University of Manchester

Department of Computer Science

An Introduction to Unix for CS Foundation Year Students

Dr Thomas Carroll, February 2022

This document aims to introduce you to the Unix operating system and X window system which run on many PCs in the Department; some of the information contained in this document is specific to the setup here in CS, but most applies to any Unix system (eg: Linux, Mac OSX). 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 thomas.carroll@manchester.ac.uk.

Part 1: Introducing Unix

We imagine that the tasks in this document will take you 4-5 hours. 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, So please take your time!

If there's anything you feel you don't understand, please don't be shy: stick up your hand and ask us!

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 if you continue with us in Computer Science after your foundation 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

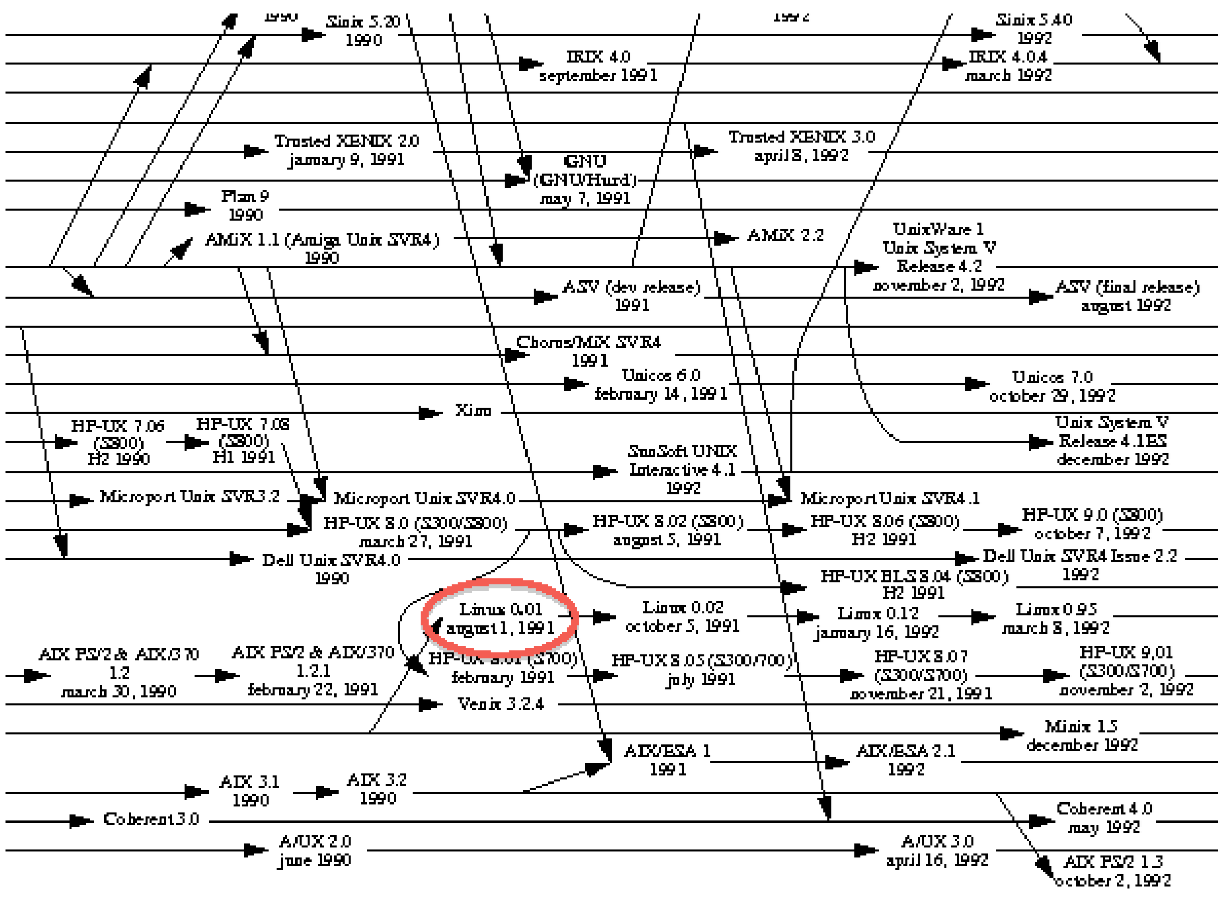

Figure 1.1: A fragment of Éric Lévénez's Unix History chart, reproduced with permission.

jealously. Figure 1.1 shows a tiny fragment of the Unix's 'family tree' (the full diagram is *many* 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 millennium to drop their own—highly popular, but ageing—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 unaware 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 Department 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 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 Departmental Unix network

The Department 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 Reading this document

In this document, things that you have to type appear in a typewriter font (`like this`). 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 *ell*, l, and a digit *one*, 1 – unfortunately these two characters look very similar in many fonts.

1.4 Getting Started

The first step is to log on to the machine.

You should be presented with a black screen with white text, saying “log in” or “username”.

If you can see a windows log in screen, then you need to restart the computer- please call over a member of staff who can help with this.

Simply type your computer username (this is the same that you use for the windows machines at university) and press ↲.

Now, type your password and press ↲. You may not see “stars” or “dots” as you type your password – don’t worry, your password is still being received.

Now, you should see a dark screen with pale text.

It should display the following text: `bash-4.2$`

this is known as the **prompt**, and it is where we can input commands via the keyboard to get the computer to do something for us.

We are actually now interacting with the computer within an environment known as the **terminal** or **shell** - it is a very powerful way of interacting with the computer. If you’re thinking that this is a trip back to the 70’s, then please don’t worry - you will soon find that using the terminal to perform tasks can often be much quicker and way more powerful than using your mouse in a graphical program!

Now, we can start the graphical desktop by typing in the following:

```
startx↲
```

You should now see a desktop appear!

Click “Applications” at the top left of the screen and search through the menus of programs.

Find the Firefox Web Browser, and open it to navigate to Blackboard. You can then coninue with this task either using the paper copy or electronic copy available on Blackboard.

You should now also click on the Application menu at the top left again, and look for a program called “Terminal” and open it. This will bring you to a terminal again, where we can continue with our work….

NB: To log out, click in the top right of the screen and find “log out”.

This will return to a terminal.

You must then type “exit” and press enter.

You should now practice logging out and logging in again, and going to your graphical desktop.

Part 2: Manipulating Files and Directories

This part 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.

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 MSDOS 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:, D:, for Floppy disk, Hard disk, CD respectively)).

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 in future years, if you continue your Computer Science career with us. 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, m12345ab's home directory is /home/m12345ab – in other words, the directory m12345ab 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 m12345ab who is working on INTRO, exercise 2, should have the directory /home/m12345ab/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/m12345ab/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/m12345ab/INTRO

the same file can just be called lightbulbs

while from /home it can be referred to as m12345aB/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.)

Please now ensure that you are logged on to the machine and you can see the prompt.

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 the terminal, type

```
pwd
```

and press ↲ (you should press ↲ after every command!) 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 (but maybe not identical to):

```
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 (but maybe not identical to):

```
.
.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 software environments available. 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.

2.4 Creating a directory structure

Firstly, we need to make sure we are in our home directory by using pwd. If we are *not* in our home directory, then type:

```
 cd
```

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` (short for “Remove Directory”) 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!).

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

```
home
        INTRO
                ex1
                ex2
                ex3
```

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.

Among the many things it does, it can set 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 (ie: this exercise).

Let's look at its value – type

```
echo $INTRO
```

echo is a command which just displays its arguments – very useful just here.

If the command echo $INTRO printed out nothing, then we need to do a very short task:

Type

```
nano ~/.bash_profile
```

This will open a text-editor like display in your terminal window, where you can see the contents of the file. Use your **down arrow keys (not your mouse!)** to move the cursor to the bottom of the file.

Now do a line break by typing ↵

Now type the following:

```
INTRO=/opt/info/courses/INTRO↵
export INTRO
```

Now, on your keyboard press [CTRL] and [X] together.
Nano will ask if you want to "save modified buffer" (this is the same as asking if you want to save the file).

Type [Y] and then press ↵, ensuring the “File Name To Write” is /home/m12345ab/.bash_profile (where m12345ab is your username).

Now you need to reload the .bash_profile for it to take effect:
In the same terminal window, type:

```
source ~/.bash_profile
```

Now test that it has worked by typing into a terminal

```
echo $INTRO
```

If shell variables didn’t exist, or if we hadn’t taken the trouble to define $INTRO (etc.) , you’d have to remember and type long path names like that one frequently – starting in the next section! T
here 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 : 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.

If no file called lightbulbs has appeared, the following will probably provide an explanation...
Ifit 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 ?

Now look for a file explorer program and open it; explore your home directory./ What do you notice?

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.

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. I**f 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 (gedit, nedit or nano) 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 gedit, nedit, or nano 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

```
ls *fred*
```

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

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 *bash*
```

and

```
ls .*bash*
```

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 at some point in the future! 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

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 concatenate 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 . (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

```
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 actually 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 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 thekeys at the right time.

Part 3: Text Editing

3.1 Computer text processing

This session will introduce you to Nedit , a simple but powerful text editor. The techniques you will learn today are fundamental, and you should practice them until you can do them without thinking, because you will use text editors a lot when you are writing computer programs.

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.. Other text editors like Nedit, nano, VS Code are available.
- A **text processor** takes text, with extra formatting commands, and formats it neatly, using a variety of fonts and formatting conventions. We will learn about such a system, called LaTeX (pronounced "lay- tek") in a future session.
- 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.

3.3 Creating a small piece of text

You first need to open nedit. Type `nedit` into your terminal (what happens if you press <CTRL>c in the terminal when nedit is running?

Now we will create a small text file.

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 (**backspace**) 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.
- Right click and 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 right click menu , and the text will reappear in the correct place.

The 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, the keyboard accelerators for these operations are:
<Ctrl>x for Cut
<Ctrl>v for Paste
<Ctrl>c for Copy

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 a terminal that is NOT running nedit, 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.

If you cut something **you didn’t mean to**, Undo (found in the right click menu) will **restore it for you**. In fact Undo will undo the last thing you did, whatever it was. <Ctrl>z is just a short cut for Undo.

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 terminal windows – it has to be set separately. (This is a good thing – often you might want more than one terminal, and more than one nedit window, each with different current working directories.)

To save your text into a file called lbjoke, click the Save button in the top right. This will pop-up a window . Do you recognise that you can now see (graphically) the directory structure that you created earlier?

You should navigate to INTRO/ex3/ using your mouse to click on directories and buttons, and then add lbjoke to the File Name 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 Save (or <Ctrl>s, 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…

nedit can sometimes keep the previous version as backup, 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 terminal 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 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. 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.

We can move around the document by selecting the various components of the scroll bar. 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.

3.13 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. You may wish to take a look at the exercises (at the end of the document) to improve your skill.

Acknowledgements

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

Section 4: Exercises

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

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. 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.