

Part 3 Console basics

Having installed Linux using *Fedora Core 3*, *Jarrod Spiga* introduces the command line console and explains how to put it to work.



Skill level

Beginner

Requirements

PC with DVD drive
• Internet connection
• printer (optional).

Time to complete

4 hours

Bonus DVD software

Linux commands
cheatsheet

At this stage of the *Mastering Linux* Workshops, you should have a running Linux system and have experimented a little with the X Windows GUI. But in order to master Linux, you'll need to become familiar with the command line console.

It might sound nostalgic, but back in 1995 Microsoft managed to hide the DOS prompt with the release of Windows 95. Since then, most PC users have used grown accustomed to using their mouse to do everything.

Before this, PCs usually booted to a DOS-based command line. Occasionally, though, the clever tinkerer would modify their `autoexec.bat` file so that Windows 3.x was launched automatically, helping users who weren't familiar with DOS to get the GUI up without any hassles.

BACK TO THE FUTURE

Today's Linux is also a command line operating system, but you can load up a GUI interface if you wish. In the interests of being more user-friendly, most Linux distributions (including Fedora Core) automatically boot to the X Windows GUI if you installed it — otherwise, you'll have to work at the command line console.

But even if your Linux system boots to the X Windows GUI, you should have a good understanding of how to do things from the console. Consider for a moment that something goes awry with your X Windows configuration file, and your system can't initialise the GUI. In this case, there's no other option but to log in to a command line console in order to fix the problem. And it's at times like these that you'll be thankful you have at least some familiarity with the command line.

But the console is useful for many other reasons. It's far more efficient when it comes to performing some tasks from a command line, not to mention that most GUI applications rely on running many of the command-line functions in the background.

Understanding the command line console is essential to working with Linux or Unix. However,

this understanding generally doesn't come overnight. Right from the outset, you should read this guide from beginning to end. Don't worry if you don't understand it all straight away. Some of the concepts explained in this guide will only become clear once you've had more exposure to Linux and the console.

CONSOLE OTHERS

To be blunt, the Linux console is a boring-looking text interface that provides a basic interface to the operating system. Unix has been around since the 1970s, and in those days there were no fancy graphics cards. In fact, you were lucky if you had a monochrome screen capable of displaying more than 40 columns of text and a keyboard with as many as 90 keys — assuming that you didn't access the console via a terminal device.

Because Unix is a true multi-user operating system, it's possible for a number of users to have access to consoles at the same time. This leads to the concept of virtual consoles, allowing a single user to have multiple independent sessions running on the Unix server at the same time. This is akin to having multiple maximised windows displaying on your desktop, in that each one allows you to do different things simultaneously.

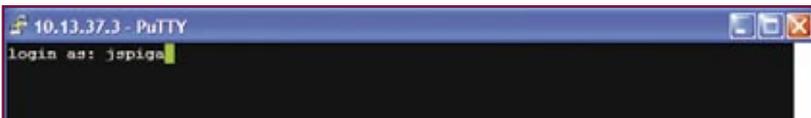
By default, seven virtual consoles are active under Fedora Core 3. The X Windows GUI usually runs on the seventh and last console, while consoles one through to six are normally reserved for text-based command line work. To switch to a different virtual console, hold down the Ctrl and Alt keys with one hand, and strike F1 through to F7 depending on the virtual console you'd like to view. As you'd expect, F1 corresponds to the first console, F2 the second and so forth. Ctrl+Alt+F7 should return you to the X Windows GUI if you're using it.

LOGGING IN AT A CONSOLE

As the name suggests, a virtual console runs in exactly the same manner as a regular console session. Just as you have to log in to your Linux system as soon as it starts up, you'll also have to log in to a virtual console as soon as you switch to it for the first time.

1 When switching to a virtual console for the first time, you'll be greeted with a simple login screen. It usually details which distribution of Linux you're using, the kernel version and architecture of the system, followed by the actual login prompt. Type in your username, then hit Enter. You'll be then asked for your password. Type out this information and hit Enter.

For this exercise, log in as a non-root user, ►



1 Switching to the console brings up a very basic text-based login prompt.

► preferably the user account you set up towards the end of the Linux installation process.

2 If you've logged in successfully, your shell will load, a message of the day may be displayed and a command prompt will appear.

GO WELL, GO SHELL

The shell is a Linux program that accepts instructions from the user and translates those instructions so the kernel can understand them. The kernel itself is a very complicated program designed to communicate with the electronic components in your system. The shell is designed to take your input and convert it to a set of instructions that the kernel can understand. In other words, it's used as an intermediary in order to make the command line easier to use.

There are a number of shells available for use with the Linux kernel. Like most other distributions of Linux, Fedora Core 3 uses the Bourne-Again Shell (bash) by default. This shell was originally modelled on the Bourne Shell (sh) found on many Unix systems. If you use other Linux and Unix systems, you may come across other shells, including the Korn Shell (ksh).

WORKING THE SHELL

Entering commands at the shell can be daunting for those new to Linux, but by remembering a few general conventions most users can get the hang of things a little more easily.

The structure of all Linux commands is made up from three basic elements: the actual command; the option or flag; and the argument. The actual command is the compulsory element and instructs the shell about what type of specific task needs to be done.

The option or flag is usually an optional component that modifies the behaviour of the actual command, and is generally preceded with one or two hyphens (-). The meaning of each option and flag is usually unique to each command. However, one exception is the help flag, which is always `-?` or `--help`.

Arguments are other components that provide additional data to a command in order to execute properly, or to an option to modify the commands behaviour in a proper manner.

The documentation for commands often uses the following simple convention when detailing how to use a command:

```

jspiga@fedora:~
login as: jspiga
jspiga@10.13.37.3's password:
Last login: Thu Dec 2 09:00:43 2004
[jspiga@fedora ~]$

```

2 After you've logged on to the console, a short message of the day will appear. In this case, the time that jspiga last logged on.

```

command -option1 argument1 [-option2 argument2]

```

The actual command is always written in regular typeface, while options and arguments are always italicised. Options or arguments that may or may not be included are normally surrounded by brackets. Basically, the above line states that at least one option and one argument must be used after the command in question, while further options and arguments may or may not follow.

THE LINUX FILE SYSTEM

Working at the command line requires some knowledge of the Linux file system because there aren't any fancy Windows Explorer-style directory trees or search functions to help you find stuff. However, as mentioned in part two of the *Mastering Linux* series, the Linux file system is significantly different to the file systems used under Windows.

For starters, Windows has a root directory for every drive on your system, with a directory tree under that. More recently, Microsoft adopted the idea that My Computer was the root element and that all drives should branch off from this. But for the most part, each drive is still seen as its own file system.

Under Linux, every directory on the system is connected to the root (/) directory. Other drives are mounted to a location somewhere underneath the root file system. Even other input and output devices in your PC are accessible from under the root file system. Much like Windows, the Linux file system also consists of files and directories (directories are often called folders from Windows 95 onward), and directories are simply a storage location for other files.

GO HOME

Another aspect of the Linux file system you should be aware of is the concept of home directories. Home directories are similar to the My Documents folder under Windows; they are a place reserved for the files and directories that belong to each user.

A home directory is usually created for each user on your Linux installation, and stored under the `/home/` directory. For example, a user named craig will have a home directory set up in `/home/craig`. Similarly, `/home/patrick` will be the home directory set up for a user named patrick.

However, there are some exceptions to this rule. Accounts that are generally set up for security restrictions don't usually have home directories. One example is the account normally set up for use by the Apache Web server: `apache`. Also, the root user has a different location for their home directory — it's usually located at `/root/`. This location is called the root users home directory and should not be confused with the root directory of the file system.

The home directory allows Linux to keep the files and settings that belong to each user separate. It also provides a security mechanism that can prevent others from seeing your files. Home directories help prevent a system from becoming cluttered. If every user stores their own files in their own repository, finding them is a much simpler task.

FILE SYSTEM NAVIGATION

Once you've logged in to the command line console, your commands will be executed at a location in the file system called the working directory. As soon as you log in, your home directory will be set as your current working directory by default. The command prompt may give you a little hint as to where you are in the file system, but to find out where exactly you are (to Print your Working Directory), use the `pwd` command:

```
pwd
```

3 This command displays the full path to your current working directory (`/home/jspiga` as shown in figure three). With that in mind, you now know that any command executed will run from this directory. To demonstrate this, run the `ls` command.

```
ls
```

► By itself, `ls` will generate a colour-coded listing of all of the files and directories in the current working directory (similar to the `dir` command under DOS). As shown in the screenshot, the home directory contains a number of executable files (coloured green), three directories (blue), a couple of compressed archives (red) and a symbolic link (a more technical name for a shortcut under Windows, and coloured light blue).

It's possible that your home directory may be empty, particularly if you haven't been using Linux that much since you installed it. You can obtain a listing of the files and folders located in another directory by passing the location of that directory to the `ls` command as an argument. For example, to obtain a listing of all the files in the root (`/`) directory, enter the command:

```
ls /
```

4 In this case, the file system root only contains directories.

In order to change your working directory to a different location, use the `cd` command. In most cases, you'll pass an

argument to this command for the path that you want to change your working directory to. For example, to change the working directory to `/usr`, enter the following command:

```
cd /usr
pwd
ls
```

5 The output of the `pwd` command should show that your working directory is now `/usr`. The `ls` command should also display the contents of the `/usr` directory.

Note that a forward slash was used before `usr`. This indicates that you want to change to the `usr` directory that is below the root directory. If this forward slash was omitted, the shell would expect to change the working directory to the `usr` directory relative to your current location. For example:

```
cd etc
pwd
```

```
cd /etc
pwd
```

After the first directory change, you've switched to the `etc` directory under the current working directory (`/usr`). By adding the forward slash before `etc`, we changed to the `etc` directory that is under the file system root. If you don't pass an argument on to the `cd` command, you'll be taken back to your home directory.

FILE AND DIRECTORY MANAGEMENT

Once back in your home directory, you can start meddling with a few files. But first you need to create a new directory to run some practical experiments. Use the following command:

```
mkdir test1
```

6 By using a relative path, you should have created a directory in the current working directory. Alternatively, you could use a full path to create a directory anywhere in the file system. To demonstrate this, create a second directory under the first test directory as follows:

```
mkdir /home/<username>/test1/
test2
ls test1
```

where `<username>` refers to the username you are currently logged in as. The `ls` command should show the contents of the first test directory you created, and should contain nothing but the second test directory.

The `touch` command is generally used to update the timestamp on a file, but can also be used to create empty files. It expects at least one argument — the name of the file you want to update the timestamp (or create, if the file doesn't exist):

```
touch example.txt
ls
```

Like the example above, the argument that is passed into the `touch` command may be a full or relative path to the file. Using the full path is one way that you can create files anywhere in the file system regardless of what your current working directory is. Now that you have an example file to ►

```
jspiga@fedora:~$ pwd
/home/jspiga
[jspiga@fedora ~]$ ls
apc_dvd      archive-0410.gz  Desktop      #mkdvdiso.sh#
apc_test.iso archive-0411.gz  mastering_linux  mkdvdiso.sh
[jspiga@fedora ~]$
```

3 The `pwd` and `ls` commands in action, showing the path to, and the contents of the home directory.

```
jspiga@fedora:~$ ls /
bin  dev  home  lib      media  mnt  proc  sbin  srv  tmp  var
boot etc  initrd lost+found  misc  opt  root  selinux  sys  usr
[jspiga@fedora ~]$
```

4 The `ls` command can be used to view the location of any directory (such as the file system root) by passing the location as an argument to the command.

```
jspiga@fedora:~/usr
[jspiga@fedora ~]$ cd /usr
[jspiga@fedora usr]$ pwd
/usr
[jspiga@fedora usr]$ ls
bin  games  kerberos  libexec  sbin  src  X11R6
etc  include  lib      local  share  tmp
[jspiga@fedora usr]$
```

5 The `cd` command can be used to change the working directory to another location on the file system (such as `/usr`).

► play around with, try making a copy of it by using the `cp` command as shown below:

```
cp example.txt firstcopy.txt
ls
```

The directory listing should now indicate the presence of another file called `secondfile.txt`. But what happens if you wanted the new copy of the file to have the same name as the original, but be placed in a different directory? In this case, you'd replace the second argument to the `cp` command with a directory path instead of a file name. For example:

```
cp example.txt test1
ls
ls test1
```

7 Because there is already a directory named `test1` in the current working directory, the shell assumes that you want the copy of the `example.txt` file placed in this directory. The Linux file system won't allow you to have the same name for a file and a directory within the same directory. As with the previous examples, you could use a full path to copy the file anywhere in the file system.

It's just as simple to move a file from one location to another, but using the `mv` command. When `mv` is executed, it's actually copying the original file to the destination, then deleting the original. The `mv` command can also be used to rename files — if the second argument passed to the command is not a directory, the file will be renamed in the process of being moved.

CLEANING UP

It's all well and good to have these files and directories floating about in the home directory, but they don't do anything so get rid of them.

You can only remove a directory once it has been emptied, so the first thing to do is delete all the files you've created by using the `rm` (remove) command:

```
cd
rm example.txt firstcopy.txt
test1/test2/example.txt
```

In this case, three arguments are being passed to the `rm` command, each argument

```
jspiga@fedora:~$ mkdir /home/jspiga/test1/test2
[jspiga@fedora ~]$ ls test1
test2
[jspiga@fedora ~]$
```

6 Use the `mkdir` command to create directories, which will help you organise your files.

```
jspiga@fedora:~$ touch example.txt
[jspiga@fedora ~]$ ls
apc_dvd      archive-0410.gz  Desktop      mastering_linux  mkdvdiso.sh
apc_test.iso archive-0411.gz  example.txt  #mkdvdiso.sh#   test1
[jspiga@fedora ~]$ cp example.txt firstcopy.txt
[jspiga@fedora ~]$ ls
apc_dvd      archive-0411.gz  firstcopy.txt  mkdvdiso.sh
apc_test.iso Desktop          mastering_linux  test1
archive-0410.gz example.txt      #mkdvdiso.sh#
[jspiga@fedora ~]$ cp example.txt test1/
[jspiga@fedora ~]$ ls
apc_dvd      archive-0411.gz  firstcopy.txt  mkdvdiso.sh
apc_test.iso Desktop          mastering_linux  test1
archive-0410.gz example.txt      #mkdvdiso.sh#
[jspiga@fedora ~]$ ls test1/
example.txt  test2
[jspiga@fedora ~]$
```

7 A demonstration on how the `cp` command works. If the second supplied argument exists and is a directory, the source file will be copied to that directory.

containing the relative path to the files that you wish to remove. The last thing to do is remove the directories using the `rmdir` command:

```
rmdir test1/test2
rmdir test1
```

Technically, most of the files and directories could have been removed in one fell swoop by using a couple of the options available to the `rm` command, but these switches make it very easy to delete large sections of the file system — something a new Linux user wouldn't want to do. If you'd like to read up on what options and arguments can be given to any command, refer to the `man` (manual) pages for each command by entering:

```
man <command>
```

While inside the main page viewer, use the arrow keys to scroll through the document. Hitting `q` will quit from the viewer.

DOTS AND DOUBLE DOTS

So far, you've only used the `ls` command to generate a basic listing of the files in a directory. Much like with the `rm` command, `ls` has a number of options that can be used to alter the behaviour of the command.

In this case, though, the options aren't as damaging.

Two of the most commonly used options for the `ls` command are `-l`, which produces a long and detailed listing of all of the files, and `-a`, which lists all of the files in the directory, whether they are hidden or not (without the `-a` option, only the visible files will be listed). You can test this out on your home directory (even if it's empty):

```
ls -la
```

If your home directory is empty, you'll probably see a couple of entries appearing, starting with a number of files beginning with a period (`.`). These are hidden files, most of which will contain user settings and preferences for different applications on your Linux system.

At least another two directories should also be in the list: one with a single period (`.`) as its name; and another with a double period (`..`). Technically, these aren't subdirectories under your current working directory. The single period directory is actually a reference for the current directory you're in, while the double period entry corresponds to the directory that is the parent of the current working directory.

To demonstrate this, change the working directory to the `/home`

► directory using the method described above and then obtain a few different directory listings:

```
cd /home
ls
ls /home
ls .
ls /
ls ..
```

8 The first three `ls` commands should have generated the same output: the contents of the `/home` directory. The latter two commands would have shown you the contents of the root (`/`) directory, the parent directory of `/home`.

You can use the `.` or `..` directories in your command line arguments. For example, the following would copy the `example.txt` file to the parent directory:

```
cp example.txt ../
```

or to change the working directory to the parent directory, you could use:

```
cd ..
```

For an example of a more complicated use, if you wanted to copy the `example.txt` file from your home directory to whatever directory you're working in at that time, you could use:

```
cp /home/<username>/example.txt .
```

SHELL SHORTCUTS

Now that you know some of the really basic command line functions, there's no reason

why you can't take advantage of some of the shortcuts available at the bash console.

The most frequently used shortcut is the auto-complete function. By hitting the Tab key, the bash shell can attempt to complete a path, command name or file name that you're currently in the process of typing. For example, perform the following:

```
cd /
cd ho[Tab]
```

Once you hit the Tab key, the shell should automatically complete the rest of the directory name, in this case `home`. The auto-complete function will only work when you've entered enough characters for the shell to be sure which directory you want. For example, entering the following would not generate an auto-complete response:

```
cd /b[Tab]
```

This is because there's more than one directory under the file system root that begins with the letter `b` (`bin` and `boot`). However, enter in the second letter of the desired directory, hit Tab, and auto-complete will work.

If you hit Tab and find that your command element has not been automatically completed, try hitting Tab a second time. This should return a list of all of the entries that match the criteria you've entered so far. For example:

```
cd /b[Tab][Tab]
```

will produce the output shown in image six.

The bash shell also keeps a record of the most recent commands you've executed,

allowing you to quickly bring them up again for re-execution. To scroll through the log of recent commands, use the up and down arrow keys. On a similar note, it is possible to view text that has scrolled off of the screen from the console. To do so, hold Shift+PgUp or Shift+PgDn to review your console history.

One last shortcut worth remembering is that the tilde character (`~`) refers to your home directory. For example, the following command will list the contents of your home directory, regardless of the current working directory:

```
ls ~
```

LINKS

If you recall, in figure three (the directory listing of a home directory) there are a number of light blue entries. These aren't actually files, they're links.

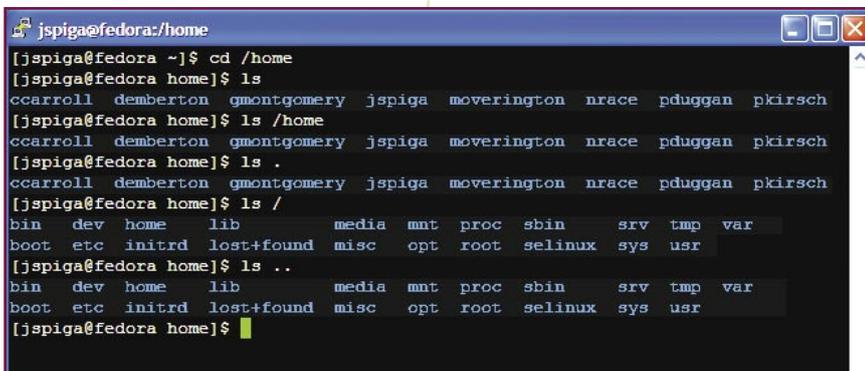
People with a moderate amount of programming experience can think of links as pointers, while Windows users may prefer to think of them as shortcuts. In either case, a link points to another link, file or directory. They are often used in order to conserve disk space (why have two copies of the same file when you can have a link pointing to the original), or to make things more convenient.

If you perform a task on a symbolic link, the task will be applied on the original file in most cases. For example, if you enter in a command to edit the symbolic link, you'll actually be editing the file that the symbolic link points to. Conversely, if you delete a symbolic link by using the `rm` command, you'll be removing the link only, not deleting the actual file.

Symbolic links can be created by using the `ln` (link) command. For example, execute the following commands:

```
cd
mkdir test_dir
touch test_dir/file.txt
ln -s test_dir link
ls
ls test_dir
ls link
```

9 The first command simply switched to the home directory. From here, a test directory was created, as well as a text file in this directory. Then a symbolic



```
jspiga@fedora:~/home
[jspiga@fedora ~]$ cd /home
[jspiga@fedora home]$ ls
ccarroll demberton gmontgomery jspiga moverington nrace pduggan pkirsch
[jspiga@fedora home]$ ls /home
ccarroll demberton gmontgomery jspiga moverington nrace pduggan pkirsch
[jspiga@fedora home]$ ls .
ccarroll demberton gmontgomery jspiga moverington nrace pduggan pkirsch
[jspiga@fedora home]$ ls /
bin dev home lib media mnt proc sbin srv tmp var
boot etc initrd lost+found misc opt root selinux sys usr
[jspiga@fedora home]$ ls ..
bin dev home lib media mnt proc sbin srv tmp var
boot etc initrd lost+found misc opt root selinux sys usr
[jspiga@fedora home]$
```

8 You don't always have to know the full path to a location on your system. Relative paths (using `.` and `..`) can also be passed to most commands.

link (note the `-s` option, which is used to create the symbolic link) was made. The first argument to this command is the location of the file or directory that you're linking to. The second argument is the name of the link.

The last three commands show you how the file system looks as a result of the changes. The first command should show `test_dir` in dark blue (indicating that it's a directory) and `link` in light blue (indicating that it's a symbolic link).

The second last line shows the contents to the test directory you created — the `file.txt` file should be the only thing here.

The last line shows the contents of `link`, which points to the test directory. As you would expect, the contents of `link` should be the same as your test directory because they are actually the same location.

I PWNEED JOOR PERMS!

For any operating system to be considered secure, provisions need to be made at many levels throughout the OS, including the file system. Linux has simple yet effective security settings that are applied on every object in the file system in order to restrict access appropriately. Every object in the file system has two unique attributes:

- It must belong to a user on the Linux system. The permissions system keeps track of which users or groups own every file or directory.
- It has a defined set of access restrictions for the user who owns the file, the user group that has ownership of the file, and everyone else. These restrictions are generally known as the permissions of the object.

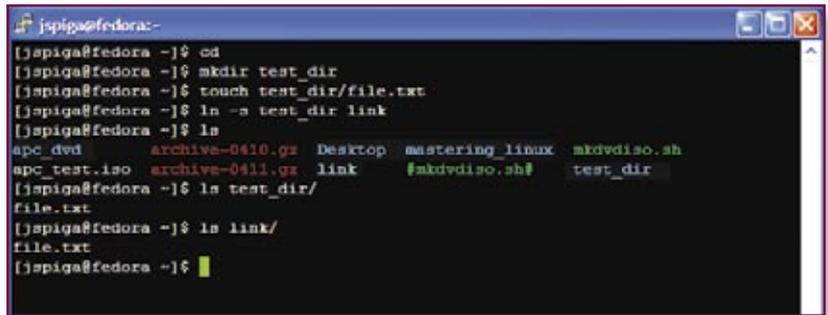
When you create a file or directory, default ownership and permissions settings are applied to it. These attributes can be changed if you have the right level of access. But before doing this, you should become familiar with checking the attributes of a file.

Earlier in this guide, you would have used the `-l` and `-a` options to the `ls` command to generate a list of all the files in your home directory. These options can also be used to generate a listing of all files and directories, showing the ownership and permissions details for each entry:

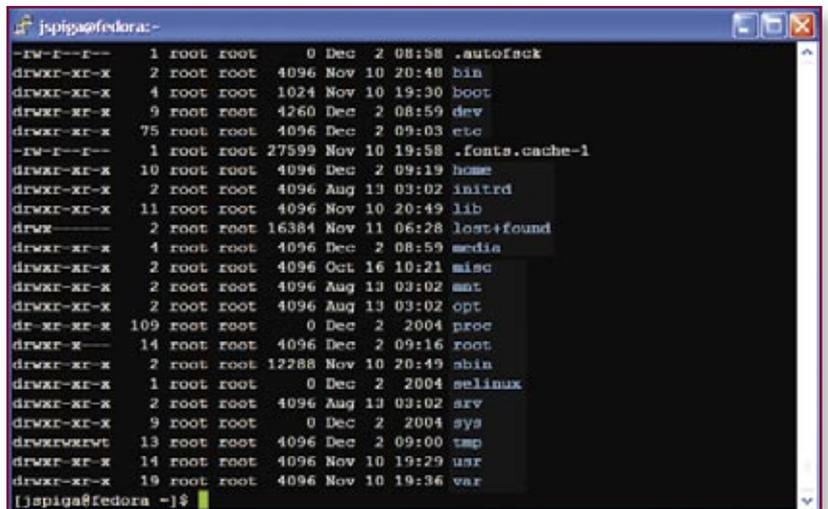
```
ls -la /
```

10 This listing shows a great deal of information about all of the files and directories in the file system root. The column on the far left shows the permissions of the object.

Moving across, the second column shows the number of hard links to that object (a hard link is similar to a symbolic link, see www.apcmag.com/info/hard_link for a more thorough description).



9 Symbolic links can be used like short cuts — they can be used to point to another location on the file system for quick or convenient access.



10 A long (`-l`) directory listing of all (`-a`) files in the root directory. Notice the file/directory permissions, owners and timestamp.

This is followed by the user who owns the object, the group that owns the object, the size of the object (in bytes), the object's creation date and the name of the object.

ABOUT USER ACCOUNTS AND GROUPS

You're probably already familiar with the idea of user accounts, after all, you're using one from the moment you log in to your Linux PC. However, Linux also uses a number of groups; collections of user accounts designed to ease administration while maintaining security. A user account must belong to at least one group, and may belong to several.

Every object in the Linux file system is owned by a group in the same way that they're owned by a single user. From the above example, you can see that the root user and the root group holds ownership of all of the objects in the file system root (this is the case by default). However, it's a different story in your home directory:

```
ls -la ~
```

What the numbers mean

7	rwX	read, write and execute access
6	rw-	read and write access
5	r-x	read and execute access
4	r--	read-only access
3	-wX	write and execute access*
2	-w-	write-only access*
1	--X	execute-only access
0	---	no access

*To many users, this doesn't make much sense. Normally, you need to be able to read a file in order to make modifications to it or to execute it. However, this permission is often granted to directories where you want people to be able to write files, but not give them the access to change or execute them.

► **11&12** As you can see, the majority of objects in this directory are owned by you and your username also appears as the group owner. Don't be confused — this is a group account you are looking at, which is different to your user account. When your user account was created, a group with the same name as your user account was also created and your user account was assigned to this group.

PERMISSIONS

In the column on the left of this directory listing, you'll see a grouping of characters that represent the permissions for the object. The first character always represents the file type. The letter **d** is used to represent directories, while **l** is used to represent symbolic links. If a hyphen (-) appears as the first character, then that object is a regular file. Back in the days of monochrome screens, this was the easiest

way of knowing what type of file you were looking at.

The remaining nine characters can be broken into three sections containing three characters each. The first bunch of letters relates to the permissions that are applied for the owner user. The second grouping dictates the permissions for the owner group, while the last three show the permissions for everyone else.

The first column of each of these three groupings is used to determine read/directory listing access — the presence of an **r** in this column indicates that the user/group/everyone has read access to the file, or has the ability to obtain a listing of the contents of the directory. The second column dictates write access — a **w** in this column allows a user/group/everyone to create or remove files or directories, or modify existing files. The third column indicates execution rights, if an **x** appears in

this column, a user/group/everyone can execute the contents of that file, or change the working directory to that directory.

To illustrate this, run the following command:

```
ls -l /home
```

Observe the permissions on the entry that represents your home directory (you should see **drwxr-xr-x**). This shows that the object is indeed a directory, that you have read, write and execute permissions, and that other users in your group have only read and execute permissions (they don't have the ability to make modifications to your data). Similarly, everyone who is not included in your group also have the ability to read and execute only.

MODIFYING PERMISSIONS

There are two ways that you can change permissions on a file system object: the symbolic method; or the numeric method. The **chmod** (change mode) command supports both methods. Which you use is generally a matter of preference.

THE NUMERIC METHOD

It was mentioned earlier that the permissions shown in a directory listing are actually made up of three sections containing three letters each. Each of these three letters shows the read/write/execute access to that file for the respective user/group/everyone.

These permissions are represented as letters to assist with readability, it's much easier to see the letter **r** and know that read-access is granted. However, you could just as easily substitute the letters for binary digits, where 1 indicates that the permission field is active and 0 indicates that it's not. Using your home directory as an example, its permissions would then become 111101101.

If you then express each grouping of three binary digits as a decimal number, you would get a three digit number that represents the permissions for a file — **755** would then represent the permissions of your home directory.

In order to quickly change the permission on a file system object, pass this number to the **chmod** command as the first argument and specify the object you want to change the permission of as the second argument. For example, to stop your home directory from being accessible

```
jspiga@fedora:~$ ls -l /
-rw-rw-r-- 1 jspiga jspiga 27599 Dec 2 09:16 .fonts.cache-1
drwx----- 4 jspiga jspiga 4096 Dec 2 09:00 .gconf
drwx----- 2 jspiga jspiga 4096 Dec 2 09:21 .gconfd
drwx----- 5 jspiga jspiga 4096 Nov 10 20:01 .gnome
drwx----- 7 jspiga jspiga 4096 Nov 10 20:54 .gnome2
drwx----- 2 jspiga jspiga 4096 Nov 10 20:00 .gnome2_private
drwxr-xr-x 2 jspiga jspiga 4096 Nov 10 20:00 .gstreamer-0.8
-rw-r--r-- 1 jspiga jspiga 120 Nov 10 19:59 .gtkrc
-rw-rw-r-- 1 jspiga jspiga 137 Nov 10 20:00 .gtkrc-1.2-gnome2
-rw----- 1 jspiga jspiga 193 Dec 2 09:00 .ICEauthority
lrwxrwxrwx 1 jspiga jspiga 8 Dec 2 09:22 link -> test_dir
drwxrwxr-x 2 jspiga jspiga 4096 Dec 2 09:06 mastering_linux
drwx----- 3 jspiga jspiga 4096 Nov 10 20:00 .metacity
-rwxrwxr-x 1 jspiga jspiga 2255 Nov 10 20:16 #mkdvdiso.sh#
-rwxrwxr-x 1 jspiga jspiga 2255 Nov 10 10:24 mkdvdiso.sh
drwxrwxr-x 3 jspiga jspiga 4096 Nov 10 20:02 .mozilla
drwxr-xr-x 3 jspiga jspiga 4096 Nov 10 20:00 .nautilus
-rw----- 1 jspiga jspiga 466 Nov 10 20:19 .recently-used
drwx----- 2 jspiga jspiga 4096 Nov 10 20:00 .rhn-applet
-rw----- 1 jspiga jspiga 497 Nov 10 20:00 .rhn-applet.conf
drwxrwxr-x 2 jspiga jspiga 4096 Dec 2 09:22 test_dir
drwx----- 4 jspiga jspiga 4096 Nov 10 20:26 .thumbnails
-rw----- 1 jspiga jspiga 134 Dec 2 09:00 .Xauthority
[jspiga@fedora ~]$
```

11 Compare the long listing of the root directory to a long listing of your home directory. You're the owner of everything within your home by default.

```
jspiga@fedora:~$ ls -l /home/
total 64
drwx----- 2 ccarroll ccarroll 4096 Dec 2 09:18 ccarroll
drwx----- 2 demberton demberton 4096 Dec 2 09:18 demberton
drwx----- 2 gmontgomery gmontgomery 4096 Dec 2 09:18 gmontgomery
drwx----- 20 jspiga jspiga 4096 Dec 2 09:22 jspiga
drwx----- 2 moverington moverington 4096 Dec 2 09:17 moverington
drwx----- 2 nrace nrace 4096 Dec 2 09:17 nrace
drwx----- 2 pduggan pduggan 4096 Dec 2 09:18 pduggan
drwx----- 2 pkirsch pkirsch 4096 Dec 2 09:19 pkirsch
[jspiga@fedora ~]$
```

12 The /home directory should contain a sub-directory for every user account on the system with the exception of the root user.

▶ to users who are not in your group, you could use the following:

```
chmod 750 ~
```

THE SYMBOLIC METHOD

a similar process is followed to the symbolic method is used to change the permissions on a file system object. The `chmod` command is also used, and the name of the object you're modifying is still passed as the second argument to the command. However, the structure of the first argument is different.

The symbolic code you need to generate is made up of three parts:

■ **The scope:** using one or more of the letters `u`, `g` or `o` for the owning user, owning group or everyone else (or other) respectively. The scope defines exactly which elements in the permissions are changed

■ **The modifier:** either a `+`, `-` or `=` sign depending on whether you want to add, remove or explicitly define the permissions respectively.

■ **The access code:** using one or a combination of the letters `r`, `w` or `x` for read, write or execute access respectively. The access code determines which permissions are changed.

Repeating the above example, to remove everyone who is not a member of your group from accessing your home directory, you could use:

```
chmod o-rwx ~
```

CHANGING OWNERS

Before moving any further, we should restore the permissions of your home directory to how they were. Execute one of the following commands:

```
chmod 755 ~
```

or

```
chmod o+rx ~
```

There may be times where you want to change the owning user or the owning group of a file. For example, I may want to change the permissions of the file I'm writing this guide in to allow the APC production team to modify it so that they can correct any spelling, punctuation and

```

jospiga@fedora:/home/jospiga/mastering_linux
[root@fedora mastering_linux]# chown pduggan 03_console_basics.doc
[root@fedora mastering_linux]# chgrp production 03_console_basics.doc
[root@fedora mastering_linux]# ls -l
total 796
-rw-rw-r-- 1 jospiga jospiga 234775 Oct 15 12:38 01_preparation.doc
-rw-rw-r-- 1 jospiga jospiga 430371 Oct 15 12:36 02_installation.doc
-rw-rw-r-- 1 pduggan production 114703 Oct 21 04:21 03_console_basics.doc
[root@fedora mastering_linux]#

```

13 In order to manage your files and directories in a secure fashion, you'll need to change the user and group ownership of file system objects from time to time.

```

jospiga@fedora:/home/jospiga/mastering_linux
[jospiga@fedora mastering_linux]$ su
Password:
[root@fedora mastering_linux]# chown jospiga 03_console_basics.doc
[root@fedora mastering_linux]# ls -l
total 796
-rw-rw-r-- 1 jospiga jospiga 234775 Oct 15 12:38 01_preparation.doc
-rw-rw-r-- 1 jospiga jospiga 430371 Oct 15 12:36 02_installation.doc
-rw-rw-r-- 1 jospiga production 114703 Oct 21 04:21 03_console_basics.doc
[root@fedora mastering_linux]#

```

14 To change the ownership of a file that you don't have access to, you'll need to switch to the root account using the `su` command.

grammatical errors.

Two commands can be used in order to achieve this - `chown` (change owner) and `chgrp` (change group). The usage of this command is very similar to `chmod`, and there are no prizes for guessing this. The user or group that you want to be the new owner of the object is used as the first argument while the object name is passed on to the command as the second one.

Using the above example, I could use the following commands to hand over my document to the production team:

```

chown pduggan ~/mastering_linux/
03_console_basics.doc
chgrp production ~/mastering_
linux/03_console_basics.doc

```

13 As you can now see from the output of the directory listing, `pduggan` (the production editor) is now the owning user of the file and the production team are the owning group.

Another way I could have achieved the same result is by using one of the short cuts with the `chown` command. This also allows you to change the owning group of an object with a slight modification to the first argument. Instead of just specifying the user you want as the new owner, you can specify both the new owning user and group separated by a colon, as demonstrated below:

```
chown pduggan:production ~/
mastering_linux/03_console_
basics.doc
```

SWITCHING USERS

Before proceeding, observe the permissions on the file now:

```
ls -l ~
```

Considering that I am neither `pduggan` nor a member of the production group, I can now only read my document. Obviously, I can't allow this to remain the case, otherwise I won't be able to continue to save my document. Perhaps I should have changed the permissions on my file before changing the ownership. This lack of foresight has caused a problem where I have insufficient rights to save my document. Since I no longer have write access to the file, I also don't have the ability to modify its permissions.

However, since I know the details of the root user account on my system, I can use these credentials to change the permissions of the file. (Remember last month's guide? The root user has the ability to do everything.) To easily switch to the root user account, I can use the `su` (switch user) command. If no arguments are specified, the command assumes that you want to switch to the root user account. However, if you want to switch to a different user account, simply supply the user name ▶

▶ as an argument to the command.

14 Once you're successfully authenticated as the root user, you then have access to modify the file's properties again. In my case, I'd simply execute:

```
chown jspiga ~/mastering_linux/  
03_console_basics.doc
```

Now, I am the owner of the file again, but the production team is still set as the group.

As a result, I can save my document and they can still make the necessary changes before this gets rendered as a PDF.

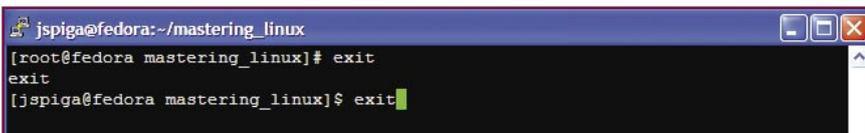
EXIT, STAGE LEFT

Even after executing the command to change the ownership of the file back to my user account, I'm still logged in to the console as the root user. To relinquish these authentication credentials for the time being and get back to my user account, use

the exit command.

```
exit
```

15 The same command can be used to log out of the system. Once you do so, the screen will be cleared and the login prompt will display - ready for the next user to log in to the console. [ETC](#)



```
jspiga@fedora:~/mastering_linux  
[root@fedora mastering_linux]# exit  
exit  
[jspiga@fedora mastering_linux]$ exit
```

15 To exit the *su* shell or to log out from the console, use the *exit* command.

Next month . . .

Next month's *Mastering Linux* guide will focus on getting familiar with the X Windows desktop. It will also reveal a couple of secrets that will help you use the Linux GUI more efficiently.