Math 4610 Fundamentals of Computational Mathematics - Topic 5.

This topic covers information on how to work once a terminal is up and running on your computer. A Linux/Unix operating system should be running in the terminal. In most real computational settings, it is important to be able to work using a command line to create files, modify these files, compile code, and a number of other tasks. You can invoke multiple terminals and work on multiple files at the same time. Your instructor has three or four terminals open at any given time. One to edit files, one to compile and execute code, and the third to display results. The homework and projects assigned in the class will typically require the use of a terminal to complete the work.

We will eventually use High Performance Computing (HPC) resources at the Center for High Performance Computing (CHPC) at the University of Utah to work on a project or two. The access these resources, we will need to do this through terminals or terminal emulators running Linux/Unix operating systems. So, it is important to be familiar with at least a few basic Linux/Unix commands. In this section of the notes we will work through a few of the more important commands needed. We will also introduce more commands with options as we work through the semester.

In general, a Linux/Unix command can be written in the form:

```
koebbe% command [options] [input parameters]
```

where koebbe% is the prompt for entering a command and "command" is replaced with a command. For example, replacing "command" with the string "pwd" gives

```
koebbe% pwd
```

The options and parameters in the general form may or may not be needed depending on what you are trying to do. Note that the pwd command will display the (p)resent (w)orking (d)irectory. In most cases, the pwd command will not need any options or parameters. When we compile code, it is common to use a number of options on the compiler commands. For example, one of the assignments will require the development of a shared library which entails compiling code in multiple files and then archiving the compiled versions of the files into the library - more on this later.
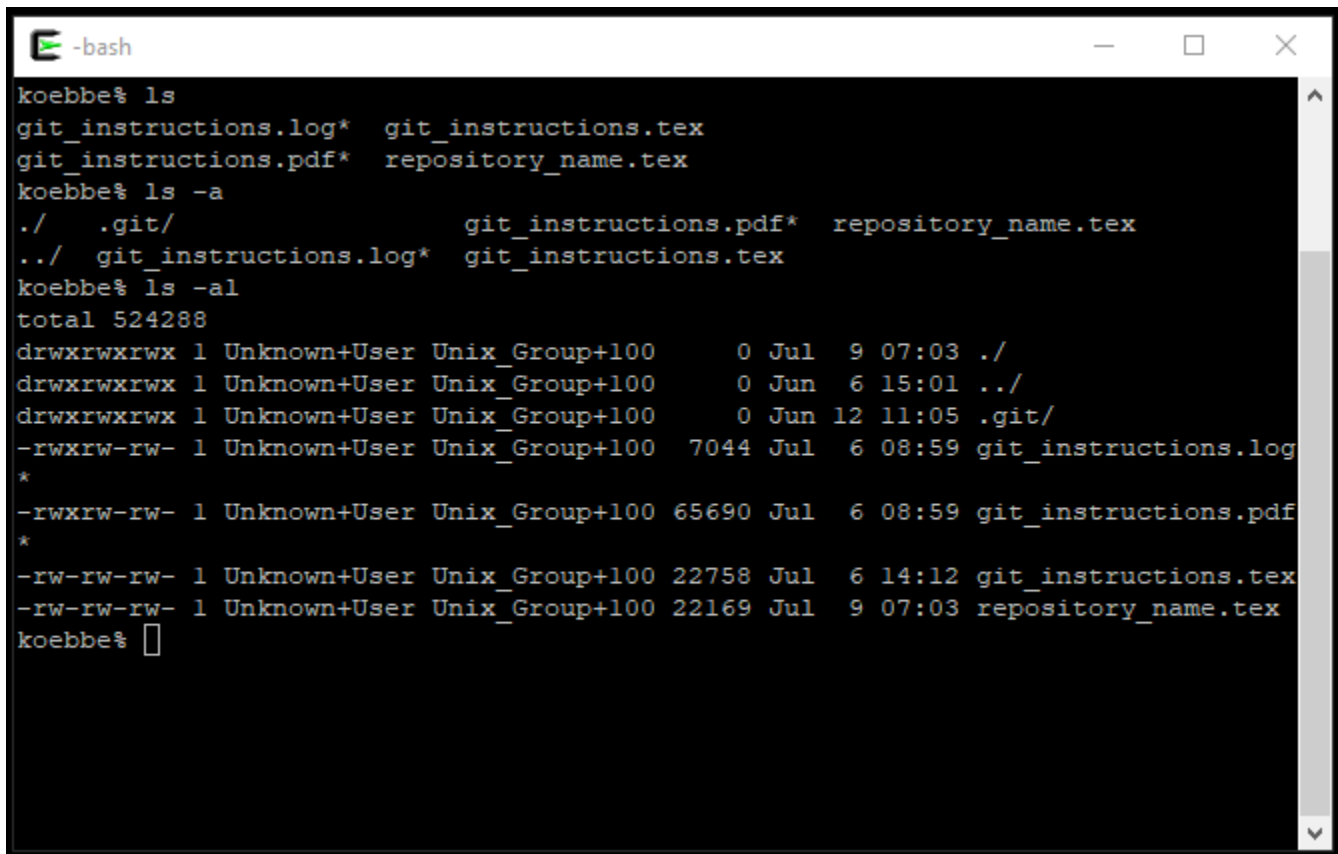
Each of the subsections below will cover a single type of command to help you get started. There are hundreds of commands that can be used from within a terminal. Some of these commands can be unforgiving - there is no trash folder that will save you from accidently removing a file. Note that there are some safety measures that one can add. See the file remove command (rm) below.

Another important issue that any computational person should be aware of is that there are a number of shells that you can work in. A command shell, say bash, will have a slightly different flavor of commands and syntax. The shells that most computational scientists use are bash, csh, tcsh, and ksh. It is probably best to select a shell, like bash or tcsh, and stick with that throughout the semester. Finally, a good computational person is able to chain together a set of commands to perform a sequence of commands. This sometimes referred to as shell programming or shell scripting. This is an incredibly useful skill to have. However, shell programming is beyond the scope of this course - meaning we can get by without it.

List the Contents of the Home Folder

Once a terminal is running, the first thing we probably want to do is to look at what is in a the directory we are in. It is also important to know where you are in a directory tree and the like. This also serves as a first linux or unix command. In the screenshot below there are a couple of versions of a command that will list files and folders with more or less information. Note that most linux commands look like the general command form given above. The example in the figure shows to versions of the "ls" command. The first is a simple version that lists files in the folder. The second example adds the " -a" flag. This results in a list of all files including hidden files in the folder. The output shows a couple more files. The last example uses the flag(s) " -al" which gives a

long listing for the files in the folder. There are lots of variations that can be used to obtain information about the files in a folder.
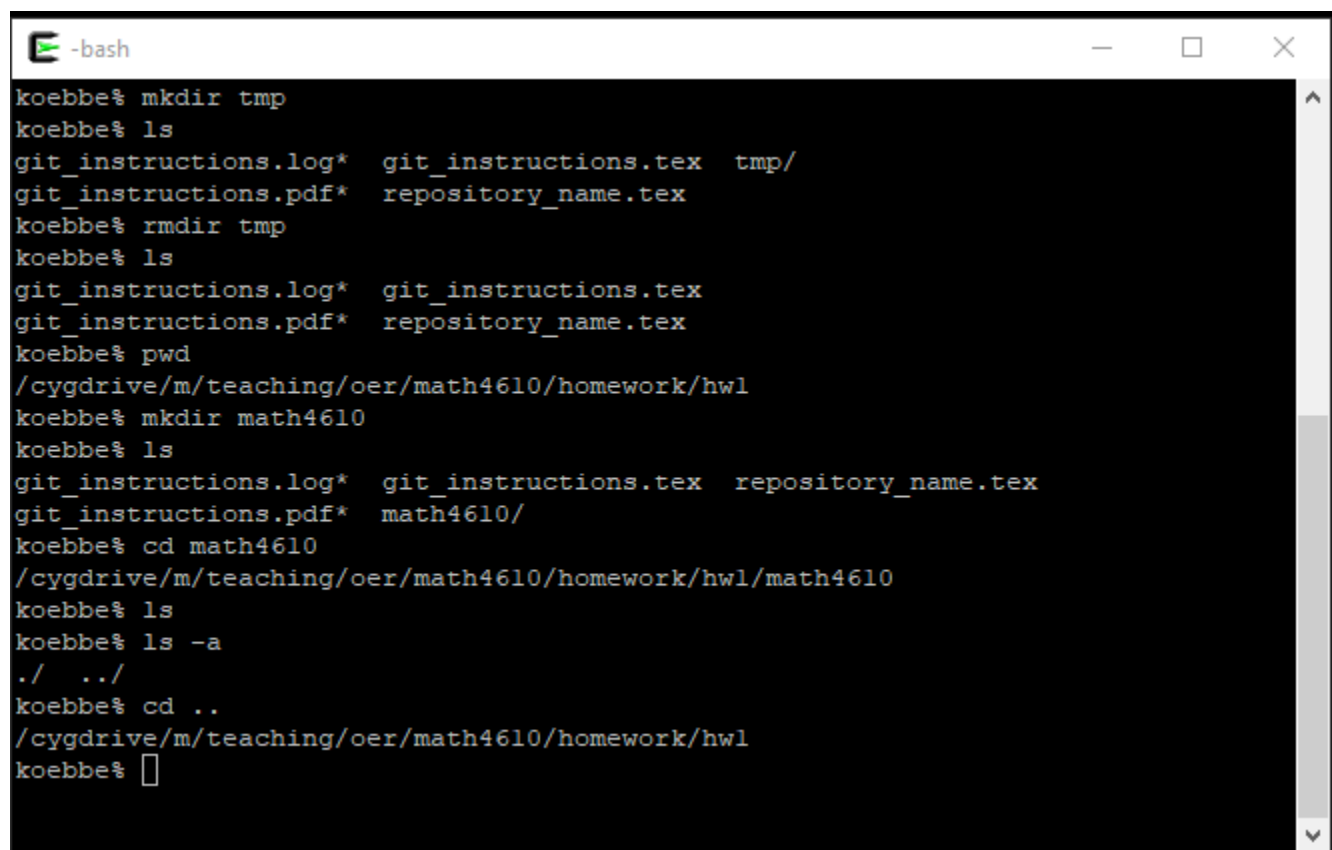

Figure 1: Three versions of the ls command used to list the contents of the curtent directory. Screenshot taken using **Snip & Sketch**. This is an app on my Windows 10 box

Directory/Folder Commands

You will need to create files and folders, move files and folders, remove files, and perform other operations to directories to keep work organized. The **mkdir** command allows a persion to create a new directory in the current working directory. This is the same thing tha Windows Explorer allows you to do with a popup menu (New Folder). There will be many places where a directory structure will be required. You can remove a directory with the **rmdir** command. The **cd** command can be used to navigate through a directory structure. Finally, on this screen capture, the **pwd** command is used to determine the current working directory. This can be used to figure out where you are in a directory structure.

```
koebbe% pwd               current working directory
koebbe% cd                change working directory
koebbe% mkdir             make a new directory
koebbe% rmdir             remove an existing directory
```


Figure 2: Commands for making and removing folders, changing the working directory and checking the location of the current folder files and folders. Screenshot taken using **Snip & Sketch**. This is an app on my Windows 10 box

There are a few short cut commands for moving around your directory structure in Linux. Here are a few that may save you some time.

```
koebbe% cd ~                 change working directory to the home directory
koebbe% cd ..                change working directory to the parent directory
koebbe% cd ../subfolder      change working directory up one folder and then
                             to subfolder
koebbe% cd subfolder         change working directory to a subfolder of the
                             current folder
koebbe% cd ./subfolder       same as above
koebbe% mkdirs folder        make the specfied folder and all subfolders
koebbe% rmdir -rf folder     remove an existing directory with prejudice
```

The which and man Commands

You will probably want to know what is available for doing work within Cygwin or any other terminal emulator. The which command allows you determine if apps or other executables are available on your version of Cygwin. In particular, it is important to know if certain compilers (e.g, javac, gcc, f77) are available. A significant number of tasks you will be asked to complete will require the use of a compiler and Cygwin has a number of (good) standard compilers for C, C++, and fortran. The syntax for the command is the following.
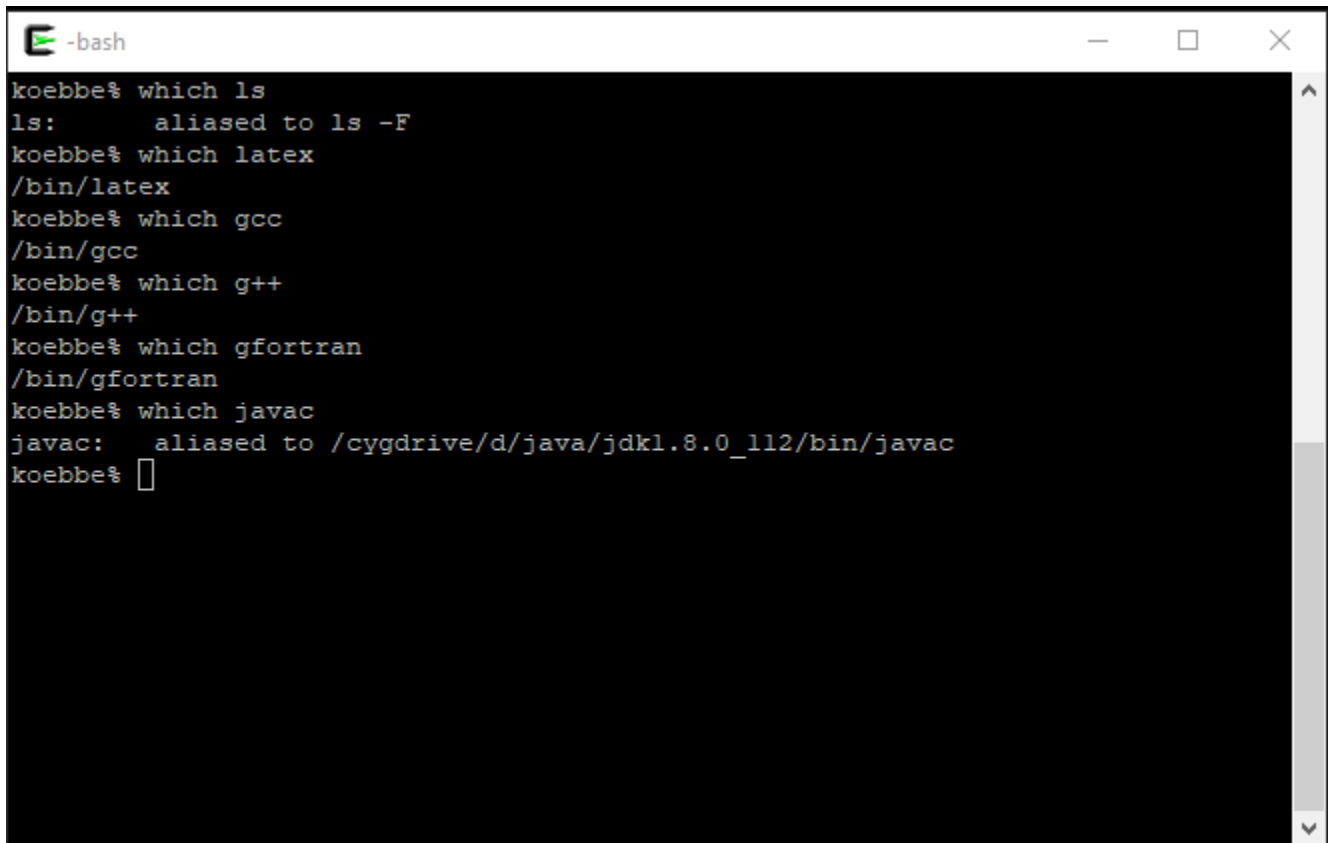
```
koebbe% which command-name
```

If the command results in command not found, this means that you typed the name incorrectly (a constant source of frustration for your instructor) or the app or command has not been installed. If in Cygwin, you can use the setup script to find and install what is needed.

In the example shown below, the which command is used to test for 4 different compilers. In each case, the location of the file associated with the command is given. Once you know a command is available, you can use the man command to get information about command usage. For example, typing

```
koebbe% man gcc
```

will display information on the gcc compiler along with explanations and documentation on how to specify flags and options for the compiler.

```
koebbe% which ls
ls:        aliased to ls -F
koebbe% which latex
/bin/latex
koebbe% which gcc
/bin/gcc
koebbe% which g++
/bin/g++
koebbe% which gfortran
/bin/gfortran
koebbe% which javac
javac:   aliased to /cygdrive/d/java/jdk1.8.0_112/bin/javac
koebbe% []
```
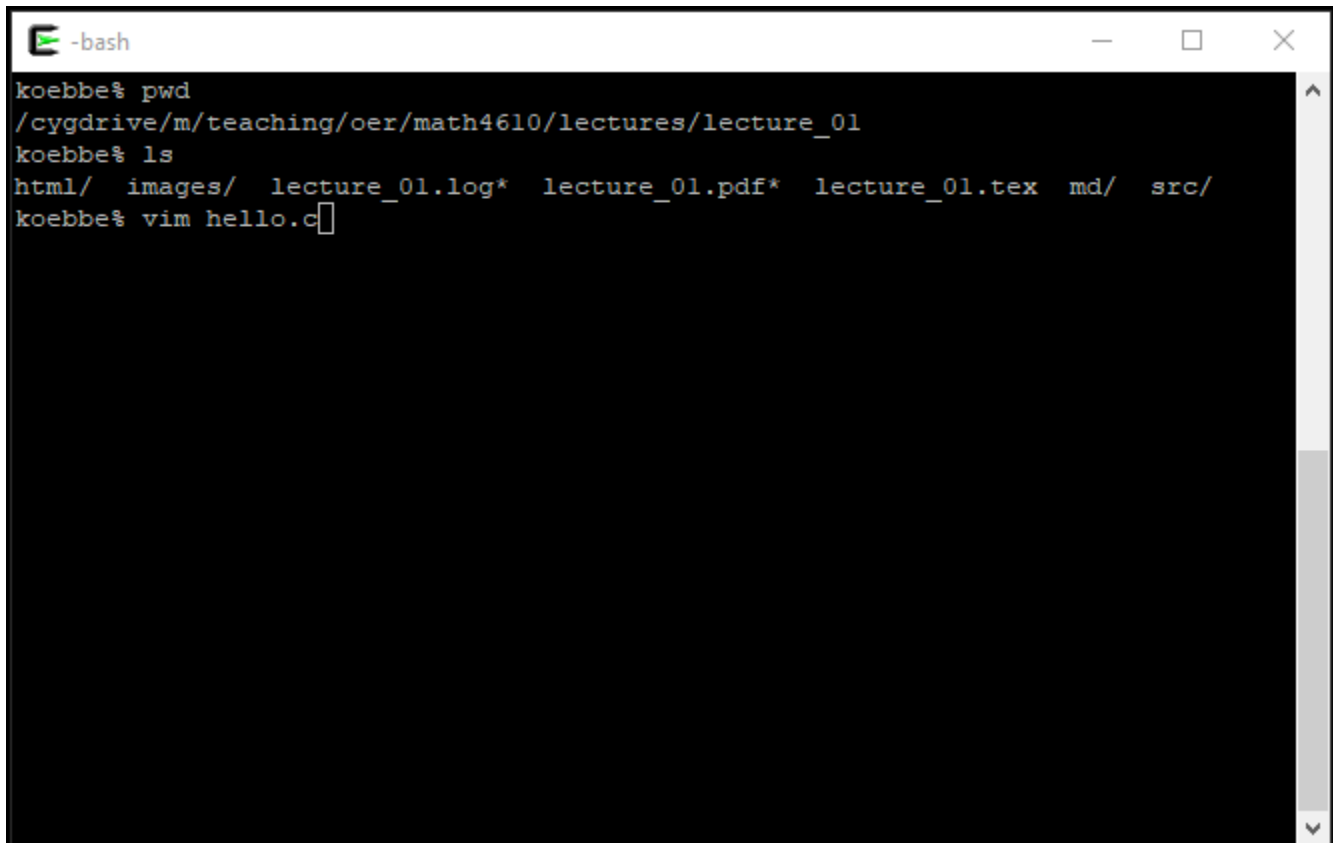
Figure 3: Using the which command to determine if compilers are installed. Screenshot taken using **Snip & Sketch**. This is an app on my Windows 10 box

A Simple Editing Program

You will need an editor to create text files. There are a number of editors that can be downloaded and used in any Cygwin installation. The standard editor that is always available for linux and unix boxes is 'vi'This editor is a bit rudimentary, but works. Another editor which will be used in by the instructor in the course is 'vim', a modified and improved version of vi. The syntax for starting the editor in a window is the following.

```
    koebbe% vim filename
```

There are a lot of escape sequences vim uses to insert text, write a file and so on. If you are new to vim, you will need to learn at least a few of these editing commands.

```
koebbe% pwd
/cygdrive/m/teaching/oer/math4610/lectures/lecture_01
koebbe% ls
html/  images/  lecture_01.log*  lecture_01.pdf*  lecture_01.tex  md/  src/
koebbe% vim hello.c
```

Figure 4: How to start the vim editor. Screenshot taken using vim. The name of the file being edited is hello.c. **Snip & Sketch**. This is an app on my Windows 10 box

---

First View of the vim Editor

The terminal will turn what is shown below into when you start up the editor on a new file. To get out of the editor, you can use any of the following commands inside the editor.

```
    :x                  write and exit the editor - saves changes in the file
    :q                  exit the editor if no changes have been made
    :x!                 force a write and exit the editor - saves the file
    :q!                 force an exit of the editor - no changes are saved
```

You need to type in the colon, :, to indicate you want to enter a command. Note that there are a few other commands that can be used to save changes. For example

```
    :w                  write and stay in the editor
    :w!                 force a write and stay in the editor
```
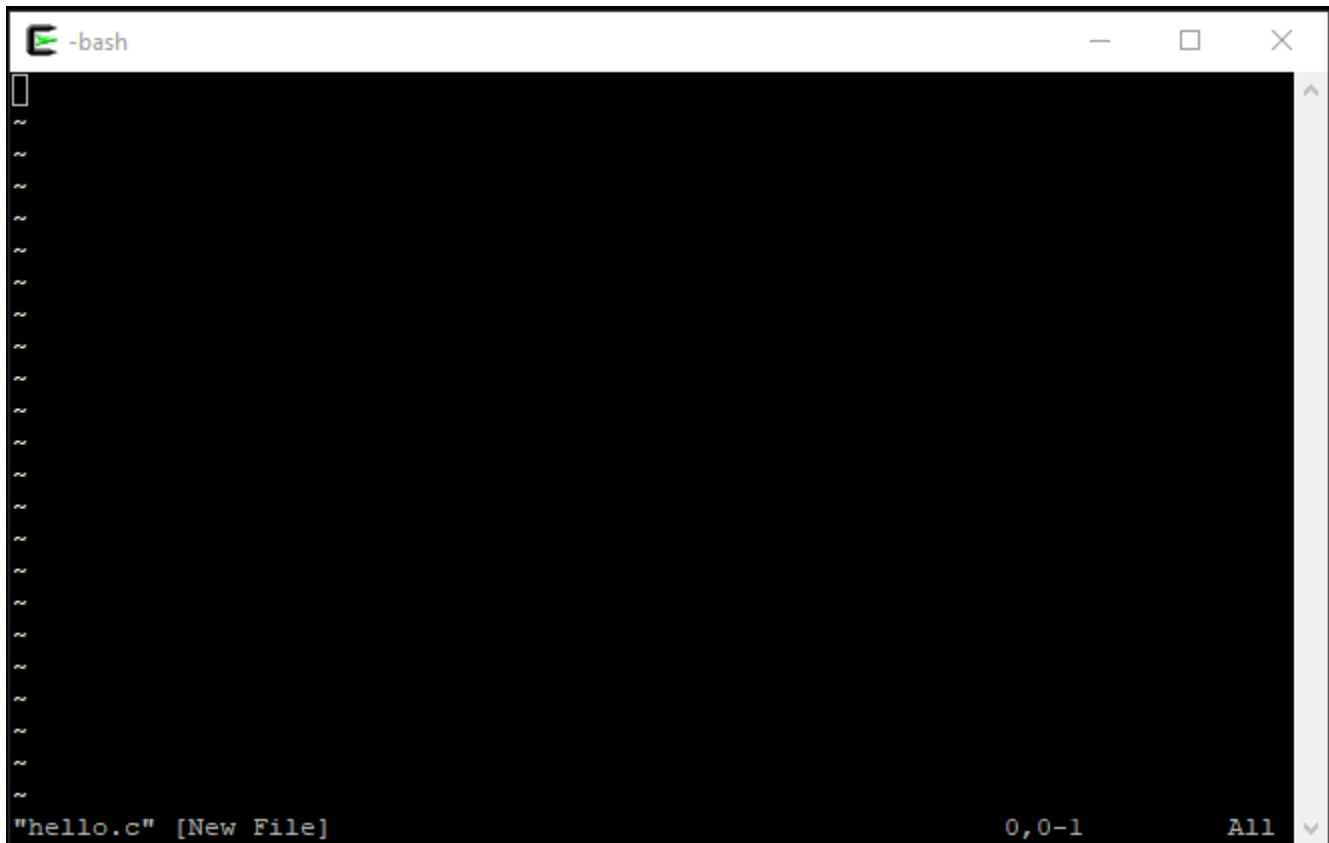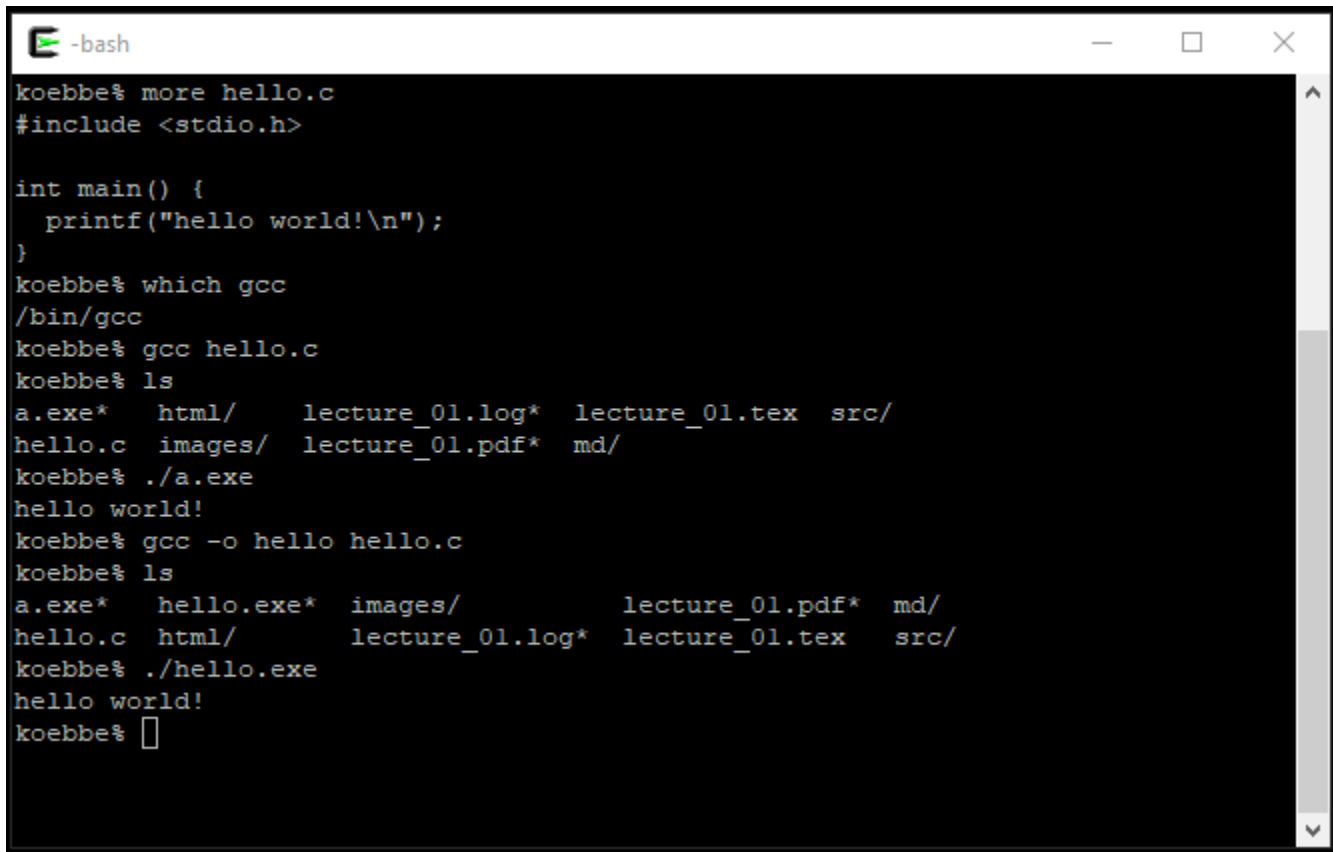

Figure 5: Opening up a new file in vim. Screenshot taken using **Snip & Sketch**. This is an app on my Windows 10 box

---

An Example of a Text File/Program

The following screenshot shows a few lines that have been typed into vim that deinfes a standard hello world example for C. To insert/append characters in the text file, you can use the following commands to do this. Note that the commands below do not show up on the screen and the chnages are made where the cursor is currently located.

```
a                    append text at this point in the file
o                    open a line after the current line
O                    open a line before the current line
```

To end adding or inserting text, use a single escape character. Again, the commands will not show up on screen. Learning everything about vi or vim is a time consuming process. It is one of those things that you figure out as you go. The advantage of vi is that it exists on every flavor of Linux/Unix.

Figure 6: Adding lines to a text file. The text gives an example of the "hello world" program from a C course. Screenshot taken using **Snip & Sketch**. This is an app on my Windows 10 box

---

Compiling a Program

Compiling a program is relatively easy at this point in time. To compile the program shown in the previous section you can type:

```
koebbe% gcc hello.c
```

The result is an exeutable as seen below. If you want to name the executable something besides 'a' then type the following.

```
koebbe% gcc -o hello hello.c
```

```
E- -bash                                                              —   □   ×
koebbe% more hello.c
#include <stdio.h>

int main() {
  printf("hello world!\n");
}
koebbe% which gcc
/bin/gcc
koebbe% gcc hello.c
koebbe% ls
a.exe*   html/     lecture_01.log*  lecture_01.tex  src/
hello.c  images/   lecture_01.pdf*  md/
koebbe% ./a.exe
hello world!
koebbe% gcc -o hello hello.c
koebbe% ls
a.exe*   hello.exe*  images/          lecture_01.pdf*  md/
hello.c  html/       lecture_01.log*  lecture_01.tex   src/
koebbe% ./hello.exe
hello world!
koebbe% []
```
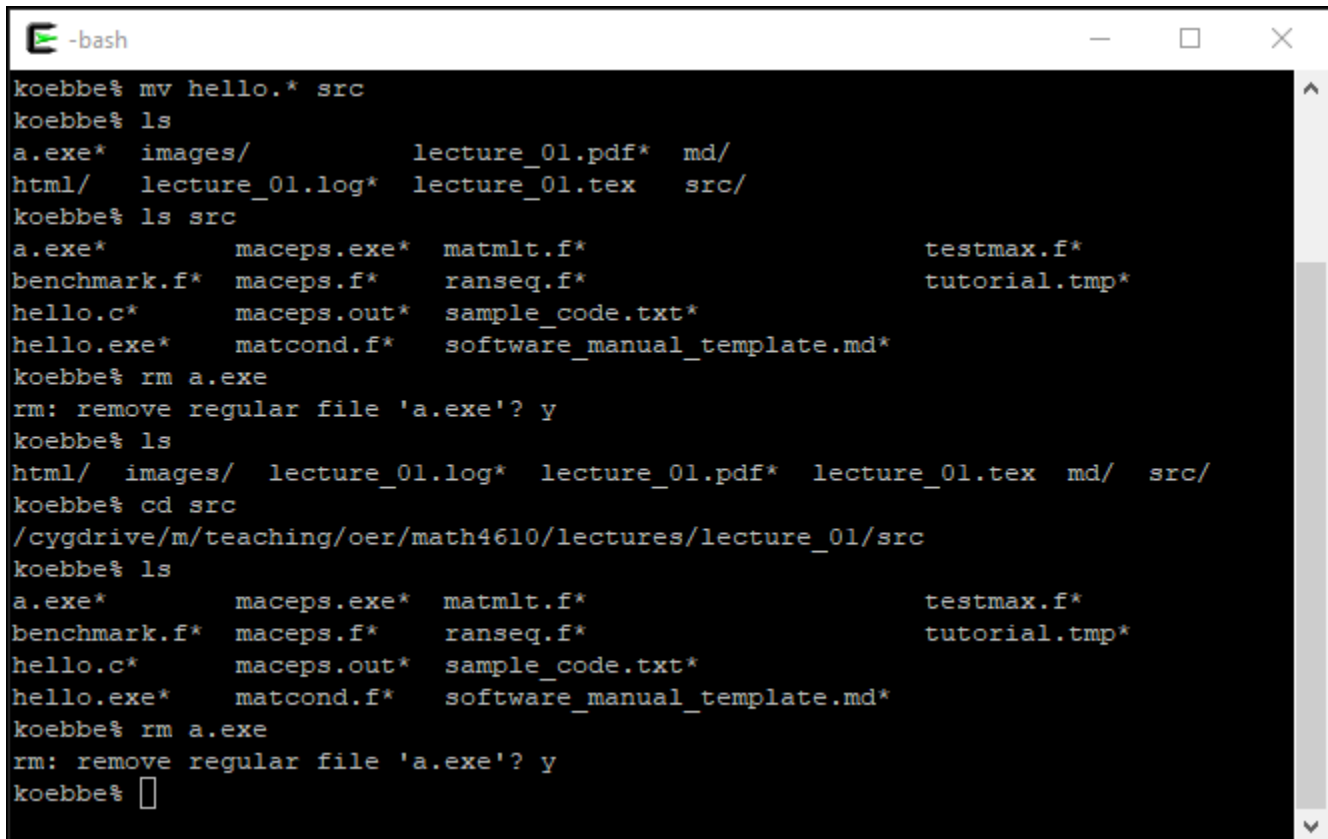
Figure 7: Compiling the hello.c program. Screenshot taken using **Snip & Sketch**. This is an app on my Windows 10 box

Keeping Track of Working Code

It is a good idea to organize your work within assignments and projects. There is a standard set of folders/directories in linux and unix that most have adopted. Your instructor follows this idea and usually creates a list of folders including /src, /data, /bin, and /doc. When computer literate folks see these folders, they know what is stored in the folders. As an example, type the commands

```
koebbe% mkdir src
koebbe% mkdir bin
```

to create folders for the executable the text file: src for the text files and bin for the binary/executable files.

```
koebbe% mv hello.* src
koebbe% ls
a.exe*   images/           lecture_01.pdf*   md/
html/    lecture_01.log*   lecture_01.tex    src/
koebbe% ls src
a.exe*        maceps.exe*   matmlt.f*                    testmax.f*
benchmark.f*  maceps.f*     ranseq.f*                    tutorial.tmp*
hello.c*      maceps.out*   sample_code.txt*
hello.exe*    matcond.f*    software_manual_template.md*
koebbe% rm a.exe
rm: remove regular file 'a.exe'? y
koebbe% ls
html/  images/  lecture_01.log*  lecture_01.pdf*  lecture_01.tex  md/  src/
koebbe% cd src
/cygdrive/m/teaching/oer/math4610/lectures/lecture_01/src
koebbe% ls
a.exe*        maceps.exe*   matmlt.f*                    testmax.f*
benchmark.f*  maceps.f*     ranseq.f*                    tutorial.tmp*
hello.c*      maceps.out*   sample_code.txt*
hello.exe*    matcond.f*    software_manual_template.md*
koebbe% rm a.exe
rm: remove regular file 'a.exe'? y
koebbe% []
```

Figure 8: Some commands to reorganize files. Screenshot taken using **Snip & Sketch**. This is an app on my Windows 10 box