Compiling and running programs on the Computer Science Unix machines

Note: If you are doing the G52CPP module then you will know all of this already so can safely ignore this document. The only real differences from the G52CPP module document is that the suggested name of the directory changed from cpp to osc and a C file rather than C++ file is created and compiled (using gcc rather than g++). Both documents are designed to get people started with Linux C/C++ on the Computer Science servers rather than being module specific.

Creating, compiling and executing your first C program

The purpose of this part of the lab work is to ensure that you understand the basics of compiling a C or C++ program, and of using the Unix command line. To simplify matters, your first program will be compiled and executed under Unix . (And so will some of the informal courseworks so it is important that you know these basics.)

The initial programs will be compiled and executed on the Computer Science Unix machines. You should all be aware of how to login to the Unix machines, and how to use the command line to change directory, create directories, etc, but a reminder follows for those who may have forgotten:

The following instructions take you step by step through creating and compiling various simple programs so that people who are not so experienced with using the Unix machines or the command line are not disadvantaged. Sorry if some of this seems very basic to you, I was attempting to make it too obvious rather than raise a lot of questions.

Step 1: Logging in

You must first login to `bann.cs.nott.ac.uk` and get a shell prompt. There are at least two ways to do this from the PCs in the A32 lab, and you may have other programs installed on your own PC, but we will use Putty.

Note on server machines: I have written these instructions to use the Linux machine called `bann`, since this is a level 2 module and that is the machine which is suggested for use by second year undergraduate students. These instructions should work with any of the other Linux machines so please feel free to use the one which you usually use.

Note on Unix shell commands: You will be using simple Unix/Linux shell commands in the first part of these notes. After typing a command you should press Enter/Return to execute it. Steps 2 to 4 give a quick reminder of some of the basics of creating directories and changing the current directory. You can feel free to skip to step 5 if you are happy with logging into the Unix machines and the basics of the Unix shell.

Logging in with PuTTY

There are many ssh clients available. You can download a free ssh client called PuTTY if you wish (see `www.putty.org`). PuTTY is installed on the Computer Science computers, so you can use it in the labs.

On the school computers, open PuTTY. The icon may not be in the desktop, so you may have to find it in the Start Menu.

When the window appears for PuTTY, enter the hostname of `bann.cs.nott.ac.uk` and your university username (i.e. `psXXX`), then click "Connect". Enter your password when prompted to do so.

You should then have a window with a `xxx00u@bann$` prompt, where `xxx00u` is your username.

Step 2: Check where you are:

At the `bann$` prompt, type '`pwd`' (**p**rint **w**orking **d**irectory) to see the directory name of the directory that you are currently in.

```
jaa@bann$pwd
/stug/ug/<your_username>
```

Type '`ls`' to get a list of files.

```
jaa@bann$ls
<A list of the files appears here>
```

Step 2: Check where you are:

At the `bann$` prompt, type '`pwd`' (print working directory) to see the directory name of the directory that you are currently in.

```
jaa@bann$pwd
/stug/ug/<your_username>
```

Type '`ls`' to get a list of files.

```
jaa@bann$ls
<A list of the files appears here>
```

Step 3: Create a directory

You have two options here. Either you can create your directory inside your private directory, or you need to make the directory that you create private (readable by only you). If you do not make your directory private, then anyone can read it. That isn't a good idea when you will be creating your coursework on these machines since anybody will be able to read and copy it.

Option 1: create your directory inside your private directory

First move to the Private directory (type '`cd Private`') then create a directory for your g52osc work: type '`mkdir osc`'

```
jaa@bann$cd Private
jaa@bann$mkdir osc
jaa@bann$ls
<A list of the files appears here, including a directory called osc>
```

Option 2: Make your directory private

First create a directory for your g52osc work (type 'mkdir osc') then use the chmod command to make it accessible to only you:

```
jaa@bann$mkdir osc
jaa@bann$chmod 700 osc
jaa@bann$ls
<A list of the files appears here, including a directory called osc>
```

Step 4: Move into the osc directory

Enter the osc directory:

```
jaa@bann$cd osc
jaa@bann$pwd
/stug/ug/<your_username>/osc
```

If you are using one of the Computer Science lab PCs then you should see that the directory that you have created has also appeared in your home directory in windows explorer (under your h: drive, you may need to refresh the display to make it appear).

In future, when you login (as in step 1 above), you can (of course) skip steps 2 and 3 and go straight to step 4.

Step 5: Create your source code file

You need to create text files with your source code in them. To create a text file you can either use one of the text-only `Unix` editors or edit the files within windows using any text editor (even **Notepad** or **Wordpad**), see below. There are a number of powerful source code editors available, but for the simple programs we are using here a simple text editor will suffice. This allows us to concentrate on the language rather than the editor.

Create a file called `hello.c`, with the following contents:

```
#include <stdio.h>
int main( int argc, char* argv[] )
{
   printf("Hello world!");
   return 0;
}
```

Note: For all of the example programs, you should be able to cut and paste straight from this pdf file, but if you do so then you may need to ensure that the quotation marks are correct. Assuming that you are using a UK keyboard, all double quotes should be the character gained by pressing shift-2, and single quotes the character obtained from pressing the key with the @ symbol on it (without shift). You may find that you learn better by typing the programs in yourself, however, since it may help you to understand the details.

Creating and editing files

You can use any text editor to edit the files. Under `Unix` you can use one of the text-based editors (e.g. `pico`, `vi`, `vim`, or `emacs`).

Assuming that your files are in your home directory, your home directory on `bann` is also mapped as the `H:` drive. This means that you can, if you prefer, create and edit your source code files under windows (you may find this a lot easier). Either **Notepad** or **Wordpad** will be sufficient, however, since **Visual Studio 2010** is installed on the lab PCs you may wish to use that as an editor (we will be using **Visual Studio** later in the module so it may be worth getting to know it now). **Visual Studio** has the advantage that it will do keyword colouring, and shows line numbers (unlike **Notepad**, for example). If you do so, then you need to ensure that you use it only for editing (at the moment), not for compiling (otherwise you would have to know about how to create Visual Studio projects). The Microsoft compiler has some non-standard features and you can NOT guarantee that a program which will compile under it will also compile under `gcc`, or vice versa. In many ways it is more relaxed than `gcc`, in that it will allow you to write non-standard C code and will still compile it. In other ways it is more strict, and will (for example) sometimes complain about the use of standard library functions if Microsoft have produced their own (non-standard) equivalent that they think you should use instead.

Warning for Visual Studio

The following should not happen to you, but in case it does here are some instructions. If you do use **Visual Studio**, and this is the first time that you have done so, then you MAY (but shouldn't) be prompted for a default environment to use. Select the C++ option if that happens. You may then have to wait for quite a long time while it sets up its environment, and it may complain a few times that you don't have administrator access, but eventually it will open your file and will allow you to edit it. Subsequent uses of **Visual Studio** should be much quicker. Note that you do NOT need to close the source file in the editor in order to compile your program (step 6), so you can keep the file open, compile in the shell window, fix any problems in the editor, or make any changes, and re-compile in the shell prompt.

Warning for Notepad

If you created your file under `Unix` and try to open it in **Notepad** the file may all appear on one line. The problem is that the end of line character differs between `Unix` and Windows, and **Notepad** (unlike **Wordpad** or **Visual Studio**) cannot cope with this. You will be fine opening any file which was created using a windows text editor (including **Notepad**).

Warning for Wordpad

When you save your file you need to make sure that you save it as a text (only) file. i.e. choose 'save as' not 'save' and choose the '`text document`' type. If you do not do this, your file will contain formatting information and will not compile.

No warnings for the `Unix` editors

I have no warnings for you for the `Unix` editors. The only disadvantage with using them is that you may not know how to yet. Maybe now is the time to learn?

Step 6: Compile your program

You can compile the file using the following command in the shell window using `gcc` as follows:

```
jaa@bann$gcc hello.c -o hello
```

This will compile the source file called '`hello.c`' and write the output to a file called '`hello`'. The '`-o`' command line option specifies the output filename. If you do not provide this a file called '`a.out`' will be used. This will still contain your program, but will have a less useful name.

(Note: If you compile under Microsoft Windows you may instead get a default file called something else, for example '`a.exe`'.)

Step 7: Execute your program

You can then execute the program as follows, where the '.' refers to the current directory, so this means execute the file called '`hello`' in the current directory. :

```
jaa@bann$./hello
```

Congratulations, you have now successfully created, compiled and executed a C program.