Getting started with UNIX/Linux for G51PRG and G51CSA

David F. Brailsford
Steven R. Bagley

1. Introduction

These first exercises are very simple and are primarily to get you used to the systems we shall be using during G51PRG and G51CSA. Once, you've got things working, you should demonstrate to one of the Preceptors that you can compile and run both of the C examples — and also the ARM assembler examples, under Komodo. The Preceptor may ask you to demonstrate how you might make a simple modification to your program.

2. Deciding on which UNIX/Linux editor to use

You will need to decide on which text editor you want to use for preparing files on a UNIX or Linux system. The three most popular choices are called **`pico`** (some Linux distributions use an open-source clone called **`nano`** — see later), **`vi`**, or **`emacs`**. The simplest of these is **`pico`** which in some respects is an utterly basic editor similar in spirit to 'Notepad' on Windows systems. **However, the main difference between `pico` and 'Notepad' is that `pico` does NOT respond to the mouse cursor**. It is designed for utterly 'dumb' terminals with only the cursor keys available and no mouse at all.

Be aware that almost nothing causes so many arguments among UNIX users as the choice of text editor. Steve will use **`emacs`** in his lectures. I (Dave) will use **`pico`**. It's an intensely personal thing. The choice is yours!.

So, **`pico`** is the editor we shall cover in section 2.1 of this document. If you already have some knowledge of UNIX/Linux and know **`vi`** or **`emacs`** then feel free to use them. All three are available on our Linux servers.

3. Logging on

At any of the (Windows 7) machines in the A32 terminal room log on to the system using the username and password you were given at Registration. Once you are successfully logged in you will see the usual Windows desktop. One of the icons available there is entitled NX and it is your gateway (across the network) into one of our new Linux servers (Note: in what follows we'll use the terms UNIX and Linux interchangeably). Double click on this icon and wait a few moments. You should be projected into an X Windows desktop (X Windows is the underlying windowing technology of UNIX. Its development began around the same time as Microsoft Windows).

Once you are properly connected you should see a display allowing you to open terminal windows on the School's teaching servers. For undergraduate first year use your server is Avon. Note that there is a 'Terminal' option at the bottom left of the NX screen. Click on that to open up a terminal window. Then click on it again to open up a second window. You can drag these windows to places on the screen that feel comfortable for you

Choose one of your windows and move the mouse cursor over it. The rectangle after the prompt string should turn black to show it is ready for you to type a command. So now type:

```
cd Private
```

This moves into your `Private` directory (or folder, as it would be known on a Mac or Windows machine). This directory's permissions are set to ensure that only you can access it. It is advisable that you store all your work in this directory to stop people hacking into your directories to plagiarise your work. Now, we need to create directory to store your *G51PRG* work, so type:

```
mkdir G51PRG
```

This creates a new directory called G51PRG which we'll use for all your G51PRG exercises. To jump down into this directory use the UNIX `cd` command which stands for 'change directory'. So, at the command prompt, just type:

```
cd G51PRG
```

Let's now look at calling up the **`pico`** text editor in this window to create our first C program file.

3.1. Editing with `pico` to create `hello.c`

Note that **`pico`** originated as an editor supplied with the **`pine`** email system developed by the University of Washington. However, **`pico`** has also been reimplemented by the GNU people under the name of **`nano`**. They are functionally very similar. If you don't find **`pico`** on some new UNIX system just try **`nano`** — it may very well be there.

There is a good tutorial on **`pico`** at:

`http://www.ncsu.edu/it/essentials/managing_files/text_editors/pico_tutor/index.html`

So you are in your G51PRG directory and you can now type:

```
pico hello.c
```

The **`pico`** editor window now appears with no text in it. Just start typing and don't worry too much about making mistakes. The whole idea of a text editor is that you can put things right and nothing becomes permanent until you save what you have done. The tutorial cited above will give you detailed guidance but the essentials are as follows (Note that things like **`^X`** mean "hold down the Control key and type X")

- Use the normal cursor keys to wander around the **`pico`** screen
- Remember that **`pico`** does NOT respond to any placements of the cursor that you do with the mouse — use the cursor keys only to get the cursor where you want it to be.
- The Backspace key deletes characters behind the current cursor position
- The `DEL` key, or `^D`, both have the effect of deleting the current character under the cursor.
- Use of `^A` and `^E` zonks the cursor respectively to the beginning, or end, of the current line
- `^K` kills a whole line. Successively killed lines are stored in a "kill buffer" which can be squirted back onto the screen anywhere you want by using ^U. Just position the cursor in the right place first.
- When you are totally happy with what you have created use `^O` (that's `<CTRL>` plus the capital letter 'Oh') to write what you have created back into **`hello.c`**.

- To quit **`pico`** use `^X`. This returns you to the Linux command line
- There is a useful summary of the available ˆ options at the bottom of the **`pico`** screen

4. Compiling your `hello.c` file

Using the editor of your choice you should now create the file **`hello.c`** with the following contents:

```
#include <stdio.h>

int main (int argc, char *argv[])
{
        printf("Hello World\n");
}
```

Immediately, it should be evident that there are two sections in the file. The first section:

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

includes definitions of the standard I/O (input/output) routines in the C language (such as `printf`, which we use to print things out).

In C, all programs begin executing with the procedure `main()` and this is defined in the second half of our example. This procedure takes two parameters (or arguments) that we shall look at later. Inside the `main()` procedure, we find the code it executes. In this example, there is a solitary call to `printf()`, a very powerful library procedure that enables us to print output in a formatted fashion. There are also variations that output into strings and to files.

4.1. The Compiler

The GNU C compiler is a UNIX command-line tool that is accessed by the `gcc` command like this:

```
gcc hello.c
```

This will compile the file `hello.c` into a UNIX executable (i.e. a program) called `a.out` (this default filename is a UNIX compiler convention). This isn't very convenient so we tend to use the `-o` flag to specify a more meaningful output filename.

To compile your file just quit your editor and then type, on the command line:

```
gcc -o hello hello.c
```

Here we use the `-o` flag to name the program `hello`. Recall that if you don't use the `-o` option then your binary file will be called `a.out`. If all is well no errors will show up, and the command prompt appears once more waiting for the next command. Of course, if there are errors, then you will have to use **`pico`** to alter your file, followed by recompiling it all over again with `gcc`.

When there are no further errors just type at the command prompt:

```
./hello
```

or

```
./a.out
```

if you didn't explicitly name your binary file. This will cause your program to execute and print out "Hello World".

If you get this far successfully try modifying your program to carry on and print out “Goodbye Universe!”. If you get stuck ask one of the Preceptors for help.

5. Getting started with `kmd` and `aasm`

In the window where you have been doing your C work just type:

```
cd ..
```

at the command prompt. This returns you to the directory above G51PRG i.e. your ‘home’ directory, where you were initially put at login time. We now need to create a separate directory for G51CSA work this semester. So do:

```
mkdir G51CSA
```

followed by

```
cd G51CSA
```

now move your mouse cursor over to the other window (the one we haven’t used yet) and in that window you now do:

```
cd G51CSA
```

to descend into the new directory that you have just created in the original window.

You now have two windows, both tied to the G51CSA directory at the moment, ready for use with Komodo. You will use one of these for editing your file of ARM assembler commands. The other one will be used for running Komodo itself. In the window you choose for editing assembler files, type the following on the command line:

```
cp /lhome/mod-g51csa/hello.s .
```

This copies over a **hello.s** ARM assembly language program file that will print out ‘Hello World’ once it has been assembled and made into a binary file. The final full-stop in the above command means do the copy “... into the current directory”, which is G51CSA at the moment.

Now move to your other unused window and type in the command:

```
kmd -e
```

to run the Komodo ARM emulator. As you can see, this presents an interactive interface with buttons, using X Windows technology. Press the ‘Browse’ button next to the ‘Compile’ text box. Since you are already in the G51CSA directory the display should show up your **hello.s** file and its full path name appears in the ‘Compile’ text box. Of course, for us, ‘Compile’ really means ‘run the **aasm** assembler on the named file’. This is exactly what it does, on the file ‘hello.s’. If all is well there will be no error messages in the monitor window pane at the bottom of the display. Rather it will just tell you that 2 passes of the assembler have been performed and all is well. The program that has now been produced for you is called **hello.kmd** denoting that it’s not a proper executable on real hardware but is intended only for the Komodo emulator.

Notice that the full path name of **hello.kmd** has appeared in the ‘Load’ text box. So now just press the ‘Load’ button and you’ll see a bewildering array of binary and source code appear in the various Komodo panes. Don’t panic!! Your program is loaded but is not yet running.

Since the program prints out “Hello World” we need a simulated Input-Output window to see this happen. This is provided in Komodo by just pressing the ‘Features’ button on the left of the display. Now press the ‘Run’ button at the top of the screen and “Hello World” should appear in your ‘Features’ window. If you are so bedazzled by this that you want to run it again then press the ‘Reset’ button followed by ‘Run’.

Finally press ‘Reset’ one more time followed by successive presses of the ‘Single Step’ button. Watch the highlighting in the top pane illuminate each low-level instruction, in turn, until eventually, ‘Hello World’ appears once again,

As we’ll find out later, single-stepping, breakpoints and watchpoints are vital tools for debugging assembler programs.

6. Last task

When you have successfully compiled and executed **`hello.s`** go back into your editor window and use **`pico`** (or other editor of your choice) to re-open **`hello.s`** for editing. Now try and figure out how you would alter this program to also print out ‘Goodbye Universe !’ . If you get stuck, just ask a Preceptor — they are there to help.

Once you’ve amended your **`hello.s`** file, move to the Komodo window and use the ‘Compile’ button again, followed by ‘Load’ and ‘Run’ if assembly is successful.

7. Finally

Don’t get alarmed if some of the programming (particularly the ARM stuff) seems bewildering at the moment. The main idea of these first exercises is to get you familiar with using a text editor under UNIX and with a few simple UNIX commands like `cd` and `cp`.

You can ask *any* Preceptor for help in the labs. Later on this week (or early next week) you should know who your personal Preceptor is (this person runs your weekly tutorial and marks your work).

Don’t forget to quit NX by using the Log Out option at the bottom left of the screen. You also need to log out from your MS-Windows terminal before leaving the lab.