Start VMware Workstation. Click "Power on this virtual machine", which should be the virtual machine you loaded on the first day of class.

You should see something similar to this mysterious screen:
So, our plan today is to run C programs in a Linux environment using VMware. As mentioned previously, VMware allows us to “play” another operating system as though it were just another application. Just as I can have a window open with a PowerPoint application running, and a window open with a Firefox application running, I can—using VMware—have a window open with the Linux operating system running.

So… shall we write some C programs?

Well… not quite yet. First let's refresh ourselves on how the Linux directory structure is organized.

First, we will usually work within the terminal window. The terminal window is shown below.

You enter commands at the prompt. The prompt in the picture above is what says

```
midshipman@EC310:~ $ 
```

All users of a Linux operating system have an account name (also referred to as a user name or a login name) and a password. When your Linux account is created, you are also given a home directory where all of your files and folders will reside. Your home directory has the same name as your account name.

You may be wondering: Hey, I’m right now using Linux in EC310 and I was never asked for an account name and password while logging on? That is because your textbook author (Jon Erickson) has set up your VMware software to provide Linux “already open” for you. We have, however, changed your account name to midshipman since that is, after all, your first name.

Even more specifically, the command line interface (where we enter commands) is called the bash shell. Every time you enter a command, you are entering the command at the bash shell’s prompt. The bash shell’s prompt for ordinary users is the dollar sign. Before the prompt, you will see your account name and your computer's name.

```
midshipman@EC310:~ $ 
```

Your account name  Your computer's name  The prompt

There is one additional item in the picture above that you may have noticed: the tilde symbol (~). The tilde is an abbreviation for your home directory. When you log in, you are placed by default in your home directory.
Suppose you wander up to a computer and notice that someone is logged on, and you see

then the user whose account name is *jose* has logged in but has forgotten to log out. What a dope. Shame on him. He is evil. Evil Jose.

If you ever forget who you are, even though your account name is staring you in the face, you can enter:

    whoami

as shown below:

Go ahead... Enter **whoami** (you know you want to ) and confirm that you are indeed the user named *midshipman*.

In Linux, just as with Windows, there are files. And in Linux, just as with Windows, there are directories (in Windows terminology, these are referred to as *folders*), which hold files (or other directories).

A Linux system (like a Windows system) may support multiple users. In such cases, each user is given his own home directory. When you logon, you are automatically placed in your home directory. When Jose logs on, he is automatically placed in his home directory. Your home directory is the natural location for any directories or files that you create. You can leave your home directory and move to other directories. Whatever directory you find yourself in, that directory is termed your *working directory*.

A portion of your Linux file system (also called a *directory structure*) looks like this:
At the very top is the root directory, denoted /. The root directory contains all directories and files.

Every Linux system has a special user named root. The root user is the great-and-all-powerful system administrator of the Linux system. The root user can access any file on the system, including the files of individual users. The root user can read the files of all users, can write over any files, and can delete any files. The root user can load any software onto the system (e.g., programs). The root user owns the system.

The dream of all hackers is to somehow become the root user. In Linux, the root user has a special prompt, the pound sign (#). If you walk up to a computer and see this:

![root@EC310:// #](attachment:image)

that means the root user has logged in and left the computer unattended. That would be bad, since that would mean you could look at all files on the system (for all users) and add any software you like to the system (including malware).

**Listing Files** You can list the contents of the working directory by using the ls command.

**Changing Your Working Directory** Right now, your working directory is the same as your home directory: midshipman. Suppose you want to change your working directory to booksrc (see the directory structure shown on the prior page). To do this, type:

```
cd booksrc
```

When you change your working directory, the command line will update to indicate your new working directory. You should now see your prompt as:

```
midshipman@EC310:~ $ cd booksrc
midshipman@EC310:~:/books$rc
```

Working directory changed to ~/booksr

c

At any time, you can travel "up" one directory by typing

```
cd ..
```

Go ahead and enter this: That is, the two letters cd, followed by a space, followed by two periods (with no space between the periods). Note that cd stands for change directory. Since you were in the booksr directory, you should have moved up one level back to your home directory. In other words, you should see this:

```
midshipman@EC310:/booksrc $ cd ..
midshipman@EC310:~ $ 
```

If you are navigating around the directory structure, and you forget where you are, you can enter the command pwd which stands for print working directory.
If you find yourself lost in the file system, you can instantly reset your working directory back to your home directory by simply typing `cd` by itself (i.e., *without* the two periods). Go ahead and type

```
cd
```

and confirm you are back in your home directory.

**Part 2: Your First C Program**

Now where were we... *Oh yes!* The C program!

Before we begin, type the following at the prompt: `cd work`. That is, enter what is shown in bold below:

```
midshipman@EC310:~ $ cd work
```

This is the prompt  This is what you enter

Your prompt should now be
```
midshipman@EC310:~/work $
```

We will enter all of our programs in the *work* directory.

**Question 3.** List the current contents of your *work* directory.

We are going to enter our C program using a simple text editor named *nano*. Let’s name our first C program `lastname_2_1.c` where you use your own last name. So, if my last name is `smith`, I would name this program `smith_2_1.c`.

So, enter the following at the prompt (again, using *your own name* instead of “smith”):

```
midshipman@EC310:~/work $ nano smith_2_1.c
```

This is the prompt  This is what you enter

You should see the editor opened with the correct file name at the top as shown below.
Now, carefully type the program below into the file `smith_2_1.c`

```
#include <stdio.h>
int main()
{
}
```

Your screen should look like this:
Now we want to save this file. In *nano*, to save a file we use Control + o (that is, we press the Control key and the small letter o key at the same time).

At that point *nano* will ask you if you want to still save the file under the original name. Just hit enter.

Now, exit from *nano* by using Control + x. You should be back to the terminal prompt:

![EC310 Terminal](image)

You may now be wondering: How do I know the file that I just typed exists? Where is it?

You already know the answer! To see all of your files enter the letters `ls` at the command prompt. So, at the prompt, type `ls`:

```
midshipman@EC310:~/work $ ls
```

and you should see:

![EC310 Terminal](image)

So, your C file is there!

Now remember, your C program is source code. The CPU does not understand source code—it only speaks in machine language. So we have to compile the source code into machine language using a compiler.

The compiler we will use is *gcc*.

So, at the prompt, compile your program by typing in *gcc* followed by the name of your C program. For me, I will enter:

```
midshipman@EC310:~/work $ gcc smith_2_1.c
```

Looking at the resulting screen…it looks like nothing happened. Linux just went right back to the prompt.

Ask yourself… What should have happened? Linux should have created a machine language file. Did it?
Linux automatically names the output of the C compiler as `a.out`. So...do you have a file named `a.out`?

Let’s see. Type `ls` at the prompt and check if you have a file named `a.out`.

Alright...what do we do after we compile our program? We execute it! To execute the program, we simply type a period, followed by a slash, followed by the name of the executable code (the machine language file) at the prompt and hit enter. So, we should type

```
midshipman@EC310:~/work $ ./a.out
```

and hit enter. However, since we’ve written our program with no useful lines of source code, it does nothing.

**Part 3: A C program that does something!**

It’s kind of a law of Computer Engineering that your first real program has to be a program that prints the message *Hello World!* to the monitor. Since this is the Naval Academy, we will modify this and write a more appropriate program that prints the message *Hello Cruel World!* to the monitor.

Here are the steps.

(a) Using `nano`, open your existing file (for me, this file is `smith_2_1.c`).

(b) Modify your source code so that it will print the desired message with a blank line above and a blank line below the message. You should only need to add a single line of code to the program: a single `printf` statement. Remember that to get a new line you use the escape sequence: `\n`.

(c) Compile your program.

(d) Run your program.
UNIX Tips and Tricks

Are you getting tired of typing in the exact same commands again and again?

As you develop a file, you may have to open it for editing numerous times (by typing `nano smith_2_1.c`), you may have to compile it again and again (by typing `gcc smith_2_1.c`), and you may have to execute it again and again (by typing `./a.out`). Engineers often find themselves typing the same commands again and again and again.

In an effort to make life easier, UNIX (well, actually it’s the bash shell) remembers the recent commands that you have entered at the command line. You can view the recent commands that you have entered by pressing the up-arrow; each press of the up-arrow moves us back by one earlier command. So, say you want to enter

```
nano smith_2_1.c
```

and you know that you have recently entered this command. You can press the up-arrow repeatedly until you find the command, and then hit enter. UNIX will treat this as though you have typed in the command and pressed enter.

You are urged to try this (since it allows you to avoid a huge amount of repetitive typing). Ask your instructor for help if, after reading this, you do not understand this feature.