

# HOW TO Use Linux, Text-Edit, Compile, and More

Using the Linux/UNIX operating system (to work in your eng101 directory):

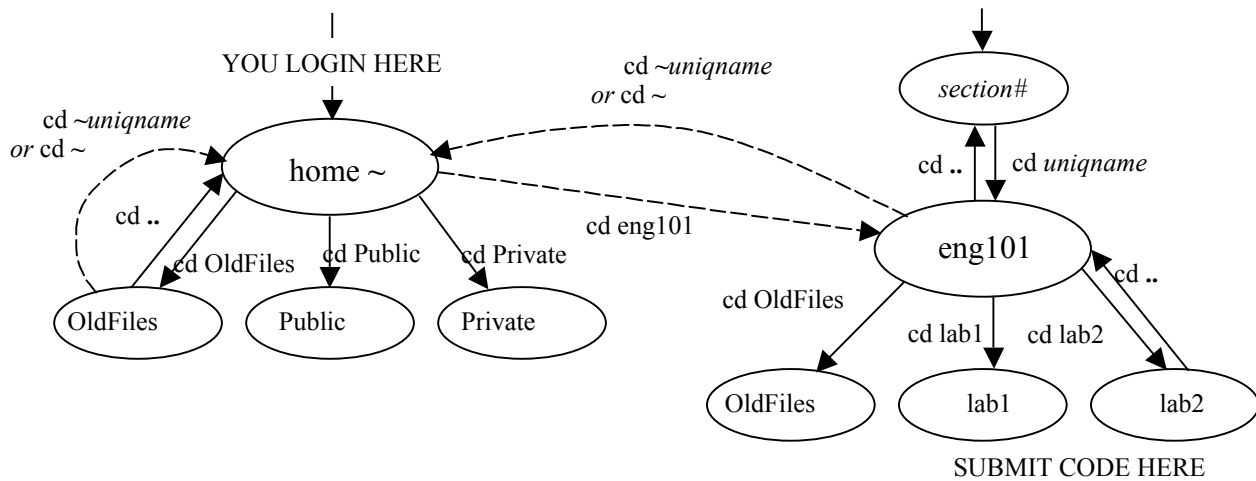


Figure 1. Diagram of the Linux/UNIX directory structure. The ovals represent directories and the arrows show where the cd commands take you. The dashed arrows represent links. Ovals underneath other ovals represent subdirectories whereas the ovals overhead represent parent directories.

1. Start a terminal window by right-clicking on the desktop then select **New Terminal**. You will see a prompt, which is the name of your computer, like this:  
**ruby%**
2. To display the path of your working directory (or where you are), at the prompt type:  
**ruby% pwd**  
Your home directory path should be:  
**/afs/engin.umich.edu/u/firstletter/second/username**  
This is where you keep your personal (non-Eng101) files, up to 200Mb worth.
3. To change your working directory, type:  
**ruby% cd Public**
4. In this case we've changed to the Public directory. Again, type:  
**ruby% pwd**  
You should see something that looks like:  
**/afs/engin.umich.edu/u/firstletter/second/username/Public**  
Anything in this directory can be accessed by anyone else.
5. To make a web-accessible subdirectory (sometimes called a folder), type:  
**ruby% mkdir html**  
Note that Linux/UNIX is case-sensitive. The name html is different than HTML. The html directory in your Public directory can be accessed by anyone via the WWW.
6. To list the contents of your working directory, Public, type:  
**ruby% ls**  
You should see your html subdirectory now created.
7. To change to your html subdirectory, type:  
**ruby% cd html**
8. To change back to your Public parent directory, type:  
**ruby% cd ..**  
'..' refers to the folder which contains the current folder, up one level, up one level

## Some useful Linux/UNIX commands:

<b>pwd</b>	prints path of <u>w</u> orking <u>d</u> irectory	<b>cp</b> <i>from to</i>	<u>c</u> opies a file
<b>ls</b>	<u>l</u> ists directory contents	<b>mv</b> <i>from to</i>	<u>m</u> oves a file
<b>ls -l</b>	<u>l</u> ists <u>l</u> ong detailed directory contents	<b>rm</b> <i>name</i>	<u>r</u> emoves a file
<b>cd</b> <i>name</i>	<u>c</u> hanges <u>d</u> irectory (into subdirectory)	<b>mkdir</b> <i>name</i>	<u>m</u> akes a <u>d</u> irectory
<b>cd ..</b>	<u>c</u> hanges to parent <u>d</u> irectory	<b>rmdir</b> <i>name</i>	<u>r</u> emoves a <u>d</u> irectory
<b>cd ~</b>	<u>c</u> hanges to your home <u>d</u> irectory	<i>command --help</i>	shows options <u>h</u> elp

## Using the Nedit graphical text editor (to write text files and C++ source codes):

1. To change back to your home folder, just type:  
**ruby% cd**  
This always goes to your personal home folder
2. You also have a class directory. To get to this type:  
**ruby% cd eng101**  
Use this folder to submit projects. The professors and GSIs can read anything in this folder. All work on class programs should be done here so they can help you! This is not in your home folder – use pwd to check that out. The eng101 folder you saw in your home directory is really a 'link' to this directory.
3. Create a folder for your first lab assignment  
**ruby% mkdir lab1**  
Also create folders lab2 and lab3.
4. Go back to your html directory  
**ruby% cd ~/Public/html**  
~ is a shortcut meaning your home directory. So 'cd ~' is the same thing as just 'cd'
5. Start up the NEdit text editor.  
**ruby% nedit index.html &**  
The & (ampersand) allows NEdit to run in the background, detached from the terminal.
6. Use the menu with the mouse to open, edit, and save your text.

```
<html>
<head><title>My Site</title></head>
<body>
<h1>My Name</h1>
<a href="resume.html">My resume</a>
</body></html>
```

7. Make sure you save your file as index.html. Now open up a browser, and point to **http://www-personal.engin.umich.edu/~uniquname**

Other graphical text editors you may try are **kwrite**, and **gedit**.

8. Now use the editor to create a `resume.html` file in your `Public/html` directory. Try to experiment with different html commands when creating your resume page.

## Some useful HTML/XHTML in addition to the basic <html>, <head>, <title>, <body>:

<code>&lt;b&gt;text&lt;/b&gt;</code>	<u>bold</u>	<code>&lt;i&gt;text&lt;/i&gt;</code>	<u>italics</u>
<code>&lt;u&gt;text&lt;/u&gt;</code>	<u>underline</u>	<code>&lt;p&gt;text&lt;/p&gt;</code>	<u>paragraph</u>
<code>&lt;br&gt;</code> (option: <code>&lt;br /&gt;</code> )	<u>line break</u>	<code>&lt;h1&gt;text&lt;/h1&gt;</code>	<u>header size 1</u>
<code>&lt;ul&gt;</code> <code>&lt;/ul&gt;</code>	<u>unordered list</u>	<code>&lt;h6&gt;text&lt;/h6&gt;</code>	<u>header size 6</u>
<code>&lt;ol&gt;</code> <code>&lt;/ol&gt;</code>	<u>ordered list</u>	<code>&amp;amp;</code>	<u>ampersand symbol &amp;</u>
<code>&lt;li&gt;text&lt;/li&gt;</code>	<u>list element</u>	<code>&amp;lt;</code>	<u>less than symbol &lt;</u>
<code>&lt;a name="label"&gt;</code> <code>&lt;/a&gt;</code>			<u>anchor name/label</u>
<code>&lt;a href="#label"&gt;text&lt;/a&gt;</code>			<u>anchor link to label</u>
<code>&lt;a href="page.htm"&gt;text&lt;/a&gt;</code>			<u>anchor to local page</u>
<code>&lt;a href="http://www.somewhere.com"&gt;text&lt;/a&gt;</code>			<u>anchor to www</u>
<code>&lt;a href="mailto:email@address"&gt;text&lt;/a&gt;</code>			<u>anchor to email</u>
<code>&lt;img src="link.jpg" alt="text"&gt;</code> <code>&lt;/img&gt;</code>			<u>image</u>
<code>&lt;body bgcolor="#00FFFF" text="#000000"&gt;</code>			<u>body tag with color</u>

Note, some HTML element end tags are optional (`</p>` and `</img>`, `</a>` after name anchor). XHTML is a variation of HTML, and XHTML requires all element tags be closed with an end tag, or a trailing backslash. Therefore a line break in XHTML can be written as `<br />`.

Color codes are composed of three 2-digit hex numbers, one for each RGB color. For web use, each color should have one of six values: 00, 33, 66, 99, CC or FF.

## Using Nedit again to write, compile and run a C++ program

1. Edit the file hello.cpp  
**ruby% nedit hello.cpp &**
2. Type the following, then compile, run, and print it as described below:

```
#include <iostream>
using namespace std;

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

3. Save your program has hello.cpp. Use ls to make sure it is there.

The above text is really what is called "C++ code" or "C++" or "code". C++ is the name of the language we use to give instructions to the computer. The computer is not smart enough to understand English, so we must use the invented language C++. "Code" is just a word to describe any text or instructions that are written in a computer language.

4. Compiling is how we turn the human-readable text file we just made into a machine language file a computer can execute. To do this, we use a program called g++. To compile your program, type:

**ruby% g++ hello.cpp -o hello**

I know this looks complicated but don't get overwhelmed! Let's break it down.

**g++:** This is just the name of the command (just like `ls`, `cd`, or `mkdir`). This command runs the compiler.

**hello.cpp:** This is the name of the file you just made! This is an argument to `g++` to tell it what file to compile.

**-o:** This is what is called a "flag". You will see this a lot in Linux commands. It means "the next argument means something special." In this case, the `-o` means that the next argument is going to be the *output* of the command. Remember that the compiler converts a C++ text file into a machine language file, so the output is the machine language file, or the "executable."

**hello:** This is a word we made up to name our executable. Notice that we now have two files. The C++ file is called "hello.cpp" and the executable is simply called "hello".

After the compiler finished, look at the screen. When you are compiling a program, no news is good news. If you got an error (called a "compilation error" because it occurred during the compile step.) You will have to go back and edit your code. For now, just make sure you typed it in exactly as above. Soon we will discuss what the compiler errors mean and how to fix them.

5. Once you have successfully compiled, type:

**ruby% ls**

Verify that your new "hello" executable is in the current directory.

6. Finally, we get to run our program! To do this, we just type the name of our program. (Notice how we have been using other commands such as `cd`, `mkdir`, and `g++`, and now we have a *new* one called hello!)

**ruby% hello**

You should see the text "Hello World!" printed to the screen! You just made your first C++ program!

### Using the g++ compiler (to make executables of your C++ program):

1. In a terminal window, change to the directory that contains your code, and type:  
**ruby% g++ filename.cpp -o executablename**  
where "**filename.cpp**" is the name of the C++ source code ready to be compiled, "**-o executablename**" denotes the outputted executable program name.  
If no "**-o executablename**" is given, the executable is named "**a.out**" by default.  
ex: **ruby% g++ myprog.cpp** (executable is named "**a.out**")  
ex: **ruby% g++ myprog.cpp -o runprog** (executable is named runmyprog)
2. To run the compiled C++ program, just type the executable name:  
ex: **ruby% a.out** (if no executable name was specified at compiling)  
ex: **ruby% ./a.out** (ensures program in working directory is run)  
ex: **ruby% runprog**
3. Other `g++` options you might use:  

<code>-s</code>	“strips” the executable, making it smaller (rarely needed in eng101)
<code>-O2</code>	turns on level 2 optimization, makes your program run faster
<code>-Wall</code>	displays “all Warning” messages when compiling

## Using the enscript, lpq, and lprm print commands (to print your text files and C++ code):

To print a text file, typically change to the directory that your file is in, then type:

```
ruby% enscript options -Pprintername path/filename  
ex: ruby% enscript -Pb505pierpont myprog.cpp (simplest usage) or  
ex: ruby% enscript -2Gr -Pb505pierpont myprog.cpp  
      (with 2-column, fancy header, landscape formatting)
```

Printer names can be found on or near the physical printer in any CAEN lab like:  
**b505pierpont**, **b507pierpont**, and **b521pierpont**.

You may also use the non-formatting standard **lpr** print command instead.

```
ex: ruby% lpr -Pb505pierpont myprog.cpp
```

To check the print queue, type:

```
ruby% lpq -Pprintername  
ex: ruby% lpq -Pb505pierpont
```

To remove your own print job from the print queue, type:

```
ruby% lprm -Pprintername jobnumber  
ex: ruby% lprm -Pb505pierpont 30
```

## Using additional resources:

CAEN provides excellent technical notes on the following topics: *Email • AFS • Printing • Communications • Databases • Connecting to CAEN • UNIX • Terminal Servers • Web Resources • Programming • Applications • Mathematics • Text Editors • Graphics & CAD • and more.*

Technotes are found at: <http://www.engin.umich.edu/caen/technotes>