A Simple UNIX Warmup

In lab we will be using the UNIX shell for most programming exercises. If you have never worked in this environment before, it is not too hard to come up to speed. The exercises below can help get you started.

Objectives:

- Getting familiar with the UNIX environment
- Compiling and running a simple program

Most of you should already have a working CS account. However, if you do not yet have one or are simply unsure, flag down a TA.

If you do not yet have a CS account, try to follow through the steps of the lab with a neighbor who is doing the exercises. Or, if you happen to have a Mac or a Linux personal laptop with you, you can do the lab from there.

Steps:

1. Login to your account on the lab Linux machines. Find the ***Terminal*** application and start it.
2. Inside the terminal window you will be able to type commands; Let's start with a few:
   - Type in the command "`pwd`" (without the quotes) and press return. This command is for "Present Working Directory"
   - Now try "`ls`"
   - Try "`whoami`"
   - Try "`ls ..`" (what do you think ".."means?)
   - Let's make a subdirectory with these commands:
     - `mkdir cs251`
     - `ls`
     - `ls -a`

     (discuss with your neighbor what the output might mean).
   - Now let's move into the subdirectory; enter the following commands:
     - `cd cs251`
     - `pwd`
     - `ls`
     - `ls ..`

       (hmm: what do you think "`..`" might mean?)
     - `pwd`

Ok, now we have some understanding of the directory structure, how to figure out where we are, create subdirectories and navigate the structure).

Let's write a `C++` program!

EDITORS: First you'll need to pick an editor. Three of the most popular terminal based editors are:

> `vi` (and the `vim` variant)
> `emacs`
> `pico` (not as powerful or as popular but easy to get started in).
> `Visual Studio Code` (not to be confused with the full-blown IDE `Visual Studio`)

`Visual Studio Code` is installed on the lab machines. You can find it under the `Applications` menu. It is more "modern" than terminal-based editors like vim which may be more to your liking.

We want to create a text file called `hello.cpp` and containing the following:

```
/* Hello World program */

#include <iostream>

using namespace std;

int main(){
  cout << "Hello World" << endl;
  return 0;
}
```

First, assuming you will use `vim`, you will need to know the basics.

> Enter the command "`vimtutor`". It will print to the screen a crash course on the very basics of *vi* -- read it! The most important thing to understand is that vi has "modes" -- in particular a ***command*** mode and an ***insert*** mode.
>
> Or, spend a few minutes with the following nifty `vi` interactive tutorial:
>
> > **http://www.openvim.com/tutorial.html**
>
> This looks like fun too: **https://vim-adventures.com/**

A bare bones usage of vim to create the hello.c file is:

type “`vim hello.cpp`”

you will now be in the `vim` editor with an empty document

hit the **‘`i`’** key. You are now in insert mode (you start out in navigation mode).

Now type in the program text above as normal as if you were using a typewriter.

When you are done, hit `ESC`. This will put you back into navigation mode.

Now type **‘`:w`’** This saves (writes) your file. Notice also that the colon brought you to the bottom of the terminal window -- this is where you see commands in navigate mode. Vim already knows that your file is supposed to be called `hello.c`.

Now enter **‘`:q`’** to quit. This will terminate `vim` and bring you back to the command line.

Some additional `vim` functionality you might find useful:

(in navigate mode):

`ctrl-f` advances you one "screen" forward in the file.

`ctrl-b` moves you backwards one screen.

`:set nu` enables line numbering

`:set nonu` disables line numbering

`:<line-num>` moves you to the specified line

`:$` moves you to the last line in the file

`w` advances you one "word" in the file

`A` moves you to the end of the current line and puts you into insert mode.

`/<some-string>` searches for the specified string

Ok, we’re back at the command line. Let’s look around. Try these commands:

```
ls
ls -l
pwd
```

Try this:

```
cat hello.cpp
```

Looks like the file really exists! Let’s compile it. Compile? Yes -- run another program (called a compiler) which reads `hello.c` and translates it into an “executable file” containing code the machine understands directly. Let’s try the compiler `clang` (the compiler gcc is more well known, but clang is very good and gives nicer error messages).

```
g++ hello.cpp
```

What happened? Any error messages? Any warnings? Hopefully nothing more than a warning.

Let look around again:

```
ls
```

You should see a new file called `a.out.`   That’s the executable we were talking about. Let’s run it! Type the following:

```
./a.out
```

Ta-Da - we have a working program!

Now do some exploring. Some of the most basic and useful UNIX shell commands to try:

`mv` (for moving files)
`cp` (for copying files)

Try the ‘`which`’ command like:

```
which g++
which vim
```

Kind of neat.

Exploring Further

Editors

An interactive VIM tutorial

A nifty visual cheat sheet

UNIX Shell

A cheat sheet.

A nice tutorial

Some random things:

Some folks find keyboard focussed editors like vim to be more efficient/productive than editors that take a lot of mouse input. Why? Because you aren't always moving your hand off of the keys to work the mouse and then re-orienting your fingers on the keyboard!

In the spirit of "keeping your fingers on the keys," try to get in the habit of using the **h, j, k, l** keys for navigation in vim instead of the arrow keys.