Linux Principles

by Andy Pepperdine

This paper is intended to provide the reader with an understanding of the principles on which a Linux system operates and can be maintained. There is so much in the system as a whole, that only a small portion of the whole can be covered, but it is hoped that enough is conveyed to give an idea of how to find out more when it is needed.

Examples will be taken from Ubuntu 10.04, Lucid Lynx, implementation.

To see what is happening on a live system, I will use command line programs. To follow along, start up a command line interface through Applications → Accessories → Terminal so you can enter the example commands on your system.

What is Linux?

Strictly speaking, *Linux* refers only to a small but essential core of the systems that people have downloaded and refer to as Linux. A better name for the whole system is *GNU/Linux*, where Linux refers to the kernel on which all programs run, and GNU the necessary tools and libraries that support programs running on the kernel.

So far as this paper is concerned, I shall use the name *Linux* as that is how the vast majority of its readers will refer to it when they acquire a system and install it.

The file system

Although the term *folder* has become entrenched among the general public for a collection of files, I shall here continue to use the term *directory* to refer to a collection of files and subdirectories, as that is the technical term used in almost all documentation and will be used in answer to questions on help websites.

Like all Unix systems and their derivatives, every file wherever it resides, can be found by a *path* from the root directory known as '`/`' (pronounced 'root' or 'slash') through a series of other subdirectories. The path consists of a sequence of names of subdirectories separated by slashes ('`/`'), ending with the name of a file. File names and directory names can contain any character whatsoever with the exception of the slash character. In practice do not use characters you cannot display and type or manipulating the file could be difficult.

A name preceding a slash is known as a *parent*, and one following is called a *child*.

Each directory contains, by convention, two special subdirectory entries. The first is known as dot (literally a dot, or full-stop, '`.`') which refers to the directory itself. The second is dot-dot ('`..`') which refers to the parent of the directory.

By convention, names that begin with a dot are not listed by default. Many programs use this fact to hide the directories and files they need to keep track of what a user needs from time to time. This is where they store user settings and other information connected with the operation of the program.

There is no intrinsic difference between the names of files and directories, and I shall try to use the term *pathname* to refer to either.

To see where you are, try printing the working directory using the `pwd` command:

```
andy@desk:~$ pwd
/home/andy
```

The initial part of the line is known as the *prompt*. Here it is the string: `andy@desk:~$`

The command name you will enter is the string: `pwd`. And the response is displayed on the following line. In this case, my home directory (see next section).

To list the names of files in a directory, use the command `ls`. Using with no parameters, will list the names in the current working directory, and will list them in columns in alphabetical order. It will ignore all files whose names begin with a dot.

To list all files in a directory, including those beginning with a dot, use the `-a` option:

```
andy@desk:~$ ls -a
```

The result will be a much longer list sorted in alphabetical order, ignoring the initial dot where present.

It has been said that the test of a true geek is to ask about all the parameters that this unassuming little command can take. They are legion but you will need no more than a very few. However, they can be invaluable to find out what is really going on with files.

Accounts

To access a Unix-type system, a user must supply a valid name of an account and a password. In practice, there are variations to this to be described later.

All systems contain one account with the conventional name *root*, which allows access to all parts of the system with no exceptions. This privileged account is also referred to as an *administrator*, or system account. I will use the term *administrator* to refer to the person who has access to the root account and is setting up the system to dictate who can do what and what is installed. In a typical home installation, there is a single user account who can switch to act as administrator.

All other accounts are user accounts with limited abilities. How much each user is allowed to do is determined by the administrator. Root always exists and always has full access.

I am not here concerned with secure Linux which has further restrictions on who can do what, we will consider only the Linux normally downloaded for a home desktop.

Each user account is provided with an area for their own files and is known as the home directory. Conventionally all the home directories are placed under the general directory `/home`, and `/home` should not be used for anything else.

Some accounts are used by the system for special purposes and you usually cannot log in to these accounts. You can log in to normal user accounts, and this case, there is a directory known as the *home directory* and unless changed will be found at `/home/username` for account `username`.

Note that going to System → Administration → Users and Groups does not list all accounts. To obtain a complete list of all accounts, you will need some command line magic. Run this:

```
awk -F':' '{print $1}' < /etc/passwd
```

The list will include all system accounts for special purposes, like management of the printers, etc. You will rarely, if ever, need to know them all.

Special accounts

You will see from the list of all the accounts that there are several you will never have heard of. These accounts are there for the benefit of the system, and it is not expected that you will ever need or wish to log in to them. In fact, they have been set up without passwords, not even an empty password, and that makes it impossible to log in to these accounts. They exist purely to act as owners for certain files. It is however, possible to run programs as though they were being executed by one of those accounts. Or to temporarily operate as though you were logged in, but I've never needed to do so.

Root

The account known as root gives access to all places and functions. There are several ways in which a Unix system could be set up to allow control of how root is accessed and used.

All Unix systems support a command `su` (for switch user). This command allows a user to switch to operating under the guise of another user, but will challenge for the login password before switching. It cannot be used to switch to an account with no login password.

Ubuntu sets up the root account with no login password. Instead, it uses another more recently introduced method that allows a single command to be executed as though by root. It does not login as root, but merely allows one command only. This command is `sudo`, or the graphical form `gksudo`. It has a control file describing who can do what as the administrator in `/etc/sudoers`. This file of course is under the ownership of the root account.

To see the effect of sudo, try this command:

```
andy@desk:~$ cat /etc/sudoers
/etc/sudoers: Permission denied
```

As you will see, the permissions will prevent you from looking into this file. But if you do it under the aegis of the `sudo` command:

```
andy@desk:~$ sudo cat /etc/sudoers
[sudo] password for andy:
# /etc/sudoers
#
```

It will list the whole file, but only after you have been challenged for your own login password. The root account has no login password, so it can ask only for you own.

Groups

Accounts can be put together in *groups*. Each group will have a set of abilities defined by the administrator, and membership of a group will convey those abilities. The control of a user's actions is usually performed by allowing access to certain paths within the file system. Each account is in one *primary* group, and some or none other groups.

To see the list of groups, you can go to the User and Groups application and by hitting the Manage Groups button you will see them all listed. You can also see what groups you are a member of by using the `groups` command. The list available under the Advanced Settings of the Users and Groups dialog does not say how the groups are named, but does indicate what functions they have.

Most of the time, the normal user does not need to know much about groups, but there is one special usage that you should be aware of. As described in the last section, certain users can access root permissions. The `sudo` command allows, in Ubuntu by default, all users who are in the group named `admin` to be able to use root permissions on presentation of their own login password. There is a timeout on this usage (15 minutes by default), so commands executed in rapid succession will challenge only the first time for the password.

File permissions

Each path has an *owner* which is an account name. By default, when a path is created it is owned by the account that created it. Each path also has an associated group. By default it is the primary group of the account that created it.

There are three basic permissions a path can have. The first is whether it can be read, the second whether it can be written to, and the third whether it can be executed. These three are independent of one another.

These three permissions apply independently to three types of account. The first is the owner of the path, the second refers to accounts that are in the associated group, and the third to all accounts.

A note about the execute permission: for ordinary files, this refers to whether it is possible to execute it as a program or application or script, etc. For directories, the execute permission refers to the ability to traverse the directory into its subdirectories. In this way, a directory can be marked to prevent anyone looking further into its contents, for example, to prevent a guest account discovering security keys.

To examine the permissions that a file has, use another of the options on the `ls` command, and this time, we'll put the name of a file in to get details of just that file.

```
andy@desk:~$ ls -l /bin/true
-rwxr-xr-x 1 root root 30204 2010-03-05 03:29 /bin/true
```

The option used is the small letter `l` (for long).

The output shows the permissions on the left in the first ten characters. The third and fourth entries in the output show the user name of the owner, and the group; in this case, the user name is `root`, and so is the group name.

The very first character on the line indicates the type of file, where a minus sign means a regular file. The next three characters apply to respectively to the read, write and execute permissions given to the owner of the file. So in this case, `rwx` means that the owner can read from it, write to it, and is allowed to execute it.

The next three apply to all the members of the group. The final three of the first field in the line apply to everyone else. In this case, they can read the file and execute it, but cannot write to it, since the position for the write permission is a minus sign.

Using the `ls` command with the name of a directory will by default list all the contents of the directory. To see the permissions of the directory itself, use the `-d` option:

```
andy@desk:~$ ls -ld /home/andy
drwxr-xr-x 78 andy andy 4096 2010-08-10 08:38 /home/andy
```

Applications or programs

Traditionally, all executable programs under Unix-like systems **have no extension** to their names.

They are marked by the permissions they grant to the user. If the file is marked as executable, then it can be executed, whatever the name of the file.

Whenever an application or program is running (or executing), it is doing so in the environment of an account. There is always an account associated with the execution, no exceptions.

Some programs start up by default when an account logs on. This is how the display is initialised and the system can present a user with the graphical interface of their choice. There are a number of supporting programs running in the background to provide those services.

If you want to see how many processes are currently running on your system, run the command:

```
  ps -ef
```

or

```
  ps -ejHf
```

to get a list of them, their user accounts, and the command lines that started them. The second form shows a tree structure of what process started which other processes. These commands may truncate the command line description due to lack of space in the output window. If you want to see the whole line in use, then add `ww` to the options.

By convention, programs use no file extension, but that is pure historical convention. They typically reside in a directory named `bin` (for *binary* as opposed to text) somewhere in the file system, and different types of programs are found at different locations. Other conventions come into play to tell the system which directories to look in for a program. To find where a program is which you can start from the command line, try the command:

```
  which eog
```

and it will show the place it will pull the program `eog` (an image viewer) from.

By convention, programs and applications are stored in directories named `bin` (originally standing for 'binary'). They may be at the root level (`/bin`) or at a lower level (e.g. /usr/bin). Programs that are intended to be executed only by root, are often in directories named `sbin`.

Packages

Modern Linux distributions come as a large set of packages. Each package contains all the necessary files to perform a certain function. The process of initial installation is to load all the packages that have been provided by the supplier as the default set that you will be allowed to use. After installation, you can then tailor what you have by downloading and installing more packages from the servers that the distributor provides. This service is free of charge, but they would all welcome some support in any way that you can. They have to be paid for somehow.

Each package has been wrapped up with instructions as to which other packages are needed for it to work properly. So when you ask for one package, it may also say it will install others it need as well. These others may also need further packages, and so on, until all necessary packages have been identified.

One aspect of Linux has been the uppermost desire by the engineers to ensure that compatibility is maintained between versions. This is especially true at the level of the program interfaces and isolates the development of each package from others. For example, if A needs B to work, then updating B will not affect A, except any improvements that happen in B automatically become available to A through B. However, the same is not quite so true between major releases when significant functional changes may be made.

To see what has been installed, bring up Synaptic (Ubuntu) from the System → Administration menu. When it starts up, it will list along the bottom of the window the number of packages it has installed. It also lists the number it has access to under the number 'listed', which provide a dazzlingly large number of things you could do if you wanted to.

To see details on what is installed, select the Status button in the left pane, and then hit the line Installed. All of the packages listed will now show a green square to their left which indicates they are installed. There is a full list of what those squares mean in the reference.

Further properties of each package can be found by either right clicking on the package, or hitting the Properties icon. In particular, the Installed Files tab show which files have been included and put into the system as part of this package.

Some packages are supplied and supported officially by Ubuntu and these are marked with a red symbol by the package name. The others are included from a third party and will be updated by them.

Each day when you start up your system, a process will begin that will check the list of packages you have against the list at Ubuntu's repositories to see whether any you have installed can be replaced by a more up to date version. If there are any found, you will get a notification in the form of a window for System Update. When you open that window, you can update the out of date packages it has detected.

On very rare occasions, it may happen that the list of updatable packages can itself be out of date and you may get an error when it attempts to locate and download them. In this case, you should cancel the update, and then hit the Check button to refresh the lists and try again.

Updates to packages are viable between versions of a package within the same release number; but if a package increases its release number, then the distribution may delay its introduction until one of its major releases is produced. This is to ensure that all packages that form a co-ordinated set are all synchronised.

More on the directory structure in Linux

At the root level, there are a number of more or less standard named directories with recognised uses. Unix is very flexible in its configuration, and these descriptions are what you will find for almost every Unix-like system in existence. However, for any individual system, there may be some degree of modification to the overall scheme.

They are listed here in alphabetical order within levels of structure. It is not complete as some of the areas are used only when you are developing your own software. Unix was designed by and for software developers and some of this history is still visible.

/bin

Contains programs without which the system would not be a Unix system and would not work. This is where things like the applications that run the command line, and all the simple commands, can be found.

/boot

Contains the essential stuff to help the system pull itself together when first started up.

/dev

Contains descriptions of all the known hardware devices, together with indications of the types of those devices and sometimes how to get the drivers for them. Most of the files in here are very special files with highly idiosyncratic characteristics.

/etc

Contains files describing the configuration of applications and other system information. It also contains various bits and pieces for which there is no other clear place to put them.

/home

Contains all the "home" directories for each account. There is one subdirectory here for each account.

/lib

Contains supporting libraries to allow the programs to execute. Typically, lib contains those files that programs in a bin (or sbin) directory will require at the same level, but they may in turn call libraries at higher levels.

/media

Contains the points in the directory structure that will be used when removable devices are plugged in. This is where you will find USB sticks, USB CD-readers, etc. are placed when they are detected.

/mnt

Contains the positions where extra filesystems that are mounted explicitly, e.g. across a network.

/opt

Contains optional applications and associated files. Linux uses this rarely, but is more often used in other Unix systems (see **/usr**).

/proc

This is a constructed pseudo-file structure, and does not point to any real files, even though it may look as though it does. There is no permanent data held in this directory. It is a continuously updated set of status information about the running system, like battery state on a portable, and information on the running processes.

/sbin

Contains the programs and applications intended for system administration (e.g. shutting down the system, formatting disks) and not for general use.

/sys

This is another pseudo-directory, and contains information about the hardware.

/tmp

Contains any files that an application has created for purely temporary purposes. Most systems are set up to clear this directory when they start up.

/usr

Contains the programs and applications and all associated files that a normal user will access. It contains a subdirectory structure that largely mimics the root directory, `'/'`, and has subdirectories for applications, libraries, and so on. Things here are not essential for the system to run, but without them, a normal user would find there was little they could do.

/usr/bin

Contains all the standard applications that have been installed, and can contain a very large number of files. These applications are those considered more or less essential for normal users to need to run other things (see **/usr/local**).

/usr/lib

Contains the libraries needed by the programs in `/usr/bin`.

/usr/local

Contains the applications that a particular installation has decided to put in place, rather than the "standard" ones that are always present for a system like this. It contains its own subdirectories in a familiar form: **/usr/local/bin**, **/use/local/lib**, **/usr/local/etc**, **/usr/local/share**, and so on.

/usr/share

Contains data that is shared among the applications installed on this system, for example, fonts.

/var

Contains files that are varying in content as the system runs. Most system files will not change after the relevant package was installed. But this directory is where to find printer spool queues, log files, error reports, and other varying data.

Finding what a command does

The traditional way of discovering what a command does is to use the `man` command (for manual). Be warned, however, that these descriptions are terse to the point of bafflement on occasion, but they are still the most complete and accurate documentation for system commands. The command takes the name of the command as parameter and will start listing the documentation until it runs out of space in the window. The simplest keystrokes to control the listing are:

space bar: list the next window full of text

return key (CR): show the next line only

q: quit and return to the prompt

/<pattern> CR: search for <pattern> and stop listing when found

Try typing:

```
andy@desk:~$ man man
```

to see all the features.

For graphical interfaces, the dialog sometimes has a Help button to provide information relevant to the context.

Today, though, in practice, Internet search engines can lead you to more expansive and sometimes more readable descriptions of commands.

Finding a command to do something

This is much harder, and command taxonomy is neither a refined art nor a science. Often the best bet is to ask someone, or do an Internet search with a few relevant keywords. Traditionally, there is an option `-k` to the `man` command, but it is admittedly of limited use. In Ubuntu, the help button is a good place to start your search, but be prepared to have to ask.

Finally …

… when you have a problem, ask. There is a lot of help around.

Further information

More on Synaptic here: https://help.ubuntu.com/community/SynapticHowto

File system structure is described at: http://www.debianadmin.com/linux-directory-structure-overview.html