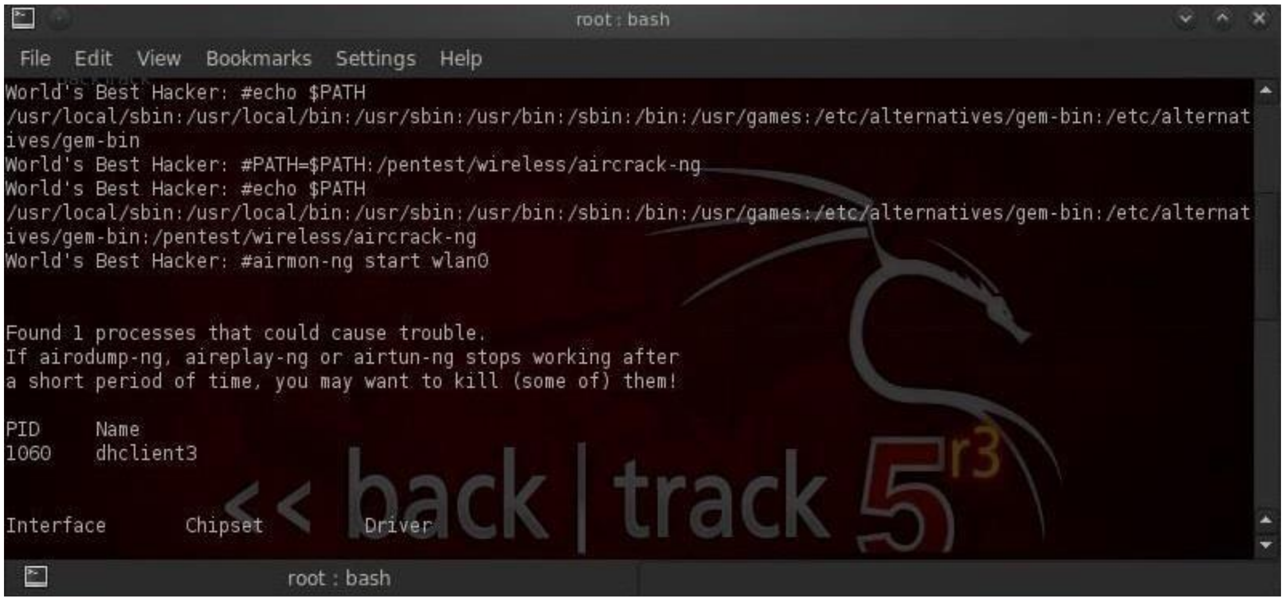
Now, if we want to add our wireless hacking application to our PATH variable, we can type:

- **PATH=$PATH:/pentest/wireless/aircrack-ng**



```
World's Best Hacker: #echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/etc/alternatives/gem-bin:/etc/alternat
ives/gem-bin
World's Best Hacker: #PATH=$PATH:/pentest/wireless/aircrack-ng
World's Best Hacker: #echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/etc/alternatives/gem-bin:/etc/alternat
ives/gem-bin:/pentest/wireless/aircrack-ng
World's Best Hacker: #
```

Now when we want to run aircrack-ng, we don't need to navigate to the **pentest/wireless/aircrack-ng** directory. We now can execute aircrack-ng applications from anywhere in BackTrack!



```
World's Best Hacker: #echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/etc/alternatives/gem-bin:/etc/alternat
ives/gem-bin
World's Best Hacker: #PATH=$PATH:/pentest/wireless/aircrack-ng
World's Best Hacker: #echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/etc/alternatives/gem-bin:/etc/alternat
ives/gem-bin:/pentest/wireless/aircrack-ng
World's Best Hacker: #airmon-ng start wlan0


Found 1 processes that could cause trouble.
If airodump-ng, aireplay-ng or airtun-ng stops working after
a short period of time, you may want to kill (some of) them!

PID     Name
1060    dhclient3


Interface       Chipset         Driver
```

This can be a very useful technique for directories that we use often, but be careful to not add too many directories to your PATH variable as the system will have to search through every directory in the PATH to find commands and could potentially slow down your terminal.

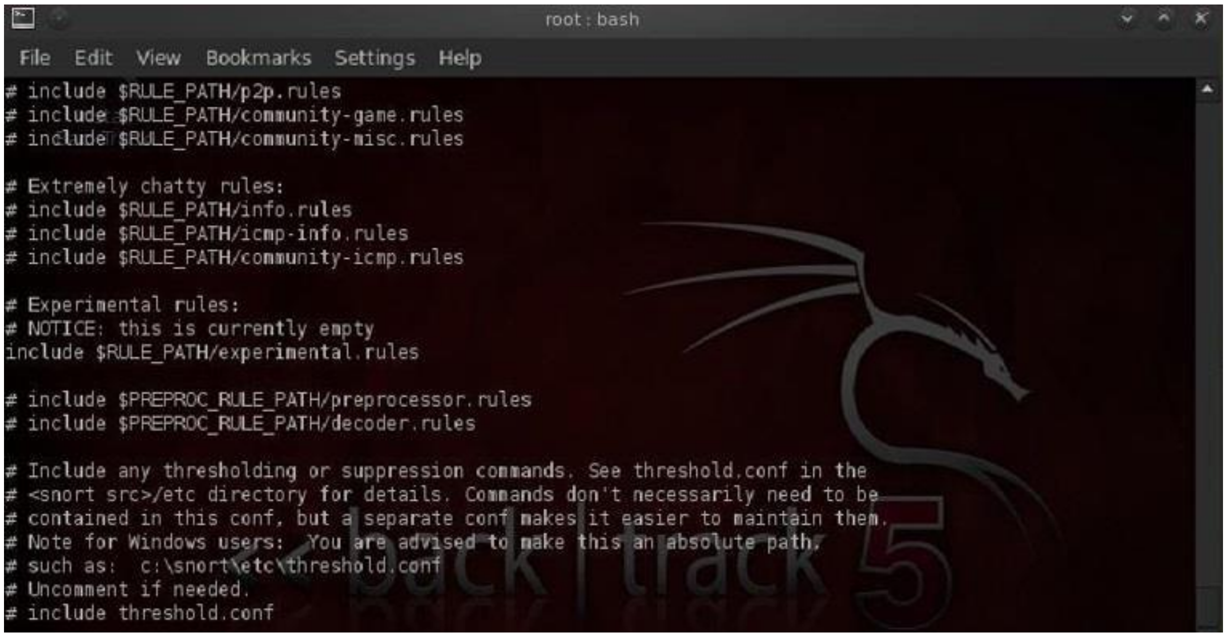
Manipulating Text

With so many text files, manipulating text becomes crucial in managing Linux and Linux applications. In this tutorial, we'll look at several of the commands and techniques for manipulating text in Linux. For demonstration purposes, we'll use files from the world's best NIDS, Snort.

Step 1 Cat That File

cat is probably the most basic text display command. Let's cat the Snort config file found in */etc/snort*.

- **cat /etc/snort/snort.conf**



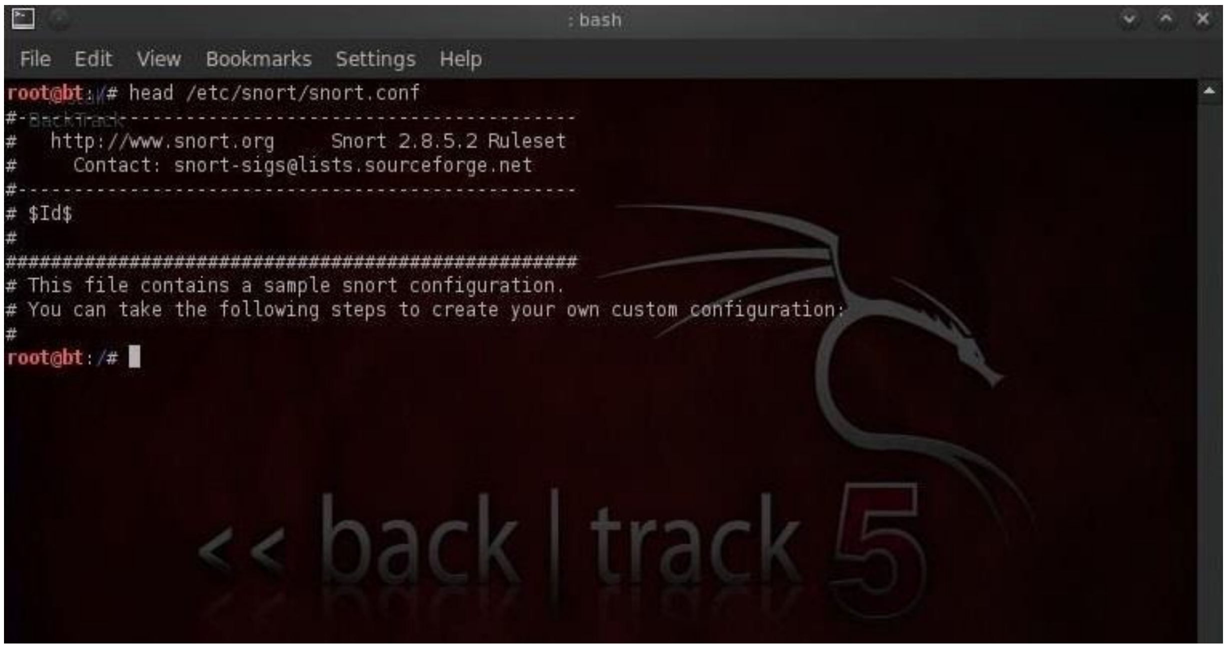
```
# include $RULE_PATH/p2p.rules
# include $RULE_PATH/community-game.rules
# include $RULE_PATH/community-misc.rules

# Extremely chatty rules:
# include $RULE_PATH/info.rules
# include $RULE_PATH/icmp-info.rules
# include $RULE_PATH/community-icmp.rules

# Experimental rules:
# NOTICE: this is currently empty
include $RULE_PATH/experimental.rules

# include $PREPROC_RULE_PATH/preprocessor.rules
# include $PREPROC_RULE_PATH/decoder.rules

# Include any thresholding or suppression commands. See threshold.conf in the
# <snort src>/etc directory for details. Commands don't necessarily need to be
# contained in this conf, but a separate conf makes it easier to maintain them.
# Note for Windows users:  You are advised to make this an absolute path,
# such as:  c:\snort\etc\threshold.conf
# Uncomment if needed.
# include threshold.conf
```

As you can see, the snort.conf is displayed on our screen until it comes to the end of the file. Not the most convenient way to work with this file.

Step 2 Take the Head

If we just want to view the beginning of a file, we can use the **head** command. This command displays the first 10 lines of a file, by default.

- **head /etc/snort/snort.conf**



```
root@bt:/# head /etc/snort/snort.conf
#--------------------------------------------------
#   http://www.snort.org     Snort 2.8.5.2 Ruleset
#     Contact: snort-sigs@lists.sourceforge.net
#--------------------------------------------------
# $Id$
#
###################################################
# This file contains a sample snort configuration.
# You can take the following steps to create your own custom configuration:
#
root@bt:/#
```

If we want to see more or less than the default 10 lines, we can tell head how many lines we want to see by putting the number of lines we want to see (with the **-** switch) between the command and the file name.

- **head -30 /etc/snort/snort.conf**

Apache Web Servers

One area that's critical that we haven't covered yet is building and managing an Apache web server.

Apache is found on over 60% of the globe's web servers, so any self-respecting Linux admin should be familiar with it. As a hacker aspiring to hack websites, it's critical to understand the inner workings of Apache, websites, and the backend databases of these sites.

In addition, by setting up your own web server, you could serve up malware to anyone who visits your site. If you're thinking of building a botnet, this is one of the best ways of doing that (I'll do a tutorial on building a botnet in the near future).

Getting Apache on Your System

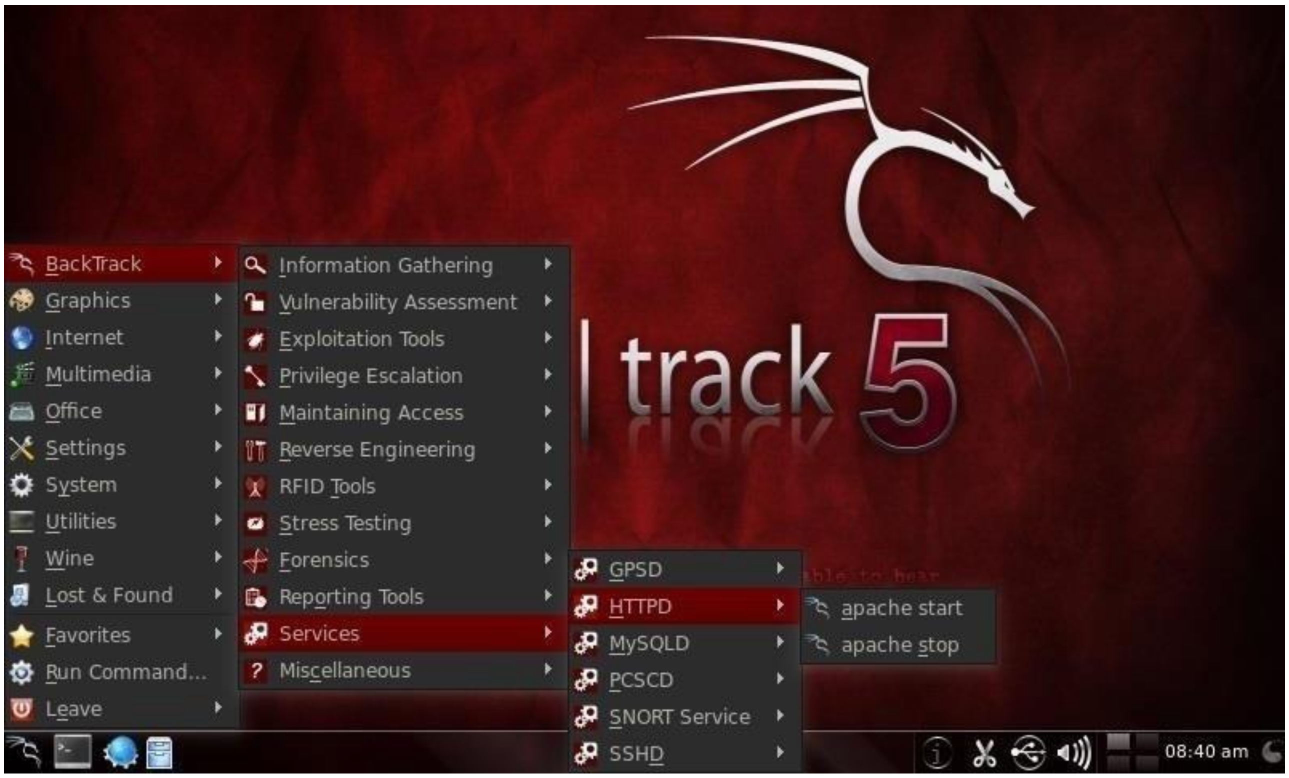
If you have **BackTrack** running on your system, Apache is already installed. Many other Linux distros have it installed by default as well. If you don't have Apache installed, you can download and install the **LAMP stack**.

LAMP is an acronym for Linux, Apache, MySQL, PERL, and PHP. These are the most widely used tools for developing websites in the Linux world, and they're very popular in the Microsoft world too, only it's generally referred to as WAMP, where the W simply stands for Windows.

Simply download this LAMP stack and install it on your system, and then I will take you through the care and feeding of your LAMP stack to serve up webpages. In addition, we'll download and install a website that we can use for web and database hacking in future tutorials.

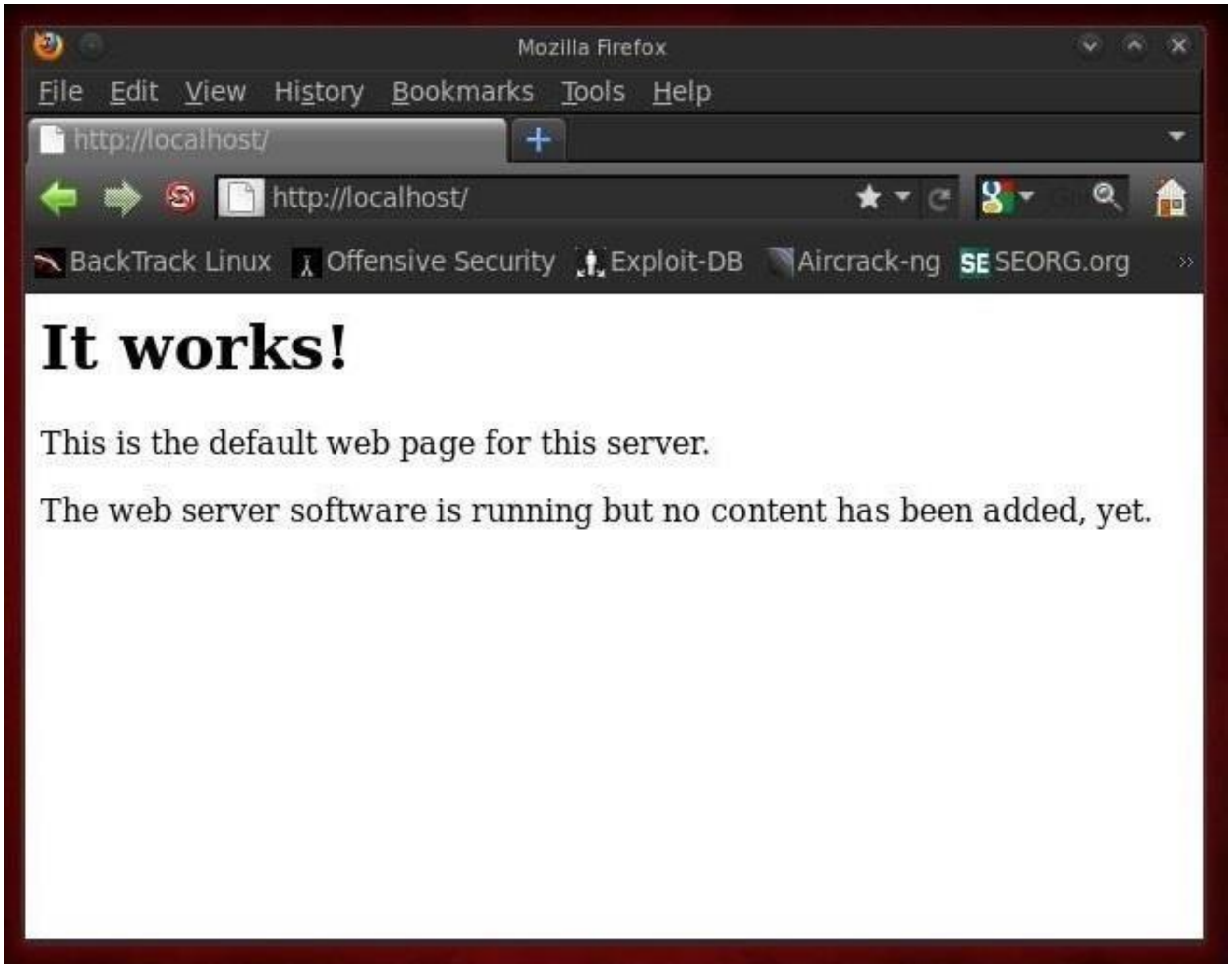
Step 1 Start Your Apache Daemon

The first step, of course, is to start our Apache daemon. In BackTrack, go the **BackTrack** -> **Services** -> **HTTPD** and click on **apache start**.



Step 2 Open the Default Website

Now that Apache is running, it should be able to serve up its default webpage. Let's type **http://localhost/** in your favorite web browser.



It works!

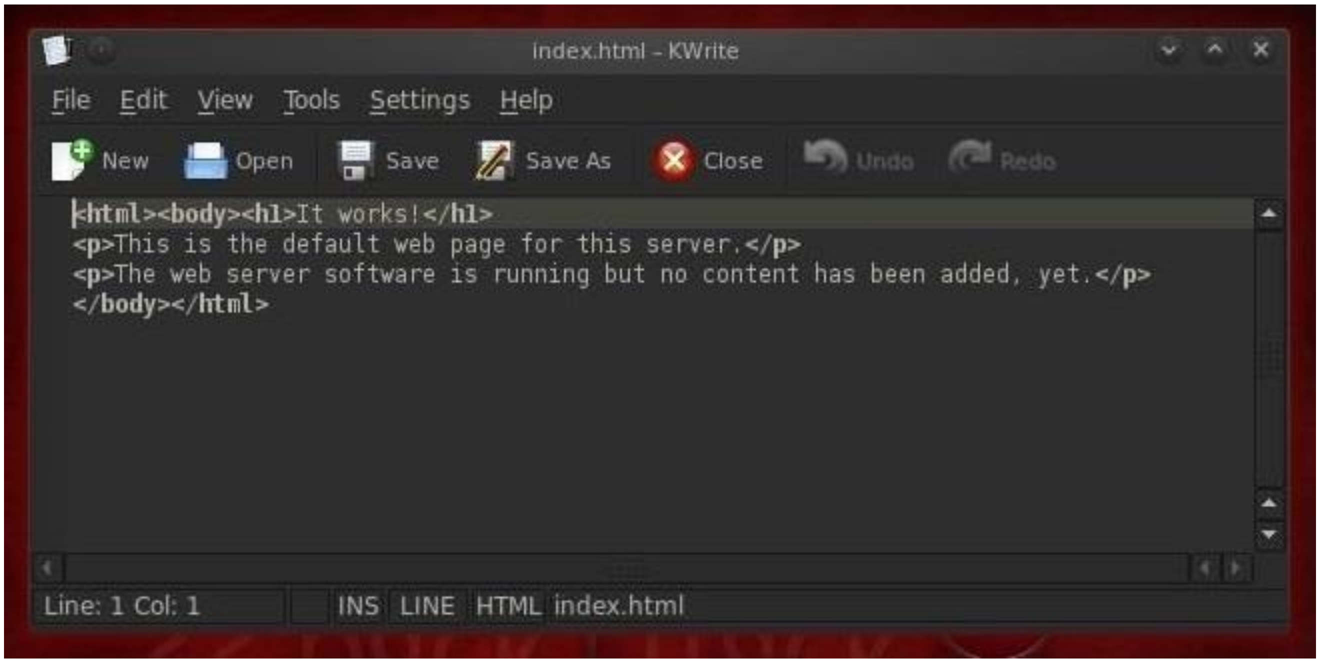
This is the default web page for this server.

The web server software is running but no content has been added, yet.

Step 3 Open the Index.html File

Apache's default webpage is **/var/www/index.html**. We can edit that file and get Apache to serve up whatever webpage we want, so let's create our own.

Use any text editor you please, including vi, gedit, Kate, KWrite, emacs, etc. For demonstration purposes here, I'll open the **/var/www/index.html** with KWrite.


index.html - KWrite
File Edit View Tools Settings Help
New Open Save Save As Close Undo Redo

```
<html><body><h1>It works!</h1>
<p>This is the default web page for this server.</p>
<p>The web server software is running but no content has been added, yet.</p>
</body></html>
```

Line: 1 Col: 1 INS LINE HTML index.html

Note here that the default webpage has exactly the text that was displayed when we opened our browser to **localhost**, but in **html** format. All we need to do is edit this file to have our web server display the information we want.

Step 4 Add Some Html

Now that we have the web server running and the index file open, we can add whatever text we'd like the web server to serve up. We will create some simple html blocks.

Let's serve up this page:

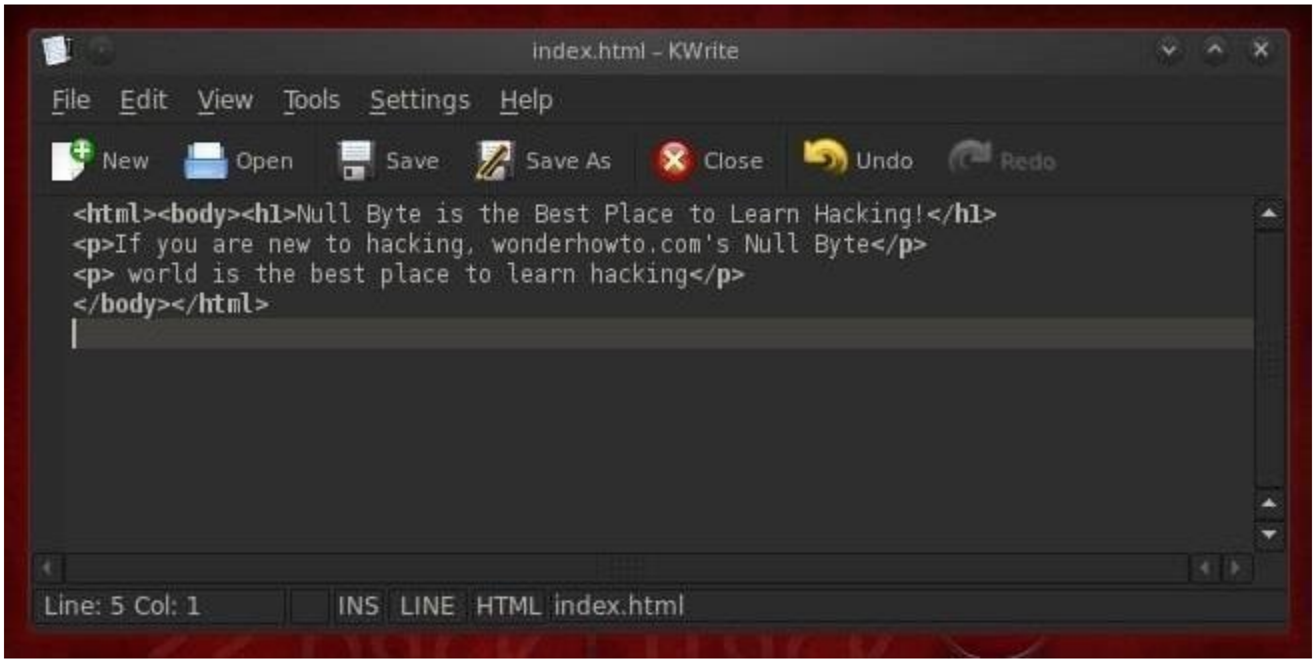
<html>
<body>

<h1> Null Byte is the Best! </h1>

<p> If you are new to hacking, wonderhowto.com's Null Byte</p>
<p>world is the best place to learn hacking!</p>

</body>
</html>

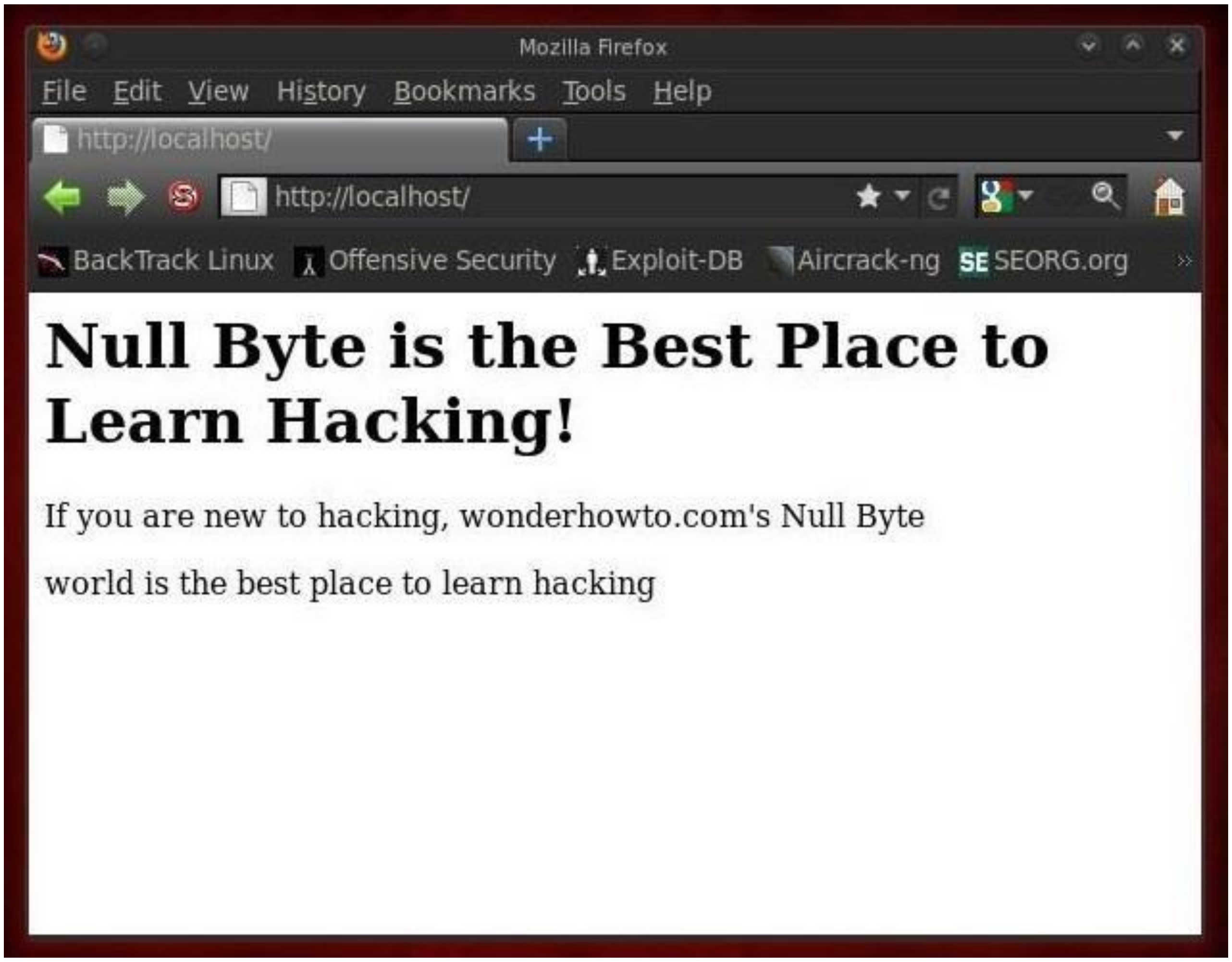
Now, save this file and close KWrite.


index.html – KWrite
File Edit View Tools Settings Help
New Open Save Save As Close Undo Redo
Line: 5 Col: 1 INS LINE HTML index.html

```
<html><body><h1>Null Byte is the Best Place to Learn Hacking!</h1>
<p>If you are new to hacking, wonderhowto.com's Null Byte</p>
<p> world is the best place to learn hacking</p>
</body></html>
```

Step 5 Let's See What Happens

Now that we have saved our **/var/www/index.html** file, we can check to see what Apache will serve up. Navigate your browser once again to **http://localhost.**


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/
http://localhost/
BackTrack Linux Offensive Security Exploit-DB Aircrack-ng SEORG.org

Null Byte is the Best Place to Learn Hacking!

If you are new to hacking, wonderhowto.com's Null Byte

world is the best place to learn hacking

Apache has served up our webpage just as we created it!

Step 6 Download & Install DVWA

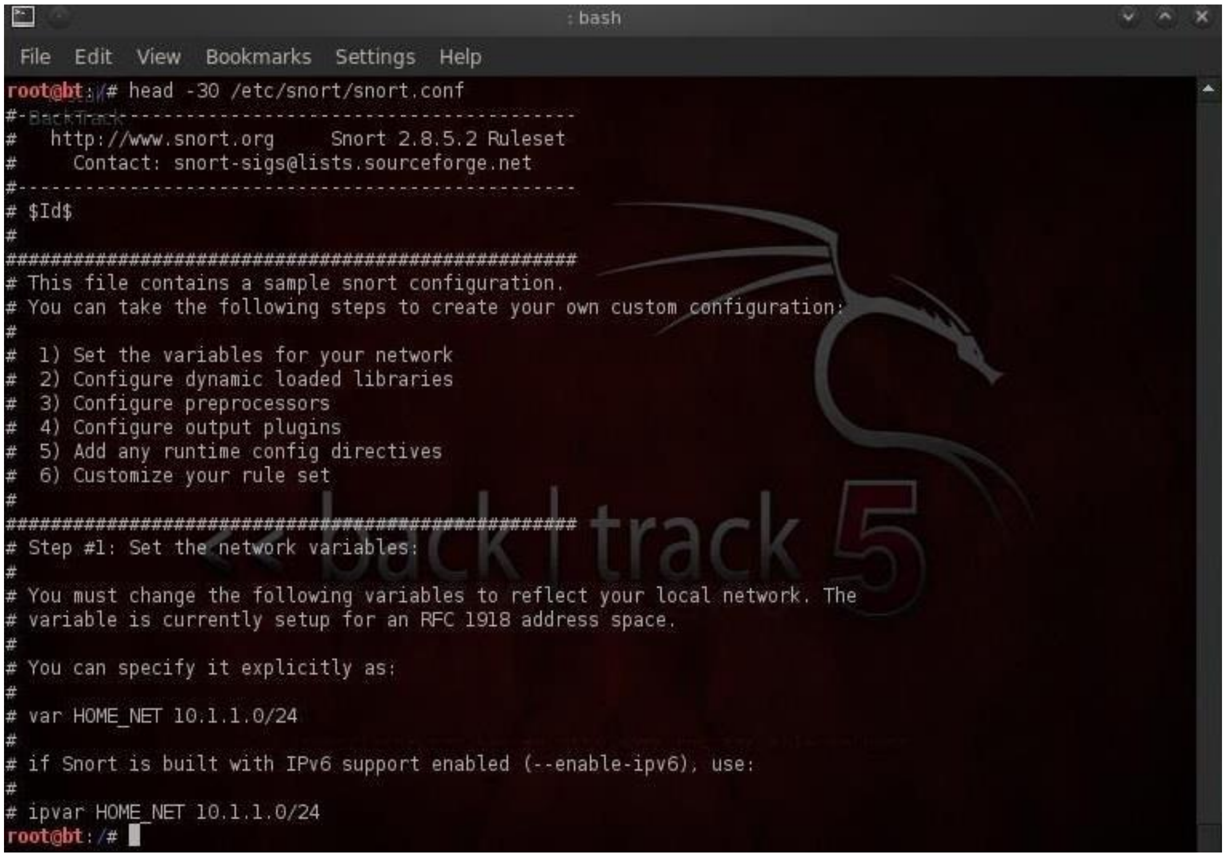
Now that we have our web server up and running, we want to download and install a website designed especially for hacking, known as the **Damn Vulnerable Web Application** or **DVWA**. Let's download it from here, then unzip it. To unzip it, type:

- **unzip DVWA-1.0.8.zip -d /var/www**

Next, we need to change permissions to give us (root) execute permissions.

- **chmod 755 DVWA-1.0.8**

In my next Linux tutorial



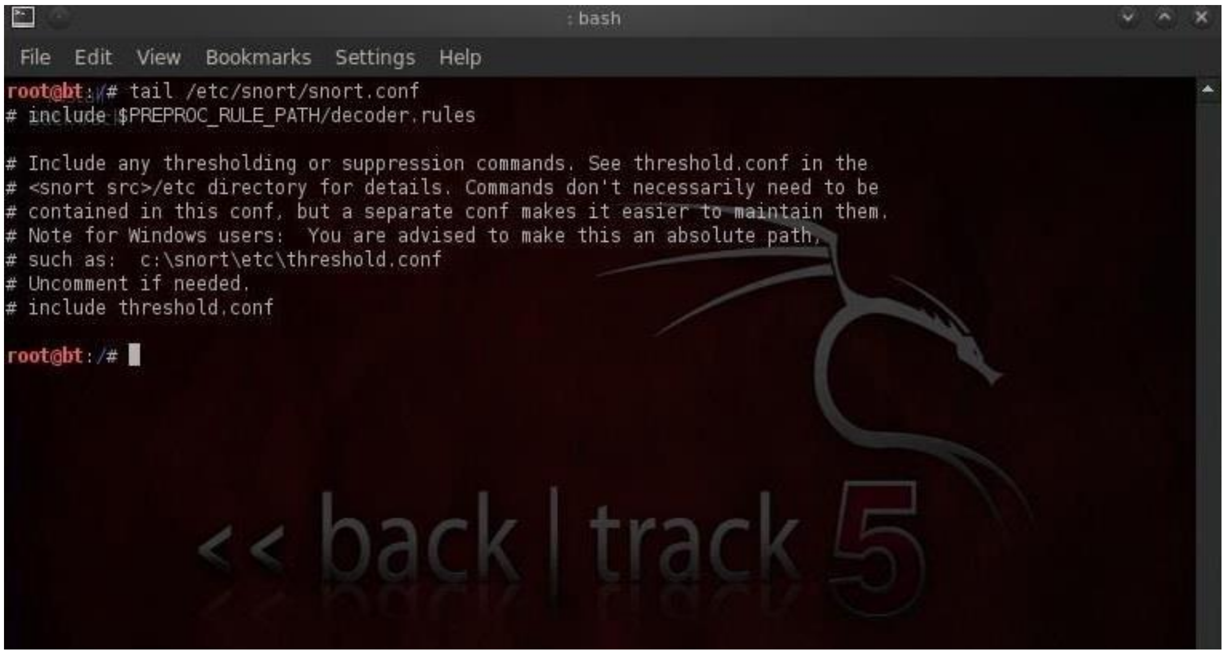
```
root@bt:/# head -30 /etc/snort/snort.conf
#--------------------------------------------------
#   http://www.snort.org     Snort 2.8.5.2 Ruleset
#     Contact: snort-sigs@lists.sourceforge.net
#--------------------------------------------------
# $Id$
#
###################################################
# This file contains a sample snort configuration.
# You can take the following steps to create your own custom configuration:
#
#  1) Set the variables for your network
#  2) Configure dynamic loaded libraries
#  3) Configure preprocessors
#  4) Configure output plugins
#  5) Add any runtime config directives
#  6) Customize your rule set
#
###################################################
# Step #1: Set the network variables:
#
# You must change the following variables to reflect your local network. The
# variable is currently setup for an RFC 1918 address space.
#
# You can specify it explicitly as:
#
# var HOME_NET 10.1.1.0/24
#
# if Snort is built with IPv6 support enabled (--enable-ipv6), use:
#
# ipvar HOME_NET 10.1.1.0/24
root@bt:/# 
```

Here we can see that only the first 30 lines of snort.conf are displayed.

Step 3 Grab That Tail

Similar to the head command, we view the last lines of a file by using the **tail** command. Let's use it on the snort.conf.

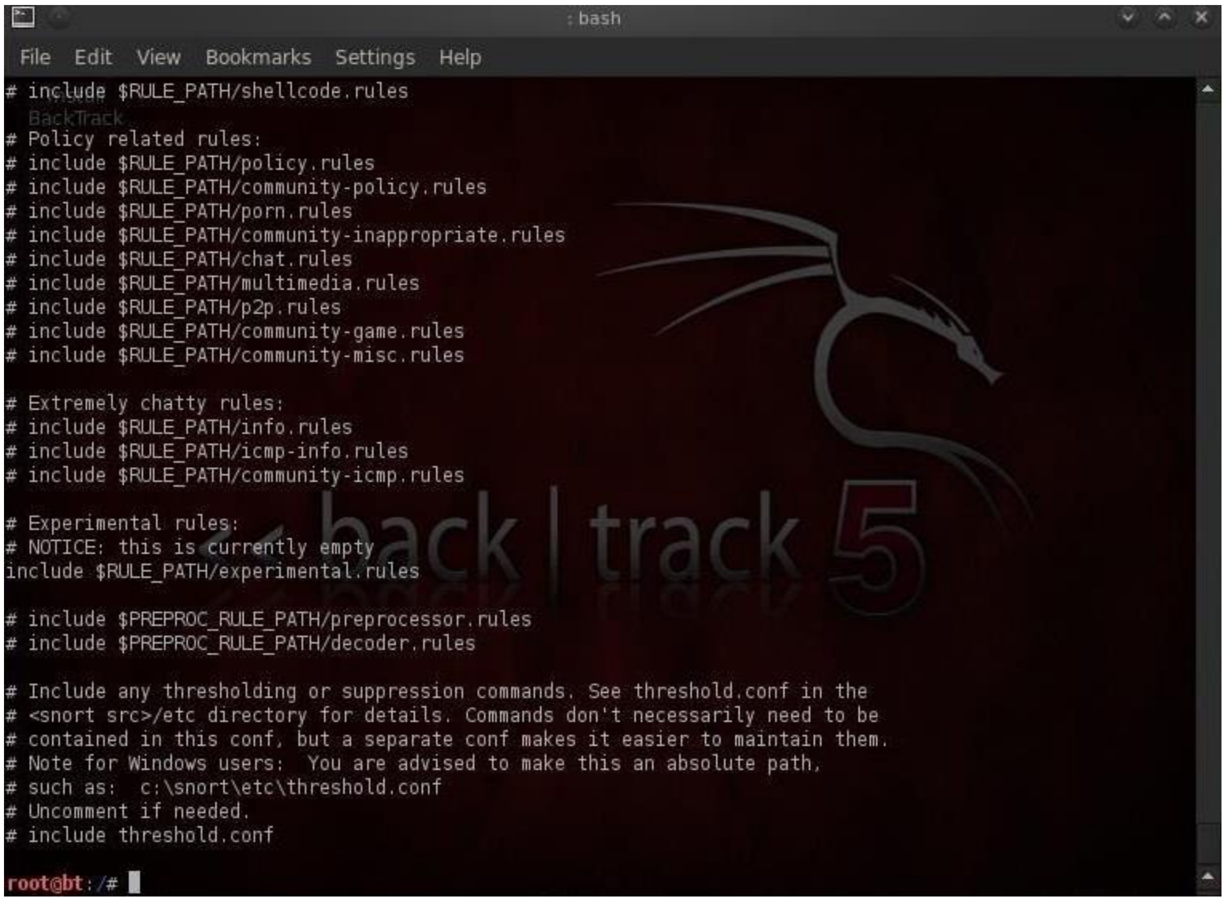
- **tail /etc/snort/snort.conf**



```
root@bt:/# tail /etc/snort/snort.conf
# include $PREPROC_RULE_PATH/decoder.rules

# Include any thresholding or suppression commands. See threshold.conf in the
# <snort src>/etc directory for details. Commands don't necessarily need to be
# contained in this conf, but a separate conf makes it easier to maintain them.
# Note for Windows users:  You are advised to make this an absolute path,
# such as:  c:\snort\etc\threshold.conf
# Uncomment if needed.
# include threshold.conf

root@bt:/# 
```

Notice that it displays some of the last "includes" of the rules files, but not all of them. Let's now see if we can display all the rule "includes" by grabbing the last 40 lines of the snort.conf.

- **tail -40 /etc/snort/snort.conf**



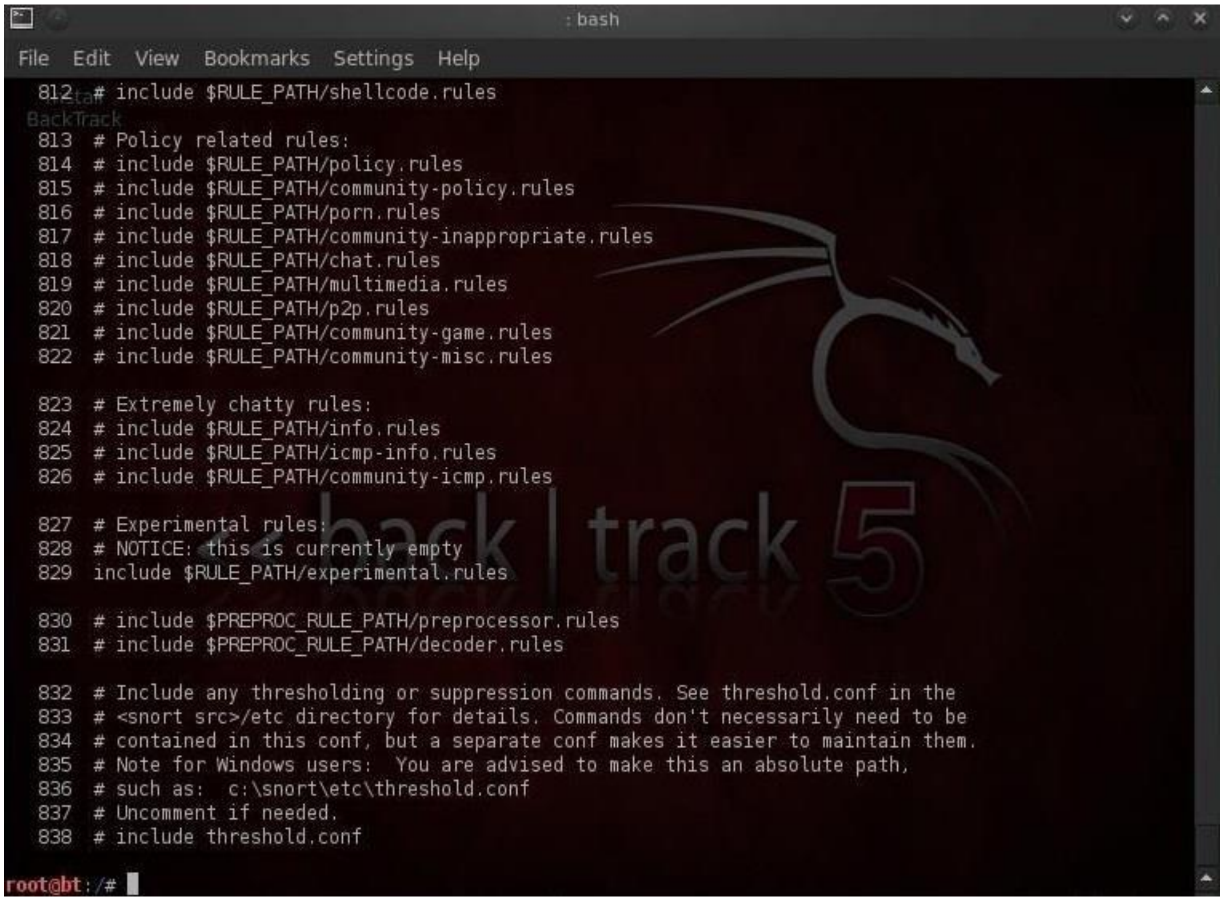
```
# include $RULE_PATH/shellcode.rules

# Policy related rules:
# include $RULE_PATH/policy.rules
# include $RULE_PATH/community-policy.rules
# include $RULE_PATH/porn.rules
# include $RULE_PATH/community-inappropriate.rules
# include $RULE_PATH/chat.rules
# include $RULE_PATH/multimedia.rules
# include $RULE_PATH/p2p.rules
# include $RULE_PATH/community-game.rules
# include $RULE_PATH/community-misc.rules

# Extremely chatty rules:
# include $RULE_PATH/info.rules
# include $RULE_PATH/icmp-info.rules
# include $RULE_PATH/community-icmp.rules

# Experimental rules:
# NOTICE: this is currently empty
include $RULE_PATH/experimental.rules

# include $PREPROC_RULE_PATH/preprocessor.rules
# include $PREPROC_RULE_PATH/decoder.rules

# Include any thresholding or suppression commands. See threshold.conf in the
# <snort src>/etc directory for details. Commands don't necessarily need to be
# contained in this conf, but a separate conf makes it easier to maintain them.
# Note for Windows users:  You are advised to make this an absolute path,
# such as:  c:\snort\etc\threshold.conf
# Uncomment if needed.
# include threshold.conf

root@bt:/#
```

Now we can view nearly all the rule includes all on one screen.

Step 4 Numbering Those Lines

Sometimes—especially with very long files—we may want the file displayed with line numbers. This is probably the case with the snort.conf, as it has 838 lines. This makes it easier to reference changes and come back to the same place within a file. To display a file with line number, we simply type:

- **nl snort.conf**



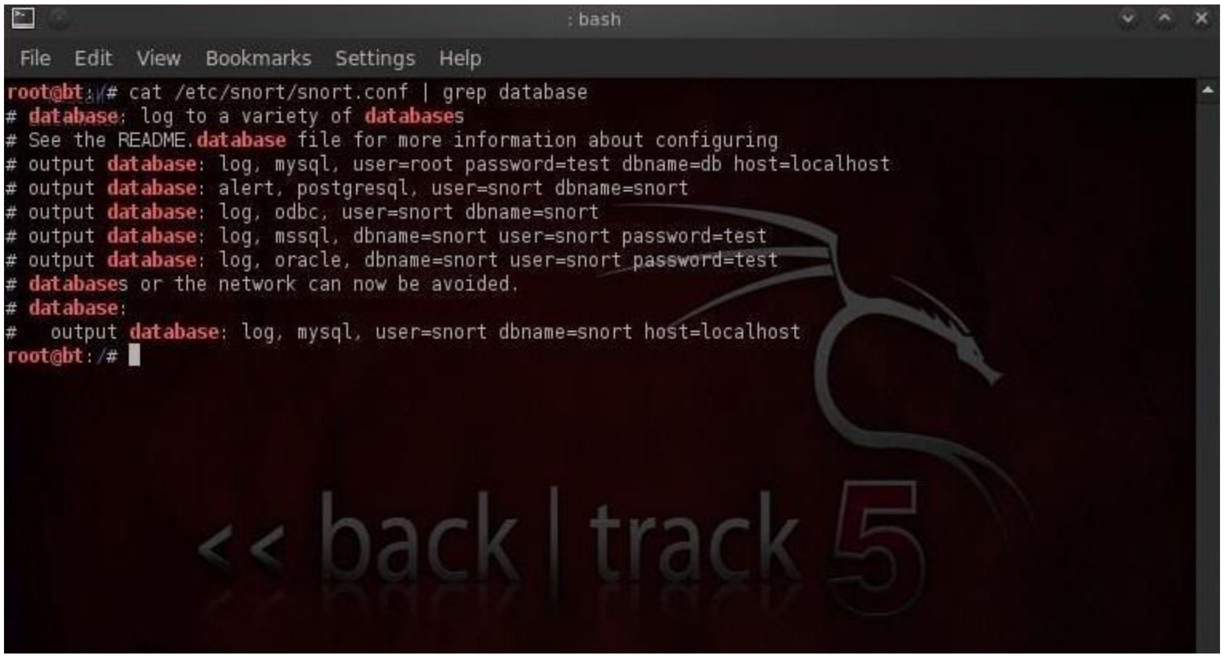
```
812  # include $RULE_PATH/shellcode.rules

813  # Policy related rules:
814  # include $RULE_PATH/policy.rules
815  # include $RULE_PATH/community-policy.rules
816  # include $RULE_PATH/porn.rules
817  # include $RULE_PATH/community-inappropriate.rules
818  # include $RULE_PATH/chat.rules
819  # include $RULE_PATH/multimedia.rules
820  # include $RULE_PATH/p2p.rules
821  # include $RULE_PATH/community-game.rules
822  # include $RULE_PATH/community-misc.rules

823  # Extremely chatty rules:
824  # include $RULE_PATH/info.rules
825  # include $RULE_PATH/icmp-info.rules
826  # include $RULE_PATH/community-icmp.rules

827  # Experimental rules:
828  # NOTICE: this is currently empty
829  include $RULE_PATH/experimental.rules

830  # include $PREPROC_RULE_PATH/preprocessor.rules
831  # include $PREPROC_RULE_PATH/decoder.rules

832  # Include any thresholding or suppression commands. See threshold.conf in the
833  # <snort src>/etc directory for details. Commands don't necessarily need to be
834  # contained in this conf, but a separate conf makes it easier to maintain them.
835  # Note for Windows users:  You are advised to make this an absolute path,
836  # such as:  c:\snort\etc\threshold.conf
837  # Uncomment if needed.
838  # include threshold.conf

root@bt:/#
```

Note that each line now has a number making referencing much easier.

Step 5 I Grep That

After cat, **grep** is probably the most widely used text manipulation command. It's a filtering command; in other words, it enables us to filter the content of a file for display. If for instance, we wanted to see all the instances of where the word "database" occurs in our snort.conf file, we could ask cat to only display those lines where it occurs by typing:

- **cat /etc/snort/ snort.conf | grep database**



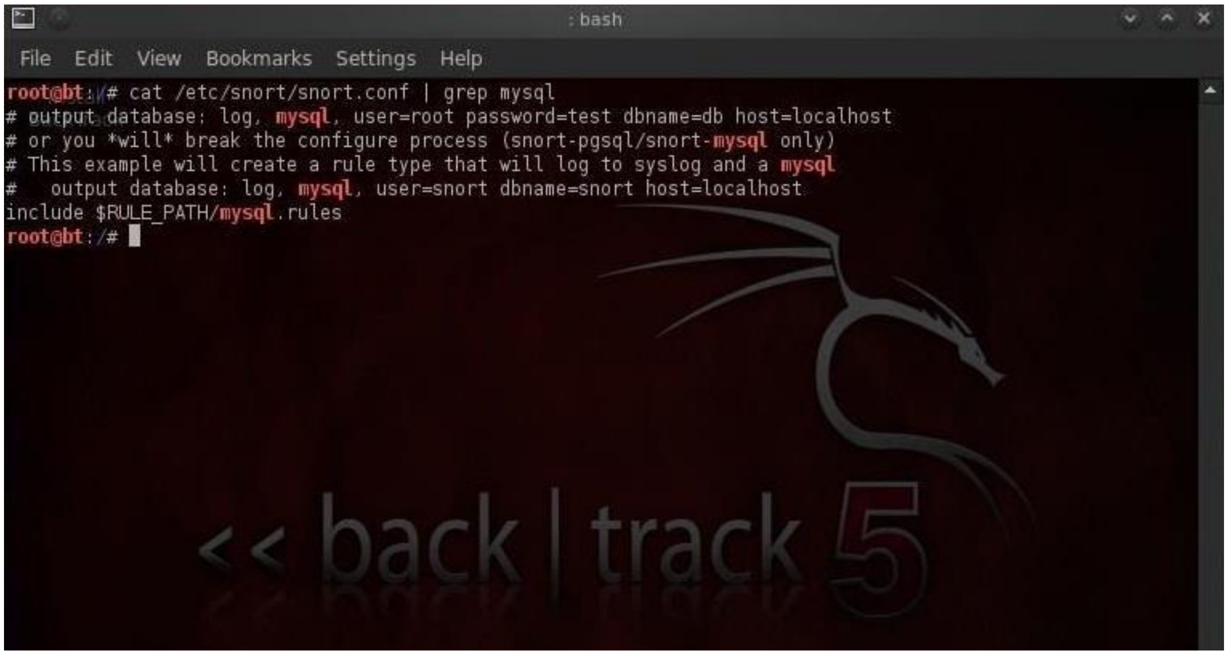
```
root@bt:/# cat /etc/snort/snort.conf | grep database
# database: log to a variety of databases
# See the README.database file for more information about configuring
# output database: log, mysql, user=root password=test dbname=db host=localhost
# output database: alert, postgresql, user=snort dbname=snort
# output database: log, odbc, user=snort dbname=snort
# output database: log, mssql, dbname=snort user=snort password=test
# output database: log, oracle, dbname=snort user=snort password=test
# databases or the network can now be avoided.
# database:
#   output database: log, mysql, user=snort dbname=snort host=localhost
root@bt:/# 
```

This command will first grab the snort.conf and then "pipe" it (|) to grep which will take it as input and then look for the occurrences of the word "database" and only display those lines. Grep is a powerful and essential command for working in Linux as it can save us hours searching for every occurrence of a word or command.

Step 6 I Sed That Works

The **sed** command essentially allows us to search for occurrences of a word or text pattern and then do some work on it. The name comes from the concept of a stream editor and is a contraction of those two words. In its most basic form, sed operates like the find and replace function in Windows. Let's search for the word "mysql" in the snort.conf file using grep.

- **cat /etc/snort/snort.conf | grep mysql**



```
root@bt:/# cat /etc/snort/snort.conf | grep mysql
# output database: log, mysql, user=root password=test dbname=db host=localhost
# or you *will* break the configure process (snort-pgsql/snort-mysql only)
# This example will create a rule type that will log to syslog and a mysql
#   output database: log, mysql, user=snort dbname=snort host=localhost
include $RULE_PATH/mysql.rules
root@bt:/#
```

We can see that the grep command found five occurrences of the word mysql.

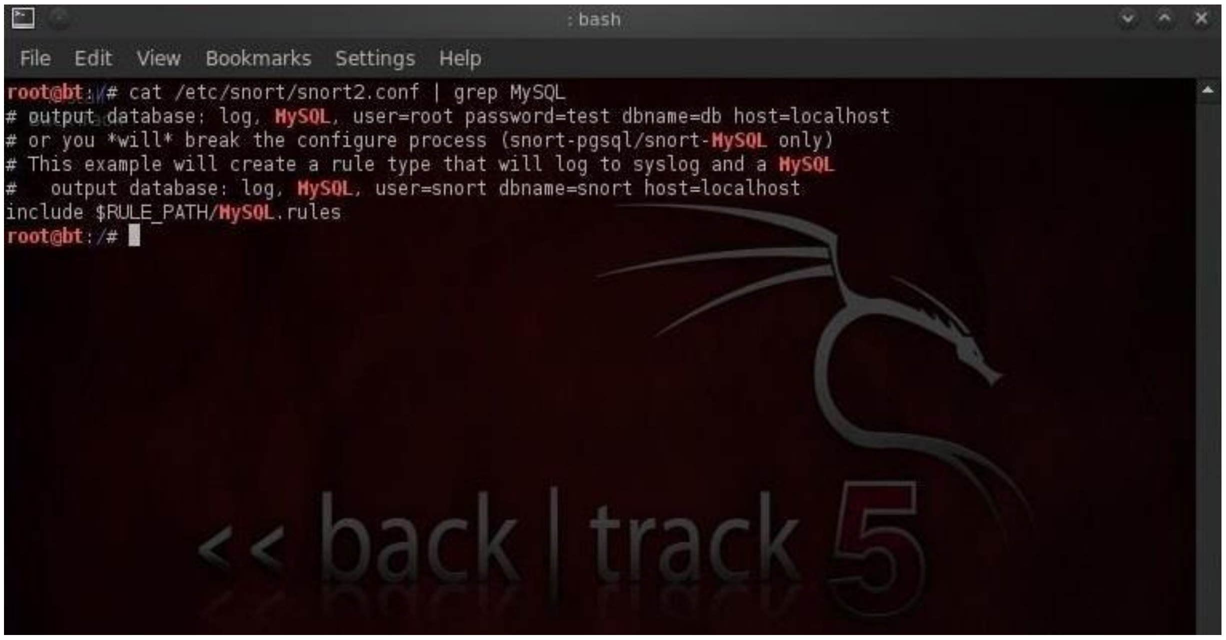
Let's say we want sed to replace every occurrence of mysql and with MySQL (remember, Linux is case sensitive) and then save the new file to snort2.conf. We could do this by typing:

- **sed s/mysql/MySQL/g snort.conf > snort2.conf**

This command says, "search (s) for the word mysql and replace it with the word MySQL globally (i.e. wherever you find it in the file)."

Now, when we grep snort2.conf for mysql, we see that none were found and when we grep for MySQL, we find five occurrences of MySQL.

- **cat /etc/snort/snort.conf | grep MySQL**



```
root@bt:/# cat /etc/snort/snort2.conf | grep MySQL
# output database: log, MySQL, user=root password=test dbname=db host=localhost
# or you *will* break the configure process (snort-pgsql/snort-MySQL only)
# This example will create a rule type that will log to syslog and a MySQL
#   output database: log, MySQL, user=snort dbname=snort host=localhost
include $RULE_PATH/MySQL.rules
root@bt:/#
```

If we just want to replace only the first occurrence of the word mysql, we could leave out the trailing **g** and it would only replace the first occurrence.

- **sed s/mysql/MySQL/ snort.conf > snort2.conf**

The sed command can also be used to find and replace any specific occurrence of a word. For instance, if I want to only replace the third occurrence of the word mysql, I can simply place the number of the occurrence at the end of the command and sed will only replace the third occurrence of the word "mysql" with "MySQL".

- **sed s/mysql/MySQL/3 snort.conf > snort2.conf**

Loadable Kernel Modules

I now want to address Loadable kernel modules (LKMs), which are key to the Linux administrator because they provide us the capability to add functionality to the kernel without having to recompile the kernel. Things like video and other device drivers can now be added to the kernel without shutting down the system, recompiling, and rebooting.

Loadable kernel modules are critical to the hacker because if we can get the Linux admin to load a new module to their kernel, we not only own their system, but because we are at the kernel level of their operating system, we can control even what their system is reporting to them in terms of processes, ports, services, hard drive space, etc.

So, if we can offer the Linux user/admin a "new and improved" video driver with our rootkit embedded in it, we can take control of his system and kernel. This is the way some of the most insidious rootkits take advantage of the Linux OS.

So, I hope it's clear that understanding LKMs is key to being an effective Linux admin and being a VERY effective and stealthy hacker.

Step 1 What Is a Kernel Module?

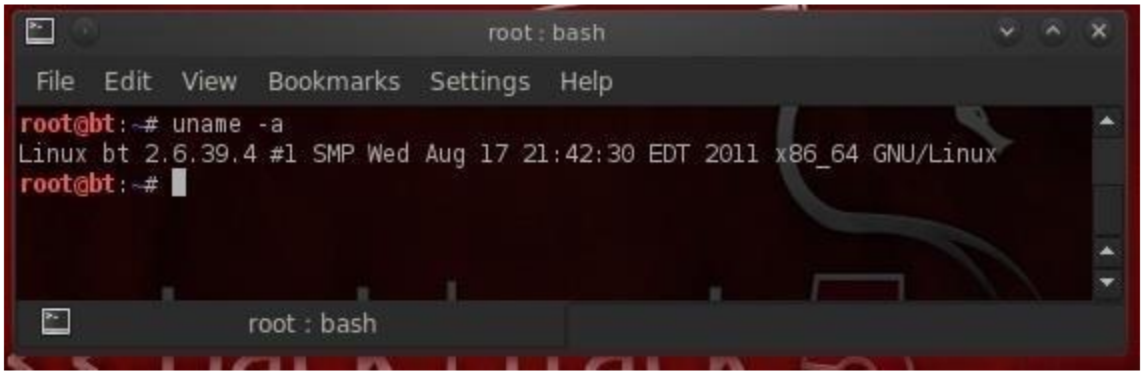
The kernel is a core component of any Linux operating system, including our BackTrack System. The kernel is the central nervous system of our operating system, controlling everything an operating system does, including managing the interactions between the hardware components and starting the necessary services. The kernel operates between user applications and the hardware such as the CPU, memory, the hard drive, etc.

As the kernel manages all that is taking place with the operating system, sometimes it needs updates. These updates might include new device drivers (such as video card or USB devices), file system drivers, and even system extensions. This is where LKMs come in. We can now simply load and unload kernel modules as we need them without recompiling the kernel.

Step 2 Checking the Kernel

The first thing we want to do is check to see what kernel our system is running. There are at least two ways to do this. We can type:

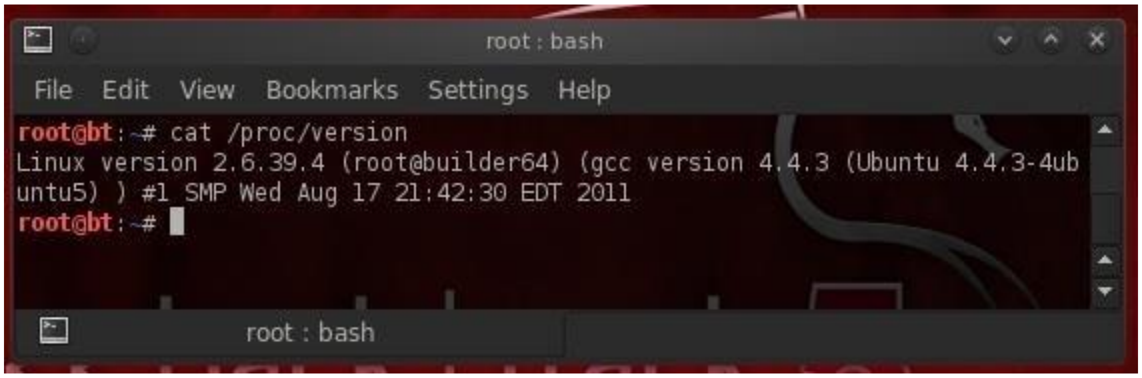
- **uname -a**



```
root@bt:~# uname -a
Linux bt 2.6.39.4 #1 SMP Wed Aug 17 21:42:30 EDT 2011 x86_64 GNU/Linux
root@bt:~# 
```

Note that the kernel tells us its kernel build (2.6.39.4), but also the architecture it is built for (x86_64). We can also get this info by "catting" the /proc/version file, which actually gives up even more info.

- **cat /proc/version**



```
root@bt:~# cat /proc/version
Linux version 2.6.39.4 (root@builder64) (gcc version 4.4.3 (Ubuntu 4.4.3-4ub
untu5) ) #1 SMP Wed Aug 17 21:42:30 EDT 2011
root@bt:~# 
```

Step 3 Kernel Tuning with Sysctl

Sometimes, a Linux admin will want to "tune" the kernel. This might include changing memory allocations, enabling networking feature, and even hardening the kernel from hackers.

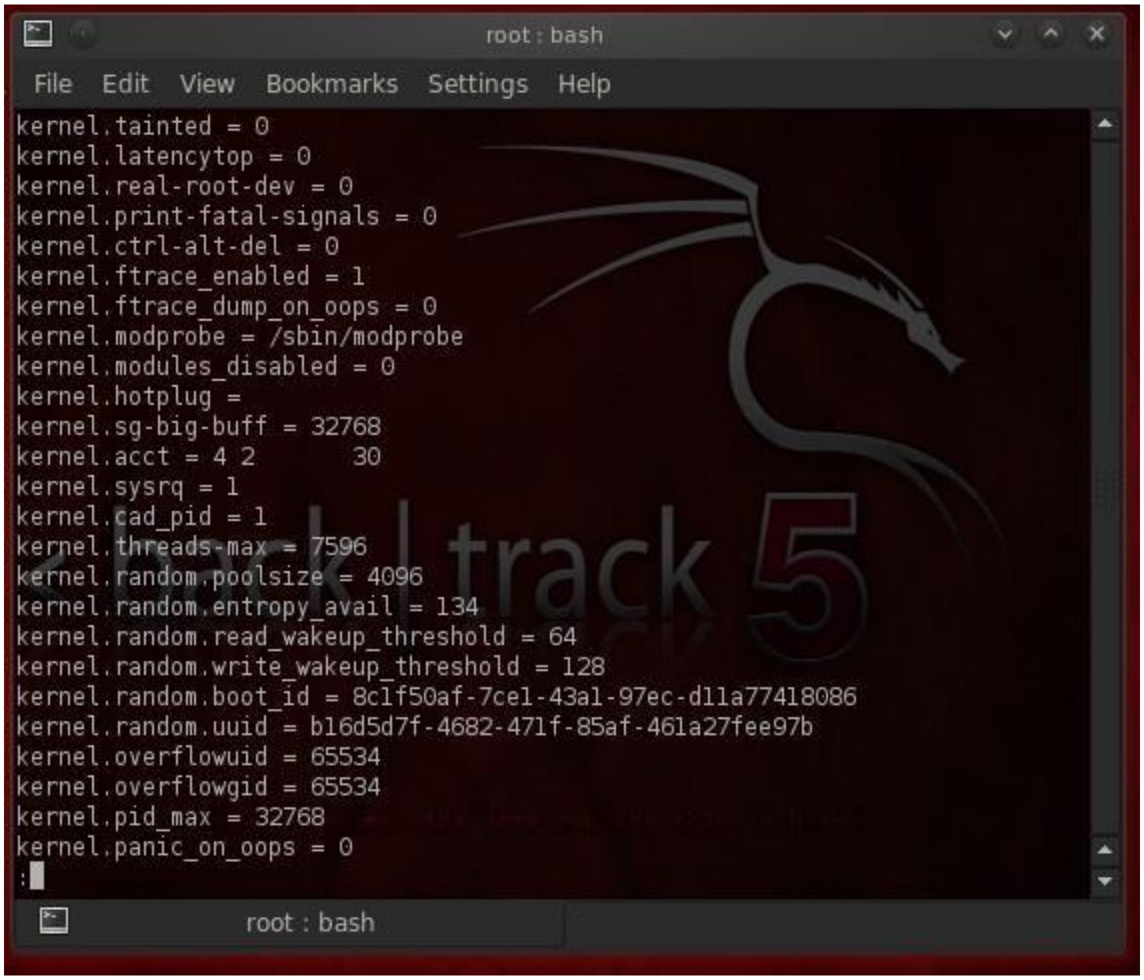
With modern Linux kernels, we have the **sysctl** command to tune kernel options. All changes you make with the sysctl remain in effect only until you reboot the system. To make any changes permanent, the configuration file for sysctl must be edited at /etc/sysctl.conf.

Be careful in using systctl because without the proper knowledge and experience, you can easily make your system unbootable and unusable. Let's take a look at the contents of sysctl now.

- **sysctl -a |less**

To view the configuration file for sysctl, we can get it at /etc/sysctl.conf.

- **less /etc/sysctl.conf**



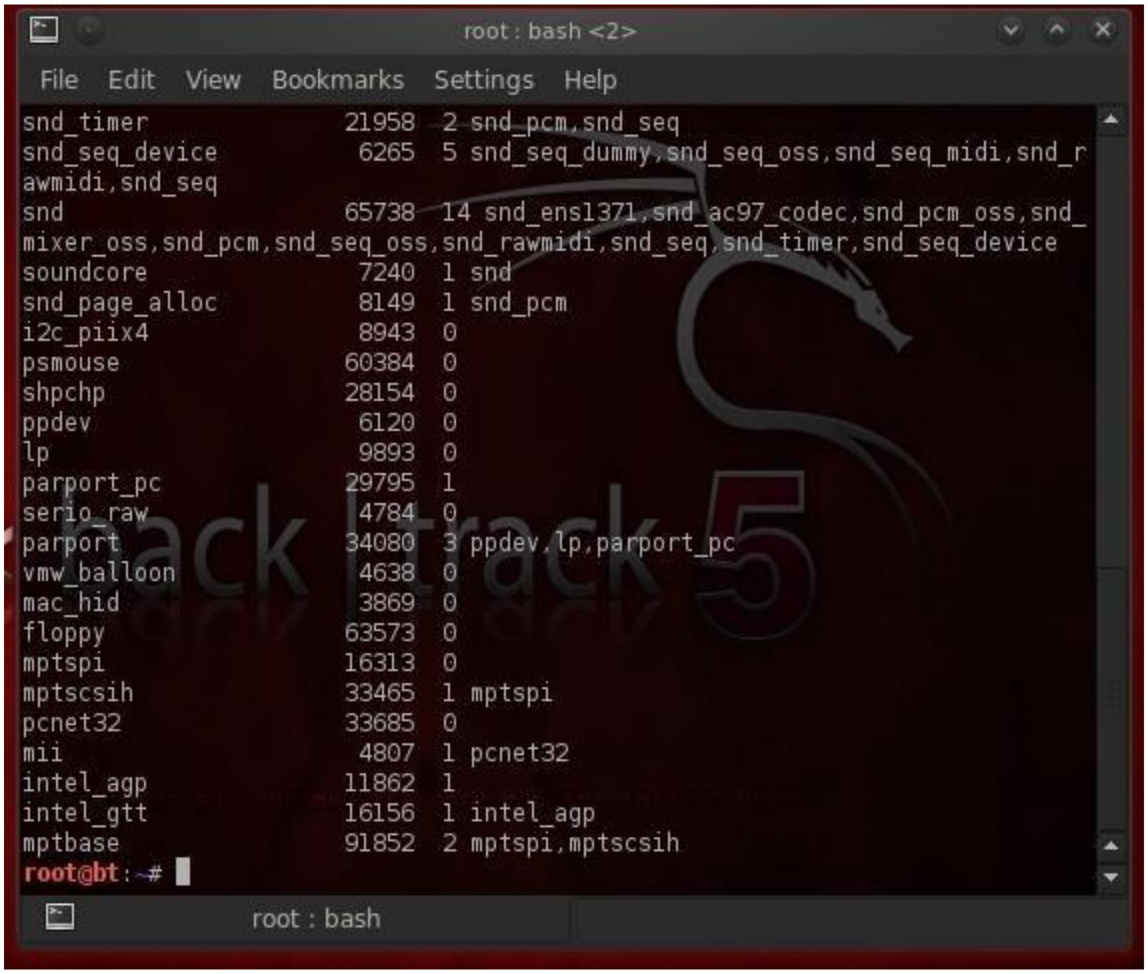
```
kernel.tainted = 0
kernel.latencytop = 0
kernel.real-root-dev = 0
kernel.print-fatal-signals = 0
kernel.ctrl-alt-del = 0
kernel.ftrace_enabled = 1
kernel.ftrace_dump_on_oops = 0
kernel.modprobe = /sbin/modprobe
kernel.modules_disabled = 0
kernel.hotplug =
kernel.sg-big-buff = 32768
kernel.acct = 4 2       30
kernel.sysrq = 1
kernel.cad_pid = 1
kernel.threads-max = 7596
kernel.random.poolsize = 4096
kernel.random.entropy_avail = 134
kernel.random.read_wakeup_threshold = 64
kernel.random.write_wakeup_threshold = 128
kernel.random.boot_id = 8c1f50af-7ce1-43a1-97ec-d11a77418086
kernel.random.uuid = b16d5d7f-4682-471f-85af-461a27fee97b
kernel.overflowuid = 65534
kernel.overflowgid = 65534
kernel.pid_max = 32768
kernel.panic_on_oops = 0
:
```

One of the ways we may want to use sysctl for hacking is to enable ipforwarding (net.ipv4.conf.default.forwarding) for man-in-the-middle attacks. From a hardening perspective, we can disable ICMP echo requests (net.ipv4.icmp_echo_ignore_all) so as to make more difficult, but not impossible, for hackers to find our system.

Step 4 Kernel Modules

To manage our kernels, Linux has at least two ways to do it. The older way is to use a group of commands built around the **insmod** command. Here we use one of those—**lsmod**—to list the installed modules in kernel.

- **lsmod**



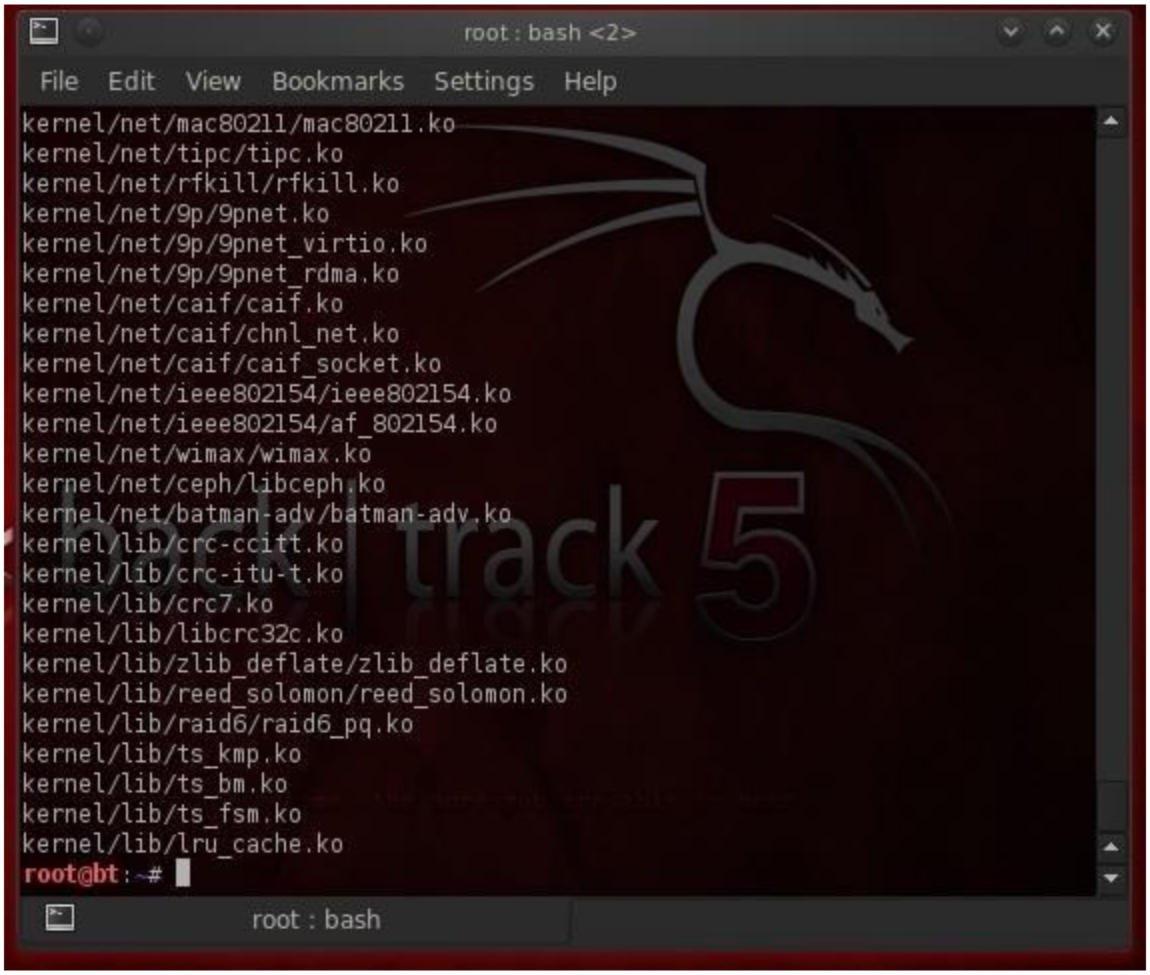
```
snd_timer              21958  2 snd_pcm,snd_seq
snd_seq_device          6265  5 snd_seq_dummy,snd_seq_oss,snd_seq_midi,snd_r
awmidi,snd_seq
snd                    65738  14 snd_ens1371,snd_ac97_codec,snd_pcm_oss,snd_
mixer_oss,snd_pcm,snd_seq_oss,snd_rawmidi,snd_seq,snd_timer,snd_seq_device
soundcore               7240  1 snd
snd_page_alloc          8149  1 snd_pcm
i2c_piix4               8943  0
psmouse                60384  0
shpchp                 28154  0
ppdev                   6120  0
lp                      9893  0
parport_pc             29795  1
serio_raw               4784  0
parport                34080  3 ppdev,lp,parport_pc
vmw_balloon             4638  0
mac_hid                 3869  0
floppy                 63573  0
mptspi                 16313  0
mptscsih               33465  1 mptspi
pcnet32                33685  0
mii                     4807  1 pcnet32
intel_agp              11862  1
intel_gtt              16156  1 intel_agp
mptbase                91852  2 mptspi,mptscsih
root@bt:~#
```

We can load or insert a module with **insmod** and remove a module with **rmmod**.

Step 5 Modprobe

Most newer distributions of Linux, including our BackTrack 5v3, have converted to the **modprobe** command for LKM management. To see what modules are installed in our kernel, we can type:

- **modprobe -l**



```
kernel/net/mac80211/mac80211.ko
kernel/net/tipc/tipc.ko
kernel/net/rfkill/rfkill.ko
kernel/net/9p/9pnet.ko
kernel/net/9p/9pnet_virtio.ko
kernel/net/9p/9pnet_rdma.ko
kernel/net/caif/caif.ko
kernel/net/caif/chnl_net.ko
kernel/net/caif/caif_socket.ko
kernel/net/ieee802154/ieee802154.ko
kernel/net/ieee802154/af_802154.ko
kernel/net/wimax/wimax.ko
kernel/net/ceph/libceph.ko
kernel/net/batman-adv/batman-adv.ko
kernel/lib/crc-ccitt.ko
kernel/lib/crc-itu-t.ko
kernel/lib/crc7.ko
kernel/lib/libcrc32c.ko
kernel/lib/zlib_deflate/zlib_deflate.ko
kernel/lib/reed_solomon/reed_solomon.ko
kernel/lib/raid6/raid6_pq.ko
kernel/lib/ts_kmp.ko
kernel/lib/ts_bm.ko
kernel/lib/ts_fsm.ko
kernel/lib/lru_cache.ko
root@bt:~#
```

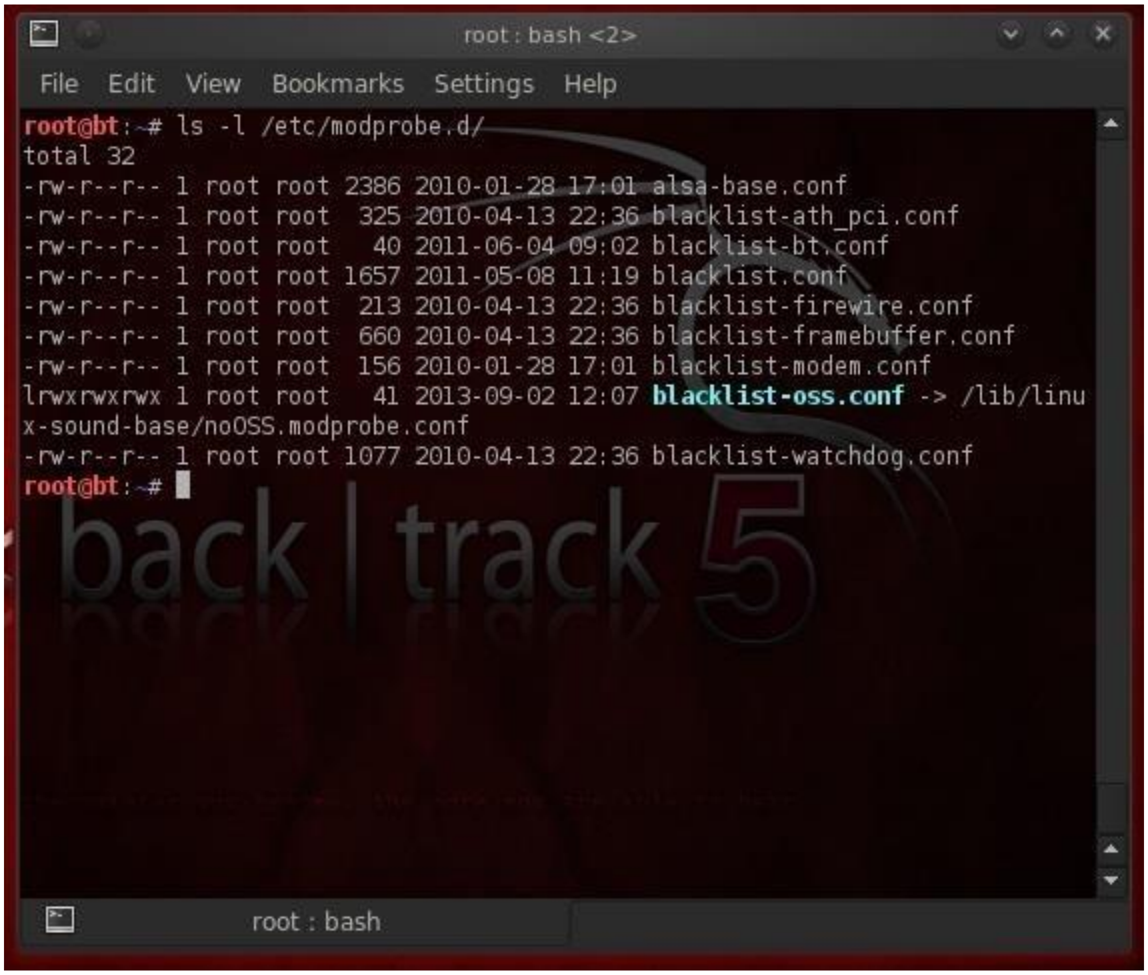
To remove a module, we simply use the -r switch with modprobe.

- **modprobe -r**

A major advantage of modprobe is that understands dependencies, options, and installation and removal procedures for our kernel modules.

To see configuration files for the installed modules, we list the contents of the /etc/modprobe.d/ directory.

- **ls -l /etc/modprobe.d/**



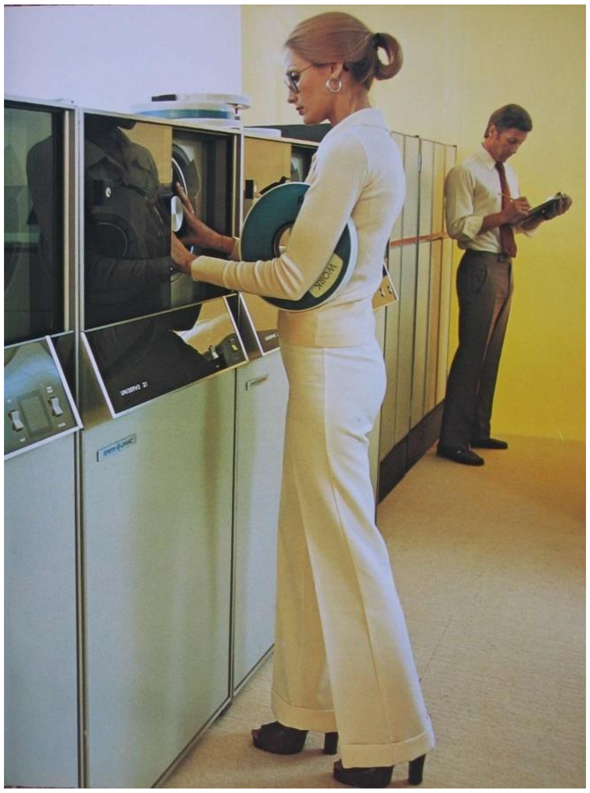
```
root@bt:~# ls -l /etc/modprobe.d/
total 32
-rw-r--r-- 1 root root 2386 2010-01-28 17:01 alsa-base.conf
-rw-r--r-- 1 root root  325 2010-04-13 22:36 blacklist-ath_pci.conf
-rw-r--r-- 1 root root   40 2011-06-04 09:02 blacklist-bt.conf
-rw-r--r-- 1 root root 1657 2011-05-08 11:19 blacklist.conf
-rw-r--r-- 1 root root  213 2010-04-13 22:36 blacklist-firewire.conf
-rw-r--r-- 1 root root  660 2010-04-13 22:36 blacklist-framebuffer.conf
-rw-r--r-- 1 root root  156 2010-01-28 17:01 blacklist-modem.conf
lrwxrwxrwx 1 root root   41 2013-09-02 12:07 blacklist-oss.conf -> /lib/linu
x-sound-base/noOSS.modprobe.conf
-rw-r--r-- 1 root root 1077 2010-04-13 22:36 blacklist-watchdog.conf
root@bt:~# 
```

Remember, the LKM modules are a convenience to a Linux user/admin, but are a major security weakness of Linux and one the professional hacker should be familiar with. As I said before, the LKM can be the perfect vehicle to get your rootkit into the kernel and wreak havoc!

Mounting Drives & Devices

One of those areas of Linux that Windows users invariably struggle with is the concept of "mounting" devices and drives. In the Windows world, drives and devices are automatically "mounted" without any user effort or knowledge. Well, maybe a bit of knowledge. Most Windows users know to unmount their flash drive before removing it, but they usually think of it as "ejecting" it.

The mount command has a history back to the prehistoric days of computing (the 1970s) when computer operators physically mounted tape drives to the the behemoth, gymnasium-sized computers. These tape drives were the storage medium of choice (as hard drives had not been invented yet) and the operator had to tell the machine that they were mounting the tape before it could be read.

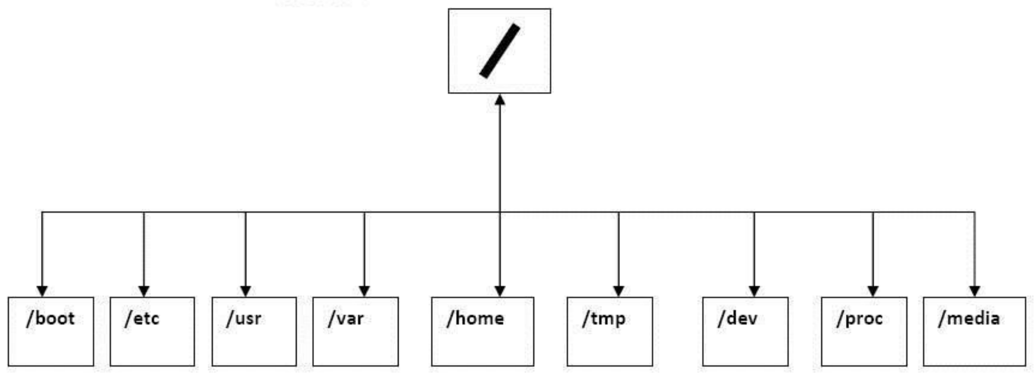


Windows generally auto-mounts drives and devices with the PnP service, so users don't need to think about mounting. Each drive or device then is assigned with a letter mount point such as C:, D:, E:, etc.

In more recent distributions of Linux, auto-mount is often enabled as well, but the true Linux admin needs to understand the mount command and the mounting process as they will someday need to mount a device or drive that does not auto-mount. This is true for the everyday ordinary sysadmin in Linux and especially true for the digital forensic investigator and hacker as many times the devices will not be automatically mounted.

Step 1 File Structure

Remember, Linux has a single tree structure for its file system (unlike Windows) with a root for every drive and device. This means that all drives and devices are part of a single filesystem tree with / at the top. Any other drives must be "mounted" to this tree. We can do this with the mount command.



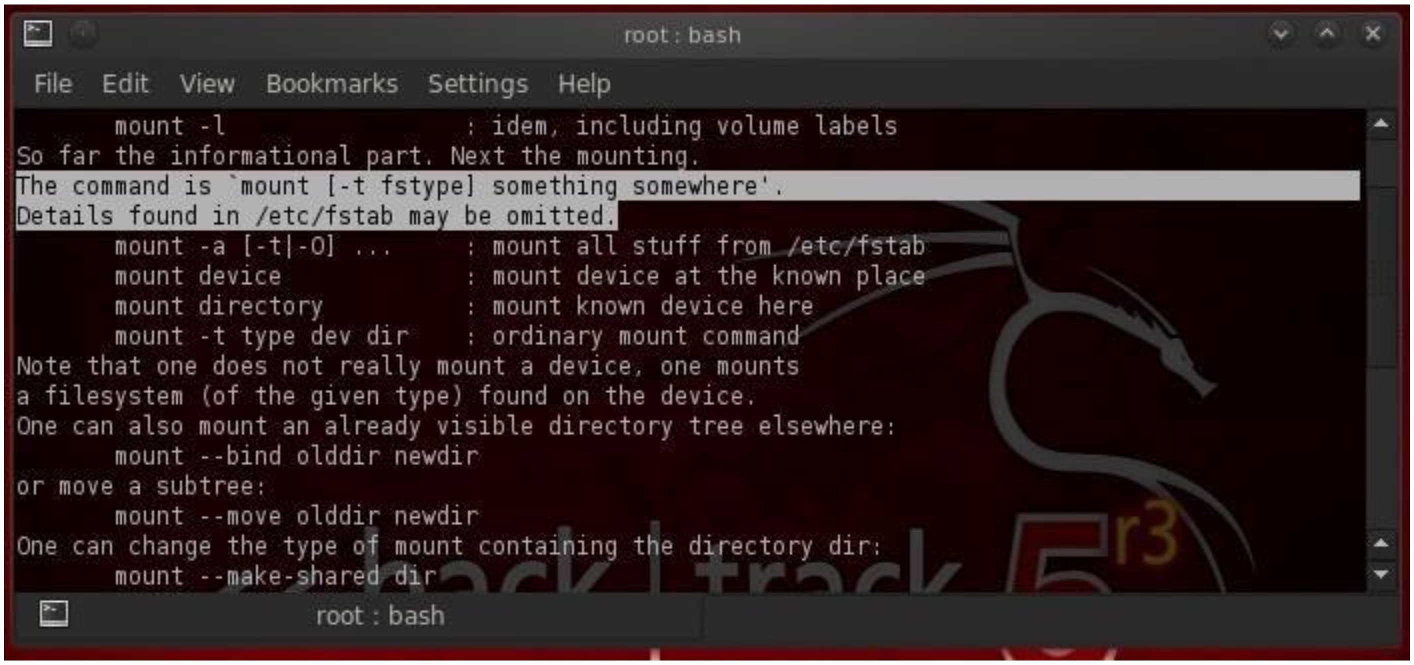
When we mount a device, we mount it to a directory and it becomes part of the tree. We can mount a device to ANY directory, but when we do so, that directory that we mount our device to is "covered" and unavailable to us. This means we can't access any of the files in that directory. It goes without saying—I think—that's not good. That's why we have special, empty directories for mounting devices. These will vary by distribution of Linux, but generally they are /mnt and /media.

Step 2 Mount Command

Let's take a look at the mount command. Type in:

- **mount -h**

This brings up the help screen displayed below.



```
      mount -l               : idem, including volume labels
So far the informational part. Next the mounting.
The command is `mount [-t fstype] something somewhere'.
Details found in /etc/fstab may be omitted.
      mount -a [-t|-O] ...     : mount all stuff from /etc/fstab
      mount device             : mount device at the known place
      mount directory          : mount known device here
      mount -t type dev dir    : ordinary mount command
Note that one does not really mount a device, one mounts
a filesystem (of the given type) found on the device.
One can also mount an already visible directory tree elsewhere:
      mount --bind olddir newdir
or move a subtree:
      mount --move olddir newdir
One can change the type of mount containing the directory dir:
      mount --make-shared dir
```

I have highlighted the crucial part regarding the syntax of the command. Basically, it is:

- **mount -t filesystemtype location**

This command will "mount" a filesystem of the type (-t) at the location specified. So, for instance, we could mount cdrom at the media directory by typing:

- **mount -t /dev/cdrom /media**

This will mount the cdrom device at the /media directory on the filesystem tree.

We also have numerous options we can use when mounting a device including:

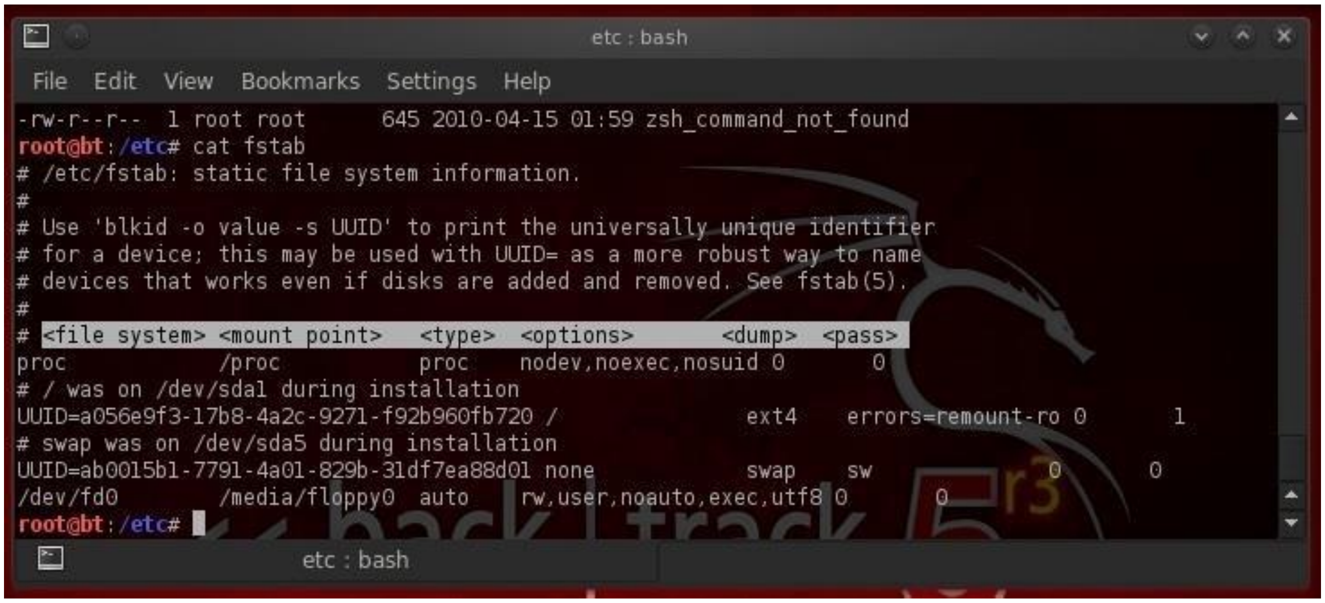
- **rw** - mount read/write
- **ro** - mount read only
- **user** - permit any user to mount
- **auto**/**noauto** - file system will or will NOT automatically mount
- **exec**/**noexec** - permit or prevent the execution of binaries on the mounted device

As always, you can check the man page for *mount* to learn all the options:

- **man mount**

Step 3 Setting up Automounting with Fstab

The fstab is the "**F**ile **s**ystem **tab**le". It a system configuration file in Linux. The mount command reads the fstab to determine what options to use when mounting a filesystem. In this way, it defines the options automatically when we mount the device. It simply reads the entry in the fstab table for that device and applies those options defined there.



```
-rw-r--r--  1 root root      645 2010-04-15 01:59 zsh_command_not_found
root@bt:/etc# cat fstab
# /etc/fstab: static file system information.
#
# Use 'blkid -o value -s UUID' to print the universally unique identifier
# for a device; this may be used with UUID= as a more robust way to name
# devices that works even if disks are added and removed. See fstab(5).
#
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
proc            /proc           proc    nodev,noexec,nosuid 0       0
# / was on /dev/sda1 during installation
UUID=a056e9f3-17b8-4a2c-9271-f92b960fb720 /               ext4    errors=remount-ro 0       1
# swap was on /dev/sda5 during installation
UUID=ab0015b1-7791-4a01-829b-31df7ea88d01 none            swap    sw              0       0
/dev/fd0        /media/floppy0  auto    rw,user,noauto,exec,utf8 0       0
root@bt:/etc#
```

As we can see in the screenshot above, we have simply displayed the contents of fstab with the cat command.

- **cat fstab**

The fstab table is comprised of six (6) columns. These are:

1. **Device** - the UUID
2. **Mount point** - the directory where we want to attach the device
3. **Type** - the filesystem type such ext2, ext3, swap, ISO9660, etc.
4. **Options** - these rw (read/write), auto, nouser, async, suid, etc
5. **Dump** - indicates how often to backup the device
6. **Pass** - specifies the pass when fsck should check the filesystem

Step 4 Umount

When want to unmount a drive or device, the command we use is **umount** (that's right. I didn't spell it wrong. It is umount, not unmount).

To unmount our cdrom device that we mounted above, we type:

- **umount /dev/cdrom**

You can NOT unmount a drive or device that is currently being used by the system.

MySQL

This tutorial we will focus on the MySQL database. Although this is not strictly a Linux tutorial, MySQL is the database of choice on most Linux distributions. In addition, it is the most widely used database behind database driven web applications. This installment is critical to understand before we progress to hacking MySQL databases and before we hack web applications that use MySQL (which there are literally thousands).

MySQL is an open source, GPL licensed database. That is probably the primary reason you will find it on nearly every Linux distribution. As you know, Linux is also open source and GPL licensed. First developed by MySQL AB of Sweden in 1995, it was purchased by Sun Microsystems in 2008 and Sun Microsystems was then purchased by Oracle in 2009.

As Oracle is the world's largest database software publisher, the open source community has significant trepidations about Oracle's commitment to keep MySQL open source. As a result, there is now a fork of the MySQL database software called Maria that IS committed to keeping this software and its subsequent versions open source.

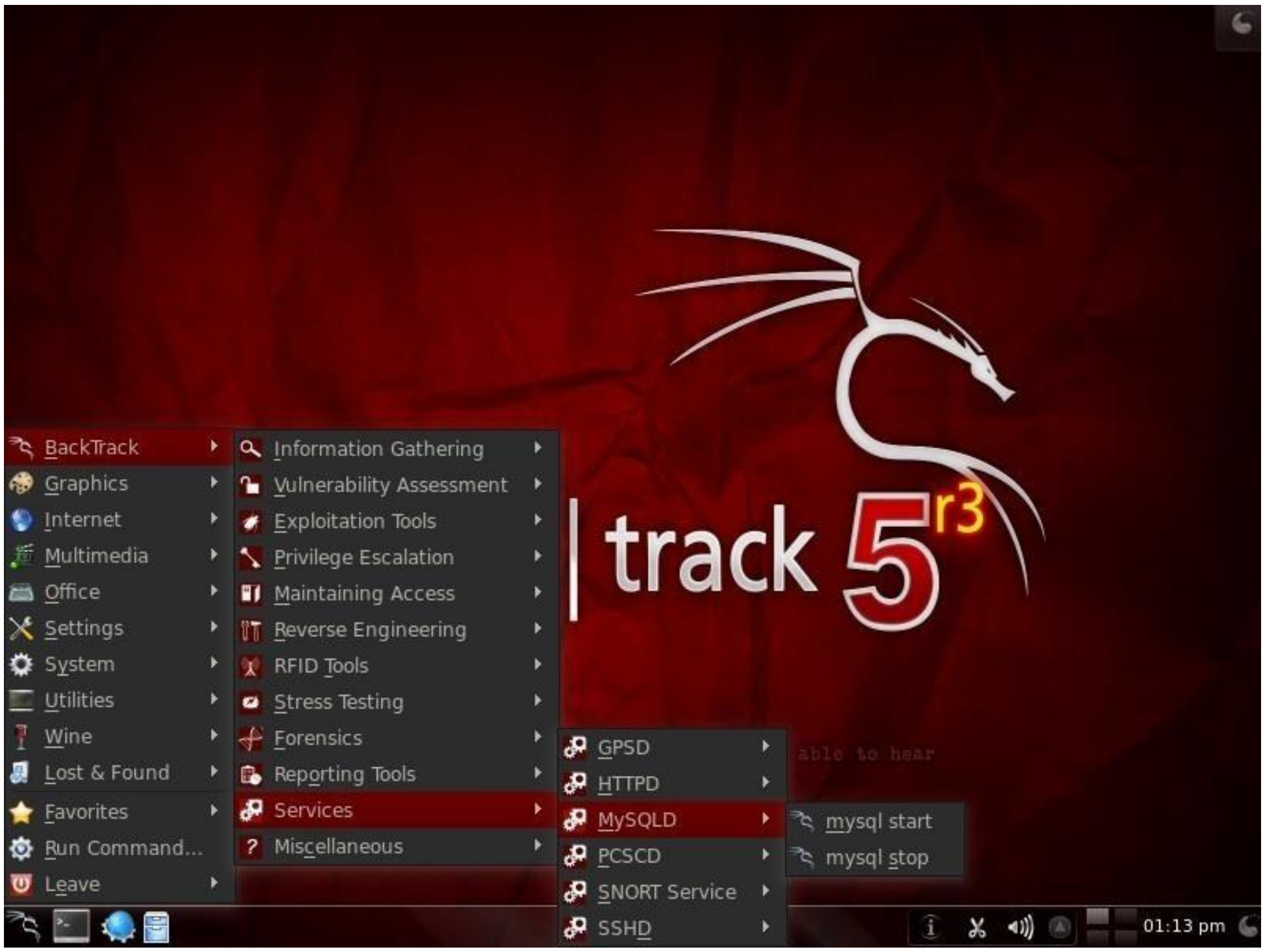
Because it's free, MySQL has become the database of choice for many web applications. Sites and apps that use it include:

- WordPress
- Facebook
- LinkedIn
- Twitter
- Kayak
- Walmart.com
- Wikipedia
- YouTube

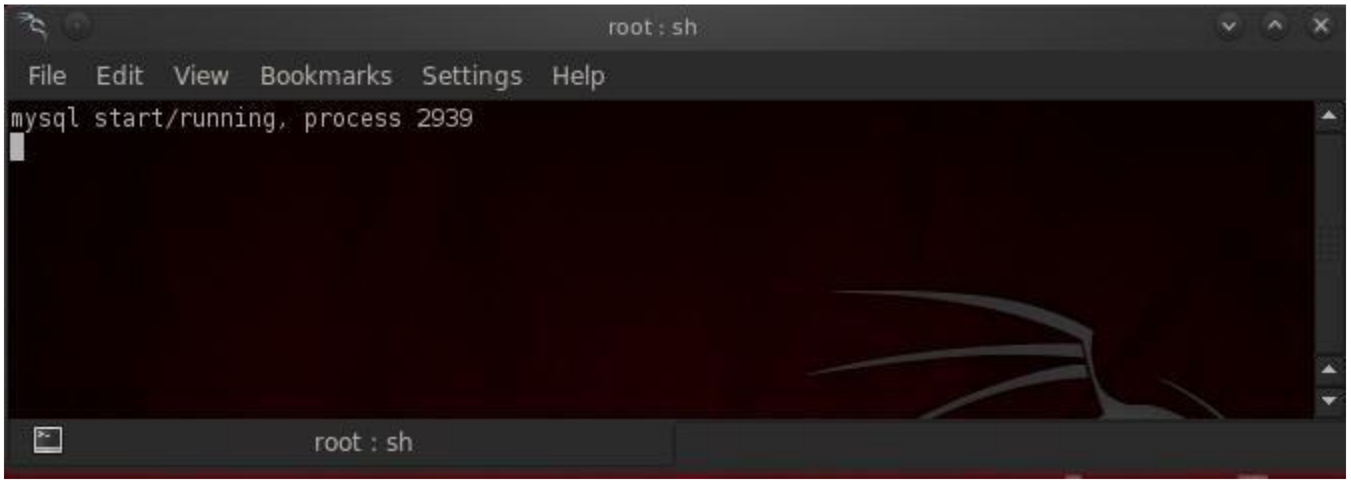
Other popular Content Management Systems(CMS) such as Joomla, Drupal, and Ruby on Rails all use MySQL. You get the idea. If you want to develop or attack web applications, you should know MySQL. So, let's get started.

Step 1 Start MySQL

Luckily, BackTrack has MySQL already installed (if you are using another distribution, you can usually download and install MySQL from the software repository) and has a graphical start and stop. Let's start our MySQL service.



When we do so, we should see a screen like that below pop up briefly and then disappear.



```
mysql start/running, process 2939
```

Step 2 Logging in to MySQL

Now that our MySQL service is started, we can begin to use it. First, we need to authenticate ourselves by logging in.

Open a terminal and type:

- **mysql -u root -p**

You will be prompted for your password, which is "toor" in BackTrack. It may be different on other systems. Please note that although the username and password for MySQL is the same as the BackTrack username and password, that is not necessarily so on other distributions of Linux and MySQL. Usernames and passwords for the operating system (here is it Linux Ubuntu) and MySQL are separate and distinct.

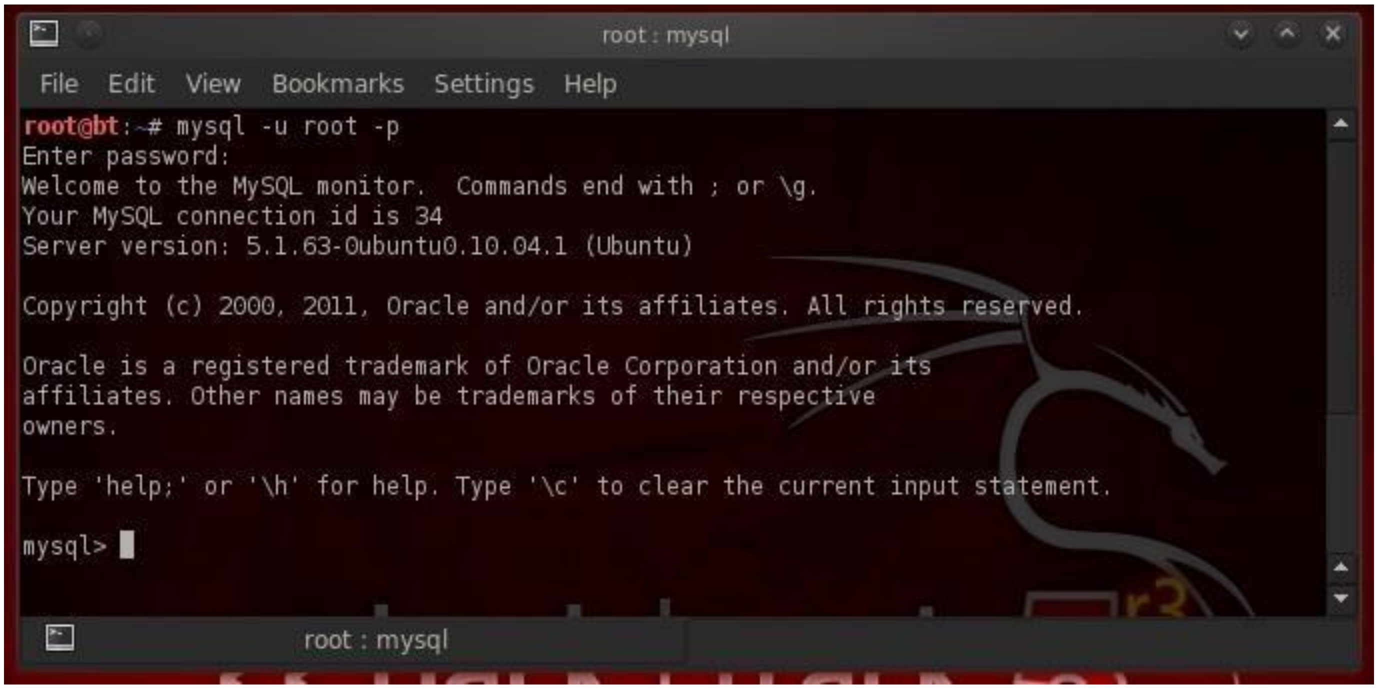
This syntax, mysql -u <username> -p, works if we are trying to access a MySQL database on our localhost. This command defaults to using the MySQL instance on the localhost, if not given a hostname or IP address. For remote access (and that will likely be the case as a hacker), we need to provide the hostname or IP address of the system that is hosting the MySQL database, such as:

- **mysql -u root -p 192.168.1.101**

This will connect us to the MySQL instance at 192.168.1.101 and prompt us for a password.

This opens up the MySQL command line interface that provides us with the **mysql>** prompt. Like Microsoft's SQL Server, MySQL has a GUI interface both native (MySQL Workbench) and third party (Navicat and TOAD for MySQL). Let's look athe command line interface first and then will will advance to the GUI interface

As a hacker, the command line interface may be our best opportunity for exploiting the MySQL database, so we should focus on it. It's unlikely that as an unauthorized entrant to the database you will be presented with an easy to use GUI.



```
root@bt:~# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 34
Server version: 5.1.63-0ubuntu0.10.04.1 (Ubuntu)

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> 
```

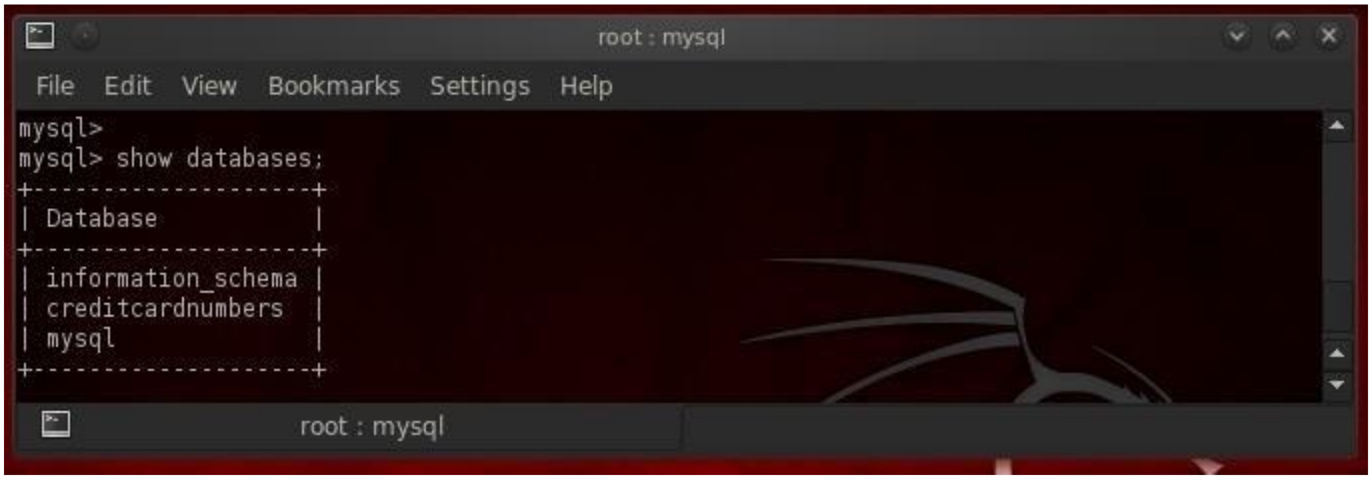
Please note that this screen reminds us that all commands end in " ;"or "\g" (unlike Microsoft's SQL Server) and that we can get help by typing help; or \h.

As we are now logged as the systadmin (root), we can navigate unimpeded through the database. If we had logged in as a regular user, our navigation would be limited by the permissions provided us by the system administrator for that user.

Step 3 Show Databases

Now that we are logged in to the MySQL database as root, our next step is to find out whether there are any databases worth hacking. The command to find databases is:

- **show databases;**



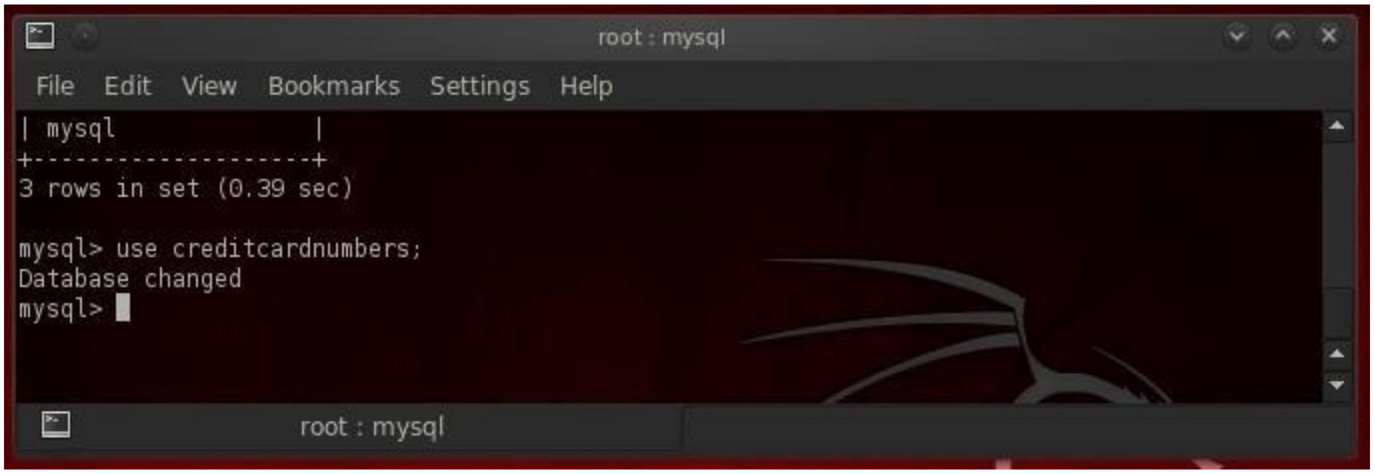
```
mysql>
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| creditcardnumbers  |
| mysql              |
+--------------------+
```

Ah Hah! We found a database worth exploring here named "*creditcardnumbers*".

Step 4 Connect to a Database

Once we have logged into the MySQL instance, our next step is to connect to a particular database. In MySQL, like other database management systems, we can connect to the database we are interested in by typing **use** <databasename>. Since we now know that the database we are interested in is named "creditcardnumbers", we simply type:

- **use creditcardnumbers;**



```
| mysql              |
+--------------------+
3 rows in set (0.39 sec)

mysql> use creditcardnumbers;
Database changed
mysql> 
```

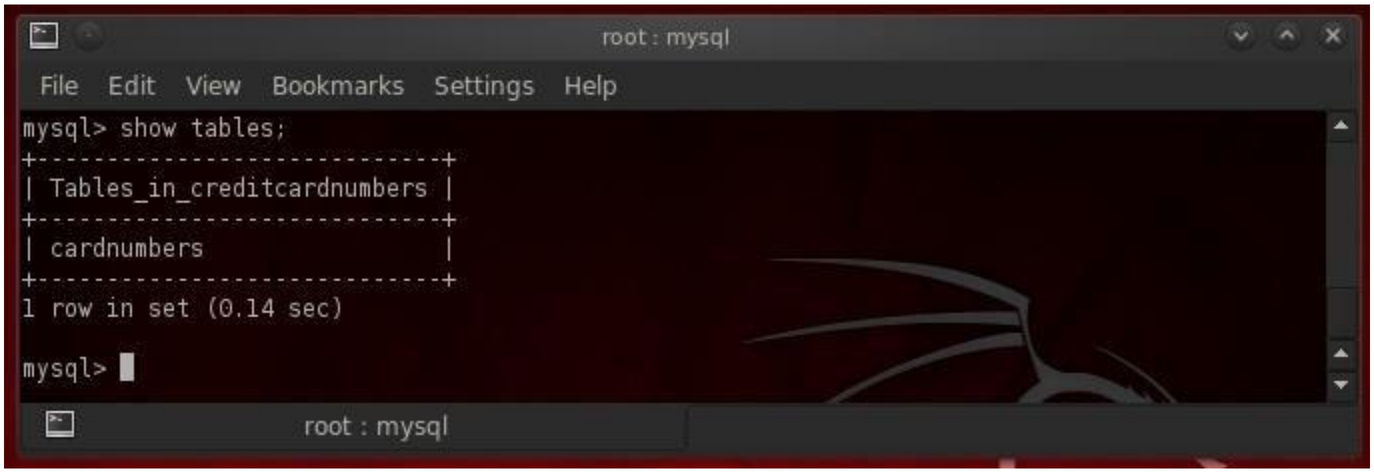
As you can see, MySQL responds with "***Database changed***", indicating that we are now connected to the "creditcardnumbers" database.

Of course, I hope it goes without saying, that you should use the appropriate database name in place here of "creditcardnumbers". Its unlikely that a database admin will be so kind and accommodating as to name a database with such an easily recognizable name, so you may need to do a bit of exploring to find the database of interest.

Step 5 Finding the Tables

Now we are connected to the "creditcardnumbers" database and we can do a bit of exploring to see what might be in that database. We can find out what tables are in this database by typing:

- **show tables;**



```
mysql> show tables;
+---------------------------+
| Tables_in_creditcardnumbers |
+---------------------------+
| cardnumbers               |
+---------------------------+
1 row in set (0.14 sec)

mysql> 
```

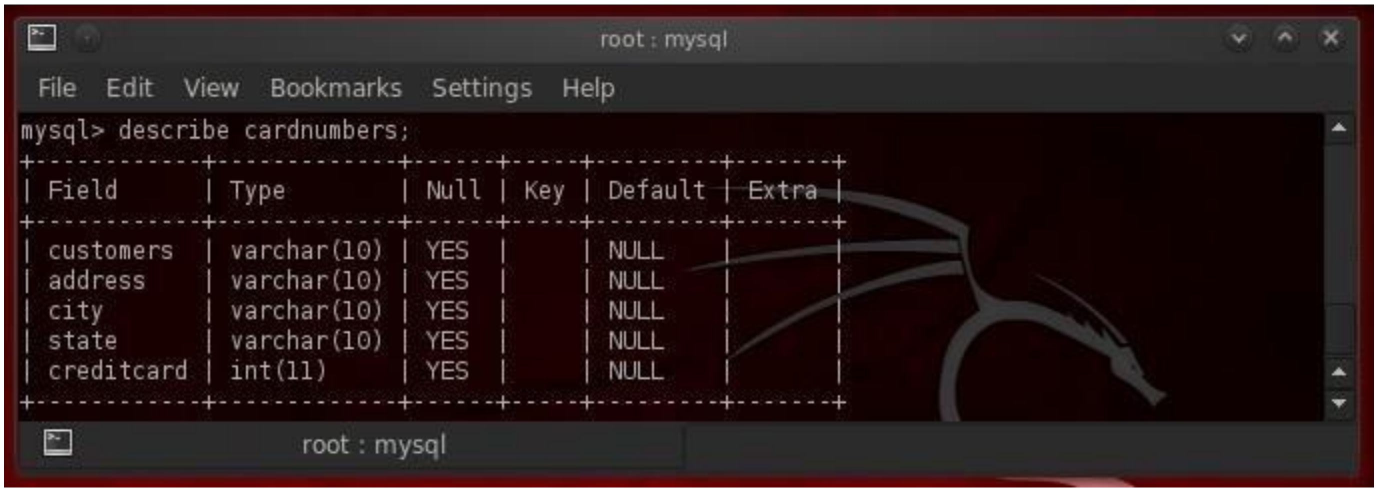
In the screenshot above, we can see that this database has just one table in it called "cardnumbers". Generally, databases will have numerous tables in them, but we are fortunate here as we can focus our attention on this single table to extract the hackers "golden fleece"!

Step 6 Describe the Table to Discover Its Structure

Since we can focus our efforts on this single table, we will need to understand the structure of that table. In subsequent tutorials--when we are hacking this database--we will see that understanding the structure is critical to a successful hack.

We can see the structure of the table by typing:

- **describe cardnumbers;**



```
mysql> describe cardnumbers;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| customers  | varchar(10) | YES  |     | NULL    |       |
| address    | varchar(10) | YES  |     | NULL    |       |
| city       | varchar(10) | YES  |     | NULL    |       |
| state      | varchar(10) | YES  |     | NULL    |       |
| creditcard | int(11)     | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
```

MySQL responds with the critical infornation on the structure of our table of interest. We can see each of the fields and their data type (varchar or int), whether it will accept NULL's, the key, the default values and extra.

Step 7 SELECT Data

To actually see the data in the table, we can use the SELECT command. The SELECT command requires to know:

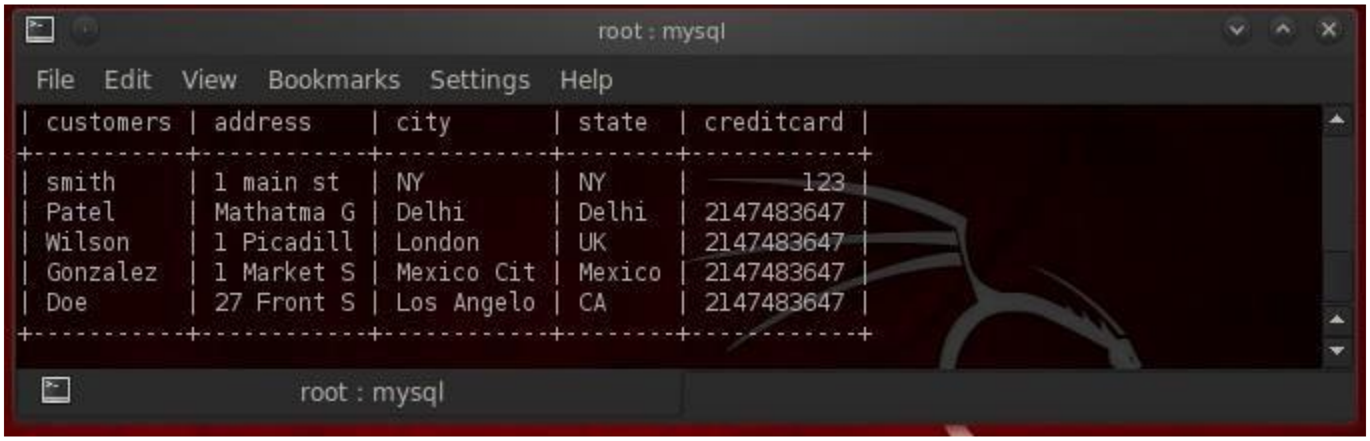
1. The table we want to view
2. The columns within that table we want to view

Using the format:

- **SELECT <columns> FROM <table>**

As a handy shortcut if we want to see data from all the columns, we can use the asterix ("*") as a wildcard instead of typing out every single column name. So, to see a dump of all the data from the cardnumbers table, we type:

- **SELECT * FROM cardnumbers;**



```
| customers | address     | city        | state  | creditcard |
+-----------+-------------+-------------+--------+------------+
| smith     | 1 main st   | NY          | NY     |        123 |
| Patel     | Mathatma G  | Delhi       | Delhi  | 2147483647 |
| Wilson    | 1 Picadill  | London      | UK     | 2147483647 |
| Gonzalez  | 1 Market S  | Mexico Cit  | Mexico | 2147483647 |
| Doe       | 27 Front S  | Los Angelo  | CA     | 2147483647 |
+-----------+-------------+-------------+--------+------------+
```

As we can see, MySQL displayed all the information from the cardnumbers table to our screen.

Step 8 Export Data

Now that we know where the data is, we need to export it so that we can use it. MySQL has a command called **mysqldump**. Generally, it is used to create a backup copy of the data. You can run it from any command prompt, but you will need:

1. A username (root)
2. The password for that username (toor)
3. The name of the database you want data from (creditcardnumbers)
4. The table within the database you want (cardnumbers)
5. The directory you want to dump to (/tmp)

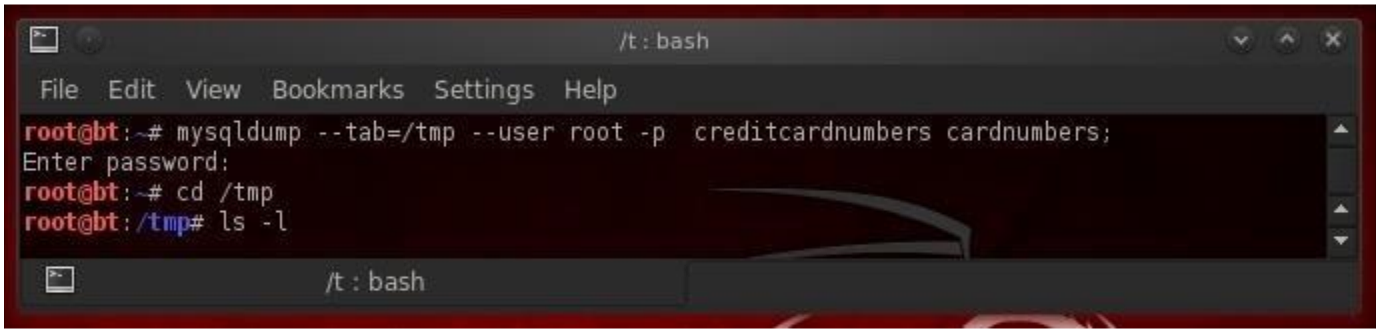
So, to "dump" the data from command line we simply type:

- **mysql --tab = /tmp --user root -p creditcardnumbers cardnumbers;**

This will send the data to the directory we designated, in this case /tmp.

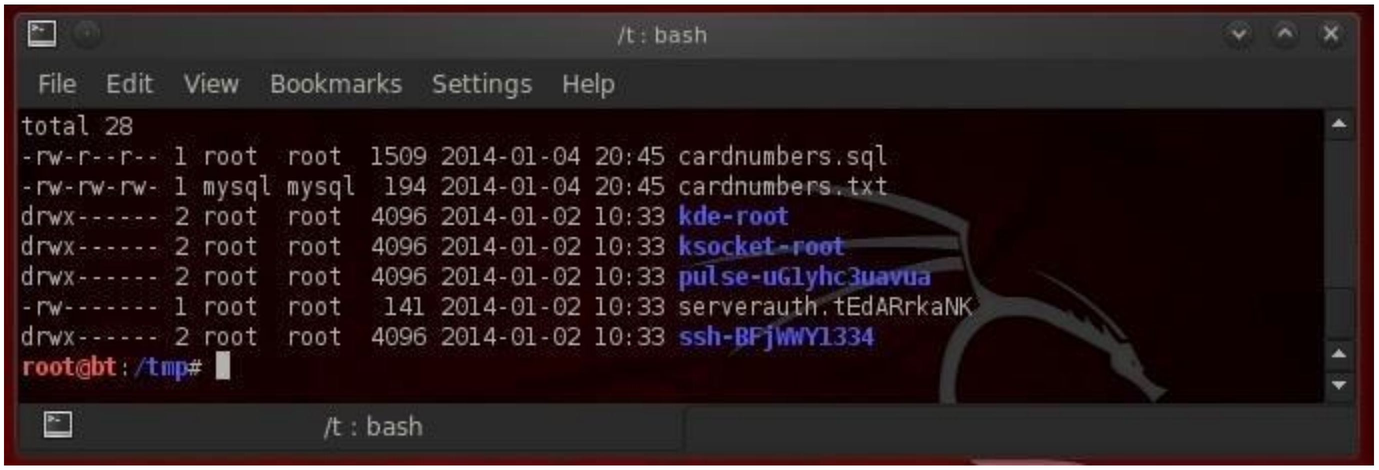
Success!

As we can see below (after we changed to the /tmp directory and then listed that directory) we have created two files, **cardnumbers.sql** and **cardnumbers.txt**. The first, cardnumbers.sql, contains a script to create the table necessary to hold the data and the second, cardnumbers.txt, contains the data.



```
root@bt:~# mysqldump --tab=/tmp --user root -p  creditcardnumbers cardnumbers;
Enter password:
root@bt:~# cd /tmp
root@bt:/tmp# ls -l
```

Now, we have successfully acquired the key and valuable information from this database, essentially having found the golden fleece of hacking!



```
total 28
-rw-r--r-- 1 root  root  1509 2014-01-04 20:45 cardnumbers.sql
-rw-rw-rw- 1 mysql mysql  194 2014-01-04 20:45 cardnumbers.txt
drwx------ 2 root  root  4096 2014-01-02 10:33 kde-root
drwx------ 2 root  root  4096 2014-01-02 10:33 ksocket-root
drwx------ 2 root  root  4096 2014-01-02 10:33 pulse-uGlyhc3uavua
-rw------- 1 root  root   141 2014-01-02 10:33 serverauth.tEdARrkaNK
drwx------ 2 root  root  4096 2014-01-02 10:33 ssh-BFjWWY1334
root@bt:/tmp#
```

Since MySQL is SO critical to web apps, we will be spending a few tutorials on understanding it, then how to find it and finally how to hack it, so keeps coming back my greenhorn hackers for more adventures in Hackerland.

Creating a Secure Tunnel to MySQL

In my continuing effort to build your basic Linux skills for hacking, I want to show you how to build a secure "tunnel" to MySQL.

Of course, the techniques I use here could be used for any application, but since MySQL is such a critical app on most Linux installations, and since un-encrypted sessions to your MySQL database server could easily be sniffed and confidential information exposed, we'll use our database server as our example in this tutorial.

This is not to say that an encrypted tunnel is foolproof from being hacked. As with anything, it can be hacked, but it makes it many times more difficult. If we leave the data un-encrypted, any script kiddie with a sniffer can see and grab our traffic to our database.

We'll be using SSH or Secure Shell to encrypt our traffic. Every Linux distribution has a SSH server and client built in, unlike Windows where you will need to download one of many SSH clients such as PuTTY. Our BackTrack has BSD OpenSSH built-in, so don't need to download and install anything to build a secure connection between our client and server.

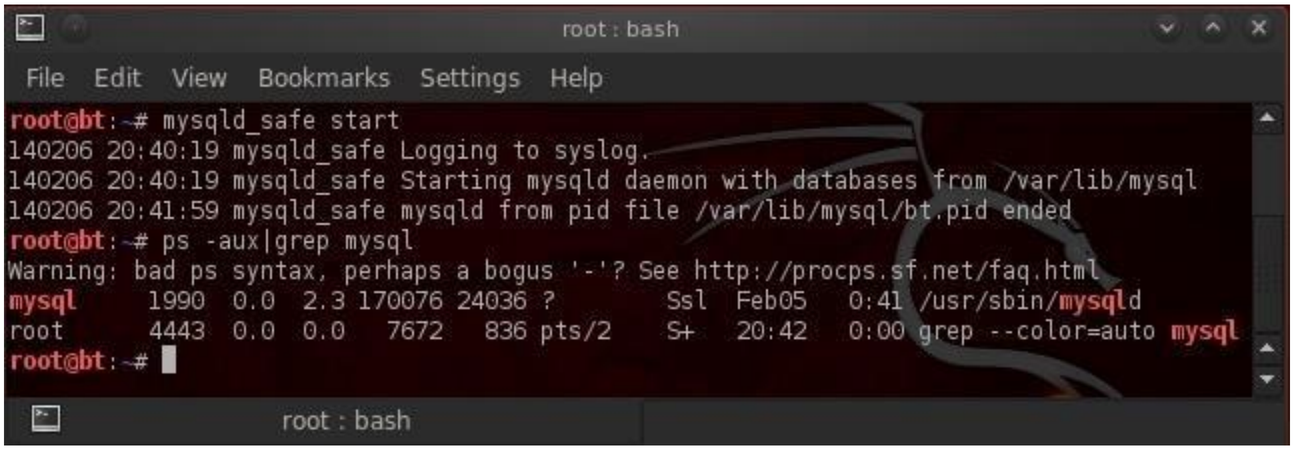
Like so many other applications in Linux, SSH operates on a server/client architecture. To successfully connect, you must have both the server and the client running.

Step 1 Open BackTrack & Start MySQL

MySQL has the capability of using SSH, but you must configure and compile it to do so. Since the default configuration of MySQL, such as ours on BackTrack, does not have SSH built-in, we need to do a workaround using the SSH built into the operating system and then connecting to MySQL. This will create an encrypted "tunnel" to our database, so that hackers can't view our transmissions back and forth to the database.

In our example here, we'll be connecting between two BackTrack installations. I have shown you how to start MySQL from the GUI in the previous Linux basics guide, but in many distributions of Linux you won't have that luxury, so let's start MySQL from the command line.

- **bt > mysql_safe start**



```
root@bt:~# mysqld_safe start
140206 20:40:19 mysqld_safe Logging to syslog.
140206 20:40:19 mysqld_safe Starting mysqld daemon with databases from /var/lib/mysql
140206 20:41:59 mysqld_safe mysqld from pid file /var/lib/mysql/bt.pid ended
root@bt:~# ps -aux|grep mysql
Warning: bad ps syntax, perhaps a bogus '-'? See http://procps.sf.net/faq.html
mysql     1990  0.0  2.3 170076 24036 ?        Ssl  Feb05   0:41 /usr/sbin/mysqld
root      4443  0.0  0.0   7672   836 pts/2    S+   20:42   0:00 grep --color=auto mysql
root@bt:~# 
```

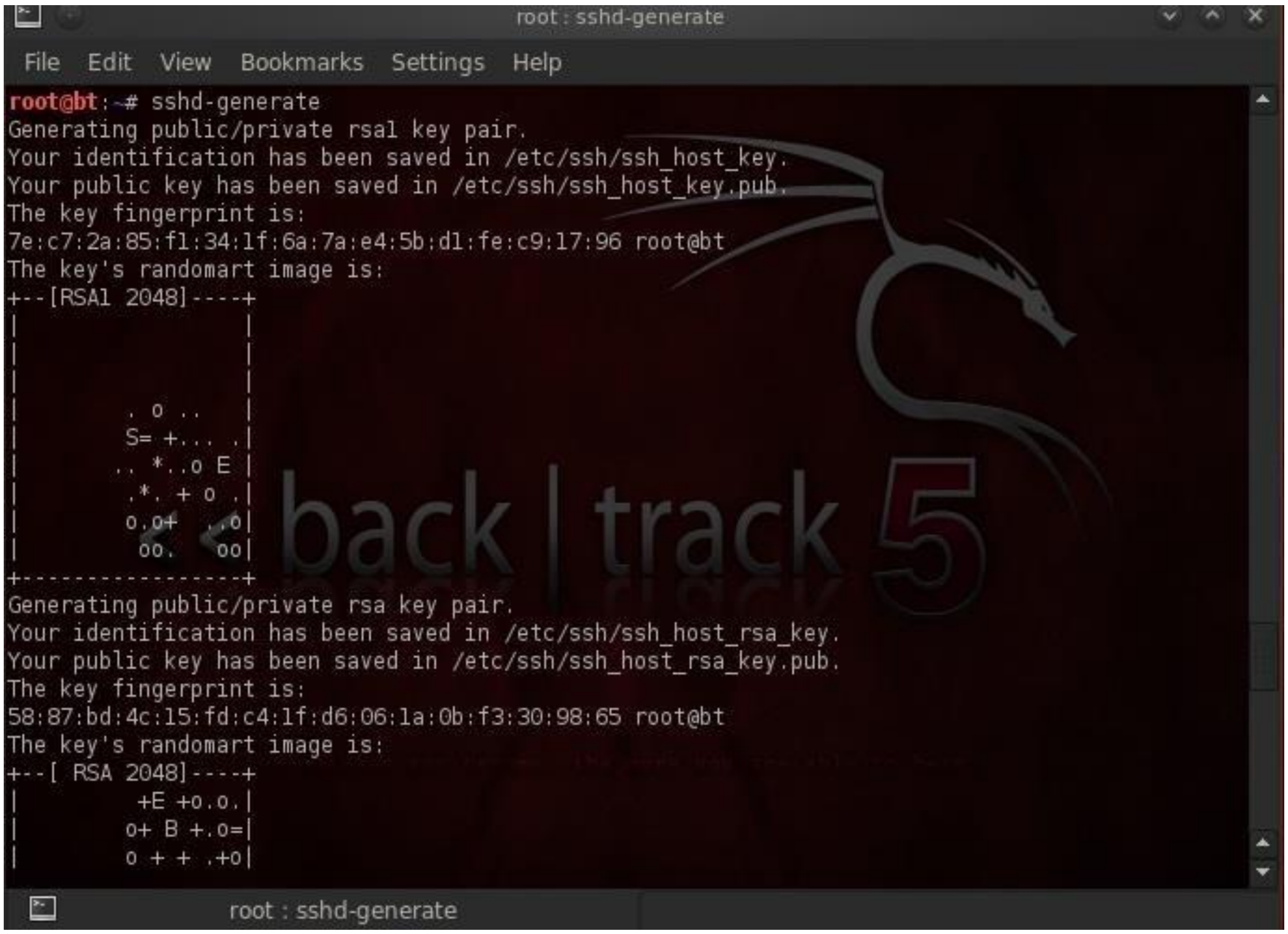
Now, let's make certain that it started by checking our processes and "grepping" for mysql.

- **bt > ps aux | grep mysql**

Step 2 Generate Keys

In order for SSH to create its encrypted tunnel, it must first generate a key pair, a private key and a public key. These two keys will be used to encrypt and then decrypt the traffic over the tunnel. We can do this by typing:

- **bt >sshd-generate**



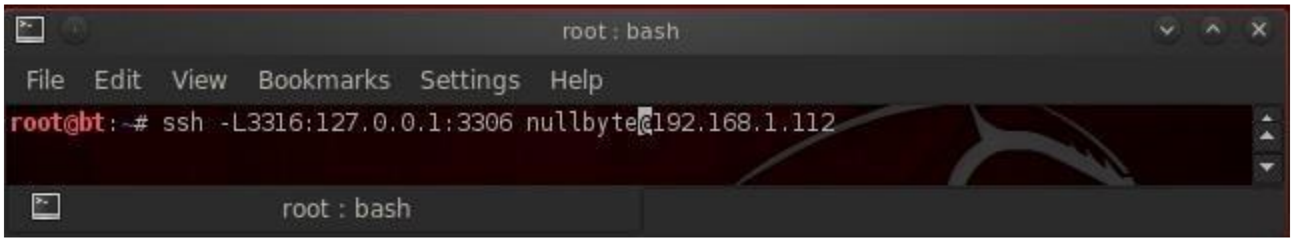
```
root@bt:~# sshd-generate
Generating public/private rsa1 key pair.
Your identification has been saved in /etc/ssh/ssh_host_key.
Your public key has been saved in /etc/ssh/ssh_host_key.pub.
The key fingerprint is:
7e:c7:2a:85:f1:34:1f:6a:7a:e4:5b:d1:fe:c9:17:96 root@bt
The key's randomart image is:
+--[RSA1 2048]----+
|                 |
|                 |
|                 |
|         . o ..  |
|         S= +... .|
|        .. *..o E |
|         .*. + o .|
|         o.o+  ..o|
|          oo.   oo|
+-----------------+
Generating public/private rsa key pair.
Your identification has been saved in /etc/ssh/ssh_host_rsa_key.
Your public key has been saved in /etc/ssh/ssh_host_rsa_key.pub.
The key fingerprint is:
58:87:bd:4c:15:fd:c4:1f:d6:06:1a:0b:f3:30:98:65 root@bt
The key's randomart image is:
+--[ RSA 2048]----+
|          +E +o.o.|
|         o+ B +.o=|
|         o + + .+o|
```

As we can see, SSH has generated a key pair from which it will now be able to generate our secure tunnel. I have created a user named "nullbyte" on this server that we will use to connect to this machine.

Step 3 Start SSH

From the **client** machine, we can now connect to that SSH server by typing:

- **ssh -L3316:127.0.0.1:3306 nullbyte@192.168.1.112**

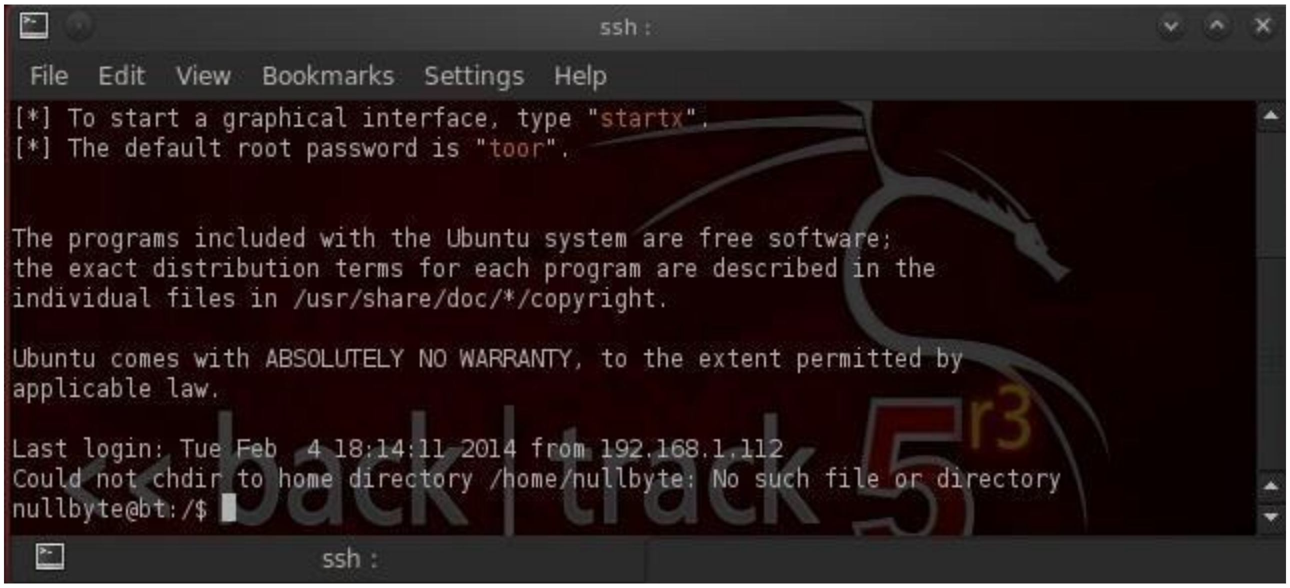


```
root@bt:~# ssh -L3316:127.0.0.1:3306 nullbyte@192.168.1.112
```

Here's a breakdown of what's in the command above.

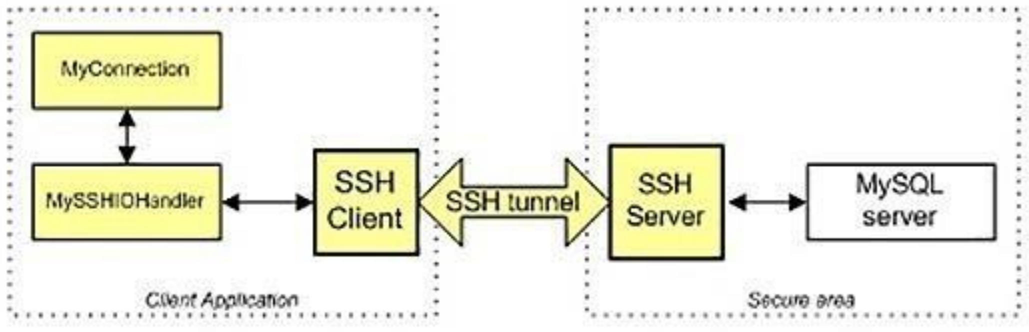
- **ssh** is the client command
- **-L3316** listens on port 3316
- **127.0.0.1** is the location of the SSH client daemon on the client machine
- **3306** is the default port of MySQL that we want the tunnel to on the server machine
- **nullbyte** is a user on the operating system on the server
- **192.168.1.112** the IP address of the MySQL server

When we execute this command, we get a connection to the remote machine on nullbyte's account as shown below.



```
[*] To start a graphical interface, type "startx".
[*] The default root password is "toor".


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

Last login: Tue Feb  4 18:14:11 2014 from 192.168.1.112
Could not chdir to home directory /home/nullbyte: No such file or directory
nullbyte@bt:/$ 
```

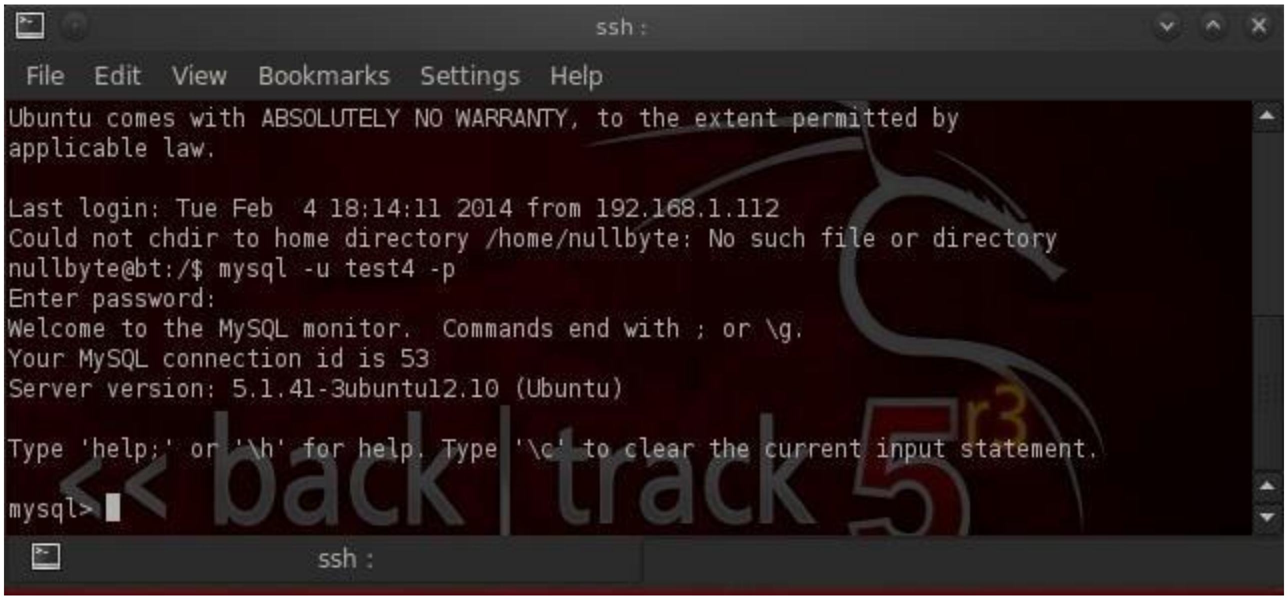
What we have done here is to connect to the SSH client daemon on our client system that then connects via port 3331 to the SSH server that then connects to port 3306 to connect to MySQL.



Step 4 Connect to MySQL Securely

Now that we are securely connected to the server that contains the MySQL database, we can now login to the database over this tunnel. I have a MySQL user named "test4" (not the OS user—we connected via an OS user and we connect to MySQL via a MySQL user) on that database. Let's now login to that user's account.

- **mysql -u test4 -p**

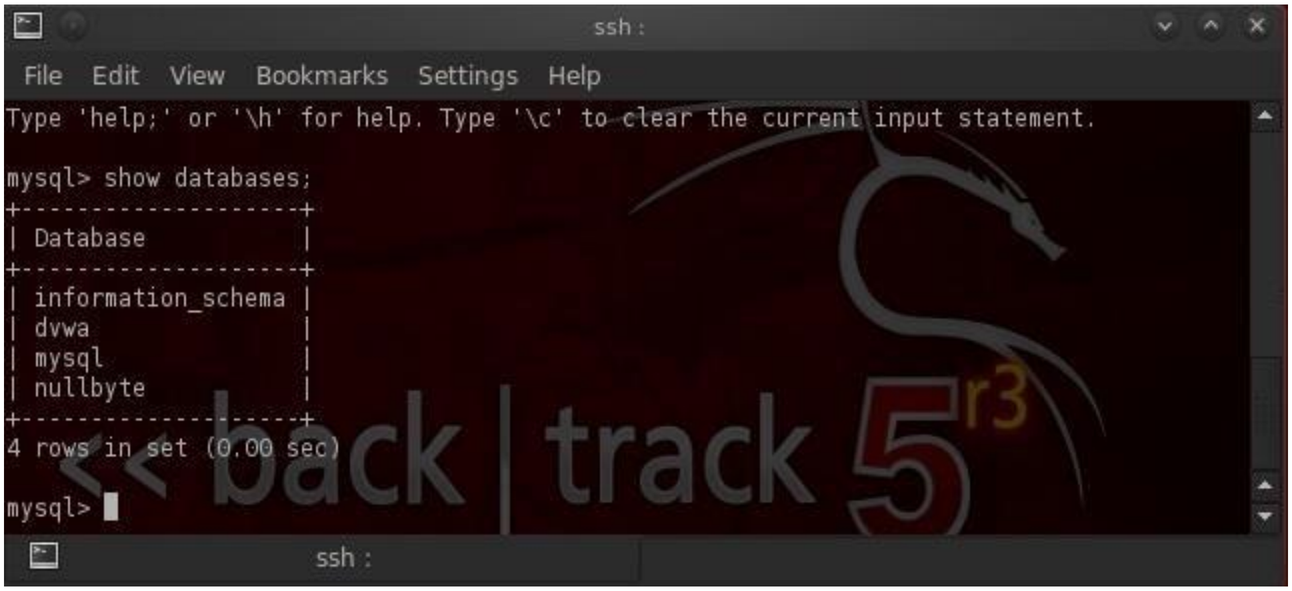


```
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

Last login: Tue Feb  4 18:14:11 2014 from 192.168.1.112
Could not chdir to home directory /home/nullbyte: No such file or directory
nullbyte@bt:/$ mysql -u test4 -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 53
Server version: 5.1.41-3ubuntu12.10 (Ubuntu)

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

As you can see above, we have successfully connected to MySQL over this tunnel! All of the traffic between us and the remote database server is encrypted.

To make certain that we connected to the remote server and not our local database, I created a database on the remote server named "nullbyte".

Let's issue the "show databases;" command from the MySQL prompt.



```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| dvwa               |
| mysql              |
| nullbyte           |
+--------------------+
4 rows in set (0.00 sec)

mysql> 
```

As you can see, we have connected to the remote database server as it has the "nullbyte" database.

Stdin, Stdout, & Stderror

Along the way, I realized that I've failed to provide you with some basic background material on the **stdin**, **stdout**, and **stderror**.

In human language, these are often referred to as standard input (stdin), standard output (stdout), and standard error (stderror). These represent how and where Linux sends the output from your commands (stdout), where it receives its input (stdin), and where it sendsits error messages (stderror).

Since you've now been using Linux for awhile, you realize that both standard output and standard error are sent to your computer screen. Both as the name implies, this is the standard place and not necessarily the only place. Linux let's us define where our output and error messages should go.

Before we go any further, let's take a moment to define some terms.

Standard Output (1)

Whenever you complete a command, it must know where to send the output. You might want to send it to a file, a printer, the screen, etc. The default for standard output is the computer screen. Standard output is often referred to as stdout or simply use the numeric representation of 1.

Standard Input (0)

Standard input is where the program or command gets the information it needs. By default, in Linux this is the keyboard, but can be a file, etc. Standard input is often referred to as stdin or simply represented by the numeric representation of 0.

Standard Error (2)

When we make a mistake or our program throws an error, it send the error message to stanadard error. By default, this is our computer screen. Standard error is often referred to as stderror or simply represented by the numeral 2.

When we want to direct any of these three from the command line or a script, we use the numeric representation of each, 0 for stdin, 1 for stdout, and 2 for stderr.

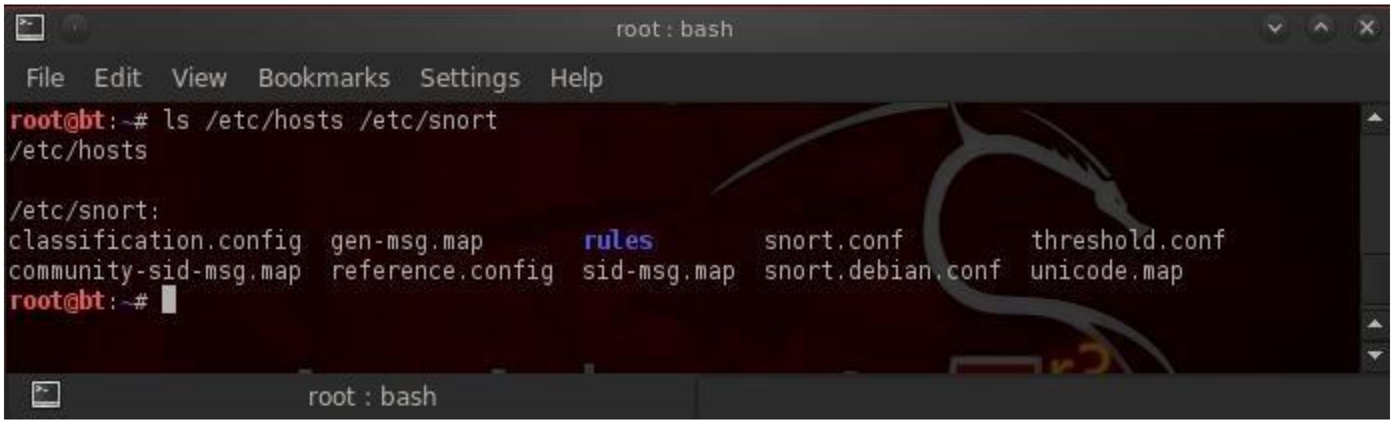
Step 1 List Two Directories

To demonstrate how we can use and manipulation these I/O streams, let's do a listing of two different directories, */etc/hosts* and */etc/snort*.

In Linux, you can do listings of more than one directory at a time. The */etc/snort* directory is where our configuration file for snort resides and */etc/hosts* is a directory where we can set static name resolution in Linux (I'll do a new Linux tutorial on DNS and name resolution in Linux soon).

If we wanted to see the two directories, we could type:

- **ls /etc/hosts /etc/snort**

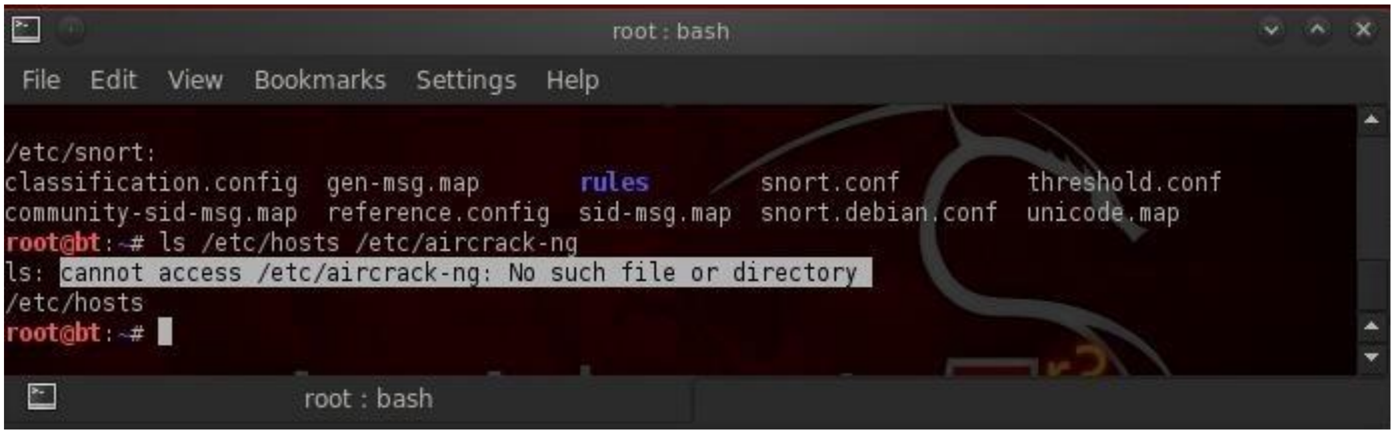


```
root@bt:~# ls /etc/hosts /etc/snort
/etc/hosts

/etc/snort:
classification.config  gen-msg.map      rules        snort.conf         threshold.conf
community-sid-msg.map  reference.config  sid-msg.map  snort.debian.conf  unicode.map
root@bt:~# 
```

As you can see, the listing comes back to us by the standard output to our computer screen showing us the listing of both directories.

Now, let's try the same thing, but this time let's list a directory that doesn't exist, such as */etc/aircrack-ng*.

- **ls /etc/hosts /etc/aircrack-ng**



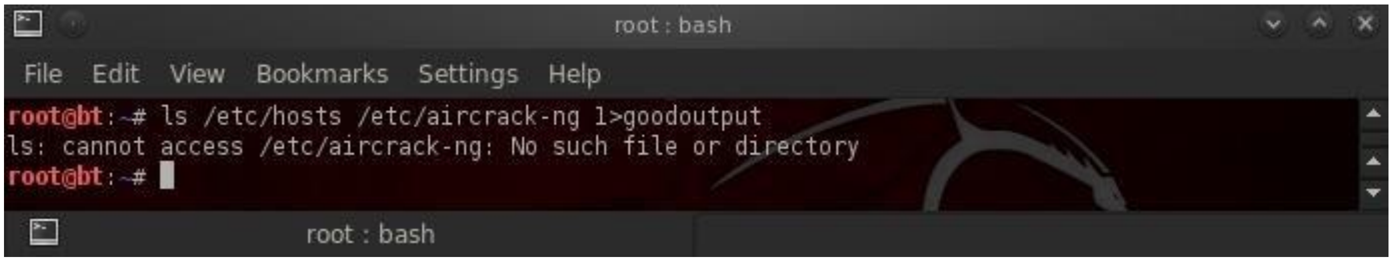
```
/etc/snort:
classification.config  gen-msg.map       rules       snort.conf          threshold.conf
community-sid-msg.map  reference.config  sid-msg.map  snort.debian.conf  unicode.map
root@bt:~# ls /etc/hosts /etc/aircrack-ng
ls: cannot access /etc/aircrack-ng: No such file or directory
/etc/hosts
root@bt:~#
```

As you can see, our BASH shell comes back with two outputs, the standard output from *etc/hosts* and the standard error from the non-existent directory.

Step 2 Send Standard Output to a File

Next, let's suppose that we want to separate our standard output from our standard error. Imagine we're running a script where we don't want to see our output messages until after the script has run, but we need to see error messages on our screen immediately. We could rewrite our command as:

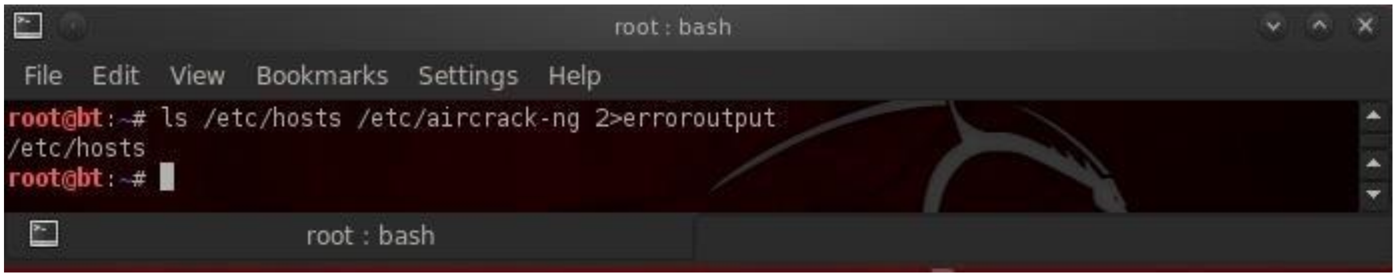
- **ls /etc/hosts /etc/aircrack-ng 1>goodoutput**



```
root@bt:~# ls /etc/hosts /etc/aircrack-ng 1>goodoutput
ls: cannot access /etc/aircrack-ng: No such file or directory
root@bt:~#
```

Let's now imagine just the reverse of our previous scenario where instead we want to see our output on screen, but store our error messages to a separate file for viewing later. We could write:

- **ls /etc/hosts /etc/aircrack-ng 2>erroroutput**



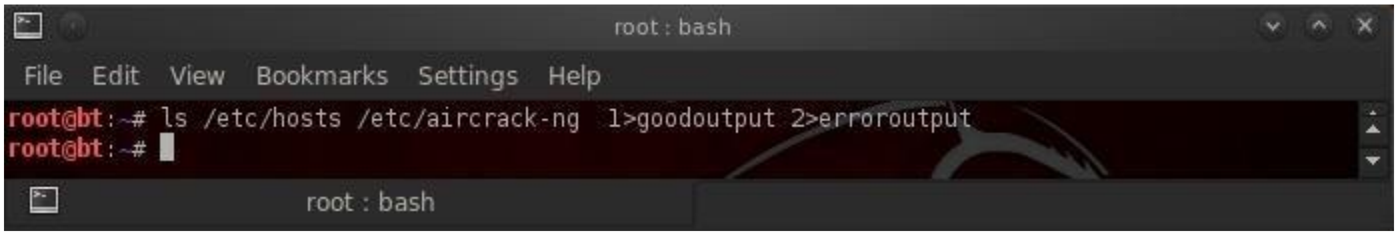
```
root@bt:~# ls /etc/hosts /etc/aircrack-ng 2>erroroutput
/etc/hosts
root@bt:~#
```

Now, after the command has been run, we can go back and cat the *erroroutput* file to view any possible error messages.

Step 3 Send Standard Output & Standard Error to Separate File

Now, let's imagine a script where we want both our standard output and standard error to be directed to separate files for viewing later. We can type:

- **ls /etc/hosts /etc/aircrack-ng 1>goodoutput 2>erroroutput**



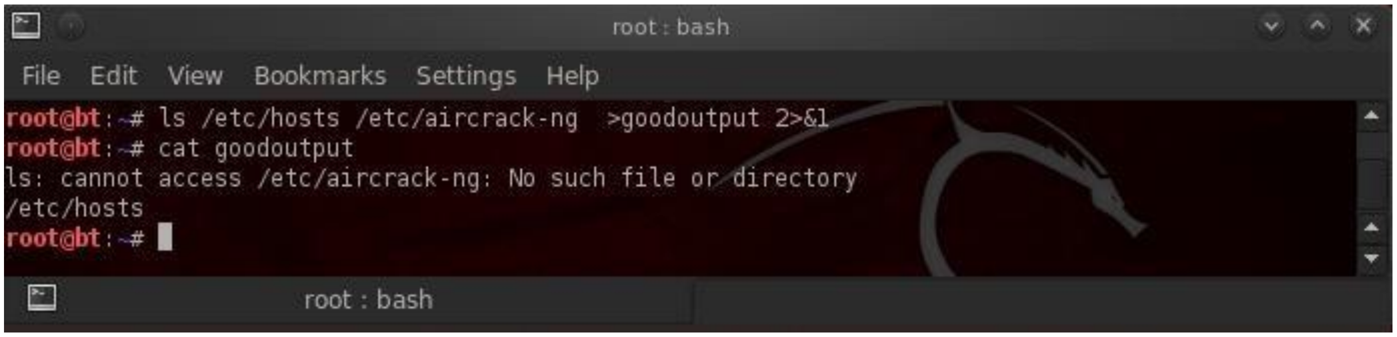
```
root@bt:~# ls /etc/hosts /etc/aircrack-ng  1>goodoutput 2>erroroutput
root@bt:~#
```

Notice that nothing comes back to our screen, neither standard output nor standard error.

Step 4 Send Both Standard Output & Standard Input to Same File

Finally, what if we wanted both standard error and standard output to be written to same file? We could type:

- **ls /etc/hosts /etc/aircrack-ng >goodoutput 2>&1**



```
root@bt:~# ls /etc/hosts /etc/aircrack-ng  >goodoutput 2>&1
root@bt:~# cat goodoutput
ls: cannot access /etc/aircrack-ng: No such file or directory
/etc/hosts
root@bt:~# 
```

Notice that I did not use the 1 before the *>goodoutput* as BASH defaults to stdout if no number is used.

Client DNS

Domain Name System (DNS) is one of those things we seldom think about unless it doesn't work. Then, it can be very frustrating when we attempt to navigate to a website and we get that frustrating error message.

DNS enables us to type in a domain name in our browser, such as wonderhowto.com, rather than a numerical IP address of the site we are trying to reach. In its simplest form, DNS simply translates domain names into IP addresses, making our lives much simpler. Can you imagine trying to remember all of the IP addresses of the hundreds of sites you visit daily?

For most of us working in Linux, we have two DNS concerns. First, as a client we need to access DNS services to translate our domain names into IP addresses. Second, as a server we need to provide DNS services. Here, I will limit myself to managing DNS from a **client** perspective and leave providing DNS services to another tutorial.

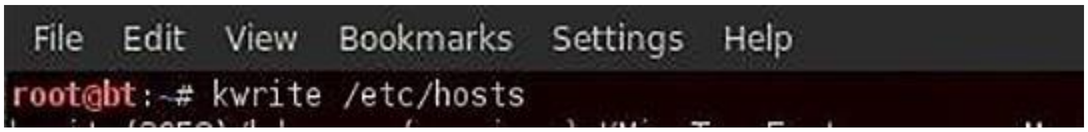
It's important to remind you here that in Linux, nearly everything is a file, and configuration is usually through the editing of a simple text file. This rule certainly applies to DNS.

Step 1 /Etc/Hosts

In Linux, we have what is referred to as a "hosts" file. It's found where nearly all the configuration files are in the */etc* directory, so */etc/hosts*. This hosts file acts similarly to DNS, but it is static. This means that it's not updated like DNS is. The hosts file is the simplest and fastest method for mapping hostnames to IP addresses, but also the most time consuming.

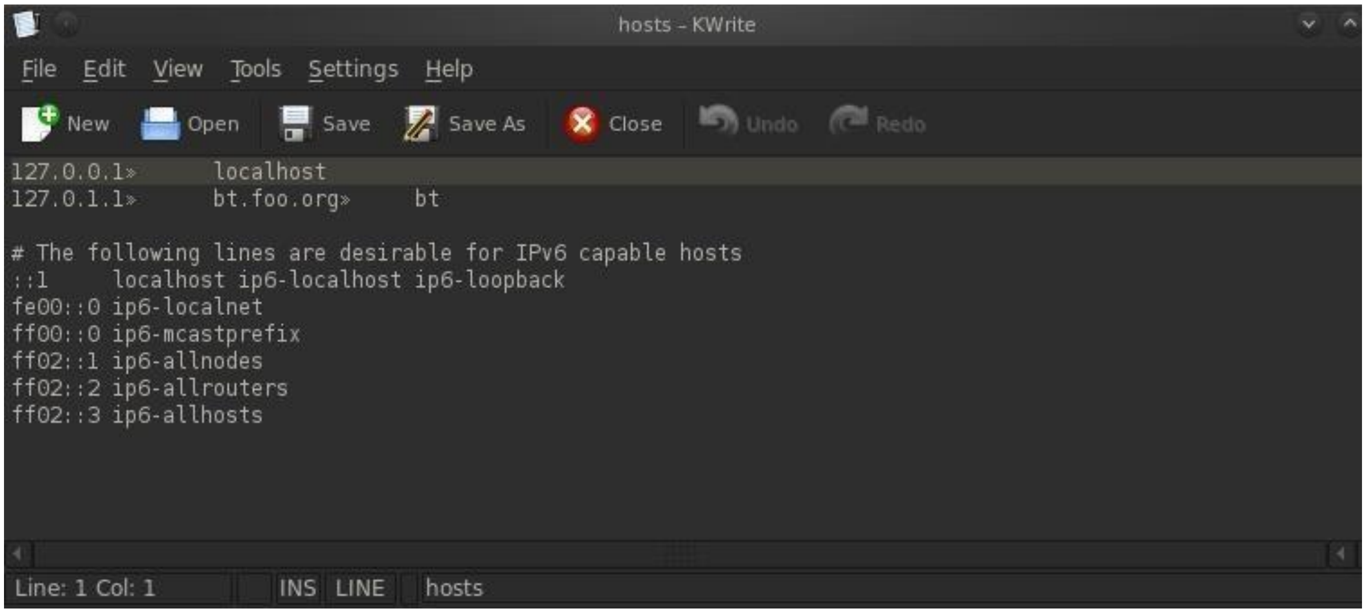
Let's look at the */etc/hosts* file in BackTrack. Type:

- **bt > kwrite /etc/hosts**



```
root@bt:~# kwrite /etc/hosts
```

This will open the following file. Note that the default configuration in BackTrack has just the entries for localhost at 127.0.0.1 and then some notes on IPv6.


hosts - KWrite
File Edit View Tools Settings Help
New Open Save Save As Close Undo Redo

```
127.0.0.1»      localhost
127.0.1.1»      bt.foo.org»     bt

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts
```

Line: 1 Col: 1
INS LINE hosts

We could add additional lines to this file to provide simple name resolution services. If we wanted to resolve the word "hacker" to a system on our internal network, we could simply add a line to our hosts file, such as:

- **192.168.116.7 hacker**

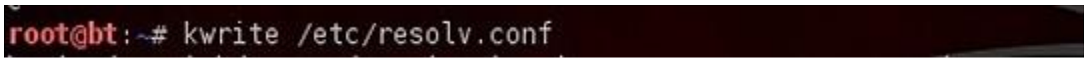
When we save our */etc/hosts* and type "hacker" into our browser, we will be directed to the IP 192.168.117.7.

Step 2 /Etc/resolv.conf

The primary file for pointing your system to a DNS server is the */etc/resolv.conf*. Please note that the file name is similar to the English word resolve, but without the "e" at the end. It is here that we tell our system where to look for DNS services.

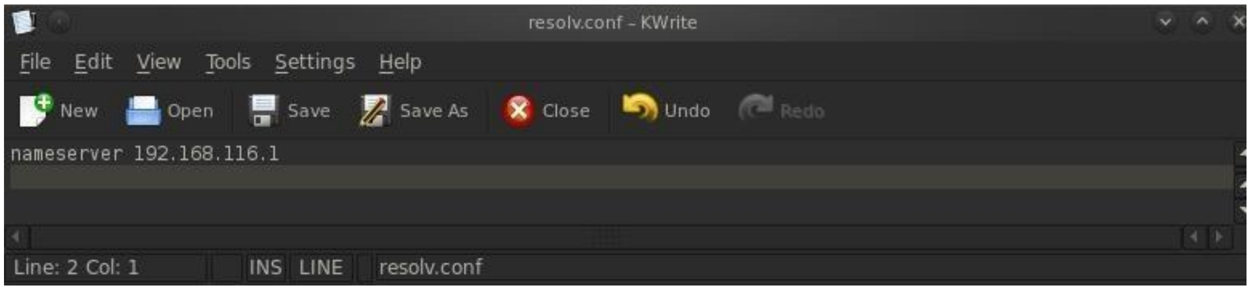
Let's open it with kwrite.

- **bt> kwrite /etc/resolv.conf**



```
root@bt:~# kwrite /etc/resolv.conf
```

When we hit ENTER, kwrite opens the file as below.


resolv.conf – KWrite
File Edit View Tools Settings Help
New Open Save Save As Close Undo Redo
Line: 2 Col: 1 INS LINE resolv.conf

```
nameserver 192.168.116.1
```

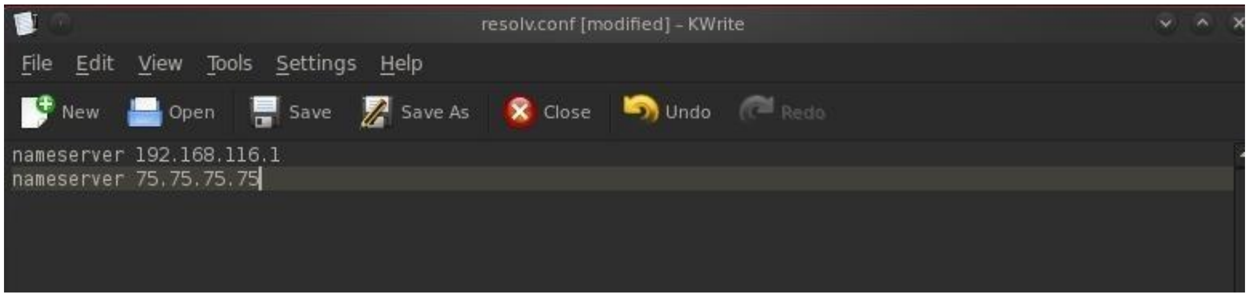
The format of this file is:

- **nameserver IPaddress**

As you can see, my */etc/resolv.conf* is pointing to a DNS server on my local network, 192.168.116.1. I can change this to point to any public DNS server by simply editing and deleting the internal IP address with that of a public DNS server, such as Comcast's at 75.75.75.75.

If you have an internal DNS server, you would probably prefer to use it as it will give you faster responses, but people very often will put in **both** an internal DNS server first and then a public DNS server, second. In this way, your system will check the internal DNS server first and if it doesn't find a listing on that DNS server, it will then progress to the public IP server and will hopefully find it there.

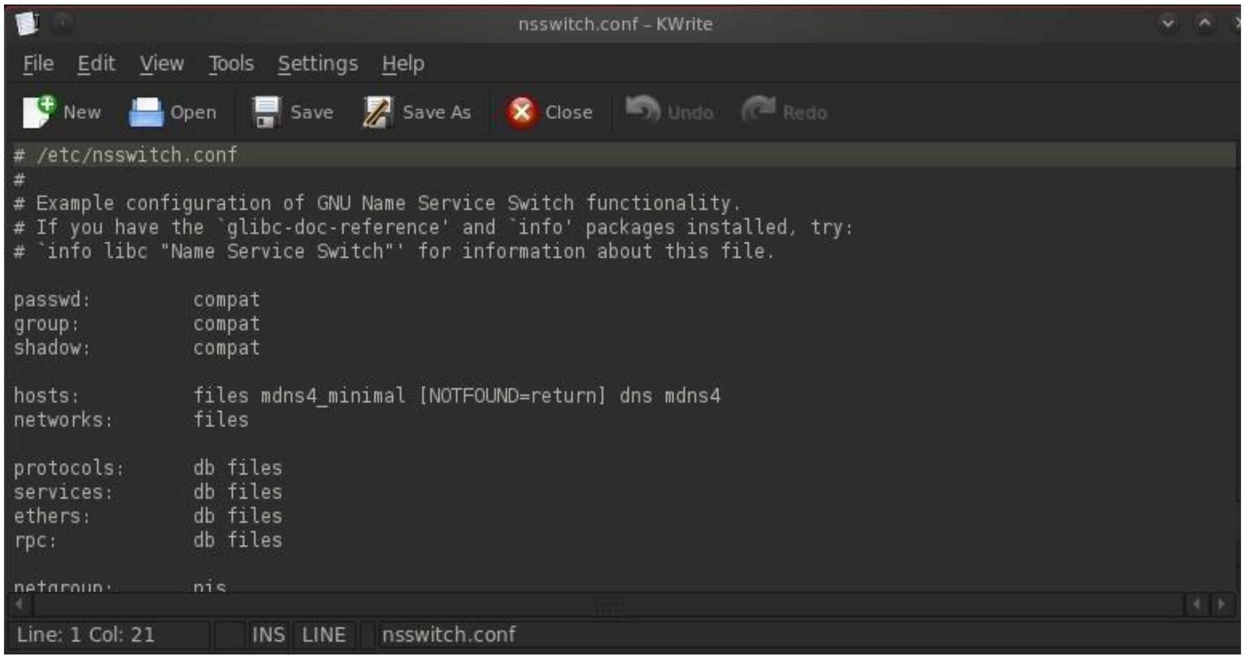
I've edited my */etc/resolv.conf* to include the public DNS server for Comcast at 75.75.75.75. All I do now is **Save** the */etc/resolv.conf* file and my system will look to my internal DNS server first and then to the Comcast public DNS server, if it doesn't find the name in my private DNS server.


resolv.conf [modified] – KWrite
File Edit View Tools Settings Help
New Open Save Save As Close Undo Redo

```
nameserver 192.168.116.1
nameserver 75.75.75.75
```

Step 3 /Etc/nsswitch.conf

Lastly, we have the */etc/nsswitch.conf* file. Here is where we tell our system the order of where to look for name resolution. We have opened it with kwrite and have displayed it below.


nsswitch.conf - KWrite
File Edit View Tools Settings Help
New Open Save Save As Close Undo Redo

```
# /etc/nsswitch.conf
#
# Example configuration of GNU Name Service Switch functionality.
# If you have the `glibc-doc-reference' and `info' packages installed, try:
# `info libc "Name Service Switch"' for information about this file.

passwd:         compat
group:          compat
shadow:         compat

hosts:          files mdns4_minimal [NOTFOUND=return] dns mdns4
networks:       files

protocols:      db files
services:       db files
ethers:         db files
rpc:            db files

netgroup:       nis
```

Note the line that begins with "hosts". This line line tells the system the order of which to search for name resolution. The system will try each in order until it finds the name it is looking for. Let's examine each separately.

- **files** - Refers to the */etc/hosts* file. Generally, we want the system to look here first as it is the fastest.
- **mdns4_minimal** - This is a legacy multi-cast DNS protocol.
- **dns** - This tells the system to go to the */etc/resolv.conf* for find a DNS server.
- **[NOTFOUND=return]** - This indicates that if the *mdns_minimal* search returns NOTFOUND, this should be treated as authoritative and the search ceases.
- **mdns4** - This is multicast DNS, a relatively rare DNS-like protocol for small networks without DNS servers.

Scheduling Jobs

Linux has a built-in functionality that allows us to schedule such jobs as these on a regular schedule. In Linux, this is called **cron**, or **crond**, for the **daemon** that runs these services (a daemon, or demon, is simply a process that runs in the background).

How Cron Works in Linux

Typically used to schedule such mundane, but necessary, tasks such as doing scheduled regular backups at a regular time each week, we can use it to schedule our scans or other nefarious "jobs".

The cron daemon starts when the system boots and continues to run as long as the system is up and running. It reads a configuration file or files that consist of the jobs to be run and the schedule they are to be run on. Almost anything we can do from the command line can be scheduled to be done on a regular schedule using cron.

Let's take a look how it works and how we can use it as a hacker.

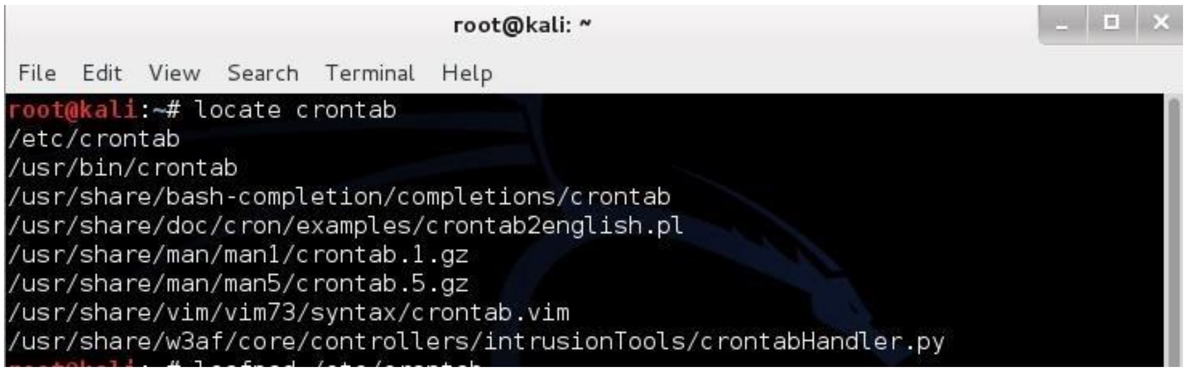
Step 1 Locating Crontab

Cron is one of those functions that is almost identical across Linux distributions, so what you learn here can be used in Ubuntu, Red Hat, Suse, Mint, Slackware, CentOS, etc. It has been part of the Linux/UNIX family since the 1970s, so it is tried and true and has proven its value.

Like so many things in Linux, the cron functionality is controlled by a configuration file that is a plain text file. In a multi-user environment, each user can have their own cron configuration file, but here we will concentrate on the root user in Kali.

For cron, the configuration file is the **crontab**, or "cron table", file. To find the crontab file, type:

locate crontab



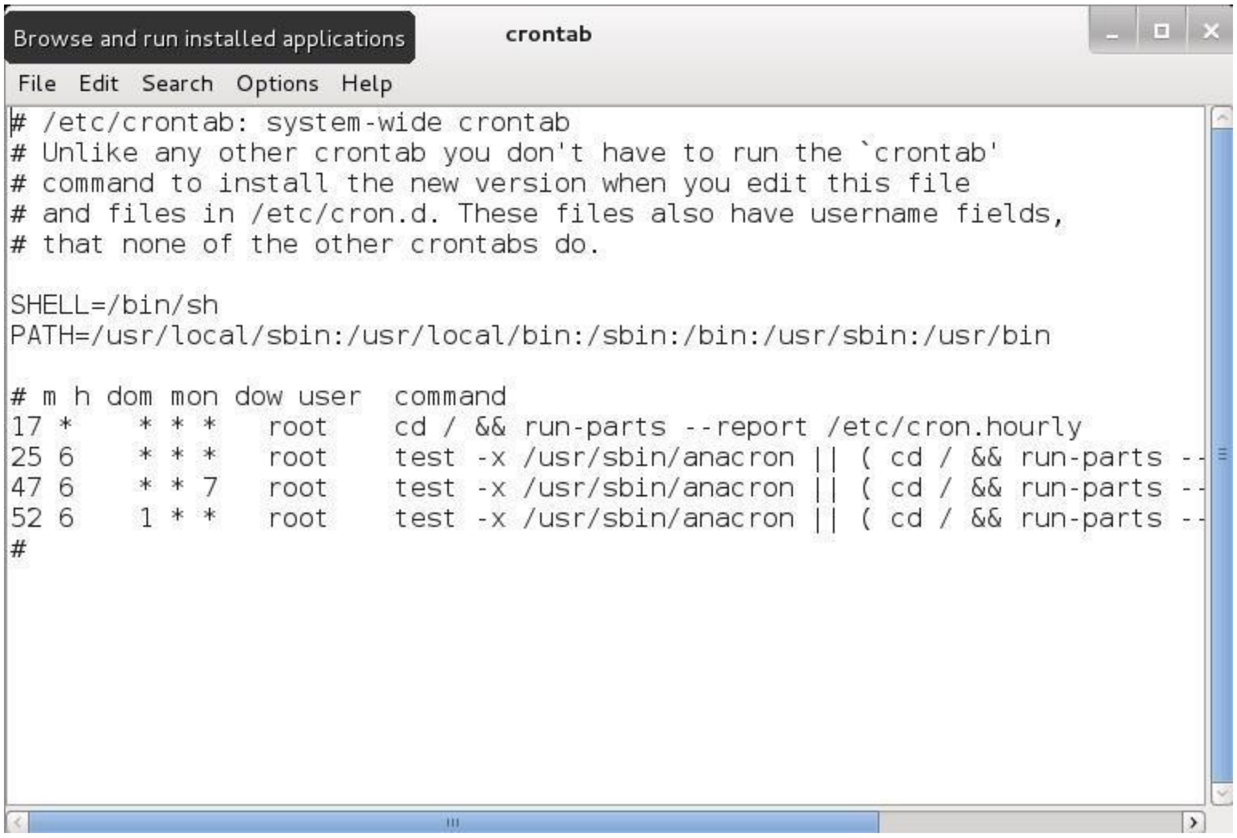
```
root@kali:~# locate crontab
/etc/crontab
/usr/bin/crontab
/usr/share/bash-completion/completions/crontab
/usr/share/doc/cron/examples/crontab2english.pl
/usr/share/man/man1/crontab.1.gz
/usr/share/man/man5/crontab.5.gz
/usr/share/vim/vim73/syntax/crontab.vim
/usr/share/w3af/core/controllers/intrusionTools/crontabHandler.py
```

As you can see, it is located in the */etc* directory like nearly every other configuration file in Linux (there are exceptions, of course).

Step 2 Opening Crontab

Let's open it and look around. We can open it with any text editor, but here let's use the graphical text editor built into Kali, Leafpad. Type:

leafpad /etc/crontab


Browse and run installed applications
crontab
File Edit Search Options Help

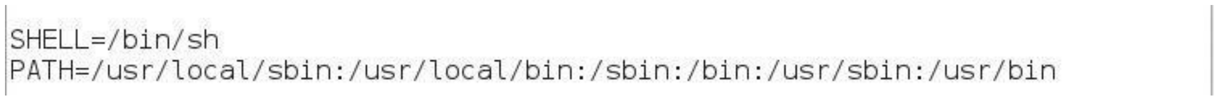
```
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --
#
```

The Debian version that Kali is built on has a newer version of crontab that is slightly easier to work with than earlier versions. Unlike earlier versions, they have labeled the fields and added a new field to denote the user that will run the job.

Step 3 The Anatomy of a Crontab

Let's break down the parts. As you can see in the screenshot above, the crontab starts with 5 lines that are commented (#) out. These lines are simply an explanation and notes, they are not seen or executed by the system.

After the commented lines, you see a couple of lines together.



```
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
```

The first of these sets the shell to run the jobs from. In our case, we have designating the BASH shell with the following command. If want to use a different shell, we could designate it here.

SHELL=/bin/sh

The second line sets the PATH variable. The PATH variable is an environment variable (there is one in Windows, too), that tells the system where to look for commands that are being used in the cron job. Typically, these are *bin* and *sbin* directories (binary) that contain the system commands that we use in Linux (**ls**, **echo**, **ps**, **cd**, etc.).

Here the default settings are:

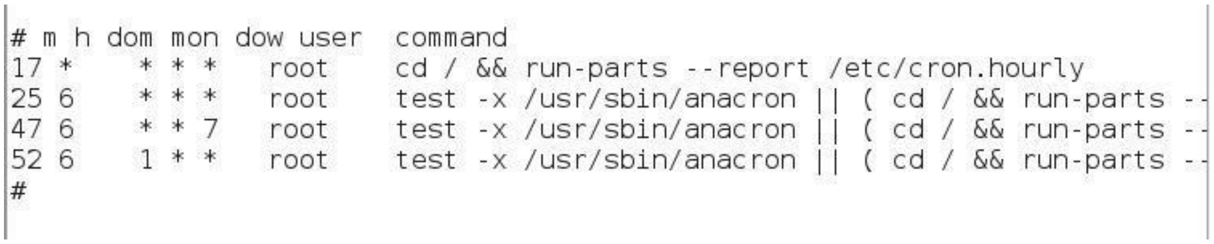
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

This simply means that the cron daemon will look in those directories for the commands that you use in your scheduled jobs. If you are using a command or file not in those directories, simply edit that line and add that directory to the end of the line after putting in a colon (:), such as:

PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/newdir

Step 4 Scheduling Jobs

Now comes the meat of the crontab file. In this third section, we schedule the jobs that we want to run.



```
# m h dom mon dow user   command
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --
#
```

As you can see, each line represents a scheduled job that will run automatically at whatever day, date, or time you have it scheduled for.

The fields of the crontab file are as follows:

- **Minute** -- the exact minute when the command or will be run (0-59)
- **Hour** -- the exact hour when the command or job will be run (0-23)
- **Day of the Month** -- the exact day of the month when the command or job will be run (1-31)
- **Month** -- the exact month when the command or job will be run (1-12)
- **Day of the week** -- the exact day when you want the command to run (0-6, Sunday=0)
- **User** -- the user permissions that the job will run as
- **Command** -- the command or job you want to run

The asterisk (*), or star, represents any, so it may be any day, any hour, or any minute.

Using Cron to Find Vulnerable Servers

Now, let's imagine that we want to scan the globe for IP addresses that are vulnerable to the Heartbleed vulnerability.

Reportedly, there are over 300,000 servers that are still unpatched. Although that's a very large number, with over 2 billion IP addresses on the planet; that represents 1 out of every 10,000 IP addresses that are vulnerable. This means we will need to set up a scanner to repeatedly search thousands of IP's to find just one vulnerable server.

This is a perfect task for a cron job!

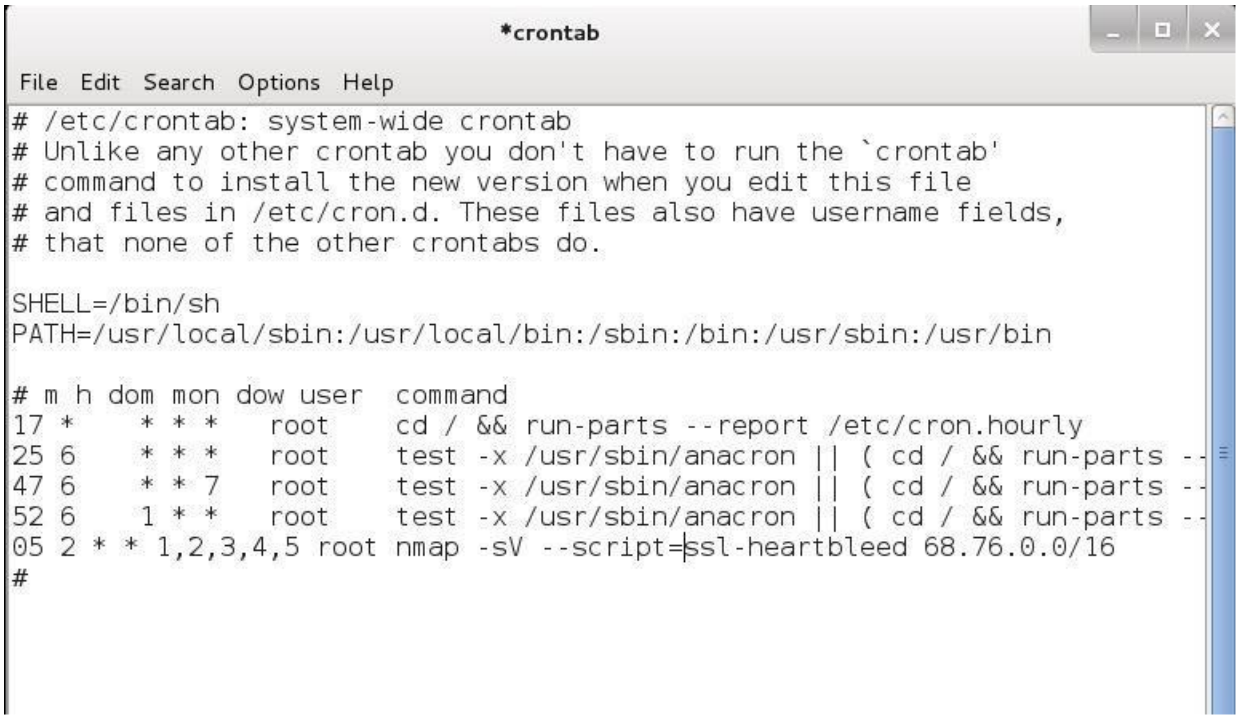
Step 5 Scheduling Our Heartbleed Scanner

We can schedule the scanner to run every night while we are asleep and hopefully, awake each morning with a new potential victim(s)!

Let's open that cron tab file again in any text editor.

Now we are going to add a line to run our nmap scanner each weeknight at 2:05 am. Simply add this line to our crontab file:

05 2 * * 1,2,3,4,5 root nmap -sV --script=ssl-heartbleed 68.76.0.0/16



```
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --
05 2 * * 1,2,3,4,5 root nmap -sV --script=ssl-heartbleed 68.76.0.0/16
#
```

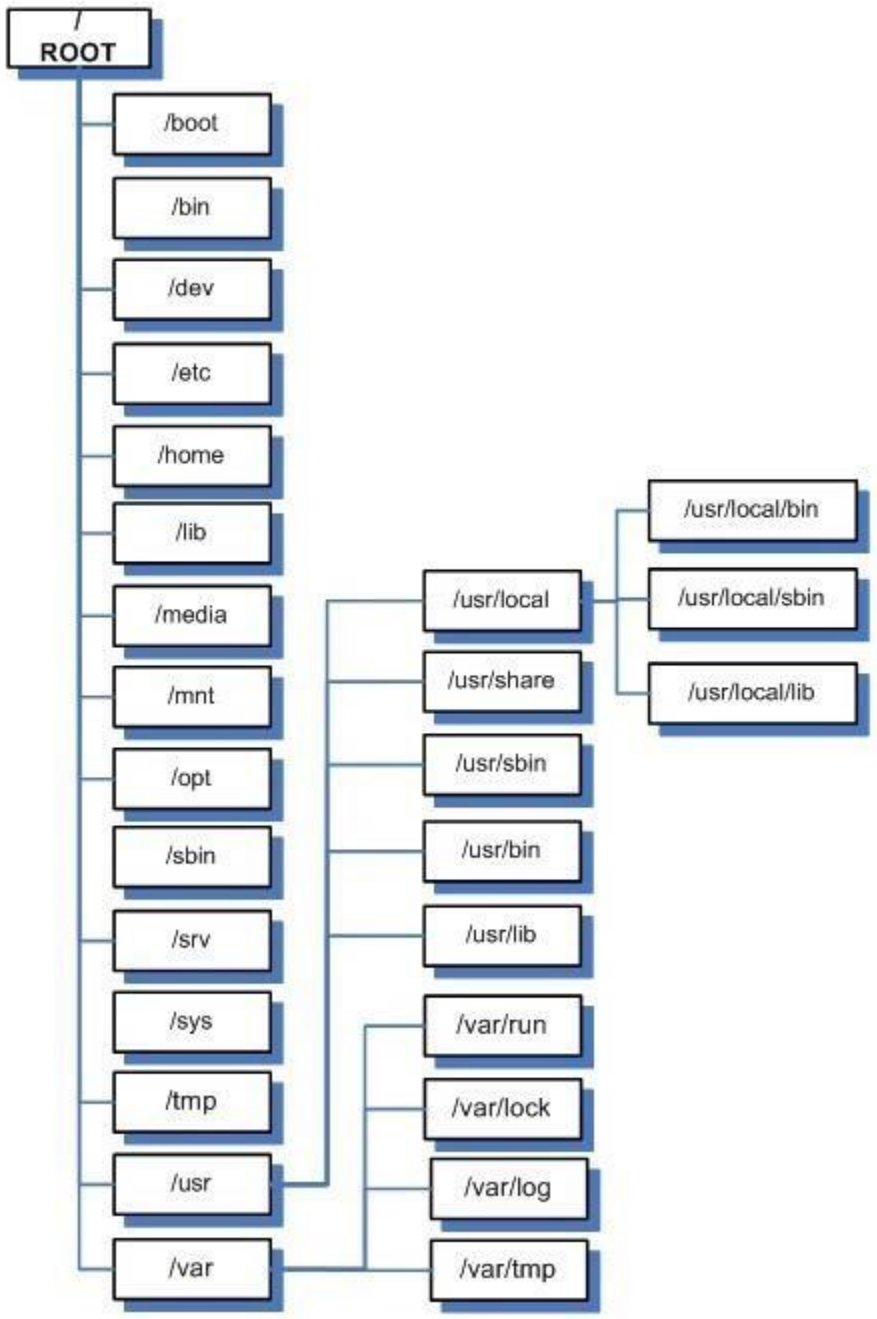
Now, save and close this file.

This would schedule our nmap Heartbleed scanner to run Monday, Tuesday, Wednesday, Thursday, and Friday at precisely 2:05 am for the Heartbleed vulnerability across 65,536 IP addresses.

That's good start.

Linking Files

As you have probably discovered by now, the file system in Linux is structured differently from Windows. There are no physical drives—just a logical file system tree with root at the top (yes, I know, roots should be at the bottom, but this is an upside-down tree).



In addition, filenames are often very long and complex with lots of dashes (-), dots (.), and numbers in their names. This can make typing them difficult for those of us with limited keyboard skills (remember, you can always use the tab key to autocomplete if you are in the right directory).

Sometimes we want to simplify the names of a file or we want to link a file in one directory with another in a separate directory. Linux has at least two ways to do this—symbolic (or soft) links and hard links. To understand how these two work and how they are different, we need to delve into some basic background on the Linux file system's internal structure.

Linux File Structure

We know that the Linux file system hierarchical structure is different than the Windows hierarchical structure, but from the inside, Linux's ext2 or ext3 file system is very different from Windows NTFS. Linux stores files at a structural level in three main sections:

- **The Superblock**
- **The Inode Table**
- **Data Blocks**

Let's take brief look at each of these.

Superblocks

The superblock is the section that contains the information about the file system, in general. This includes such things as the number of inodes and data blocks as well as how much data is in each file. It's kind of a overseer and housekeeper of the file system.

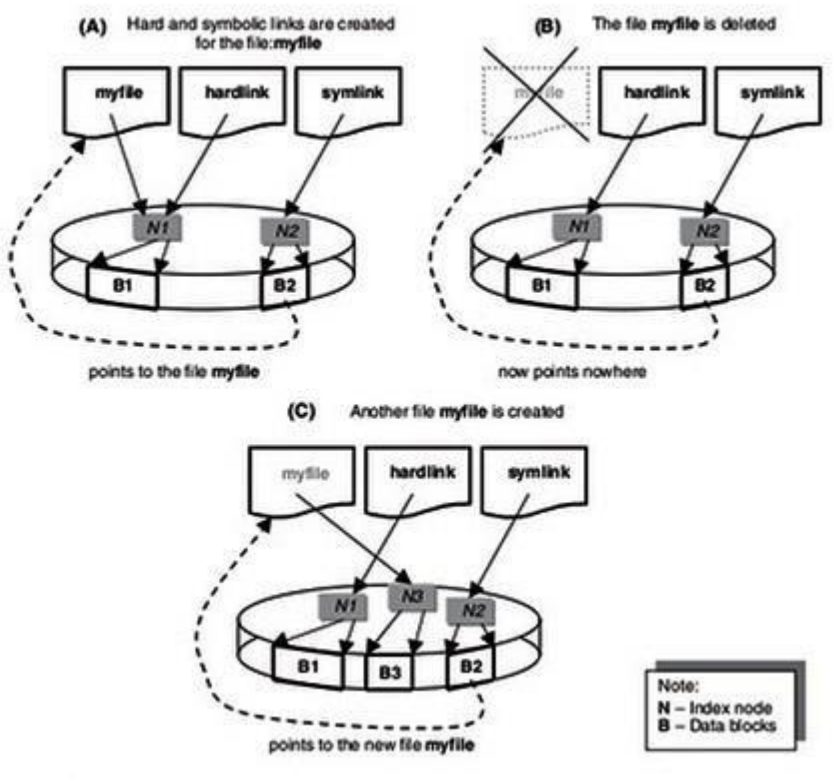
Inode Table

The inode table contains several inodes (information nodes) that describe a file or directory in the file system. Essentally, it is simply a record that describes a file (or directory) with all its critical information such as date created, location, date modified, permissions, and ownership. It does not, however, contain the data in the file.

It's important to understand from a forensic perspective that when a file is deleted, only the inode is removed.

Data Blocks

Data blocks are where the data that is in the file is stored, as well as the file name. Now with that understanding, let's introduce two ways of linking files, the hard link and the soft or symbolic link.


(A) Hard and symbolic links are created for the file:myfile
myfile
hardlink
symlink
N1
N2
B1
B2
points to the file myfile
(B) The file myfile is deleted
myfile
hardlink
symlink
N1
N2
B1
B2
now points nowhere
(C) Another file myfile is created
myfile
hardlink
symlink
N1
N3
N2
B1
B3
B2
points to the new file myfile
Note:
N – Index node
B – Data blocks

Hard Links

Hard linked files are identical. They have the same size and the same inode. When one hard linked file is modified or changed, it's linked file changes as well. You can hard link a file as many times as you need, but the link cannot cross file systems. They must be on the same file system as they share an inode.

Soft or Symbolic Links

Symbolic or soft links are different from hard links in that they do not share the same inode. A symbolic link is simply a pointer to the other file, similar to links in Windows, and they have different file sizes too. Unlike hard links, symbolic links do NOT need to be on the same file system.

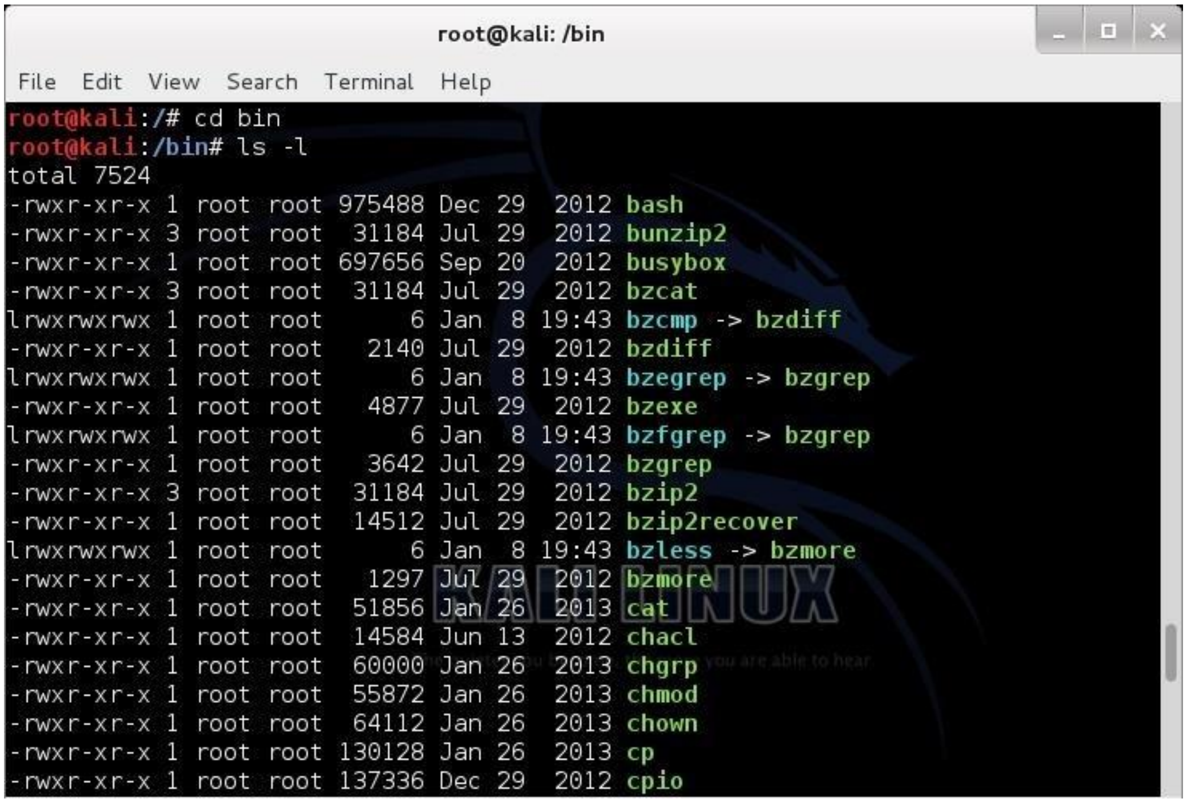
Step 1 Viewing Links

Let's take a look at what links look like in our filesystem on Kali. Let's navigate to the */bin* directory. Remember that the */bin* directory is just below the root of the file system and contains most the commands that we use on a daily basis in Linux.

kali> cd /bin

Now, let's look at the files in the bin directory.

kali > ls -l



```
root@kali:/# cd bin
root@kali:/bin# ls -l
total 7524
-rwxr-xr-x 1 root root 975488 Dec 29  2012 bash
-rwxr-xr-x 3 root root  31184 Jul 29  2012 bunzip2
-rwxr-xr-x 1 root root 697656 Sep 20  2012 busybox
-rwxr-xr-x 3 root root  31184 Jul 29  2012 bzcat
lrwxrwxrwx 1 root root      6 Jan  8 19:43 bzcmp -> bzdiff
-rwxr-xr-x 1 root root   2140 Jul 29  2012 bzdiff
lrwxrwxrwx 1 root root      6 Jan  8 19:43 bzegrep -> bzgrep
-rwxr-xr-x 1 root root   4877 Jul 29  2012 bzexe
lrwxrwxrwx 1 root root      6 Jan  8 19:43 bzfgrep -> bzgrep
-rwxr-xr-x 1 root root   3642 Jul 29  2012 bzgrep
-rwxr-xr-x 3 root root  31184 Jul 29  2012 bzip2
-rwxr-xr-x 1 root root  14512 Jul 29  2012 bzip2recover
lrwxrwxrwx 1 root root      6 Jan  8 19:43 bzless -> bzmore
-rwxr-xr-x 1 root root   1297 Jul 29  2012 bzmore
-rwxr-xr-x 1 root root  51856 Jan 26  2013 cat
-rwxr-xr-x 1 root root  14584 Jun 13  2012 chacl
-rwxr-xr-x 1 root root  60000 Jan 26  2013 chgrp
-rwxr-xr-x 1 root root  55872 Jan 26  2013 chmod
-rwxr-xr-x 1 root root  64112 Jan 26  2013 chown
-rwxr-xr-x 1 root root 130128 Jan 26  2013 cp
-rwxr-xr-x 1 root root 137336 Dec 29  2012 cpio
```

Notice that several files here show an arrow (->) pointing to another file. These are **symbolic links**. Also, note how small they are. Each is only 6 bytes. That's because they are only pointers, pointing to another file. The data block of the link simply contains the path to the file it is linked to.

When you edit the symbolically linked file, you are actually editing the target file as the symbolic file is only that path to the target file. Hope that makes sense.

Step 2 Creating Symbolic Links

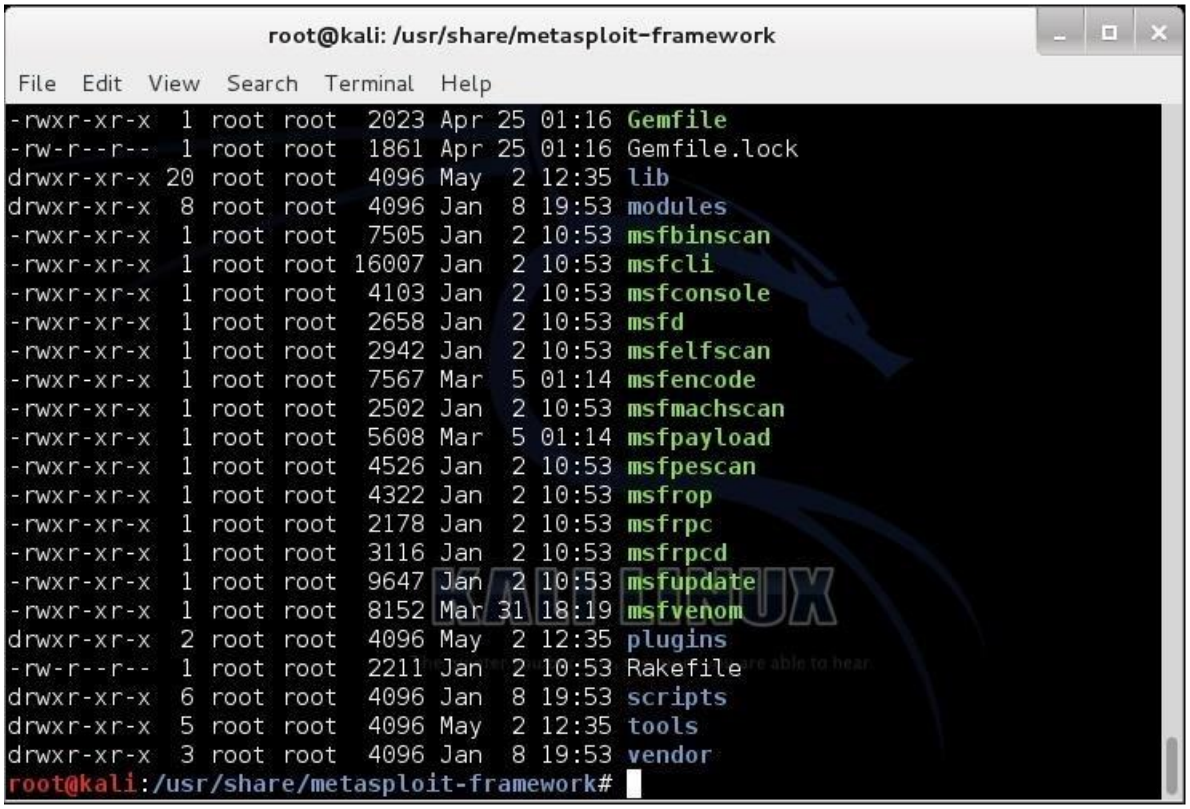
Now, let's create some links. Let's start with symbolic links as they are probably the most common on most people's systems. Although symbolic links can be created anywhere, we will be creating them in the *metasploit-framework* directory to make starting the **msfconsole** a touch easier.

Move to the */usr/share/metasploit-framework* directory, first.

kali > cd /usr/share/metasploit-console

Now, let's take a look at the this directory..

kali > ls -l



```
-rwxr-xr-x  1 root root  2023 Apr 25 01:16 Gemfile
-rw-r--r--  1 root root  1861 Apr 25 01:16 Gemfile.lock
drwxr-xr-x 20 root root  4096 May  2 12:35 lib
drwxr-xr-x  8 root root  4096 Jan  8 19:53 modules
-rwxr-xr-x  1 root root  7505 Jan  2 10:53 msfbinscan
-rwxr-xr-x  1 root root 16007 Jan  2 10:53 msfcli
-rwxr-xr-x  1 root root  4103 Jan  2 10:53 msfconsole
-rwxr-xr-x  1 root root  2658 Jan  2 10:53 msfd
-rwxr-xr-x  1 root root  2942 Jan  2 10:53 msfelfscan
-rwxr-xr-x  1 root root  7567 Mar  5 01:14 msfencode
-rwxr-xr-x  1 root root  2502 Jan  2 10:53 msfmachscan
-rwxr-xr-x  1 root root  5608 Mar  5 01:14 msfpayload
-rwxr-xr-x  1 root root  4526 Jan  2 10:53 msfpescan
-rwxr-xr-x  1 root root  4322 Jan  2 10:53 msfrop
-rwxr-xr-x  1 root root  2178 Jan  2 10:53 msfrpc
-rwxr-xr-x  1 root root  3116 Jan  2 10:53 msfrpcd
-rwxr-xr-x  1 root root  9647 Jan  2 10:53 msfupdate
-rwxr-xr-x  1 root root  8152 Mar 31 18:19 msfvenom
drwxr-xr-x  2 root root  4096 May  2 12:35 plugins
-rw-r--r--  1 root root  2211 Jan  2 10:53 Rakefile
drwxr-xr-x  6 root root  4096 Jan  8 19:53 scripts
drwxr-xr-x  5 root root  4096 May  2 12:35 tools
drwxr-xr-x  3 root root  4096 Jan  8 19:53 vendor
root@kali:/usr/share/metasploit-framework#
```

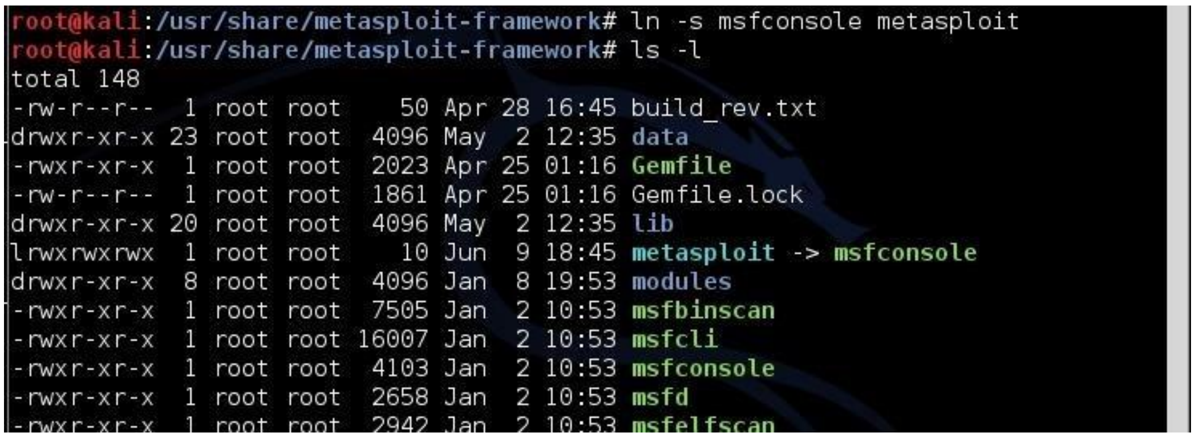
root@kali: /usr/share/metasploit-framework
File Edit View Search Terminal Help

To create a symbolic or soft link, we use the **ln** (link) command with the **-s** switch (symbolic) and the name of the file we want to link to (the target) and the name of the link we want to create. You can use either relative paths or absolute paths to link the two files.

Usually, when we want to enter the Metasploit console, we type msfconsole, remember? Now, let's say we want to change it so that we can simply type **metasploit** to enter the console rather having to remember msfconsole. We can link a new file, metasploit, to the old file, msfconsole, so that whenever we type **metasploit** it links or redirects to msfconsole.

Here is how we would do that.

kali >ln -s msfconsole metasploit



```
root@kali:/usr/share/metasploit-framework# ln -s msfconsole metasploit
root@kali:/usr/share/metasploit-framework# ls -l
total 148
-rw-r--r--  1 root root    50 Apr 28 16:45 build_rev.txt
drwxr-xr-x 23 root root  4096 May  2 12:35 data
-rwxr-xr-x  1 root root  2023 Apr 25 01:16 Gemfile
-rw-r--r--  1 root root  1861 Apr 25 01:16 Gemfile.lock
drwxr-xr-x 20 root root  4096 May  2 12:35 lib
lrwxrwxrwx  1 root root    10 Jun  9 18:45 metasploit -> msfconsole
drwxr-xr-x  8 root root  4096 Jan  8 19:53 modules
-rwxr-xr-x  1 root root  7505 Jan  2 10:53 msfbinscan
-rwxr-xr-x  1 root root 16007 Jan  2 10:53 msfcli
-rwxr-xr-x  1 root root  4103 Jan  2 10:53 msfconsole
-rwxr-xr-x  1 root root  2658 Jan  2 10:53 msfd
-rwxr-xr-x  1 root root  2942 Jan  2 10:53 msfelfscan
```

Note how small the symbolic link file, metasploit, is. It's just 12 bytes, because it is only a pointer. A path to the file it is linked to.

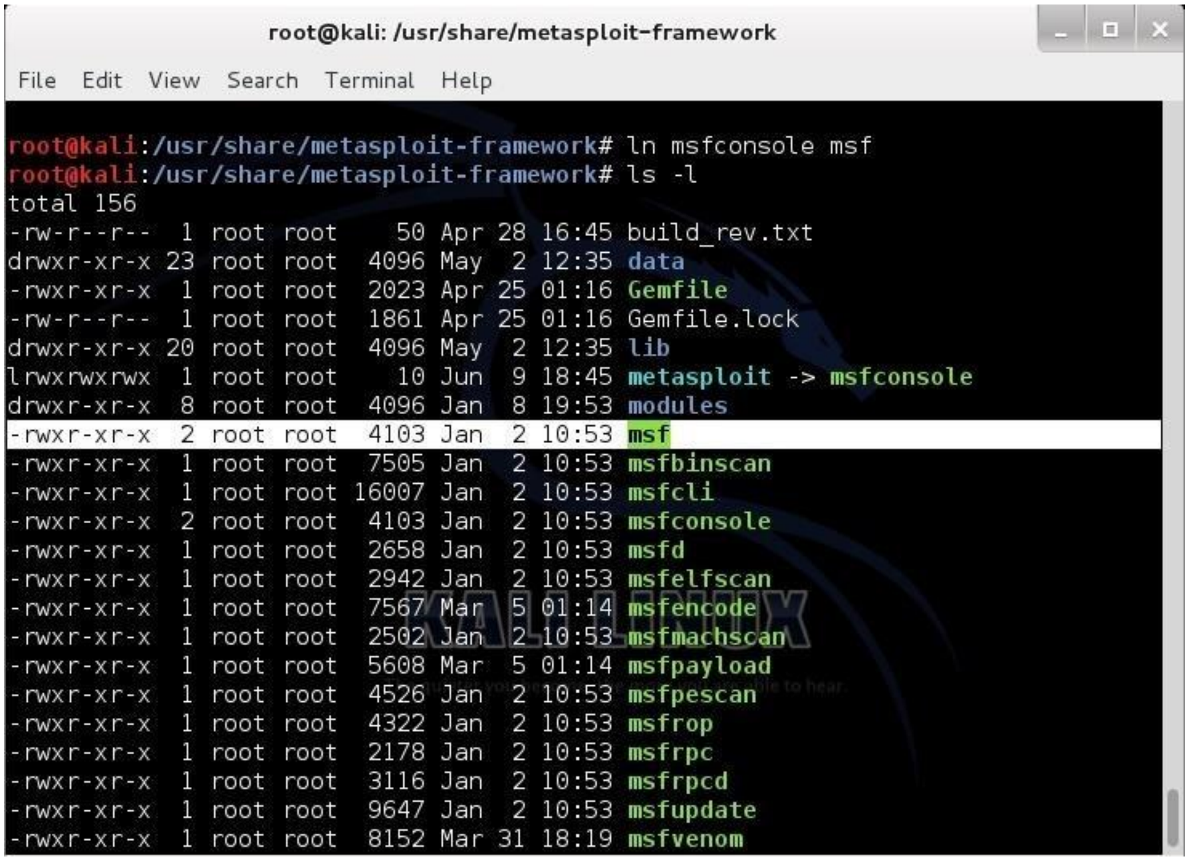
Now, to get into the msfconsole, I can type either metasploit or msfconsole and both will take me to the same place—the Metasploit Framework console.

Step 3 Creating Hard Links

To create a hard link, the syntax is very similar with the exception that we use the **ln** (link) command without the -s (symbolic) switch, then the existing file to hard link to, and finally, the target file that will be created to the existing file.

Back to our msfconsole example, let's add a hard link between msfconsole to a simpler command, **msf**. We can do this by typing:

kali > ln msfconsole msf



```
root@kali:/usr/share/metasploit-framework# ln msfconsole msf
root@kali:/usr/share/metasploit-framework# ls -l
total 156
-rw-r--r--  1 root root    50 Apr 28 16:45 build_rev.txt
drwxr-xr-x 23 root root  4096 May  2 12:35 data
-rwxr-xr-x  1 root root  2023 Apr 25 01:16 Gemfile
-rw-r--r--  1 root root  1861 Apr 25 01:16 Gemfile.lock
drwxr-xr-x 20 root root  4096 May  2 12:35 lib
lrwxrwxrwx  1 root root    10 Jun  9 18:45 metasploit -> msfconsole
drwxr-xr-x  8 root root  4096 Jan  8 19:53 modules
-rwxr-xr-x  2 root root  4103 Jan  2 10:53 msf
-rwxr-xr-x  1 root root  7505 Jan  2 10:53 msfbinscan
-rwxr-xr-x  1 root root 16007 Jan  2 10:53 msfcli
-rwxr-xr-x  2 root root  4103 Jan  2 10:53 msfconsole
-rwxr-xr-x  1 root root  2658 Jan  2 10:53 msfd
-rwxr-xr-x  1 root root  2942 Jan  2 10:53 msfelfscan
-rwxr-xr-x  1 root root  7567 Mar  5 01:14 msfencode
-rwxr-xr-x  1 root root  2502 Jan  2 10:53 msfmachscan
-rwxr-xr-x  1 root root  5608 Mar  5 01:14 msfpayload
-rwxr-xr-x  1 root root  4526 Jan  2 10:53 msfpescan
-rwxr-xr-x  1 root root  4322 Jan  2 10:53 msfrop
-rwxr-xr-x  1 root root  2178 Jan  2 10:53 msfrpc
-rwxr-xr-x  1 root root  3116 Jan  2 10:53 msfrpcd
-rwxr-xr-x  1 root root  9647 Jan  2 10:53 msfupdate
-rwxr-xr-x  1 root root  8152 Mar 31 18:19 msfvenom
```

As you can see above, now we have created a hard link called **msf**. Remember, hard links share an inode with the linked file, so they are exactly the same size. Notice that our new file, msf, is exactly the same size as msfconsole, 4103 bytes. Now, when we want to invoke (start) the msfconsole, we have the option to type **metasploit**, **msf**, and the original **msfconsole**. All will work equally well.

Devices Files

Fundamental to understanding how to use and administer hard drives and other devices in Linux is an understanding of how Linux specifies these devices in its file system.

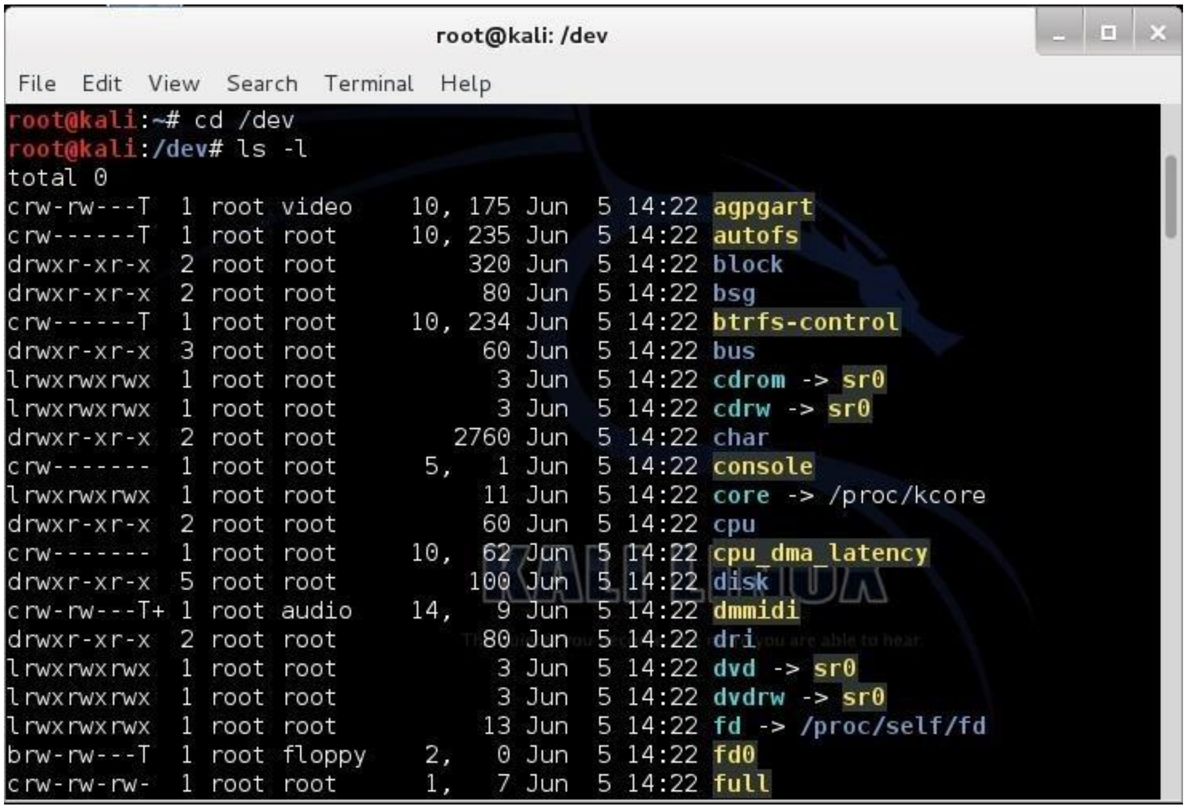
Very often, if we are using are hard drive in a hack or in forensics, we will need to specifically address its device file name. These device file names allow the device (e.g. hard drive) to interact with system software and kernel through system calls. These files are NOT device drivers, but rather rendezvous points that are used to communicate to the drivers. Linux maintains a device file for **every** device in the system in the */dev* directory.

In this tutorial, we will examine how Linux names and interacts with the file system through the */dev* directory and its files.

The /Dev Directory

The */dev* directory contains all the files that represent physical peripheral devices present on the system such as disk drives, terminals, and printers. The */dev* directory is directly below the */* directory. If we navigate there, we will see an entry for all of our devices.

```
kali > cd /dev
kali > ls -l
```



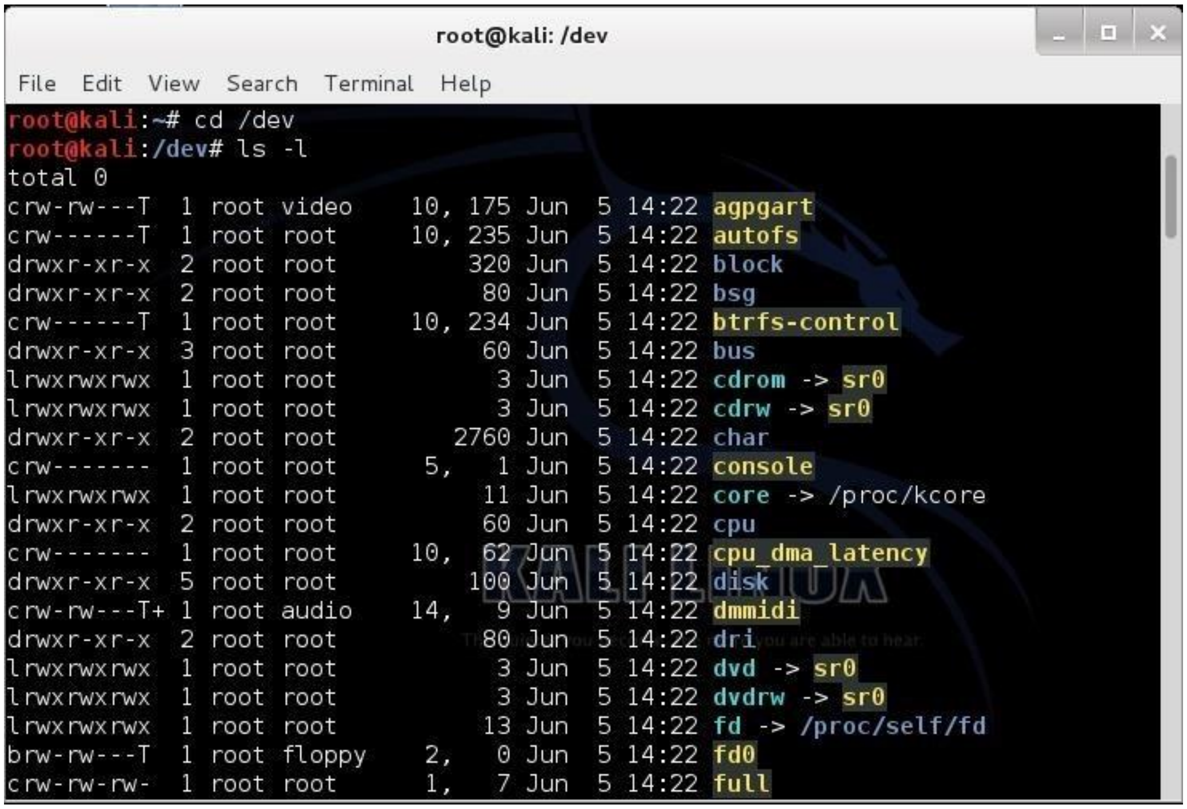
```
root@kali:~# cd /dev
root@kali:/dev# ls -l
total 0
crw-rw---T  1 root video    10, 175 Jun  5 14:22 agpgart
crw------T  1 root root     10, 235 Jun  5 14:22 autofs
drwxr-xr-x  2 root root         320 Jun  5 14:22 block
drwxr-xr-x  2 root root          80 Jun  5 14:22 bsg
crw------T  1 root root     10, 234 Jun  5 14:22 btrfs-control
drwxr-xr-x  3 root root          60 Jun  5 14:22 bus
lrwxrwxrwx  1 root root           3 Jun  5 14:22 cdrom -> sr0
lrwxrwxrwx  1 root root           3 Jun  5 14:22 cdrw -> sr0
drwxr-xr-x  2 root root        2760 Jun  5 14:22 char
crw-------  1 root root      5,   1 Jun  5 14:22 console
lrwxrwxrwx  1 root root          11 Jun  5 14:22 core -> /proc/kcore
drwxr-xr-x  2 root root          60 Jun  5 14:22 cpu
crw-------  1 root root     10,  62 Jun  5 14:22 cpu_dma_latency
drwxr-xr-x  5 root root         100 Jun  5 14:22 disk
crw-rw---T+ 1 root audio    14,   9 Jun  5 14:22 dmmidi
drwxr-xr-x  2 root root          80 Jun  5 14:22 dri
lrwxrwxrwx  1 root root           3 Jun  5 14:22 dvd -> sr0
lrwxrwxrwx  1 root root           3 Jun  5 14:22 dvdrw -> sr0
lrwxrwxrwx  1 root root          13 Jun  5 14:22 fd -> /proc/self/fd
brw-rw---T  1 root floppy    2,   0 Jun  5 14:22 fd0
crw-rw-rw-  1 root root      1,   7 Jun  5 14:22 full
```

Block v. Character Devices

Linux makes a distinction between block and character devices. Character devices are those that stream data into and out of the machine unbuffered and directly. These would include your keyboard, mouse, tape, and monitor. Because the data is unbuffered, it tends to be a slow process.

On the other hand, block devices are those that stream data into and out of the machine in buffered blocks. These include such devices as hard drives, CDs, DVDs, floppies, flash drives, etc. This data transfer tends to be much faster.

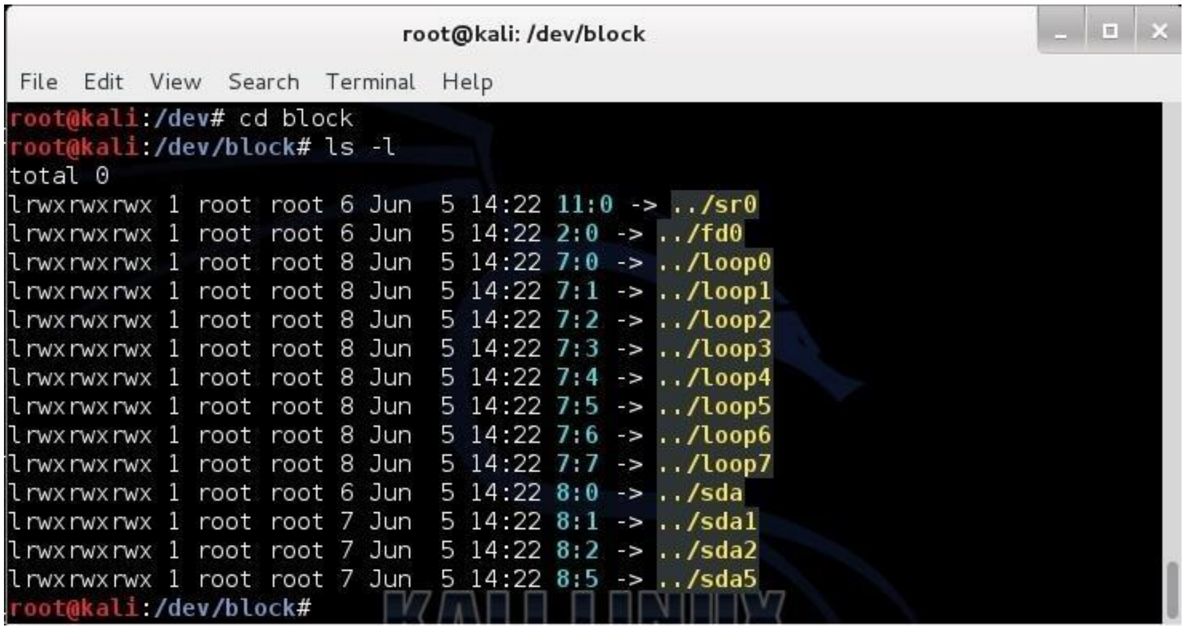
Notice in the long listing of the */dev* directory that some files begin with a "c" and some with a "b". Character devices begin a with "c " and block devices begin with a "b".



```
root@kali:~# cd /dev
root@kali:/dev# ls -l
total 0
crw-rw---T  1 root video    10, 175 Jun  5 14:22 agpgart
crw------T  1 root root     10, 235 Jun  5 14:22 autofs
drwxr-xr-x  2 root root         320 Jun  5 14:22 block
drwxr-xr-x  2 root root          80 Jun  5 14:22 bsg
crw------T  1 root root     10, 234 Jun  5 14:22 btrfs-control
drwxr-xr-x  3 root root          60 Jun  5 14:22 bus
lrwxrwxrwx  1 root root           3 Jun  5 14:22 cdrom -> sr0
lrwxrwxrwx  1 root root           3 Jun  5 14:22 cdrw -> sr0
drwxr-xr-x  2 root root        2760 Jun  5 14:22 char
crw-------  1 root root      5,   1 Jun  5 14:22 console
lrwxrwxrwx  1 root root          11 Jun  5 14:22 core -> /proc/kcore
drwxr-xr-x  2 root root          60 Jun  5 14:22 cpu
crw-------  1 root root     10,  62 Jun  5 14:22 cpu_dma_latency
drwxr-xr-x  5 root root         100 Jun  5 14:22 disk
crw-rw---T+ 1 root audio    14,   9 Jun  5 14:22 dmmidi
drwxr-xr-x  2 root root          80 Jun  5 14:22 dri
lrwxrwxrwx  1 root root           3 Jun  5 14:22 dvd -> sr0
lrwxrwxrwx  1 root root           3 Jun  5 14:22 dvdrw -> sr0
lrwxrwxrwx  1 root root          13 Jun  5 14:22 fd -> /proc/self/fd
brw-rw---T  1 root floppy    2,   0 Jun  5 14:22 fd0
crw-rw-rw-  1 root root      1,   7 Jun  5 14:22 full
```

You will notice in the third line of the */dev* directory listing that there is directory named "block". Let's navigate there and do a long list.

kali > cd /block
kali > ls -l



```
root@kali:/dev# cd block
root@kali:/dev/block# ls -l
total 0
lrwxrwxrwx 1 root root 6 Jun  5 14:22 11:0 -> ../sr0
lrwxrwxrwx 1 root root 6 Jun  5 14:22 2:0 -> ../fd0
lrwxrwxrwx 1 root root 8 Jun  5 14:22 7:0 -> ../loop0
lrwxrwxrwx 1 root root 8 Jun  5 14:22 7:1 -> ../loop1
lrwxrwxrwx 1 root root 8 Jun  5 14:22 7:2 -> ../loop2
lrwxrwxrwx 1 root root 8 Jun  5 14:22 7:3 -> ../loop3
lrwxrwxrwx 1 root root 8 Jun  5 14:22 7:4 -> ../loop4
lrwxrwxrwx 1 root root 8 Jun  5 14:22 7:5 -> ../loop5
lrwxrwxrwx 1 root root 8 Jun  5 14:22 7:6 -> ../loop6
lrwxrwxrwx 1 root root 8 Jun  5 14:22 7:7 -> ../loop7
lrwxrwxrwx 1 root root 6 Jun  5 14:22 8:0 -> ../sda
lrwxrwxrwx 1 root root 7 Jun  5 14:22 8:1 -> ../sda1
lrwxrwxrwx 1 root root 7 Jun  5 14:22 8:2 -> ../sda2
lrwxrwxrwx 1 root root 7 Jun  5 14:22 8:5 -> ../sda5
root@kali:/dev/block#
```

Here we see a listing of all the block devices. In the first line we see **sr0**; that would be the first CD-ROM (Linux tends to begin counting at 0, not 1). Near the bottom, we see **sda, sda1, sda2, sda5** (yours may be different), where sda1 represents the first primary partition on the SATA drive, and sda2 represents the second primary partition on the SAME drive.

Naming Conventions of Devices in Linux

Originally, hard drives were two types, IDE or SCSI. IDE (or later, E-IDE) was designed as a low cost alternative for low cost PCs. They were relatively slow and only allowed four devices per machine. In addition, they had to be configured in a master and slave configuration. Each master and slave combination had one cable and controller.

A faster, but more expensive alternative was the SCSI (Small Computer System Interface) drive. SCSI drives were (are) faster and pricier. Besides their speed advantage, they did not need a master/slave configuration, but rather were configured with a controller and a series of devices up to 15.

Linux would designate IDE hard drives with an **hd** and SCSI hard drives with an **sd**. In recent years, with the development and proliferation of SATA drives, we see that Linux designates these drives with **sd**, just like SCSI drives.

The first IDE drive was designated with an **hda**, the second **hdb**, the third **hdc**, and so on. The same happens with SCSI and now SATA drives; the first is designated with **sda**, the second **sdb**, the third **sdc**, and so on.

Some other devices files include:

- **/dev/usb** - USB devices
- **/dev/lp** - parallel port printer
- **/dev/tty** - local terminal
- **/dev/fd** - floppy drive (does anyone still use floppies?)

Logical vs. Physical Partitions of Hard Drives

Linux is able to recognize four (4) primary hard drive partitions per operating system. This doesn't limit us to four hard drives or four partitions as we can also use logical or extended partitions. We can have up to 15 logical or extended partitions per disk and each of these partitions acts as its own hard drive and operates just as fast as a primary partition.

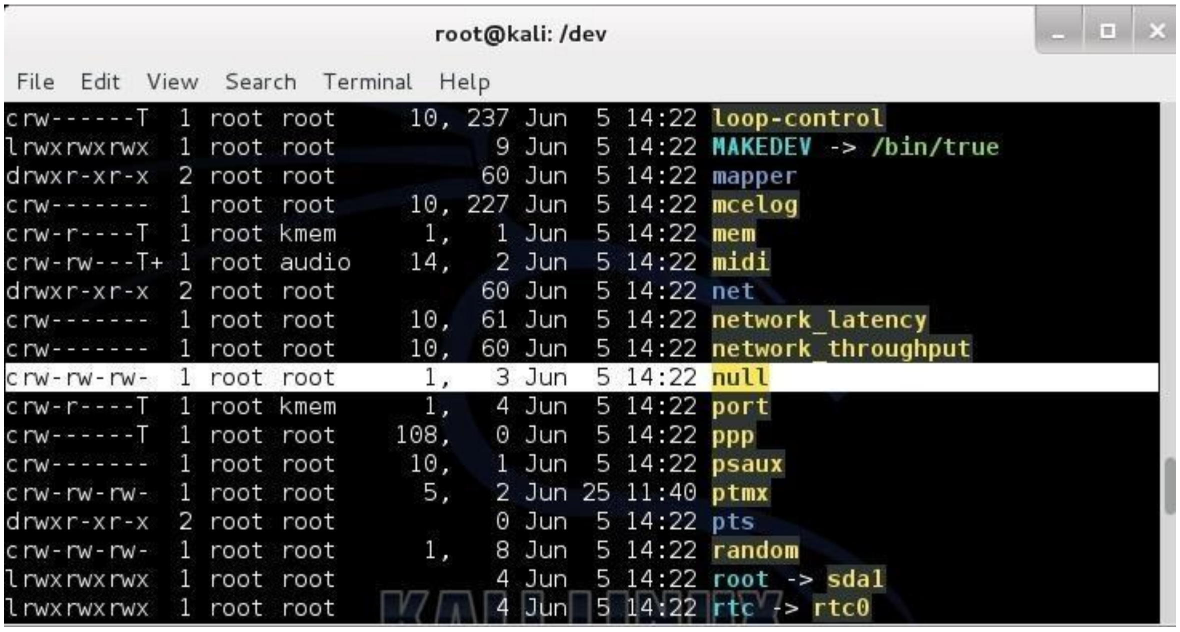
The first primary partition in Linux with a SATA drive would be **sda1**, the second **sda2**, the third **sda3**, and the fourth **sda4**. Beyond these primary partitions, we can still partition the drive, but they are now logical partitions. The first logical partition would be **sda5** with a SATA drive. This can then be followed by 14 more logical drives, if needed, with the last logical drive on the first SATA drive being **sda19** (4 primary and 15 logical partitions).

In Linux, we generally have a separate partition for swap space. Swap space is that area of the hard drive that is used as virtual memory. This means that when we run out of memory (RAM) for a particular process or application, the operating system will then use this hard drive space as if it were RAM, but obviously, much slower (about 1,000x slower).

Special Devices

Linux has a number of special device files. This is a list of a few of the most important special device files.

/dev/null



```
crw------T  1 root root     10, 237 Jun  5 14:22 loop-control
lrwxrwxrwx  1 root root           9 Jun  5 14:22 MAKEDEV -> /bin/true
drwxr-xr-x  2 root root          60 Jun  5 14:22 mapper
crw-------  1 root root     10, 227 Jun  5 14:22 mcelog
crw-r----T  1 root kmem      1,   1 Jun  5 14:22 mem
crw-rw---T+ 1 root audio    14,   2 Jun  5 14:22 midi
drwxr-xr-x  2 root root          60 Jun  5 14:22 net
crw-------  1 root root     10,  61 Jun  5 14:22 network_latency
crw-------  1 root root     10,  60 Jun  5 14:22 network_throughput
crw-rw-rw-  1 root root      1,   3 Jun  5 14:22 null
crw-r----T  1 root kmem      1,   4 Jun  5 14:22 port
crw------T  1 root root    108,   0 Jun  5 14:22 ppp
crw-------  1 root root     10,   1 Jun  5 14:22 psaux
crw-rw-rw-  1 root root      5,   2 Jun 25 11:40 ptmx
drwxr-xr-x  2 root root           0 Jun  5 14:22 pts
crw-rw-rw-  1 root root      1,   8 Jun  5 14:22 random
lrwxrwxrwx  1 root root           4 Jun  5 14:22 root -> sda1
lrwxrwxrwx  1 root root           4 Jun  5 14:22 rtc -> rtc0
```

This device is a data sink or "bit bucket". It makes data disappear. If you redirect output to this device it will disappear. If you read from */dev/null*, you will get a null string. If I wanted to wipe a drive clean, deleting all the data, I could use:

dd if=/dev/null of=/dev/sda
/dev/zero

This device can be used as an input file to provide as many null bytes (0x00) as necessary. It is often used to initialize a file or hard drive.

/dev/ full

This device is a special file that always returns the "device full" error. Usually, it is used to test how a program reacts to a "disk full" error. It is also able to provide an infinite number of null byte characters to any process for testing.

/dev/random

This device can be used as an input to fill a file or partition with random, or more precisely, pseudo-random data. I might use this to overwrite a file or partition to make it much harder to recover deleted files by a forensic investigator.

It's almost impossible to remove evidence of a file from recovery by a skilled forensic investigator with unlimited time and money. Since few forensic investigators have the skill or the time or the money, this technique will inhibit most forensic investigations.

To do this, we can use the **dd** command. For instance:

```
dd if=/dev/random of=evidencefile bs=1 count=1024
```

GRUB Bootloader

Many of you have installed Kali_Linux as a virtual machine (VM) using VMware or VirtualBox, while others have installed Kali (or BackTrack) in a dual-boot system. The drawback to installing these hacking systems as a VM is that it then requires an external wireless adapter (your wireless adapter is piped through the VM as a wired device, eth0), but it makes for a great place to test your hacks while honing your skills.

Using Kali in a dual-boot system doesn't require another wireless adapter and enables you to use the full resources of your physical system without having to go through a hypervisor. In this way, when you need your Windows, you boot into Windows, and when you need Kali, you boot into Kali.

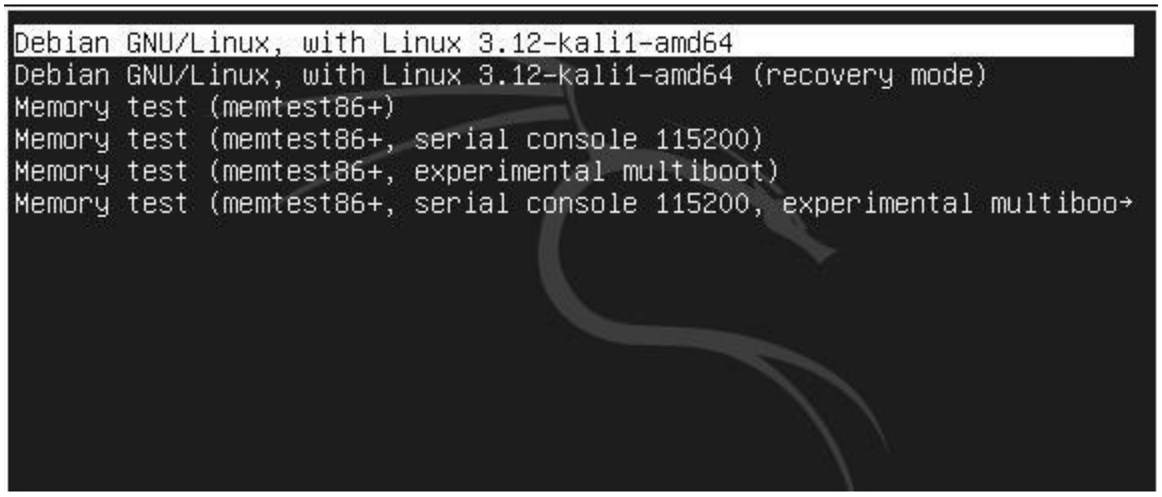
The Bootloader

To be able to run a dual-boot system, you must have a bootloader, and therein lies the issue. The bootloader enables us to choose which operating system we want to boot into. To be comfortable with this arrangement, we need to understand a bit about this thing called a bootloader.

Historically, Linux has used two bootloaders, LILO and GRUB. You may find LILO on some older, legacy systems, but it has largely been replaced by GRUB. GRUB has two versions, the original GRUB and now the new improved GRUB2! Our Kali system comes with GRUB2 by default, so I will focus my attention on this newer version here.

A Little Background on GRUB

GRUB is an acronym for GRand Unified Bootloader. It largely replaces the legacy bootloader found on many older Linux version, LILO. GRUB works by intercepting control of the boot process when the BIOS transfers control to the Master Boot Record (MBR) in the first sector of the primary hard drive. Rather than the MBR then finding the first active partition, GRUB replaces the MBR with its own code that controls which partition is booted.



```
Debian GNU/Linux, with Linux 3.12-kali1-amd64
Debian GNU/Linux, with Linux 3.12-kali1-amd64 (recovery mode)
Memory test (memtest86+)
Memory test (memtest86+, serial console 115200)
Memory test (memtest86+, experimental multiboot)
Memory test (memtest86+, serial console 115200, experimental multiboo→
```

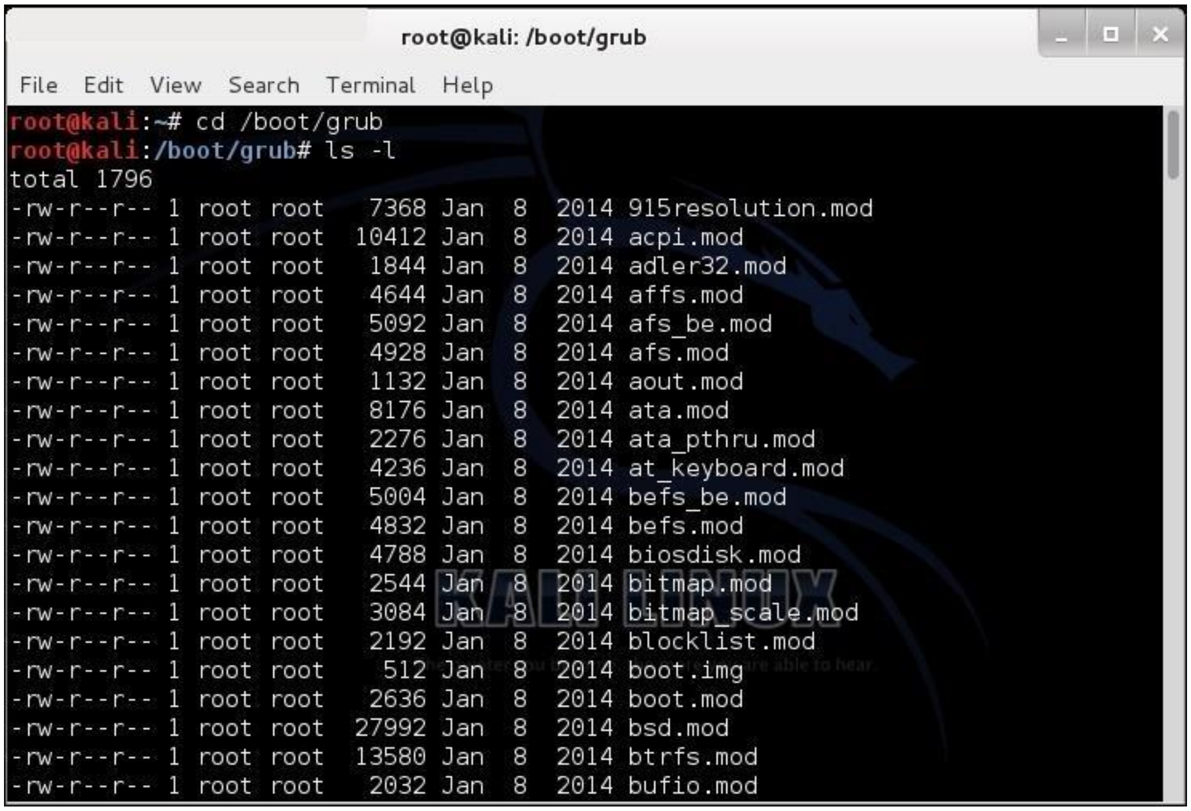
Step 1 Exploring GRUB

Let's take a look at GRUB2 as it is installed on Kali. GRUB2, unlike the original GRUB, has all its files installed into three main locations. These are:

- **/boot/grub/grub.cfg** - this is the main configuration file (replaces menu.lst)
- **/etc/grub.d** - this directory contains the scripts that build the grub.cfg
- **/etc/default/grub** - this file contains the GRUB menu settings

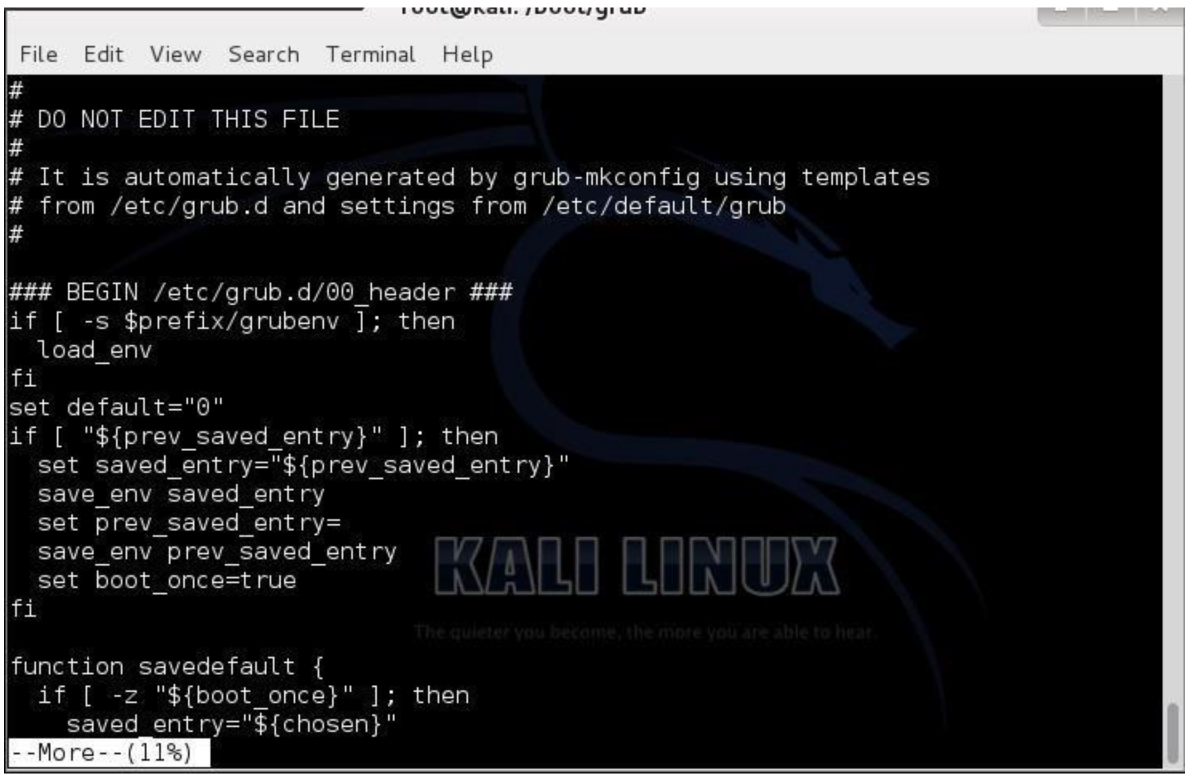
Now, let's begin by navigating and looking inside the GRUB directory.

kali > cd /boot/grub
kali >ls -l



```
root@kali:~# cd /boot/grub
root@kali:/boot/grub# ls -l
total 1796
-rw-r--r-- 1 root root   7368 Jan  8  2014 915resolution.mod
-rw-r--r-- 1 root root  10412 Jan  8  2014 acpi.mod
-rw-r--r-- 1 root root   1844 Jan  8  2014 adler32.mod
-rw-r--r-- 1 root root   4644 Jan  8  2014 affs.mod
-rw-r--r-- 1 root root   5092 Jan  8  2014 afs_be.mod
-rw-r--r-- 1 root root   4928 Jan  8  2014 afs.mod
-rw-r--r-- 1 root root   1132 Jan  8  2014 aout.mod
-rw-r--r-- 1 root root   8176 Jan  8  2014 ata.mod
-rw-r--r-- 1 root root   2276 Jan  8  2014 ata_pthru.mod
-rw-r--r-- 1 root root   4236 Jan  8  2014 at_keyboard.mod
-rw-r--r-- 1 root root   5004 Jan  8  2014 befs_be.mod
-rw-r--r-- 1 root root   4832 Jan  8  2014 befs.mod
-rw-r--r-- 1 root root   4788 Jan  8  2014 biosdisk.mod
-rw-r--r-- 1 root root   2544 Jan  8  2014 bitmap.mod
-rw-r--r-- 1 root root   3084 Jan  8  2014 bitmap_scale.mod
-rw-r--r-- 1 root root   2192 Jan  8  2014 blocklist.mod
-rw-r--r-- 1 root root    512 Jan  8  2014 boot.img
-rw-r--r-- 1 root root   2636 Jan  8  2014 boot.mod
-rw-r--r-- 1 root root  27992 Jan  8  2014 bsd.mod
-rw-r--r-- 1 root root  13580 Jan  8  2014 btrfs.mod
-rw-r--r-- 1 root root   2032 Jan  8  2014 bufio.mod
```

As you can see, there are many files in this directory, but the one we want to focus on here is the **grub.cfg**. This is the configuration file in the new GRUB2 that comes with Kali. It replaces the old **menu.lst** that you will find on the original GRUB. Let's open it with the **more** command and examine it.



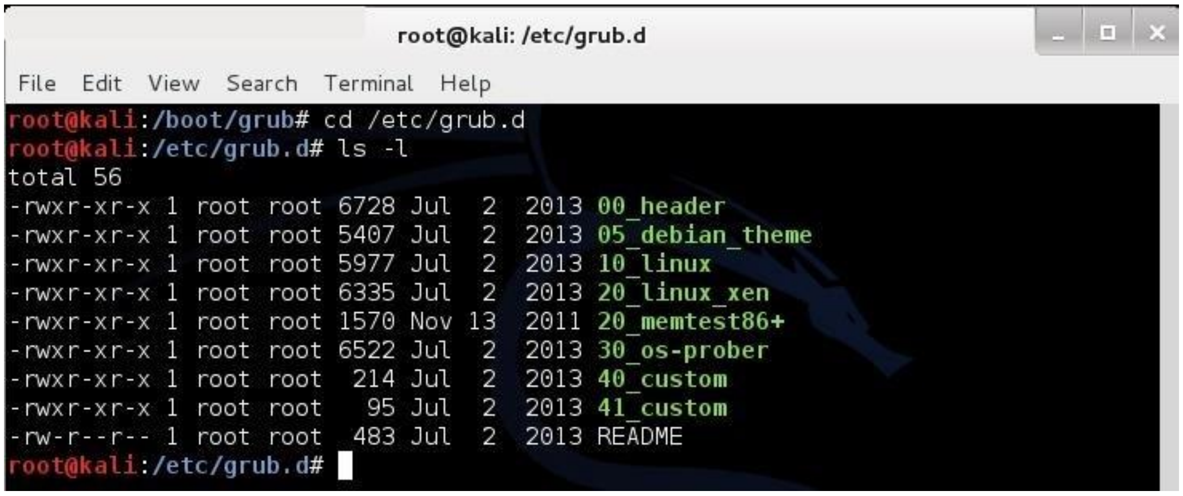
```
#
# DO NOT EDIT THIS FILE
#
# It is automatically generated by grub-mkconfig using templates
# from /etc/grub.d and settings from /etc/default/grub
#

### BEGIN /etc/grub.d/00_header ###
if [ -s $prefix/grubenv ]; then
  load_env
fi
set default="0"
if [ "${prev_saved_entry}" ]; then
  set saved_entry="${prev_saved_entry}"
  save_env saved_entry
  set prev_saved_entry=
  save_env prev_saved_entry
  set boot_once=true
fi

function savedefault {
  if [ -z "${boot_once}" ]; then
    saved_entry="${chosen}"
--More--(11%)
```

Grub.cfg is basically a script for running and configuring GRUB2. It is generated by the scripts in **/etc/grub.d** and you generally should NOT try editing it directly (note the warning on the second line of the file).

Step 2 The /Etc/grub.d Directory

Next, let's look at the **/etc/grub.d** directory.

kali > cd /etc/grub.d
kali > ls -l



```
root@kali:/boot/grub# cd /etc/grub.d
root@kali:/etc/grub.d# ls -l
total 56
-rwxr-xr-x 1 root root 6728 Jul  2  2013 00_header
-rwxr-xr-x 1 root root 5407 Jul  2  2013 05_debian_theme
-rwxr-xr-x 1 root root 5977 Jul  2  2013 10_linux
-rwxr-xr-x 1 root root 6335 Jul  2  2013 20_linux_xen
-rwxr-xr-x 1 root root 1570 Nov 13  2011 20_memtest86+
-rwxr-xr-x 1 root root 6522 Jul  2  2013 30_os-prober
-rwxr-xr-x 1 root root  214 Jul  2  2013 40_custom
-rwxr-xr-x 1 root root   95 Jul  2  2013 41_custom
-rw-r--r-- 1 root root  483 Jul  2  2013 README
root@kali:/etc/grub.d# 
```

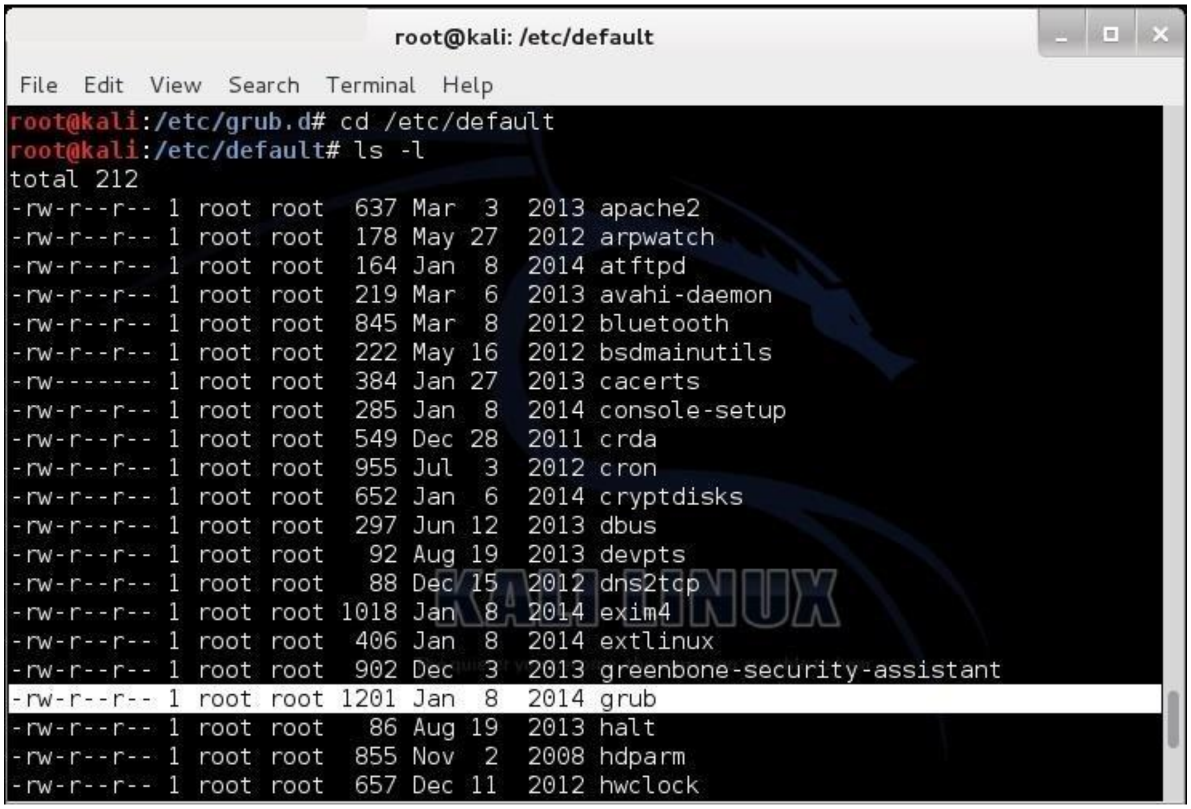
As you can see in the screenshot above, this directory has a number of scripts that are run to create the **grub.cfg file**. Let's look at the key entries here.

- **00_header** - this script loads the settings from /etc/default/grub
- **05_debian_theme** - this script defines the colors, background, etc.
- **10_linux** - this script loads the menu entries
- **20_memtest86** - this script loads the memory tester
- **30_os-prober** - this script scans the hard drives for other operating systems
- **40_custom** - this is a template for manually adding other menu entries

Step 3 Exploring /Etc/Default/Grub

Now, let's go to the **/etc/default** directory and look to see what is in this directory.

kali > cd /etc/default
kali > ls -l

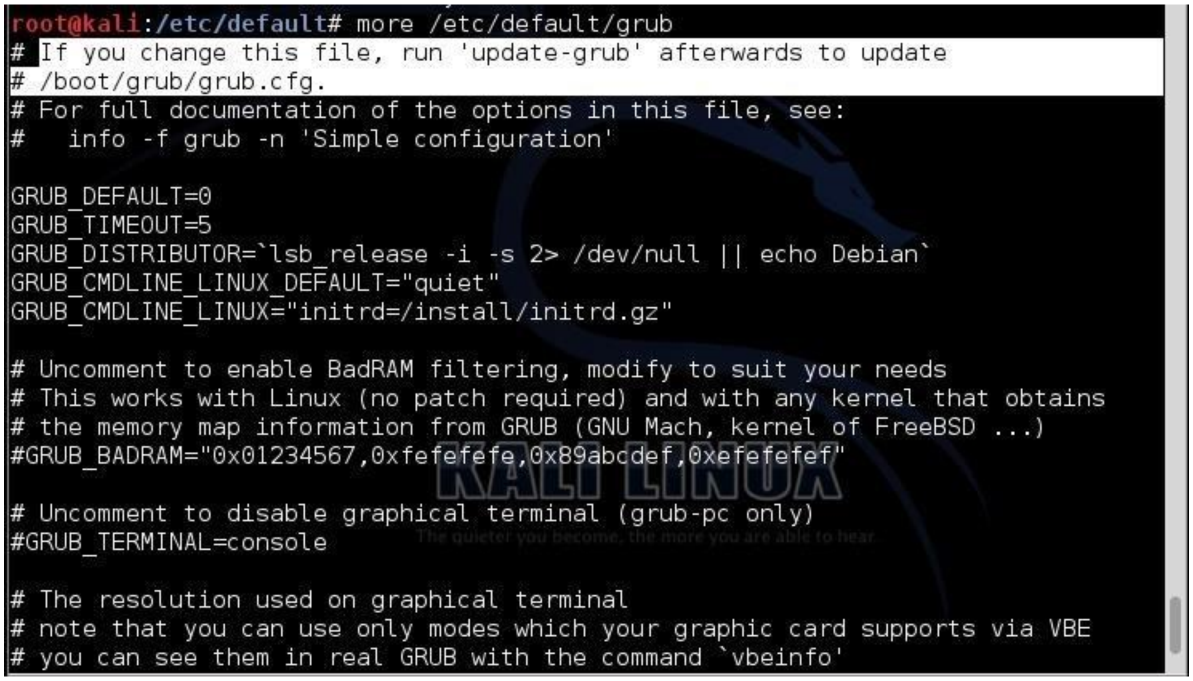


```
root@kali:/etc/grub.d# cd /etc/default
root@kali:/etc/default# ls -l
total 212
-rw-r--r-- 1 root root  637 Mar  3  2013 apache2
-rw-r--r-- 1 root root  178 May 27  2012 arpwatch
-rw-r--r-- 1 root root  164 Jan  8  2014 atftpd
-rw-r--r-- 1 root root  219 Mar  6  2013 avahi-daemon
-rw-r--r-- 1 root root  845 Mar  8  2012 bluetooth
-rw-r--r-- 1 root root  222 May 16  2012 bsdmainutils
-rw------- 1 root root  384 Jan 27  2013 cacerts
-rw-r--r-- 1 root root  285 Jan  8  2014 console-setup
-rw-r--r-- 1 root root  549 Dec 28  2011 crda
-rw-r--r-- 1 root root  955 Jul  3  2012 cron
-rw-r--r-- 1 root root  652 Jan  6  2014 cryptdisks
-rw-r--r-- 1 root root  297 Jun 12  2013 dbus
-rw-r--r-- 1 root root   92 Aug 19  2013 devpts
-rw-r--r-- 1 root root   88 Dec 15  2012 dns2tcp
-rw-r--r-- 1 root root 1018 Jan  8  2014 exim4
-rw-r--r-- 1 root root  406 Jan  8  2014 extlinux
-rw-r--r-- 1 root root  902 Dec  3  2013 greenbone-security-assistant
-rw-r--r-- 1 root root 1201 Jan  8  2014 grub
-rw-r--r-- 1 root root   86 Aug 19  2013 halt
-rw-r--r-- 1 root root  855 Nov  2  2008 hdparm
-rw-r--r-- 1 root root  657 Dec 11  2012 hwclock
```

This directory contains many files and scripts that configure various daemons or services in Linux. The only one we are interested here is the **grub** file that I highlighted in the screenshot above. Let's open that file with the more command.

kali > more /etc/default/grub

When we do so, we will see the following output.

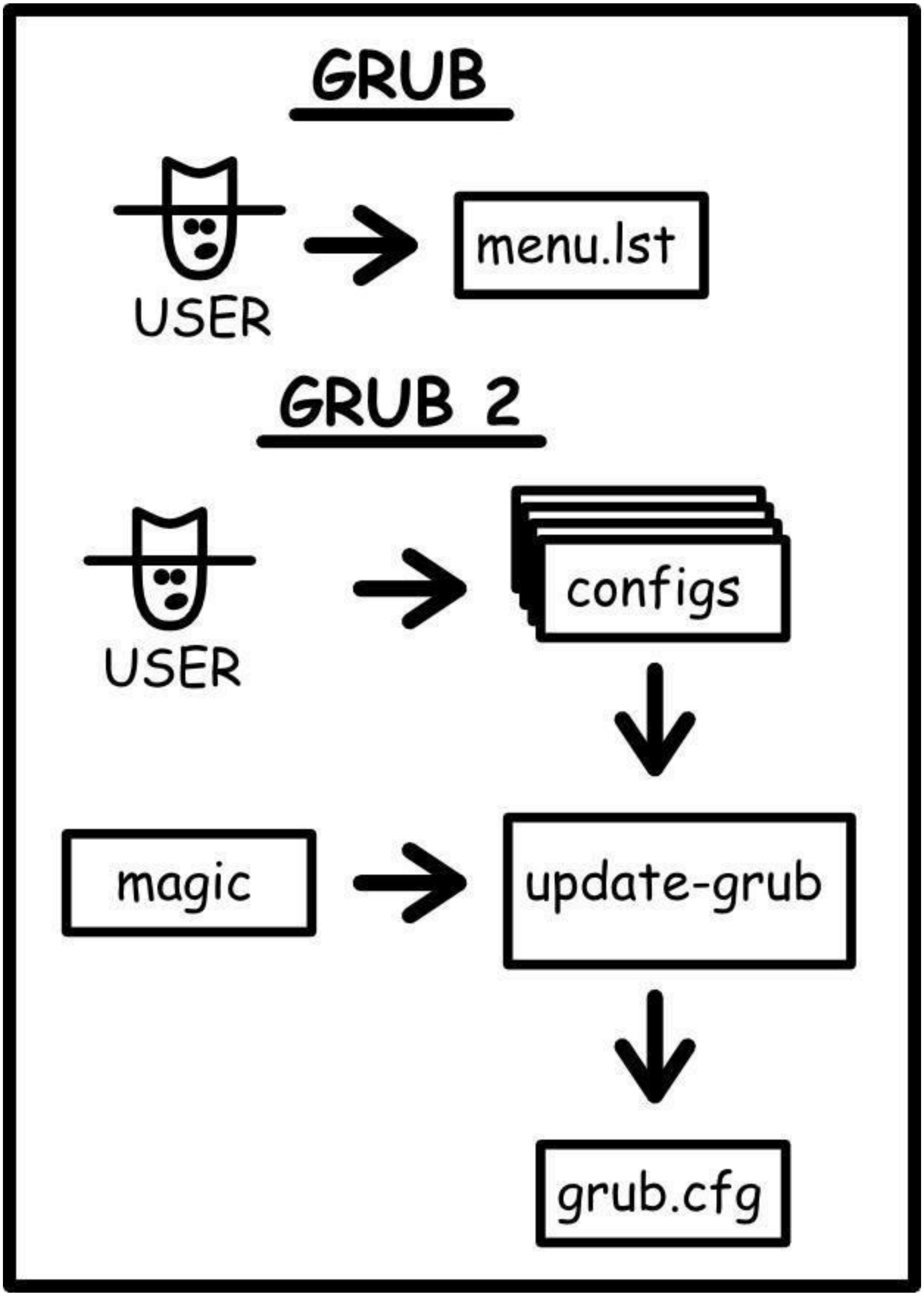


```
root@kali:/etc/default# more /etc/default/grub
# If you change this file, run 'update-grub' afterwards to update
# /boot/grub/grub.cfg.
# For full documentation of the options in this file, see:
#   info -f grub -n 'Simple configuration'

GRUB_DEFAULT=0
GRUB_TIMEOUT=5
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="quiet"
GRUB_CMDLINE_LINUX="initrd=/install/initrd.gz"

# Uncomment to enable BadRAM filtering, modify to suit your needs
# This works with Linux (no patch required) and with any kernel that obtains
# the memory map information from GRUB (GNU Mach, kernel of FreeBSD ...)
#GRUB_BADRAM="0x01234567,0xfefefefe,0x89abcdef,0xefefefef"

# Uncomment to disable graphical terminal (grub-pc only)
#GRUB_TERMINAL=console

# The resolution used on graphical terminal
# note that you can use only modes which your graphic card supports via VBE
# you can see them in real GRUB with the command `vbeinfo'
```

This file contains many of the customization parameters for GRUB such as the TIMEOUT and other parameters. If you change anything it this file, you must run "update-grub" for those changes to take effect as it then makes changes to the **grub.cfg** file.

Step 4 How GRUB2 Works

Before we go any further, I want to take a moment to detail how GRUB2 works. It's quite different than the original GRUB and you need to understand this before you attempt any edits or changes to GRUB2.


GRUB
USER
menu.lst
GRUB 2
USER
configs
magic
update-grub
grub.cfg

The differences in configuring GRUB and GRUB2. *Image via jEriko*

The **/etc/default/grub** file contains customization parameters.

The **/etc/grub.d/** directory contains scripts for the GRUB menu information and the scripts to boot the various operating systems. When the **update-grub** command is run, it reads the contents of the grub file and the **grub.d** scripts and creates the grub.cfg file.

To change the **grub.cfg** file, you need to edit the grub file or the scripts under **grub.d**.

Samba

Those of you who use Windows in a LAN environment understand that Windows machines can share directories, files, printers, etc. using "shares." This protocol dates back to the 1980s when the then dominant computer firm, IBM, developed a way for computers to communicate over the LAN by just using computer names rather than MAC or IP addresses.

Eventually, Microsoft and Intel developed a similar protocol for file sharing, originally named NetBIOS, and eventually renamed Server Message Block (SMB). By 1992, Andrew Tridgell, reverse-engineered SMB for Linux/Unix (including Apple's Mac OS X) and named it Samba. It is a server-side implementation of SMB and requires no software to be installed on the client. Samba provides:

- File sharing
- Network printing
- Authentication and authorization
- Name resolution
- Service announcement (browsing)

Samba functionality is implemented by two daemons, **smbd** and **nmbd**. These daemons (services) are installed and run on nearly every distribution of Unix and Linux. Samba, like Windows NetBios/SMB, runs on ports 135, 137, and 445.

Just like Window's SMB, Linux's Samba has been one of the weakest and most often exploited protocols. There is a long list of vulnerabilities and exploits that take advantage of Linux's Samba, and when I want to exploit a Linux system, one of the first things I test is Samba. Samba 3.6.3 and earlier versions allow anonymous users to gain root access through Samba 's remote procedure call.

The more we know and understand Samba, the better network admin's we will be and the better Linux hackers we will be. Let's take a little time to understand this essential and, often, very vulnerable protocol in Linux.

Step 1 Locate Samba

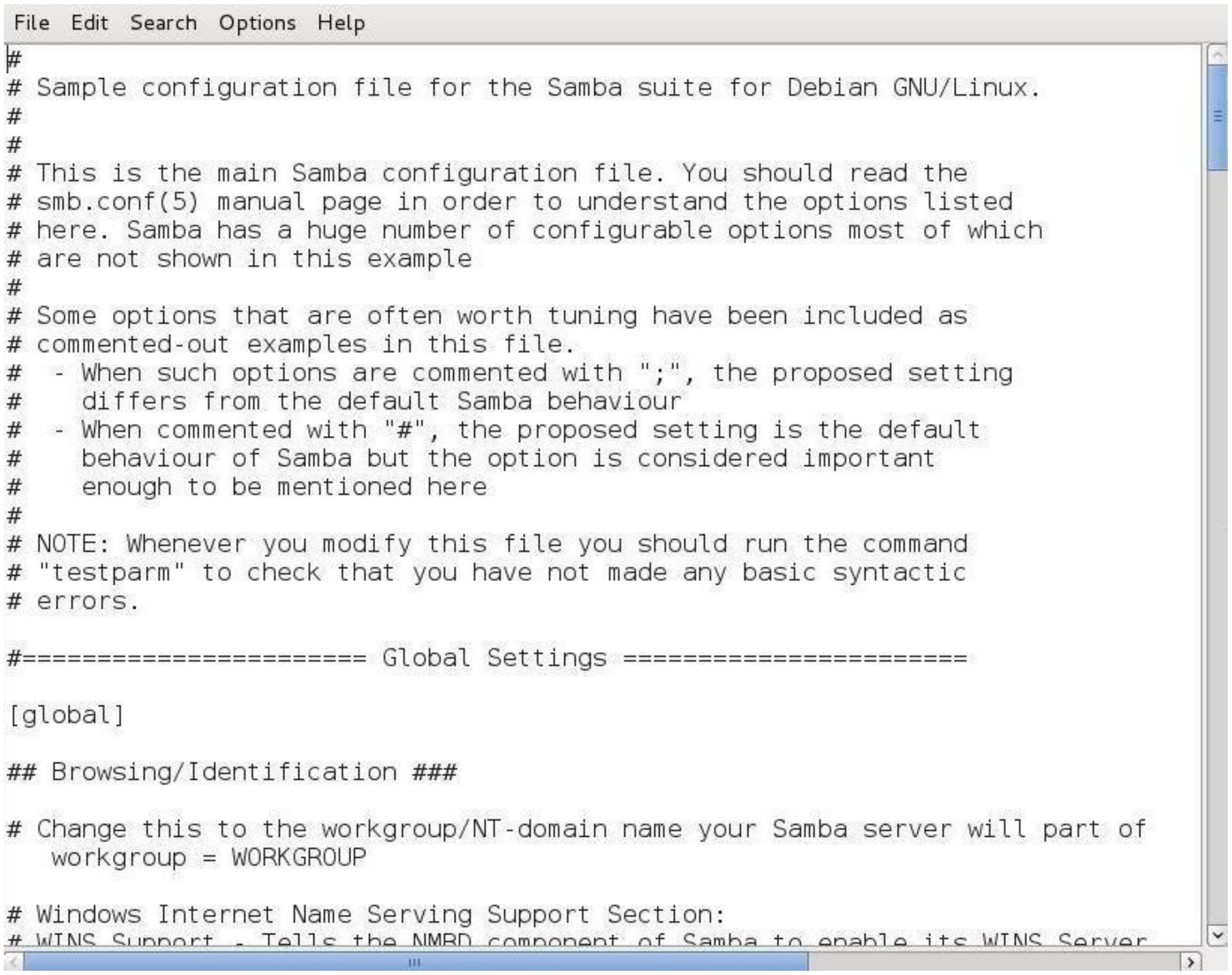
To find Samba files in our Kali, let's just type:

- **locate smb**

When you do so, you will see dozens of files with SMB or Samba in their name, including numerous Metasploit modules for exploiting Samba. Near the top of that list is the main configuration file for Samba, */etc/samba/smb.conf*. Like nearly every application or daemon in Linux, there is a configuration file located in the */etc* directory. These are simple text files that can be edited and saved to alter the configuration of the application or daemon. Samba is no different.

Let's open that now with the text editor of your choice. In this case, I'll use Leafpad in Kali.

- **kali > leafpad /etc/samba/smb.conf**



```
#
# Sample configuration file for the Samba suite for Debian GNU/Linux.
#
#
# This is the main Samba configuration file. You should read the
# smb.conf(5) manual page in order to understand the options listed
# here. Samba has a huge number of configurable options most of which
# are not shown in this example
#
# Some options that are often worth tuning have been included as
# commented-out examples in this file.
#  - When such options are commented with ";", the proposed setting
#    differs from the default Samba behaviour
#  - When commented with "#", the proposed setting is the default
#    behaviour of Samba but the option is considered important
#    enough to be mentioned here
#
# NOTE: Whenever you modify this file you should run the command
# "testparm" to check that you have not made any basic syntactic
# errors.

#======================= Global Settings =======================

[global]

## Browsing/Identification ###

# Change this to the workgroup/NT-domain name your Samba server will part of
   workgroup = WORKGROUP

# Windows Internet Name Serving Support Section:
# WINS Support - Tells the NMBD component of Samba to enable its WINS Server
```

As you can see, this configuration file is well commented. For more information on configuring Samba, you can also type:

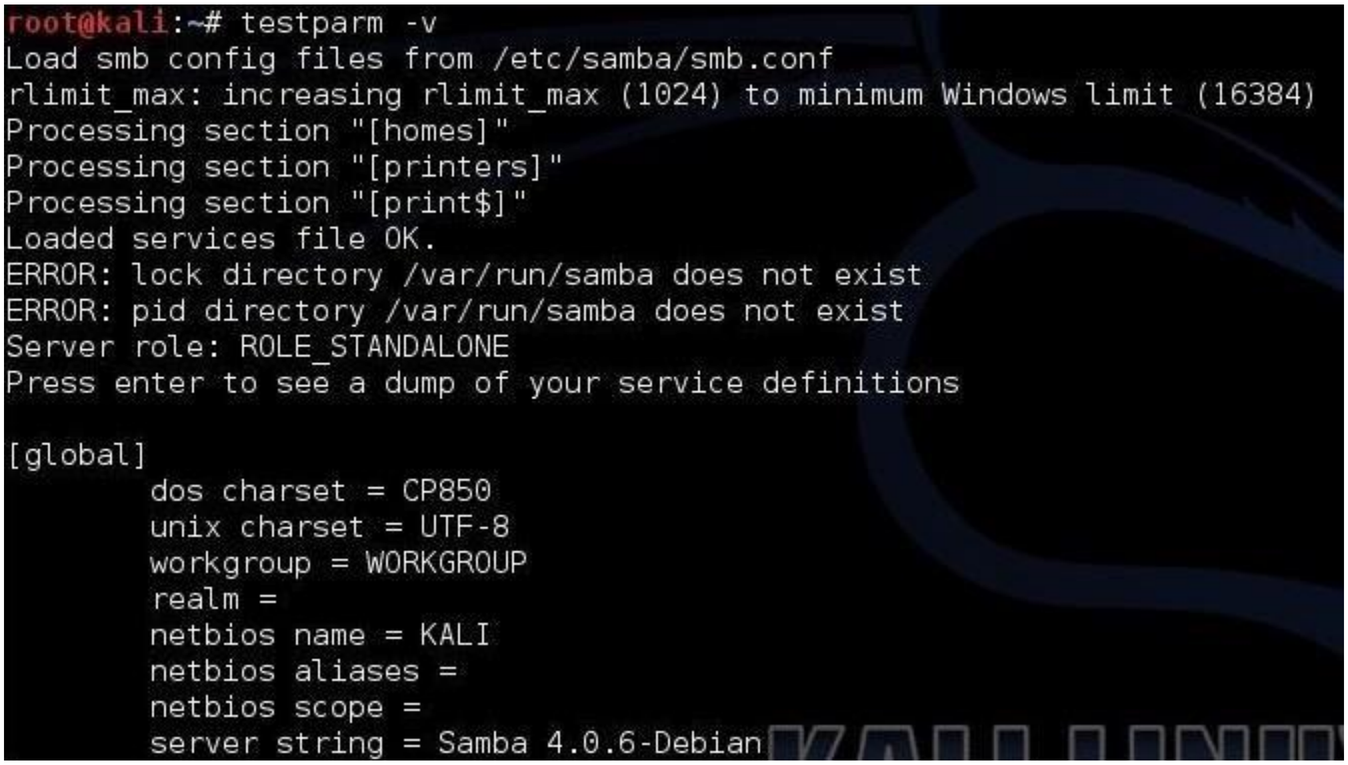
- **kali > man samba**

I should point out now that this configuration file uses two different types of comments. The first is the all familiar "**#**" , but it also the uses "**;**" too. You will see both in this configuration file.

Step 2 Configure Samba

Samba has a command that allows you to test the syntax of your configuration file. This is:

- **kali > testparm -v**



```
root@kali:~# testparm -v
Load smb config files from /etc/samba/smb.conf
rlimit_max: increasing rlimit_max (1024) to minimum Windows limit (16384)
Processing section "[homes]"
Processing section "[printers]"
Processing section "[print$]"
Loaded services file OK.
ERROR: lock directory /var/run/samba does not exist
ERROR: pid directory /var/run/samba does not exist
Server role: ROLE_STANDALONE
Press enter to see a dump of your service definitions

[global]
        dos charset = CP850
        unix charset = UTF-8
        workgroup = WORKGROUP
        realm =
        netbios name = KALI
        netbios aliases =
        netbios scope =
        server string = Samba 4.0.6-Debian
```

When you type this command, it checks your parameters and syntax to see whether they are workable.

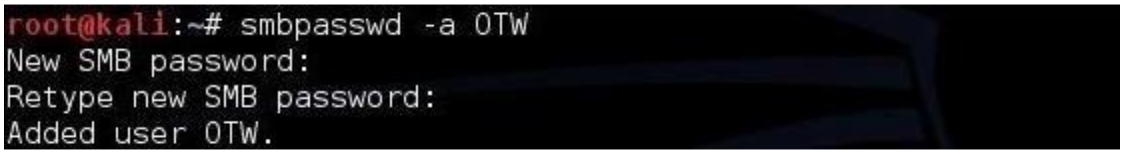
It is important to note that whenever you make changes to this configuration file, you must restart Samba for them to be implemented. You can do this with:

- **kali > service smb restart**

Step 3 Create a Samba User

Samba has its own password authentication system. You can create users with access to Samba, if they exist, in the */etc/passwd* file (in other words, they are a user on the system) with the command:

- **kali > smbpasswd -a <username>**

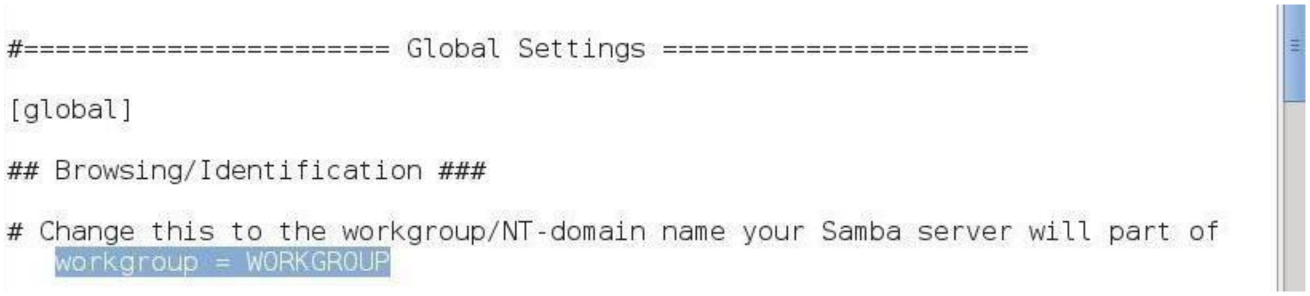


```
root@kali:~# smbpasswd -a OTW
New SMB password:
Retype new SMB password:
Added user OTW.
```

As you can see, I have added myself, OTW, to the Samba user list with my password.

Step 4 Setting the WorkGroup

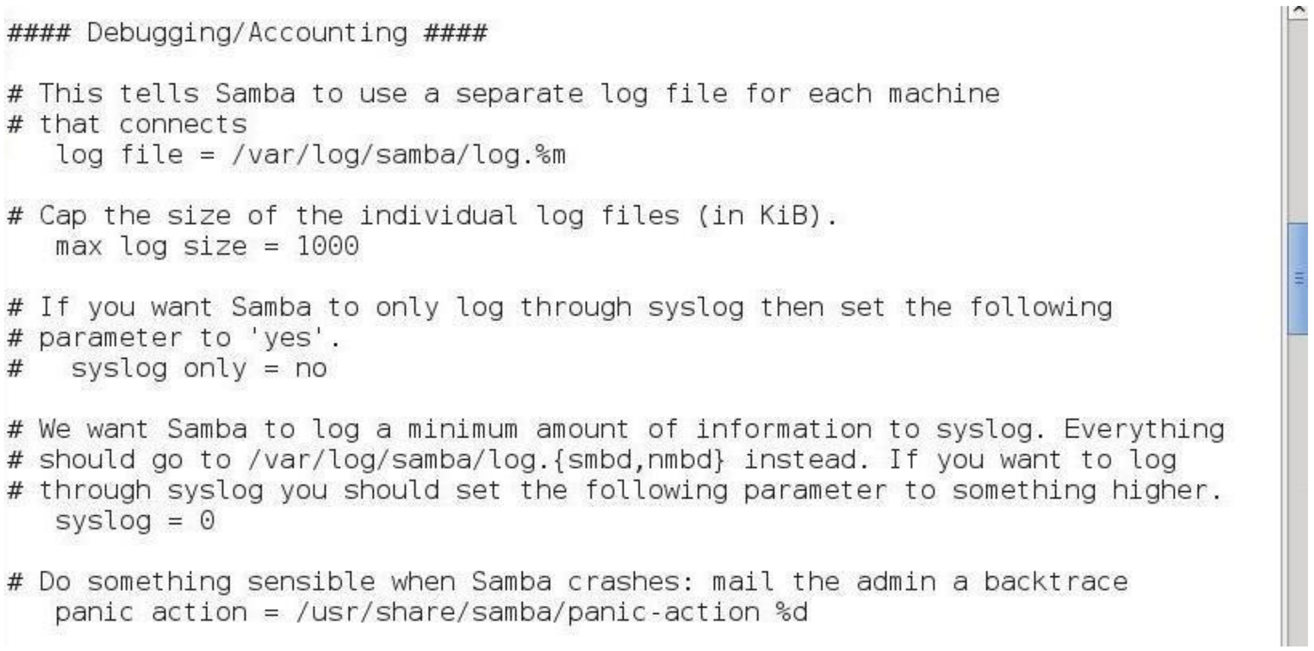
If you want to connect to the workgroup from a Windows machine, you will need the name of the Windows workgroup. In the "Global Settings," you can see the default is set to workgroup but, of course, this should set to the name of the Windows workgroup that Samba will be connected to.



```
#======================= Global Settings =======================

[global]

## Browsing/Identification ###

# Change this to the workgroup/NT-domain name your Samba server will part of
   workgroup = WORKGROUP
```

Step 5 Accounting/Logging

Next, let's navigate down within the Global Settings to the "Debugging/Accounting" section.



```
#### Debugging/Accounting ####

# This tells Samba to use a separate log file for each machine
# that connects
   log file = /var/log/samba/log.%m

# Cap the size of the individual log files (in KiB).
   max log size = 1000

# If you want Samba to only log through syslog then set the following
# parameter to 'yes'.
#   syslog only = no

# We want Samba to log a minimum amount of information to syslog. Everything
# should go to /var/log/samba/log.{smbd,nmbd} instead. If you want to log
# through syslog you should set the following parameter to something higher.
   syslog = 0

# Do something sensible when Samba crashes: mail the admin a backtrace
   panic action = /usr/share/samba/panic-action %d
```

Here you can see we can set the location of the log file:

- **log file = /var/log/samba/log.%m**

The maximum log size (KB):

- **max log size = 1000**

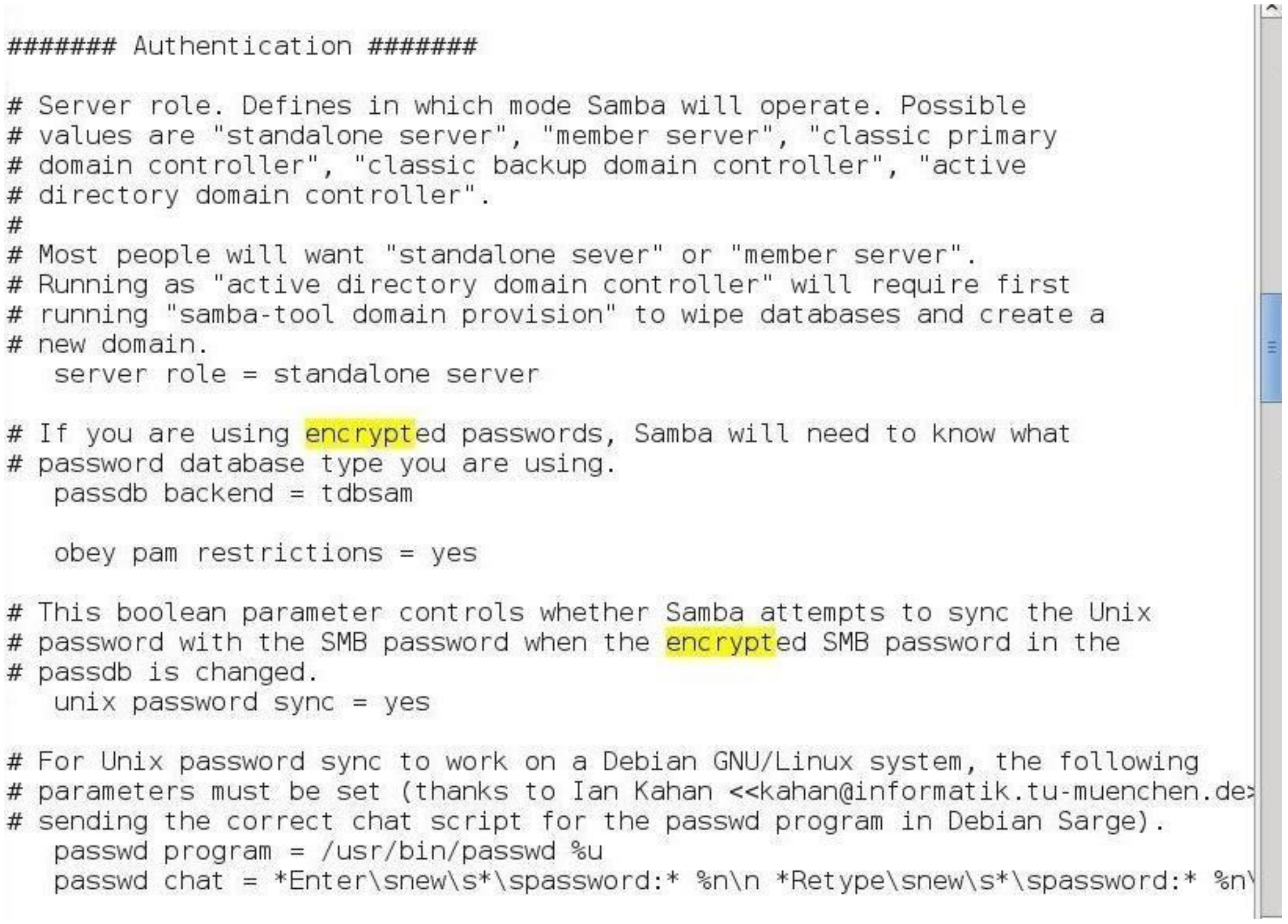
And whether to only use syslog to centralize our logging to a dedicated logging system:

- **syslog only = no**

Among many other things.

Step 6 Authentication

If we navigate a bit lower in the Global Settings, we come to the "Authentication" section.



```
####### Authentication #######

# Server role. Defines in which mode Samba will operate. Possible
# values are "standalone server", "member server", "classic primary
# domain controller", "classic backup domain controller", "active
# directory domain controller".
#
# Most people will want "standalone sever" or "member server".
# Running as "active directory domain controller" will require first
# running "samba-tool domain provision" to wipe databases and create a
# new domain.
   server role = standalone server

# If you are using encrypted passwords, Samba will need to know what
# password database type you are using.
   passdb backend = tdbsam

   obey pam restrictions = yes

# This boolean parameter controls whether Samba attempts to sync the Unix
# password with the SMB password when the encrypted SMB password in the
# passdb is changed.
   unix password sync = yes

# For Unix password sync to work on a Debian GNU/Linux system, the following
# parameters must be set (thanks to Ian Kahan <<kahan@informatik.tu-muenchen.de>
# sending the correct chat script for the passwd program in Debian Sarge).
   passwd program = /usr/bin/passwd %u
   passwd chat = *Enter\snew\s*\spassword:* %n\n *Retype\snew\s*\spassword:* %n\
```

Here we can set what type of server Samba will be:

- **server role = standalone server**

Whether Samba should obey/use pluggable authentication modules (PAM):

- **obey pam restrictions = yes**

Whether Samba syncs passwords with the Linux/Unix password system:

- **unix password sync = yes**

And finally, how Samba handles password setting.

Step 7 Samba Variables

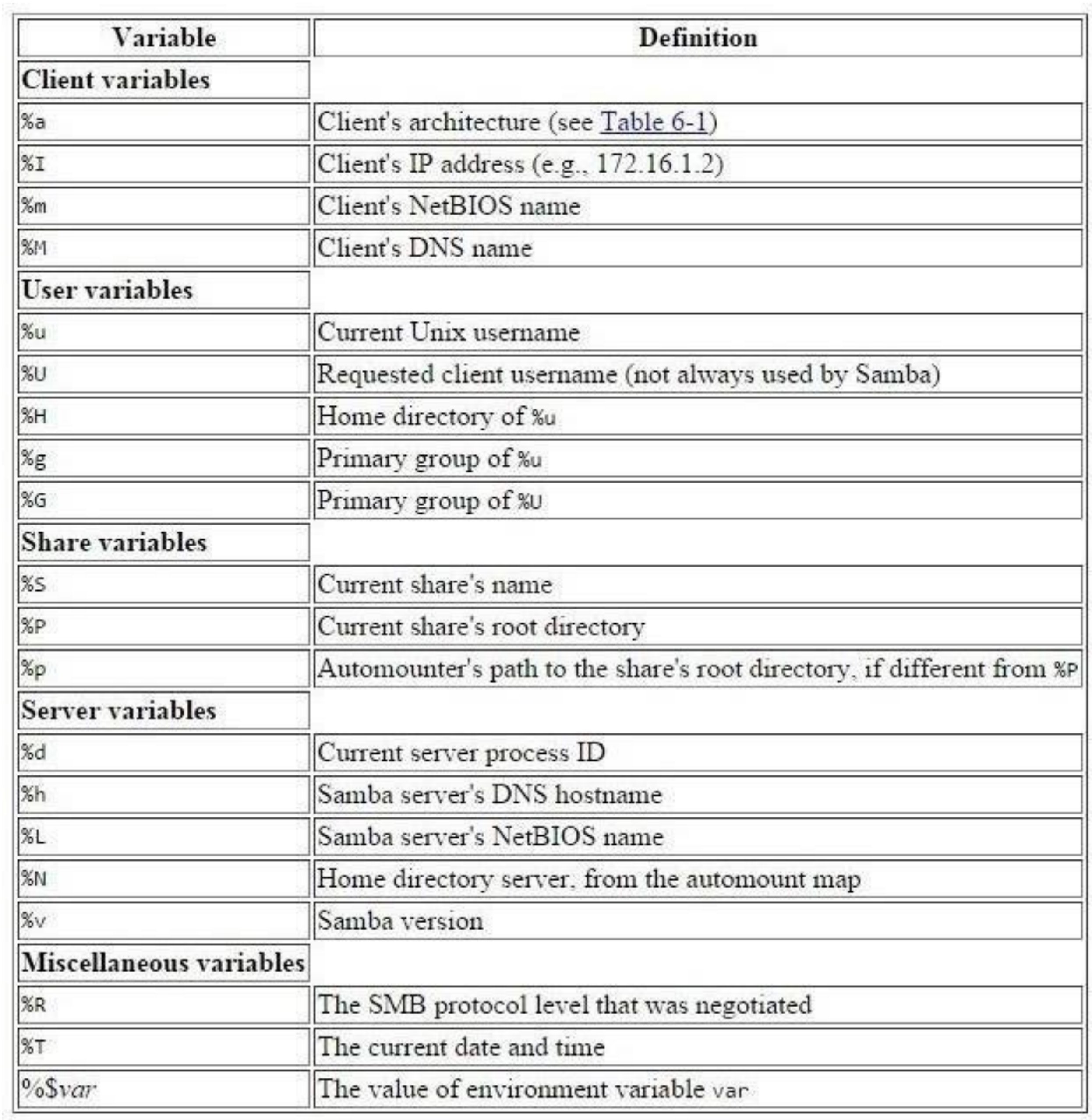
You will notice that throughout the Samba configuration file, variables are used that begin with "%". For instance, in the logging section above, we saw this line:

- **log file = /var/log/samba/log.%m**

The "%m" at the end represents a variable. If we use the table below, we can see that the "%m" represents the "The client's NetBIOS name." If the NetBIOS name of the computer were "NullByte," this would mean that the logs would be found for that system at:

- **log file = /var/log/samba/log.NullByte**

When we replaced the variable with the value of the NetBIOS name.



Variable	Definition
Client variables	
%a	Client's architecture (see Table 6-1)
%I	Client's IP address (e.g., 172.16.1.2)
%m	Client's NetBIOS name
%M	Client's DNS name
User variables	
%u	Current Unix username
%U	Requested client username (not always used by Samba)
%H	Home directory of %u
%g	Primary group of %u
%G	Primary group of %U
Share variables	
%S	Current share's name
%P	Current share's root directory
%p	Automounter's path to the share's root directory, if different from %P
Server variables	
%d	Current server process ID
%h	Samba server's DNS hostname
%L	Samba server's NetBIOS name
%N	Home directory server, from the automount map
%v	Samba version
Miscellaneous variables	
%R	The SMB protocol level that was negotiated
%T	The current date and time
%$*var*	The value of environment variable var

Although there are far more changes we can make to this configuration file, in many cases the default setting will suffice in most circumstances, although not optimally. I'll be doing a second Samba tutorial in the near future where we will configure Samba for optimal file sharing, so keep coming back, my aspiring hackers.

Logging

When you are using and administering Linux, it is important to be conversant in the use of the log files. As you know, log files are the repository for much information about our system, including errors and security alerts.

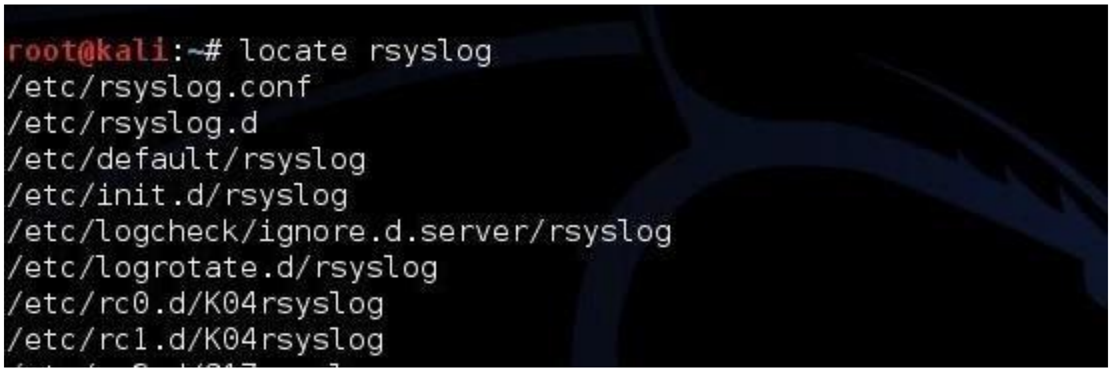
If we are trying to secure Linux, it is crucial to know how to manage the logging functions to be able to know if your system has been attacked, and to then decipher what actually happened and who did it. If you are the attacker, it is crucial to understand what information can be gathered about you and your methods from these same log files so you can avoid leaving any trace behind.

Generally, Linux uses a daemon (a program that runs in the background) called syslogd that is a unified system for logging events on your computer. Several variations of syslog exist, including rsyslog and syslog-ng. Although they operate very similarly, they do have some minor differences. Since our Kali Linux is built on Debian, it comes with rsyslog by default, so that is what we will be using in this tutorial.

Step 1 Open a Terminal in Kali

Let's begin by opening a terminal in terminal. To find our rsyslog, we can simply type:

kali > locate rsyslog

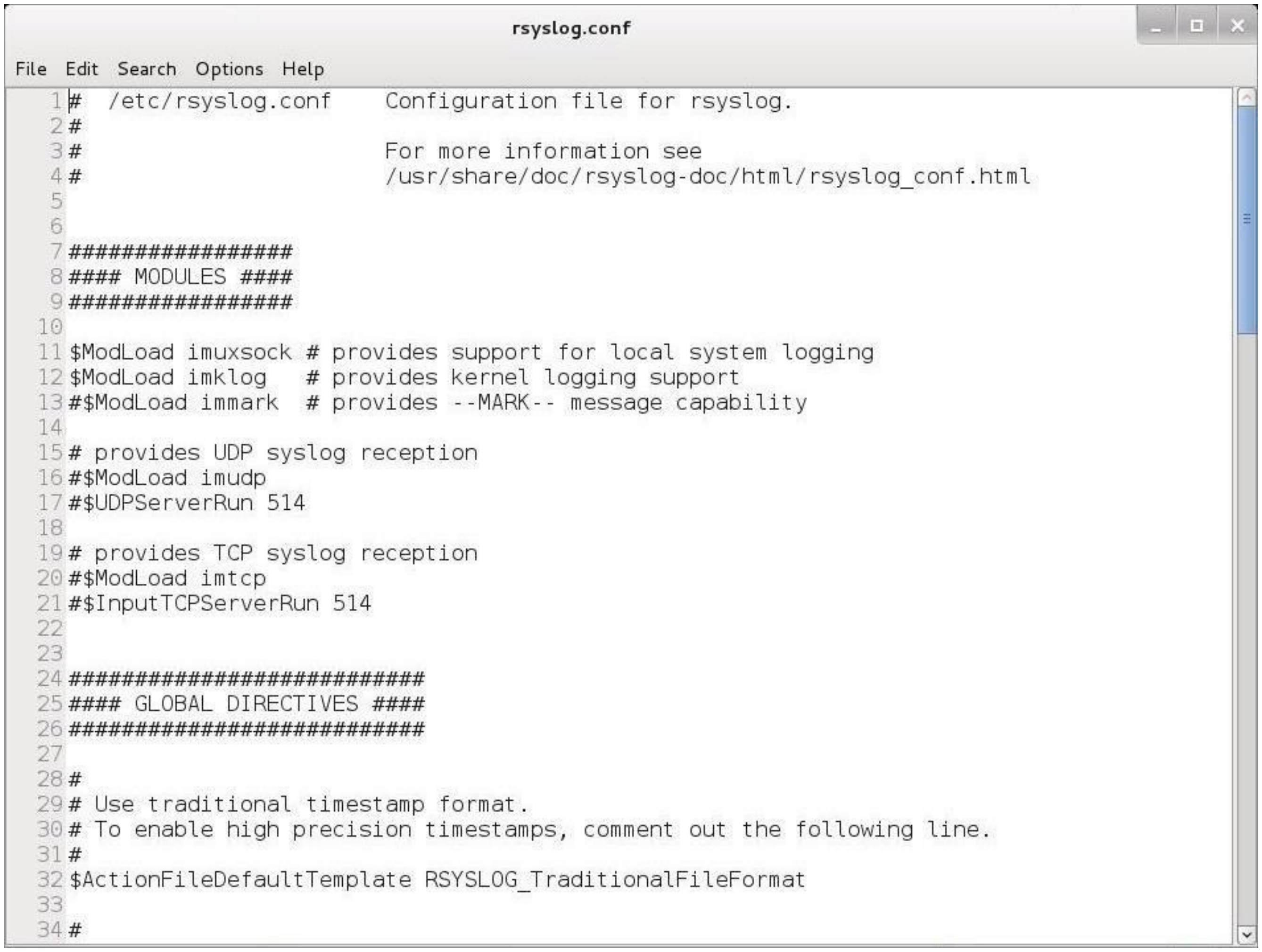


```
root@kali:~# locate rsyslog
/etc/rsyslog.conf
/etc/rsyslog.d
/etc/default/rsyslog
/etc/init.d/rsyslog
/etc/logcheck/ignore.d.server/rsyslog
/etc/logrotate.d/rsyslog
/etc/rc0.d/K04rsyslog
/etc/rc1.d/K04rsyslog
```

Step 2 Open the Rsyslog Configuration File (rsyslog.conf)

Like nearly every application in Linux, rsyslog is managed and configured by a plain text configuration file. As you hopefully learned in earlier Linux tutorials, generally, configuration files are located in the */etc* directory. In the case of rsyslog, it is located at */etc/rsyslog.conf*. Let's open that file with any text editor (here I will use Leafpad).

kali > leafpad /etc/rsyslog.conf



```
#  /etc/rsyslog.conf    Configuration file for rsyslog.
#
#                       For more information see
#                       /usr/share/doc/rsyslog-doc/html/rsyslog_conf.html


#################
#### MODULES ####
#################

$ModLoad imuxsock # provides support for local system logging
$ModLoad imklog   # provides kernel logging support
#$ModLoad immark  # provides --MARK-- message capability

# provides UDP syslog reception
#$ModLoad imudp
#$UDPServerRun 514

# provides TCP syslog reception
#$ModLoad imtcp
#$InputTCPServerRun 514


###########################
#### GLOBAL DIRECTIVES ####
###########################

#
# Use traditional timestamp format.
# To enable high precision timestamps, comment out the following line.
#
$ActionFileDefaultTemplate RSYSLOG_TraditionalFileFormat

#
```

As you can see above, the rsyslog.conf file comes well documented with numerous comments explaining its use.

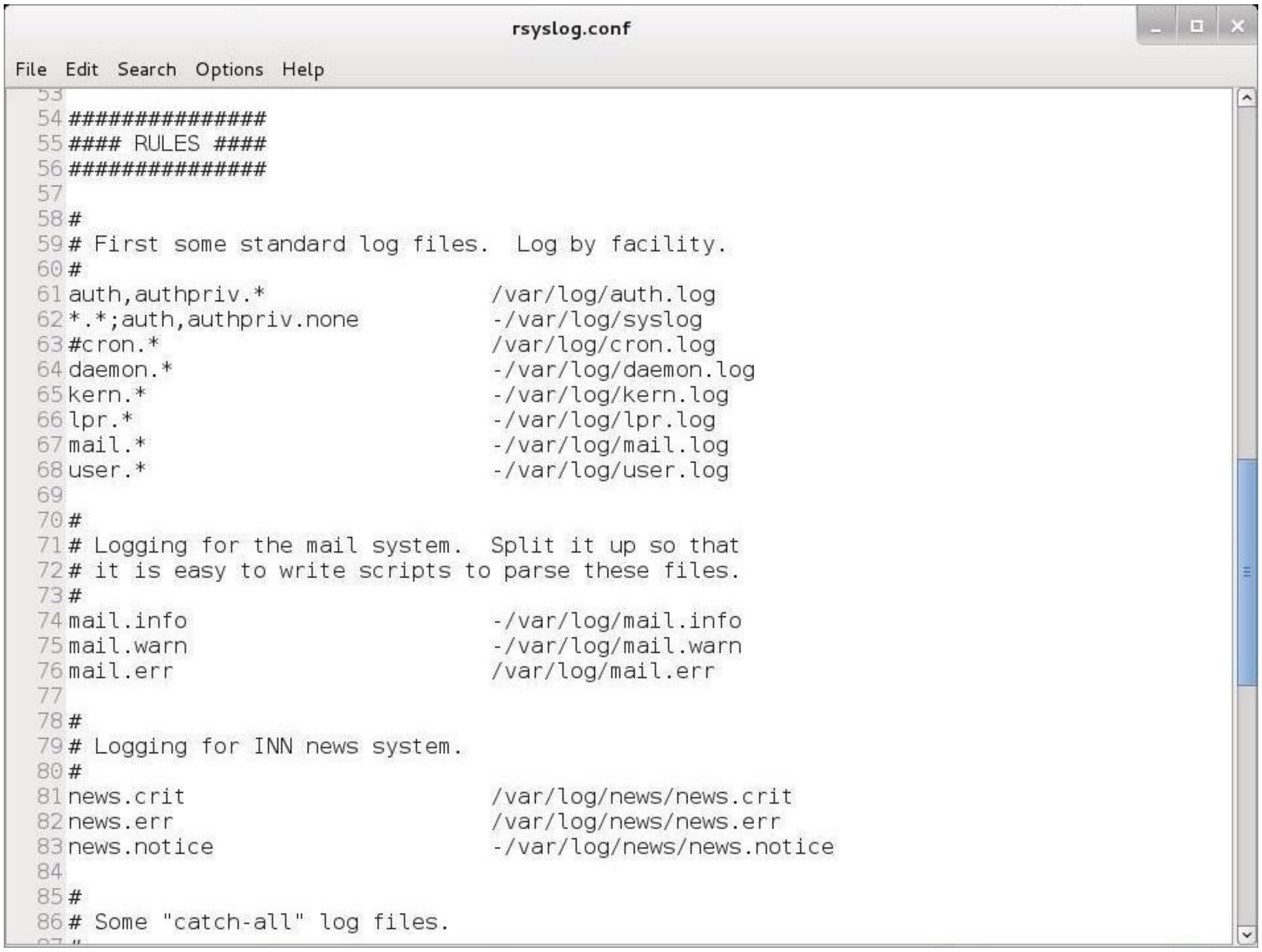
Step 3 Format of the Rsyslog Configuration File

Let's navigate down a bit to below Line 50. Here begins the "Rules" section. This is where we can set the rules for what Linux logs.

The basic format for these rules is:

facility.priority action

The **facility** word references the program (such as *mail*, *kernel*, *lpr*, etc.) that generates the message to be logged. The **priority** keyword references the importance of the log message. Finally, the **action** keyword references the location that the log is to be sent to. This can be a file, remote computer, etc.



```
###############
#### RULES ####
###############

#
# First some standard log files.  Log by facility.
#
auth,authpriv.*                 /var/log/auth.log
*.*;auth,authpriv.none          -/var/log/syslog
#cron.*                         /var/log/cron.log
daemon.*                        -/var/log/daemon.log
kern.*                          -/var/log/kern.log
lpr.*                           -/var/log/lpr.log
mail.*                          -/var/log/mail.log
user.*                          -/var/log/user.log

#
# Logging for the mail system.  Split it up so that
# it is easy to write scripts to parse these files.
#
mail.info                       -/var/log/mail.info
mail.warn                       -/var/log/mail.warn
mail.err                        /var/log/mail.err

#
# Logging for INN news system.
#
news.crit                       /var/log/news/news.crit
news.err                        /var/log/news/news.err
news.notice                     -/var/log/news/news.notice

#
# Some "catch-all" log files.
```

Facilities

The valid codes to put in place of the facility keyword in our configuration file rules include:

- *auth*
- *authpriv*
- *daemon*
- *kern*
- *lpr*
- *mail*
- *mark*
- *news*

- *security*
- *syslog*
- *user*
- *uucp*

An asterisk (*) refers to all facilities. You can select more than one facility by listing them separated by a comma.

Priorities

The valid codes for priority are:

- *debug*
- *info*
- *notice*
- *warning*
- *warn*
- *error*
- *err*
- *crit*
- *alert*
- *emerg*
- *panic*

The priority codes are listed from lowest (*debug*) priority to highest (*emerg*, *panic*). The *warning* is the same as *warn*, *error* is the same as *err*, and *emerg* is the same as *panic*. *Error*, *warn*, and *panic* are all deprecated and should not be used.

For instance, if you specify a priority code of *alert*, the system will log messages that are classified as *alert* or *emerg*, but not *crit* or below.

Actions

The action is usually a file name with its location. For instance, */var/log/messages*.

Step 4 Examples of Facility.Priority Action

Let's look at some examples.

mail.* /var/log/mail

This example will log *mail* events of all (*) priorities to */var/log/mail*.

kern.crit /var/log/kernel

This example will log kernel (*kern*) events of critical (*crit*) priority or higher to *var/log/kernel*.

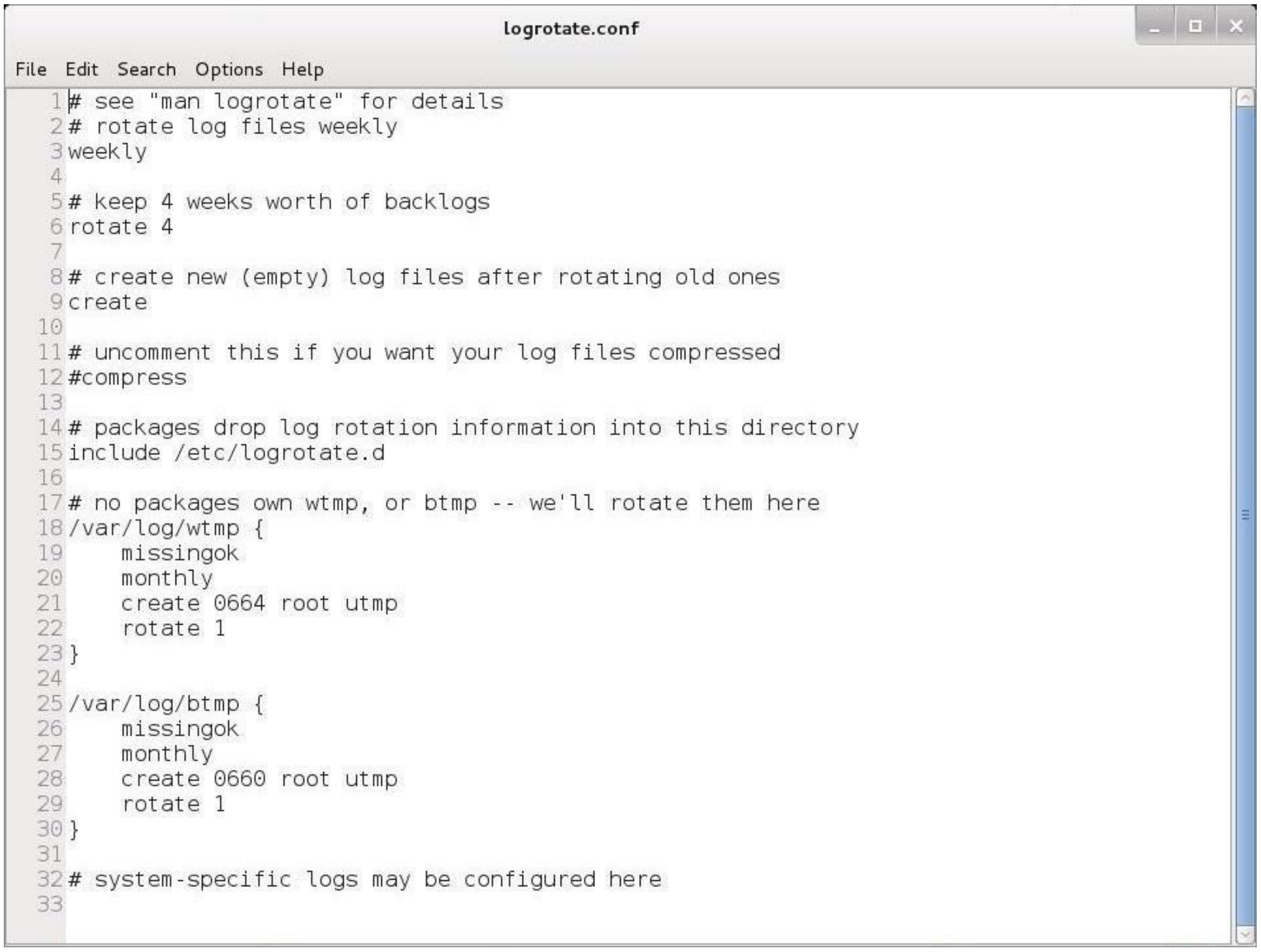
****.emerg ****

This example will log all events (*) of the emergency priority (*emerg*) to all logged on users.

Step 5 Logrotate

If you don't delete your log files, they will eventually fill your entire hard drive. On the other hand, if you delete your log files too frequently, you will not have them for an investigation at some future point in time. We can use **logrotate** to determine the balance between these opposing requirements.

We can rotate our log files by creating a cron job that periodically rotates our log files through the logrotate utility. The logroate utility is configured with the */etc/logrotate.conf*. Let's open it with a text editor and take a look at it.



```
 1 # see "man logrotate" for details
 2 # rotate log files weekly
 3 weekly
 4
 5 # keep 4 weeks worth of backlogs
 6 rotate 4
 7
 8 # create new (empty) log files after rotating old ones
 9 create
10
11 # uncomment this if you want your log files compressed
12 #compress
13
14 # packages drop log rotation information into this directory
15 include /etc/logrotate.d
16
17 # no packages own wtmp, or btmp -- we'll rotate them here
18 /var/log/wtmp {
19     missingok
20     monthly
21     create 0664 root utmp
22     rotate 1
23 }
24
25 /var/log/btmp {
26     missingok
27     monthly
28     create 0660 root utmp
29     rotate 1
30 }
31
32 # system-specific logs may be configured here
33
```

In most cases, the default configuration will work for most people, but note on Line 6 that the default setting is to rotate logs every 4 weeks. If you want to keep your logs for a longer or shorter time, this is the setting you should change

The Linux Philosophy

I'd like to take this moment to explain the philosophy around the Linux operating system.

When I use the term "philosophy," I am not referring to such questions as "what is the meaning of life" or "does God exist," but rather what was the underlying logic and reasoning behind the design of this ubiquitous and love-lived operating system.

As many of you already know, I am strong advocate for the Linux operating system. Although Linux may be ideally suited to hacking and many other applications, I think it is important to understand the philosophy underlying the Linux/Unix structure and model for *any* environment.

In this article, I will use the term Unix/Linux to designate this operating system. Unix was the original, developed by Thompson and Ritchie, and Linux was a re-engineer of Unix by Linux Torvalds and team. Mac OS X, iOS, Android, Solaris, AIX, HP-UX, and IRIX are all forms of Unix/Linux.

In addition, Red Hat, Ubuntu, Mint, Fedora, Debian, Slackware, and SUSE are all distributions of Linux. A distribution of Linux is simply an operating system that uses the Linux kernel, but then adds in its own additional components. These components vary, but may include applications, utilities, modules, the GUI, and others.

This variability in the distributions is often confusing and frustrating to the novice, but it is actually part of the Linux beauty and strength. Unix/Linux are designed to be flexible and portable, allowing the end-user to work the way they are comfortable, rather than the way the software developer thinks you should work.

Unix was first developed in the early 1970s by Dennis Ritchie and Ken Thompson at AT&T Labs. The fact that it is still being used over 40 years later tells you something about the quality, durability, and efficiency of this operating system. These guys did something right! How many things in computing are still around from the early 1970s?

If anything, rather than this "ancient" operating system fading away, it is gaining ground nearly every day. Chrome, Android, iOS, Linux, and Mac OS X are all based on this 40-year-old operating system. If we look at the fastest growing market—mobile devices—it is **dominated** by Unix variants with iOS and Android compromising over 91% of the market. It appears that the mobile market in the near future will be nearly 100% Unix/Linux.

What about this modest operating system has made it this durable and long-lasting? Let's take a look then at some of the tenants of this design philosophy that has made Linux so successful.

Assume the User Is Computer Literate

The developers of Unix (and thereby Linux) made a radical assumption: That the users are computer literate. We can't say the same for many other operating systems. In many cases, the operating system developers assume we are ignorant, illiterate Neanderthals who need to be protected ourselves. Not so with Unix/Linux.

As one sage said, "Unix(Linux) was not designed to stop its users from doing stupid things as that would also keep them from doing clever things."

Perfect! Could not have said it better myself!

Complete Control

One of key reasons that hackers use Linux and only Linux, is that it gives us complete control. Other operating systems try to hide some of their operations and features from us, afraid we will screw things up. Linux is totally transparent and enables us to see and use everything.

Choose Portability over Efficiency

Unix was the first portable operating system, meaning it could be used on many different hardware platforms. This has served it well as Unix/Linux has now been ported and compiled for over near 60 hardware platforms. This has been a critical element in its longevity and ability to adopt to an ever-changing technological environment.

Store Data in Flat Text Files

Unix/Linux stores data in flat text files unlike other operating systems. This makes the data as portable, or more portable, than the code itself. Nearly all systems can import and use flat text files.

Use Shell Scripts to Increase Leverage & Portability

Shell scripts enhance the power of our applications. By writing a script, we can automate an application to do something as many times as we would like, as well as leverage the capabilities of other applications simultaneously. In addition, these scripts are then portable to other systems without having to recompile them.

Allow the User to Tailor Their Environment

Unix/Linux was designed to allow the user to tailor their environment to their liking. The user is in control and not the software developer. Unix/Linux implements mechanisms for doing things, but they don't dictate *how* you do things. This tailoring can take many forms including the graphical user interface (GUI). There are numerous GUIs available for Linux including GNOME (the default on Kali and the most widely used), KDE, Unity (Ubuntu's default), Sugar, Trinity, Xfce, Enlightenment, and many more. In most cases, despite the default GUI that might come with your system, you can install and use any one of the other interfaces, if you please.

Make the Kernel Small & Lightweight

Although many operating system kernels continue to add features to the main kernel to offer users greater capability, they make it more and more bloated. The Unix/Linux model is to keep the kernel small and lightweight, but allow the developers and users to add components and modules as they please.

User Lowercase & Keep It Short

lowercase names and commands are a unix/linux tradition.

Silence Is Golden

Unix/Linux commands tend to be silent when you have done things correctly. This can drive some new users a bit batty when they, for instance, copy a file from one location to another and Unix/Linux has nothing to say. Not a confirmation or even a pat on the back.

Think Hierarchically

The Unix/Linux operating system was the first to develop a file system organized into a hierarchical tree. This hierarchical thinking has extended into many other areas of the operating system, such as networking and object-oriented programming.

I hope this little foray into the philosophy of Linux helps you to understand why Linux is so different than those other operating systems. The result of this philosophy is an operating system that is small, lightweight, and flexible, which treats all users with respect.

Inetd, the Super Daemon

In my ongoing attempts to familiarize aspiring hackers with Linux (nearly *all* hacking is done with Linux, I want to address a rather obscure, but powerful process. There is one *super* process that is called **inetd** or **xinetd** or **rlinetd**. I know, I know... that's confusing, but bear with me.

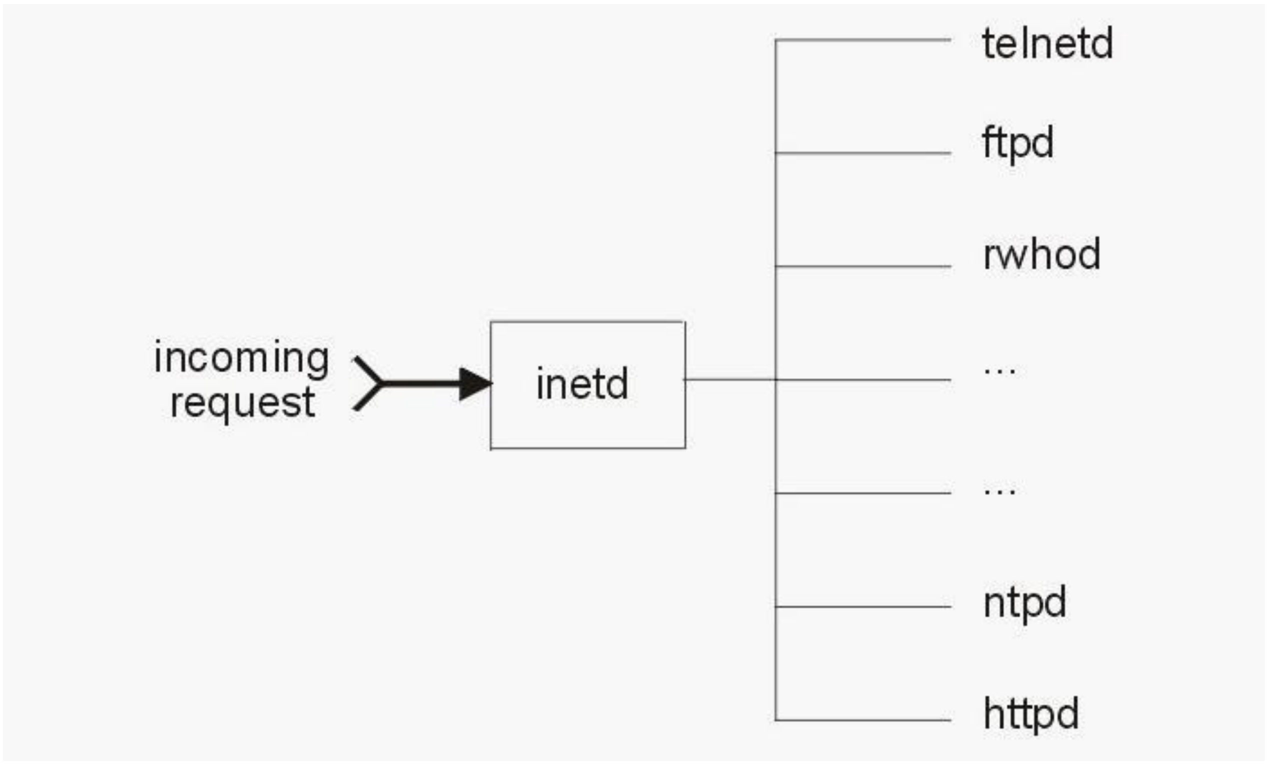
Before I discuss the inetd process, I want to explain that in Linux and Unix, processes that run in the background are called daemons. In some places, you we will even see them referred to as "demons," but that is incorrect. A daemon is a spirit that influences one's character or personality. They are not representatives of good or evil, but rather encourage independent thought and will. This is in contrast to a "demon," which we know is something quite different.

Image via Unknown

Now, back to inetd. In the beginning—well, at least in the beginning of Unix—all daemons started at boot time and ran continuously. As you might imagine, this meant that processes that were not being used were using resources and depleting performance. This obviously was an inefficient way of doing business.

As systems gained more and more services, it became readily apparent that something different needed to be done. As a result, a programmer at Berkeley decided that it might be best to create a daemon that would control all other daemons, a sort of super daemon. Thus, began inetd, or the Internet daemon.

Inetd always runs in the background and it then decides when to start and stop other daemons. So, if a call comes in on port 21 for FTP services, inetd starts the FTP daemon. When a call comes in on port 80 for HTTP services, inetd starts HTTP services, and so on. In this way, inetd conserves resources and improves overall system performance.



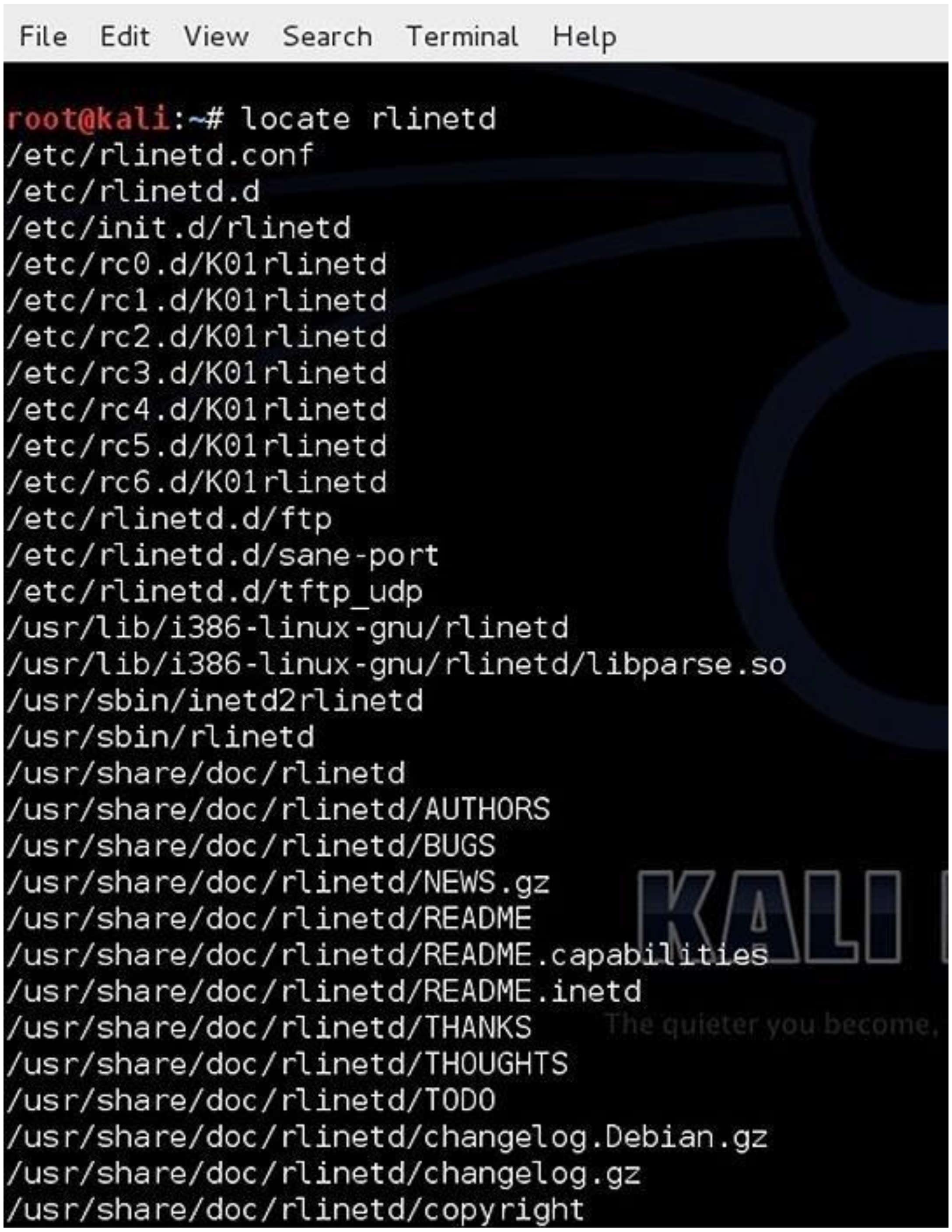
Eventually, this super daemon was exploited by hackers (imagine that) in a number of ways. If you think about it, if I can exploit the super daemon that controls *all* of the other daemons, I can control the entire system. At the very least, if I can control the super daemon, I can probably DoS the system. This is exactly what *did* happen and, as a result, we got a new and improved super daemon called xinetd.

Xinetd was developed to address some of the security vulnerabilities in inetd and was rather rapidly adopted by the commercial Linux distributions, Red Hat and SUSE. Debian and Ubuntu, which are the underlying Linux distributions of Kali and BackTrack, respectively, stayed with the older inetd, initially. But now Debian has transitioned to a newer version of inetd, labelled rlinetd.

Find Rlinetd

We can find rlinetd in our Kali system by typing the following.

kali > locate rlinetd



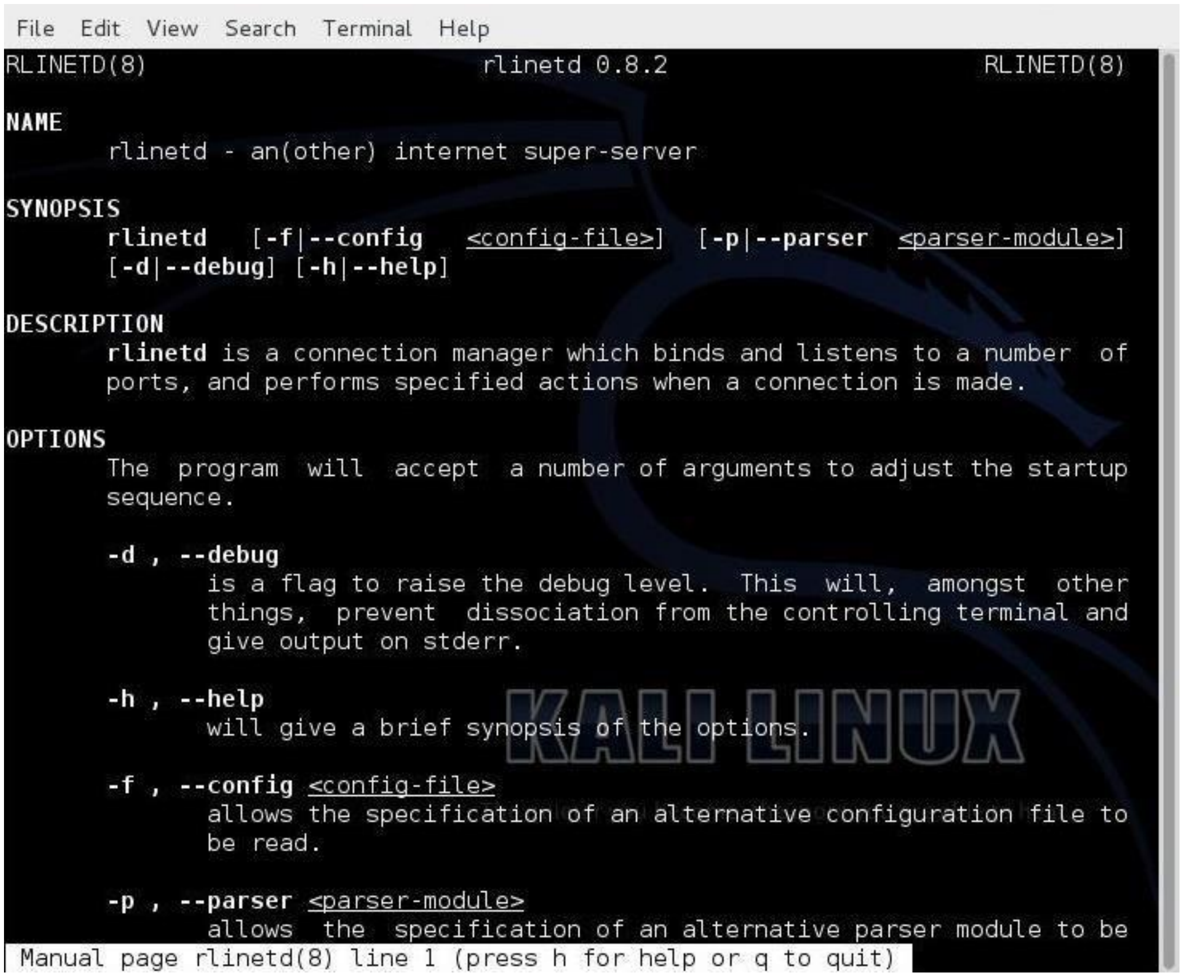
```
root@kali:~# locate rlinetd
/etc/rlinetd.conf
/etc/rlinetd.d
/etc/init.d/rlinetd
/etc/rc0.d/K01rlinetd
/etc/rc1.d/K01rlinetd
/etc/rc2.d/K01rlinetd
/etc/rc3.d/K01rlinetd
/etc/rc4.d/K01rlinetd
/etc/rc5.d/K01rlinetd
/etc/rc6.d/K01rlinetd
/etc/rlinetd.d/ftp
/etc/rlinetd.d/sane-port
/etc/rlinetd.d/tftp_udp
/usr/lib/i386-linux-gnu/rlinetd
/usr/lib/i386-linux-gnu/rlinetd/libparse.so
/usr/sbin/inetd2rlinetd
/usr/sbin/rlinetd
/usr/share/doc/rlinetd
/usr/share/doc/rlinetd/AUTHORS
/usr/share/doc/rlinetd/BUGS
/usr/share/doc/rlinetd/NEWS.gz
/usr/share/doc/rlinetd/README
/usr/share/doc/rlinetd/README.capabilities
/usr/share/doc/rlinetd/README.inetd
/usr/share/doc/rlinetd/THANKS
/usr/share/doc/rlinetd/THOUGHTS
/usr/share/doc/rlinetd/TODO
/usr/share/doc/rlinetd/changelog.Debian.gz
/usr/share/doc/rlinetd/changelog.gz
/usr/share/doc/rlinetd/copyright
```

We can see at the top of the list, the configuration file for rlinetd and the daemon file itself.

Rlinetd Manual

As I mentioned in earlier articles, whenever we want to know something about a particular command in Linux, we can, of course, Google it. Alternatively, we can also use the man, or manual, file. We simply type "man" before the command and the system will pull up the manual file for that command. Let's check out the manual for rlinetd.

kali > man rlinetd

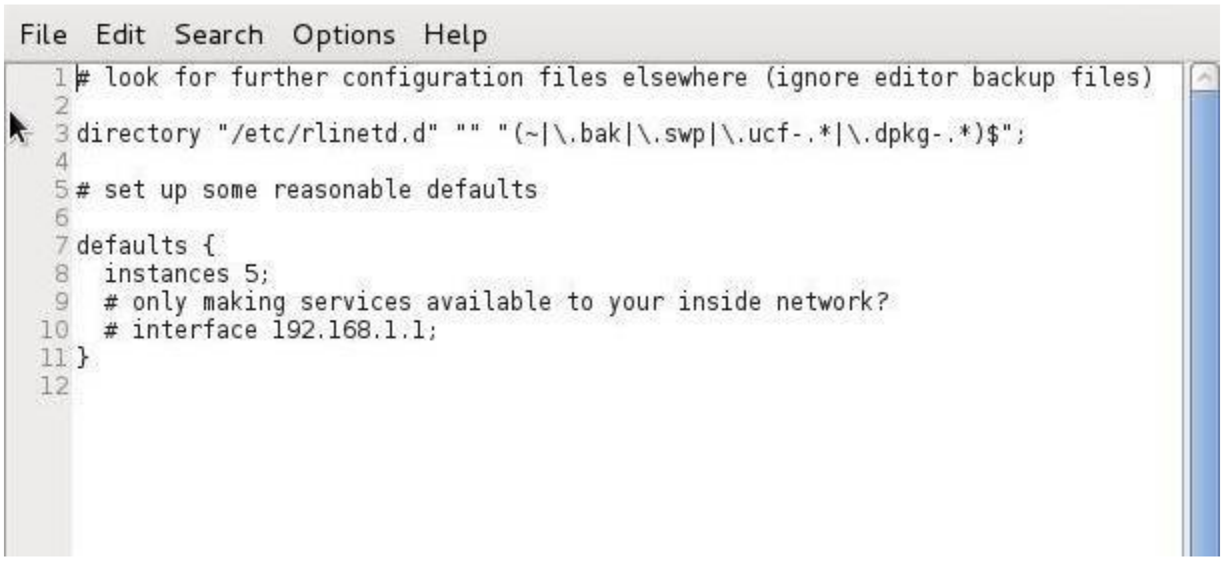


```
RLINETD(8)                      rlinetd 0.8.2                      RLINETD(8)

NAME
       rlinetd - an(other) internet super-server

SYNOPSIS
       rlinetd   [-f|--config   <config-file>]  [-p|--parser  <parser-module>]
       [-d|--debug] [-h|--help]

DESCRIPTION
       rlinetd is a connection manager which binds and listens to a number  of
       ports, and performs specified actions when a connection is made.

OPTIONS
       The  program  will  accept  a number of arguments to adjust the startup
       sequence.

       -d , --debug
              is a flag to raise the debug level.  This  will,  amongst  other
              things,  prevent  dissociation from the controlling terminal and
              give output on stderr.

       -h , --help
              will give a brief synopsis of the options.

       -f , --config <config-file>
              allows the specification of an alternative configuration file to
              be read.

       -p , --parser <parser-module>
              allows  the  specification of an alternative parser module to be
 Manual page rlinetd(8) line 1 (press h for help or q to quit)
```

Take a Look at rlinetd.conf

Finally, let's take a look at the configuration file for rlinetd. Let's open it with Leafpad or any text editor.

kali > leadpad rlinetd



```
# look for further configuration files elsewhere (ignore editor backup files)

directory "/etc/rlinetd.d" "" "(~|\.bak|\.swp|\.ucf-.*|\.dpkg-.*)$";

# set up some reasonable defaults

defaults {
  instances 5;
  # only making services available to your inside network?
  # interface 192.168.1.1;
}

```

We can make our Linux system more secure by setting some default values in the rlinetd.conf file. For instance, if the system were only used for FTP services, it not only would be inefficient to run any other service, but also less secure. For example, if an attacker were trying to exploit HTTP and HTTP was disabled in the rlinetd.conf, they would not have much luck.

We could also change the rlinetd.conf to only start FTP services as needed and nothing else. If you only want this system accessible to a list of IP addresses or just your internal network, you could configure that access in the rlinetd.conf.

As a beginner with Linux, I recommend *not* making any changes to the rlinetd as you are more likely to sabotage and disable your system than making it more secure or efficient, but now you understand what inetd is. With more system admin experience, you can manage this super daemon to make your system safer and more efficient.

Don't Confuse Inetd with Init.d

Linux novices often confuse init.d and inetd. Init.d is an initialization daemon that runs when the system starts up. It determines the runlevel and the daemons that activate at start up. When a computer is turned on, the kernel starts the systems init.d, which always has a Process ID (PID) of 1.

The init process starts a series of scripts that get the system ready for use. These are things such as checking the filesystem and then mounting it and starting any system daemons that are required. These scripts are often referred as rc files because they all begin with the rc.(run command). I'll explain more on init.d in a subsequent tutorial, but I wanted to make certain that this distinction was clear.