UNIX Basics

This handout was written by Andy Gray, and updated by Justin Haugh and Jerry Cain.

This handout covers some of the basics of using the Leland UNIX environment to edit, compile, and submit your programs:

- ⇨ Logging in, class directory, starter files
- ⇨ Editing your source files
- ⇨ Compiling your program
- ⇨ Submitting your program

If you find that you need more individualized help getting started, be sure to come to office hours next week. Office hours will be posted to the web site in the next few days or so, after the CS107 TAs have finalized their own schedules.

Getting Started

All of you will use the **`leland`** machines to submit your work. Most of you will probably do all of your development on the **`leland`** systems as well. The tools on the UNIX machines are excellent, but somewhat more difficult to get started with. Although we will be able to offer some support for those of you using your own PC or Macintosh, we suggest you use the UNIX tools whenever possible. Even though ANSI C/C++ is highly portable, we will be grading your submissions on the **`leland`** systems, so it is your responsibility to make sure that your program written on another platform works on the **`leland`** machines as well. In addition, working on the **`leland`**s makes your files available from any remote machine and backs up your work each night.

Once you've logged on, you'll find yourself in your home directory. Here's a quick overview of changing directories, creating directories, and copying files. If you're completely new to UNIX, it would probably be a good idea to stop by office hours so that one of us can show you around. The best UNIX reference for this class is CS107 Handout 6: UNIX Development, available on the website for those who didn't pick up a hardcopy. I'm also a fan of the following Stanford site, which summarizes of good bit of UNIX relevant to CS107:

`http://unixdocs.stanford.edu/`

The "UNIX Command Summary" handout is particularly useful.

You can create a new directory for all of your projects with the following (text that you type is bold):

```
myth5> mkdir cs107
```

and then change to the directory with

```
myth5> cd cs107
myth5>
```

Notice that the prompt changes to reflect the directory you are currently working in. (The **myth5** part indicates the name of the machine we're currently logged into, but there's nothing particularly special about this machine.)

The directory for class materials is **/usr/class/cs107/**. In here you'll find starter files that we provide for the assignments. You'll also find code samples. Let's copy one of the sample programs over to your brand new **cs107** directory:

```
myth5> cd /usr/class/cs107/assignments
myth5> ls
assn-0-small-programs  assn-0-small-programs-data  assn-1-rsg  assn-1-rsg-data
myth5> cp -r assn-0-small-programs ~/cs107
```

The last command copies all of the files in the sample directory recursively **(-r)** into your **cs107** directory (**~** is the same as your home directory, whatever it may be). Now, we've copied the sample directory over to your own **cs107** directory.

Later when it's time to start working on the real assignments, you'll copy the starting files from **/usr/class/cs107/assignments** the same way.) You can get to your copy of the **assn-0-small-programs** directory this way:

```
myth5> cd
myth5> cd cs107/assn-0-small-programs
myth5> ls
01-ruth-aaron   02-word-games
myth5> cd 01-ruth-aaron
myth5> ls
Makefile     README      ruth-aaron-numbers.c
```

Now that you've copied these files over, you can start editing the source code and compiling the program.

Editing Files

There are a number of editors you can use in the UNIX environment, but we'll just go through a brief overview of the most popular, **emacs**. Although it's incredibly powerful and really nice to use once you've figured it out, **emacs** can be a bit intimidating at first. If you go to Terman or Gates to work (or run an X server on your local machine), you

can run **emacs** in a graphical mode or use its sister program **xemacs**. Both **emacs** and **xemacs** support plain old **telnet** and **ssh** sessions too, but you'll need to rely on the command keys since you won't have any menus to use.

To run **emacs**, you just type **emacs** at the prompt (or **xemacs** for, surprisingly enough, **xemacs**):

```
myth5> emacs
```

This brings up the program. If you're physically in front of a UNIX machine, a new window will pop up. If you're running it over ssh, it will just take over the telnet window. You can open a file for editing by using the **Files | Open File**... menu, or you can type **C-x C-f** (where **C** is the control key) and then the name of the file you want. If that seems a bit cryptic, it is. If you're new to this, you'll find it easier if you use the menus at first and then gradually introduce the keyboard shortcuts into your routine until you don't need the menus at all.

To open a file for editing on the command line when you start the program, just type the name(s) of the file(s):

```
myth5> emacs  ruth-aaron-numbers.c  &
```

This brings up **emacs** with **ruth-aaron-numbers.c** already in it. Here's a little tip if you work at the computer cluster or run an X server: typing an ampersand at the end of a command puts it in the background and frees up the command prompt for other tasks. Otherwise, the shell is tied up with **emacs** and you don't get the prompt back to do things like type **make**. But if you try this when you're running **emacs** in a telnet window, the program is immediately suspended and it looks as if it didn't start at all. Now you have the sample C program open for editing. You can take a look at the code, and make changes to it if you want to.

Compiling

Now the real fun begins. If you've used Makefiles in other classes, then there's really nothing new to learn here. Actually, even in CS107 we won't require you to fully understand how Makefiles work. For each of the assignments we will provide you with a Makefile for the project (although in some cases you may need to add your own files to the **SRCS** line). To compile with a Makefile, go to the directory that contains the source files and the Makefile itself, and run **make**:

```
myth5> make
gcc -g -Wall -MM ruth-aaron-numbers.c > Makefile.dependencies
gcc -g -Wall   -c -o ruth-aaron-numbers.o ruth-aaron-numbers.c
gcc -o ruth-aaron-numbers ruth-aaron-numbers.o -lm
myth5>
```

If all goes well, you'll get a few lines of intermediate output but no warnings or errors. If none of the source files have changed since the last compile, **`make`** will let you know (this often comes up when you forget to save your changes before recompiling).

Sometimes (for other classes), you may see **`gcc`** above instead of **`g++`**. **`gcc`** is the GNU C Compiler. It's actually both a C and C++ compiler (C++ is a virtually a strict superset of C), but if you invoke it as **`g++`** you have a better chance of getting it to recognize your C++ code as such (as opposed to C code) and linking against the right libraries. If you stick to using the Makefiles that we provide, then you won't need to worry about this issue. If you have questions about how to edit Makefiles, email us at the question queue and we'll be happy to help you.

To clean up your workspace (delete `.o` object files, etc.) run **`make clean`**. The submission script described below will require you to do this before turning in your work.

Submitting

Before you submit your program you will need to write a **`README`** file that describes your submission. In it, you should describe how to run your program, and note any extra features and bugs (if any).

Once you've finished writing your program, there's just one more step you need to complete: electronic submission. To submit your assignment, change to your assignment directory. Then run the submit script:

```
myth5> /usr/class/cs107/bin/submit
```

The script presents a list of assignments and asks you to choose which one you are submitting. It then copies source files, the Makefile (if any), and **`README`** from the current directory to our submit directory. If after submitting you decide that you need to make some additional changes, you can run the submit program again and send us the new version of your code. We will look at the last submission we receive from you (and we'll determine the number of late days based on when you turned in the last submission). Submit instructions are also available at