

Introduction to Linux

- Where does Linux come from?
- Why do we use it?
- Familiar applications: browsing, office applications and mail
- Further Linux: using the command line, text editors and writing shell scripts

Introduction to Linux

Michaelmas 2007

These notes and exercises originate from a series of short courses given by Dr Jon Lockley from Comlab, University of Oxford and Susan Hutchinson, Department of Statistics, University of Oxford.

They are intended as an introduction to Linux for Windows users. There are two sections: an Introduction to Linux which covers familiar applications such as browsing and word processing, and Further Linux which introduces less familiar concepts such as the command line, text editors and shell scripts.

Where appropriate URLs are provided at the end of each page. These link to sites which provide further information or useful resources.

Where does Linux come from?

- Loosely based on the Unix operating system.
- Unix was developed in the late 1960s at Bell Labs.
- Unix was the first operating system designed to run on many different types of computer.
- Unix philosophy: *programs should do one thing and do that well.*
- In 1991 Linus Torvalds started work on a version of Minix, a micro kernel Unix used for teaching

Introduction to Linux

Michaelmas 2007

The Unix operating system was designed as an operating system for technical users in the late 1960s. In 1975 the first widely available version of Unix was issued. Before the development of Unix each computer manufacturer had their own proprietary operating system. Unix, however, was designed to run on many different platforms. During the following decades Unix became increasingly popular, particularly in Universities as an environment for program development.

Linux is a fairly recent development of Unix. Linus Torvalds began work on Linux, a freely distributable version of Unix in 1991. However without the GNU tools created by the Free Software Foundation Linux would not be as successful as it is today. Many prefer to call it GNU/Linux.

<http://www.bell-labs.com/history/unix/>

<http://www.linux.org/>

<http://www.gnu.org/>

Why do we use Linux?

- Open Source software: transparency, freedom
- Flexible: choice, tailor your environment to suit
- Powerful: a good programming environment

There are many reasons why people choose to use Linux.

There is certainly a case that Open Source software – that is software developed by a group of committed developers – is more robust and reliable. Software that is unreliable stops being used: only the fittest applications survive.

Linux – and Unix in general – provides a very flexible programming environment. In particular, bioinformaticians and mathematical geneticists who need to analyse genetic and genomic data using models of evolution often use Open Source applications such as perl. Powerful compute clusters usually run some Unix based operating system.

The Linux Desktop

Linux gives you choice. Commonly used desktop environments are:

- Gnome
- KDE

There are others including fvwm and Afterstep.

Linux comes with a choice of desktop environments. A desktop environment gives an integrated and coherent desktop to the user. Windows 2000 and Windows XP are Microsoft desktop environments. The most commonly used are Gnome (version 2.6) and KDE (version 3.2.2-6).

We will look at the Gnome desktop in more detail. If you want to use a different desktop this can be changed at log in. Choose **Menu Session Type** from the **Welcome to Linux on machine** login window and select the desktop environment you want to use.

<http://www.gnome.org/>

<http://www.kde.org/>

Familiar applications

- Browsers:
Firefox, Konqueror
- email readers:
pine, Thunderbird, webmail
- Office software
Open Office, abiword



Browsing, reading email, and reading and writing documents are familiar applications for most users. In Linux there is often a choice of package so part of learning about Linux is finding which application best suits your needs.

For example there are at least two browsers available: Firefox – an upgraded version of Mozilla, which in turn was based on Netscape – and Konqueror which is the KDE browser (also available in Gnome).

There will be more details about setting up an email reader later. Thunderbird will usually meet most people's needs.

Open Office provides similar functionality to Microsoft Office products. These slides were prepared with the equivalent of Powerpoint.

Gnome Desktop

The standard desktop contains

- Icons
- Background

The Panel contains

- Start Menu
- Application Icons
- Workspace Switcher



As you can see the standard desktop has icons and a strip at the bottom of the screen called the panel. In Linux icons are opened with a single click, rather than a double click. New users to Linux often find that they have created two of everything!

The panel contains quick starts for various applications and the workspace switcher.

Click on the red hat at the left hand corner of the panel for the start menu.

Window Operations

- Maximise – make full screen size
- Minimise – close and save on the panel
- Restore – open a minimised window from the panel
- Close – exit from the window
- Resize – change the size of a window
- Display options – show the window menu



Use the symbols at the top right of any window to minimise and maximise windows.

Left click on a minimised window on the panel to restore it.

Use the X in the top right hand corner of a window to exit that application.

Resize the window by moving the mouse to a corner of the window, then wait until the pointer changes to a double headed arrow and then drag the pointer while holding down the left mouse button until the window is the size you want.

If you right click on the blue bar at the top of the window you will see a list of window operations.

The Start Menu

Start Menu: starts applications such as

- the emacs editor
 - Open Office software
 - a terminal window
- and many, many more



Applications can be started either from the Start Menu or from the command line. We will be looking at the command line later.

Often when people start using Linux they begin by using the Start Menu and then move onto the command line later.

You will often find that there are many different ways to start an application. The exercises will test this.

One useful application that is found from the Start Menu is found at **Preferences -> Control Center**. This application allows you to change how the Gnome desktop looks.

The Panel and Workspaces

Multiple Workspaces have many useful features:

- Organise work
- Reduce clutter

The workspace switcher grid allows you to move between workspaces.

There are many ways to move between workspaces and move windows between workspaces.



One very useful feature of both Gnome and KDE is the ability to configure more than one workspace. This allows you to organise your work and reduce desktop clutter. You can configure as many workspaces as you like but four is usually a good number to start with.

Configuring your desktop environment

The Control Center allows you to configure your desktop.

- Start Menu -> Preferences -> Control Center
- Change the background
- Experiment (briefly) with themes
- Set up keyboard shortcuts



One very useful feature that can be configured is keyboard shortcuts which allow you to toggle between workspaces.

Web browsing

Find a browser either on the panel or from the Start Menu.

- Configure the home page
- Subscribe to a Usenet newsgroup
- Remove unwanted links
- Add some useful links



The exercises suggest various configuration changes that can be done for your browser.

Reading email

There are several different email readers.

- Thunderbird
- Pine: a simple command line email reader
- Webmail
- Many others: mutt, Evolution and Kmail



We can look at configuring mozilla to read email during the practical session.

Office Software

Used for compatibility with Microsoft Office products.

- Microsoft Word **oowriter**
- Microsoft Excel **oocalc**
- Microsoft Powerpoint **ooimpress**
- Microsoft Paint **oodraw**

All can be run from the Start Menu or from the Panel or from the command line.



These applications can be used to read Microsoft Office products. You will be finding out more about them later in today.

Further Linux

End of basic Linux where we've covered

- The desktop
- Browsers and email
- Office software

Further Linux will introduce

- Command line
- Text editors
- Shell scripts



So far we have looked at the similarities between Windows and Linux. We will now begin to look at ways in which Linux and Windows differ. First we will look at the Command Line.

The aim of this sessions is to create confident Linux users. Much of the information covered could be considered a general introduction to Unix and so applicable to other Unix systems such as Solaris.

Unix/Linux Commands

- Each program should do one thing well.
- If a new requirement arises start again – don't add features to an existing command.

Advantages and disadvantages with this approach.



Most of the commands available on early Unix systems are still used today, as well, of course, as many new ones that have since been added.

Unix commands tend to be small and have a single function. This means that there are a lot of commands to remember. Another side effect of the long history is that there is generally more than one way to achieve the end you want. Often there is no correct solution although some are neater than others.

It might also seem that giving commands a deliberately obscure name is part of the philosophy. I'm not so sure about that - generally there is an explanation for the name of a command although it isn't always immediately apparent. Believe it or not there is a sensible explanation for why the commands **cat**, **grep** and **awk** are so called!

The Command Line

- A graphical user interface (GUI) is available in both Windows and Linux.
- The command line is often unfamiliar to Windows users.
- Windows and Linux – reading a PDF file or starting a browser.



We have now spent some time looking at Fedora's Graphical User Interface. We will now focus on the command line. This can seem an unfamiliar place for Windows users.

For example when reading a PDF file in Windows you will either click on the picture or open the application from the start menu and browse to the file. In Linux you are just as likely to enter

acroread *filename*

on the command line. Similarly the mozilla browser can be started by entering

firefox

How commands work

- A command at its simplest is of the form:
command
- Often commands use an argument like a file name
command *file*
- Some examples are

date [to display the time and date]

emacs newfile [start **emacs** to change the file
newfile]



This is the format of many commands.

It is also possible to modify the behaviour of a command using options. So for example the command **ls** on its own will display the name of all the files in a directory.

ls

By including options you can display details about the size, last changed date and access permissions of a file and many many other details. Options are specified by a hyphen and a single character so

ls -l

would give a long listing. This is something of a simplification but will do for now.

Finally ... some commands

Orientation and navigation or where am I and where do I want to go?

- **pwd** [print working directory]
- **cd** *directory* [change directory]
- **cd** [go to your home directory]
- **cd ..** [go up one directory]



You can explore the directory hierarchy using

cd ..

to move up to the next level and

cd

to return to your home directory. Your home directory is where files you create are stored and where you start each new session. Your home directory will be something like

`/export/rs/username`

or

`/export/home/student/username`

... and more ...

File and directory manipulation

- **ls** list files in a directory
 - **ls -a** list all files in a directory
 - **ls -l** display a long listing
 - **ls -la** display a long listing of all files
- **mkdir** *directory* create a directory
- **rmdir** *directory* delete a directory
- **cp** *file1 file2* copy file1 to file2
- **rm** *file* delete a file
- **mv** *file1 file2* move file1 to file2



These are standard file and directory manipulation commands. When you use

ls -a

you will see some files that you haven't seen before. These files are known as hidden files and are there for set up and customisation purposes. You might see a `.emacs` file which contains your own emacs editor settings and a `.bashrc` file which includes shell customisation.

Note that a single dot on its own means the current directory the one you are in at the moment so

ls .

and

ls

produce the same output.

IMPORTANT: the command **rm** really DOES delete files. Unlike Windows where deleting files merely moves them to the Recycle Bin in Linux files that are deleted go and cannot be retrieved unless there is a backup.

... and more

There are several ways of looking at the contents of a file

- **cat** *file* look at the contents of 1 or more files
- **more** *file* look at the contents of a file and prompt at the end of each screenful.
- **less** *file* like **more** only more powerful.



The **cat** command is said to be named after the verb concatenate as it can also be used to concatenate several files into one. The command displays the contents of a file on the screen. It is only really useful for small files.

more displays a file, pausing at the end of each screenful.

<space> moves to the next page

q quits at any time

<return> moves on a line

less has all the above options and several more. One useful one is

G move to the end of the file

Other commonly used commands

- **man** get help with a command
- **whatis** short description of a command
- **whereis** locate a command's binary, source and man page
- **lp** print a file

Some rather more powerful commands

- **find** find files matching a pattern
- **grep** search for a pattern in a file
- **wc** report the number of characters/words/lines in a file

How to find all the files modified in the last 5 days containing the word 'statistics'

```
find . -mtime -5 -exec grep -i statistics {} \; -ls
```



A digression on paths

There are three sorts of paths:

- simple – a file or directory name
newfile
- relative – a reference to a file or directory from the current directory
../tmp/newfile
- absolute – a reference that will work anywhere
/export/support/hutchins/tmp/newfile



The **man** command is particularly useful for finding out all the available options in a file. So

man ls

will give you a vast amount of information on **ls** including the options. If you are not sure of the name of a command but know what it does then you can do a keyword search with the **apropos** command. If you can't remember what a command does then use

whatis ls

The commands **find** and **grep** are particularly useful. **find** can search down through directories looking for files matching a given condition such as all files changed within the last 7 days. **grep** will search a given file for a particular pattern. The two commands are often used in conjunction like this

```
find . -mtime -7 -exec grep -i hello {} \; -ls
```

It has to be said that the syntax is obscure but this command would search all files changed in the last 7 days containing the string "hello".

Linux like Windows organises files in a hierarchical directory structure. There are some minor differences: the directory separator in Linux is / whereas in Windows it is \.

Files can be reference in three ways. So if there is a file called **myfile** in your home directory

```
/export/home/student/username
```

you can use more to view it in three different ways:

```
more myfile
```

```
more ../myfile
```

```
more /export/home/student/username/myfile
```

Making use of the shell

- What is a shell?
- How can we use it more efficiently?
- filename and command completion
- use the arrow keys to recall and alter previous commands



The shell is the command that interprets the commands we enter. As you might expect there are many different shells. We will be using the bash shell.

There are many shortcuts that allow you to use the command line more efficiently. For example, the tab key can be used to complete both commands and filenames. If you enter the command

ac

and then press <tab> once you should hear a beep. That's because there is more than one command beginning with **ac**. Pressing <tab> twice in quick succession should display a list of commands beginning with **ac**. If you then enter the letter **r** so that you have **acr** and press <tab> again the command **acoread** should appear.

Wild cards and globbing

This slide could be called file name expansion! The most commonly used special characters to match parts of a file name are:

- ***** matches none or more characters
- **?** matches a single character
- **[]** matches any characters in a given range
- **[!]** matches any characters *not* in a given range

To list all files whose name ends with **.txt** use **ls *.txt**



The use of wildcards is another way that the shell makes specifying filenames more efficient. So

ls *.txt

will match all files ending in **.txt**

ls ?.txt

will match all files with a single letter before the **.txt**.

ls [a-z].txt

will match all files of the form **a.txt, b.txt, ... z.txt**. Note, too that wildcards can be used together so

ls [a-z]*.txt

will match all files beginning with **a-z** and ending with **.txt** containing any characters in between.

Redirection

- Redirection allows you to change where the input and output from a command goes.
- Instead of reading from the keyboard input can come from a file.
- Output can be saved in a file rather than displayed on the screen.

```
ls -l /usr/bin > output
```



The > (greater than) symbol is used to redirect output from the screen to a file and the < (less than) symbol is used to read information from a file.

By convention Unix programs have a single way of accepting input called *standard input*, a single way of displaying output called *standard output*, and a single way of displaying errors called *standard error*. The standard input is keyboard, the standard output and error are the terminal.

This may seem a little technical but results in several useful features. We can easily change where a program sends its output and reads its data.

Pipes

The majority of Unix commands are technically known as filters.

A pipe allows you to use the output from one command as the input to the next.

```
ls -l /usr/bin | more
```



Another useful side effect of programs using standard input and output is that commands can be linked together with the output of one command being used as input to another.

In the above example we have used a pipe to send the output from **ls -l /usr/bin** to the **more** command. This allows us to read a screenful at a time rather than have the output scroll off the top of the page.

It is also possible to use several pipes together.

Although we have not covered all these commands one way to find the find largest files in a directory would be

```
du -sk /usr/bin/* | sort -n | tail -5
```

Other differences

- Case sensitivity
- Guidelines for file names
 - spaces
 - hyphens
 - / and \



One significant difference between Windows and Linux is that Linux is case sensitive. This means that a files called

big

Big

BIG

are all different files. In Windows this is not so. This difference is important if you are sharing information between Windows and Linux.

As we have seen Linux commands use space as a separator between commands and files. This means that spaces in file names can cause problems.

How would you read a file called **My File**?

Similar hyphens (-), slashes (/) and backslashes (\) are best avoided in filenames.

Text Editors

Why do we need a text editor?

- Typing in program code
- Changing configuration files
- LaTeX



Text editors are one of the most important applications in the Unix world. They are used in many different contexts.

Text editing underlies nearly every activity on a Linux system. For example, a text editor is often used to customize your environment to suit your needs. When you start to manage your own Linux machine many configuration changes will be done with a text editor. In some cases only a text editor will be available; there will be no friendly graphical user interface (GUI) to help you.

Today we will look at two different program editors. These are vi and emacs.

But first a small digression...

What is a file?

Windows

- includes formatting and application information
- used only by one application

Linux

- usually plain text
- used by many applications



vi

- Where did it come from?
- What does it look like?
- What is it like to use?



A file in Windows is usually (but not always) closely linked to an application. So a .doc file is considered to be a Word document, a .xls file an Excel spreadsheet and so on. Although we cannot see it, a lot of information about the file and the application that runs it is included in the file.

In Linux, files generally contain only plain text. When you look at the contents of the file you see all there is. Files are not tied so closely to one application: a file can be viewed, changed and compiled or used by several different programs.

Both these editors are standard in all Linux systems.

The choice of editor can arouse strong feelings in some of the Unix community. Both the editors we will be looking at today have fierce advocates and sometimes the strength of their views can disconcert newcomers.

Personally I use both editors and find that each performs a particular set of tasks best. I find vi especially useful when doing system administration: changing configuration files, for example. I use emacs when writing longer documents such course handouts and notes or changing HTML (web page) files.

GNU emacs

- Simple interface
- Powerful – many configuration options
- Too many choices?



emacs (properly GNU emacs) is a very powerful editor. What does powerful mean here? The editor has many features and indeed is sometimes referred to as an integrated environment rather than just an editor. As well as editing files it is possible to read and send email and compile and run programs from within emacs. It is also highly customizable. This means that you can configure most settings to suit yourself.

emacs is initially easier to learn than vi but its size and complexity can mean it is very possible to be overwhelmed by choice.

The name emacs originates from the acronym **Editor MACroS**.

Shell scripting

- What is a shell script?
- Why write shell scripts?
- What are shell scripts used for?
- When should shell scripts not be used?



The scripts that we will develop are small and simple. However the techniques described here should give enough detail to allow you to go on to develop more powerful scripts of your own.

What is a shell script?

- A file containing commands written in a shell like bash (**B**ourne **A**gain **S**hell)
- Executed like a command
- Interpreted not compiled



A very simple script

The file **hello** contains this text:

```
#!/bin/bash  
echo "Hello world"
```

and the contents are executed using

```
./hello
```

producing the output

```
Hello world
```



There are several different shells. The original shell – the Bourne shell – is a small shell that lacks many useful command line features such as a mechanism for recalling previous commands. It is still used as a scripting language because it is the most widely used and portable shell.

Another popular shell is the csh shell. This was popular 10 or more years ago and is still used as a command interpreter, but is not considered robust enough for shell programming. It has too many restrictions, particularly in the area of file manipulation.

We will be using the bash shell. It is a flexible and powerful shell but perhaps not quite as portable as the original Bourne shell.

The idea of a script is very simple. A plain file contains a series of commands that are executed sequentially unless otherwise directed.

In the above example the first line, which begins with a #, describes which shell is to be used to interpret the commands. The next line contains the only command to be executed by this script: that is to display on the screen the text contained between the open and close " (quotes). So in the example above `Hello world` would appear on the screen.

Why write shell scripts?

- Quick to write
- Tailored to a specific task
- Automate routinely performed actions



The idea of a script is very simple. A plain file contains a series of commands that are executed sequentially unless otherwise directed.

In the above example the first line, which begins with a #, describes which shell is to be used to interpret the commands. The next line contains the only command to be executed by this script: that is to display on the screen the text contained between the open and close " (quotes). So in the example above `Hello world` would appear on the screen.

What are shell scripts used for?

- File manipulation
- System administration tasks
- Data extraction
- Lots of others



I write shell scripts very often. I use them for

- monitoring user's disk usage – I can find out whether someone is over quota and warn them;
- monitoring system disk usage – every night I check that no disks are getting full unexpectedly;
- system tasks such as running backups;
- manipulating user files and data.

In Statistics researchers often have large files of genome sequence data. Extracting information based on particular fields or matching particular patterns and storing this information in a different set of files is straightforward.

When should scripts *not* be used?

- Large – need something more powerful
- Number crunching
- You need structure – functions and subroutines
- Mission critical – use programs
- Need to work on many systems – not always portable



We have already seen that shell scripts can be seen as a quick and dirty solution. With this in mind any problem that requires a carefully designed solution is not a good candidate for a shell script.

Any application that needs to be fast or involves number crunching or large scale data manipulation should not be solved by a shell script. A program is needed.

Any application that needs a structured solution involving subroutines and functions is too big a problem for a shell script.

A technical term

Command line arguments

more longfile

A shell script example

./hello Homer

Homer in this case is a command line argument



We will be using a couple of technical terms today.

The phrase command line argument has cropped up during the previous sessions and will be used frequently during the exercises. It simply refers to the information you give a command to determine what it does. So if you want to display the contents of the file **longfile** on the screen this is the command line argument you should give the command **more**. You enter

more longfile

We will be using command line arguments extensively in the exercises.

The Exercises

The exercises need to be done while using a Linux PC. If your machine is not configured to run Linux then please mail

ithelp@stats.ox.ac.uk .

Introduction to Linux



Michaelmas 2007

When you have completed the shell scripting part of the exercises you should be able to write a shell script using:

- command line arguments
- simple input and output
- control flow constructs

```
if then else  
for done  
while done
```

Please let me know if you have found these notes useful. Any suggestions for improvement would be gratefully received.