Getting to grips with Linux Permissions

DO IT WITH PERMISSION

With any operating system it is important to ensure that users remain in control of their files and directories and are prevented from tampering with those belonging to other users, or the system. This is what the Linux permissions system is all about, as Jono Bacon explains.

Permissions are at the heart of how Linux works. Some operating systems (such as MS-DOS and some variants of Microsoft Windows) treat all files in the same way. This means that any user can change any file. Usually, there is only one kind of user.

Linux does things in a completely different way. Under Linux many different users may be using the system at once (that's why you need to log in when your system has booted.) Each user has their own slice of the system that they can use to store their files (the path */home/username*). When you have more than one user using the system you need to ensure that other users cannot modify your files if you don't want them to, and that users cannot look into files that you wish to be private. Examples of this are the system configuration files in directory /etc which cannot be viewed by users other than root. (This is because those files may contain details that a malicious user could use to attack the system.)

You may be wondering how this is useful if you are the only person using your system. You may even be thinking that this is just a time-consuming irritation. Many new users think this at first. But you will come to realise that permissions offer many advantages. An example of this is on my own system. I run the latest developer version of KDE, but sometimes I need to use KDE1.x and GNOME. So that I don't have to constantly move files around to revert to these systems I simply have a different login for each, and a general directory where all users can share files. This saves a lot of time over changing the settings the manual way.

Linux permissions are basically split into two areas. These are:

- File ownership
- File access permissions

Every file has an owner. This is usually the user who created the file, although this can be changed. Users can also be classed into groups, so similar users can be grouped together.

The other element is the access permissions for the file. These are split into three areas:

- Who can read (view) the file
- Who can write to the file
- Who can run the file (this only applies to files that *can* be run)

Each file has three sets of permissions: permissions for the owner of the file, permissions for the group the file is assigned to, and permissions for all other users on the system.

Let's look an example. Open a console and do a directory listing by typing:

```
ls -al
```

at the command line. The result on my system can be seen in Fig 1. This is simply a directory listing, but lets look at one line as an example:

```
-rw-r r    1 jono     jono        1701 Jul 13 ➥
15:23 nickmail.txt
```

A lot of information is given. Reading the information from left to right, this is what it means:

```
-                File type indicator (- mean➥
s normal file)
rw-r r           File permissions
jono             Owner
jono             Group
1701             File size
Jul 12 15:23     File creation time and date
nickmail.txt     Filename
```

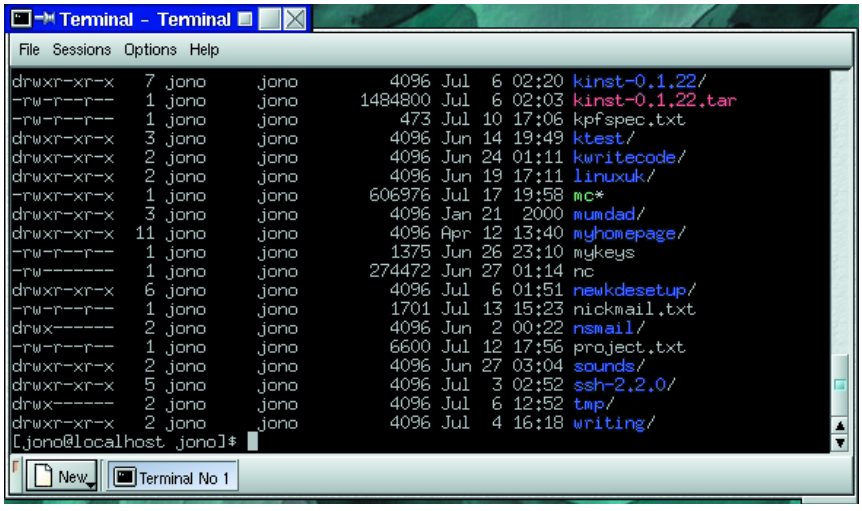
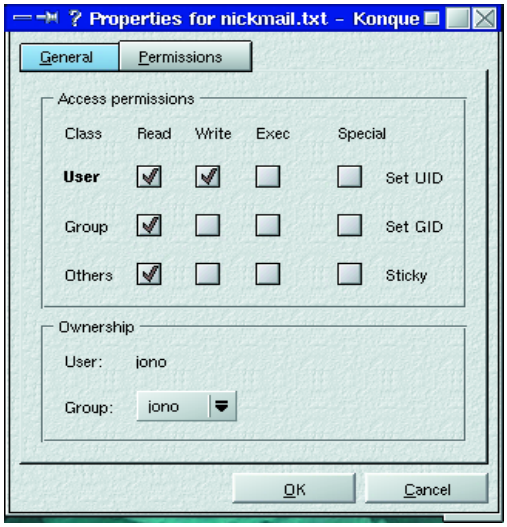
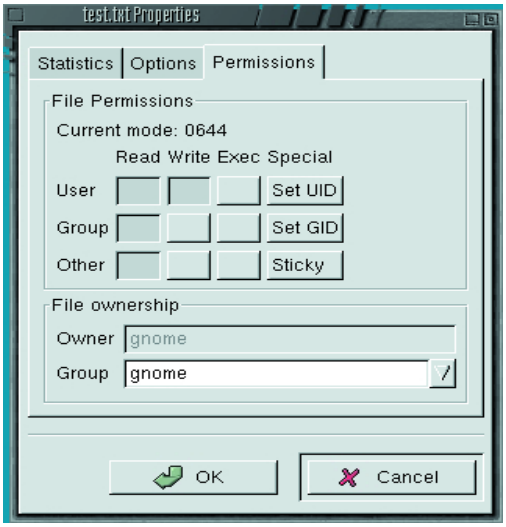


Fig 1: An example directory list





[left]
Fig 2: The KDE file permissions tab

[right]
Fig 3: The GNOME file permissions tab

The parts we are interested in primarily are the first four elements; file type indicator, file permissions, owner and group.

The file type indicator tells us what type of file it is. Everything in Linux is treated as a file, even things such as devices (just do an **ls -al** in /dev to see what I mean). With everything being treated as a file, it is handy to have something to say whether it is a normal file, a directory, a device or whatever. Below are some examples of different uses:

```
drwxr-xr-x   4 jono     jono        1024 Jul 27 00:36 .kde
```

This line shows that .kde is a directory as the file type indicator is set to 'd'.

```
brw——-   1 jono     floppy     2,   0 May  5  1998 fd0
```

This line shows that fd0 (the floppy drive) is a block device by the 'b' in the file type indicator.
Now look at the permissions part that is next to the file type indicator for nickmail.txt. This reads as:**rw-r–r–**. The permissions part is split into three sections, each having three characters. Each section reflects a type of user on the system, and whether they can read (r), write (w) or execute (x) the file. On this file each section breaks down as:

rw- The owners section. The owner can (r)ead and (w)rite to the file. The owner (see below) cannot execute the file, but this can be changed as the owner can (w)rite to the file.

r— The group section. The group (see below) can just (r)ead the file but cannot write to it.

r— The 'all other users on the system' section. All other users who are not in the same group and do not own the file cannot write or execute it.

To find out who the owner of the the group the file belongs to, we need to look at the next two pieces of information in the file's listing. For nickmail.txt we can see that the owner is 'jono' and the group is 'jono'. Now we know what the different bits of a file listing mean and how we stop certain users using or accessing files. How do we change the permissions?

Changing a files permissions in KDE

To change permissions using KDE is very simple.

1. Start the KDE file manager (kfm for KDE1.x and Konqueror for KDE2.x).
2. Right-click the file whose permissions you would like to change.
3. Select Properties from the menu. Click on the Permissions tab (Fig 2).
4. To change the file's permissions, click the relevant check boxes to select the Owner, Group and Other permissions that you need.

Bear in mind that to change permissions on a file you need to have permission! You must have write access to the file. You will notice that there are three additional boxes you can select; Set UID, Set GID and Sticky. These are explained in the box »Special Permissions.«

Changing a file's permissions in GNOME

Changing permissions in GNOME is virtually the same as in KDE.

1. Start the GNOME file manager (gmc).
2. Right-click the file whose permissions you wish to change. Select Properties from the menu. Click the Permissions tab.
3. A box like Fig. 3 will appear. As you can see it looks very similar to the one in KDE, and functions in exactly the same way.

Changing a file's permission in the command line

In Linux, virtually every operation that you can do in a GUI such as KDE or GNOME can be done using the command line. Changing permissions is no dif-

ferent. To change the permissions on a file we use the **chmod** command, which has the format:

```
chmod <permission(s)> <filename(s)>
```

The chmod command is a very versatile command, and can change the permissions in a number of ways. Probably the easiest way to remember is by changing the permissions using the same letters to set them as they are displayed (r, w and x). To do this you must first specify the section that you want to change (owner (u), group (g) or other (o)). You must then specify + or - to indicate whether you are giving (+) permissions or removing (-) them. Suppose that you would like to change the file nickmail.txt so that all other users on the system can read and write to it. You would use:

```
chmod o+rw nickmail.txt
```

This command basically says »give every other user on the system (o) addition permissions (+) that are read (r) and write (w) on the file nickmail.txt«. Now let's assume we had a change of heart and wanted to let any other user read the file, but not write to it. To take away write access we would use:

```
chmod o-w nickmail.txt
```

Here are some more examples of changing permissions:

chmod u+x file.txt Lets the owner execute the file.

chmod g+wx file.txt Lets the group write and execute the file.

chmod o-r mydir Stops any other user seeing what is in the directory mydir (e.g - when doing an 'ls -al').

As we said, permissions are based around who owns the file and what group the file belongs to naturally need commands to change the owner of the file and their group. Those commands are chown and chgrp. The chown command is very simple. Let's assume we want to make Bob the new owner of the file nickmail:

```
chown bob nickmail.txt
```

It's as simple as that. When you list the file now with a 'ls -al' you'll see that the owner section of the information has changed to Bob. Luckily, the chgrp command is just as easy to use as the chown command. Suppose we want to change the group of nickmail.txt to bob as well. We would type:

```
chgrp bob nickmail.txt
```

Again, if you look at the file by doing an 'ls -al' you will see that the group section of the information has changed to Bob. However, this can only be done by root.

Are permissions useful?

After all this, you may be wondering what all of this has got to do with anything. Well… quite a bit actually. The first thing that it is useful for solving problems. Many problems that you can encounter on Linux are simply down to the fact that a particular user does not have permission to do something. A common example of this is when mounting disks. Traditionally, only root can mount a disk (such as a CDROM or floppy) but there are many cases when a normal user needs to do so as well.

Another use of permissions is to make shell scripts executable. This is done by setting the 'x' permissions bit. To demonstrate this create a plain text file (call it diskfree for the sake of example) using your favourite text editor containing the following text:

```
echo
echo »Hello...I am now going to list your hard disk space:«
echo
df
echo
echo »There we go...all done. :-) »
echo
```

Once you have created it, set the execute bit by either changing it in as described above, or by typing:

```
chmod a+x diskfree
```

You can now run the file by typing:

```
./diskfree
```

As you can see some text is printed and your disk space is shown.

When you created the file it was simply text, although the text contained commands understood by your command shell. By setting the executable bit the file can be run so that the commands are executed. This is called a Shell Script. Although the example was pretty trivial, shell scripts can be used to do some amazing things.

Conclusion

Permissions are not just an important part of Linux, but an essential part. A good understanding of how permissions work and how to deal with problems with permissions is important if your system is to work well. Like many things in Linux, we can always delve deeper into the subject. For further information on working with permissions look at the following resources:

- the chmod, chown, chgrp man pages
- the Linux Documentation Project manuals and documents http://www.linuxdoc.org/
- IRC Chatrooms (#linuxuk on irc.openprojects.net or irc.linux.com, #linux on efnet.demon.co.uk)

If I can offer one final piece of advice it is: »Assume nothing«. Don't assume that your system is going to work the ways you think it will, and don't assume your security is tight enough for a networked environment. Permissions are there to protect you both as a user and a system administrator, and if you are the only user of one Linux computer you get to wear both hats! ■

Special Permissions

Linux has some special permission settings. You won't often need to use them, but this is what they are:

Set UID

This setting causes the process that is executed by the file to run with as if the file owner was running it. This can be useful in cases where you need root access to do something (such as using a device). Be careful when using this setting as it could infringe some security on your system if used incorrectly.

Set GID

This is similar in many ways to Set UID except that the process will execute with the same group ID as the owner.

Sticky

This unusual bit will save the image of the program into the system's swap memory for increased performance.

*Check the chmod man page (run **man chmod**) for more details on using these special bits if you ever need to.*

■