COMP1511 18s2

Week-01 Laboratory Exercises

Objectives

In this Lab, you will:

- Learn how to access your CSE account
- Become familiar with the linux environment
- use a text editor to create small C programs
- use dcc to compile a C program

Topics

Getting Started

Normally we have the first COMP1511 lecture of the week before before the first tutorial of the week. This year because of the number students wanting to do COMP1511, we've been unable to do this.

This means if you have a Monday tut-lab, you haven't had the lecture the following questions talk about.

We thought about cancelling the week 1 tut-labs, but decided they were still worth having because you get to meet and know your tutor, your fellow students and see the lab environment.

For tut-labs before the first lecture - tutors will try to explain what they can as part of the tutorial but don't worry if you don't understand lots of things.

Of course there are no marks for the week 1 tut-lab.

If you have a chance watch beforehand either:

- this video of the 2017 session 2 COMP1511 lecture (starting at 17:03) or
- this video of a 2017 session 1 COMP1511 lecture starting at 30:59

But if you can't watch the videos don't worry.

In this first COMP1511 lab, the primary aim is for you to familiarize yourself with the process of creating, compiling and executing simple C programs.

The material you will need to know can be found in the week 1 lecture notes.

This exercise will simply ensure you are capable of writing a small program and getting it to run.

There are no marks for this week's work, but you should practice testing and submitting your answers in preparation for future weeks.

Once you've done this, you are free to explore some other unix commands.

Pairs

COMP1511 lab exercises will be completed in pairs. Your tutor will assign you a partner. You will work together on lab exercises until your tutor rearranges the pairs in 3 or 4 weeks time.

Instructions

Note that the following instructions assume that you are using text commands inside a Linux terminal window. Some of the steps (e.g. creating a new directory) can also be completed using the graphical user interface, similar to what you are used to from Windows.

Logging In

Once you are in your lab, one student from your pair should log into the lab computer.

To log in, you use your zID (which looks something like `z1234567`) and your zPass (which is used to log into all other university online services).

The CSE labs use the Linux operating system (not Windows or OSX or ...).

When you log in, by default, you'll be using the xfce4 window manager, and you'll see a linux desktop.

If you are asked to select a panel click on **Default**.

Along with menus that you can see, such as the Applications Menu in the top left corner, it also has a simple menu you can access by **right-clicking anywhere on the desktop**.

Have a look around and see if you can work out how to open a web browser.

Exercise 0: Trying out some Linux Commands

Although there are lots of things you can do by using the graphical user interface (GUI) - clicking on menu items and icons, we really want you understand how to use the Linux command-line.

If you get stuck with any of this, don't hesitate to ask one of your tutors for help, that's why they're here!

If there isn't a terminal open, right click on the desktop and select 'Open Terminal Here'

This will bring up a window where you can type in Linux commands.

There are a few commands that you can use to check your account details:

- `rquota` tells you how much of your disk quota you've used up (from creating files)
- `pp` tells you your name, what classes you're in, where your home directory is and other details that may or may not be useful

Each of these is a command, and when entering into your terminal, you need to press enter (or return) after each command. Try the two we just mentioned above.

Now try running the command `date` - which prints the current time and date.

You should see something like this:

```
$ date
Monday 26 February  12:15:20 AEDT 2018
```

Now we're going to learn about some commands for working with directories (directories are often called folders on other operating systems).

- pwd
- ls
- cd
- mkdir

The Linux command `pwd` tells you what directory you are in, often called your *current working directory*.
`pwd` stands for '**p**rint **w**orking **d**irectory'.

If your username is 'z7654321', and you enter the 'pwd' command into the terminal, it might tell you:

```
$ pwd
/import/cage/1/z7654321/Desktop
```

This means that I'm in the Desktop directory inside my (z7654321's) home directoryand my home directory is stored on the file server called *cage*.
What does `pwd` print for you?

The Linux command `ls` prints a **l**i**s**t of the files in the present directory. Since you have a new account there will probably be nothing in your Desktop directory.

```
$ ls
```

The Linux command `cd` is a command that you can use to **c**hange your **d**irectory.
If you type in cd on its own as follows, you will move into your home directory

Try this out and confirm your directory has changed by running `pwd` again

```
$ pwd
/import/cage/1/z7654321/Desktop
$ cd
$ pwd
/import/cage/1/z7654321
```

If you run `ls` you'll see what is in your home directory. Since you have a new account, your home directory will most likely only contain your Desktop and a directory called public_html (which is where you can create files to create websites).

```
$ ls
Desktop public_html
```

The Linux command `mkdir` **mak**es a new **dir**ectory. To use it you must supply the name of the directory you wish to create. Lets use it to create a directory for this week's lab exercises.

```
$ mkdir lab01
```

Now the command `ls` should show you the directory you just created.

```
$ ls
Desktop lab01 public_html
```

Now we will use `cd` in a slightly different way where we tell it what directory we want to change into

```
$ cd lab01
```

To confirm we really are in our new created *lab01* directory type the `pwd` command again.

```
$ pwd
/import/cage/1/z7654321/lab01
```

If all this has worked, it's time to try compiling your first program!

Exercise 1: Your first program

Now it's time to create a file. We are going to use the text editor *gedit*.

Tip: you can run a command to automatically configure gedit to properly set up indentation etc.
Type:

```
$ 1511 setup-gedit
```

To create a file using gedit, type:

```
$ gedit bad_pun.c &
```

This will open up a graphical editor where you can manipulate text.
Adding the `&` to the end of a command allows the GUI to return control to the terminal as soon as it starts (i.e. run in the background) rather than after it is closed. If you forget the `&`, you will not be able to type commands into the terminal until the GUI application is finished. This is always a good idea when running a command which invokes a graphical user interface (GUI).

Here's an example of a simple C program. Copy it into the gedit window, and save it.

```c
// A simple C program that attempts to be punny
// Written 23/2/2017
// by Angela Finlayson (angf@cse.unsw.edu.au)
// for COMP1511 Lab 01 Exercise 1

#include <stdio.h>

int main(void) {

   printf("Hello, it is good to C you!\n");

   return 0;
}
```

When you save, it will place the contents of the editor into the file `bad_pun.c`.

Handy Tip 1: On many linux systems copying can be achieved by simply highlighting with the left mouse button and pasting can be achieved by simply clicking with the middle button

Handy Tip 2: Make sure gedit is displaying line numbers down the left hand side. This is important for when we need to fix compile time errors.
If it is not displaying line numbers, go to the Edit->Preferences menu item and check the 'Display Line Numbers' option.

Once you have pressed saved, click on the Terminal window again and type this command to compile your program:

```
$ dcc -o bad_pun bad_pun.c
```

The `-o bad_pun` option tells the compiler to give the newly compiled program the name *bad_pun*.

Test what happens if you leave the `-o bad_pun` option out. What is the default name `dcc` uses?

You may wish to use the `rm` command to delete (**re**m**o**ve) the file created.
(You can Google *rm* or *rm linux* if you are not sure how it works)

If `dcc` does not print out any error messages, then your program has been successfully compiled. Otherwise you will need to find the errors in your code, fix them, save the file and compile again.

Handy Tip 3: Look for the line numbers that are displayed in the error messages as they are major clues to where the problem is in your code.

Handy Tip 4: Always start with fixing the first error first. Sometimes fixing one compile error, saving and recompiling can make all or some of the other errors go away!

After successfully compiling you can check that dcc has produced an executable by typing `ls -l` and looking for a newly-created `bad_pun` file (check the file creation time to see if it really is new).

A useful Unix command is **man** short for **man**ual. Find out what the `-l` option for the `ls` command you used before does. Type:

```
$ man ls
```

Press 'q' to exit man.

Run the program to test that it works. Type:

```
$ ./bad_pun
```

The `./` before the program name specifies that the program is found in the current directory. Note that Unix executable (program) names do not require the `.exe` extension that you might have have seen under Windows.
The COMP1511 class account contains a script that automatically checks your solution to lab exercises.

While it is not important this week, lets run the `autotest` script so you can see what it does.

You can check `bad_pun.c` like this:

```
$ 1511 autotest wk01_bad_pun bad_pun.c
Test 1 (./bad_pun) - passed
1 tests passed  0 tests failed
```

Don't worry if you fail the check - and you can't see why. There are no marks for this week's lab but practice submitting your work electronically.
You submit your work with the `give` command like this:

```
$ give cs1511 wk01_bad_pun bad_pun.c
```

In future weeks both members of each lab pair need to submit your work using `give`.

Exercise 2: Creating an ASCII Bird

Now use the Linux command `cp` to copy `bad_pun.c` to a new file named `bird.c`, like this:

```
$ cp bad_pun.c bird.c
```

You are now ready to start editing the file `bird.c` using your favourite editor.

```
$ gedit bird.c &
```

Note that the basic structure of the program can be retained; you just need to change the comments, and modify and/or add printf statements to the program. Edit the file `bird.c` to produce a new program that behaves as follows:

```
$ dcc -o bird bird.c
$ ./bird

  ___
 ('v')
((___))
 ^   ^
```

Make sure you save your modified program before you compile it.

Make sure you re-compile your program every time you modify the code.

Handy Tip 5 : Linux remembers the commands we have recently typed in. By pressing the `UPARROW` key, it will bring up your previous command and save you retyping it in! Try it. You can press the `UPARROW` key repeatedly to go back to the second last command, third last command and so on.

Handy Tip 6 : At your Linux command prompt, type in `./b` and then press the `tab` key. Linux will automatically try to fill in your partially typed command for you! Again try the automatic checking script in the class account

```
$ 1511 autotest wk01_bird bird.c
Test 1 (./bird) - passed
1 tests passed  0 tests failed
```

Again don't worry if you fail the checking - and you can't see why. There are no marks for this week's lab but practice submitting your work electronically.
You submit your work with the `give` command like this:

```
$ give cs1511 wk01_bird bird.c
```

In future weeks both members of each lab pair need to submit your work using `give`.

Exercise 3: Fixing errors in an ASCII Kangaroo

Copy the program `kangaroo.c` from the course account to your directory by typing (make sure you type the dot at the end):

```
$ cp ~cs1511/public_html/18s2/tlb/01/kangaroo.c .
```

The dot '.' is a shorthand for the current directory and there is a space between `kangaroo.c` and the next dot. You can check that the file has been copied by typing:

```
$ ls
bad_pun bad_pun.c bird bird.c kangaroo.c
```

You can examine the contents of the file by typing:

```
$ less kangaroo.c
// A simple C program that prints an ASCII kangaroo
// Written 7/3/2017
/  by Andrew Taylor (andrewt@unsw.edu.au)
// as a lab example for COMP1511
...
```

(`less` is an improved version of an earlier utility called `more`, which shows you long text files one page at a time and allows you to go forward and back using the space bar and the 'B' key. Less is more! Press 'q' to exit less.) Now try to compile kangaroo.c. You should see a list of confusing error messages.

```
$ dcc -o kangaroo kangaroo.c
kangaroo.c:4:1: error: expected identifier or ?(? before ?/? token
 /  by Andrew Taylor (andrewt@cse.unsw.edu.au)
...
```

Your job is to fix the errors.

You'll need to use an editor to change the file `kangaroo.c`

Run gedit like this:

```
$ gedit kangaroo.c &
```

When you have fixed all the errors you should be able to do this.

```
$ dcc -o kangaroo kangaroo.c
$ ./kangaroo

 /\   ___
<__\_/    \
   \_  /  _\
     \_\ / \\
       //   \\
     =//      \==
```

Hint: the error messages are confusing but they usually indicate where the problem is.
Hint: treat it as a puzzle. Look for differences between your program and the working programs you have been shown.

Hint: all the errors but one involve a single missing character.

Hint: if it looks like there are too many backslashes in the program, it is correct. Don't delete any backslashes (you do need to add one backslash). It was mentioned in lectures that the backslash has a special interpretation in C.

Again try the automatic checking script in the class account

```
$ 1511 autotest wk01_kangaroo kangaroo.c
Test 1 (./kangaroo) - passed
1 tests passed  0 tests failed
```

Again don't worry if you fail the checking - and you can't see why.
Submit your work with the `give` command like this:

```
$ give cs1511 wk01_kangaroo kangaroo.c
```

In future weeks both members of each lab pair need to submit their work using `give`.

Exercise 4: Receiving Email from your tutor

Your tutor has sent an email to your official UNSW email account. You need to reply to it. We want to make sure you can receive official emails!

Exercise 5: Setting Up Your Own Computer to Work on COMP1511

Don't panic if you don't own a laptop - you don't need one to study COMP1511.
If you don't have a computer where you live, you will need to spend time in CSE's labs outside scheduled lab classes. You can check lab availability here.

If you finish the above exercises within your lab-time and/or your partner has a laptop with you, try following the instructions on the course home page for *home computing* to set it up so you can work on COMP1511 on your own computer.

It's useful to try and set up your laptop (if you have one) in a lab class because your tutor or other students may be able to help you with any problems.

Otherwise, try to set up your laptop and/or desktop this week so you can work on COMP1511 at home.