Introduction to Unix

Revised July 2019

Contents

1 Instructions for all students

Work your way through this handout. As you complete each section of the lab write your answers to the exercises in the file **lab01-answers.txt** which opened automatically when you started this lab.

This lab is worth 0.5%. Be sure to get your work marked off before leaving the lab.

You can call a lab demonstrator for help or marking by running the command **democall** in a terminal window.

2 Getting Started (241 students skip this section)

This section is only for new second year Computer Science students. If you did COSC241 during the first semester then go straight on to section 3.

2.1 Introduction

Unix 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.

In our second year labs we use Linux[1], which 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. Two of the more popular Linux distributions are Ubuntu and Fedora. 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.

If you would like to run Linux on a computer at home you can find reviews as well as links to many of the most popular Linux distributions available at `http://distrowatch.com/`. This semester we will be running a recent version of Fedora on the lab computers.

2.2 The Command Line Interface (CLI)

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). A CLI 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. Whereas the CLI has more of steep learning curve, but is much more powerful once it is learned. If you look around the menus you will find that there around 200 programs which can run from the GUI. Compared to this, the folder which contains most of the programs available at the CLI has over 3000 programs.

This introduction provides an overview of *some* CLI commands– specifically those ones which you will need to use on a regular basis. The best way to become familiar with the CLI is to try things out and spend some time getting used to the system. We recommend that you do this early in the semester, rather than later on when you will have assessments due. Although some of the initial commands you will learn can be done just as easily from

[1]Most of the commands in this guide should work fine in OSX as well, since it is also a Unix based operating system.

the GUI, it is a better idea to use the CLI whenever possible since that will help you to quickly become proficient at using it. As you learn how to use the CLI, 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.

2.2.1 Logging in

Before you can do anything you need to log in. To log in to a computer just enter your *computer science* user-code and your password at the login screen.

> Your user-code is usually the first letter of your first name followed by your last name (possibly with a middle initial, if someone else has a similar name to you). So if your name is *Mary Jane Smith* your user-code could be `msmith` or `mjsmith` (note that it is all in lowercase).
>
> If you are new to the department then your password will initially be your Student ID Number.

Once you have successfully logged in, hunt around until you find the *Terminal* application and start it up. The terminal window is the place where you will enter commands.

You can customise lots of things about your desktop including which desktop environment you use. We will leave it up to you to play around and customise things according to your preferences. It is probably a good idea to create a shortcut to the *Terminal* application on your desktop, panel, or both, since a lot of the work you do this year will be done in a terminal.

Now it's time to start getting acquainted with the CLI. If you don't have a terminal window open please open one now. This window is where you will be entering commands.

Throughout this tutorial commands are shown in a box with a '$' character as the prompt[2] as shown below.

```
$ date
```

Try entering the command above and you will see the current date and time printed in the terminal window.

Note: Linux is case sensitive – if you entered `Date` instead of `date` you would get the error message: `command not found.`

[2]The prompt appears on the left hand side of the terminal window, and indicates that the terminal is ready for you to enter commands.

2.2.2 Updating your configuration files

There are a number of 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. If you didn't do COSC241 last semester then run the command

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

to ensure that you have the latest version of a number of config files. Now close your terminal window, and open a new one. Your prompt should be a different colour and include information such as username and computer number etc.

2.2.3 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[3] 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.

2.2.4 Email

Email will sometimes be sent to the whole class and you are expected to read it. It is very important that you check your University email on a regular basis.

[3] To 'echo' something means to print it on the screen.

3 All students continue here

All students should do the lab from this point on. If you did 241 last semester parts of the lab should be pretty familiar to you. However, you should find it much easier to complete than when you first did it.

3.1 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[4] is like a folder on a Mac, and can contain a number of files.

3.1.1 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[5]. Because your home directory is such a common place to be it is often represented by a tilde character ~, so your prompt should look similar to this

```
username@computer:~$
```

as an abbreviation for the longer version

```
username@computer:/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 7 shows a graphical representation of where you are.

You can print the full path to the directory you are currently in using the command pwd (path to working directory).

```
$ pwd
```

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

[4] A directory is really just a special type of file.

[5] *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.

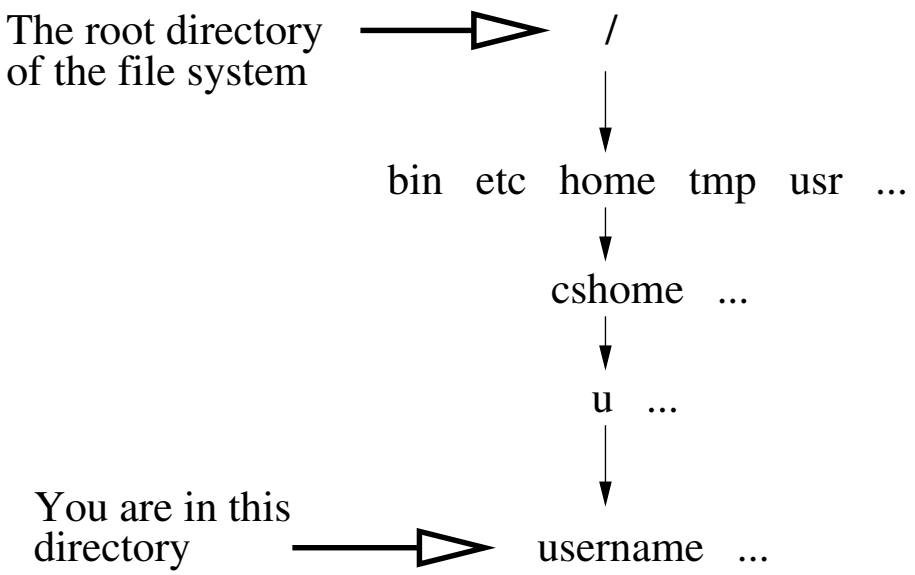

Figure 1: A graphical view of the file system.

3.1.2 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.`

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

- As already mentioned $\sim$ 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).

Exercise 1

Write down whether each of the following files or directories is specified by an absolute or relative path.

a) `/home/cshome/u/username`

b) `244/bin`

c) `/bin/ls`

d) `bin/sort`

e) `/usr/bin/wc`

f) `home/cshome/u/username`

g) `../cshome/u/username`

h) `~/newdir`

i) `/bin/sort`

3.2 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:/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         include     lib64       local       share       tmp
games       lib         libexec     sbin        src
```

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

1. `$ cd /home/cshome/u/username`
2. `$ cd ../home/cshome/u/username`
3. `$ cd`
4. `$ 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 ..
```
[6]

Exercise 2

1. Change to directory `/home/cshome/coursework/244/files/01`. List all of the files in that directory and write down their names.

2. Change back into your home directory. List all of the files found in the directory `/home/cshome/coursework/244/files/01/dir1` by entering the command

   ```
   $ ls /home/cshome/coursework/244/files/01/dir1
   ```

   and write down the file names.

3.3 Creating and Deleting Directories

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

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

```
$ 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: ‘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:

```
$ 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
```

3.4 The Emacs Editor

An editor is a program which enables you to create and manipulate files. There are lots of different editors available for Linux. In this tutorial we will show you how to use one called *emacs*. To open a new emacs window just enter `emacs` at the prompt.

```
$ emacs
```

Note: Lots of commands in emacs can performed by holding down the Control key and pressing other keys (abbreviated as `C-`***key***). For example `C-x C-f` means to hold down the Control key (labeled Ctrl) and type an 'x' followed by an 'f', then release the Control key.

3.4.1 The built-in Emacs tutorial

Emacs has it's own built-in tutorial which you can access using `C-h t`. Press those keys now and work your way through the tutorial. Even if you have been using Emacs for a while you will probably still find some things that you didn't know about.

Exercise 3

In Emacs

1. What is the command to move to the start of a line?
2. What is the command to move to the end of a line?
3. What is the command to delete a line?
4. What is the command to split the screen into two windows?
5. What is the command to remove all windows except the one the cursor is in?
6. What does `C-l` do? *(that's an el not a one)*
7. Which of the above commands also work in a terminal window?

3.5 Writing and Running Java programs

From your home directory, change into the `244/01` directory and open a new Java source file to write a small program in using Emacs.

```
$ emacs HelloWorld.java
```

Type the following code into the editor window, save it, and exit.

```
public class HelloWorld {

   public static void main(String[] args) {
      System.out.println("Hello, world!");
   }

}
```

Now compile your program using the Java compiler.

```
$ javac HelloWorld.java
```

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

```
$ java HelloWorld
```

You can use these steps to compile and run Java programs on any platform where Java is available.

3.6 Copying, Deleting and Renaming Files

Change back into your home directory using the command `cd`. 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
```

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
```

3.7 Review

We have covered a lot of ground so far, so let's do some review. In the previous section we learned the basics of some commands for creating and manipulating files and directories. A summary of these commands is given in Table 1. Use these commands to create a file structure within your home directory which matches Figure 2. Make all of the names which end in *dir* be directories[7]. Make all of the names which end in *.txt* be text files.

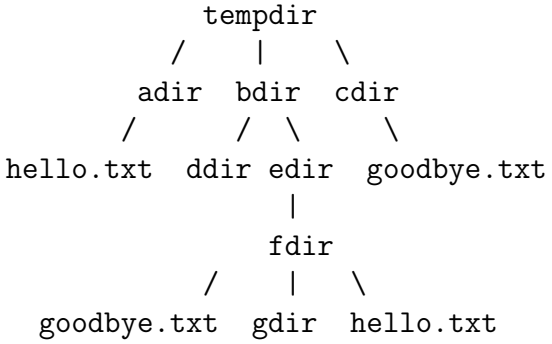

Figure 2: A small file structure

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

```
$ tree tempdir
```

Don't delete `tempdir` until it has been checked by a demonstrator at the end of this lab.

Command	*Description*
`cd`	Change to a new directory.
`cp`	Create a copy of a file.
`emacs` *filename*	Create a new file (`C-x C-s` to save).
`ls`	List the contents of a directory.
`mkdir`	Make a new directory.
`mv`	Rename a file or directory.
`rm`	Delete a file.
`rmdir`	Delete a directory (must be empty).

Table 1: Commands for manipulating files and directories.

3.8 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`.

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

```
$ ls
```

```
$ ls tempdir/bdir/edir/fdir
```

```
$ ls -l
```

```
$ ls -l tempdir/bdir/edir/fdir
```

The first one just lists the contents of the current working directory. The second one lists the contents of the directory fdir because the path to fdir is given as an argument. The third one gives a long listing[8] 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 fdir because it has the `-l` option and `tempdir/bdir/edir/fdir` given as an argument.

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

```
total 12
drwx------    2 ihewson  csstaff      4096 Feb  5 13:47 gdir
-rw-------    1 ihewson  csstaff         8 Feb  5 13:47 goodbye.txt
-rw-------    1 ihewson  csstaff         6 Feb  5 13:47 hello.txt
```

Figure 3: Output from the command `ls -l tempdir/bdir/edir/fdir`.

3.9 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`[9]. 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.

[8] The `-l` (lowercase L, not the number one) option to `ls` provides lots of extra information about the files listed. An example of a long listing is given in Figure 3.

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

3.10 File Permissions

When you executed the `ls` command with the `-l` option you would have noticed that it tells you more information than just the file names (see Figure 3). 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`.

3.11 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*.

3.11.1 Concurrent jobs

One of the useful things a shell will let you do is several tasks at once. You might have noticed that when you entered a command like

```
$ emacs filename
```

if you went back to the terminal window without exiting from emacs and tried to enter more commands that it wouldn't respond. You can get the shell to start emacs while still accepting input from you[10] by including an ampersand character '&' at the end of the command.

```
$ emacs filename &
```

[10]This is called running a process or job in the background.

Alternatively you could have gone back to the terminal and pressed `C-z` to suspend the running program (emacs) and then entered the command `bg` to resume running the program in the background.

3.11.2 Completion

Another 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 is in the directory `/home/cshome/coursework/244/labs`. 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.

3.11.3 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 last 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[11]. You can also edit the command (using left-arrow, backspace etc) to alter it if you wish before executing it.

3.11.4 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 `programs.txt`.

```
$ ls /usr/bin > programs.txt
```

Open up `programs.txt` in Emacs and you can see all of the programs which are in the `/usr/bin` directory.

```
$ emacs programs.txt &
```

[11] 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.

Exit from Emacs by pressing `C-x C-c`. Now let's count how many programs there are in our file. To do this we will use a program called `wc` (word count).

```
$ wc -w < programs.txt
```

This will print out how many words (and therefore program names) are in the file `programs.txt`. These small examples are not very useful. However in the course of program development, it is often very useful to generate test files which can be used to quickly and repeatedly check that your program produces the desired output when given different input.

If you try to repeat the command

```
$ ls /usr/bin > programs.txt
```

you will get the error message: `cannot overwrite existing file`.
This is because the shell can be told to prevent you from accidentally overwriting files[12]. You can force the redirection to happen by using a vertical bar after the redirection symbol `'>|'` like this.

```
$ ls /usr/bin >| programs.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 >> programs.txt
```

3.11.5 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 > programs.txt
```

and

```
$ wc -w < programs.txt
```

could have been combined in a single command which didn't use the file `programs.txt` 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`.

[12] When you redirect output to a file it will cause the files existing contents to be replaced.

3.11.6 Expansion

We saw in the first lab that the tilde character '∼' can be used as an abbreviation for your home directory. Another way of saying this is that '∼' gets expanded into 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 /home/cshome/coursework/244/samples/*.java .
```

Copies all the `.java` files from `/home/cshome/coursework/244/samples` to the current directory.

3.11.7 Other features

There are many more features to shells. They are in fact programming languages in their own right. We finish this section showing you how to use variables and to execute a small shell program (called a shell script). Variables are assigned in the bash shell like this:

```
$ MY_VAR="hello there"
```

You can print out a variable like this:

```
$ echo $MY_VAR
```

Note: We don't use a $ sign when we assign a value to a variable. But we must use one when we refer to a variable.

Open a new text file called `myscript` and type in these three lines:

```
ls -l
my_var="How are you?"
echo "$my_var my_var $my_var"
ls
```

After you have saved the file you can run it from the command line like this:

```
$ bash myscript
```

This tells bash to interpret this file and perform each command it finds in there. You can run the file directly by making it executable and then entering the file name at the prompt.

```
$ chmod u+x myscript
```

```
$ ./myscript
```

Note: It is important to prefix our program name with `./` when we run it directly. This specifies the exact location of our program (because the `.` before the `/` gets expanded into the full path to the current directory) and ensures that we are running the program which we intend and not another one with the same name.

You can use a script to save a series of commands that you often run. Instead of typing them all each time just put them into a script and run that instead.

Now try adding a simple loop to your script like this:

```
for i in $(ls)
do
   echo "There is an item called $i"
done
```

The command `ls` gets executed because it is contained within `$(` and `)`. This produces a list of filenames from the current directory. The variable `$i` gets assigned the value of each filename in the list in turn and the body of the for loop gets executed. Once all of the filenames have been used the loop terminates.

3.12 Further study

There are countless online resources for learning to use Linux which any search engine can find. If you are wanting to further your Java skills you might find it helpful to have a look at the Java Tutorial which can be found online at *http://docs.oracle.com/javase/tutorial/*. The two sections 'Learning the Java Language' and 'Essential Java Classes' are the most relevant for purposes.

We also have a number of Unix and general programming books available for you to read in the lab.

3.13 Counting word frequency

This last exercise is longer and more difficult than all of the previous ones. Do your best, but don't be concerned if you don't manage to complete it. If you get within 10 minutes

of the end of the lab and haven't finished the lab the call a demonstrator to mark you off anyway. Whether you completed the final exercise or not, before you leave the lab take a look at a 4 minute screencast which shows how this program could have been written quickly and easily using some command line tools. Type **view-lab01** in a terminal to view the screencast.

Exercise 4

Write a Java program that will read a stream of text and count the frequency of words a given text. A word is a sequence of letters and/or numbers delimited by white space. White space is defined as a space, carriage return, newline or tab character. Your program should ignore all other characters (e.g. punctuation and control characters). Your program should convert all letters in the incoming text stream to lower-case. Your program should output the words and their frequencies, sorted first by frequency and then alphabetically.

3.13.1 Sample input

```
The rain in Spain falls mainly on the plain.
The rain in Auckland (often) falls in short bursts.
The sum of thirty-seven and sixteen is 53.
```

3.13.2 Sample output

```
1 53
1 and
1 auckland
1 bursts
1 is
1 mainly
1 of
1 often
1 on
1 plain
1 short
1 sixteen
1 spain
1 sum
1 thirtyseven
2 falls
2 rain
3 in
4 the
```

4 Command Line Quick Reference Guide

`<` - *Redirect input from a file.*

Example:

```
sort < words.txt
```

`>` - *Redirect output to a file.*

Example:

```
ls > ls.out
```

`>>` - *Append output to a file.*

Example:

```
ls >> existing.txt
```

`>|` - *Redirect output to a file even if the file already exists.*

Example:

```
ls >| clobbered.txt
```

`|` - *Pipe output from one command to another.*

Example:

Count the number of files in a directory

```
ls | wc -l
```

`*` - *Wildcard symbol, gets replaced by zero or more characters.*

Example:

List all files ending in .java

```
ls *.java
```

`&` - *Run a job in the background (keep the current shell active).*

Example:

```
emacs &
```

`cal` - *Print a calendar for a given month and/or year.*

Examples:

Print a calendar for the current month

```
cal
```

Print a calendar for the year 2009

```
cal 2009
```

`cat` - *Concatenate (join together) files and print them on the standard output. If no filename is given then read from standard input until EOF (`C-d`).*

Examples:

Display a file

```
cat myfile
```

Join together three files and store result in all.txt

```
cat ch1.txt ch2.txt ch3.txt > all.txt
```

Read input from the keyboard and store result in newfile

```
cat > newfile
```

`cd` - *Change to a different directory.*

Examples:

Change to the given absolute directory name

```
cd /home/cshome/coursework/244
```

Change to a given subdirectory of your home directory

```
cd  ~/java/src
```

Change to your home directory

```
cd
```

`chmod` - *Change access mode (permissions) of one or more files.*

Examples:

Give everyone read access to a file

```
chmod a+r filename
```

Remove all access permissions for all other users (group and world) to a file

```
chmod go-rwx filename
```

`cp` - *Copy files and directories.*

Examples:

Create a new copy of a given file

```
cp filename newfilename
```

Copy two text files and all Java files to a given directory

```
cp file1.txt file2.txt *.java  ~/somedir
```

Recursively copy an entire directory structure (including all subdirectories and files) into the current directory

```
cp -r /home/cshome/coursework/244/labs/01 .
```

`date` - *Print the current time and date. You can use a format string the specify the output.*

Examples:

Print current time and date using the default format

```
date
```

Print the time and date in the specified format

```
date +" Date is %D %nTime is %T"
```

`diff` - *Find the differences between two files. When output is displayed lines from the first file are prefixed by < and lines from the second file are prefixed by >. Lines are shown to have been added, changed or deleted by the letters a,c,d.*

Examples:

Show the differences between two files

```
diff file1.txt file2.txt
```

Show the differences between two files ignoring difference in case (-i) and difference in the amount of whitespace (-w).

```
diff -iw file1.txt file2.txt
```

`echo` - *Display text on the screen.*

Examples:

Show the contents of an environment variable

```
echo "Your path is $PATH"
```

Append a line to a file

```
echo "The end." >> filename
```

`emacs` - *A text editor and all purpose work environment.*

Examples:

Start Emacs as a background job

```
emacs &
```

Start Emacs and open all Java files in the current directory

```
emacs *.java &
```

`evince` - *View PostScript and PDF files.*

Examples:

View a PDF file

```
evince Lab1.pdf
```

View a PostScript file

```
evince Lab2.ps
```

`find` - *Search for files in a directory structure.*

Examples:

Find all Java files in the current directory and below

```
find -name "*.java"
```

Find all files in your home directory and below which are newer than a given file

```
find  ~ -newer filename
```

`grep` - *Print lines matching a pattern. The name comes from a command used in the line editor ed - Global Regular Expression Print.*

Examples:

Print all lines from a file which contain a string

```
grep "import" filename
```

Print all lines, with file names and line numbers, of each Java file which contains the given string

```
grep -Hn "print" *.java
```

`kill` - *Kill a process. To find out the number of a process use* ***`ps`****.*

Examples:

Kill process number 12345

```
kill -9 12345
```

Kill all processes

```
kill -9 -1
```

`less` - *View files a screen full at a time.*

Example:

```
less filename
```

Commands:

`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`

`ls` - *List directory contents.*

Examples:

Print a long listing of the contents of the current directory

```
ls -l
```

List all Java files from newest to oldest

```
ls -t *.java
```

`man` - *Display the on-line manual pages. These can be tricky for the beginner to read, but are a very useful source of information as you become more experienced. The program 'less' is used to display the pages which makes them easy to search through.*

Example:

Display the manual pages for ls

```
man ls
```

`mkdir` - *Make a new directory.*

Examples:

Create a directory called newdir

```
mkdir newdir
```

Create three new directories

```
mkdir dir1 dir2 dir3
```

`mv` - *Move (rename) a file or directory.*

Examples:

Rename file1 as file2

```
mv file1 file2
```

Move all Java files into a directory

```
mv *.java javafiles
```

`passwd` - *Change your password.*

`ps` - *List current processes.*

Examples:

List processes owned by user msmith

```
ps -u msmith
```

List all processes

```
ps -A
```

`pwd` - *Display the path to the working directory (the one you are currently in).*

Examples:

Print working directory including symbolic links

```
pwd
```

Print fully resolved name of working directory (no symbolic links)

```
pwd -P
```

`rm` - *Remove (delete) files and directories.*

Examples:

Remove (asking for confirmation) a file.

```
rm -i filename
```

Remove a directory and all its contents (BE VERY CAREFUL) without asking for confirmation

```
rm -rf newdir
```

`rmdir` - *Remove an empty directory.*

Example:

```
rmdir dirname
```

`sort` - *Sort lines of text files.*

Examples:

Print all lines of a file in sorted order removing duplicate entries

```
sort -u words.txt
```

Print all lines of a file sorted using the third field as the key

```
sort -k 3 file.txt
```

`ssh` - *Log in to and execute commands on a remote machine using secure encrypted communications.*

Examples:

Log in to hex (a machine via which you can access your lab files from off campus)

```
ssh hex.otago.ac.nz
```

Log in to hex as user msmith

```
ssh msmith@hex.otago.ac.nz
```

`tar` - *Put files into, or extract files from an archive.*

Examples:

Create a new compressed archive called filename.tgz and put the entire directory structure contained in some-dir into it

```
tar -cvzf filename.tgz some-dir
```

Extract the contents of the archive filename.tgz into the current directory

```
tar -xvzf filename.tgz
```

`time` - *run a command and output the time taken to execute it.*

Example:

Show how long it takes to sort a file.

```
time sort words.txt > sorted-words.txt
```

`top` - *show what processes are using the most resources.*

Example:

Run the program *top* (press q to quit from it).

```
top
```

`tr` - *Translate or delete characters.*

Examples:

Translate all lowercase letters in file1 into uppercase and store in file2

```
tr '[a-z]' '[A-Z]' < file1 > file2
```

Take a file and translate every group of tabs, spaces, and newlines into a single newline. This produces one word per line and could be piped to sort or wc

```
tr -s '\t \n' '\n' < file.txt
```

`wc` - *Count words, lines and characters in a file.*

Examples:

Count the number of words in a file

```
wc -w essay.txt
```

Count the number of lines, words and characters in a file

```
wc < essay.txt
```