Linux is an operating system. What does that mean? One definition of an operating system is that it is an environment within which other programs can do useful work. Microsoft Windows and Apple OS X are two commonly used operating systems. This tutorial is designed to introduce you to the Linux operating system. Linux is a clone of Unix – a powerful, flexible, and portable operating system developed at Bell Laboratories in 1969. Linux was originally written by Linus Torvalds at the University of Helsinki in Finland during the early 1990s. It continues to be developed and maintained by lots of programmers around the world under the GNU General Public License which means that its source code can be freely obtained.

An operating system by itself isn't much use. Many organisations package the Linux kernel with thousands of programs on one or more CDs or DVDs and make them available for a small charge or as a free download. Some of the more popular Desktop Linux distributions are LinuxMint, Fedora, Ubuntu, and MX Linux. Although the underlying operating system is the same, at first glance, some Linux distributions look as different from each other as Windows and OS X. Linux is also the operating system underlying Android smart phones. If you would like to run Linux on your personal computer you can find reviews as well as links to the most popular distributions at `http://distrowatch.com/`.

The Command Line Interface

Like most operating systems, Linux provides a Graphical User Interface (GUI) that you can use to interact with the computer. In addition to this, it has a very powerful Command Line Interface (CLI). The command line is a text-only way of interacting with a computer – commands are entered at a *prompt* and everything happens in response to the commands that are typed. In contrast to this a GUI provides things like a desktop, icons, menus etc which can be manipulated using an input device such as a mouse. The CLI and GUI each have their advantages and disadvantages, so it is best to think of them as complementary rather than competing ways of interacting with a computer.

In general, the visual elements of a GUI make it more accessible for new users. On the other hand the CLI has a steeper learning curve, but is much more powerful once it is learned. If you look around the menus you could find a few hundred programs which can run from the GUI. Compared to this, there are a few thousand programs which can be run from the command line.

This week's lab material provides an overview of *some* commands– specifically those ones which you will need to use on a regular basis. The best way to become familiar with the command line is to try things out and spend some time getting used to it. We recommend that you do this early in the semester, rather than later on when tests or assignments are approaching. Although some of the initial commands you will learn can be done just as easily from the GUI, it is a better idea to use the command line whenever possible since that will help you to quickly become proficient at using it. As you learn how to use the command line, it will enable you to do things quickly and easily which would be time consuming, if not impossible, using the GUI.

Be sure to take advantage of the expertise of the teaching staff and lab demonstrators. You could spend hours struggling with a problem that could be solved in a few seconds with some help. Also make friends with your classmates and help each other. Most questions that you have will be shared by others, and may have already been answered. Cooperating with others can be more productive than going it alone.

The Terminal Window

Using the CLI generally involves entering commands into a *Terminal* window. It is probably a good idea to create a shortcut to the Terminal application, since a lot of the work you do this year will be done in a terminal.

If you don't have a terminal window open please open one now. And then try entering the command `date`.

```
date
```

When you type the command above and press return you will see the current date and time printed in the terminal window.

```
Fri 17 Jan 2020 17:02:35 NZDT
```

Note: Linux is case sensitive. If you entered '`Date`' instead of '`date`' you would get an error message — `Date: command not found.`

It's very common to display text on the screen. The command to do this is `echo`. If you type echo and then press return all that happens is that a newline is printed (like using `println` in Java without giving it a string to print). Try printing a few words like this:

```
echo Here is some text
```

Which produces:

```
Here is some text
```

Unlike in Java, you don't have to enclose the text to be printed in quotation marks. However, it is often a good idea to do so since some characters have a special meaning when used on the command line.

Updating your configuration files

There are a number of text files which are used to configure various aspects of your environment. These are usually just plain text files, but you shouldn't edit them unless you know what you are doing. To ensure that you have the latest version of a number of config files run the command

```
/home/cshome/coursework/241/update-config
```

Now close your terminal window, and open a new one. Your prompt should be coloured and include information such as username and computer number etc.

Getting Help

If you need help with anything then you can ask a question on MS Teams in **COSC241 Labs**. If it is during a scheduled lab time there will be demonstrators online ready to assist you.

Changing your password

Now change your password from the one you were given by entering the command below. Make sure that your new password doesn't use common words, or a sequence of numbers and letters which is easily connected to you (like your name or phone number). It should be at least 8 characters long and should include unusual punctuation or digits, as well as uppercase and lowercase letters.

```
/usr/bin/passwd
```

You will notice that when you type a password the characters are not echoed to the screen. It is polite to look away when someone beside you is entering their password.

Log out now, and log back in using your new password.

The File System

The file system is a logical method for organizing and storing large amounts of information in a way that makes it easy to manage. Its primary components are files and directories. A file is the smallest unit in which a user can store information. A directory is like a folder on a Mac or Windows, and can contain a number of files.

The root directory

The root directory of the file system is indicated by a leading / character. The / is also used as a separator character between directories. This means that any file can be uniquely identified by the path taken through the file system to reach it. Your prompt has been set up to automatically show you (in blue) the path to the directory you are currently in. When you first open a terminal window you are placed in your home directory[1]. Because your home directory is such a common place to be it is often represented by a tilde character ~, so your prompt should look like this

```
username@oucs1658:~$
```

[1] *Your* home directory is a directory with the same name as your user-code. This is not the same as *the* home directory, which is the directory which contains all the users home directories.

as an abbreviation for the longer version

```
username@oucs1658:/home/cshome/u/username$
```

This means that you are inside the directory `username` (this will be your own username), which is inside the directory `u` (this will be the first letter of your own username), which is inside the directory `cshome`, which is inside the directory `home`, which is inside the root directory '/'. Figure 1 on page 4 shows a graphical representation of where you are.

You can find out the path to the directory you are currently in by using the command pwd (path to working directory).

```
pwd
```

This will print the full path to the current directory:

```
/home/cshome/u/username
```

Different ways of specifying Pathnames

You have a choice when it comes to referring to a file or directory. One way is to use the absolute pathname. This means giving the whole path starting from the root directory. e.g. `/home/cshome/u/username`. Another option is to use a relative path which identifies a file or directory from where you are currently. For example if you were in the `u` directory you could just use `username`. Or if you were in the `home` directory you could use `cshome/u/username`.

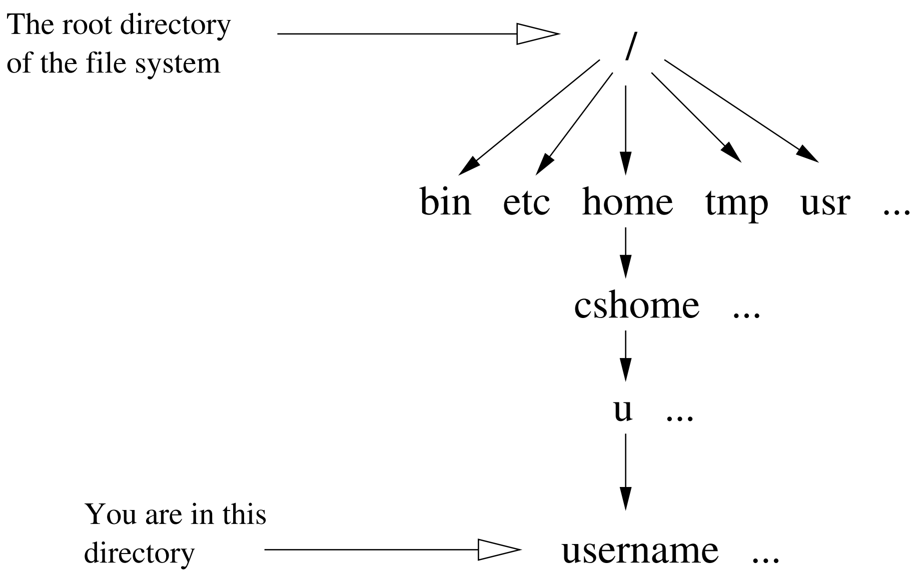
The root directory of the file system
/
bin
etc
home
tmp
usr
...
cshome
...
u
...
You are in this directory
username
...

Figure 1: A graphical view of the file system.

There are also some abbreviations you will want to be familiar with.

- As already mentioned ~ stands for your home directory.
- . stands for the directory you are currently in (called the current or working directory).
- .. stands for the directory immediately above the one you are currently in (called the parent directory).

Moving Between Directories and Listing Files

The command to change your working directory (the directory you are currently in) is `cd` (change directory).

```
cd /usr
```

Try entering this command and you will notice your prompt will change to reflect your new location, like this:

```
username@oucs1658:/usr$
```

The command to list the contents of a directory is `ls`.

```
ls
```

This will list all of the files inside the `usr` directory.

```
bin      games    lib      libexec  sbin     src
etc      include  lib64    local    share    tmp
```

To change back into your own home directory you can enter any of the following four commands.

```
cd /home/cshome/u/username
```

```
cd ../home/cshome/u/username
```

```
cd
```

```
cd ~
```

The first command will change to your home directory from anywhere in the file system because it specifies exactly where your home directory is. Remember the initial / stands for the root of the file system and means that this is an absolute pathname.

The second command will only change to your home directory if you are in a directory which is at the same level as the `home` directory. i.e. it is a relative pathname.

The third command will move you to your home directory from anywhere in the file system. This is because you will often want to change to your home directory, so by default the `cd` command will take you there if you don't give it an argument.

The fourth command will also take you to your home directory. It is more commonly used to move straight to a subdirectory of your home directory like this

```
cd ~/somedir
```

Note: Remember that .. stands for the directory immediately above the one you are currently in. You can move into the directory above the one you are in by using the command

```
cd ..
```
[2]

Creating and Deleting Directories

The command to create a new directory is `mkdir` (make directory). Make sure you are in your home directory.

```
cd
```

Now make a new directory called tempdir.

```
mkdir tempdir
```

Use the `ls` command to see that it has been created.

```
ls
```

Now change into your newly created directory.

```
cd tempdir
```

Create another directory called anotherdir and then move back up into your home directory.

```
mkdir anotherdir
```

```
cd ..
```

The command to delete a directory is `rmdir` (remove directory).

```
rmdir tempdir
```

When you try to remove the directory tempdir and you will get an error message like this

```
rmdir:  failed to remove 'tempdir':  Directory not empty
```

This is because the `rmdir` command will not let you delete a directory if it has anything in it. To remove tempdir you first need to change into tempdir then remove anotherdir, change back into your home directory and you can then successfully remove tempdir. The commands needed to do this are:

[2]There is a space between `cd` and the 2 dots (unlike in DOS).

```
cd tempdir
```

```
rmdir anotherdir
```

```
cd ..
```

```
rmdir tempdir
```

Can you think of a way you could have removed the directory `anotherdir` without first changing into `tempdir`?

You could have given the path to anotherdir as an argument to the `rmdir` command. Using a relative path the command would be:

```
rmdir tempdir/anotherdir
```

Using an absolute path the command would be:

```
rmdir /home/cshome/u/username/tempdir/anotherdir
```

The Shell

The program that you have been interacting with in the terminal window, which provides you with a CLI, is called a *shell*. There are lots of different shells available for Linux. The default shell which we use on the lab machines is called *bash*.

Completion

One useful feature of a shell is the ability to perform completion of commands and filenames. For example, let's say you wanted to see what files are in the directory `/home/cshome/coursework/241/tests`. Instead of typing it all out just type the first letter of each part of the path, press `TAB` and the shell will complete it for you. If there is more than one possibility then it will beep. If you press `TAB` again it will list all of the possible completions. This feature not only saves you time it also helps you to avoid spelling mistakes.

History

Often when you are working in a terminal window you will find that you want to enter a command again that have entered previously. If you press the up-arrow the shell will place the previous command you entered at the prompt. By repeatedly pressing the up-arrow you can scroll up (or down with the down arrow) through your command history. Once you have found the command you want you can re-execute it by pressing enter[3]. You can also edit the command (using left-arrow, backspace etc) to alter it if you wish before executing it.

[3]You can be anywhere within a command when you press enter. The shell will still process the whole command even if you aren't at the end of the line.

Keyboard shortcuts

There are lots of keyboard shortcuts that you can use while using the command line. If you want to leave your hands in the touch-typing position then you can use Ctl-p (previous command) instead of the up-arrow.

Note: Ctl-p means hold down the Control key and type p and then release the Control key.

You can also use Ctl-b (backward) instead of the left-arrow and Ctl-f (forward) instead of the right-arrow. You can move to the start of a line with Ctl-a, and move to the end of a line with Ctl-e. You can delete all of the text from the cursor to the end of the line with Ctl-k (kill). You can paste previously killed text with Ctl-y (yank).

Redirection

Usually input is read from the keyboard (the 'standard input stream' is referred to as *stdin*) and output is sent to the screen (the 'standard output stream' is referred to as *stdout*). However you can alter this using the redirection symbols `'>'` and `'<'`. The `'>'` symbol means redirect the output. The `'<'` symbol means redirect the input. Try listing all of the files in the `/usr/bin` directory.

```
ls /usr/bin
```

Now execute this command again and send the output to a new file which we will call `program-list.txt`.

```
ls /usr/bin > program-list.txt
```

We can view the contents of this file using the command `cat`. The cat command concatenates (or joins) and prints the contents of a number files to the screen. You can view the first few or last few lines of a file using the commands `head` or `tail` respectively.

Now let's count how many program names there are in our file. To do this we will use a program called `wc` (word count).

```
wc -l < program-list.txt
```

This will print out how many lines, and therefore program names, are contained in the file `program-list.txt` (the character after the - is lowercase L, not number 1).

If you try to repeat the command

```
ls /usr/bin > program-list.txt
```

you will get the error message: `cannot overwrite existing file`.
This is because the shell has been told to prevent you from accidentally overwriting

files[4]. You can force the redirection to happen by using a vertical bar after the redirection symbol `'>|'` like this.

```
ls /usr/bin >| program-list.txt
```

Sometimes you may want to append some output to a file. You can do this by using two redirection symbols '>>' like this.

```
ls /bin » program-list.txt
```

Pipelines

Another feature shells give you is the ability to connect the output of one command to the input of another command. The vertical bar `'|'` is used as the pipe symbol. The two commands which we used previously

```
ls /usr/bin > program-list.txt
```

and

```
wc -w < program-list.txt
```

could have been combined in a single command which didn't use a file at all by using the pipe symbol.

```
ls /usr/bin | wc -w
```

This means that the command `wc -w` gets its input from the output of the command `ls /usr/bin`.

This is one of the fundamental concepts underlying Unix-like systems. A program can read some input and perform an operation on it to produce some output. This can then be sent to any number of subsequent programs which progressively transform the output in order to produce the desired result. This is analogous to the way simple Java methods can be composed within a complex program.

Copying, Deleting and Renaming Files

Make sure that you are in your home directory using the command `cd`.

Create a test file with the contents "This is a test" called `testfile.txt`, using `echo` and the redirection (>) symbol. Use the `cat` command to make sure that the contents of `testfile.txt` is what you expect it to be.

After entering each command in this section use `ls` to check that the expected action has happened. The command to make a copy of a file is `cp` (copy). Try making a copy of testfile.txt like this

```
cp testfile.txt same.txt
```

[4]When you redirect output to a file it will cause the file's existing contents to be replaced.

The name of the existing file comes first, then the name of the new file. As before you can use relative or absolute pathnames for the files.

The command to delete a file is `rm` (remove). Try removing testfile.txt.

```
rm testfile.txt
```

You need to take care when removing files because once they have been deleted you can't get them back.

To rename a file you use the command `mv` (move). Use `mv` to change the name of your file back to testfile.txt.

```
mv same.txt testfile.txt
```

You could have got the same result by using copy followed by remove.

```
cp same.txt testfile.txt
```

```
rm same.txt
```

Expansion

We saw earlier that the tilde character '∼' can be used as an abbreviation for your home directory. Another way of saying this is that '∼' gets expanded into the path to your home directory. There are number of other 'special' characters which the shell will expand for you. The most commonly used one is the asterisk `'*'`. It expands into any string of zero or more characters. Here are some examples of its use (you don't need to type all of these in since they may refer to non-existent files).

```
ls p*
```

Lists all files in the current directory starting with p.

```
ls *.java
```

Lists all files in the current directory which have names ending in `.java`.

```
mv *.txt ~/textfiles
```

Moves all files in the current directory which have names ending in `.txt` to a directory called textfiles which is a subdirectory of your home directory.

```
cp /imaginary/directory/samples/*.java .
```

Copies all the `.java` files from `/imaginary/directory/samples` to the current directory.

There are many more features to shells. They are in fact programming languages in their own right. You can assign values to variables, use arrays and various kinds of loops as well as call functions etc.

Arguments and Options

Most CLI commands will take arguments and options. An argument is typically an object, such as a filename or a string of characters, on which a command acts. Options modify the way in which the command is executed. To demonstrate this change to your home directory and try these four examples of using the command `ls`.

```
ls
```

```
ls /home/cshome/coursework/241/pickup
```

```
ls -l
```

```
ls -l /home/cshome/coursework/241/pickup
```

The first one just lists the contents of the current working directory. The second one lists the contents of the directory `pickup` because the path to `pickup` is given as an argument. The third one gives a long listing[5] of the current directory because the `-l` option has been given to the command. The fourth one gives a long listing of the contents of directory `pickup` because it has the `-l` option and has been given the path `/home/cshome/coursework/241/pickup` as an argument.

Options are usually distinguished from arguments by being preceded by a minus sign.

Man and Less

If you want to find out information about a command such as which options and arguments it expects you can use the command `man` (manual pages).

```
man ls
```

Man pages are often quite long. So they are passed to a program which allows you to scroll through text called `less`[6]. Some of the basic commands which work in `less` (and `man`) are:

`up-arrow`	Scroll up one line.
`down-arrow`	Scroll down one line.
`f`	Scroll forward one page.
`b`	Scroll back one page.
`/`*`text`*`<enter>`	Search forward for *text*.
`?`*`text`*`<enter>`	Search back for *text*.
`n`	Move to next occurrence (when searching).
`N`	Move to previous occurrence (when searching).
`q`	Quit.

[5]The `-l` (lowercase L, not the number one) option to `ls` provides lots of extra information about the files listed.

[6]There is an older program called `more` which also is used to scroll through text. Ironically `less` has more features than `more`.

File Permissions

When you executed the `ls` command with the `-l` option you would have noticed that it told you more information than just the file names. Immediately to the left of the filename it tells you when it was last modified and its size in bytes. It also tells you which user owns the file and its group ownership. But what about the series of letters and dashes on the far left? If the first column contains a `d` it means that the file is a directory. The remaining nine columns can be read in three groups. These groups are user, group and others. User refers to you, group refers to other stage 2 students, and other refers to everyone else (sometimes called world). The three columns in each group are read, write and execute. If you have read permission (`r`) you can look at a file. If you have write permission (`w`) you may change a file. If you have execute permission (`x`) then you may run the file as a program, except if it is a directory, where '`x`' means you may enter the directory.

The command to change the permissions of a file is `chmod`. Here are a couple of examples of its use.

```
chmod go-rwx filename
```

```
chmod u+x filename
```

The first example says take read, write, and execute permission for filename away from everyone except you. The second example gives you execute permission for filename. The option that you give to `chmod` is any combination of `u`,`g` and `o` followed by a `+` or a `-`, followed by any combination of `r`, `w` and `x`.

Editing Files

An editor is a program which enables you to create and manipulate files. There are lots of different editors available for Linux. In this lab we will use one called *emacs*. You can use whatever editor you like during 241, or even an IDE if you prefer. However, when you sit the practical tests you may only use a regular text editor. In view of this, it's a good idea to become proficient at using a regular text editor so that you can sit the tests in a familiar environment.

To open a new emacs window just enter `emacs`, followed by the name of the file you want to edit, at the prompt.

```
emacs testfile.txt &
```

The & at the end of the command means that emacs should run in the background. If you didn't include it then the terminal window wouldn't respond to commands until emacs had finished running.

If you forgot to include the & and went back to the terminal window without exiting from emacs, when you you tried to enter more commands in the terminal it wouldn't respond. If this ever happens you can go to the terminal and press `Control-z` to suspend the running program (emacs) and then enter the command `bg` to resume running the program in the background.

You can use the menus in emacs to perform all of the various operations that you would expect in an editor e.g. opening and saving files, cut, copy, paste, etc. You can also switch between various open files using the *buffers* menu. If you want to learn various key combinations to make editing text faster then select *Emacs tutorial* from the *Help* menu. Many of the keyboard shortcuts that you use on the command line will also work in Emacs.

If you would like to try a different type of editor then you might consider using Vim which is a modal editor. It is arguably harder to learn (in the initial stages) but is also very powerful. You can open an introductory tutorial by entering `vimtutor` in a terminal.

If you prefer to use a simpler editor then you might want to try using `gedit`.

Accessing Your Files From Home

The department has provided a machine called *hextreme* which all students can use in order to work from home remotely, in the same environment as the Linux machines in the lab. Please read the guidelines at

https://www.cs.otago.ac.nz/student/resreg/prv/offcampus.php

before attempting to connect from off campus.

Using either Linux or OSX you should be able to connect using the command

```
ssh username@hextreme.otago.ac.nz
```

You will be prompted to accept the host key the first time you connect, and then asked to enter your password.

If you using Microsoft Windows then you will need to install an ssh client first. A popular choice is `PuTTY` which is available at

http://www.chiark.greenend.org.uk/~sgtatham/putty/

Terminating Processes

Occasionally you might end up with a process running on *hex* that doesn't stop running. You can list all processes owned by you using the command

```
ps -au username
```

Which will list the names of all processes owned by you as well as their PROCESSID (the number at the start of the line). You can then terminate them using:

```
kill PROCESSID
```

or

```
kill -9 PROCESSID
```

using the matching PROCESSID of the process that wish to terminate. The -9 option can be added to terminate an unresponsive process.

Assessed Exercises

Make sure you are in your home directory and then create a new directory called `241`. Create another directory inside your `241` directory called `01`.

Note: The ∼`/241/01` directory is where you will do all of your assessed work for this week's lab material.

Exercise 1: Creating files and directories

Use the commands learned previously to create a file structure within your ∼`/241/01` directory which matches Figure 2. All of the names which end in *dir* should be directories[7]. All of the names which end in *.txt* should be text files.

```
            topdir
          /    |    \
        adir  bdir  cdir
       /      /  \     \
hello.txt  ddir edir  goodbye.txt
                 |
                fdir
            /    |   \
  goodbye.txt  gdir  hello.txt
```

Figure 2: A small file structure

When you have done this you can check the file structure at glance using the command

```
tree topdir
```

Ensure the output looks correct and then save it into a file called `topdir-tree.txt` using the command:

```
tree topdir > topdir-tree.txt
```

[7]Normally directories should have names which indicate what is in them.

Exercise 2: Writing and Running Java programs

Use an editor to create a new Java source file called `Hi.java`.

Add code so that your program prints out the message:

```
Hello, world!
```

Now compile your program using the Java compiler (the -Xlint option causes some extra checks to be performed).

```
javac -Xlint Hi.java
```

If there are no errors or warnings then run it using the Java application launcher.

```
java Hi
```

This should produce the output:

```
Hello, world!
```

Once you have got your program working correctly open a new file in your `~/241/01` directory called `errors.txt`. Make the following changes to your `Hi.java` and record what happens in each case, including any error messages, in your `errors.txt` file.

1. change Hi to hi
2. change Hello to hello
3. remove the first quotation mark in the string
4. remove the last quotation mark in the string
5. change main to man
6. change println to printline
7. remove the semicolon at the end of the println statement
8. remove the last brace in the program

Exercise 3: Absolute and Relative Paths

Create a file called `paths.txt` and write either *absolute* or *relative* on each of the first eight lines to show whether each of the following paths is absolute or relative.

1. `/home/cshome/u/username`
2. `241/bin`
3. `/bin/ls`
4. `bin/sort`

5. `home/cshome/u/username`
6. `../cshome/u/username`
7. `~/newdir`
8. `/bin/sort`

Exercise 4: Counting lines and words

Write a Java program in a file called `Counter.java` which reads its input from `System.in` and prints out the number of lines and words that it read. A word is defined as a consecutive series of characters preceded and followed by whitespace (or the beginning/end of input). You might find it useful to use the `Scanner` class from `java.util`. Put a package declaration at the top of your class like this:

```
package week01;
```

and compile your program using this command:

```
javac -d .  -Xlint Counter.java
```

The `-d .` option will cause a directory to be created which matches the package name and the class file will be put inside it. Run your program using the command:

```
java week01.Counter
```

Type a few lines of text and then press Ctrl-d at the start of a line to send an end-of-file (EOF) signal to your program. Your program should print the number of lines and words to the screen and then exit. You can use your program to count the number of lines in your source file like this:

```
java week01.Counter < Counter.java
```

The output should look just like this (although the numbers can be different):

```
lines: 20
words: 49
```

Marking

When you have completed the lab and all of the exercises check that it is correct by running the command:

```
241-check
```

Follow the prompts, correcting anything that needs to be fixed, until your work is passing everything. You can check just the layout and Javadoc comments using the checkstyle command.

```
checkstyle Filename.java
```

You can see what checkstyle's requirements are by running **checkstyle -h**. When your work meets all of the requirements you can submit it by running the command:

```
241-submit
```

You have until 10:00 pm on Sunday to complete and submit your lab work each week. It's a good idea to complete the work during your scheduled lab times so that if you have any problems you can ask the demonstrators for help and guidance.
Note: checkstyle doesn't check that your comments actually make sense. From time to time we will inspect the comments of work that has been submitted, and if we find nonsense or inaccurate comments then marks may be lost.

Command Summary

We have covered a lot of ground in this lab. Don't worry if you can't remember everything. Just put aside some time to practice using the command line and it will soon become second nature. There are many other useful commands that we haven't covered in this introduction. It is likely that you will have finished this lab with time to spare. If that is the case then you might want to try experimenting with some other commands. Here is a summary of the commands covered in this tutorial, along with some others that you might want to try using.

- cal - print a calendar for a given month and/or year
- cd - change directory
- chmod - change access permissions on files and directories
- cp - copy files
- cut - cut out selected portions of each line of input
- date - display the date and time
- diff - find the differences between files
- echo - display text on the screen
- emacs, gedit, vim - edit text files
- find - search for files matching a given criteria
- grep - find matching patterns within input lines
- java, javac - compile and run Java programs
- ls - list files in a directory
- mkdir - make a new directory
- mv - rename files or directories

- passwd - change your password
- pwd - print the path to the working (current) directory
- rm - remove files and directories
- rmdir - remove empty directories
- sort - sort lines of input
- tar - put files into, or extract files from an archive
- time - measure the time taken to execute a command
- uniq - report or filter out repeated lines