



Open Source Computing

Free Software Techniques for Normal People

Term 1: Notes



www.hbclinux.net.nz

Getting Comfortable

The purpose of this lesson is to help you become comfortable within a typical Free Software Environment.

Free Software graphics are built on a system called X-Windows, a free windowing environment from UC-Berkely. Most linuxes use the X-Windows version from XOrg, while some use XFree86. The difference is that Xfree86 is not fully GPL compliant.

A modern computer desktop uses a *Graphical User Interface* or GUI (gooey). Most GUIs have basic principles in common.

Different kinds of GUI used in free software are:

- **Gnome** (GNU Network Object Model Environment) from the Free Software Foundation. Designed to be a rich desktop which uses minimal resources and fully GNU GPL compliant.
- **KDE** (Kool desktop Environment) from KDE.org is a full-featured environment known for being very windows-like. While initially non-free, it is now fully GPL.
- **Xfce** (X-Office) from X.org is the native X-Windows desktop, known for it's simplicity. This is the standard for lightweight distributions like Zenwalk.
- **Other:** There are many more (i.e. **enlightenment**) - any casual search for linux desktops will turn up hundreds. The above are the big three.



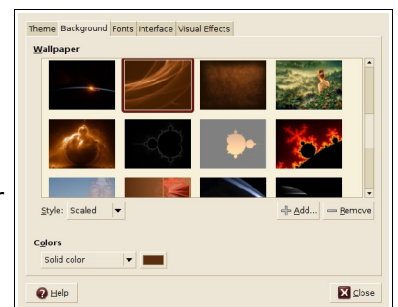
Customizing the Desktop:

Almost everything can be changed by rt-clicking on it and selecting "*preferences*". To find out what something is, point the mouse at it and wain a bit. (This is called "*hovering* the mouse".)

Wallpaper/Appearance

The big picture at the back in the *wallpaper*, it can be changed to any image or none at all by rt-clicking on it and selecting *change desktop background*. To add a background to the selection, use the *add* button or just drag the image-icon onto the dialog.

Select a wallpaper by double-clicking on it. The window can be dragged out of the way from the title bar, or alt+click anywhere.



Panels

The bars across the top and bottom are called panels. They are modified from the rt-click menu. You can change the color, make them transparent, or use a background image (like the bottom panel in the example).

Desktop Icons (Launchers)

Launchers are icons which appear in the panels or an the desktop which start programs running. The word icon is more properly reserved for those that represent objects like pictures or documents.

A launcher can be added to the desktop from any menu by dragging the menu item across. The exception are, *home*, *network*, *computer*, and *trash*. These have a special method in Gnome. They are added to the desktop from

Applications > Sysem Tools > Configuration Editor | apps > nautilus > desktop.

Windows

The overall theme of the windows and icons can be modified from the appearance dialog,

where you changed the wallpaper. Finer details can be modified from the Edit > preferences and Edit > backgrounds and emblems. Finally, System > Preferences > Window modifies the way the windows respond to controls.

Feel free to fiddle with the settings and dialogs until you have a comfortable interface.

Advanced Eyecandy

People with a fancy video card will have access to the Compiz-Fusion desktop effects. A basic set of simple effects comes with Ubuntu already, these can be activated from the appearance dialog. For a more complete set of effects, you need to install the *compizconfig settings manager*. This is not available from the *add/remove* menu, you have to use *synaptic* or the commandline (which is covered later).

Exercise:

See if you can locate:

- “About Ubuntu” message
- CD/DVD Creator
- “The Gimp” image editor
- Nelson Mandela
- Synaptic Package Manager
- Screensaver



Important Tools

In the applications menu there are two tools that you will end up using quite a lot. These are the Gnome Terminal and GNU Edit.

Gterm: This is the Commandline Interface (CLI) generically known as a terminal. Online assistance will take the form of CLI commands, even when a GUI method is available, because it avoids second-guessing. It is important to learn about this, so there is a lesson devoted to it. Here, I'll concentrate on getting you comfy:

Applications > Accessories > Terminal
(drag the menu item to the desktop).

The default terminal is rather harsh (black on white), we can change this.

Open a terminal. Edit > Profiles

Click “new”, name it for yourself, and an edit dialog pops up.

Of particular interest are the colors and effects tabs. A typical uber-hacker terminal will use grey on black color scheme with a transparent background, but you feel free to experiment.

Gedit: This is the standard text editor for Gnome – it is used to modify plain text files. This, too, tends to be black on white, so we'll want to alter that.

Applications > Accessories > Text Editor
(Drag the menu item to the desktop.)

The editor theme is altered from Edit > Preferences.

By now you should be getting a feel for how the GUI works.

System Layout

When you open your home folder, you get a special window run by the file manager. There are different file managers, the one in Gnome is called Nautilus.

At the top of the nautilus window is a location bar which shows you where nautilus is looking in your file system. The button on the far left toggles the view – click it and see.

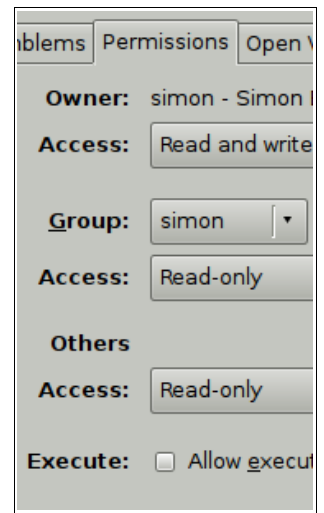
In all GNU/Linux distributions, everything in your computer is represented as files. GNU/Linux uses UNIX structure files. The features are:

Unified Tree

Hardware information is irrelevant to the file, so it is not represented in the file structure. There is no C:\ drive. The bottom of the file tree is called the root directory. It's icon is a disc drive, though it may be a partition on a drive. It's name is written "/".

Permissions

Not everyone can do everything with every file. Rt-click on a file and select properties. Look in the permissions tab. Typically every file has a set of UNIX permissions. Every file has an owner and a group. Permissions get set to read, write, or execute the file according to whether the user trying it is the owner, a group member or other.



A side effect of permissions is that the file extension is actually meaningless. A dot-exe file is not executable (you cannot run it) unless you have execution *permission*. In unix systems, executable files usually do not have any particular extension.

Nautilus usually uses the file extension only as a rough guide to what to do with the file. It actually looks inside to see what it is. This means that the icons you see are only a best guess as to what sort of file you have. You have to open it to be sure.

Superuser

The top-level user in Unix is called root (or rube). This user has total access, hence the saying "Root, God, what's the difference?" In Ubuntu linux the root user has been disabled by default. Instead, the first created user (that's you) has "administration priviledges". This means that your normal user password will grant you limited god-powers.

Hidden Files

Edit > Preferences > Show Hidden

Your home directory contains a lot of information about your account and your personal settings. These files clutter things up enormously so they are kept hidden

Roles of the Directories

GNU/Linux	Windows	notes
/bin /sbin	C:\WINDOWS; C:\WINNT	Binary files that are part of the core system. /bin is for main programs, /sbin is for utilities.
/boot	none	Files vital to the system's boot process.
/etc	C:\WINDOWS; C:\WINNT	System Configuration Files
/home	C:\Documents and Settings	User Workspaces
/lib	C:\WINDOWS	Libraries. Linux libraries are like .dll files.
/media	D:\; E:\; F:\ etc	Removable Media mount points
/mnt	H:\ etc	Non-removable media mount points
/opt	none	Large non-nix programs stored here
/proc	none	Used by the kernel to access processes. It also holds vital status info.
/root	C:\Documents and Settings\administrator.	Superuser home directory
/tmp	C:\temp, C:\WINDOWS\temp	Temporary Files
/usr	C:\Program Files	Files for non-core programs.
/var	none	Files that change frequently, like logs. /var/log is the equivalent of the "Event Viewer" in Windows.

exersize: find out where icons and fonts are kept.

Common Tasks

Now the environment feels comfortable, it is time to explore the things you would most often do with your computer.

Network

The network tool is a widget that looks like two monitors. Click on it to find which networks, wired or wireless, are available to you.

Printing

The Common Unix Printing Tool (CUPS) and it's plugins handle printers. Access from System > Administration > Printing. Configured printers will show up here. You can add a new printer, but only supported printers will work. Fortunately, there are thousands of them.

Internet

Few ISPs provide explicit Linux support, however, it is always possible to connect anyway. You need to know the following things (shown with examples):

Primary DNS Server IP	203.248.100.100
Secondary DNS Server IP	203.248.100.101
Incoming Mail Server URL	pop.myisp.com
Outgoing Mail server URL	smtp.myisp.com
Encryption (If Any)	none
Authentication Method	password
Mail Account Password	1337accntpwd

It is unlikely that you will need the first two. After the ISP has finished telling you they don't support Linux, you tell them "That's OK, I just need the connection details." And then ask for the things on that list.

Web Browsing: Firefox

Firefox is configured from Edit > Preferences – it is a good idea to tighten up the security quite a bit here, and install some plugins. The most common firefox plugins are handled from the add/remove menu. Otherwise you use the tools menu from within firefox.

E-Mail: Evolution

The Evolution PIM suite includes an e-mail client. When you first start out, it will tell you what you need. Fill out the forms. There is a gotcha in the outgoing mail form: NZ ISPs very seldom require a password for outgoing mail. If you have trouble, try unchecking that box.

Effective mail settings include, disabling images and html in e-mail, and sending in plain text only. Disabling those things is considered prudent, and sending only plain text is considered polite.

Productivity

Applications > Office

Have a go starting OpenOffice.org (OOo) Writer. It has much the same look and feel as MS Works Writer or MS Word. OOo stores it's data as XML. If you rename a .odt file as .zip, then open it, you'll be able to see what's inside.

An intreguing extra is the "export to pdf" option in the File menu. PDF is an open source format and international standard.

There are example files for use with OOo in the examples folder. These all use free formats. While OOo can read restricted formats, like DOC, it is getting very good at reproducing macros and VB code effects. This means that you become vulnerable to some MS Word viruses. The secure way to view .doc files is to change it's extension to .txt and use a text editor.

There are online OOO tutorials at:
<http://www.tutorialsforopenoffice.org/>

Adding Fonts

New fonts can be added to the system by entering "fonts:////" into the text location bar in nautilus, then dragging the font icons over to the open window.

Image Manipulation

Applications > Graphics > GIMP

The GIMP is the GNU Image Manipulation Program – it is a GUI for the Image Magic programs.

Photo Management

F-Spot a photograph indexing and sharing program. It allows export to web photo sharing applications, including picasaweb and flickr, as well as automated mailing.

Multimedia

Applications > Sound and Video

Rythmbox

Alternatives: Amarok, XMMS

Totem

Alternatives: Mplayer, Banshee

Installing Software

Applications > Add/Remove Software

System > Administration > Synaptic Package Manager

Software is obtained, in the first instance, from centralised repositories. These repos store software in a form that is easy to install to the particular distros they are designed for, called packages. The official Canonical (Ubuntu) repos contain over 4000 seperate packages, but you typically need several packages for one application. On top of that, there are numerous 3rd party repos which add functions that the official company may be unwilling to support.

Repositories are managed from

System > Administration > Software Sources

... the official repositories are included by default, but not all are enabled. Any enabled repositories will be routinely checked for updates. If you are short on bandwidth, you should leave only the security updates repo enabled, activating the rest as you need them.

The most popular repo to add is called medibuntu. This contains support for restricted format multimedia which Canonical is unwilling to supply for various legal and ideological reasons. These will be covered in more detail later in the course.

Programming:

Access to the source code is an important freedom in GNU/Linux, this means that you have unprecedented access to all aspects of the system. It is useful, therefore, to know a little programming.

Ubuntu does not come with programming tools or source code by default.

Build-Essentials – includes everything needed for programming in a GNU/Debian (i.e. Ubuntu) environment. There are also plugins for a range of languages besides C++

Free Pascal + Lazarus - Pascal is a teaching language and Lazarus is a GUI development library for Pascal.

dotGNU project: frees C# from the .NET framework.

Media Center

The **Elisa** package runs on top of the Gstreamer

LinuxMCE

<http://www.linuxis.us/linux/media/howto/linux-htpc/>
... this is a package designed to run on top of kubuntu

Linux HTPC

<http://www.linuxmce.org/>
... this a set of HTPC project howto's.

Geneology

Genealogical Research and Analysis Management Program

An Open Source genealogy program written in Python, GRAMPS has the ability to import GEDCOM files that are used in such programs as FamilyTree Maker for Windows and can produce reports in various formats such as the popular ABIWord and OpenOffice.org formats as well as HTML and PDF.

Eye Candy

www.gnomelook.org

This site includes files to help customize every aspect of the GUI environment.

Exersizes:

Look through the examples, play the files, edit some of them, read the notices.
Have a go at the GIMP tutorial included. Make a personal Logo.

Homework:

Create a windows share for your public folder. (rt-click on the folder and select share.) The share type is SMB and the workgroup should be "HBCLUG".

Commandline

Congratulations, you have entered the world of GNU/Linux and Open Source. While you have just been exposed to the Gnome desktop, we won't actually be needing the GUI much the rest of the time. Instead we will be strongly focussed (OK: obsessively focussed) on the Command Line Interface (CLI) or "Shell".

Tip: I use the Bitstream Vera fonts extensively. But when I want to indicate text that appears onscreen or something you should be typing, perhaps, I will use a typewriter style called FreeMono.

Terminally Yours

A shell is a program that allows you to start other programs. Linux *terminal* programs start a shell automatically. You already know how to start the Gnome Terminal.

The shell is taught because it provides a very fast way to gain understanding about how your Linux computer works. Or, indeed, any Linux or UNIX computer. What you will learn here is completely generic and will work on all UNIX-like machines.

Open the terminal. It will show a prompt:

```
simon@hbc1ug01:~$
```

The general form of this is user@computer (the ~\$ part I'll get to later). This is called the shell prompt – and it tells you that the terminal is already running a shell for you.

Joan Bloggs, will have a prompt that looks more like:

```
jb1o001@ubuntu:~$
```

But I will use my name and so on in examples.

Lots of other programs can run in a terminal, but Gnome always starts a shell by default since you can start *any* other program from that. For example:

Hello World

At the shell prompt, enter the following: `echo "hello world"`

```
simon@hbclug01:~$ echo "hello world"
hello world
simon@hbclug01:~$
```

the **"echo"** part is the command, the **"hello world"** part is the argument or input to the command. All echo does, is write whatever comes after it to the screen. In general, commands will have three main parts:

```
<command> -<option> <input>
```

The option will modify how the command behaves. For eg. `echo -n hello world`. (try it now.) To find out more about what a command does, you read the manual. If you enter `man echo`, for example, you will get a description of the command and its options. In general, the manual page for a command is `man <command>`. for this reason, the manual pages are known as "man" pages. If a man page is too big to fit on the screen, you can scroll through it with the arrow keys. When you are done reading, exit by pressing "q". You can read a man page and use the terminal at the same time by opening two terminals on your screen (or one on another screen if this one gets too cluttered). Just click the terminal icon again.

Exercise: Read the echo man page. From the information in that, work out what you have to enter to get the following output:

```
simon@hbclug01:~$ echo <your input here>
*****
*** Hello World ***
*****
simon@hbclug01:~$
```

To find out more about the man command, just enter **man man** .

Tip: For a quick look at command options, enter: `<command> -h` or `<command> --help` will usually give you a summary of the man page. ("`--`" double-dash is often used for options that use whole words while "`-`" single-dash is for single-letter options.) For example:

```
simon@hbclug01:~$ man --help
usage: man [-c|-f|-k|-w|-tZT device] [-i|-I] [-adlhu7V] [-Mpath] [-Ppager]
          [-Cfile] [-Slist] [-msystem] [-pstring] [-Llocale] [-eextension]
          [section] page ...
```

Of course, there is a command for pretty much everything you may want to do. The trick is knowing what they are. A good way to find out is to use the apropos command (`man apropos`). `apropos <argument>` will search the man page names and descriptions and produce a list of all the commands which have something to do with `<argument>`. For example:

```
simon@hbclug01:~$ apropos e-mail
evolution (1)          - e-mail, calendar, addressbook, and to do list application
```

Enter `man evolution`, to find out what this does. Enter `evolution` to run the program.

Exercise: Use the apropos command to find out which program will start a text-based browser. (text based browsers are useful for avoiding popups and advertisements.)

More useful commands...

Make a simple text file:

```
echo "hello world" > myfile
```

By default, echo writes to the screen. But if I point it to a name with the ">" character, it will create a file with that name and write to it. If the file already exists, it won't bother to create a new one. To read the file, try: `cat myfile`
`rm myfile` when you don't want it any more.

Tip: Command completion: When you are unsure of a command, or just don't want to type so much, you can get away with typing the first few letters and pressing <tab>. If there is more than one possible command, the shell will provide you with examples. If there is only one obvious command, then it will complete that command for you.
Try it: type `ap` then press tab. Again with `ec` and `ech`.

How to Manipulate Documents:

Make sure the file "gnugplv2.txt" is in your home folder. This is the file containing the rules under which Linux is distributed, we are going to use this to practise reading and editing text files.

The cat command will put the content of a text file on the screen. So let's try to read the file:

```
cat gnugplv2.txt
```

Got that?

The cat command is more useful for smaller files: `cat /etc/passwd` :) It is also used to combine text files quickly. Now try:

```
more gnugplv2.txt
```

Better? The program gets its name from the "more" prompt at the bottom of each page displayed. (Press q to exit.)

This is OK for reading a document all the way through like this. But what about hunting back and forward through a large document? The more program will only let you go forward: if you zip past what you want you must start again!

```
less gnugplv2.txt
```

i.e. "less is more" ... get it?

This program lets you step through a document with the arrow keys. Press q to exit.

Tip: Here we have a lot of commands that are basically the same. It is a pain to keep writing things out all the time – so there is a shortcut. If you press the *up-arrow*, the previous command will show up at the prompt (try it).

From there you can *back-arrow* to the bit you want to change, *delete* the unwanted text, and write in the new stuff. i.e. In the last command, we only needed to change "more" into "less". To do this quickly, we could use the up-arrow (twice now) until the more line showed, then the back-arrow until the cursor is at the first "g" in "gnugplv2", then *backspace* over the "more", then type "less", then press enter. (It's easier to *do* than to describe.)

Of course, if you want to alter the document, rather than just read it, you need a text editor.

```
nano gnugplv2.txt
```

This is a very small text editor which runs inside the terminal. It is quite useful and intuitive to use. The commands are along the bottom.

Use the arrow keys to move the cursor around the screen, text will be inserted where the cursor sits, use backspace and delete keys to remove unwanted text.

^H = ctrl+H will give you some hints

^O = ctrl+O will save the file

^X = ctrl+X will exit (answer No, don't save the modified buffer)

However, most people prefer to use something prettier, like:

```
gedit gnugplv2.txt
```

You will have to exit the program to get your command prompt back.

Exercise: Use nano or gedit to make a text file which only contains the preamble to the GNU GPL and save it as "gpl-preamble.txt" in your home folder.

Finding out about the computer:

Try each of these suggestions and see what you make of them.

`lspci` lists all the pci cards in the computer. See if you can identify the sound card. The graphics card? Are there any modems?

`uname` shows information about the kind of Linux you are using.

`uname -r` tells you the kernel version (Linux version as opposed to the Ubuntu version)

`uname -a` tells you everything

`dmesg` shows you the system log. Useful troubleshooting information shows up here.

`dmesg` is a very long file. To read it all, you could write it to a text file and read it with `less`:

```
dmesg > log.txt
less log.txt
rm log.txt
```

This can be tedious ... instead of redirecting the output of `dmesg` to a file, how about redirecting it into the `less` command? You do this with a pipe (`|`):

```
dmesg | less
dmesg | grep error
```

`grep` is a powerful command, just look at its man page. Its basic function is to look for keywords in a file. In this case it looks through the entire system log for the word "error". You can look for other things:

```
cat /etc/passwd | grep simon
```

... looking for my password are you? Try your own username. (This used to be an easy way to discover someones password, but has since been fixed).

And, while we're on the subject:

`passwd` will change your password.

Files Away

In Linux, everything boils down to files. A file is a chunk of memory containing ones and zeros, which has a beginning, an end, and a location. A file that contains the location of other files is called a *directory*. This creates a hierarchical structure a bit like a tree.

Files are often thought of as bits of paper while directories are thought of as folders. Extending this metaphor, folders are said to contain files. However, it is a useful discipline to talk about directories and files rather than folders and documents.

The shell can access directly any file in your working directory. To find out what your working directory is, enter `pwd` (print working directory) and you should get something like this:

```
simon@hbclug01:~$ pwd
/home/simon
```

This tells me I am "in" my home directory. Have a close look at this: the first dash "/" stands for the base directory. This is called the *root* directory. Everything else is attached to this. In some computers this would be called `C:\` but in Linux we do not distinguish hardware like this.

Instead, the filesystem contained in the hardware is represented as a directory and merged seamlessly into the overall file tree. The hardware can be anything that stores data, a hard disk, a flash drive, or a CD-ROM.

Directory *names* get separated by a slash. The full line is called the path. So my home directory is called "simon" and its path is `/home/simon` (The forward-slash in Linux plays the same role as the back-slash in certain legacy operating systems.)

You can have a look at what is in the working directory with the `ls` (list) command:

```
simon@hbclug01:~$ ls
Examples  music          photos          projects        tutorial.todo
tux_giant.png
```

You'll have different things. Notice the names are colour-coded: you can work out what these colours mean by comparing with the desktop. In the example, blue is a directory, cyan is a link to another directory, and pink is a picture. (Colour coding is annoying to type, so I will leave it off from now on - unless it's important.)

To get the same effect without colours do:

```
simon@hbclug01:~$ ls -F
Examples@  music/      photos/     projects/   tutorial.todo
tux_giant.png
```

So the link has an @ sign, and directories get a /
Technically, all directories should be written with a trailing slash, we just get lazy sometimes and leave it off when the meaning is obvious.

To find out more about a file, try `ls -l`

```
simon@hbclug01:~$ ls -l Examples
lrwxrwxrwx 1 simon simon 26 2006-10-02 20:58 Examples -> /usr/share/example-
content
```

If you don't have the `Examples` file, just pick another one to try.
(What happens if you try `ls -l examples` ? Now you know what "case sensitive" means.)

Long Format Demystified:

`ls -l` prints "long format". Each part is there to tell you something about the file.
The first character tells you what kind of file you have. In this case, l for link. A link is a way to make a file in some other place else act like it is right here. You could also have d for directory, and just a "-" (dash) for a regular file. There are others you will learn about in due course.
The "rwx" part, three times, are the security permissions. These tell what you can and cannot do with this file. For more about permissions, see the manual file for `chmod` (`man chmod`).

The number (1, in this case) is the number of files associated with this file. A directory can contain hundreds of files.

My name (twice) means that I am the owner of the file (1st time) and that the file is the member of the group named after me (2nd time). It is possible to specify different permissions for group members. See the man page for `chgrp` and `chmod` for more details.

The next number (26) is the file's *size*, in bytes: this is a very small file since it doesn't do much.

Next is the date and time it was last altered. This is called the *timestamp*.

Next is the name of the file, followed by any notes. In this case the notes indicate that the file is a link to the `/usr/share/example/content` directory.

If you try this on a directory (`ls -l music`) you will get a listing of all the files *inside* the directory. If you want to find the properties of a directory, you have to use the `-d` option:

```
ls -dl music
```

Aside: Permissions at work-

You've seen `man chmod`, so you know that this program can change the permissions on a file.

Let's make the `Examples` link more restrictive by stopping everyone from being able to alter it. This involved removing the write permission by `chmod -w Examples` (do it now).

Can you explain what happened? Perhaps it would help if you looked at the `chmod` program itself: `ls -l /bin/chmod` (Who owns the program? Who gets to `x=execute` it?)

Special Files:

Try `ls -a`

Suddenly you see more files than you thought you had! This is because many files are hidden, for a variety of reasons. Some of the things you may find are explained below:

`.mozilla`

Any file starting in a dot is a hidden system file (used to store your personal configuration).

`myfile~`

Any file ending in a tilde (or squiggle) is a backup file. Programs like `gedit` can create backups whenever you edit a text file. These can end up taking up a lot of space, so you need to be aware of this.

`.`

Yes, that's a dot. The dot directory is an alias for the current working directory. It is used to direct the computer to look in the current directory to find something. Useful for running your own programs (outside the scope of this tutorial).

`..`

Two dots this time. The double-dot directory is the parent directory.

To access things not in the working directory, you have to specify the path to it:

`ls projects/tutorial` uses a path which is *relative* to the working directory

`ls ..` uses the *relative* path which is the same as `ls /home`

`ls /usr/share/examples` is a *complete* path from the root directory

Try these on your computer with *your* directories.

Bonus: now you know about hidden files and text editors, you can try something *real* ! The file that handles your terminal is called `.bashrc` and it lives in your home directory. Reading this file is a fast way to learn about how the shell is set up. If you don't have colors, for example, you can edit this file to correct that.

Check you have the file in your home directory with

```
ls ~/.bashrc
```

(If you *don't* have it, you'll have to make it. I have a sample file called "bashrc" in the tutorial tarball. Copy that over to your home directory and rename it to ".bashrc" so it vanishes.)

Open this file with your favorite text editor.

```
gedit ~/.bashrc
```

Any line that starts with a `#` sign is a comment and is ignored by the computer. Look for something like this:

```
# enable color support of ls and also add handy aliases
if [ "$TERM" != "dumb" ]; then
    eval "`dircolors -b`"
    alias ls='ls --color=auto'
    #alias dir='ls --color=auto --format=vertical'
    #alias vdir='ls --color=auto --format=long'
fi
```

The "`alias ls=`" line has to be uncommented to make colors work. You can uncomment the other two alias lines too, if you want. Can you see what they do? Anyway, when you are finished, save and exit. Now try those commands out.

To change your working directory, use the `cd` command:

```
simon@hbclug01:~$ cd Examples
```

```
simon@hbclug01:~/Examples$
```

Notice: the command prompt has now changed!

Understanding the Command Prompt:

You've noticed that the prompt tells you who you are and which computer you are working at? Well it also tells you your working directory. That “~” in there stands for your home directory! Confirm this by entering `pwd` now.

If you do `cd ~` from anywhere in the entire filesystem, you will go right back to home: useful if you get lost or confused. Try it.

That dollar sign on the end is special too – it means you are working as an ordinary user. If you were working as an administrator, it would change to a hash “#”. It is unlikely you will see this, it is more secure to use other means.)

If you enter `ls` now, you will get the list of files in the new working directory.

Manipulating Files:

There are all sorts of things you'd like to be able to do with files. If you invoke **apropos file** you will see a whole slew of possibilities. Here are the common ones, try them out as you read them: (remember – everything will happen in your current working directory unless you specify otherwise.)

`touch myfile`

... creates a dummy file with nothing in it. It also updates an existing file's timestamp.

`mkdir mydirectory`

... creates a directory

`cp myfile myfile2`

... copies a file. You can copy to and from other directories by specifying the path with the filename.

`cp /usr/share/pixmaps/mozilla-firefox.png ~`

... should copy the firefox logo into your home directory.

`mv mozilla-firefox.png firefox.png`

... renames a file. Can also move it if you specify a path. Test it out:

`mv firefox.png mydirectory`

`ls mydirectory`

`rm myfile`

... deletes a file. Note: `rm *` (**don't do this**) will remove **every** file in your working directory, including the system files. These are *permanently* deleted – go directly to oblivion, do not go to *trash*, do not get restored.

If the working directory is your *home* directory, this will *break your account* on the computer and you need help from the administrator (and we've not had that lesson yet) to get it working again.

Tip: Create a work directory for your experimental stuff so a mistake won't be too costly.

`mkdir ~/stuff`

`cd ~/stuff`

... notice the full path? This is because I am not making assumptions about what your current working directory is. This leaves you free to play around and explore but still let me get the instructions right.

`mv myfile ~/.Trash`

... moves a file to trash instead of deleting it.

`rm mydirectory`

... we don't need it anymore – but wait: what happened and why?

How about `rmdir mydirectory`

... removes a directory – but wait: what now?

The security system is saving you from yourself. Check out the man page of `rmdir` and `rm` to see if you can find out how to delete a directory even when it is not empty.

Wildcards and Variables:

In the `rm` section, I showed you one use of the `*` wildcard. If you don't know about these, here is a quick list:

card `*` = any combination of characters; eg. `ls -a .*` lists all hidden files.

card `?` = any single character; eg. `ls assignment??*.pdf` lists all files called “assignment” which end in two additional characters (like numbers 00 to 99)

We can also make short words stand for longer instructions like this:

```
export foo="a random string"
echo $foo
```

... here `foo` is the variable *name*, it is assigned whatever is inside the quotes. To see what a variable contains, the `echo` command has to be used with a `$` sign in front of the variable name. Just saying `echo foo` doesn't work – try it and see why. Another use:

```
export favdir=/usr/share/pixmaps
cd $favdir
```

... saves typing if you visit `/usr/share/pixmaps` a great deal.

Linux has a set of handy variables already programmed in called *environment* variables. These all use capital letters, like:

```
echo $HOME
... the same as ~
echo $PATH
```

... shows the list of directories the shell looks in when you type a command. Whenever someone talks about a program being in your *path*, they mean it is in one of these directories.

Tip: Create a personal programs directory

```
mkdir ~/bin
export PATH=$PATH:~/bin
```

... this makes a directory called `bin` in your home directory and adds it to the `PATH` environment. Any programs you now put into your `bin` directory can be executed (run) just by typing their name into the terminal. (You have to make sure the name is unique though.) You cannot do a lot with environment variables as a normal user. Indeed, the full use of variables and so on is outside the scope of this tutorial. However, you will need them later and it helps to be aware that they exist.

Important Files

There are a lot of special files governing the way your computer works, here is a short tour.

`/proc`

This is a useful directory to explore – warning: you may look but do not touch. This is a place to learn far more than you ever wanted about how your system runs.

```
cat /proc/cpuinfo
```

`/var/log`

This contains the system log files – very important for troubleshooting.

`/etc`

contains a lot of configuration information. Before editing any configuration file, create a backup: `cp file file-old` This way, if you mess up, you can easily rescue your system.

So – **before we go on:**

Worst Case Scenario: Help my computer won't start!

At some stage you will find you have fiddled something to the extent that your computer will no longer boot. Linux is so easy to fiddle with, and so tempting, that this situation is inevitable. Don't panic – the very fiddleableness that got you into this mess can get you out.

Before You start:

If you prepare ahead of time, recover becomes simple.

1. Make a backup before you start experimenting
2. Backup any file you intend to edit
3. Use the Commandline
4. Log your activity – by hand

Small mistakes in configuration will seldom do more than stop a particular program running quite right. Correcting this is usually a matter of restoring the old, working, version of your configuration file.

If you backed up a file with:

```
cp foo.bar foo-old.bar
```

Then you can restore it with

```
cp foo-old.bar foo.bar
```

Larger mistakes may leave the system unbootable.

Any Linux CD that can act as a live distro is a rescue disk. It uses it's own configuration files and loads into RAM. You'll typically access your hard-drive through the live disks *places* menu and restore the errant file that way. The fine details of *mounting* will be covered under administration.

Now that's out of the way;

/etc/apt/sources.list

... this is the file that is used to configure repositories.

/etc/X11/xorg.conf

... this is the file that handles human interface devices like video and keyboards.

/etc/fstab

... this file handles the way file-systems are handled in linux.

```
man fstab  
man mount
```

/etc/sudoers

...this file controls who gets what kind of administration priviledges.

Administration

The admin functions are accessed with the sudo command.

```
man sudo
```

You have already met command that would not run because you didn't have the correct permissions. Sudo (**s**witch **u**ser **d**o) allows you to run cammands as if you are somone else. Usually you choose to be root.

Try:

```
fdisk -l
sudo fdisk -l
```

The first time you use it for a particular task, sudo will ask for your password. Repeated uses in a short time do not require a password, but leave it a while and you have to reauthenticate yourself.

Installing Software

Software can be installed from repositories. The tool which handles this is in Ubuntu is called APT (Advanced Package Manager). In openSUSE it is YaST (Yet another SetUp Tool) and in fedora it is YUM (Yellowdog Update Manager). There are others, but APT and YUM are the biggies.

In Ubuntu, APT is implimented with apt-get. So to install the Alien Arena game you enter:

```
sudo apt-get install alien-arena
```

Using it means you need to knew the name of the package in advance. There are ways to look for packages, it's just that synaptic is easier. Administrators use apt-get mostly for updates:

```
sudo apt-get update
```

And you'll find that people will always tell you the apt-get command when they think you need to install something. eg. If you have dual or quad core CPUs, you may benefit from the IRQ-Scaling package, which shares the load more evenly between them.

I could tell you to open synaptic, then search for "IRQ" and check the box next to "irqscaling", or I could just tell you to enter:

```
sudo apt-get install irqscaling
```

Sometimes you need programs that are not in the repositories. Not often because, as soon as something gets popular, someone puts it in a repo some place. But still... Ubuntu packages use the debian format, and have file extension .deb. The tool that handles these directly is called dpkg.

If you want to install myprogram.deb, you cd into the directory where you saved it and enter:

```
sudo dpkg -i myprogram.deb
```

Filesystems fstab and mount

Computer files live on some physical media, a copper disk in a hard-drive, a pattern of etch-marks in aluminium in a CD-ROM, whatever.

The rules for accessing the physical device are stored in block special files in the /dev directory.

Lets have a look at a hard drive:

HDDs use block special files like this: /dev/sdxn where the x is a letter and the n is a number.

x = the physical hard drive plugged in to your computer

n = the number of the partition.

example: the windows C:\ drive will be /dev/sda1

Common filesystems are:

- **fat32:** (vfat) used by DOS, Win98, iPODs and key drives.
- **ntfs:** used by NT, XP and Vista
- **ext3:** used by linux
- **Xfs:** used by Unix and linux

- **Reiserfs:** used by linux

To access the filesystem, it has to be mounted – the command is:

```
sudo mount -t filesystem /dev/device /mountpoint
```

A mountpoint is an empty directory where you want the files in the media to start out in. It can be anywhere, but usually you put it in /mnt or /media.

examples:

a keydrive may mount like this:

```
sudo mount -t vfat /dev/sde5 /media/keydrive
```

(keydrives are normally fat32, and keep their data on the *fifth* partition.)

a cd-rom may mount like this

```
sudo mount -t iso9660 /dev/sdc /media/cdrom
```

(no partition number: cd-roms are not usually partitioned)

a floppy drive like this:

```
sudo mount -t vfat /dev/fda /media/floppy
```

(floppies have a special name for historical reasons)

Sometimes the user files are kept on a special partition

```
sudo mount -t ext3 /dev/sda3 /home
```

The exception to this pattern is ntfs. This is a secret file system, so it needs a tool to handle it.

```
ntfs-3g /dev/sda1 /mnt/win
```

A typical GNU/Linux installation needs to be able to handle lots of partitions and file systems without bothering the admin, so the fstab file was created. This lists all the partitions and devices that the computer is expected to handle all by itself. A typical fstab file will look like:

```
~$ cat /etc/fstab
# /etc/fstab: static file system information.
#
# <file system> <mount point> <type> <options>          <dump>  <pass>
proc           /proc         proc         defaults                0        0
/dev/sdb1      /             ext3         defaults,errors=remount-ro 0        1
/dev/sdb3      /home        ext3         defaults                0        2
/dev/sdb2      none         swap         sw                      0        0
/dev/sdc       /media/cdrom0 iso9660      user,noauto,exec       0        0
/dev/sdd       /media/dvd0  udf         user,noauto,exec       0        0
/dev/fd0       /media/floppy0 auto         rw,user,noauto,exec    0        0
```

(This has been tidied up to make it more readable.)

Any line starting with a # is a comment, it is not read by the computer.

The UUID in your fstab file is an abstract way to refer to the disk without specifying the dev file.

The first column tells linux which device file to use, the next says where it goes, then what type of file system is on there. The options say how the data is treated. Dump is for troubleshooting and pass refers to the order the file system is accessed at boot.

The above example is a dual boot. /dev/sda is the windows C:\ drive. It has a single partition occupying the entire drive, which is formatted ntfs. The user wants this filesystem mounted at boot time. The user needs to:

1. Create the mountpoint:

```
sudo mkdir /mnt/win
```

2. Add it's description to fstab:

```
cp /etc/fstab /etc/fstab-old
nano /etc/fstab
```

Add the line:

```
/dev/sda1 /mnt/win ntfs-3g defaults 0 0
```

3. Save and exit. Reboot.

When the user reboots, /mnt/win will show up in places. It can be dragged to the desktop for handy access.

Note: it is possible to access the linux partitions from Windows. This is not a good idea because Windows notoriously does not like to share. As a general rule, do not let windows assign a drive letter to linux or reformat the linux drives (it sometimes insists that linux is malware!)

Creating a backup

```
cp -a /home /media/usbdrive
```

This is a very simple backup method, it just copies all your files to the storage media. Restoring the files is a matter of copying the files back:

```
cp -a /media/usbdrive /home
```

Archiving the backup:

```
tar zcvf /media/usbdrive/home.tar.gz /home
```

... the **tape archiver** is very flexible and can be used to create incremental backups too.

Restore the backup with

```
tar zxvf /media/usbdrive/home.tar.gz -C /
```

Imaging an entire disk:

```
dd if=/dev/hda | gzip --fast > /media/usbdrive/hda.img.gz
```

```
gzcat -dc /media/usbdrive/hda.img.gz | dd of=/dev/hda
```

... equivalent to the basic functions of Norton Ghost.

When you use this method, the filesystem you are copying must not be mounted! So it is best to do this from a rescue CD.

The dd method can also be used to rip a disk:

```
dd if=/dev/cdrom of=cdrom.raw
```

... copies the complete binary data off the CD, including any copy-protection measures.

Scripting

Bash scripts are the equivalent to DOS batch files. They contain a sequence of commands. You've already seen the ~/.bashrc – script files have a number of things in common.

Open a text editor and write the following into it

```
#!/bin/bash
echo "hello world"
exit
```

Save it as "hello" - no special extensions are needed. The #! at the top is pronounced "sha-bang" and the line right after it tells the computer where to find the shell to open. The rest of the file invokes commands like you'd normally just type them in.

To run the script you enter ./hello (or just hello if you created it in that ~/bin directory earlier.) Try it.

What happened?

You cannot accidentally execute any old file. You have to make it executable first.

```
sudo chmod +x hello
```

Now the file has changed color in the list, and you can execute it. Write a script that does what you did in one line earlier. You'll find it's more convenient to use three echo statements instead of one: helps get the layout right.

Another use of a script is to perform repetitive tasks like converting your entire mp3 collection to ogg/vorbis.

```
#!/bin/bash
for file in *
do
ffmpeg2theora $file
done
echo "all finished"
exit
```

To do this you need the ffmpeg2theora package.

```
sudo apt-get install ffmpeg2theora
```

A full bash scripting guide is available from the Linux Documentation Project

<http://tldp.org/LDP/abs/html/>

There are different script languages. The above is a bash script. `#!/bin/p1` is a perl script and `#!/bin/py` is a python script. And so on.

Scripting is one step away from programming... so, hang onto your hats:

Programming Basics

Much of linux is written in C/C++ using the GNU Compiler Collection (GCC). Some is written in C#, pascal, python, ruby, fortran, basic, even assembly and a very small amount directly in machine-code. Anything that works, and is compatible with the GNU GPL v2 is acceptable.

A typical programming course will start out teaching you Java and/or Pascal and then move to languages which are considered more appropriate for the tasks you are likely to face. I'm going to put you in close to the deep end and go right to C and GCC.

Feel the Source: GCC

One of the main advantages of GNU/Linux in any flavour is the ability to modify the programs you use. To understand GNU/Linux properly, you will need to understand about programming, even if you never do any actual programming yourself.

This is a very basic introduction only. A more complete course is available at http://www.physics.drexel.edu/courses/Comp_Phys/General/C_basics/

In this short version, I'll just cover the base concepts, then I'll walk you through writing and compiling your own program. Meanwhile I'll be tidying up some terminology you may have learned.

Basic Concepts:

Hackers are people who care about their programs and value the ability to rule the computer.

Users are people who just want do wordprocessing and surf the internet and couldn't care less about how it all goes, just so it goes. Students are encouraged to strive to be hackers.

Computer programs are written in plain text (that's *real* programs - "visual" programming is a toy for users) . The actual instructions are in a precisely defined variety of English, called a *programming language*, which the computer cannot understand. The computer understands *binary* (also called *machine code*). So a program called a *compiler* acts as an interpreter, turning the programming language into machine code.

The instructions written in a computer language is called *source code*, or *code* for short. A

program ready for the computer is called a *binary*, if it can be run it is called an *executable*. (Note, only users *run* programs, hackers *execute* them!) The text file containing the source code is called a *source file* or *the source* for short. The process of turning code into a binary is called *compiling*.

C++ and GCC

The most common programming language in GNU/Linux is called C++ (see plusplus), indeed the heart of Ubuntu, the Linux Kernel, and all of GNU, is mainly written in C++.

The compiler is called GCC (GNU C Compiler), which is actually one of the more powerful and versatile around. It understands a lot of different languages with the help of other programs like G++ (GNU C++ Compiler) and so on.

```
man gcc
```

You are not expected to be able to understand the man page right away. The best way to learn is by doing, so let's write a program.

Writing Program

The first step is to get into a working directory you are comfy with. If you used the tip earlier and created a bin directory which you added to your path, that's a good place.

```
cd ~/bin
```

If you didn't, that's OK, use your home directory instead. The next step is to create a text file to write the code into.

```
gedit hello.c
```

Type the following into the editor and save:

```
#include <stdio.h>

void main()
{
    printf("\nHello World\n\n");
}
```

Now you have a source file called *hello.c* in your working directory, you can compile it.

```
simon@lab01:~/bin$ gcc -o hello hello.c
hello.c: In function 'main':
hello.c:4: warning: return type of 'main' is not 'int'
simon@lab01:~/bin$
```

(The first line, from the \$ sign, is the command to enter.)

If you list your files, now, you'll see a new file called "hello". Check its properties and you'll see it is executable. Execute it by entering its name.

```
simon@lab01:~/bin$ hello
```

```
Hello World
```

```
simon@lab01:~/bin$
```

(If you are doing this in your home directory, you will have to enter `./hello` to execute the program.)

Congratulations: you've just become a programmer!

Anatomy of: `hello.c`

Source files have a naming convention which is partly tradition and mostly common sense.

hello.c is no accident. The program is called "hello" and it is written using only that part of the C++ language known as C. As you progress you'll learn the different parts. Any text file you see called myfile.c is c-code, myfile.cpp is c++ code. This one starts with:

```
#include<stdio.h>
```

In C/C++, all the special functions that tell the compiler what to tell the computer must be *defined* before they are used. Very common functions have definitions that are already written by other programmers and shared in special files called *libraries*.

The first line of the program, therefore, tells the compiler to include the definitions in a library called `stdio` (standard input/output) which deals with writing stuff to the standard output (normally a terminal window) and reading things from the standard input (normally a keyboard).

The triangle brackets <...> tell the compiler this is a standard library file. If it were non-standard, we'd put it in quotes and specify the full path. The `.h` in the name says this is a *header* file. Which is logical, as it contains text that goes at the top of a program.

```
void main()
```

All programs have to have at least one customized function, otherwise we cannot do anything *new*. In C/C++ that function is called **main()**. In this case, main doesn't end up storing anything, so we write `void` ahead of it, and it doesn't take any input, so we write empty brackets `()` after it. Writing a function name out like this is called *declaring* it.

```
{ (open curly brackets)
```

All the instructions that belong to a function are included in curly brackets after it is declared. The `}` at the end finishes the program. The curly brackets are given their own line when doing so makes things clearer. It doesn't have to be like this, for example:

```
int main(){main=1+1}
```

- is all on one line. And if you can get this working properly, you're a hacker already and there really isn't much more I can teach you that you cannot find out for yourself.

The main function in hello.c, however, contains one function called **printf**

```
printf("\nHello World\n\n");
```

- this function is already declared in the `stdio` library, so we can just use it here. The *argument* in brackets is enclosed in quotes to show that it is a *string* (a collection of characters that represent text.) Remove the quotes to see what happens.

They are not *curly* brackets because we are *using* the function, not defining it.

`\n` is a special character representing a newline. Why do we need two newlines at the end? Try removing one and see.

To see why we need them at all, see what happens when you use the code below as your `main()`,

```
printf{\n};  
printf{"Hello W"};  
printf{"orld"};  
printf{"\n\n"};
```

The semi-colon shows where the line ends.

That Compile Command

The compiler instruction follows the standard format for any commandline instruction

```
gcc -o hello hello.c
```

The program being run is called `gcc`, the input file is called `hello.c`, this much is plain. The `-o` is an option meaning "set the output filename to" whatever follows. In this case, we chose it to be "hello".

What happens if you leave off the `-o hello` part? Try it and see.

Sharp eyed readers will have noticed that there was a special message after the compile command.

```
hello.c: In function 'main':
hello.c:4: warning: return type of 'main' is not 'int'
```

This is GCC telling me that there is a mistake in the program. It tells me the name of the file and the function with the mistake. Then it gives me the line number (4) where the mistake was encountered. The mistake isn't serious, it is a **warning**. If it were bad, it would say **error** and stop compiling. In this case, the code will work, and GCC warns me because it may not do what I expect.

Then it tells me what it thinks the mistake could involve, to help me correct it. In this case it complains that `main()` is not an **integer**. We know this, we said it was a **void** in the program. So it is safe to continue.

See if you can figure out how to edit `hello.c` so it will compile without any warnings at all.

Exercise:

Write a program that produces the following output:

```
simon@Lab01:~/bin$ hello
*****
*** Hello World ***
*****
simon@Lab01:~$
```

There is much more to programming than that of course. But the basics are all there. Now you have a chance to understand what you are doing.

Installing From Source



Very occasionally there are no deb packages available. Perhaps what you want is too new? If you have the programming tools available, you may attempt to install from source code packages.

The source code comes in a tarball i.e. `vultures-2.1.0-full.tar.bz2` for the *vulture's claw* game from DarkArts. Extracting this file reveals a directory called `vultures-2.1.0` in the working directory. This is called the *top level* directory.

Inside the top level directory are the files which make the unique parts of the program.

The important files in a tarball like this are:

- `readme.lst`: read this, it tells you what else you need and where to get instructions
- `install.txt`: the instructions
- `configure`: an executable text file, usually a bash script, which does basic checks
- `makefile`: this is a list of instructions the computer uses to build the program
- `foo.c`: source code files have a `c` or `cpp` or `c++` extension.

The sequence of steps to compile a program are, usually:

1. `untar` the file and `cd` into the top level directory
2. `./configure` (check the output for errors)
3. `make` (compiles the source code and builds the application.)
4. `sudo make install` (puts all the binaries in the right places)

There are no end of complications to all this. Which is why we use package managers.

The compile command is now **make**, notice? Actually, the `make` utility automates a great deal of the process of compiling large programs. Typically, lots of source files need to be compiled, then combined to make the program or application. The overall process is called *building*. The

make utility calls *gcc* to do the actual compiling part.

Other Languages:

There is native GNU/Linux support for any foss programming language or script. Popular languages include python, perl, pascal, fortran, free-basic, ruby, the list is endless. Many of these are designed for novice programmers and there are online tutorials to help.

Creating Proper Documents

The text-only environment of the CLI is, at first glance, extremely limited when it comes to representing your message. What happens if you want to use italics?

This is where markup language comes in.

The markup languages to explore here are: hypertext (HTML), extended (XML) and LaTeX.

All markup languages use tags to denote the way a particular bit of text is supposed to look. They all rely on a special application to view the document correctly. They all have international standards associated with them so that everyone can agree what the tags mean.

HTML

This is the first language of the internet. The idea is to make sharing ideas easy. An html document is called a web page, and they are viewed in a web browser. Here's hello.html

```
<html>
<head>
<title>Greetings</title>

<meta name="author" contents="Me" />
</head>
<body>
<p><large>Hello World</large></p>
</body>
</html>
```



The authority in html (and xhtml) is the World Wide Web Consortium (W3C).

<http://www.w3schools.com/html/default.asp>

<http://www.w3schools.com/xhtml/default.asp>

XML

This is fast becoming the standard markup language for offline productivity documents. It is very much like html. In fact, open formats are little more than web pages in an archive. Rename any odt file as .zip, open it and see. Here's the content of hello.xml

```
- <note>
<to>World</to>
<from>Me</from>
<heading>Greetings</heading>
<body>Hello World</body>
</note>
```

XML is viewed in a web browser, or a productivity suite like OOo. hello.xml describes what each bit of text means. How it is presented depends on another file called a Cascading Style Sheet (CSS). The css file is usually supplied by the application, or in the odt archive.

<http://www.w3schools.com/xml/default.asp>

LaTeX

(Lay-tek) This is a publishing standard much used in the scientific community. It is based on TeX, which dates back to the time before everything went graphical. html and xml are based on this. Here's the content of hello.tex

```
\documentclass [10pt, a4paper, draft] {article}

\begin{document}
\title{Greetings}
\author{Me}
\date{\today}

\maketitle

\Huge Hello World

\end{document}
```

The image shows the word "LATEX" in a large, black, serif font. The letters are spaced out, and the 'T' and 'E' are particularly prominent.

This document must be interpreted with the LaTeX program, which turns it into device independent format. DVI format can be turned into anything else, like post-script (readable by printers) and pdf.

<http://www.latex-project.org/>

LaTeX is not installed by default:
`sudo apt-get install latex`

How to Get Help

Commercial OSs (SLED, RHEL, Xandros, Linspire etc.) have commercial support which you have paid for. It follows that if you have a commercial distro, the support company is your first port of call. These companies usually maintain a website with a FAQ list and lots of resources.

The same applies to proprietary OSs. Apply to the IP owner first. Especially with closed source software because they are the ones with the access you need. Nobody else can help, unless your issue is with the user interface or some such.

Free and Open Source software is supported, in the first instance, by "the community". That's you :) so the use of Free Software implies a moral obligation to act as a member of the community.

For GNU/Linux, the local manifestation of this community is the GNU/Linux User Groups or LUGs. These are usually informal computer clubs with different kinds of organisation depending on the members.

- NZLUG exists as a web site and a mailing list. You are encouraged to subscribe to this list.
- AuckLUG is some guys who organise monthly meetings with guest speakers.
- UALUG facilitates GNU/Linux at Auckland University, but does no advocacy.
- HBCLUG is a fledgling LUG servicing the Hibiscus Coast – and anyone who wants to help out.



HBCLUG has a web page
www.hbclinux.net.nz/hbclug.html

Major events are:
Installfest – each term, associated with the ACE course.

Software Freedom Day – annually

And there's the course. No regular meetings are planned and web hosting is paid for by HBCLinux support fees.

Membership entails no obligations and no fees. It helps to have a membership list – so I can say how many GNU/Linux users the LUG can claim to represent. If you want to contribute to events, let me know.

The LUG is the local community, after that it gets global.

GNU/Linux is very much a web-based phenomenon. There are a lot of assistance oriented pages, particularly for newcomers. Google provides a linux search engine. IBM supplies a range of articles on migration to linux aimed at businesses. Free distro maintainers keep forums and wiki pages.

The forums tend to act as a hub for all this. Experienced forum members know where to look for the right information and are willing to pass that information on. The understanding is that you will help others in your time. Since there is always someone more ignorant than you, your time will come sooner than you think.

The largest (over a million active members) and more useful international forums is Linux Questions (LQ). You are encouraged to sign up.

Some points to remember:

- Edit your profile – it needs to contain, at least, your distro and location.
- Post an introduction asap – it's only polite.

When asking a question:

- Pick the forum carefully – if in doubt, pick newbie.
- Put your question in your title
- Expand on the question in the message body
- Include error messages verbatim
- Copy over the commands you used as well as the result
- If you refer to a web site, include the url.
- If you refer to a package, give the package name.
- State what you have attempted to solve the problem.

Be prepared to wait for a reply – NZ gets the new day ahead of everybody else. The first replies will almost always be requests for more information. Try to supply the information as well as you can.