# System Administration and your Bio-Linux Machine

# System Administration and your Bio-Linux Machine

***These notes have been written for Bio-Linux 4. If you are running an older version of Bio-Linux, we highly recommend that you update your system as soon as possible.***

## *The bare minimum of Bio-Linux system administration*

1) Get a log book.

2) Use your log book! Record anything you do to your machine from a system perspective. This includes files added or changed, users added, programs added, fixes you carried out, etc. This will make it easier for others to find out what is happening on the system when you are unavailable, and will also save you lots of time as occasions will doubtless arise where you need to repeat something you did earlier, or recall what you did earlier because of downstream implications.

3) ***ALWAYS*** make copies of system files before editing them.

4) Keep a record of the packages, programs, databases and anything else you add to your system.

5) Get to know your machine – monitor the logs, monitor the space usage, know what is running and what should not be running on your machine.

6) Keep copies of key configuration files such as /etc/fstab, /etc/passwd, and /etc/group as printouts (e.g. stapled into your log book), or at the very least, on another system. Having copies of such files can save a lot of time should things go wrong in the future.

7) Clear up after yourself on your system. Don't use the manager account as a general dumping ground – the more organized your machine is, the easier your life as sysadmin will be. And the easier it is for people to take over your responsibilities if/when necessary.

8) Bio-Linux machines are updated automatically each evening. However, it is worth running a periodic check (e.g. once a week) to see what may require manual updating by you. There are user friendly tools such as synaptic that make this a simple process.

9) Try and establish a routine for administering your system. For example, log files and space usage should be monitored daily.

The most important Bio-Linux documents can be accessed through links on the web page:

**http://envgen.nox.ac.uk/biolinux_doc.html**

## *The Bio-Linux Environment*

Information about what is installed on a Bio-Linux box, where it is, and what the default settings are is given on the following web page:

**http://envgen.nox.ac.uk/envgen/software/archives/000499.html**

## User accounts on Bio-Linux

**regular user accounts** – these have the least privileges and thus can do the least harm. All bioinformatics work should be done within a regular user account, whether it be a pre-configured one, or one you have made subsequently. All the user accounts you choose to create when running the **setup.perl** program when you install Bio-Linux 4 are added to the groups: **users, audio, cdrom, floppy**

**root** – the almighty account. Root can do anything including destroy the system. Root should be used as seldom as possible. There are very few circumstances that require you to log in as root.

**manager** – the manager account is somewhere between a user account and the root account in that it is just a regular user, but has *sudo privileges* by default. This means that if commands are run with the word **sudo** preceding them, the command is run with root privileges.

**egdcmanager** – this is an administrative account for EGTDC staff to access should you need help with your system. You are under no obligation to allow us access to your machine, but sometimes it aids us in giving you help if we can get into your system and see what is there. We will request permission before logging into your machine. Please discuss with us if you are uncomfortable about having this account on your system.

## *Creating new users*

There are a number of ways that new users can be added to the system. Two possibilities are to use the **useradd** command on the command line or to use the graphical interface provided by **webmin**.

Default files added to each user's account can be found in **/etc/skel**.

To see the defaults used when running **useradd** without flags, type:

**/usr/sbin/useradd –D**

An example of creating a new user called *user6*, a member of group *users, audio, cdrom and floppy* using the **useradd** command is to type the following all on a single line:

**sudo /usr/sbin/useradd –m –g users –G audio,cdrom,floppy –s /bin/zsh user6**

You should now change the password of the user so that they can log in.

**sudo passwd user6**

The user will need to change their password as soon as they do log in, using the **passwd** command.

Further information about creating users, including how to access **webmin**, can be found at:

**http://envgen.nox.ac.uk/envgen/software/archives/000511.html**

## The power of sudo

It is good practice not to log on as root unless absolutely necessary. However, you frequently need "superuser" powers for carrying out system administration tasks. Sudo is a mechanism by which a "regular" user can take on superuser privileges for running a command. The sudo configuration file is **/etc/sudoers.**

To use sudo, just prefix the command you wish to run by the word **sudo**. For example:

**sudo less /etc/sudoers**

You will then be asked for *your* user password, (that is, the password of the user you are logged in as). If that user has the rights to run the command, then it will be run.
You can edit the sudoers file to give a fine level of control to particular users. For example, you may wish to grant the right to some users to do certain administrative tasks. You can do this without giving them full administrative rights on the machine by adding information to the /etc/sudoers file.

For more information, see the man page for sudo, or go to the web page **http://www.sudo.ws**

## *Checking the logs*

A package called **logchecker** is installed and running on Bio-Linux by default. It parses a number of log files on the system by default and sends an email containing anything it considers to be worth reporting to manager@yourmachine. Please see our FAQ for further details as well as information on how to get the report email sent to another address that is not on the Bio-Linux system:

**http://envgen.nox.ac.uk/envgen/software/archives/000512.html#logging**

The log files monitored by default on Bio-Linux machines are:

**/var/log/syslog**
**/var/log/messages**
**/var/log/auth.log**
**/var/log/daemon.log**
**/var/log/kern.log**
**/var/log/lpr.log**
**/var/log/mail.log**
**/var/log/user.log**
**/var/log/uucp.log**

## *What is happening each day*

There are certain scripts that are set up to run on your system at regular intervals. This can be daily, weekly, hourly, or in fact, at any time you specify. The process that controls this is called **cron**. You can see the things **cron** is running by listing the scripts in the following files:

**/etc/cron.hourly**
**/etc/cron.daily**
**/etc/cron.weekly**
**/etc/cron.monthly**

You can add scripts to these directories if you wish them to run hourly, daily, weekly or monthly. You can also exert more fine tuned control using cron, but that is beyond the scope of this document.

## *Dot files*

Much of the functionality and customization of your account relies on the presence of information in "dot" files, that is, files whose names begin with a dot. E.g. **.zshrc**, **.gnome-desktop**, **.history**, etc.

These files can be listed using the command **ls –la**

Please do not delete dot files unless you know what they are for and how their deletion will affect you.

Many of these files have a system-wide counterpart in the **/etc/** directory. For example, the system-wide settings for the z-shell are in **/etc/zshrc**, while customizations in individual accounts can be done by editing the **~/.zshrc** file. The settings in **~/.zshrc** (or any dot file in a user account) usually override system wide settings.

## *Shells*

The shell is like a buffer zone between your human thinking and the ugliness of the raw machine. The shell makes it easy to communicate with the machine. It interprets your commands and runs them. When you log into a Unix/Linux machine, you will be in a shell.

There are various shells available – each has its pros and cons, and each is slightly different in the way it acts and allows you to interact with it.

Common shells

| shell name | command | location | description |
|---|---|---|---|
| bourne | sh | /bin/sh | The "original" shell. Good for shell scripting. This is the most portable shell for writing scripts in. (Watch out – the sh shell on Bio-Linux is really the Bash shell) |
| c-shell | csh | /bin/csh | More user-friendly than bourne. Has features like filename completion, history substitution, job control, and is good for interactive commands (foreach loops, etc.) Not good for shell scripting (http://www.faqs.org/faqs/unix-faq/shell/csh-whynot/ |
| t-shell | tcsh | /bin/tcsh | An enhanced version of csh. Features include command line editing. This is a very user-friendly shell. |
| z-shell | zsh | /bin/zsh | A very feature-rich shell. Has good compatibility with sh and fairly good compatibility with csh. ***The default, and only fully supported, shell in Bio-Linux.*** |
| Bash shell | bash | /bin/bash | An enhanced shell for interactive and scripting use. |

More information on these and other shells can be found at:
http://www.faqs.org/faqs/unix-faq/shell/

All shells that are valid on the machine are listed in the file **/etc/shells**. The default shell on Bio-Linux is the z-shell.

To move from one shell to another, type its name. E.g. from z-shell to bash:

**manager@mymachine[homdir] sh**

**sh-2.05a$ tcsh**

**[manager@mymachine ~]**

To leave the shell and go back to your previous shell, type **exit**.

## *Environmental Variables*

There is a lot of material available on the web about environmental variables. One document to look at is:

**http://www.ee.surrey.ac.uk/Teaching/Unix/unix8.html**

The commands used to add or change the value of an environmental variable depend on the shell you are working in.

Bourne: **export MYENV=blahblah**

csh, tcsh: **setenv MYENV blablah**

*zsh:* ***you can use either of the above***

To see the result type **printenv MYENV** or **echo $MYENV**

To see all shell variables and environmental variables, use the command

**set | less**

## History

By typing the command **history**, you can see the last 15 commands you have run. By giving it a number as a parameter, you can alter the default behaviour to show a longer list of your past commands. For example, to show the last 50 commands, you would type:

**history -50**

The record of your past commands that is used to generate this information is held in a **dot** file.

The command history is separate for each shell. The command history file for the z-shell is stored in **~userx/.zsh_history**.

## *Login initialisation files*

When a user logs in, a number of initialisation files are read. These vary according to what the default shell of the user is.

System wide, the files of interest are:

**/etc/profile**
**/etc/zsh/zshrc**
**/etc/bash.bashrc**

Users can override the defaults set centrally, or add their own settings, using "dot" files in their account. There are a number of possibilities depending on their default shell and what they exactly want.
Key files include

**~/.zshrc**
**~/.profile**
**~/.login**

These files are sourced at different times:

**.profile** and **.login** are executed when a user logs in.
**.zshrc** is executed every time a new shell is spawned.

On this basis – information stored in a **.login** or **.profile** file should be that which needs to be executed at login time. This includes such things as:

- setting the **PATH**
- setting the default file protection (with **umask**)
- setting the terminal type
- setting other environmental variables

The types of things you might want in a **.zshrc** file include:

- setting shell variables
- defining aliases

**Note:** Shell initialization files are executed before login initialisation files – e.g. first **.zshrc** and then **.login**

Tips on editing these files can be found at:

**http://wwwhepix.web.cern.ch/wwwhepix/wg/scripts/www/shells/user.html**

## *Password files*

### /etc/passwd

The information stored in **/etc/passwd** takes the form:

**username:x:uid:gid:description:home_directory:default_shell**

***Do not edit the password file by hand unless you have good reason and know what you are doing!***

In general, this file is referred to for information only. The **x** in the second field is where an encrypted copy of that user's password used to be stored. The password information is now held in **/etc/shadow** (see below).

**uid** – user identification number. Each user should have a unique **uid**. By convention, uid's of less than 100 are reserved for system accounts. Root has **uid 0**.

**gid** – group identification number. Each user belongs to at least one group, and every group has a unique number. Numbering for user groups usually starts at 100. See the section on **/etc/group** below.

The information in **/etc/passwd** is used by many tools (e.g. **ls**) to display file ownership, etc. Thus, **/etc/passwd** needs to be world readable.

You can find out the **uid** or **gid** (and other information about users) using the **id** command. To find out about other users you have to run the command with **sudo**. For example, to find out my own **uid**, I can type:

**id -u**

**Note:** If you are an experienced system administrator, you may wish the default shell for your users on Bio-Linux to conform to the default shell they are using on other systems. In the case of new users, just create the user with the appropriate default shell. For users already on the system, they can change their own default shell using the **chsh** command, or you can change it for them as manager by using sudo. For example, to change the default shell for user bsmith to the **t-shell**, you would type:

**sudo chsh –s /bin/tcsh bsmith**

Remember that to find out what shells are available on the system, you can look at the file **/etc/shells**. However, remember that Bio-Linux is set up to be used with the z-shell by default. If you use another shell and have problems running any of the bioinformatics software, please refer to your local Linux guru (if there is one!), or if not, please email the helpdesk at helpdesk@envgen.nox.ac.uk.

## /etc/shadow

Password information for accounts is all held in the **/etc/shadow** file. While **/etc/passwd** is world-readable, **/etc/shadow** is only readably by root.

**/etc/shadow** contains password information, and other information such as account and password expirations, etc. The information in this file takes the form:

**username:encrypted_passwd:######:0:99999:#::::**

A blank entry in the password section means that no password is required to log on (generally a very bad idea!) A * in this section means the account has been disabled.

The third field is the number of days since Jan 1, 1970 since the password was last changed.

The fourth field is the number of days before the password may be changed

The fifth field is the number of days after which a password *must* be changed (99999 is a long time).

The sixth field is how many days a user should be warned before their password expires.

The seventh field is the number of days after the account expires that it should be disabled.

The eighth field is the number of days since Jan 1, 1970 that an account has been disabled.

The final field is a reserved field that could get assigned in the future.

### An aside into groups - /etc/group

Groups are the easiest mechanism to enable users to share files and other system resources.

All groups should be defined in the **/etc/group** file. The information in that file takes the following form:

**groupname:x:gid:users_in_group**

Note that by default when a new user is created, a group of the same name is created, and the default umask (explained later) allows full read and write access to both user and group. Although this in essence means that only the user themselves can access files they create by default, *this is probably **not** a particularly sensible default in most circumstances!*

The information on the command **useradd** given above shows how to circumvent this default behaviour.

If you look at the default **/etc/group** file on Bio-Linux, it should be clear what groups users on the system belong to, and in conjunction with the **/etc/sudoers** file, why the manager has sudo rights.

## *Files and permissions*

Basic file permissions are covered in the Introduction to Bio-Linux course. Here we cover how to grant permissions in more depth, and consider cases which are more likely to arise as the system administrator of a multi-user machine.

The command to change permissions is **chmod**. You have to specify who you are modifying the permissions of, what the new permissions are, and what file or directory to act upon.

The format of the chmod command is:

**chmod who ± permissions filename**

Who can be:

| | |
|---|---|
| **u** | means **user** and refers to the owner of the file |
| **g** | means **group**, and refers to the group the file belongs to |
| **o** | means **others**, everyone apart from those above |
| **a** | means **all** three, i.e. user, group and others |

Permissions can be:

| | |
|---|---|
| **r** | means **read** permission |
| **w** | means **write** permission |
| **x** | means **execute** permission |

An alternative way to define the permission level is by using octal numbers where

| | |
|---|---|
| **r** | is worth 4 |
| **w** | is worth 2 |
| **x** | is worth 1 |

The general command syntax using octals is **chmod ###** where the first # is a number representing the permission levels of the owner of the file, the second # is the permission levels for the group, and the third # gives the levels for all other users of the system.

So, saying **chmod 755** would give the owner 7 (r + w+ x ), the group 5 (r + x) and all other users 5 (r + x).

The command **umask** is used to set the default permission given to any new files and directories created. If you type **umask** without any arguments, then it will show you the current default setting.

The **umask** is a *bit mask* – that is it specifies permissions to ***deny*** when creating a file.

For example, if you set the **umask** to 333, all files will be created with read permissions for all users.

This is because 3 = 2+1, that is 3 = w and x. **Umask** takes away permissions, so if **umask** is given a 3, it will take away w and x.

Setting **umask** to 377 would give just read permission to the owner of the file, and no permissions to anyone else. 3 = w and x, and 7 = (4+2+1) = r and w and x. So **umask 377** takes away the owner's w and x permission, and takes away all permissions for the group and other users.

Note that setting **umask** to 000 would give the same permissions as **umask 111** on *files*. Execute permissions are never placed on files by default - you have to explicitly set execute permissions if you need them.

This is not the case for directories! (See the exercises.)

Directories must be ***executable*** to be accessible. If you wish to grant access to a file to someone, you must make sure that all directories in the path they must traverse from their account to yours are executable.

**Geek's Corner**

The reason that **umask** doesn't set the execute permissions on a file is that **umask** is applied to permissions given to the **open()** system call. Almost all applications give the mode 0666 to this call because they are creating data files (as opposed to executable files).

Examples using **chmod**:

To give read permission to someone in the same group for a file called "filename" in ~/intro_course.

**chmod g+r ~** is the same as **chmod 740 ~**
This gives permission to people in my group to read my home directory….too bad it's still not executable though! How will they get into the directory to read what's there?

**chmod g+x ~** is the same as **chmod 710 ~**
This gives permission to people in my group to get into my home directory…but they can't list anything in the directory.

If you grant execute permission on a directory, but not read permission, you can access files for which you know the exact name, but you cannot list files in the directory, or use the filename completion facilities of the shell. This is a good way to keep information in an account private, while granting access to specific files.

**chmod g+rx ~/intro_course** is the same as **chmod 750 ~/intro_course**
This gives permission to people in the group to list files in the intro_course directory

**chmod g+r ~/intro_course/myfile** is the same as **chmod 740 ~/intro_course/myfile**
This gives permission to people in the group to read the file myfile (assuming they already have permission to "see" into the intro_course directory).

## *Accessing files from other areas of your machine*

There may be situations where you want someone else to be able to copy or read one of your files, or, as the administrator of the machine, you may need to arrange for certain directories or files to be shared by various users of your system.

For the majority of Bio-Linux systems, assigning appropriate user groups and file permissions will be the best way forward

Information on permissions and how to share files with other users can be found at:
**http://www.linux-mag.com/2002-11/power_01.html**

## *Editing files*

In the basic course, we looked briefly at the use of **nano** and **nedit**.

When editing system files, we could encourage you to use **vi** or **emacs**.

**Vi** is not an easy editor for anyone who has only used programs like Word in the past. However, you will find it is fast, efficient, and often the best choice for the type of changes usually necessary for system files.

In fact, **vi** on Bio-Linux is not "true" **vi**, it is **vim**, an improved version of **vi**. There is lots of information available about **vim** through:
**www.vim.org**

A **vim** tutorial is available by typing **vimtutor** on the command line.

### Rules of good practice for editing system files:

Rule 1: **Make copies of system files BEFORE you edit them**.

Rule 2: **Make copies of system files BEFORE you edit them**

Rule 3: **See rules 1 and 2.**

It is always tempting to just change something quickly and not create a copy first. Any old hand at system administration will have tales to tell about the horrors experienced because of that one time they didn't follow rules 1, 2 or 3. Corrupted files can occur for a variety of reasons. These include a system crash in the middle of editing a file, or the replacement of a space with a tab or a tab with a space, or other problems with non-visible characters. Going back to an old working copy of a file is often the most efficient thing to do!

Rule 4: **Set up a naming system for copied files and follow it.**

For example, you could take a system file like **/etc/fstab**, copy it to **/etc/fstab_old** and then edit **/etc/fstab**. The next time you edit the file, you copy **/etc/fstab** to **/etc/fstab_old**, thereby overwriting the old **/etc/fstab_old** and still ensuring you have a relatively recent working copy if it should all go horribly wrong.

Another route to take is to name your files according to the date so you can backtrack versions. For example, you could have copied **/etc/fstab** to **/etc/fstab_070603** if it were, saying June 6, 2003, then you would rename it something else the next time you changed it.

This might not be particularly useful for /**etc/fstab**, but might be for other files. The only danger here is that you could end up accumulating a LOT of files. (Refer to point 7 on the second page of this document!)

## *Linux filesystems*

### Basics

Linux/unix systems are file-based systems. The default directory structure in Linux is similar to most other unix-based systems. A listing, with explanation of some of the basic default directories can be found in the article The Linux filesystem hierarchy: a short description:

**http://www.linuxnovice.org/main_focus.php3?VIEW=VIEW&t_id=126**

### df

The command **df** shows you information about the filesystems on your machine including how much space is taken up on different partitions of your hard disk (in this case – on larger systems, it might be telling you about "volumes" which are basically virtual partitions that can be made up of sections of many disks). **df** can be run with the **–k** or **–m** flag so that space is reported in kilobytes or megabytes respectively.

**You should run df on your machine daily (at a minimum!) to monitor space usage.**

**The configuration for your filesystem is held in a file called /etc/fstab**.

Great care must be taken when editing this file. Make sure to make a backup copy first! Corrupted **/etc/fstab** files are common problems on Linux/Unix machines.

Each line of **/etc/fstab** contains six space-delimited fields.

Full details of what is contained in this file are available on the man page:

**man fstab**

## *A couple of notes*

Filesystems are "mounted" onto disk areas. Disk areas have ugly names like **/dev/hda5**. Onto this disk area, I can mount a filesystem. It is this filesystem I then work with.

A mountpoint is an existing (empty!!) directory on your system. A filesystem can be mounted onto this empty directory. If you mount a filesystem onto a directory with files in it, those files are effectively hidden until you unmount the filesystem sitting on top – always use empty directories for mountpoints!

A common place to put directories that will server as mountpoints is **/mnt**.

The third field in **/etc/fstab** is for filesystem types. There are a number of different filesystem types, and for systems with removable media (such as CD's and floppies), one option is to set this field to read: **auto**. This tells the system to probe for the filesystem type. The order of the filesystem types checked is held in a file called **/etc/filesystems**. .

Mount commands commonly used for cdroms, floppy disks and memory keys are:

**mount /media/cdrom**
**umount /media/cdrom**

**mount /media/floppy**
**umount /media/floppy**

**mount /media/usbkey**
**umount /media/usbkey**

These are actually shortcuts for the full commands which for a cdrom for example, would take the form:

**mount /dev/hdc /media/cdrom**

The system looks for the information it needs to properly mount devices like those for cdroms and floppy disks in the **/etc/fstab** file. Note the **user** in the fourth field of the entries in this file for **/dev/hdc** and **/dev/fd0**. It is this term which enables "ordinary" users to mount and unmount the CD and floppy drives.

## *Ways to log into a Bio-Linux machine*

Full details on this topic can be found at:

**http://envgen.nox.ac.uk/envgen/software/archives/000506.html**

### Who is on the machine? Who has been on the machine?

The command **who** will tell you who is currently logged onto the machine.

The command **last** will tell you who has been logged onto the machine – **last** reads a file called **/var/log/wtmp** and can tell you who has logged in, when, for how long and where from.

## *Starting up and shutting down*

An outline of what happens upon startup, before you see the login screen, is listed at:
**http://www.luv.asn.au/overheads/linux-startup.html**

The file **/var/log/dmesg** can be very informative about what exactly was happening when the machine booted up.

The first screen you see after the Dell logo is the bootloader menu. Bio-Linux uses **GRUB** as the bootloader, though there is another popular one called **LILO**.

The function of the bootloader is to select the operating system to run, and to pass any parameters to the system kernel before it starts. As soon as you select an option, or the timer runs down, **GRUB** hands control of the machine to the operating system kernel.

**GRUB** installs onto the master boot record (**MBR**) of the hard drive, and also keeps configuration files in the directory /boot/grub. If you needed to change any settings you would edit the **/boot/grub/menu.lst** file, but it is unlikely you will ever need to do this.

More information can be found at:
**http://www.gnu.org/software/grub/**

## *Run levels*

These are described in more detail at:

http://qref.sourceforge.net/Debian/reference/ch-system.en.html#s-runlevels

| Run Level | Description |
|---|---|
| 0 | Halt |
| **1** | Single user |
| **2** | Multiuser – default runlevel on Debian |
| **3, 4, 5** | The same as 2 – can be customized by user |
| **6** | Reboot |

The first process run after the kernel has loaded is **/sbin/init** – this runs with **pid 1**.

The default run level on your system is 2.

0 and 6 are used for shutdown and reboot of the machine respectively.

## /etc/inittab

**init** reads the file **/etc/inittab** to determine what to do and when to do it.

The default run level for your system is defined in this file.

Entries in the **/etc/inittab** file take the form:

**id:runlevels:action:process**

For more information on this, please refer to the Boot Process section of The CTDP Linux Startup Manual (link at the bottom of document).

The series of lines where the action is, contains the script name: **/etc/rc.d/rc**. This script takes the argument at the end of the line, (0, 1, 2, etc.), which correspond to a run level. Commands in this file cause the execution of "kill" scripts, (those that start with the letter K in the appropriate directory) to run, and then runs the startup scripts, (those that start with the letter S). This is done for each run-level, (as can be seen by the series of commands in the **/etc/inittab** file).

Note that the "**K**" and "**S**" scripts are in fact softlinks to the scripts themselves, which are stored in **/etc/init.d/** This is important to remember if you edit scripts.

The **K** and **S** scripts are executed in alphanumeric order. Thus, if you are adding scripts, you can control them in the bootup/shutdown process as they are executed, by giving an appropriate soft link name in the appropriate run level directory.

### The /etc/nologin file

If a file called **/etc/nologin** exists, no user can log in. Only root can be logged in at the console. This can be very useful when you are troubleshooting and need the whole system to be up and functional, including networking, but don't want users logging onto the system. If you put information into the /etc/nologin file, that information will be reported to any user trying to log in. E.g. you can let them know the system is down for maintenance and to try back later.

### /etc/motd

The contents of the **/etc/motd** (message of the day) file appear on a user's terminal when they first log in. This is an extremely useful way to advertise messages about impending shutdowns ***if*** people are logging into your machine using terminal sessions such as **ssh.**

## *Shutting down the machine*

This can be done by anyone with access to the console by clicking on the **System** icon.

The machine can be shut down from the command line by anyone with **sudo** or **root** privileges.

The command usually used is **shutdown**.

When you shut the machine down cleanly, a number of things happen:

- users logged in are notified the system is going down. (Hopefully you have given them advanced warning of this shutdown.)
- all running processes are sent a signal telling them to terminate
- all subsystems are shut down gracefully
- all users still on the system are logged off
- all pending disk updates are completed (**sync**)
- drives are unmounted

## *What happens when the system crashes*

If your machine does crash, upon the next boot, a full filesystem check will be carried out using **fsck**.

**Fsck** goes through the system and tries to recover any corrupt files it finds. Anything resulting from this recovery operation is placed in a directory called **/lost+found**. Files in this directory will not always be complete or usable; it depends on how successful the recovery was.

## More on fsck

**fsck**, (filesystem check) checks the filesystem's consistency, reports any problems it finds, and optionally repairs problems.

With the exception of the root system, **fsck** runs on unmounted filesystems. To run **fsck** on the root system, you must be in single user mode.

You will probably not have to run **fsck** manually often, but please make sure you read the documentation on this important program so that you will understand how it works, and how to interpret the information it gives you, should a crash occur.

## *Starting and stopping services*

You can start and stop many of the services running on your machine by typing the name of the appropriate startup script, followed by **start** or **stop**.

If the script has been written properly, you should be able to find what options are available by just typing the name of the script. For example, typing:

**/etc/init.d/sshd**

will give you information on the options available for running this script.

## *Useful sites*

A really good reference for system administration of Linux and Unix systems:
**Essential System Administration – by Aeleen Frisch, ISBN 1-56592-127-5**

There is tons of information about the Debian Linux system at:

**http://www.debian.org/**

A major source of information for Linux can be found at The Linux Documentation Project:

**http://www.ibiblio.org/mdw/**

This is a portal to howto's, faqs, man pages, and online magazines about Linux.
**Linux administration made easy**

**http://www.tldp.org/LDP/lame/LAME/linux-admin-made-easy/**

**Linux Newbie Administrator FAQ**

**http://linux.about.com/library/bl/open/newbie/blnewbie_4toc.htm**