UNIX SURVIVAL GUIDE

Introduction

Welcome to the UNIX Induction Manual put together for students learning programming in the context of the UNIX operating system and its near relatives.

This manual will help you become familiar with the UNIX-like environment provided to you as a student in the school of Computer Science and Information Technology. While this manual provides a great deal of information introducing you to the Linux servers we are using in the school, much of this information will be useful throughout the semester. We encourage you to bring along this lab sheet to future classes for this course. This lab sets a baseline for the level of familiarity we expect from you with UNIX-like environments in order to complete the assessment in this course. If any of this material is difficult for you, we encourage you to act upon this early. Seek help as soon as you notice there is a problem. You will find the lecturer and head tutor for this course will have much advice to help you fill any gaps you may have.

We recommend that you complete all exercises in this manual that you don't have time to complete in class – we are more than happy to answer questions about this on the discussion board for this course or in later classes.

Please do not copy and paste sections from this document into your terminal window. There may be additional or fewer characters pasted into the UNIX terminal as they interpret characters differently.

UNIX

UNIX is a multiuser operating system that is commonly used in industry and has many advantages over other operating systems – some of which you will discover in this and later courses. While the Windows PCs and terminal servers are available for you to use, assignments for this course need to be completed on the school provided Linux servers which are at this stage:

jupiter.csit.rmit.edu.au, **saturn.csit.rmit.edu.au** and **titan.csit.rmit.edu.au**

Once you connect to either of these machines, their behaviour is much the same. Because of that, I will refer to jupiter throughout this guide as its behaviour is the same as saturn and titan.

What is Linux?

There are actually many operating systems that are UNIX-like. The coreteaching machines mentioned above run Linux. You may have heard of some others: Solaris, FreeBSD, Mac OS, SCO, and so on. While internally these operating systems are very different, they all share a common interface, meaning the skills you learn here are generally applicable to all UNIX-like operating systems.

UNIX Survival Guide

Logging In (from a Windows Operating System)
If you are connecting from a Windows operating system you must use a terminal emulator. The most popular program is called `putty`.

To launch `putty`, click on the start button located on the bottom left of the screen. Type `putty` and press enter. `Putty` will now launch and you will see something like the following:
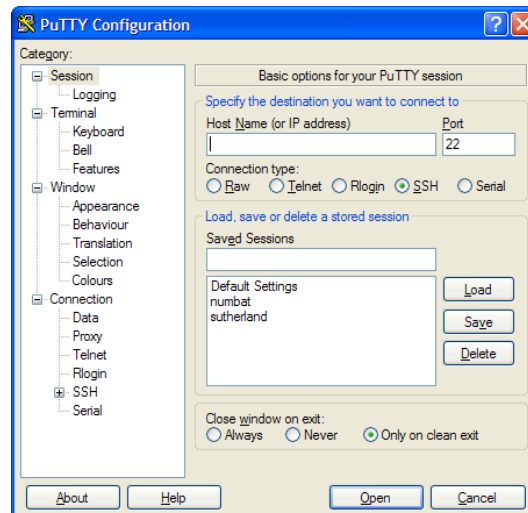

To connect to the UNIX servers, in the field "Hostname or IP address" type the address of the server you wish to connect to (one of the machines listed above):
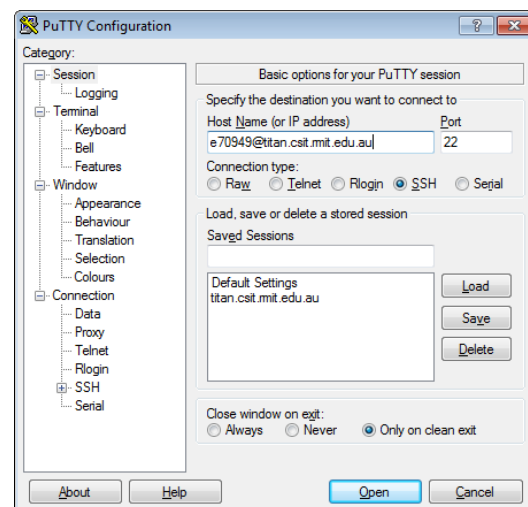
Then press enter or click Open. If this is the first time logging in with `putty` you will be presented with a security alert, as follows:
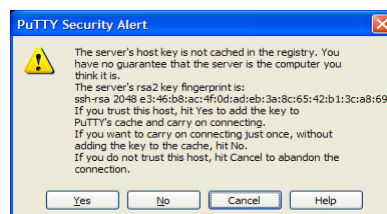

 This alert lets you know that it is the first time connecting from this computer to the server. To continue click "Yes" and then you will be prompted to login:

UNIX Survival Guide
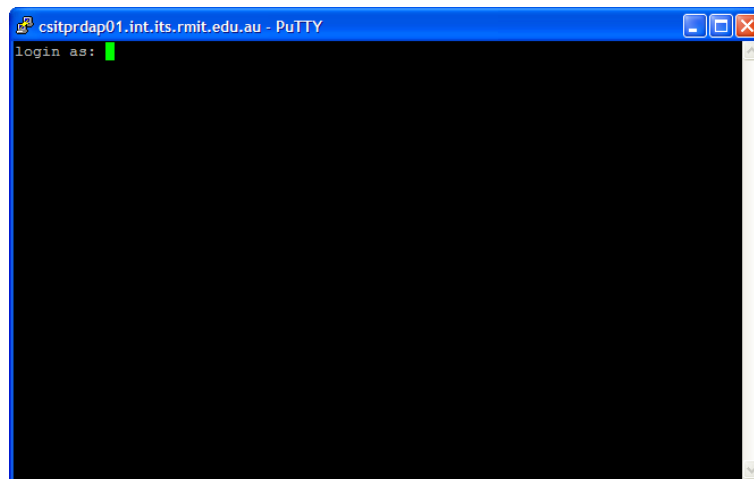
Enter your username ([s] followed by your student number) and press enter. You will then be prompted for your password. The credentials used to login are your NDS username and password – the ones you use to login at any workstation in the university's computer labs.

If you are experiencing issues logging in please go to the duty programmers desk or ask your tutor or laboratory assistant – they are there to help you.

Once logged in, you can scroll down to "using the terminal" and continue with the UNIX induction, which looks like the following (it won't be exactly the same as I am logged in as a staff member):
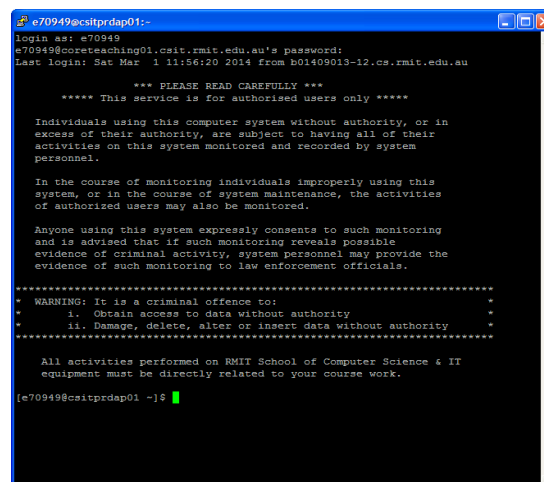
 If you would like to connect from home (on Windows) you may download **putty** from the following link:
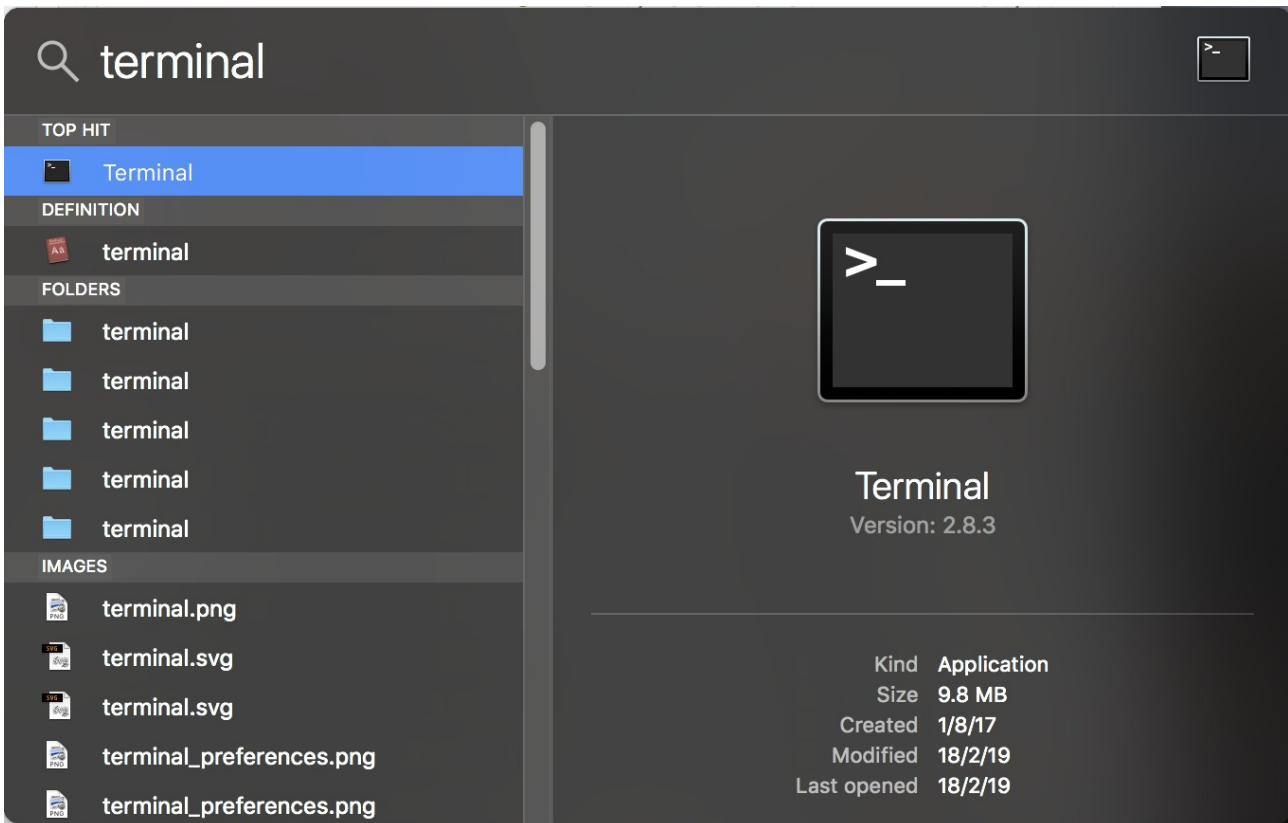
https://www.putty.org/

Logging In (from a "UNIX-Like" Operating System)

If you are connecting from a "UNIX-like" operating system such as OSX or Linux, you simply launch a terminal window. (This is built into the operating system.)

MAC OS: Top right corner click on spotlight, eg:

start typing terminal, eg:

Press [return] and the terminal application should start.

Linux: Press alt + f2 and type "gnome-terminal" or something similar – each Linux distribution will have its own program for running a terminal but generally if you are unsure, a simple google search will tell you what the program is. On Xubuntu Linux, I would type "xubuntu terminal program" into the search bar for my browser and look through the first few links, I would find that the default terminal program was `xfce4-terminal`. If you are using a kde based system, the terminal program will be called **konsole**. This will launch a terminal program. A terminal program is a command line interface that is capable of executing local system commands and connecting to remote servers. Please note that this will be different according to your distribution. If you are at all unsure, please ask your lab assistant.

The command used to connect to the school servers is `ssh`. It requires two pieces of information to connect to the server. The **hostname** or **IP address** of the server you wish to connect to and the **username** of the user you wish to login as (NDS username).

An example of the command used to connect to jupiter is:

ssh s1234567@jupiter.csit.rmit.edu.au

Make sure to substitute your username with the example username s1234567 (⌨s followed by your student number).

If you are experiencing issues logging in please go to the duty programmers desk or ask your tutor as soon as possible. Inability to login to the school servers is not grounds for an extension unless it is an issue that affects everyone. Once logged in, you can scroll down to using the terminal and continue with the UNIX induction.

Using the terminal

After the welcome message is displayed, you will see a prompt such as:

[username@csitprdap01 ~]$

This prompt indicates that the terminal is ready and waiting for you to enter a command. In this manual we usually don't show the prompt; just the commands you need to type. These commands will be in **bold** type. Output from commands will be in `this font`.

For example, try typing "date" and pressing enter:

date

This runs the program "date" and displays the output to the terminal. Most commands accept arguments, or parameters, which modify their behaviour. For example, if you type:

date -u

The output might be:
`Wed Feb 26 08:01:23 UTC 2017`

The `date` program is run with the argument `-u`, which then displays the time at GMT (a different timezone).

Note: When you need to repeat a command with the same or similar arguments, you can save typing by pressing the up-arrow on the keyboard. This shows the last command you entered. You can continue pressing the up and down keys to move back and forth through the history of commands you have typed in the current session.

You can also use the "history" command to get a listing of all previous commands in your shell sessions. You can also search through previous commands by pressing ctrl-R and then entering the text that as contained in the command you are after.

Getting help

When you know the name of a command, but can't remember what arguments it takes, you can look it up in the man pages (short for manual pages). For example, to find out all the options for the date program:

man date

You can scroll through the page with the spacebar and the up / down arrow keys. To quit viewing the manual page and return to the prompt, press ⌨q.

If you don't remember the name of the command, man won't help. Another option in this case might be the **apropos** command. Apropos means "related to" and so the command `apropos` will show you commands that are related to that word.

For example, try **apropos stdio**. Stdio is one of the "header" files used in the C programming language. We include header files in a source file when we want to let the source file know about a C function defined elsewhere.

apropos stdio

gives us output like:

```
DBD::Gofer::Transport::stream (3pm)  - DBD::Gofer transport for
stdio streaming
_fbufsize [stdio_ext] (3)  - interfaces to stdio FILE structure
__flbf [stdio_ext]   (3)  - interfaces to stdio FILE structure
flockfile            (3)  - lock FILE for stdio
flockfile            (3p) - stdio locking functions
_flushlbf [stdio_ext] (3)  - interfaces to stdio FILE structure
__fpending [stdio_ext] (3)  - interfaces to stdio FILE structure
__fpurge [stdio_ext] (3)  - interfaces to stdio FILE structure
__freadable [stdio_ext] (3)  - interfaces to stdio FILE structure
```

… and lots more. this might be useful when you know the header file but not the name of the function you wish to use, for example. The name is square brackets indicates the name of the header file. The number in round brackets is the "section" of the manual that the man page comes from. Sometimes there will be several man pages that refer to the same name but refer to different things. For example above `flockfile` appears in two different sections. If you want the second manpage you might need to type the following command:

man -s3 flockfile

The flag `-s` that we passed to man tells it which section to look in.

Files and Directories
Most people are familiar with Windows drive letters where disks are accessed as C:, D:, and so on. In UNIX there is no such distinction, instead it follows a hierarchical structure. All files, regardless of where they are stored, are accessible through the root directory or "/". Whereas on Windows directories (sometimes called folders) are separated in a path with a backslash ("\"), on UNIX you must use a forward slash ("/").

Here are some example directory paths:

```
/
/home
/home/el9/
/home/el9/e70949
/usr/local/bin
```

Paths are read from left to right. The first slash ("/") means to start at the root directory (you can start in some other places as well, as we shall see shortly).

You can use the **ls** command to list (that's actually what the command means) the contents of a directory:

ls /

This will print out all files and directories directly below the root directory. Some of these directories and their purpose are explained below:

Directory	Contents
/bin	Standard UNIX programs
/etc	Configuration files
/home	home directories
/tmp	Temporary files

Look inside the /home directory

ls /home

This directory contains every student and staff member's personal home directory organised first into subdirectories.

Now, try to list the contents of your own home directory. Your home directory is the place where you "land" when you first login to jupiter/saturn. As a shortcut to writing out your whole home directory path, you can simply write ⌐ (tilde character, in the top-left corner of the keyboard).

You are sharing the jupiter or saturn or titan machine you are logged into with many other staff and students, but there are systems in place that prevent you from reading or modifying other people's data (if they have the correct permissions in place), as well as data that could interfere with the upkeep of the system.

Ensure you are back in your home directory (type ⌨c⌨d and press ⏎) and type **ls** and press enter. You will notice that there are two subdirectories (actually symbolic links but we will get to them later) for storage space managed by the university. You are advised to **never** save files while editing to the Hdrive – the reason for this is that Hdrive is a windows based drive. Windows locks are different to unix locks and so it is possible to lose data through lack of care in this regard.

You will want to edit files in your home directory and back them up somewhere. Losing your files due to not backing up your files will not be grounds for an extension or exemption from assessment. Losing files when you work as a programmer is not grounds to get more time to do your job; the same applies in this course.
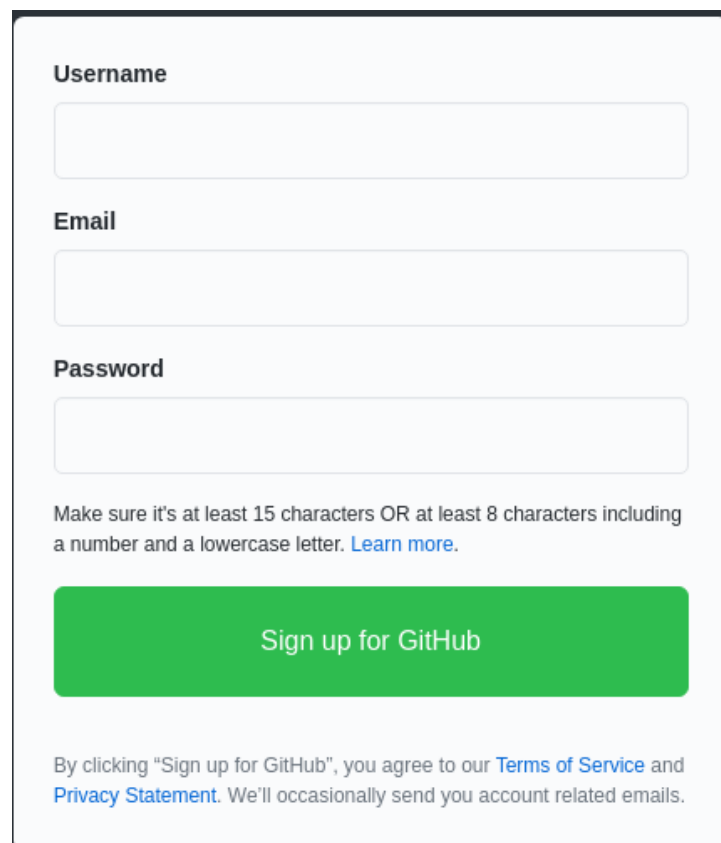
Test Yourself
1   What is the path to your home directory?
2   What files and directories exist in your home directory? Can you guess what they are?
3   Log into saturn now. Can you see any differences in either your home directory or the root directory? Why do you think that might be? Return to jupiter when you are done.

Backup and Recovery

We all have mishaps from time to time and need to recover. We recommend you become familiar with a "source control" system such as "git". Over the next few pages we will show you how to set up a **git** repository on **github**.

Go to https://github.com and sign up for an account:

Enter your desired username – it may be anything you could choose but ideally something you would happy for other people to see. Ensure that you provide your rmit student email address.
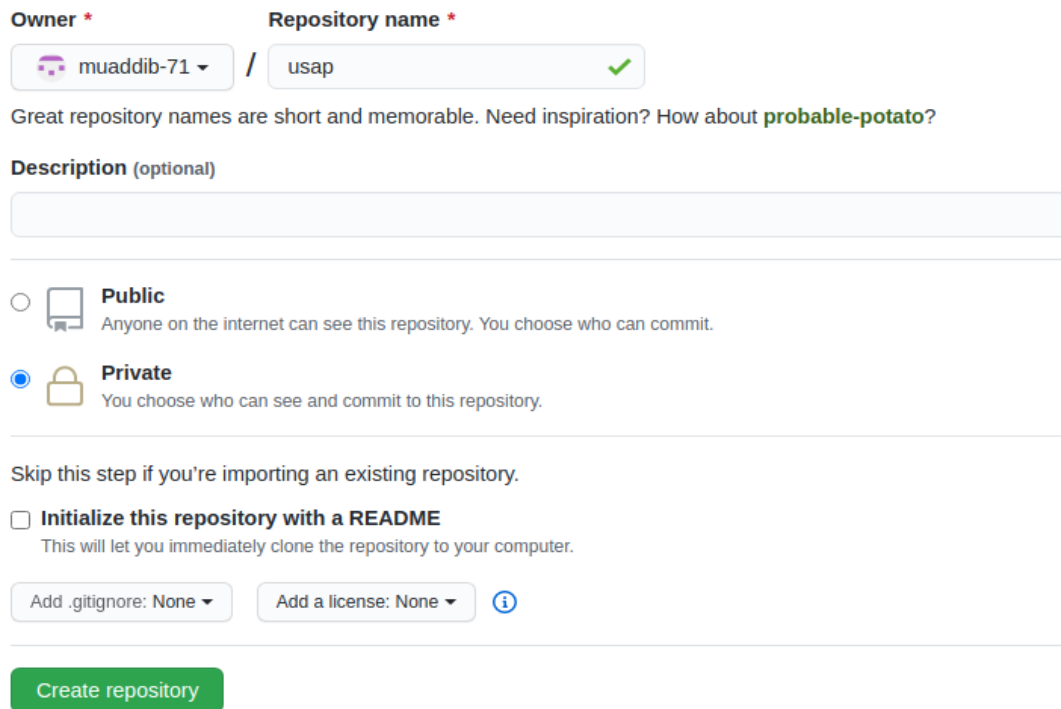
Follow the prompts to verify your account and the short survey on how you will use your account.

Next, go to your outlook email account and check for the validation email from github. Click the "veryify account" link and log in.

Finally "Create Repository" to create a new repository once you have verified your email account.

Follow the prompts and create a private repository called "usap" where you will synchronise all the work you do for this course.

The settings you choose for your repository should be something like the following:

Once, you are satisified, click the "Create repository" button.

Congratulations, you now have a repository.

On the next screen, you can start adding some content to your repository. Follow the instructions to create a local repository to push your content from.

You can find a git cheat sheet at https://github.github.com/training-kit/downloads/github-git-cheat-sheet.pdf

Please note that out of the box the services provided by github are a little limited. You will need to apply for an education pack later in the course, so you might as well do it now:

Go to https://education.github.com/students and click the button that says "Get benefits for students". Follow the prompts, provide any proof that is required.

Working directory

Every terminal window you have open has a current working directory. This is the directory that applications will load and save their data to or from by default. You can print the entire path to the working directory with the **pwd** command:

pwd

If the directory or file in which you are interested is in the current working directory, you don't need to specify a complete path to it. For example, you can list the contents of the current working directory by typing ls without any arguments:

ls

Creating directories
There is not much interesting in your home directory yet, so let's create some directories. To create a directory, use the **mkdir** command.

Before you type these commands, make sure your current directory is your home directory. You can type 'cd' from any location to change back to your home directory. Let's say you want to create a directory called "courses" within your home directory:

mkdir courses

Remember that since your current working directory is your home directory you didn't need to type in the whole path. List the contents of your home directory now and make sure you can see courses as one of the items. Let's assume you are taking 3 courses: "maths", "programming" and "databases", and create a directory under courses for each one:

mkdir courses/maths
mkdir courses/programming
mkdir courses/databases

Now list the contents of courses and make sure you can see all of these directories.

ls courses

Changing the working directory
Earlier we saw that the current working directory was your home directory. Let's now change directory to the "courses" directory:

cd courses

Now that the working directory has changed, what do **pwd** and **ls** do?

Create one more directory under courses named "induction":

mkdir induction

Note: that we didn't write `courses/induction` this time, as we are creating the directory directly within the current working directory. List the contents of the current directory and make

sure you now see all 4 directories. The layout of directories is often referred to as a directory tree, and is displayed like this:

```
/ (root directory)
bin/
home/
        sh1/
                s3030310/
                        courses/
                                databases/
                                induction/
                                maths/
                                programming/
lib/
opt/
tmp/
```

In this diagram you can see that the s3030310 directory is the parent of courses, which in turn is the parent of the four subject directories you created.

We changed directory from s3030310 to courses by typing:

`cd courses`

You can't simply change back to s3030310 from courses by typing "**`cd s3030310`**" this is because s3030310 is not visible from courses. You can change back to s3030310 by typing its relative path (a relative path is a path from the current directory whereas an absolute path is the whole path from the root directory) :

`cd ..`

(two full-stops, commonly pronounced "dot-dot"). The dot-dot can appear anywhere in a regular path to signify "the parent" or "one level up from here":

`cd ~/..`
`pwd`
`cd /home/../tmp`
`pwd`

Test Yourself
Write down the absolute path of the following paths (an absolute path is one that begins at the root directory):

`courses/`
`courses/../../`
`~/../..`
`courses/../courses/../courses`
Create two further directories under courses/maths named "calculus" and "algebra". Change directory to algebra, from there list the contents of courses.

Tab completion

When typing the names of files or directories on the command-line, you can often type just the first few characters, then press the tab key to fill in the rest automatically. For example, you can save a lot of typing when changing to the courses/maths directory by just typing:

```
cd c(tab)m(tab)
```

becomes

```
cd courses/maths/
```

Hidden files and directories
Change to your home directory and list all the files. Now add the -a option to ls:

```
ls -a
```

You should see about 20 extra files and directories that weren't in the standard listing. These are hidden files, and have a full-stop (".") as the first character in their name. Typically they are used to store application preferences and caches. Create a hidden directory in courses named ".hidden":

```
mkdir courses/.hidden
```

Make sure you can see it only when you use the "-a" option with **ls**.

At the beginning of the listing of hidden files in each directory are the two special directories "." and "..". We already know that ".." refers to the parent directory. The "." ("dot") directory refers to the current directory. Ordinarily this is not needed on the command line, but you may find you need it at some stage. Check now that the following three commands are equivalent:

```
ls courses
ls ./courses
ls ./courses/
```

Creating and editing text files
Most of the text files that you work on this course will be plain text files (they have no formatting or fonts like a Microsoft Word file, for example). There are many programs you can use to edit these files. One very powerful and flexible editor is vim, which is introduced in the next section of this manual. Regular practise is recommended so that you become proficient editing using vim. For now, however, we will use a much simpler editor called **nano** which only runs in a terminal. Change into the courses/maths directory and start editing a new file called "assignment1":

```
cd courses/maths
nano assignment1
```

Nano behaves much like any Windows text editor, though very basic. The commands for nano are listed at the bottom of the terminal window. The ^ represents control, for example to save a file press [Ctrl]+[O] (write out). Nano will prompt you to confirm the name of the file. Press enter to confirm. A file is now created with the name assignment1, to exit nano press [Ctrl]+[X].
To check if the file was successfully created use the list command:

ls

To display the contents of the file in terminal use the command `cat`:

cat assignment1

Renaming, moving, copying and deleting files
You should now have a file assignment1 in the maths directory. Let's say you wanted to rename it to "assignment1.txt":

mv assignment1 assignment1.txt

The first argument to `mv` is the original file name, the second is the name you would like it renamed to. `mv` won't actually output anything; you will need to list the current directory contents with `ls` to check that the result is what you expected. In general, this is the approach of UNIX applications and is part of the "UNIX philosophy":

"If you can't say anything bad, don't say anything at all."

You can use the same command to move a file to another directory:

mv assignment1.txt ../induction

This moves the file assignment1.txt to the directory ../induction. Remember that the double-dots ("..") indicate to start from the parent directory, which in this case is courses. Similarly, you can move entire directories with the same command:

mv ../induction ~/

This moves the induction directory to your home directory (remember that the tilde ~ is short-hand for your home directory). The cp command works similarly, except that instead of moving or renaming a file it makes a copy:

cp ~/induction/assignment1.txt ./

This makes a copy of assignment1.txt and places it in the current working directory (remember that "." means to start from the current directory), which if you have not changed it is the maths directory. If you want to copy an entire directory you need to specify the "-r" argument to cp:

cp -r ./ ../maths-backup

This copies the current working directory (maths) to the directory ~/courses/mathsbackup. To delete a file, use the rm command:

rm ~/induction/assignment1.txt

To delete an entire directory, add the `-r` argument to `rm`:

rm -r ../databases

Careful! If you delete something you need there may be no way to get it back. The mv and cp commands won't give you any warning if you overwrite another existing file. This is also part of the

UNIX Survival Guide

UNIX philosophy. It is assumed that the user knows what they want and the operating system will obey the commands so long as they don't reduce the security of the system.

Test Yourself
Rename the maths-backup directory to make it hidden. Hint: how does the name need to change to make it not show up in a normal directory listing? Check with ls that it doesn't show up, then show that is there when you supply the appropriate argument to ls.

A note about UNIX file name conventions
You will have noticed that most of the directory and file names given in this manual are composed entirely of lower-case letters. This is entirely optional: UNIX systems allow filenames to have any form of letter, including most punctuation marks and characters from non-English character sets. Unlike Windows and DOS, however, UNIX treats upper- and lower-case letters as different. In other words, you can have a directory containing the files test.txt, TEST.TXT and Test.TXT, and they would all represent different files. As you can imagine, this can get quite confusing. For the sake of simplicity and clarity, most users elect to name their files entirely in lowercase English letters, with the addition of the hyphen ("-"), full-stop (".") and underscore ("_") punctuation marks. Some programs may not work correctly with other punctuation marks in the filename, as they have special meaning to the command environment, as you will see. You will have also noticed that unlike Windows, many files do not have an extension (like .txt). Again, these are optional, however they can help you to organise your files and they let programs know what kind of data to expect.

Remote Copying of Files

Please note that if you are on a mac or have install WSL(Windows Subsystem for Linux) on windows, or even on a Linux computer, scp is probably a better command to use.

Your can use scp as follows for a single file:

scp filename user@host:destination, eg

scp windows.txt s1234567@titan.csit.rmit.edu.au:uploads/

would upload windows.txt to the directory "uploads" which is a subdirectory of your home directory.

You could upload a whole directory by using the -r flag which stands for recursive, eg:

scp -r myproject s1234567@titan.csit.rmit.edu.au:

Using Windows files on UNIX
Often you will want to work on an assignment at home on Windows, then copy it back to the jupiter/saturn for submission. Probably the best way to copy files to the server from your windows machine is via a program called "winscp". You can get a copy of this from http://winscp.net/

There is some sample files for this guide available for download from the following url: http://saturn.csit.rmit.edu.au/~e70949/induction.zip. Download this file (a .zip file) then upload it to jupiter using winscp as follows:

Start winscp on your computer by searching for it the same way as you did with `putty`.

UNIX Survival Guide

When you first run it, you should get a window that looks something like this:
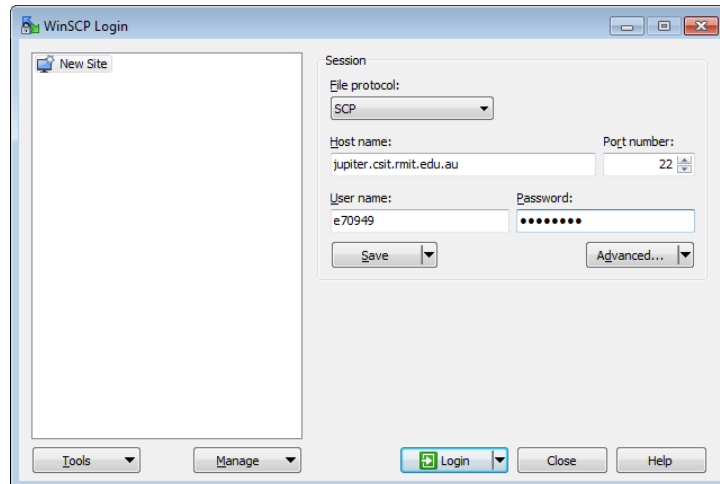
Start by changing the file protocol to "SCP" then type the hostname for your preferred server, your username and password, as per the credentials you entered earlier. Then, click "Login".

You may once again get a confirmation message if this is the first time you have connected to the server:
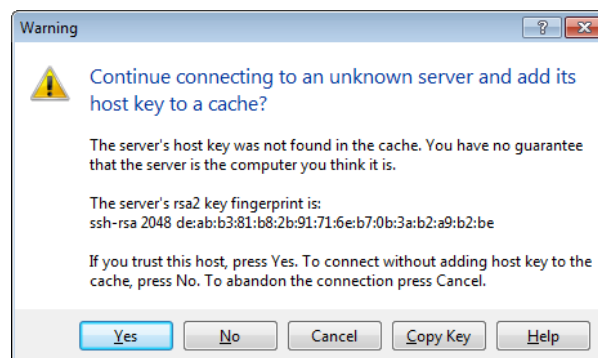
Click "Yes" and you will continue to be connected to the server.

UNIX Survival Guide

At which time, you will get a window much like the following:
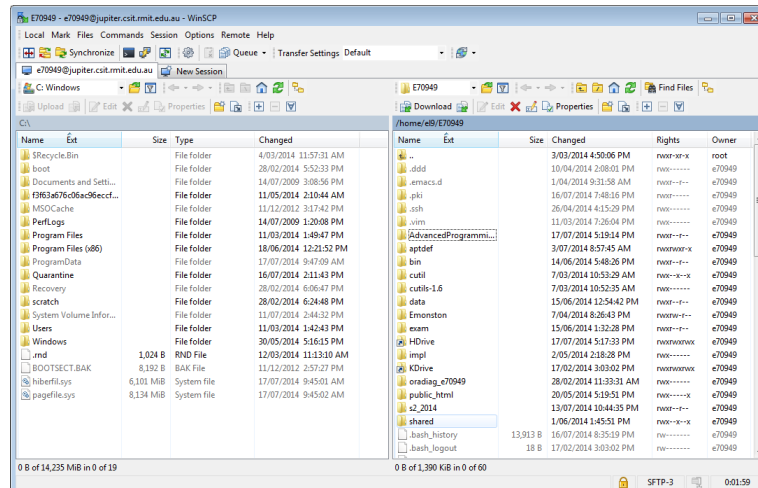
Use the navigation window on the left hand side to browse to where you saved your zip file and then drag it across to the right hand pane. You will then see a confirmation screen as follows:
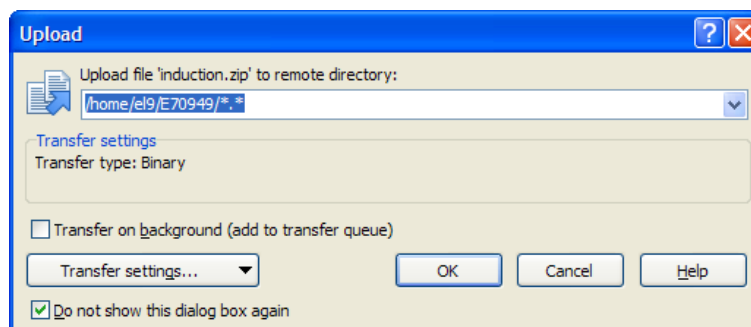
Click "OK" and your transfer will begin.

Please do not use the winscp editing window to edit your source files. The window is a very simple interface for editing small amounts of text; it is not a complete text editor for software development. We recommend using one of the text editors we discuss later in this guide.

Copy this file you have just uploaded to your induction directory, and change to that directory and extract the newly uploaded zip file:

```
cp induction.zip induction
cd induction
unzip induction.zip
```

This command will uncompress the compressed file and save each file in the zip file in the current directory. You may now display the contents of windows.txt file in that directory as follows:

```
cat windows.txt
```

Now try using cat with `-v` argument.

```
cat -v windows.txt
```

What's wrong with the text? Can you see the extra ^M characters at the end of each line? In Windows and DOS, each line of a text file is terminated by two characters: a carriage return (CR, code 13) followed by a line-feed (LF, code 10).

In UNIX, lines in a text file are terminated with just a line-feed. Some UNIX programs can handle both types of file, but most will not work correctly with the Windows style line-endings. This is particularly the case when trying to compile C programs. There are a range of characters added by windows text editors that gcc will react badly to. The best approach is to avoid these altogether.

In `cat -v`, the carriage returns appear as ^M characters, which is harmless but annoying. In Perl (a programming language), however, a program written with these characters will just fail to work. This may also affect some .c files compiled with gcc. We are better off just to convert the file.

This can be quite a shock to a student who has worked all weekend at home on an assignment only to find it does not work at all at uni. Luckily, the solution is simple: simply remove the carriage returns from the file. You can use a program called tr to remove these unwanted characters:

```
cat windows.txt | tr -d "\r" > UNIX.txt
```

This converts the file `windows.txt` and saves it as `UNIX.txt` by deleting the '\r' characters (carriage returns) that are added by windows text editors.

Check that this has worked using `cat -v`. Please note that the source file and the destination file cannot be the same – you might need to ouput to a temporary file and then move that file back to its original name.

Uploading files between UNIX-like environments
It is common that you might need to copy files between two unix-like environments and this is particularly true if you are using an apple mac or a linux or other *nix as your home computer. In order to achieve this we use a command, scp (Secure Copy). This will also work if you are using windows bash in Windows 10.

If I wish to copy a single file to the server the syntax is:
scp file user@server:/path/to/destination

For example:

```
scp myfile e70949@titan.csit.rmit.edu.au:foo
```

would copy "myfile" to a directory called foo contained in my home directory. Please note that if you want to copy a directory structure you must pass the "-r" argument to scp. Also, the ":" is important. If you leave it out, the scp program will just make a local copy of the file as it does not recognise it as a network address.

Filename globbing

List the contents of the induction directory. There are a series of files ending in .ant and .syn. These are lists of words that are antonyms or synonyms of the filename, respectively. If you were to copy all of the synonym files (those ending in .syn) into your current directory, you would need to type the cp command 8 times. Actually, there is an easier way:

Let's start by creating two subdirectories of the induction directory, synonyms and antonyms:

```
mkdir synonyms antonyms
```

Now, we want to copy all the synonym files to the synonym directory:

```
cp *.syn synonyms
```

The "*" character (an asterisk, commonly called a "star") is a placeholder, or "globbing operator" for any sequence of characters. You can use it in place of part of a filename where you want to list all the files that match the pattern. It works on all commands, not just `cp`:

```
ls induction/*.ant
ls ./d*
```

Warning: using a glob is equivalent to typing out all the filenames it matches in sequence. This is not always intuitively what you want. Consider what happens when you type:

```
mv *.txt
```

The most likely problem here is that you have left out the destination directory from your mv command. This is equivalent to typing (assuming there are two files in the directory):

```
mv file1.txt file2.txt
```

Instead of moving these two files elsewhere, you have overwritten file2.txt with file1.txt.

Finding text within files

You should now be in a directory with several .syn synonym files. If you look at these files with nedit you can see that they are just simple text files with one word per line. You can search a file for a word or phrase with the grep command:

```
grep perplexed *.syn
```

This searches all files matching the glob *.syn for the word "perplexed". If any are found they will be printed to the terminal. Specifying the "-n" option also causes the line number that the word was found on to be printed:

```
grep -n perplexed *.syn
```

This is a particularly useful feature when you start dealing with large amounts of source code, and need to find where a particular variable or routine is being used.

Finding files within directories

grep is very good for searching within files, but it doesn't help when you know the name of a file but can't remember which directory it is in. Do you remember where the assignment1.txt file is? You can use the find command here:

find ~/ -name assignment1.txt

Note that the arguments are quite different from grep. First, you specify a directory where the search will start. You could specify the root directory ("/"), but that would take a long time; here we start from the home directory. The -name option instructs find to show only files with the given filename.

You can use a glob with find to locate files with just part of the filename, but you must then surround it in quotation marks:

find ~/ -name "*.txt"

Another equivalent option to the above command would be to 'escape' the "*" by using the backslash as follows:

find ~/ -name *.txt

What this means is to ignore the special meaning of the "*" and just treat it like any other character.

Test yourself

Copy all the antonym files (those ending in .ant) from induction into your **induction/antonyms** directory.

What files contain the word "satisfied"?
What is "addlepated" a synonym for?
Where under the /usr/include directory is there a file named png.h?

Disk usage

You have a limited amount of disk space on the file servers. You can check your current usage with the quota command:

quota -vs

The "blocks" column shows the amount of disk space you are currently using, in kilobytes. The "limit" column shows how much disk space you are permitted before limits are enforced (also in kilobytes).

The second line of the quota command output has the form of a series of numbers separated by dots, followed by a colon (:) and a file path. This is because the quota command on this system sends a network request to check your quota.

The -s enforces human readable units rather than representing everything in the number of bytes and the -v flag specifies more detailed (verbose) output.

UNIX Survival Guide

When you are approaching your limit (or are over it!) you will need to delete files so you can continue working. It can be helpful to see which files or directories are taking up the most room. You can use the du command for this:

```
du -ks ~/*
```

This shows the size of each file and directory (directories recursively include the size of all files and directories within them) in kilobytes in your home directory. Don't forget to check for hidden files:

```
du -ks ~/.*
```

Note that this also lists the total size of the parent directory (".."), which you can of course ignore.

An alternative to the 'k' flag is the 'h' flag. This presents sizes in human readable units (kilobytes, megabytes, etc) rather than just in units of kilobytes which is what the 'k' flag does.

Compressing, archiving and extracting files
When you are running out of disk space, an easy way to reclaim some space is to compress files you don't use on a day-to-day basis, but don't want to delete (such as old assignments).

The most common way for compressing files on UNIX is using the programs tar and gzip. tar creates one file that contains many files, and gzip, reduces the file size of that file. Thankfully **tar** can do this all in one step:

```
tar -czf backup.tar.gz *.syn
```

We pass three options to tar: **-c** means to create a new archive, **-z** means to compress it with gzip, and **-f** is used directly before the name of the file to create. Finally, *.syn is the list of files to **backup.tar.gz** is the standard extension for files made this way, though you may also see **.tgz** sometimes.

We can see a list of the files in an archive with the -t option:

```
tar -tzf backup.tar.gz
```

To extract the contents of the archive into the current directory, use the -x option:

```
tar -xzf backup.tar.gz
```

Besides gzip, some people are starting to use bzip2 compression on their files instead; this almost always makes files smaller. Archives created this way typically have the extension `.tar.bz2`, and you can work with them by using the `-j` option instead of `-z`:

```
tar -cjf backup.tar.bz2 *.syn
tar -tjf backup.tar.bz2
tar -xjf backup.tar.bz2
```

On Windows it is more common to use zip files. You can work with them on UNIX as well with the zip and unzip commands:

```
zip backup.zip *.syn
unzip backup.zip
```

Note that none of these commands deletes the original file(s); typically if you are trying to save space you would delete the files after creating the backup and checking that its contents are correct.

File permissions
Earlier we saw that certain files and directories (such as those belonging to other students) cannot be read and gave the error message "Permission denied". UNIX file permissions are quite complicated but it is essential you understand the basics of them so you know how to protect and share your files appropriately.

First, let's look in more detail at the files in your home directory, by adding the -l (lowercase "L") flag to **ls**:

ls -l ~
```
drwx------ 2 s1234567 students 80 Jan 24 10:59 Mail
drwx------ 2 s1234567 students 80 Jan 24 10:59 News
drwxr-xr-x 4 s1234567 students 1024 Feb 2 14:05 WINDOWS
drwxr-xr-x 5 s1234567 students 1024 Feb 10 18:05 courses
drwxr-xr-x 5 s1234567 students 1024 Feb 10 19:21 induction
-rw-r--r-- 1 s1234567 students 12 Feb 10 21:31 UNIX.txt
```

Instead of just listing the names of the files, we now have a detailed listing of the files.

The columns, from left to right, are:
```
drwx------
```

The permission bits for this file or directory. These are explained in great detail below.

```
2
```

The number of hard links to the file or directory. You can probably ignore this number for your entire career.

```
s1234567
```

The owner of the file. **That's you.**

```
students
```

The group that the file belongs to. Groups are described below.

```
80
```

The size of the file, in bytes. Note that for a directory this does not include the files within it, merely the amount of space the directory itself is taking.

```
Jan 24 10:59
```

The date and time the file was last modified.

```
Mail
```

The name of the file or directory.

When you access a file, you are classified into one of three categories with respect to the file:

- You are the owner of the file.
- You belong to the group that the file belongs to.
- You are someone else.

To see what groups you belong to, use the `groups` command:

groups

You may be added to more groups for certain courses, or to access a particular resource such as a CD burner – those who have access to the cdrom drive on a Linux computer are often members of the `cdrom` group. While you can belong to many groups, a file can only belong to one group. By default, all the files you create will belong to the students group. Now look closely at the permission bits for the last file in the earlier list:

`-rw-r—r--`

Ignoring the first hyphen (it is a d for directories, hyphen for files), you can divide the remaining characters into three sets of three characters:

`rw-`

`r--`

`r--`

Each of these sets corresponds to the rules to apply for a user falling into the respective category listed above. The first set is for the owner of the file, the second for a user belonging to the group that the file belongs to, and the third set is for everybody else.

Each set can have the letters r (read), w (write), or x (execute) set. If a letter is not set, a hyphen ("-") is displayed in its place. The meaning of these letters depends on whether the file is a directory or a regular file:

File Type	r	w	x
Regular (Normal) File	The contents of the file can be read	The file can be written to or replaced.	The file is a script or program and can be executed.
Directory	The contents of the directory can be listed.	Files can be added to and deleted from this directory.	The user can change to the directory and can access files within this directory.

So, for the permission bits:
`-rw-r—r--`

The owner of the file can read and write it, members of its group can read it, and everyone else can also read it. The three categories that the permission bits address are called user (for the owner),

group (for members of the group) and world or other (for everyone else). So we would say the above file is world-readable and user-writeable.

You can use the `chmod` command to change the permissions of a file or directory that you own:

chmod g+w UNIX.txt

The g refers to the group category of users, + means to add permission, and w refers to the write permission bit. In other words, it gives write permission to members of the group. The resultant permission bits will be: **-rw-rw-r--**

chmod ug+x UNIX.txt
This adds the execute permission bit (x) for the owner (u) and group (g). You can remove permissions with a minus sign:

chmod ugo-x UNIX.txt
This removes the execute permission bit (x) for the owner (u), group (g) and others (o). So `chmod` alters the permission bits for files and directories. The left-hand side selects which users to apply the changes to (u, g or o), the right hand side selects what permission bits to change (r, w or x), and they are separated by either a + sign, to add the permission, or a - sign, to remove the permission.

We can also specify permissions in octal. Octal is a numbering system with 8 possible values for each digit from 0-7. We can represent the permissions on a file using a sequence of three (or more) octal numbers. The rights for each category of user (user, group, others) can be represented as a sum of:

Permission type	Number to represent it.
Read	4
Write	2
Execute	1

In other words, a file with a permission of 644 can be read and written to by the owner, but only read by accounts in the same group and only read by other accounts. Likewise, 755 would mean that the owner can read, write and execute the file, and all others can read and execute the file.

Note that Windows does not have the same security model as UNIX, so all permissions on files will be lost when you transfer them to a Windows computer (or a windows file server). When copying files from a Windows computer to UNIX, you will often find that all files have all the permission bits set for all users.

Try copying a file from your home directory to the `HDrive` linked directory. If you then change into that directory and type **ls -l** you will see that it now has become executable regardless of whether it was executable before or not.

Test yourself

1. How much disk space do you have left, in megabytes?
2. Which directory or file is taking up most of the space in your home directory?
3. Compress that file or directory with tar/gzip, tar/bzip2 and zip. Which method gives the best result (Hint: you could use either du or ls -l to check the file sizes)?
4. Who owns your home directory? Does this mean you can change the permission bits on it?
5. Create a new text file in your home directory. What permission bits does it have?
6. Can another student read it?
7. Why or why not? (Hint: you may need to look at the permission bits for the directory it is in, as well as the file itself).
8. Change the permission bits on the file so your friend cannot read it, if they could; or change them so they can, if they could not. Please ensure at the end of this exercise that no files in your home directory are readable by other students.
9. Set appropriate permissions on your courses directory so that other students cannot read the files within it, or even see what files are in it. Get your friend to check for you that they cannot access it.

Introduction to Vim

Vim is a powerful and flexible text editor used by users of UNIX around the world. It can be quite tricky to learn at first, but once mastered it is an invaluable tool when editing configuration files, programming and writing reports. There are so many features of Vim that there is probably not a single person alive who knows them all. Everybody has a set of commands that they use themselves though, and you will need to find out which commands suit you the best.

`vim` is an enhanced version of an older text editor called `vi`. Both `vim` and `vi` are textonly; they run directly in the terminal window (also making the suitable for use over an ssh connection).

Command mode editing
Unlike other text editors you have used, vim has two modes: command and create. You can only type text while in a create mode. While in command mode you can load and save files, do searches and replacements, import other text files, and so on.

To start `vim`:

vim file1.txt
When `vim` starts you will be in command mode with a blank file. To start typing, press ⌨i (for "insert"). Type a few sentences of nonsense, then return to command mode by pressing ⌨esc (the escape key). In summary:

Type ⌨i to enter insert mode, where you can type text.

Press ⌨esc to leave any create mode and enter command mode.

Saving and exiting
To save the file you are working on, in command mode type ⌨: ⌨w (that's a colon, followed by a lower-case ⌨w for "write") and hit enter.

To quit, in command mode type ⌨: ⌨q. You can actually save and quit in one smooth move by typing ⌨: ⌨w ⌨q.

If you try quitting without first saving, Vim will show a warning that the file is not saved. You can override this with ⌨:⌨q⌨!, which means "quit, don't save, I know what I'm doing."

Help me, what's going on?

Sometimes a simple typo can make you feel hopelessly lost in Vim as it activates a feature you have never heard of. In almost all cases, you can simply return to command mode by hitting ESC two or three times. Similarly, you can undo the last command or insertion by hitting u (for "undo").

More creation modes

You have seen i, which starts insertion mode wherever the cursor is. There are a couple of shortcut keys for starting insertion in different places relative to the cursor:

Command	What it does
⌨a	begins inserting text just after the cursor.
⌨A	begins inserting text at the end of the line.
⌨i	begins inserting text just before the cursor position.
⌨I	(capital i) begins inserting text at the beginning of the current line.
⌨o	"opens" a new paragraph below the current line.
⌨O	(capital o) "opens" a new paragraph above the current line.

Try each of these now to get a feel for how they work; they can save a lot of time that in any other editor would be spent moving the cursor.

Moving the cursor

Speaking of moving the cursor, `vim` has many ways of allowing you to move the cursor around. Here are just a few (note that with the exception of the first two you need to be in command mode):

You can use the arrow keys as with any other text editor to move the cursor around. The ⌨w and ⌨b keys move forward and back one word, respectively. The ⌨(and ⌨) keys move to the previous or next sentence. The ⌨{ and ⌨} keys move to the previous or next paragraph. The ⌨0 (zero) and ⌨$ keys move the the start and end of the current line. Press ⌨Ctrl+⌨U and ⌨Ctrl+⌨D to scroll up and down half a page at a time. The ⌨PgUp and ⌨PgUp keys scroll up and down a whole page at a time. You can move to a specific line-number by typing the number in and pressing G. For example, to go directly to line 48 you would type ⌨4⌨8⌨G. This becomes very useful as you start programming as most errors are reported with the line number on which the error was found. All of the above movement commands can be extended by specifying a number of times to repeat the command. For example, typing ⌨5⌨w moves the cursor forward 5 words. Typing ⌨3 then ⌨↓and moves down three lines. The same approach works with moving the cursor with the other arrow keys while in command mode.

Deleting text

While typing text in a create mode you can delete straight away as usual with the delete and backspace keys. In command mode you have a few more options: Press `x` to delete the character the cursor is highlighting.

Press `d``d` (you press `d` twice) to delete the current line.

Press `d` and one of the movement keys listed above to delete that amount. For example, typing `d``w` deletes one word. Typing `d``3`) deletes three sentences.

Make a selection with the mouse and press `d` – this will delete the entire selection.

Pasting text

Vim does not have separate "delete" and "cut" commands. After you delete something, it is immediately available to be pasted.

Press p to paste text just after the cursor.
Press P to paste text just before the cursor.

An easy way to fix those typos where you swap two letters around (e.g. in "teh") is to position the cursor at the first of the pair of letters and press `x``p` in succession. An easy way to swap two lines of text is to press `d``d``p` in succession.

Copying text

Vim calls copying "yanking". After you have "yanked" some text you can paste it with one of the commands above.

To yank (copy) the current line, press `y``y` (press `y` twice). As with the delete command, you can use y with a movement. For example, `y` and the down key copies two lines; `y``2``}` copies two paragraphs.

Make a selection with the mouse and press `y` to copy.

Replacing text

These are shortcuts to deleting text and then inserting new text. Press `r` to replace a single character. For example, pressing `r``b` replaces the character under the cursor with a `b`, and then returns to command mode.

Press `R` to replace a lot of text. This enters a create mode where every letter you type overwrites the existing text instead of inserting. Press c and a movement to change some text. For example, `c``w` deletes one word and then enters insert mode, perfect for changing just that word.

Make a selection with the mouse and press `c` to delete the whole selection and enter insert mode.

Indenting text

When you start programming you will find that having well-indented source code is invaluable (both for readability and getting reasonable marks on your assignments!). To indent the current line, press `>``>` (the right angle bracket twice). To unindent the current line, press `<``<`. You can use `>` or `<` with a movement. For example `<``}` will unindent the whole paragraph. Make a selection with the mouse and press `<` or `>`. This will indent / unindent the selected text.

Searching text

Check that you are in command mode first (press ⎋). Press ⟋ (forward-slash) and type the word or phrase to search for and press ⏎. For example to search for "checker" you would type /checker and hit ⏎.

To move to the next search result press Ⓝ.
To move to the previous search result press Ⓝ.
Note that searches can contain complex regular-expressions. If you don't know what a regular expression is yet, make some time to find a tutorial. You can search for occurences of the word the cursor is in simply by pressing ＊ (asterisk) when in command mode.

Replacing text

Usually you will want to do a text replacement over an entire file. In command mode, type:

```
:%s/original/replacement/g
```

and hit ⏎. An explanation of this command follows:
The colon is used before most complex commands consisting of more than one or two characters. The percent sign ％ signifies that the command is to act over the entire file. There are ways (not discussed here) of restricting it to just a section of the file. The ⓢ stands for "substitute" and is borrowed from sed's language (see the references at the end of the UNIX section of this manual). The ⟋ (forward slash) separate the original text that you are searching for with the replacement text. The ⓖ option at the end stands for "global", and instructs vim to make the replacement for every occurrence of the original text on each line. Without this flag, only the first occurrence on each line is be changed. The original text can be a regular expression, and the replacement text can contain back-references into that expression, making for a very flexible substitution scheme.

See the vim manual for examples on usage.

Getting help

Vim's online help is very comprehensive, sometimes a little too comprehensive. You can access the table of contents by typing:

```
:help
```

or search for a particular feature (say, the substitute command described above):

```
:help :s
```

Close the help window with:

```
:q
```

Vim configuration

You can create a Vim configuration file to store your preferences in your home directory. The file should be called .vimrc (note the full-stop at the start, making it a hidden file). Create this file now with the following contents:

```
syntax on
set autoindent
set ts=4 sw=4 et si
set whichwrap+=<,>,[,]
set backspace=indent,eol,start
set hlsearch
```

These options turn on some nice user-interface features, such as allowing the cursor to move anywhere, highlighting search results, replacing tabs with spaces, automatically indenting source code sensibly, and highlighting source code according to its programming language.

Congratulations!

If you have made it this far you are well on your way to becoming a Vim grand master. Remember that the commands introduced here represent less than 1% of Vim's functionality. Browse through the tips at http://www.vim.org for ways other people are using Vim. In particular you might want to investigate the use of vim plugins. For example, I have several plug-ins installed in my vim set-up that use clang for auto-completion and code formatting.

Where to go for more information

This manual represents just a sampler of the programs installed on the school's Linux servers. Every single program presented has a myriad of options for customising how data is processed and formatted. Shell scripting in bash can be extremely flexible and goes far beyond the material presented here.

Any time you spend learning more about UNIX will easily repay itself when you come to do assignments, especially using shell scripts to automate repetitive tasks. The following resources are good starting points for investigating the programs here that interest you.

Writing and Compiling your first C Program

Let's create our first C file:

```
vim first.c
```

Now, in the text editing window you have available enter the following C program:

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

int main(void)
{
    printf("my name is %s\n", "enter your name here");
    return EXIT_SUCCESS;
}
```

Now, save this program and exit. We will now compile the program.
Type the following command to compile your program:

```
gcc -ansi -Wall -pedantic first.c -o first
```

We just want you to get the experience of compiling and running a program at this point. So, to run it, type the following:

./**first**

This means you want to run a program called first which is located in the current directory.
Please note: you need to specify a more modern compiler for some of the programming features we might use. You can do this as follows:

scl enable devtoolset-7 bash

Additional Resources

Advanced Bash-Scripting Guide

http://www.tldp.org/LDP/abs/html/

Easily the best online tutorial and reference for doing anything and everything with Bash.

Getting Started with awk

http://seismo.berkeley.edu/~rallen/resources/UNIXcmds/awkqref.html

Awk is a unique programming language for processing text files as collections of data records, making it easy to manipulate the data, create summaries and perform numerical computations on columns of text.

Sed - An Introduction and Tutorial

http://www.grymoire.com/UNIX/Sed.html

Sed is a program installed on all UNIX systems. It is extremely useful for modifying text files with simple rules, for example, doing search-and-replace.

A Tutorial Introduction to GNU Emacs
http://www.lib.uchicago.edu/keith/tcl-course/emacs-tutorial.html
If you really don't like Vim (introduced in the next section), you may find Emacs and xemacs a better choice for you. They have a similar feature-set to vim, but a completely different interface. The more programmer's text editors and IDEs you are familiar with the more flexible you are as a developer, which means you will adapt quickly to changes in your work environment.

UNIX man pages
As mentioned in the first section, the manual pages (accessible by typing "man <command>" are the definitive reference for everything currently installed. Some man pages are particularly comprehensive and worth at least skimming through:

Vim documentation

http://vimdoc.sourceforge.net/htmldoc/usr_toc.html
http://www.vim.org/tips/index.php

vim is an extremely flexible text editor used by the majority of computer science students at RMIT. The next section of this manual will give a very brief introduction, but to really take advantage of Vim you need to find the features that work best for you. The first URL above is the complete Vim documentation; the second is a series of over 1000 tips submitted by users which are particularly useful for seeing just what Vim can do.

UNIX Cheat Sheet

Here is a description of some common UNIX Commands. You can get much more information about these by looking up their man page with the `man` command.

Command	*Description*
sort	sort can perform ascending and descending, alphanumeric and numeric, case sensitive or insensitive sorting on multiple columns of text.
bc	A simple numeric calculator.
cat	Reads files and prints their contents to standard output. It actually stands for "concatenate" as if we mention several files together, their contents are concatenated on output.
cd	Change the current working directory.
chmod	Change the permission bits on a file or directory.
cp	Copy a file or directory.
cut	Filter out columns of a tabulated text file.
date	Display the current date and time.
dos2unix	Convert DOS line endings to UNIX line endings in a text file.
du	Shows how much disk space a file or directory is using.
echo	Prints its arguments to standard output.
file	Identify the type of a file.
find	Search for files or folders matching a pattern.
grep	Search for text within one or more files, or within standard input.
groups	Shows what groups you belong to.
tar	Create, extract and list tar and compressed tar archives.
head	Show just the specified number of lines from the start of a text file.
kill	Send a signal to a process; commonly used to terminate processes.
ln	Create symbolic and hard links.
ls	List the contents of a directory.
man	View a UNIX manual page.
mkdir	Make a new directory.
mv	Move or rename a file or directory.
nano	A simple text editor.
ps	List currently running processes.
pwd	Print the current working directory.
quota	Shows how much of your allocated disk space you are using.
rm	Remove (delete) a file or directory.
sed	Stream editor, a program for manipulating lines in a text file.
sort	Sort the lines in a text file.
tail	Show just the specified number of lines from the end of a text file.

tar	A compression utility available in Linux. It is often used with gzip – tar maintains the file structure and gzip has better compression.
top	Shows and updates a list of processes that are the most active.
uniq	Remove duplicate adjacent lines from a text file.
unzip	Extract the contents of a zip archive file.
vim	A powerful text editor.
wc	Count the number of characters, words and lines in a text file.
which	Determine which program will be run from the PATH. Another way of thinking about this is what is the location of the program that runs when I type a certain command.
xargs	Executes a program with the standard input as its arguments
zip	Create a zip archive file.