University of Manchester
School of Computer Science

Introductory Unix Labs

Note: the work contained in this document is intended to occupy you for 2 separate sessions of approximately 5 hours together. Please do not rush through the work.

These labs are to introduce you to the Unix operating system and X window system which run on many PCs in the School; some of the information contained in this document is specific to the setup here in the CS school, but most applies to any Unix system. They can only provide a very brief introduction to several aspects of what is a very rich computing environment, but they do point you toward other sources of information, which we hope you will use as a basis for your own extensive independent exploration. The computing environment described here is one in which you will be spending a lot of time over the next few years, so take a good look around and make yourself at home.

If you find any mistakes in this document or have any suggestions for improvements or additions, please contact `dirk.koch@manchester.ac.uk` .

Will you please make sure your work is signed off with a demonstrator for each of the sessions so that we know what you have completed.

January 18th, 2018

Session 1: Introducing Unix

You have been allocated 1-2 hours to complete each of the sessions in this document. If you were to rush, you could do all of them in about an hour! But that would be pointless. Although the tasks you are actually doing are trivial (in the sense you are just following a script), unless you take the time to think about them, you will not be able to do similar things when the real laboratories start. In order to be ready for them, you need to practise, and make your stupid mistakes now – when it doesn't really matter.

So please take your time, and spend at least one hour on each session. If there's anything you feel you don't understand, please don't be shy: stick up your hand and ask us! Don't forget to tell us when you've finished each one, or else we'll think you're stuck and we'll waste time chasing you.

1.1 What is Unix anyway?

An **operating system** (OS) is a suite of software that makes computer hardware usable; it makes the 'raw computing power' of the hardware available to the user. You're probably most familiar with the **Microsoft Windows** and Apple **OS X** families of operating systems for 'desktop' computers, and **iOS** (Apple, again) and Google's **Android** for mobile devices; but many other more specialist operating systems exist, and you'll be studying some of these and the principles that underpin OS design in COMP25111 in your second year. In the meantime, a potted history of OS development will tide us over...

In the late 1950s, an American company called **Bell Laboratories** decided that they needed a system to improve the way they worked with their computer hardware (it's probably quite hard to imagine what interacting with a computer *without* an operating system might be; but it wasn't pretty and involved manually loading and running programs one by one). Together with the **General Electric Company** and the **Massachusetts Institute of Technology**, they set about the design of an operating system they called **Multics**: the 'Multiplexed Information and Computing Service'. Multics was hugely innovative, and introduced many concepts that we now take for granted in modern operating systems such as the ability for more than one program to run 'at once'; but it did rather suffer from 'design by committee', and the final system was seen at the time as being overly complex and rather bloated ('bloated' is all a matter of perspective of course: its sobering to realise though that the entire Multics operating system was only around 135Kb. Today's operating systems are something like 30,000 times this size...). In the late 1960s, a group of programmers at Bell Labs created a cut-down, leaner and cleaner version of Multics that would work on more modest hardware. Legend has it that this was to allow them to play their favourite (only!) computer game, **Space Travel**. In an early example of the trend of giving things 'punny' names, to contrast with the more clumsy Multics, they called this new system Unix. The so-called **Jargon File** is a good source of explanations of various bits of computer slang and their obscure origins, and is well worth a read: in part to give some

background history, but mostly as an insight into the minds of the computing pioneers of the past!

Even though Unix is now quite old, most Computer Scientists recognise that the designers of Unix got most of the fundamental concepts and architecture right. Given how much computing has changed since the 1960s, this was an astonishing intellectual achievement. Although Microsoft's **Windows** is by far the most common operating system on *desktop* machines, the majority of the Internet, much of the world's corporate infrastructure, virtually all supercomputers, and even some mobile devices are powered by Unix-like operating systems. So, while the polished graphical user interfaces of Windows and **OS X** appear to dominate the world of computing, most of the real hard-core and leading-edge computation relies on an elegant operating system designed nearly 50 years ago (by a team of scientists who wanted to play a game).

The history of Unix is complex and convoluted, with the system being updated, re-implemented, and mimicked repeatedly over the years, primarily by commercial companies who guarded their versions jealously. Figure 1.1 shows a tiny fragment of the Unix's 'family tree' (the full diagram is at least X times the size of the portion you can see here). Although many of the branches represent interesting innovations of one kind or another, there are perhaps two that deserve particular attention. The first of these was the decision by Apple some time around the turn of the millenium to drop their own—highly popular, but aging—bespoke operating system (unimaginatively called **Mac OS 9**) in favour of a Unix-based system (now the more familiar 'OS X', where 'X' is both the Roman numeral '10' and a nod in the direction of the uniX nature of the OS). Although the majority of Mac users are blissfully unware of the fact, behind the slick front-end of OS X, sits a variant of Unix. The second, and perhaps more profound of these events was the creation in 1991 by Finnish programmer **Linus Torvalds** of a Unix-like system, the source code to which *he gave away for free*[1]; this became known as the **Linux Kernel**. Combined with other free software created by the **Free Software Foundation**, a non-commercial version of Unix called **GNU/Linux** was born (GNU here is a recursive acronym for "GNU's not Unix", a swipe at other commercial non-Free versions; much to the annoyance of the Free Software Foundation, GNU/Linux is almost always called just 'Linux'[2].)

Linux has been, and continues to be, developed cooperatively by thousands of programmers across the world contributing their effort largely free of charge. It is amazing to think that such a project could ever happen – and it is surely a testament to the better side of Human nature. But what is interesting is the observation that these programmers are not motivated by commercial concerns, but by the desire to make good reliable software and have it used by lots of people. Thus, Linux is a good choice of Unix: it's Free, it's efficient, it's reliable, and it is now used by large corporations, governments, research labs and individuals around the world. Even Google's **Android** platform is a Linux-based mobile OS, and the **Amazon Kindle** is also a Linux box behind its user interface.

One of the results of the fact that Linux is Free is that several organisations and companies have created their own distributions of it; these vary a bit (in fact, anybody is free to make any change they like to Linux, and pass it on to whoever wants it). The distribution we use in this School is **Scientific Linux**, a clone of **Red Hat Enterprise**, which is the most widely used distribution of Linux in industry.

So, if you are to become an expert computer professional, it is important that you understand

---

1. 'free' here in the sense both of 'freedom to reuse or adapt', and also in the sense of 'without charge'.
2. Linux is pronounced "Linn-ucks", despite the fact the name was coined by its creator, and his name 'Linus' is pronounced "Leen-uss"!

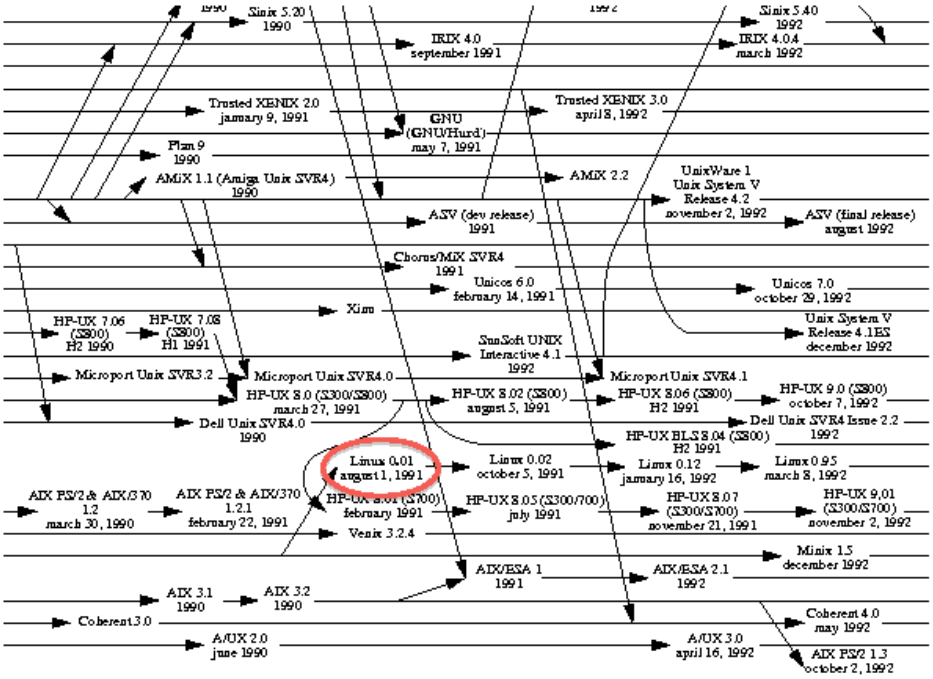


Figure 1.1: A fragment of Éric Lévénez's Unix History chart, reproduced with permission and showing the beginnings of Linux in amongst other versions of Unix.

the theory and practice of Unix based systems. Learning Unix is not only a crucial skill for any serious computer scientist, it is a very rewarding experience.

1.2 School Unix network

The School has well over 1000 PCs running various versions of the Unix operating system, nearly 300 of which are used for undergraduate teaching; most of these are 'dual-boot' machines which are also capable of running Windows, The machines are all connected by a network, so that you can sit at any machine and it 'knows who you are' – more about this later. This introduction is primarily aimed at the use of the Linux version of Unix on PCs.

1.3 User Names

To use a PC, you first have to identify yourself by typing your user name and password.

These are the same as your campus username and password that you use for Windows machines.

1.4 Reading this document

In this document, things that *you* have to type appear in a typewriter font (`like this`), to distinguish them from prompts from the machine, which appear in a bold sans serif font (**like this**). They won't actually appear like this on your screen. Remember to press ⏎ , called the Return key, after each command.

Things that appear between angle brackets (`<like this>`) are *descriptions* of things to type rather than the actual characters you type. Hopefully this will all become clear as we go on.

Also, by the way, watch out for getting mixed up between a lower case el, **l**, and a digit one, **1** – unfortunately these two characters look very similar in many fonts!

1.5 Logging on

The machine should be displaying the login prompt, which will look something like

E-C07LF16001 login:

If the machine you are using is dual-boot and is running Windows, follow the instructions displayed on the lab wall to reboot into Linux.

(**E-C07LF16001** is just the name of the machine displaying the prompt, yours will probably be different).

Note: If you use the numeric keypad to the right hand side of the keyboard during the following sections, you must use the 'num lock' facility to input numbers.

Now type your user name, e.g.

```
mXXXXXXX ←┘
```

Whether you typed your user name correctly or not, the machine will now ask for your password

Password:

You now have to type your password for you campus username. Note that when you type your password, *nothing* will appear on the screen, not even `*`'s. This is to prevent anyone looking over your shoulder seeing your password, or even how long it is. If you make a mistake either in the user name or the password then you must type both your user name and your password over again. If your password seems not to work after several tries, ask for help.

When you have logged in, you will get a prompt, which looks something like

```
[mXXXXXXX@E-C07LF16001 ~]$
```

except you will see your username before the @ and the name of the machine you are using after it.

Carefully type the command

```
/opt/scripts/copybashdotfiles
```

and press Enter. This will copy into your filestore various 'dotfiles' to make your account easier to use.

Now log out, by typing

```
<Ctrl>d
```

that is, hold the 'Control' (or 'Ctrl') key down and press 'd', then release 'Control'.

Now log back in again (but don't type anything else yet).

When you have successfully logged in, the machine responds with assorted information, and then prompts you with something like:

Manchester University, School of Computer Science
Linux Desktop Switcher

Your desktop is currently set for: AnotherLevelUp Fvwm95

Please choose your desktop from the list below:

<RETURN> Continue to use AnotherLevelUp Fvwm95
1 Use AnotherLevelUp Fvwm95
2 Use GNOME
3 Use KDE
4 Usx After Step
5 Use Window Maker
6 Use AnotherLevelUp Lesstif (Mwm)
7 Use Enlightenment
8 Use FVWM (vanilla)
9 Use twm

C Console login
I Start X Interactively (pick from available screens or layouts)
R Start X With window manager on a Remote host
Enter choice:

This is the list of **Window Manager**s you can choose to use under **X-windows**. For the moment, we don't want to go into X-windows, so type

`c` ↵

to choose the simple **console** interface (an upper or lower case `C` will do here). The machine now gives you a prompt, which will be something like

mXXXXXXX-)

to show that it is waiting for you to type a command.

1.6 Commands

Typing a command causes a program to run which does what you told it to do (not necessarily what you meant it to do!). For instance, `date` is a program which displays the current date and time. Type

`date` ↵

to see the date in a default format. Now Type

`date -d +1day` ↵

to see what the date and time will be tomorrow. Type

`date +%d/%m/%y` ↵

to get just today's date in the normal format.

fortune is a classic Unix 'game', that simply picks from its huge database, a phrase typical of those found in 'fortune cookies', and displays it for you. Now try typing

```
fortune ←┘
```

A useful thing to remember is that typing

```
<Up Arrow> ←┘
```

repeats the last command (don't forget that `<Up Arrow>` is a description of what you type – if you have just typed a left angled bracket you are not concentrating: go back to the start!)

Do this a few times to get a selection of 'fortunes'. Now type `<Up Arrow>` several times without pressing ←┘ until you see `date -d +1day` again: each time you key `<Up Arrow>` you are stepping back through your **command history**, one command at a time. You can go forward again by typing `<Down Arrow>` – try this. Now find the `date -d +1day` again, use `<Left Arrow>` to step the cursor to just after the 1, then delete the 1 and type a 2 instead, then press the ←┘ key.

Now log out, by typing

```
<Ctrl>d
```

that is, hold the 'Control' (or 'Ctrl') key down and press 'd', then release 'Control'.

1.6.1 What happened there?

Now a quick explanation of what you've just done. You have been interacting with a **shell**. A shell is a program that prompts the user for commands, accepts them, executes them and displays the results. Unix has many shells: the first was called the 'Thompson' shell (also known as just 'sh', and pronounced "shell"), written by Ken Thompson for the first Unix system; then came the 'Bourne' shell (also called 'sh'), written for a later commercial version of Unix by Stephen Bourne. You have just been using the Free Software Foundation's 'Bourne Again' shell (another pun-name taking a dig at its commercial fore-runner), or 'bash'. The various different shells offer the user different facilities: 'sh' is rather primitive compared to the more modern ones. However, their basic functionality is always the same: they accept commands from the **standard input** (for now, we can treat that as meaning 'the keyboard'), execute them, and display the results on the **standard output** (i.e. for now 'the screen', which in this case was the entire screen, or **console**). Shells repeat this process until they have reached the end of their input, and then they die. `<Ctrl>d` is the standard way of marking the end of the keyboard input file in Unix. So when you typed it, the bash you were talking to ended, and you were logged out.

The use of `<Up Arrow>` and `<Down Arrow>` to browse the command history are examples of facilities offered by 'bash' that are not in the original 'sh'.

Unix shells are rather like **Command Prompt** windows in Microsoft Windows, except that Unix shells are more sophisticated.

1.7 A simple shell 'program'

We shall now give you a taste of the sophistication of shells. Login again, and select console login as before. When you get the prompt, carefully type the following lines, *exactly* – position of spaces and the case (upper or lower) of letters is significant.

```
while read name; do ←┘
if test "$name" = "$USER"; then echo "Welcome $USER" ←┘
else echo "Who?"; fi ←┘
done ←┘
```

If you make a mistake, type `<Ctrl>c` and start again. After each line, the computer should prompt you for the next line with a simple **>** ; if not, then you have made a mistake. When you have successfully typed these lines, the computer should simply be waiting for you to enter a name. So now type a name (e.g. your first name) and press ⏎ . The computer should respond with **Who?** You can do this several times using different names, but at some point you should enter your user-name. In response to this, the computer should display **Welcome mXXXXXXX** (except with your user name). When you get bored with running this first 'shell program', you can end it by signifying the end of its input (you know how to do this don't you?).

Your prompt should reappear. Now logout.

That was a rather contrived example, but it does show a tiny bit of the power of bash, which provides a language in which we can write programs to perform common tasks using programming constructs such a 'loops', as we have done in the above example. Real experts write **command line programs** like that quite often, when there is a single one-off job to be done. However, for more complex programs, or ones which may be needed more than once, it is usual to place the commands in a file, called a **shell script**. This is similar in idea to the Windows **batch file** but much more powerful. If you would like to know more about bash, then at some point (not now) you should run the command `man bash` or wait for the Bash scripting lab.

1.8 Where's the GUI?

Its quite possible that today is the first time you've interacted with a computer in this way: almost every modern machine boots up 'out of the box' into an attractive graphical environment that allows you to 'get going' with a minimum of fuss. Whether you're used to using Windows, or Macs, or even Linux machines, it's likely that you have done so via a Graphical User Interface, so you'll be very familiar with the 'WIMP' (Windows, Icons, Menus and Pointer) paradigm, even if you didn't know it had a name. The point of WIMP systems—and in fact of Graphical User Interfaces generally—is to make computers easy to use without having to learn much of what is happening behind the scenes. They are designed to allow human intuition to help with the process of using what is in reality a complex and sophisticated piece of hardware. The principle behind most GUIs is that of 'direct manipulation', the idea being that if you present concepts to a user in a consistent manner that to some extent reflects how you would expect things to happen in the 'real world', then your intuition will help you learn how to do things that otherwise you'd have to look up in a manual. In a direct manipulation interface, you have such things as 'files' and 'folders' and 'trash bins', all of which have real-world counterparts: so you instantly know that a 'folder' is a handy way of collecting 'files' together, and that when you don't want them any more you put them in the 'trash bin'. When you want to resize a 'window', you 'grab' it with the mouse pointer, and just drag a bit of it until it's the size you want: in other words, you manipulate things 'directly' in the GUI, and these manipulations are translated into much lower level changes that take place in the machine's memory, CPU, or hard disc.

As a computer scientist, there are many upsides of using GUIs; like everybody else, we want to be able to do everyday things with our computers: browsing the web, reading email, watching films, and the GUI undoubtedly makes this easier. It also means that your intuition, and the little bit of 'learning' that you did to get to grips with the idea of the GUI transfer well from one GUI to another; Macs, PCs and Linux boxes all have interfaces with similar enough graphical paradigms that it won't take you long to work out how to use them. But as computer scientists,

rather than 'casual users', we want to be able to do much more, so the downside for you is that a lot of the things you really need to understand about the behaviour of the underlying machine gets hidden by the GUI (and there are also many tasks that are better achieved *without* using a GUI at all).

The problem you'll have to come to terms with is that, having grown up with GUIs, it will for some considerable time, be tempting to turn to the GUI whenever you want to do anything. It will, after all, feel familiar and 'easy'. Learning the alternative will be, by comparison, quite hard, and doing things a different way may feel crude, messy, convoluted, esoteric and probably quite unpleasant at first. But trust us; going beyond the GUI is a crucial skill to being a successful computer scientist, and a vital part of your learning here in the School.

The first thing to understand here is that a computer's graphical environment is not a fundamental part of the machine; it's just another program running 'on' the operating system. In the case of Windows and OS X, as well as mobile devices, you don't get much choice in terms of the look and feel of the GUI (apart from making trivial changes like tweaking the colour scheme, or changing the 'desktop wallpaper'). But this is just because the companies behind these systems have opted for 'consistency' rather than 'flexibility', and not because of anything technologically fundamental. For systems aimed at the casual user, this is a very sensible thing to do; imagine how horrible it would be trying to provide support over the phone to someone when you first have to understand what kind of GUI they have – at least with consistent GUIs you have some idea where to start. In the case of Linux—which is somewhat less designed for casual use—there is far more flexibility in the choice of GUI, and we're going to use this to explain some important fundamental principles.

Okay, time to try it out – you need to login again.

login: ↵
Password: . . . ↵

Then you get asked what kind of interface you want. Check which window manager (or 'desktop') is being listed as your current setting. It should be **KDE**, and if it is then you need now only type ↵. If some other desktop is your current setting, then you first need to type `3` ↵ to select **KDE**. If you have used Linux before, and recognise your favourite desktop on the list, go for it.

1.9 The Linux Graphical Environment

Linux itself has no graphical environment, and relies on a program called the Xserver to communicate with the graphics hardware in order to display anything on the screen beyond the crude 'terminal' view you've already seen. To get a 'desktop' environment, we also need to run a window manager (in this case AnotherLevelUp), which deals with creating and drawing the familiar windows and icons of a desktop GUI. So when you selected something from the Desktop Switcher a moment ago, two things happened: first, the Xserver was started, and then the appropriate window manager. The 'layering' of these programs is shown in figure 1.2. This division of functionality into layers is a common architectural pattern that you'll encounter over and over during your studies.

The most commonly used desktop environments in Linux are GNOME and KDE; these are extremely sophisticated GUIs, and considerable design effort (both technical and artistic) has gone into making them easy and pleasing to use. We will use in the FY labs the GUI. However, you should appreciate what is really going on 'behind' the GUI.

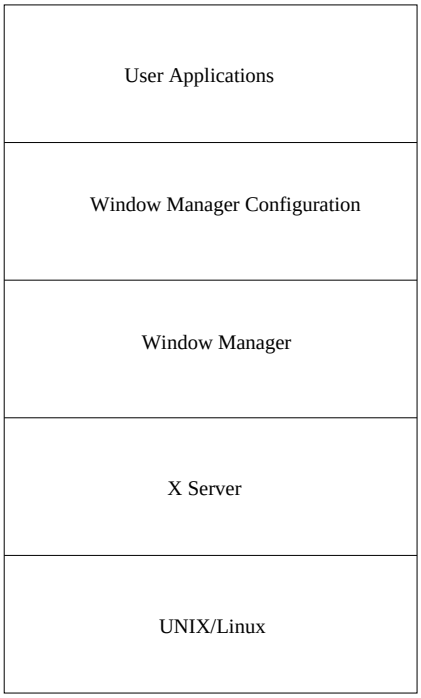


Figure 1.2: The layered structure of Linux's graphical system, with software nearest to the underlying hardware at the bottom, and software closest to the user at the top.

1.10 A quick introduction to KDE

The equivalent to the Windows Desktop is the background area in Linux which is known as the **root window**.

Navigation to our installed programs starts from the KDE start icon in the bottom right of the screen. A few basic programs include:

- **Firefox and Thunderbird** located under `Start -> Applications -> Internet` . You will use these for web browsing (Firefox) and email (Thunderbird).
- **Xterm** located under `Start -> Applications -> System -> Terminal` (Xterm). This window is running a bash (something similar to the DOS box under Windows) and you will use it for giving commands to be executed, just as you did in console mode before. As an example, move the pointer into the `xterm` window and type the `fortune` command. Run the command again (hint: you don't need to retype it!). Now run `date -d +2day` using only the arrow keys and return. Another way to get an **Xterm** window is by clicking with the right mouse button onto the background and selecting `Konsole` .
- **Nedit** located under `Start -> Applications -> Development -> Nedit` . Nedit is a text editor which you will use for manipulating text (for example, programs or reports). You will learn how to use this in a later practical session. (It is labelled Untitled because you are not currently editing a file: in general it places the name of the file you are working on in its title bar.)

1.10.1 Desktops

By default, you can switch between four different desktop windows which allow you to better organize your work. To change between the different desktops, you can simply click on the corresponding quadrant beside the clock in the bottom of the screen. Please play around with

this and open in an empty desktop a new program (e.g., Firefox) to see how this works. If you context click on the **title bar** (its a click with the rightmouse button) of a window, you can move it to another or all desktops. Again play with this.

You should think of good ways to use the different pages and desktops, for example you might be working on a laboratory exercise on one desktop, while waiting for your previous one to be marked on another. Meanwhile, to get you started, go to the first page of your *second* desktop and unhide your Firefox and Thunderbird windows by clicking on their buttons in the task bar.

1.10.2 Device and File Browser

The option labelled Device and File Browser pops up a sub-menu which allows you to start a file browser and also access devices such as hard disk, USB, CD/DVD ROM.

File Browser

`KDE Start -> File Manager` starts a file browser. Try it. We will work with files in the next session, so don't be scared if things do not look familiar to you.

Before we continue the big question: are file browsers a good idea? Certainly, everyone should agree that they are, in general, a good idea for the non-expert user. Whereas for experts, they are often much slower to use than **command line** oriented approaches like the Unix shells. And we believe that using a file browser all the time would hinder your understanding of what is really going on underneath: and as a Computer Scientist, it is most important that you gain that understanding as soon as you can. Of course, if your only choice was between a Windows Command Prompt window or Windows/Internet Explorer, then, it would be understandable to use Windows/Internet Explorer all the time. (Although Command Prompt is slowly catching up with power of bash.)

Our continued advice to you is that you should resist the temptation to simply 'transfer your previous experiences' to Linux/Unix: you will miss out on so much that is important to you.

1.10.3 System Utilities

Similarly to the Windows Task Manager, you can use `Applications -> System -> Gnome System Monitor` . Start this and explore this tool.

Another way to explore, manage and kill processes is to run a Linux (Unix) command called **top** in an `xterm` window. Top is an abbreviation for *table of processes* and it displays some system information, and then a continually updating chart of the jobs, or **processes**, currently running on the machine, presented in descending order of how much of the processor time they are using. You'll see lots of processes owned by a special user called **root**, and we are certainly not going to explain all that here! You should see some owned by you, in particular, it may well be that the process right at the top of the list is your `top` command itself - that's because your computer is hardly doing much at the moment. If you press `h`, you get the help window (or use Google for help on Linux top). If you move the mouse about a bit, you might see a process called `X` jump to near the top – that's the X server! When you get bored, close this window.

Over a period of time, you should familiarise yourself with the other options on this sub-menu, by experimenting, but not just now.

1.10.4 Ending windows

When you want to end (that is, close) a window, if the window has its own method to do this, you should always use it. For example, Firefox has a menu called File on which there is an Quit option – more about that later. So whenever you want to exit Firefox, it is better to use its Quit option than to use the Close button of its title bar. The reason for this is simple: software written for Linux comes from thousands of different sources around the world, and obviously different programmers can have different ideas of what their programs should do when given various **Signals** by a window manager. When you press the Close button on a window, the window manager sends a particular signal to the application that is running in that window, hoping it will shut down gracefully. You cannot be sure that the programmer who implemented that application was expecting his or her program to receive that particular signal as a means of being told to end, so it may not cause the program to end properly (e.g. an editor might not save the file you were editing). But you can be more sure that the application's own exit mechanism will work properly, as that was written by the same programmer (or one from the same team) that wrote the rest of the application.

The collective understanding between these different programmers is becoming more coherent and consistent, and over the past few years this problem has seriously reduced. However, it is best to play safe.

When a program has no exit mechanism of its own, or for whatever reason, it is failing, you can use the Close button of its title bar, if it has one. If this doesn't work, or the window does not have a title bar then you need to use one of the following methods.

You'll notice on the Window Operations sub-menu that there are three options, called Delete, Close and Kill. These get rid of a window, but they are slightly different:

- Delete – this invites the window to exit gracefully, but it might refuse to. This is the safest of the three.
- Close – this invites the window to exit gracefully, but if it refuses, it gets killed anyway! (This is the same as the Close button on the window's title bar, if it has one.)
- Kill – shoot first, and ask no questions: this just kills the window. This can be dangerous and should be used only as a last resort.

Whenever you use these options, you should bear these differences in mind, and be careful. For example, if you **Delete** an editor window, it's *likely* (but not guaranteed) that it will save your work before it dies gracefully, whereas if you **Kill** it, it won't get the chance. In general, only use Kill after you've tried Delete, and it has failed. It is almost always preferable to use an application's own exit mechanism if one exists.

You should now practise manipulating, creating and deleting windows on your screen until you are confident that you can arrange the layout of your screen at will.

1.11 Communicating with others

The time has now arrived to look at ways of communicating with the other people in the school and the outside world. We will look at two ways of doing this:

- **EMail**: This is the major means of communication within the school.
- **The Web**.

There are many different tools for reading and creating mail and for web browsing, and many users have their own particular favourites. For the moment we expect you to use **Firefox** and **Thunderbird**.

1.11.1 EMail

You should have Thunderbird on your screen – if you have followed this section carefully then it will be on the first page of your second desktop. Got to that page. (If you cannot remember how to do this, you are going too fast – go back and find out!)

If you don't have a Thunderbird running for some reason, start one now by selecting the New Web Browser main menu option. This (currently) actually starts both Firefox and Thunderbird. Alternatively, you could just type `thunderbird` ↵ in an xterm.

You should also have Firefox on your screen. Use it to go to `https://wiki.cs.manchester.ac.uk/index.php/How_to_set_up_Office_365_Mail_in_Thunderbird` which is a document written by fellow students telling you how to set up Thunderbird! Please follow *all* those instructions carefully, asking for help if you need it.

Now that you have successfully set up Thunderbird, let's explore it a bit.

To access your mail from the mail server, you click on the Get Mail icon. Another small window then appears asking you for your password[3]. For convenience, Thunderbird has probably already done that bit when you started it. Enter your normal Unix password in the box and click on the OK button when you are happy that it's correct. Any new mail that has arrived since you last checked will appear in the mail browser window.

You can read mail messages by clicking on their summary lines. You can also sort the messages by Subject, Sender or Date by clicking on the appropriate button above the summary lines.

Once you have read a piece of mail and decided that you don't want to keep it you should delete it by clicking on the Delete icon. However, this doesn't completely get rid of the message, but transfers it to a mail folder called *Trash* from where you can retrieve it if you delete a message by accident. To clear the Trash folder, which you should do regularly, select the Empty Trash menu item on the File menu. *If you don't do this regularly your allocated disk quota might get used up by old mail messages.*

You can reply to a message or forward it to another person by using the appropriate icons in the mail browser window. To send a new message of your own use the Write icon. Just fill in the mail address of the recipient and the subject line. Once you have finished composing the message just click on the Send icon to send it. Try sending a message to a friend or someone sitting near you in the lab. Should you decide not to send a message once the composition window is up, you can select Close from the window's File menu.

If you receive mail from someone whose address you might want to use later, you can add their name to an *Address Book* by right clicking on their email name. You can then refer to the address book when composing messages by using the Contacts icon on a composition window.

Thunderbird is itself quite configurable, and remembers most settings you make. You can change the size of the various windows, change the position of the split between multiple parts of one window, make it remember your password so you don't have to enter it each time you read your mail, and so on. Rather than overload you now with details, you are encouraged to explore the Thunderbird menus over the next few weeks.

---

3. This is because Firefox uses a protocol called IMAP to access your mail from a mail server and the password is required for security purposes.

Email is important

It is important to realise that email is one of the major means of communication in the school, so make sure you follow these rules:-

- Read your email regularly, *at least* once a day.
- If you don't receive email and suspect that there may be something wrong with your email setup, *Do Something About It Quickly*, don't just ignore the situation. Amazingly, every year there are some students whose email stops working, they do nothing about it, and get themselves into a serious mess when they have missed important announcements, meetings and deadlines.

1.11.2 The Web

Every document and resource on the Internet has a unique address known as its *uniform resource locater* (URL). A URL is made up from the document's name preceded by the hierarchy of directory names in which the file is stored (pathname), the Internet domain name of the server that hosts the file, and the software and manner by which the browser and the document's host server communicate to exchange the document (protocol).

It's a good idea right now to go to the web page for this course and bookmark it. You can do this as follows.

Go to `http://studentnet.cs.manchester.ac.uk/ugt` and make this your home page. You should add it as a bookmark as well. The find **Study & Curriculum** and then **Syllabus** which should take you to a long list of courses. Under **Service courses** you will find **COMP00900** – that is the course code we use withing CS for this course. (For some reason!) There are two links of interest **Materials** and **Syllabus**. Probably the former is the most interesting right now – select it and bookmark it.

Those of you who are interested can find lots more information about Linux at the Linux Documentation Project web pages, which can be found at `http://tldp.org/` .

1.12 RTFM

This is a frequently used Computer Science acronym. It is said to stand for 'Read The Flipping Manual' (or something similar).

You will already be aware that there are lots of things to know about Linux and computers in general, far more than you have encountered so far, and you will not remember it all. Fortunately, there is lots of documentation available and you should get into the habit of knowing how to use it properly.

We expect you to explore computer related topics using self learning techniques. This will be important not only to succeed in the course but also in the rest of your career.

The information you have available includes:

- These notes
- Various lab manuals.
- The Unix on-line manual system, described below.
- Vast amounts on-line on The World Wide Web.

Meanwhile, these sessions are about Unix and it is appropriate that you should start to understand the Unix manual system. Most Unix commands are described in the on-line manual system, which you use by typing:

```
man <name of command>  ←┘
```

e.g.

```
man fortune  ←┘
```

tells you how to use fortune, and

```
man man  ←┘
```

tells you how to use `man` itself. When you have time later, look at this manual and find out what the `-k` option does. You will find this information very useful – how are you going to make sure you don't forget to have a look at it later?

An example man page to try right now is

```
man xclock  ←┘
```

This tells you how to create a variety of clocks on your screen. Look particularly at the section headed **SYNOPSIS**, which tells you what options are available, and the one headed **OPTIONS**, which tells you what they do. If the description takes more than a page of your screen, you will see a colon at the bottom

:

When this occurs, you can see the next page by pressing the space bar. To move down the text only one line at a time, press ←┘ . To quit reading the manual page, type `q`.

1.13 Command line arguments, interrupt and background running

There's one problem with running clocks from an `xterm` . If you type:

```
xclock  ←┘
```

a clock does indeed appear. However, you then can't type any more commands in the `xterm` window, because it is waiting for the clock program to finish, and of course the clock program is designed to run forever (or at least until you logout). There are several ways to get round this:

- When you're bored with a clock, you can delete it by choosing the Delete option from the Window Operations sub-menu. The clock will disappear, and you'll get a prompt back in the `xterm` window.
- You can obtain the same effect by typing `<Ctrl>c` (hold the 'Ctrl' key down and type `c` at the same time) in the `xterm` window where you typed `xclock` . This is worth remembering, since you can use it when your own programs accidentally run forever!
- If you type an ampersand on the end of a command, e.g. `xclock &` ←┘ , the program gets run **in the background**, that is, separately from the `xterm` window. This way you can have several clocks at once and continue typing other commands in the `xterm` window.

Try all of these three approaches.

1.14 Locking the screen

Whenever you leave your computer unattended, you should lock the screen so that no-one can damage your files (or worse). The simplest way to do this is by clicking on `KDE Start` then hover over the `Leave` icon and continue to Lock Screen.

You have locked your screen, so you could in theory go away as your login session is protected. But we have a rule in our School laboratory policy: you must never lock your screen for more then 20 minutes. That's ample time enough for a toilet break, or to stretch your legs – something you are recommended to do every hour for health and safety reasons. If you are going to be away for more than 20 minutes, you must log out instead: it is very anti-social indeed to hog a machine you are not actually using, and we take a very dim view of it. (Anyone who offends repeatedly in this way, has their account set so it can only be used between 5pm and 9pm!)

Now, with the screen still locked, type ⏎. Notice that `xlock` shows your user-name and the time which has elapsed since you locked the screen. Whenever and however you lock your screen *your user-name and the elapsed time must always be shown.* This is because if the elapsed time is more than 20 minutes, anyone wanting to use the machine has a *right* to log you out (by killing the X server) and then log in. The user-name is needed so that a member of staff can tell whether or not you are supposed to be present in a laboratory which may be happening at the time. If either the user-name or the elapsed time is missing, you are breaking the rules and anyone is allowed to assume you are not supposed to be there and/or that you have had the screen locked for more than 20 minutes!

Now unlock the screen: type ⏎ to get `xlock` 's attention, then type your password. You should now be unlocked, but if it didn't work, just try it again – perhaps you mis-typed your password.

1.15 ARCADE Client

Before you finish this (long!) first session, we would like you to register your access to the ARCADE server. ARCADE is the laboratory management system we use to administer course-work marks and deadlines, etc.. You can use the client query program to look up your details, laboratory timetables, marks, and so on, throughout the year. It is recommended you do this fairly regularly, if only to check that mistakes have not been made in your marks! You need to register with ARCADE before any of your lab work can be marked, so it is much better to do it now than wait until your first deadline!

On the main menu, find the sub-menu called ARCADE and select the ARCADE Query menu item. After a short pause, a new window will pop up. In the large text box you should find a message telling you that you are not yet authenticated, and that you have just been sent an email. If this is so, then quit the program, read the email you have just been sent by the ARCADE server, and carefully follow the instructions in it. Then you will start the ARCADE Query client again, and it should now connect you to the server. If this works, then your access is set up!

Here's a question for you: can you think about how that worked? Something has just happened so that now whenever you run the client, you can access your details, but no-one else's. And others cannot access yours.

Explore the various queries of the client, and familiarise yourself with the user interface. Check that your registration details are correct. Feel free to ask for explanation if it is not obvious to you.

1.16 Really finished Session 1?

If you finish early it is better to practise manipulating the windows than to go onto the next exercise. The more familiar you are with the user interface the more time you will save, and the less costly mistakes you will make later. Try out the menu options we didn't look at. Explore the relationship between the main menu options Other Programs and Shell browser and Favourites.

Find the Open Office items on the Other Programs menu and try them out. These are 'open source' (free) versions of the well known Microsoft Office Tools.

If you've taken less than an hour, grab a piece of paper, and a pen, and make a brief list of *all* the things you've just learnt. Then scan the text and check what you missed from your list. If you score less than 90%, go back to the start and do it all again!

When you've finished, you *must* log out – otherwise the hackers will be able to impersonate you, and life will be miserable! Log out by choosing the Yes, Really Quit option of the Exit Fvwm sub-menu available at the bottom of the Start menu. *Don't forget to tell the laboratory supervisor that you have completed session 1.* If you do not finish during the session, that is okay, but please tell the laboratory supervisor during session 2 that you have completed session 1.

Session 2: Manipulating files and directories

This session concerns the practical aspects of manipulating **files** and **directory** structures. It is extremely important that you work through it carefully, in sequence, and complete everything (except possibly some of the **Exercises** at the end). The second section contains some notation which you need to understand before you start.

2.1 What is a file?

A **file** contains information, which may be in human-readable text form, or in other forms, e.g.

- An **object** program; this contains machine instructions.
- **Data** held in some format known to the programs which use it, but not readable by humans, for example, a database.
- Compressed or encoded data, e.g. some files containing accounting information about usage of the system.
- A **directory**. This is a file which contains information about other files, and generally can be regarded as something which *contains* other directories and/or files. (In Windows these are called folders.)

Although most of the files you will work with contain text, you will soon learn (if you do not know already) that all computer information is represented as binary digits (or bits). In the case of a text file, the bits are interpreted as characters. In the case of an object program, the bits are interpreted as machine instructions. In the case of a file of some other sort of items - let's call them widgets – the bits are interpreted as widgets.

2.2 Specifying filenames in Unix

In Unix, as in Windows, the files and directories you create can have more or less any name you like. It is very sensible to give them names which mean something and make their purpose clear. This is despite some of the traditional file names in Unix being rather cryptic – this is particularly true for some of the commands (which are themselves object files). You'll get used to that.

On each machine, there is one directory called the **root** which is not contained in another directory. All other files and directories are contained in root, either directly or indirectly. The root directory is written "/". Note this is the opposite slanting slash character to that used by MS-DOS and Windows. When Microsoft borrowed the idea of directories from Unix they chose the other slash to distinguish the two systems. (Those of you with Microsoft experience should also compare the principle of one root per machine with the Microsoft principle of a 'root' per *disk* on each machine (e.g. A:, C:, etc..)).

If we wish to talk about the file y within the directory, x we write x/y. y may itself be a directory, and may contain the file z, which we can describe as x/y/z.

You can think of this structure as defining a **tree**, with "/" as the **root** (hence its name), directories as **branches**, and other files as **leaves**. You will study trees as an abstract structure, later in the year. This simplified model of the Unix **file system structure** will do for now. (For those of you who are interested: Unix actually allows **links**, which means the structure can really be a cyclic graph. Links are similar to, but fundamentally not the same thing as, shortcuts in Windows.)

Apart from "/", there are two more directories of special note:

- Your **home directory** is the directory where you find yourself when you log in; this will be some way 'down the tree'. For example, mXXXXXXX's home directory is /home/mXXXXXXX – in other words, the directory mXXXXXXX within the directory home within the root directory.
- Your **current working directory** is the one you are working in at a particular moment. For example, a student mXXXXXXX who is working on INTRO, exercise 2, should have the directory /home/mXXXXXXX/INTRO/ex2 as current working directory. (You will create this directory shortly.)

A name which uniquely identifies a file (on a given machine) is called its **pathname** because you are specifying a path through the tree to reach it. Pathnames can be given in two ways:

A pathname beginning with a "/" is an **absolute pathname**, which starts from the top of the tree, e.g.

/home/mXXXXXXX/INTRO/lightbulbs

A pathname without the initial "/" is a **relative pathname**, relative to the current working directory. For example, if the current working directory is:

/home/mXXXXXXX/INTRO

the same file can just be called

lightbulbs

while from /home it can be referred to as

mXXXXXXX/INTRO/lightbulbs

It is important to realise that, unlike Windows, Unix filenames are *case sensitive*, so that the file name myfile is completely different from, and has no relationship to, the filename MyFile. Historically most Unix file names have tended to be all lower-case but this is not a rule.

Quick exercise: This method of specifying relative pathnames has a severe limitation. Think about what it is, and how it might be overcome. (Solution later – if you spot it.)

Log on and go into X with KDE, then follow the instructions in the next sections.

2.3 The `pwd` and `ls` commands

The `pwd` (short for "print working directory") command prints the absolute pathname of your current working directory. In an xterm window, type

```
pwd ←┘
```

and it will show you the name of your home directory.

The `ls` (short for "list files"!) command shows the files in your working directory. If you type

```
ls ←┘
```

you'll see something like:

```
My Music  SeaMonkey-Win  desktop.ini  folders  pine-nt
```

This doesn't actually show *all* your files, some are hidden by the normal `ls` on the assumption that you don't want to see them most of the time. The names of these hidden files all start with a dot. If you try

```
ls -a  ←┘
```

you'll see something like:

```
.               .ab_library     .mailtool-init              .xserverrc
..              .bash_logout    .openwin-init.olvwm         My Music
.OWdefaults     .bash_profile   .openwin-init.olwm          SeaMonkey-Win
.Xauthority     .bashrc           .openwin-menu             desktop.ini
.Xclients         .cs_maildir     .openwin_solaris-start    folders
.Xdefaults      .dtprofile        .openwin_sunos-start      pine-nt
.Xmodmap.sun4   .emacs            .textswrc
.Xmodmap.sun5   .fvwm2rc.m4     .twmrc
.Xresource      .mailrc           .xinitrc
```

`-a` is an **option** on the `ls` command (short for "all"). Options which are either there, or not, are also called **flags**.

Various files, most of which have names beginning with a dot, were created for you when your account was set up. The purpose of some of these 'dotfiles' will be explained properly later. Most of them are used to provide information needed by commonly used programs. For instance, the file .xinitrc controls the mechanism followed when starting up X in one of the several machine/software environments available in the school. The directory called .mozilla contains your personal setup and various other files for use by Mozilla.

Now try the `-l` flag of `ls`

```
ls -l
```

and then both `-a` and `-l` flags

```
ls -la
```

The `-l` flag gives lots more information about the files shown, rather than controls which files are shown. This information includes the date of last modification of the files, the owner of the files, and other things you don't need to worry about just this minute (see **Exercises** later on).

2.4 Creating a directory structure

You're now going to use the `mkdir` ("make directory") command to create some directories. Type

```
mkdir INTRO  ←┘
```

Check that this directory has indeed appeared using `ls`.

It's important that directories we ask you to make for your work have exactly the names we specify. Unix will let you use any names you like, but we have to impose conventions for administrative reasons.

If you made a mistake, e.g. intro instead of INTRO, you can remove the directory while it is still empty with the `rmdir` command: e.g.

```
rmdir intro  ←┘
```

And then try to make it correctly.

The `cd` ("change directory") command allows you to move around the tree by changing your current working directory. Type

```
cd INTRO ←┘
```

to make INTRO your working directory. Check that you have changed current directory (if you can't remember how – slow down!).

Note that the version of `cd` that you are using also does something similar to `pwd`.

Now make directories for each of the INTRO exercises

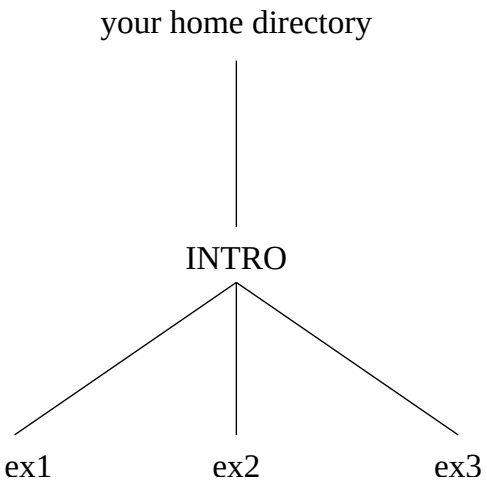
```
mkdir ex1 ex2 ex3 ←┘
```

The `cd` command on its own takes you back to your home directory, wherever you may be when you invoke it. Try this out by typing

```
cd ←┘
```

Check that you are now in your home directory.

Your directory structure should now look something like this



The easiest way to check this is to use (from your home directory) `ls` with the `-R` flag. This shows the whole tree below your current working directory (`-R` is short for "recursively" – look up the word in a dictionary, or wait until later for a definition!).

```
ls -R ←┘
```

2.5 The 'dotfiles' (and shell variables)

Every directory is created with two files already there, called "." and "..". Of course, you don't see them when you run `ls` because they start with a dot! Now run the command which will enable you to see them in your current directory.

"." is a reference to the directory itself, and ".."is a reference to the directory above it in the tree. This may seem rather bizarre at first, but they are in fact extremely useful. "..", for example, enables you to specify a *relative* pathname *up* the tree.

Try the following sequence of `cd`s, checking where you are after each one, and make sure you understand what is going on!

```
cd  ↵
cd INTRO/ex1  ↵
cd ..  ↵
cd ex2  ↵
cd ../ex1  ↵
cd ../../INTRO/ex2  ↵
cd ../..  ↵
```

As we said earlier, the other dotfiles in your home directory contain assorted useful information. For instance, the file .bash_profile sets all manner of things each time you log on – you should take a look at it some time, but not just now. Among the many things it does, it sets the value of a **shell variable** called $INTRO to the long and boring path name of the place where we keep the master copies of various files used in INTRO. Let's look at its value – type

```
echo $INTRO
```

`echo` is a command which just displays its arguments – very useful just here. If shell variables didn't exist, or if we hadn't taken the trouble to define $INTRO (etc.) for you, you'd have to remember and type long path names like that one frequently – starting in the next section!

There is another shell variable which is set for you, called `$HOME` . Find out its value. Now you can easily reference your home directory from wherever you are. Actually the shell you are using (what is it called?) provides you with an even more convenient way of referring to your home directory: you can simply use a single tilde character.

```
ls ~
```

2.6 Copying, moving, and removing files

This section introduces three commands used for copying, moving and removing files. We first describe each command and then invite you to practice using them.

The `cp` (copy) command has two forms.

The first general form is

```
cp <file> <file>  ↵
```

For example

```
cp file1 file2  ↵
```

makes a copy of the file file1 and calls it file2. If a file called file2 already exists, you will be warned and given a chance to change your mind, or have the existing file2 overwritten with a copy of file1.

The second form is slightly different:

```
cp <file(s)> <directory>
```

For example

```
cp file1 file2 file3 dirname  ↵
```

This copies the files file1, file2, file3 into the directory dirname, again overwriting any files already there with the same names.

The command `rm` (remove) is used to delete files.

```
rm <file(s)>
```

throws away the specified files – always take great care when using `rm`, it is not reversible, although you will be asked to confirm each remove.

The `mv` (move) command is similar to `cp`, but it just moves the files rather than makes copies. Again we have the two forms

```
mv <file1> <file2>      ←┘
```

and

```
mv <file(s)> <directory>      ←┘
```

The effect is like a copying followed by removing the sources of the copy, except it is more efficient than that (most of the time). For example

```
mv file1 file2     ←┘
```

is like doing

```
cp file1 file2     ←┘
rm file1
```

and

```
mv file1 file2 file3 dirname      ←┘
```

is like doing

```
cp file1 file2 file3 dirname      ←┘
rm file1 file2 file3
```

Before continuing, answer this question and check your answer with a member of staff: how do you *rename* a file in Unix? Don't guess – you know the answer, unless you are going too fast.

Now for some practice. Go to your home directory

```
cd   ←┘
```

and copy the file called lightbulbs in the $INTRO directory to your current working directory:

```
cp $INTRO/lightbulbs .      ←┘
```

Note that `$INTRO` must be in uppercase and the full stop is essential. If you now do an `ls`, you should see that the file called lightbulbs has appeared in your directory:

```
ls   ←┘
```

If no file called lightbulbs has appeared, the following will probably provide an explanation. If it did appear, read this anyway, just to check that you understand what you did right!

The `cp` command needs two arguments. In this case, the file you are copying is $INTRO/lightbulbs, and the directory you are copying it to is "." (that is, your current working directory – remember every directory has a reference to itself within it, called '.'?). If you missed out the dot, or misspelt $INTRO/lightbulbs, or missed out one of the spaces, it won't have worked. In particular, you may well have got a 'friendly', helpful error message like:

cp: missing destination file
Try 'cp --help' for more information.

or

cp: /opt/info/courses/INTRO/lightblurbs: No such file or directory

or

cp: INTRO/lightbulbs: No such file or directory

If you get the first message, it means you used the command with the wrong number of arguments, and nothing will have happened. The others are examples of what you might see if you mistype the first argument. If you do get an error message you need to give the command again, correctly, to copy the lightbulbs file across.

So you now have a copy of the file, but, it's in your home directory. You'll have to get into the habit of *not* having all your files in your home directory, otherwise you will quickly have an enormous list that will take you ages to find anything in. The use of subdirectories provides a solution to this problem, which is why you created some earlier. Moving this file to the 'correct' place gives you a chance to practice the `mv` command.

Move the file lightbulbs to your INTRO/ex2 directory.

Notice that INTRO is *your* INTRO directory, while $INTRO is (an abbreviation for) *our* INTRO directory, from which you can read things but cannot write to.

Now go to your INTRO/ex2 directory and check that the file has arrived there.

To make sure you understand `cp`, `mv`, and `rm`, go through the following sequence (in your INTRO/ex2 directory), checking the result by looking at the output from `ls` at each stage

```
cp lightbulbs bulb1      ←┘
ls   ←┘
cp lightbulbs bulb2      ←┘
ls   ←┘
mv bulb1 bulb3     ←┘
ls   ←┘
cp bulb3 bulb4     ←┘
ls   ←┘
rm bulb2    ←┘
ls   ←┘
rm bulb1    ←┘
ls   ←┘
```

Why does `rm bulb1` behave differently to `rm bulb2` ?

2.7 Looking at the contents of files

So far, we have manipulated files without caring about their contents. In practice, we often want to look at (and maybe modify) the contents of a file. There are a number of ways to do this. The most general method is to use a text editor, which is the subject of the next session. A quick and easy way to look at the contents of a file is by using the `more` command. Type

```
more lightbulbs    ←┘
```

to look at the contents of the file called lightbulbs (don't waste your time actually reading all of it!) To go to the next screen full, press the space bar. To move down a line at a time, press ←┘ . To exit from `more` before you reach the end, just type `q`.

`more` is so named because it prompts for you to tell it to show you "more" of the file when the screen is full. There is a similar command called `less` , an improved version of `more` , which allows you to move backward as well as forward through the file – using the arrow keys. Try it. You have to type `q` to leave `less` , even when you get to the end of the file.

If you want to find out more about `less` , you can always try

```
man less   ←┘
```

Try it anyway. Notice how `man` uses `less` to display manual pages. This is typical of the Unix philosophy – once a tool is available, it is used in building other tools.

Note: you may be used to using `notepad` , `wordpad` or `word` in Windows just to look at the contents of text files, even though these are editors rather than viewers. This would translate into using your Linux text editor (`nedit`) to look at files, even when you do not wish to edit them. If in a few weeks time your natural instinct is to use `nedit` rather than `less` to just look at a file, then you are missing part of the point of using Linux to help you think differently.

2.8 Wild cards

An asterisk in a filename is a **wild card** which matches any sequence of zero or more characters, so for instance, if you were to type (don't actually do it!)

```
rm *fred*  ↵
```

then all files in the current directory whose names contain the string "fred" would be removed.

Try the effect of

```
ls bulb*  ↵
```

and

```
ls *bulb*  ↵
```

Now try

```
echo *bulb*  ↵
```

Are you surprised by the result?

One exception to the above rule is provided by the dotfiles; i.e. those whose names begin with ".". The asterisk will not match a "." at the start of a file name. To see what this means try the following

```
cd  ↵
ls *xinit*  ↵
```

and

```
ls .*xinit*  ↵
```

and see the different output.

2.9 Quotas

The command

```
quota  ↵
```

shows you what your file store quota is, and how much of it you are actually using. This is only of academic interest now, but may become very important later in the year! You may find that you are unable to save files, or even receive mail, if you use more than your quota of file store. It is important that, if this happens, you do something about it immediately.

2.10 Putting commands together

Before you forget that you're in your home directory, change directory back to your INTRO/ex2.

One of the simplest (and most useful) of Unix commands is `cat` . This command has many uses, one of which is to con**cat**enate a list of files given as arguments and display their contents on the screen. For example

```
cat file1 file2 file3      ←┘
```

would display the contents of the three files file1, file2 and file3. The output from `cat` goes to what is known as the **standard output** (in this case the screen).

If you type

```
cat   ←┘
```

nothing will happen because you haven't given a file to `cat` . When run like this, it takes its data from the **standard input**, which in this case is the keyboard, and copies it to the standard output. Anything that you now type will be taken as input by `cat` , and will be output once each line of the input is complete. In Unix, end of input is signalled by `<Ctrl>d` . (Recall that typing `<Ctrl>d` in your login shell will log you out – you have told the shell to expect no more input.) So, after typing `cat` above, if you type (we will omit the ←┘ 's from now on, we hope you've got the message)

```
This is
some
text for cat to
digest
<Ctrl>d
```

you will see the input replicated on the output (interleaved line by line with the input). The first copy is the echo of what you typed as you typed it, the second copy is output from `cat` . This may not seem very useful, and you wouldn't actually use it just like that, but it illustrates the point that `cat` takes its input and copies it to its output. Using this basic idea we can do various things to change where the input comes from and where the output goes.

```
cat > fred1
```

will cause the standard output to be directed to the file fred1 in the working directory (the input still comes from the keyboard and will need a `<Ctrl>d` to terminate it. Try creating a file `fred1` using this technique, and then check its contents.

```
cat < fred1
```

will take the standard input from the file fred1 in the working directory and make it appear on the screen. This has exactly the same effect as

```
cat fred1
```

You can, of course, use < and > together, as in

```
cat < fred1 > fred2
```

which will copy the contents of the first file to the second. Try this and check the results.

We can, of course, do this type of redirection with other commands. For example, if we want to save the result of listing a directory's contents into a file we just type something like

```
ls -l > fred1
```

(this overwrites the previous contents of fred1 without warning, so be careful of this kind of use).

One of the other things that `cat` can do is to put line numbers on its output. It does this if you use the `-n` flag. Try

```
cat -n fred1
```

Now suppose, for the sake of argument, we wanted to have a listing of the names of the files in the current directory, with each line numbered, and the result saved in a file `fred3` . You have just been given all the information you need to do this – so, how would you do it? Do it now.

Unless you've met Unix before, you probably did something like this

```
ls > tmpfile
cat -n tmpfile > fred3
```

Or if you didn't then try it now and examine the contents of fred3. The file tmpfile can now be thrown away using `rm tmpfile` .

It's a shame we had to use an extra, temporary, file. Could we avoid having to? Why do you think the following would not work?

```
ls > fred3
cat -n fred3 > fred3
```

A better way of doing the task, which avoids the use of a temporary file is by use of a powerful Unix feature called the **pipe**. We just type

```
ls | cat -n > fred3
```

The **pipe** symbol `|` indicates that the **standard output** of the first command is to be **piped** into the **standard input** of the second command, so no intermediate file is needed.

We can construct another (slightly artificial) pipeline example using just `cat` .

```
cat < fred1 | cat > fred2
```

The first `cat` takes its input from fred1 and sends its output into the pipe. The second `cat` takes its input from the pipe (i.e. the output from the first `cat`) and sends its output to `fred2` . (How many other ways can you think of to do this?) This isn't a frightfully sensible thing to do, but it does illustrate the principle of piping (long pipelines of commands may be constructed), and more realistic examples will appear in the exercises.

Standard output sent to the screen may come so fast that it disappears off the top before you have had a chance to read it. There are three ways around this problem.

- Using foresight, pipe the output into the command `more` or `less` which arranges to stop after each page full. For example,

  ```
  ls -la | less
  ```

  would be a wise precaution if the current working directory held more than a screen full of entries. When `less` has shown you the first screen full, press `<Space>` to see the next screen full, or ⏎ to see just the next line.
- Without foresight, the output will rush past you at a great rate of knots. Press `<Ctrl>s` to stop it dead in its tracks (and `<Ctrl>q` to set it off again). In practice this isn't much use nowadays – in most cases the computer is just too fast for the Human to press the keys at the right time.

2.11 Printing text files

The command `lpr` can be used to send files to a printer. In its simplest form, you simply run:

```
lpr file1 file2
```

to print the given files. You could use it now to print out the lightbulbs file, but that file is quite big and we don't want to waste a lot of paper. So please don't! However, it would be nice to practice using `lpr`. So, you shall print out just the first 50 lines of it. Look at the man page for `lpr` and discover what it does if no file names are given. Now look at the man page for the command `head` and figure out how to make it output the first 50 lines of the file lightbulbs. Experiment with this to make the 50 lines appear on your screen. From what you have already learnt, you should know how to check there are exactly 50 lines using the counting option of `cat`. Now send the 50 lines to the printer – without using an temporary file. Go and collect your print output – ask for directions to where the printer is.

`lpr` is a basic printing tool for printing text (and it is also clever enough to print most types of images nowadays). A more sophisticated printing program is `a2ps`. This produces a nicer output, and is clever enough to recognise different types of text file, including program source code files, and present the various keywords of the programming language and the program identifiers in different fonts to help with readability.

Look at the man page for `a2ps`, figure out how to print the file $INTRO/SimpleJavaProgram.java and then do so. When you collect this print output you will see how pretty `a2ps` makes it. This is a good way to obtain print outs of your own work, should you wish to.

2.12 Exercises

Here are a variety of things to experiment with if you finish everything else. More details of the relevant commands can be found by using `man`.

1. As stated above, `ls -l` gives you extra information about files. Skim through the man page for `ls` to see what it means. Check the ownership and protection of your own files. Why don't you own ".." in your home directory? For more about ownership and protection, look at the manual pages for the `chown` and `chmod` commands.
2. Look at the `man` entry for `rm` and find out what would happen if you did `cd` and then `rm -rf *`
   NB. DON'T TRY IT! We once had a system administrator who, after logging in as the **superuser** (that's a special user called **root** that has the permission to do *anything*), typed the above command by accident. What do you think happened? (Hint: on many Unix systems, the superuser's home directory is /).
3. Another useful command is `grep`, which displays all lines of a file containing a particular string (in fact, the string can be a pattern with wild-cards and various other things in). The form for a simple use of `grep` is
   ```
   grep <pattern> <file(s)>
   ```
   This will result in a display of all the lines in the files which contain the given pattern. A useful file to use for experiments with `grep` is /usr/share/dict/words, which is a spelling dictionary. Try to find all words in the dictionary which contain the string "red".
4. Use a suitable pipeline to find out how many words in the dictionary contain the string "red" but not the string "fred". (Hint: The answer to the previous question gives all the words containing "red", look at the manual page for `grep` to find out how to exclude those words containing "fred". The `wc` (short for "word count") program counts words (amongst other things). Use pipes to put them all together.)

5. Investigate the `ps` command, which tells you about the processes (running programs) on your workstation, how much swap space they are using etc..
6. (Harder) Wander around the top of the directory tree, from /, and try to understand what you find there.
7. Try the exercises contained in the file $INTRO/extras.

Session 3: Text Editing

3.1 Computer text processing

This session will introduce you to `Nedit` , a simple text editor, which you will use a great deal throughout your course (unless you find another editor that suits you better).

The techniques you will learn today are fundamental, and you should practice them until you can do them without thinking.

This session only covers the basics. For information about other facilities in Nedit, and for quicker ways of doing things, see the Help menu in Nedit.

3.2 Document preparation systems

There are many computer programs available which help in the preparation of documents. They can be very roughly classified as follows:-

A text editor	allows you to type text, correct mistakes, and modify the text at will. You will use the text editor `Nedit` to write your computer programs, etc..
A text processor	takes text, with extra formatting commands, and formats it neatly, using a variety of fonts and formatting conventions. These notes are produced using such a system, called LaTeX (pronounced "lay-tek").
A word processor	such as Microsoft Word is a combination of a text editor and text processor.
A desktop publishing system	combines text editing and formatting with many other facilities such as drawing packages, spelling checkers, and automatic maintenance of references, tables of contents, footnotes and indices. Such systems (and some word processors) are often described as WYSIWYG (What You See Is What You Get) because the document appears on the screen in (more or less) the form in which it will be printed.

In practice the boundaries between these types of program have become increasingly blurred as text processors and word processors incorporate more and more desktop publishing facilities.

The remainder of this section is devoted to an introduction to the text editor `Nedit` . Nedit is only one among many text editors available on Linux; once you are familiar with the principles of text editing you may like to explore others, some of which are described on the local Software Links web page.

3.3 Creating a small piece of text

Move the mouse pointer into your Nedit window and type:

```
A: Two, one to screw it almost all the way in and the other to give
it a surprising twist at the end.

Q: How many mystery writers does it take to screw in a light bulb?
```

(Yes, we know it's the wrong way round – this is deliberate!)

Notice how the blinking vertical line (the **cursor**) indicates where you are inserting the text. You can move the cursor around in the text by pointing to where you want it and clicking the left mouse button. For short distances, it may be more convenient to use the arrow keys on the right hand side of the keyboard.

The Return key inserts a newline at the cursor. Make sure you put newlines in exactly as shown above.

The Delete key deletes the character to the right of the cursor. The backarrow above the return key deletes to the left. By moving the cursor, deleting characters, and inserting new ones, you can correct any mistakes you make. (If you haven't done so already, make some deliberate mistakes and correct them.)

3.4 Selections

Many operations in Nedit are done by selecting pieces of text. To make a selection, select the beginning with the left mouse button and then, *still holding the left button down*, **wipe** across the desired text. The selected text appears highlighted, with a different background colour.

There are quicker ways of making common sorts of selections:

- Click the left mouse button **twice** fairly rapidly to select the **word** you are pointing at.
- Click the left mouse button **three times** to select the **line** you are pointing at.
- Click the left mouse button **four times** to select the **whole text**.

Experiment with selecting various bits of the text you have typed. Try each of the methods described above.

3.5 Cut and paste

One of the main advantages of computer text processing is the way you can move text around, and change the structure of a document very easily.

Most text editors have operations called **cut** and **paste**. Cut deletes a piece of text and (invisibly) saves it somewhere. The place where it's saved is called the **clipboard**. Paste takes whatever text is currently on the clipboard and inserts it at the cursor. Hence, a cut followed by a paste can be used to move a piece of text from one place to another.

Try this on your text:

- Select the question part of the light bulb joke.
- Select the Edit menu by clicking on Edit with the left mouse button; Select Cut by clicking with the mouse button. The selected text will disappear.
- Position the cursor at the beginning of the text, where the question should have been.
- Choose Paste from the Edit menu , and the text will reappear in the correct place.

In what follows the notation: Edit ▷ Paste means select the Edit menu and choose Paste from it.

The Edit ▷ Copy option allows you to copy some text to the clipboard without deleting it. You can then paste it wherever you want to. Try using this to make an extra copy of the whole text within the Nedit window.

An alternative to using two mouse clicks, first on the menu and then the menu item, is to pull down the Edit menu by holding down the left mouse button on Edit, dragging the arrow down to the required option (e.g. Cut) and then releasing the mouse button.

The keyboard accelerators for these operations are

<Ctrl>x	cut
<Ctrl>v	paste
<Ctrl>c	copy

The menu reminds you of these. Try using these to cut and paste, you should find them much quicker than selecting menu options.

A powerful feature of the X-windows system (and many others) is that there is a selection and clipboard, shared between all the windows. This means you can easily move text from one document to another. The method which works with (nearly) all programs is to copy onto the clipboard by selecting with the left mouse button and paste with the middle mouse button. This gives an alternative way of copying and pasting in your Nedit window but is more important between windows. For example, you can insert a fortune into your text as follows:

In your Xterm window, type:

```
fortune ←┘
```

Select the text produced using the left mouse button. Now move the pointer over to the Nedit window, position the *pointer* (not the Nedit cursor) to the point where you want the text to go, and simply click the middle button. There is a good chance you didn't quite get this right: you must position the pointer to the location you want the text to go, and click *only* the middle button. If you did get it wrong, type <Ctrl>z in the Nedit window to undo it, and try again.

Now we'll explain Undo properly! If you cut something you didn't mean to, Edit ▷ Undo will restore it for you. In fact Edit ▷ Undo will undo the last thing you did, whatever it was. <Ctrl>z is just a short cut for Edit ▷ Undo.

It is possible to tear off a menu so that it remains permanently on the screen. Click on Edit then above the dashed line at the top of the menu. The Edit Tear-off menu can now be dragged with the left mouse button in the top bar to a more convenient place on the screen.

3.6 Saving text in files

The text you are editing is usually a version of some file. The name of the file is shown in the title bar at the top of the Nedit window. Currently, it says (**Untitled**), because the text hasn't been saved to a file. The editor also has a current working directory. This is independent of the current working directory in the xterm windows – it has to be set separately. (This is a good thing – often you might want more than one xterm, and more than one Nedit window, each with different current working directories.)

To save your text into a file called lbjoke, pull down the File menu by clicking with the left mouse button on the File button, then select the Save As.. option. This will pop-up a window

and you should add `INTRO/ex3/lbjoke` to the **Save File As:** line, and finally click on the **OK** button.

This will save the text in your INTRO/ex3 directory as a file called lbjoke. It will also have changed the current working directory of Nedit to your INTRO/ex3 directory. Look at the top line of the text area, and you will see the full path name of the current file. The Nedit current working directory (for a particular Nedit window) is simply the directory containing the file you are editing.

Once the name of the working file has been set, as you have just done, you can choose File ▷ Save, without selecting a name, when you next want to save the file. *You should do this from time to time whenever you are editing a document, since any edits you don't save will be lost when you log out, or if the machine crashes.* Linux is more reliable than Windows, but nothing is perfect, and **Murphy's Law** (otherwise known as Sod's Law) tells us that the one time the machine will crash is 5 minutes before an important deadline...

Once you save a file two or more times, Nedit automatically keeps the previous version (provided Make Backup Copy in the Preferences Menu is enabled), adding .bck to the end of the filename. Try making a small change to your text, and saving lbjoke a second time. Then check using `ls` in an xterm window that you now have two files called lbjoke and lbjoke.bck. (If you cannot see these files, remember that you should be looking in your INTRO/ex3 directory!)

Note: when selecting a file to load into Nedit, it is rather too easy to accidentally select the backup copy of the file, instead of the main one. If you think about it, you will see why this is a very bad idea – many new students make this mistake and lose hours of work as a result. So please be careful!

3.7 Loading and including files

We will now see how to work with a document which is too large to fit onto the screen. The document we will use is the light bulb jokes file. Copy it by:

```
cp $INTRO/lightbulbs .      ←┘
```

You did type that in an xterm window, didn't you? And you were already in your INTRO/ex3 directory weren't you? If not, then please sort it out, and don't leave a copy of the file cluttering up your home directory!

If you haven't already done so, save lbjoke. Now load lightbulbs into the editor by choosing File ▷ Open, then selecting the appropriate entry in the pop-up window, then selecting OK.

The first part of the document will appear in the window. If it doesn't, ask a demonstrator for help. Now include the contents of the file lbjoke within lightbulbs as follows:

- Position the cursor between two of the light bulb jokes.
- Choose File ▷ Include File
- Select, in the pop-up box, the file name lbjoke and click OK.

3.8 Scrolling

When editing a large document, the Nedit window is only able to display part of the text. Changing the portion of text being displayed (usually described as moving around the text) is controlled by the scroll bar, which is situated on the right hand side of the window. A picture

of the scroll bar is shown in figure 3.1. The position of the slider within the scroll bar indicates where the text you are currently looking at is in relation to the rest of the text. So, currently, we are at the beginning of the file, and the part we can see is only the tip of the iceberg.

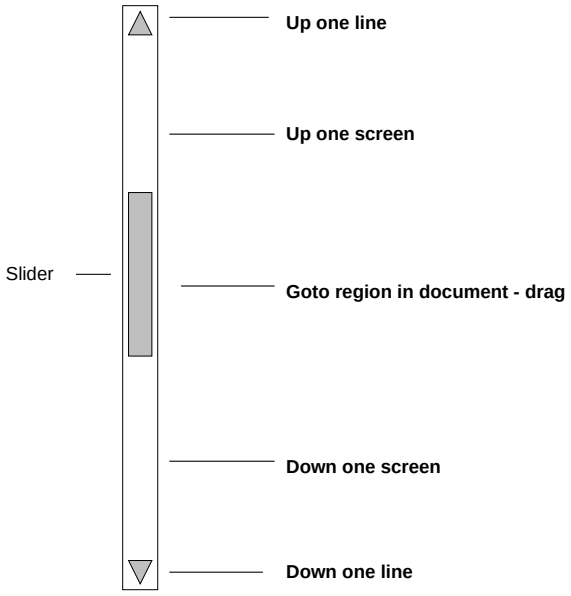


Figure 3.1: Scroll bar

We can move around the document by selecting the various components of the scroll bar, as described in figure 3.1. For example, selecting one of the **buttons** at the top or bottom will scroll up, or down, one line. The slider can be dragged to any part of the document by selecting, and dragging, the middle of the slider.

Experiment with moving around the document using the scroll bar and mouse buttons.

3.9 Searching

Suppose we want to find something specific. One way is to go to a line of the file with a particular line number; to do this, choose Search ▷ Goto Line Number and enter the appropriate line number. Now find the 142nd line of the document.

More common is to search for a string, i.e. a particular sequence of characters. For example, to look for all the light bulb jokes (LBJ's) containing the string "Three":

- Position the cursor at the start of the file (searching always starts from the current cursor position)
- Choose Search ▷ Find to display the Find window
- In the **String to Find:** box type `Three`
- Click on the **Find** button, to find the first occurrence of "Three"
- To find further occurrences, use the **Find Again** menu item (or the keyboard shortcuts: `<Ctrl>g` to search forward or `<Shift><Ctrl>G` to search backward).

Use this facility to find LBJs about doctors, Carl Sagan, and Manchester postgraduates. (Don't forget to move the cursor to the start of the file before each search.) Note that the string does not have to be a complete word, for example, try "ists".

3.9.1 Nedit Keyboard Shortcuts

Many of the actions which can be selected from Nedit's menus can also be invoked by using the keyboard. You have seen a few already. For example, your file can be saved by typing <Ctrl>s , rather than selecting the Save option from the File menu.

3.10 Find and replace

Often, we want to find a string in order to replace it with something else.

Choose Search ▷ Replace and the **replace** window will appear. This enables us to do various combinations of **Find and Replace** operations.

Suppose we wished to replace all occurrences of some string, for example to replace `light` by `dark` everywhere:

- Choose Replace to bring up the Replace Window
- Type `light` where it says **String to Find:**
- Type `dark` where it says **Replace With:**
- Click on the Replace All button.

Of course you can reverse this by replacing all `dark` with `light` (actually, this doesn't necessarily get you quite back to the original file – why not?).

Practice using the various options in the **Replace** window. The Regular Expression option is an advanced selection option which you will probably prefer to ignore for today.

3.11 Shell scripts

In section 1.7 we saw how we could use the shell to write simple programs via the command line, now we see how we can use files containing such shell scripts. Use `nedit` to create a file, called greeting, containing a slightly modified version of the lines you typed earlier, namely:

```
echo -n "Who are you? "
while read name; do
if test "$name" = "$USER"; then echo "Welcome $USER"
else echo -n "Who?"; fi
done
```

Save this file. Now make the file executable by using `chmod` to change its file permissions. Just

`chmod +x greeting`

Now execute it, by just typing its name in an `xterm` . You can find out much more about changing file permissions by looking at the man page for `chmod` .

3.12 Unix Driving Test

This section aims to test some of the skills that you have acquired in these introductory labs.

1. Copy the file $INTRO/test-first into a subdirectory of your `INTRO` directory called UDT (you will need to create this subdirectory).
2. Answer the questions given in this file and edit your copy to include your answers.
3. Send an email to John, with the subject 'Unix Driving Test Answers', attaching the file containing your answers.

3.13 Finishing

This is the end of the introductory Unix labs; if you have finished early, please go back and make sure you understand everything you've done. The skills you have been developing during these sessions will be very important to you in the future.

Don't forget to tell the laboratory supervisor you have completed this session.

Acknowledgements

This document was originally written by Graham Gough, John Latham, and Ian Watson, with additional material by Pete Jinks, Steve Pettifer and Toby Howard.