Excerpted from:

The Accidental Administrator:
Linux Server Step-by-Step Configuration Guide

First Edition

by Don R. Crawley
Linux+ and CCNA Security

Provided courtesy of

Seattle, Washington

www.soundtraining.net

SBN-10: 1-4536-8992-3
ISBN-13: 978-1-4536-8992-9

Chapter Four:
File and Directory Management

"Linux: the choice of a GNU generation"

--ksh@cis.ufl.edu

Linux File System

Debian Etch, RedHat, SuSE, and other distributions use the ext3 file system by default. ext3 is a journaling file system which offers greater stability and reliability than predecessor file systems. Another popular journaling file system is ReiserFS. V3 of reiserfs is used as the default filesystem for Lindows, FTOSX, Libranet, Xandros and Yoper. Ext4 is also now available and becoming a default file system in new Linux versions.

Linux can also read and/or write to many other file systems including FAT, FAT32, NTFS, HPFS, and others. Partitions are mounted onto existing directories called "mount-points".

Linux uses a tree model to organize directories and files. Directories are the basic unit of storage in the Linux file system. Directories can contain files or other directories. In the same way that a tree cannot exist without its roots, the Linux file system starts at root. Root is designated by "/". (The term "root" is used in three different ways in Linux: "Root" is the name of the superuser, to identify the superuser's home directory [/root], and to indicate the root of the file system [/]. It can be difficult to know which "root" someone is talking about. It helps to be clear about what is meant when referring to "root".)

Linux File Types

When you issue the "**`ls -l`**" command, Linux will display a listing of files along with information about the files. The far left hand column of the listing indicates the type of file. Three common file types are regular files, links, and directories.

Regular files

Regular files are the most common file type on Linux or UNIX systems. They can be used to store various types of data including text that you can read or binary data that can be executed by the system. It is often helpful to identify more information about the file than whether it is a regular file or not. For example, you might want to know whether the file is an ASCII text file or a shell script. You can use the "file" command to identify the file type.

```
$file [filename]
```

Links

Links are files that point to other files on the system. There are two types of links: hard links and symbolic links.

Hard links are a special type of directory entry that have certain limitations:

- Hard links can only point to a file; they cannot point to a directory.
- They cannot be distinguished from the file to which they are pointing.

Hard links are created with the "ln" command:

```
$ln [source] [target]
```

Symbolic links are special files that store a pathname to another file.

Symbolic links are created with the "ln" command, combined with the "-s" option:

```
$ln -s [source pathname] [target]
```

You can think of symbolic links as being similar to shortcuts in Microsoft Windows.

Directories

Directories are containers that hold various types of files or other directories. Directories are used for organizing the file system.

Mounting a Device

In order to make a device such as a CD-ROM or floppy drive available to the file system, it must be "mounted" to an existing mount point within the file system. Before using the "mount" command, ensure that the desired mount point already exists within the file system. A common place to locate mount points is within the /mnt directory (but they can be placed anywhere). To mount a device to the mount-point:

```
#mount /dev/cdrom /mnt/cdrom
```

You can navigate to the newly mounted device with the "cd" command:

```
#cd /mnt/cdrom
```

Before ejecting CDs or floppy disks, you must unmount them from the file system. To unmount a mount-point:

```
#umount /mnt/cdrom
```

Note that, before a mountpoint can be unmounted, you must "CD" out of the directory which you wish to unmount.

Partitions can be mounted automatically on boot through the fstab file, which is located at /etc/fstab.

Understanding Mount Points

You can think of mount points as a way of accessing a partition. Recall that in Linux, everything is oriented around the file system. The first SCSI drive on a computer might be known as /dev/sda, the second as /dev/sdb, and so on. The first IDE drive would be known as /dev/hda. Partitions are numbered, so the first partition on the first SCSI drive would be /dev/sda1, the second partition would be /dev/sda2, etc.

You cannot, however, access partitions through /dev files; you must create mount points which are simply a means of gaining access to a partition through the computer's file system.

A basic partitioning scheme will usually have three partitions: /, /boot, and a swap partition. Often, server administrators will create separate partitions for other purposes as shown below:

Mount Point	Purpose
/boot	Contains boot loader, kernel and related files
/	Root of the file system
/usr	UNIX system resources (usr) is where you find program and related files
/home	Users' home directories and profiles
/var	Variable size files including logs and print spools. Also home to WWW and FTP files.
/tmp	Temporary files

Managing File and Directory Permissions

Linux uses three types of file/directory permissions. For files:

- Read means that you can view a file's contents.
- Write means that you change or delete the file.
- Execute means that you can run the file as a program.

For directories:

- Read means you can list the contents of the directory.
- Write means you can add and remove files in the directory.
- Execute means you can list information about the files in the directory.

Permissions are assigned to both users and groups

- Read permission: Whether the file can be read or the directory contents can be listed
- Write permission: Whether the file can be modified or written to or whether changes can be made to the contents of a directory. For example, without write permission, you cannot create, delete, nor rename a file

- Execute permission: For files, whether the file can be executed. For directories, this is the permission to enter, search through the directory, or execute a program from the directory

You can list file or directory permissions by using the "ls" command with the "-l" option, for example: **ls –l.** On Red Hat-based systems, you can also use the alias "**ll"**. When you list files and folders using the "–l" option, you'll see a display like this:

#**d-rw-rw--- 1 jbach jbach 150 March 10 08:08 file1.txt**

The first column (drw-rw----) is actually ten columns which can be divided into four groups:

- The first group is a single column used to identify the type of entry. The options are:
  - "d" which indicates a directory
  - "l" is a symbolic link to another program or file elsewhere on the system
  - "-" is a regular file
- The second group is three columns used to identify the permissions of the owner
- The third group is three columns used to identify the permissions of the owner group
- The fourth group of three columns identifies the permissions of the world (everyone).

The three permissions columns are, in order: read (r), write (w), and execute (x). If the permissions are expressed as "-rw-rw----", then the entry is a file ("-") whose owner and owner group has read+write permissions, but not execute and the rest of the world is denied access.

Changing permissions

Use the "chmod" command to change permissions. You can set permissions for the user (u), group (g), and others (o). Permissions can also be set for all (a).

Permissions are set using +, -, and =.

+ adds the permission, - removes the permission, and = sets the permission as specified and can be used to copy permissions.

For example:

- #**chmod u+x file1** adds the execute permission for the user owner on file1.
- #**chmod g-w file2** removes the write permission for the group owner on file1.
- #**chmod a+r file3** adds the write permission for everyone on file3.
- #**chmod o=u file4** copies the user permissions for file4 to the world.

Octal (Numeric) permissions

Octal permissions are simply a form of shorthand for assigning access to files and folders.

- Read = 4

- Write = 2
- Execute = 1
- No access = 0

Use **chmod** to assign permissions using the numeric system. For example:

#**chmod 644 file1** would assign owner read+write (6=2+4), the owner's group and everyone would have read permission (4).

Special Permissions

- Sticky bit:
    - Can be used on "world writable directories" to prevent users from deleting other users' files

Assigning Special Permissions

- #**chmod 1766 [directory]** (1 makes it sticky)

Student Exercise 4.1: Managing file and directory permissions

In this exercise, you will use various commands to set and edit file and directory permissions.

Student Exercise 4.1.1: Viewing Permissions

1. Change to the superuser (root) account with the switch user command:

   ```
   $su -
   Password:p@ss5678
   ```

2. Navigate to the "/" directory and use the "ls –l" command to verify the existence of /demo. If it is not present, use the "mkdir" command to create a new directory called "demo":

   ```
   #mkdir demo
   ```

3. Now, use the "cd" command to navigate to the "demo" directory, then use the "ls -l" command to verify the existence of file1, file2, and file3. If they are not present, use the "touch" command to create three files:

   ```
   #cd demo

   #touch file1 file2 file3
   ```

4. Use the following command to view the permissions for the three files you just created:
   #**ls –l**

5. What are the permissions on each of the files for the user?

   _____

   Each of the files should have "rw" permission for the user.

6. What are the permissions on each of the files for the group?

   _____

   Each of the files should have "r" permission for the group.

7. What are the permissions on each of the files for the world?

   _____

   Each of the files should have "r" permission for the world (other).

Student Exercise 4.1.2: Changing Permissions Using Alphabetic Expressions

In this exercise, you will use alphabetic and octal syntax to modify file permissions using the chmod command.

1. While still in /demo, execute the following command to display the permissions for the files:

   #**ls –l file1**

2. Notice that only information about file1 is displayed because you modified the "ls –l" command by appending "file1" to the end of the command. What are the permissions for the user on file1?

   _____

   Again, it should be "rw" for the user on file1.

3. Now, use the following command:

   #**chmod u+x file1**

4. Now, execute the following command to display the permissions for the file:

   #**ls –l file1**

5. What are the permissions now for the user on file1?

   _____

   The permissions should now be "rwx" for the user.

6. Execute the following command:
   #**chmod g+w file2**

7. Now, what are the permissions for the group on file2?

   _____

   The permissions should be "rw-" for file2.

8. Execute the following command:
   #**chmod a+x file***

9. What happened to the permissions on all files in the directory?

   _____

   All files should now have "x" permission in addition to any pre-existing permissions.

10. Execute the following command:
    #**chmod o=u file3**

11. Use the ls –l command to view the new permissions. What happened to the permissions for the world on file3? Are they the same as for the user?

    _____

    The permission for the world (other) should now match the permissions for the user.

Student Exercise 4.1.3: Octal (Numeric) Permissions

In this exercise, you will practice managing permissions using octal settings instead of alphabetic expressions.

1. Execute the following command:
   #**chmod 644 file***

2. Using the **ls -l** command, display the changed permissions. What are the new permissions for the files?

   _____

   The new permissions should be "rw" for the user and "r" for the group and the world (other).

3. Execute the following command:
   #**chmod 777 file1**

4. Again, use the **ls -l** command to display the permissions. What happened?

   _____

   The permissions for file1 are now "rwx" for user, group, and world.

Setting Default Permissions

The **umask** command is the user file-creation mask command which allows you set default permissions.

The **umask** command uses an octal value that is the inverse of the values used with the **chmod** command. In other words, if you wish to set permissions for a directory to full for the owner, read for the group, and nothing for the world, you would use **chmod** as follows:

```
#chmod 740
```

To set the default permissions for *all future files and directories* created to full for the owner, read for the group, and nothing for the world, use the **umask** command with a value that is the inverse of the value used with **chmod:**

```
#umask 037
```

Note that this is a universal command and cannot be applied to a single directory.

Disk Configuration Tools

#**`fdisk /dev/hda`** starts the disk configuration utility "fdisk" ("hda" represents the first IDE drive, "sda" would represent the first SCSI drive on the system.)

Using fdisk returns a different prompt than the customary Linux command prompt:

- Command (m for help):**p** displays your disk partitions.
- Command (m for help):**d** schedules partitions for deletion (if you make a mistake and don't want to delete a partition, you can simply type **q** to quit without saving)
- Command (m for help):**l** lists known partition types
- Command (m for help):**m** lists available commands

Related commands

- **`fdisk -l`** displays information about partitions on a hard drive
- **`fdisk -t`** sets the file system for a partition
- **`mkfs`** will format a partition
- **`fsck`** will repair a corrupted file system
- **`fsck /mbr`** will repair a corrupted Master Boot Record

soundthinking point: partition management tool

The open source tool gparted is a great tool for managing disks and partitions.