Setting Up Your Computer for UCP

Updated: 14th August, 2015

If you have your own computer, you can generally set it up so that you can work on UNIX & C Programming. In fact, you have a few different options:

- Remotely log in to the saeshell0Xp machines.
- Access gcc via the web.
- Install gcc (the GNU C Compiler) directly (if already using Linux).
- Install Cygwin (www.cygwin.com) (if using Windows).
- Install Xcode (developer.apple.com/xcode) (if using Mac OS X).

In addition, if you don't have Linux and want to install it, you can:

- Install Linux on a virtual machine (e.g. VMware or VirtualBox).
- Install Linux natively, standalone. This will erase everything on your hard drive.
- Install Linux natively in a dual-boot (or triple-boot) scheme. This can be tricky, and you should definitely backup everything first.

Warning: You are basically on your own here, except in regards to the lab machines and the saeshell0Xp machines. We don't generally have the resources to help people with their own personal computers.

1 What Do I Actually Need?

Here's the software you'll need to use in UNIX & C Programming:

`gcc` (the GNU C Compiler) – the single most crucial tool. However, you may alternatively be able to use a different C compiler called `clang` (which is used in essentially the same way).

`make` – specifically GNU Make, introduced in Lecture 2 (Environments). This reads and executes makefiles in order to simplify the compilation process.

`valgrind` – introduced in Lecture 3 (Pointers). This helps find memory errors in your running program that are otherwise extremely difficult to detect.

`gdb` – introduced in Lecture 8 (Debugging). This is a general-purpose command-line debugging tool. Also discussed is `ddd`, a graphical front-end to `gdb`.

`bash` – introduced in Lecture 2 and explained more thoroughly in lecture 7 (Shell Scripting). This is the shell program.

`grep` – introduced in Lecture 7. This finds and reports specific patterns (written as *regular expressions*) in files.

A text editor – any editor will do, provided it is a *text* editor (and not a word processor!). Knowing `vim` is very handy, though. See Section 9 on editors.

2 Remotely Logging In (SSH)

This is easy to set up, but you will be restricted to terminal programs only. You must use `vim` (see below) or `nano` to edit your code.

We have four shell machines set up to accept remote logins via SSH ("Secure SHell"):

- `saeshell01p.curtin.edu.au`
- `saeshell02p.curtin.edu.au`
- `saeshell03p.curtin.edu.au`
- `saeshell04p.curtin.edu.au`

On Linux or OS X, type:

```
[user@pc]$ ssh studentID@saeshell01p.curtin.edu.au
```

On Windows, install PuTTY (www.chiark.greenend.org.uk/~sgtatham/putty). Run it and enter "saeshell01p.curtin.edu.au" (or one of the others) in the "Host Name" box.

Warning: What you do on these machines can affect other people using them. Completing UCP practical exercises is okay. But please don't run code that ties up large amounts of CPU time, memory or other resources. If in doubt, ask first.

3 Tranferring Your Files (SFTP)

If you're *not* just remotely logging in, you may need to upload or download your work to/from the University's file server.

To do this, use an SFTP ("Secure File Transfer Protocol") program. There are a few to choose from. For example:

- FileZilla (filezilla-project.org) (works under Windows, Linux and Mac OS X).
- Cyberduck (cyberduck.io) (if using Mac OS X).
- The command-line `sftp` tool.

Connect to any of the saeshell0Xp machines.

4 Online Resources

There are some websites set up to let you compile and run C code online:

- Coding Ground (www.tutorialspoint.com/compile_c_online.php)
- Ideone (ideone.com)

However, these are external services for which Curtin takes no responsibility, and you probably won't get quite the same experience.

5 Setting up Linux

If you already have Linux on your own computer, use the package manager to install the tools listed in Section 1: `gcc`, `make`, `valgrind`, `gdb`, `bash` and `grep`. (Try to run them first to see if they're already installed – `bash` and `grep` in particular often come standard.)

The exact installation procedure varies depending on which package manager you have, and that depends on your Linux distribution:

- "`apt-get`" if you're using Ubuntu, Debian or other related distributions.
- "`yum`" if you're using Fedora, Red Hat or other related distributions.
- Other distributions might use one of these, or their own.

If you're not sure where to start, you first need to find out which distribution you have. Then you can determine which package manager you have, and then you can consult the documentation for it.

Generally, it's no more complicated than:

```
[user@pc]$ sudo apt-get install gcc
```

OR

```
[user@pc]$ sudo yum install gcc
```

("`sudo`" for temporary root access, to give yourself permission to install things.)

6 Xcode on Mac OS X

If you own/use a Mac, you already have a Linux-like terminal environment, but you will need to install Xcode (developer.apple.com/xcode) to obtain gcc.

Warning: I currently can't verify whether Xcode includes `make`, `valgrind` or `gdb`. However, it should be possible to obtain them.

7 Cygwin on Windows

If you own/use a Windows computer, you can install Cygwin (www.cygwin.com). This is essentially a way of running Linux/UNIX software on Windows.

Warning: `valgrind` is officially not supported under Windows at all (even with Cygwin). There may be alternatives available, but I haven't been able to test them.

You can at least get `gcc`, `gdb` and the Linux terminal environment:

1. Download, install and run `setup-x86_64.exe` (or `setup-x86.exe` if your machine is *old*).
2. Press "Next" until you see the "Select Packages" screen – a huge list of all the things that can be installed under Cygwin.
3. Find `gcc-core`, `make` and `gdb` (in the `Devel` section) and ensure that their check-boxs in the "Bin?" column are crossed.

   For `make`, find "The GNU version of the make utility". `bash` and `grep` should already be pre-selected for installation.
4. (Optional) find `vim` and/or `gvim` in the Editors section, and select them too. (You may not choose to use `vim` now, but there's no harm in having it).
5. Press "Next". Cygwin will then (probably) tell you about resolving dependencies. It knows what it's doing (hopefully), so just press "Next" again, and it should install itself.

When you run Cygwin, you'll get a `bash` shell. This is quite different from the built-in Windows shell (`cmd.exe`), and basically works as on Linux.

Cygwin views your disk drives (C:, D:, etc.) as directories inside `/cygdrive`. So, for instance, to see the contents of C:, you type:

```
~$ ls /cygdrive/c
```

To get to your `Documents` folder, say "`C:\Users\`**`username`**`\Documents`", type:

```
~$ cd /cygdrive/c/Users/username/Documents
```

(Note the different kinds of slashes.)

8 Installing Linux

The instructions for doing this are far too complex to reproduce here.

If you want real Linux (and don't already have it), installing it on a virtual machine is usually the easiest and safest way. We don't (yet) have a virtual machine image available that mimicks our lab machines, so you need to create or obtain your own.

You can also generally install Linux *natively* on any computer. However, this can be risky:

- You'll be messing around with hard drive partitions. In doing so, it's very easy to permanently, irrevocably delete the one with all your data on it.
- Particular wireless/wired networking cards (or other hardware devices) can be painful to get working in Linux. It's best to check compatibility before you start. (See if anyone else has tried to install Linux on your particular model of laptop, for instance. Which distribution of Linux did they try, and what, if anything, went wrong?)

9 Editors

There are quite a few available editors.

Warning on IDEs: Professional software developers use "Integrated Development Environments" (IDEs), such as:

- NetBeans (netbeans.org)
- Eclipse (eclipse.org)
- Visual Studio (www.visualstudio.com)

However, this unit in part requires you to know how to use command-line development tools: `gcc`, `make`, `valgrind` and `gdb`. Using an IDE, you will not be exposed to this level of detail (particularly the construction of makefiles).

Here are the non-IDE editors installed on the lab machines:

`vim` – a terminal-based editor (see below),
`nano` – another, very simple (perhaps too simple) terminal-based editor,
`emacs` – another, very complex (perhaps too complex) terminal-based editor,
`gvim` – a graphical (non-terminal) version of vim,
`gedit` – the standard editor in the GNOME environment,
`kate` – the standard editor in the KDE environment (the KDE Advanced Text Editor),
`kwrite` – a cut-down version of `kate`,
`jedit` – a cross-platform editor.

It is easy to find text editors for any OS. jEdit (www.jedit.org) should work anywhere. Notepad++ (notepad-plus-plus.org) is a decent choice for Windows.

Note: the version of Notepad that comes standard with Windows is the single worst text editor I know of.

Notes on vim

Explaining how to use vim is beyond the scope of this document, but there are many online resources (including handy quick reference sheets). Vim has (or can have) a configuration file called `~/.vimrc` (i.e. "`.vimrc`" in your home directory). You can put settings in here that make editing code a little nicer.

Here's my `.vimrc`, for instance:

```
set smartindent
set shiftwidth=4
set tabstop=4
set expandtab
set showbreak=>>>
:syntax on
```

End of Worksheet