Lab 01: Unix Familiarization and Command Line Tools

Preliminaries

Lab Learning Goals

The goal of this lab are:

1. To familiarize yourself with the linux enviorment
2. To learn the manual pages
3. To learn about file permissions
4. To learn file parsing command line tools
5. To learn to create text files

Lab Setup:

- Run the following command in your terminal

```
~aviv/bin/ic221-up
```

- Then change into the following directory

```
cd ic221/labs/01
```

- You will find all the material you need to complete this lab in that directory.
- **During the course of this lab, we will refer to the `ic221/labs/01` as the lab directory**

Lab Worksheet

Where indicated through lab questions and directed work, you may be asked to answer questions on a worksheet or create files. You should do so in the lab directory.

Additionally, we provide directions for editing files using emacs, and directions can be found at the end of the lab (**emacs**) and on the course resources pages (**resources**). Through this course, we encourage you to learn the emacs editor. You may, however, use any editor you feel comfortable with, and the only requirement is that you become "good" at the editor. Learn the shortcuts! It will greatly improve your experience as a computer programmer.

Lab Submission

To submit this lab you will place all relevant content into your lab directory:

```
ic221/labs/01
```

Then issue the submission script

```
~aviv/bin/ic221-submit
```

Select your section number and the option for `labs/01`, and confirm. Your submission was successful if you see SUCCESS at the end. **You may submit multiple times** up until the submission deadline. Only your final submission will be considered for grading.

Part 1: The Man Pages

One of Unix'es greatest features is that it is self-documenting through a set of manuals. To access the manuals, you use the `man` command. Let's start by looking at the manual page for `ls`:

```
aviv@mich342csdtestu:~$man ls
```

This brings up the manual page, whose header looks like this:

```
LS(1)                                          User Commands

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List information about the FILEs (the current directory by default).  Sort entries alphabet
       of -cftuvSUX nor --sort is specified.

       Mandatory arguments to long options are mandatory for short options too.

       -a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..
```

The manual page provides both a brief description of the command and the arguments and options. For example, we see that the option `-a` or `--all` both list all entries for the directorie, including `.` and `..` and all hidden files starting with `.` while, conversely, the `-A` or `--almost-all` option list all hidden files while not displaying the `.` or `..`

entries.

This is just one of many options for the `ls` command, and you can scroll down using the up and down arrow key. You can quit the manual page using `q`, and if you need to panic `C^g` until you can hit `q` and exit.

Lab Questions and Tasks

Perform the following taks and answer the following questions in the `worksheet.txt` file found in your lab directory.

1. For the `ls` command, what option prints information out in long form, like `-l`, but does not print any file ownership information? In the worksheet, provide a copy of the output using `ls` with this option run from the **top level** of the lab directory.
2. Change into the `part1` directory and type `ls`. You will see a list of files `a b c d e f g`.
    a. Note that `ls` lists the files in alphabetic order. What `ls` option will list the files in *reverse* alphabetic order? Provide a copy of your output of your `ls` with the addition of `-l` in your worksheet.
    b. What `ls` options will sort the files by *size* from smallest to largest? Provide a copy of your output of your `ls` with the addition of `-l` in your worksheet.
    c. What `ls` option will sort the files in *reverse size* order from largest to smallest. Provide a copy of your output of your `ls` with the addition of `-l` in your worksheet.
3. Remove the `g` file using the `rm` command. Notice that the shell asked you to confirm removing the item. Look at the manual for `rm`, what option must have been invoked when you issued that command? What option can you use to avoid having to confirm the removal of an item?
4. (**Challenge**) Read the manual page for the `touch` command. One of the uses for `touch` is to update the last modified timestamp of a file (you can view that last modified using `ls -l`). Use the `touch` command to create a file `y2k` whose last modification time was Dec. 31 1999 at 23:59.59. Include the command you used on your worksheet and a copy of your `ls -l` output of the `y2k` file.

Part 2: `cat`, `less` and `more`

Another really import part of using the command line tools is to be able to read/view the contents of files. There are a number of ways to do this without the command line, for example, you could just open the file in an editor like `emacs` or `gedit`, but as Unix users, we know there must be a better way if we are only viewing the contents of the file.

Viewing files with `cat`

If you can think of the most basic functionality for viewing the file, this would include just printing the contents of the file directly to the terminal screen. The command to do just that is the `cat` command, which is short for *concatenate*. Here is the `man` page synopsis for `cat`:

```
NAME
     cat -- concatenate and print files

SYNOPSIS
     cat [-benstuv] [file ...]

DESCRIPTION
     The cat utility reads files sequentially, writing them to the standard output.  The file oper
     are processed in command-line order.  If file is a single dash (`-') or absent, cat reads fro
     standard input.  If file is a UNIX domain socket, cat connects to it and then reads it until
     This complements the UNIX domain binding capability available in inetd(8).
```

More simply, the `cat` command takes a file or sequence of files, and write them to standard out, which is the terminal. Let's do a quick example. Navigate to `part2` in the lab directory, and let's `cat` the output of the `GoNavy.txt` file:

```
#> cat GoNavy.txt
Beat Army!
```

The output of the `cat` command, printed to the terminal standard out, is "Beat Army!". You should verify contents of the file by opening it an editor.

`cat` can also take multiple files as input, and print there contents to the terminal one after the other, or, another way to put it, `cat` will concatenate the contents of two files by printing it to standard output.

Let's use the `cat` command to view the contents of the files in `part2` of the lab directory. Use `cd` to navigate to there now.

```
#> cat BeatArmy.txt GoNavy.txt
Go Navy!
Beat Army!
```

As you can see the contents of `BeatArmy.txt` is "Go Navy!" and the contents of `GoNavy.tx` is "Beat Army!. The concatenation of those contents is "Go Navy! Beat Army!" across two lines.

Viewing files with `less` and `more`

One draw back of viewing files with `cat` is that it clutters up your terminal, and any reasonable Unix user hates a cluttered terminal. You could always clear your terminal using `clear` or `C^l` (Control-l) — go ahead and try that now — but it can get bothersome the more you need to do so.

Instead, Unix provides two ways to view a file within a terminal application: `less` and `more`. In the jocular style of Unix design, `less` is `more` and `more` is `less`. The basic difference between the two file viewers is that `less` allows you to go forward and backwards in a file while `more` only allows you to move forward in the file, exiting at the end. Thus, in Unix, `less` is really more than `more`.

Let's see an example of why this is useful. Consider two great authors of literature, Charles Dickens and Ernest Hemingway. Dickens was paid by the word and so his stories are very long indeed, while Hemingway was a minimalists, and his stories were quite short. In the `part2` directory you have two text files, one named `dickens.txt` and one named `hemingway.txt`.

We can easily read `hemingway.txt` using `more`, it just moves forward in the file by pressing :space: to page down or the arrow key :down:. The indicator at the bottom of the `more` screen

```
--More--(95%)
```

Describes how far in the file we've progressed. Lets now use `more` to read `dickens.txt` … oh man! This is going to take forever, and there is only one way to go, forward. Clearly, we need a *more* powerful viewer, so we use `less`. The less terminal allows you to move forward and back, plust a bunch of other useful navigation tools. Here are some below

`less` Navigation

- To quit: `q`
- To search forward: `/` then type your search (regex allowed) then use the following
    - Find Next match going forwards: `n`
    - Find Prev match going backwards: `N`
- To search backward: `?` then type your search (regex allowed) then use the following
    - Find Prev going backwards: `n`
    - Find Next going forwards: `N`
- Go To line: `:` the type line number
- Start of File: `<`
- End of Fole: `>`
- Panic: `C-g` mash this if you are in a state you don't understand

Lab Questions and Tasks:

1. Use `cat` to place a "Beat Army!" at the start of Hemmingway's a very short story and "Go Navy!" at the end. Include the command you used on your worksheet.
2. Why is `less` `more`?
3. Use `less` to open `dickens.txt`:
    a. Search for the first instance of "Fagin", what is the line of that text?
    b. Find the **second to last** instance of "Fagin". Describe how you did that and the sentence it appears in.
    c. Go to line 1845, what is the name of that chapter?

Part 3: Viewing Files Conditionally with head, tail, sed and grep

When we do want to print the contents of a file to the terminal, we may not want to print the whole thing, as cat does. Instead, sometimes we'd like just to print the first *n* lines, or the last *n* lines, or some set of lines in the middle, or just print lines that match a given search string. For that we have set of very useful commands.

For the following examples, navigate to the part4 directory in the lab directory. There is a sample file sample-db.csv that you will use for this part that contains **fake** records of people entering information on a web server.

View the first or last *n* lines with head or tail

The head command line tools is used to print the head of the file. By default, head prints the first 10 lines:

```
#> head sample-db.csv
#first_name,last_name,company_name,address,city,county,state,zip,phone1,phone2,email,web
James,Butt,Benton, John B Jr,6649 N Blue Gum St,New Orleans,Orleans,LA,70116,504-621-8927,504-845-
Josephine,Darakjy,Chanay, Jeffrey A Esq,4 B Blue Ridge Blvd,Brighton,Livingston,MI,48116,810-292-9
Art,Venere,Chemel, James L Cpa,8 W Cerritos Ave #54,Bridgeport,Gloucester,NJ,08014,856-636-8749,85
Lenna,Paprocki,Feltz Printing Service,639 Main St,Anchorage,Anchorage,AK,99501,907-385-4412,907-92
Donette,Foller,Printing Dimensions,34 Center St,Hamilton,Butler,OH,45011,513-570-1893,513-549-4561
Simona,Morasca,Chapman, Ross E Esq,3 Mcauley Dr,Ashland,Ashland,OH,44805,419-503-2484,419-800-6759
Mitsue,Tollner,Morlong Associates,7 Eads St,Chicago,Cook,IL,60632,773-573-6914,773-924-8565,mitsue
Leota,Dilliard,Commercial Press,7 W Jackson Blvd,San Jose,Santa Clara,CA,95111,408-752-3500,408-81
Sage,Wieser,Truhlar And Truhlar Attys,5 Boston Ave #88,Sioux Falls,Minnehaha,SD,57105,605-414-2147
```

Similarly, tail by default will show the last 10 lines:

```
#> tail sample-db.csv
Carlee,Boulter,Tippett, Troy M Ii,8284 Hart St,Abilene,Dickinson,KS,67410,785-347-1805,785-253-704
Thaddeus,Ankeny,Atc Contracting,5 Washington St #1,Roseville,Placer,CA,95678,916-920-3571,916-459-
Jovita,Oles,Pagano, Philip G Esq,8 S Haven St,Daytona Beach,Volusia,FL,32114,386-248-4118,386-208-
Alesia,Hixenbaugh,Kwikprint,9 Front St,Washington,District of Columbia,DC,20001,202-646-7516,202-2
Lai,Harabedian,Buergi & Madden Scale,1933 Packer Ave #2,Novato,Marin,CA,94945,415-423-3294,415-926
Brittni,Gillaspie,Inner Label,67 Rv Cent,Boise,Ada,ID,83709,208-709-1235,208-206-9848,bgillaspie@g
Raylene,Kampa,Hermar Inc,2 Sw Nyberg Rd,Elkhart,Elkhart,IN,46514,574-499-1454,574-330-1884,rkampa@
Flo,Bookamer,Simonton Howe & Schneider Pc,89992 E 15th St,Alliance,Box Butte,NE,69301,308-726-2182
Jani,Biddy,Warehouse Office & Paper Prod,61556 W 20th Ave,Seattle,King,WA,98104,206-711-6498,206-3
Chauncey,Motley,Affiliated With Travelodge,63 E Aurora Dr,Orlando,Orange,FL,32804,407-413-4842,407
```

You can describe how many lines you wish to show in two ways, either by using -n argument, where n is replaced by the number of lines. For example, to print the first 3 lines:

```
#> head -3 sample-db.csv
#first_name,last_name,company_name,address,city,county,state,zip,phone1,phone2,email,web
James,Butt,Benton, John B Jr,6649 N Blue Gum St,New Orleans,Orleans,LA,70116,504-621-8927,504-845-
Josephine,Darakjy,Chanay, Jeffrey A Esq,4 B Blue Ridge Blvd,Brighton,Livingston,MI,48116,810-292-9
```

Or, by passing the number of lines, following -n like -n 3:

```
>head -n 3 sample-db.csv
```

```
#first_name,last_name,company_name,address,city,county,state,zip,phone1,phone2,email,web
James,Butt,Benton, John B Jr,6649 N Blue Gum St,New Orleans,Orleans,LA,70116,504-621-8927,504-845-
Josephine,Darakjy,Chanay, Jeffrey A Esq,4 B Blue Ridge Blvd,Brighton,Livingston,MI,48116,810-292-9
```

Printing intermediate lines with sed

The sed command is very powerful, and it has many more features than just printing intermediary lines. We will explore some of those features later in the course and today just focus on intermediary line printing.

Here is the format of the sed command:

```
Line Number Input
      |          ,--File to process
      v          v
 sed -n 3,10p filename
        ^  ^^
Start---'  ||
Finish-----''---Print those lines
```

As example, what if we want to print the first 3 lines of the database file without including the index, the line starting with #. Then, we'd like to print lines 2 through 4.

```
#> sed -n 2,4p sample-db.csv
James,Butt,Benton, John B Jr,6649 N Blue Gum St,New Orleans,Orleans,LA,70116,504-621-8927,504-845-
Josephine,Darakjy,Chanay, Jeffrey A Esq,4 B Blue Ridge Blvd,Brighton,Livingston,MI,48116,810-292-9
Art,Venere,Chemel, James L Cpa,8 W Cerritos Ave #54,Bridgeport,Gloucester,NJ,08014,856-636-8749,85
```

Printing only matching lines with grep

Finally, we need a mechanism to only process lines that match a condition. The grep command is used for that, and it is such an important command in the Unix ecosystem, it is used as a verb. For example, "We grep out lines matching the string" is something you'll hear your instructors say throughout this class, and "grep" is loosely defined as "match."

For example, let's consider trying to just print the lines where the person is from the state of New Jersey. To do that, we need to first identify a unique part of lines for people from New Jeresey, and that is "NJ" in the address field.

```
#> grep NJ sample-db.csv
Art,Venere,Chemel, James L Cpa,8 W Cerritos Ave #54,Bridgeport,Gloucester,NJ,08014,856-636-8749,85
Alisha,Slusarski,Wtlz Power 107 Fm,3273 State St,Middlesex,Middlesex,NJ,08846,732-658-3154,732-635
Ernie,Stenseth,Knwz Newsradio,45 E Liberty St,Ridgefield Park,Bergen,NJ,07660,201-709-6245,201-387
(...)
```

Note that in a grep command, the first part is the search term and the second part is the file to be searched. grep is a very powerful command that uses a special search langauge called regular expressions, and you can search for all sorts of things. This is not the focus of this course, but if you would like to learn more, speak with your instructor.

Lab Questions

1. Read the man pages for head and less, produce a command line to print the first kilobyte of the database file. Note, a kilobyte is 2^{10} or 1024 bytes. Include the command line in your worksheet.
2. Use less or grep to find the line number of "Klonowski". Produce a sed command to just print the line with "Klonowski" and the following 5 lines. Include the command line in your worksheet.
3. How many people's first name is "Pamella"? Use grep to find that out.
4. Read the man page for grep. Print out the all the lines from people who **do not** have an address in NJ. Include the command line in your worksheet.

Part 4: Pipelines and counting with cut, sort, uniq, and wc

The final piece of the puzzle for file processing is to take the output of processing a file and set it as the input to another process. These process parts can be chained together into a *pipeline*. Consider this simmple pipeline below:

```
#> cat sample-db.csv | head -20
```

The pipe or | takes the output of one command and sets it as the input of another. In the example above, the output of the cat is to print the whole contents of the file to the input of head, which then only prints the first 20 lines of its input. While this is a contrived example, you should start to see the power of the pipline. Consider the below command:

```
#> grep NJ sample-db.csv | wc -l
```

The first part of the command will print out only the lines that contain the pattern "NJ", this output is then set as the input to the wc command, which is a command line tools to count words, lines, and bytes. The -l option says to just print the line count, and thus, the command above prints out the number of lines.

Nearly all Unix command line tools have an option to either read from a file or from standard input along a pipline. For example, the above command can be rewritten with cat at the front, as follows:

```
#> cat sample-db.csv | grep NJ | wc -l
```

For more options of wc, refer to the man page.

Parsing just fields with cut

A very useful pipeline tool is cut, which is used to extract fields from a formatted file, like our database file. Here is a basic command line argument:

```
          ,--- Deliminator
        __|_           ,-----Input File, or leave off to read from stdin
       /    \          v
#> cut -d "," -f 1 sample-db.csv | head -5
             \__/
               \__.-- Field
```

The *deliminator* determines how the file is to be cut. The `sample-db.csv` file is a comma separated file, so it is delimitated by commas; that is, every item in the line is separated by comma to distinguish it from other items on the line. The above command will print the first 5 lines of output from the first delimitated item:

```
#> cut -d "," -f 1 sample-db.csv | head -5
#first_name
James
Josephine
Art
Lenna
```

That is the `first_name` field. If we wished to not include the index, `first_name`, we could use a `head` and `tail` command combination:

```
#> cut -d "," -f 1 sample-db.csv | head -6 | tail -5
#first_name
James
Josephine
Art
Lenna
Donette
```

As you can see, this quickly builds into a very powerful tool just by adding commands to the pipeline.

Sorting and removing duplicates with `sort` and `uniq`

The last two parsing tools we'll use in this lab is `sort` and `uniq`, the former will sort input and the later will remove any **adjacent** duplicate lines. Consider how these might be used in conduction to solve different file parsing problems. **We leave their definitions and usage for you to determine by reading the `man` pages.**

Lab Questions and Tasks

You should continue to work in part3 to complete these lab questions:

1. Create a pipeline to count the number of unique states represented in the database file. Include the pipeline in your worksheet.
2. How many first names in the file repeat? How many last names? Include the pipelines used to determine this.
3. Write a pipeline to first print to the terminal all the unique telephone area codes? Add to your pipeline how to sort those numerical? (Hint: read the man page for `sort`).

Part 5: Permissions and Ownership chmod and chown

Your task in the last part of this lab is to better understand the file permissions. Recall the output from `ls -l` from the last lecture.

```
.- Directory?
|    .-------Permissions                  .- Directory Name
| ___|___      .----------- Owner          |
v/       \     V                          V
drwx--x--x 2 aviv 4096 2013-12-19 17:11 ic221/
-rw-r--r-- 1 aviv  412 2013-11-07 17:09 id_rsa.pub
                    ^  \_____________/     ^
File Size ----------'       |              '- File Name
  in bytes                  |
                            |
   Last Modified -----------'
```

The permission portion of the file indicates a lot about what can be done with that file and by whom. A permission is simply a sequence of 9 bits broken into 3 octets of 3 bits each. An *octet* is a base 8 number that goes from 0 to 7, and 3 bits unique define an octet since all the numbers between 0 and 7 can be represtend in 3 bits.

Within an octet, there are three permission flags, *read*, *write* and *execute*. These are often referred to by their short hand, `r`, `w`, and `x`. The setting of a permission to *on* means that the bit is 1. Thus for a set of possible permission states, we can unique define it by a octal number

```
rwx -> 1 1 1 -> 7
r-x -> 1 0 1 -> 5
--x -> 0 0 1 -> 1
rw- -> 1 1 0 -> 6
```

A full file permission consists of the octet set in order of *user*, *group*, and *global* permission.

```
 ,-Directory Bit
|
|       ,--- Global Permission
v      / \
-rwxr-xr-x
 \_/\_/
  |  `--Group Permission
  |
   `-- User Permissoin
```

These define the permission for the *user* of the file, what users in the same group of the file, and what everyone else can do. For a full permission, we can now define it as 3 octal numbers:

```
-rwxrwxrwx -> 111 111 111 -> 7 7 7
-rwxrw-rw- -> 111 110 110 -> 7 6 6
-rwxr-xr-x -> 111 101 101 -> 7 5 5
```

To change a file permission, you use the `chmod` command and indicate the new permission through the octal. For example, in `part5` directory, there is an executable file `hello_world`. Let's try and execute it. To do so, we insert a `./` in the front to tell the shell to execute the local file.

```
> ./hello_world
-bash: ./hello_world: Permission denied
```

The shell returns with a permission denied. That's because the execute bit is not set.

```
#> ls -l hello_world
-rw------- 1 aviv scs 7856 Dec 23 13:51 hello_world
```

Let's start by making the file just executable by the user, the permission 700. And now we can execute the file:

```
#> chmod 700 hello_world
#> ls -l hello_world
-rwx------ 1 aviv scs 7856 Dec 23 13:51 hello_world
#> ./hello_world
Hellow World!
```

This file can only be execute by the user, not by anyone else because the permission for the group and the world are still 0. To add group and world permission to execute, we use the permission setting 711:

```
#> chmod 711 hello_world
#> ls -l hello_world
-rwx--x--x 1 aviv scs 7856 Dec 23 13:51 hello_world
```

At times using octets can be cumbersome, for example, when you want to set all the execute or read bits but don't want to calculate the octet. In those cases you can use shorhands.

- r, w, x shorthands for permission bit read, write and execute
- The + indicates to add a permission, as in +x or +w
- The - indacetes to remove a permission, as in -x or -w
- u, g, a shorthands for permission bit user, group, and gloabl (or all)

Then we can change the permission

```
chmod +x file   <-- set all the execute bits
chmod a+r file  <-- set the file world readable
chmod -r  file  <-- unset all the read bits
chmod gu+w file <-- set the group and user write bits to true
```

Depending on the situations, both the octets and the shorthands are preferred.

Lab Questions and Tasks:

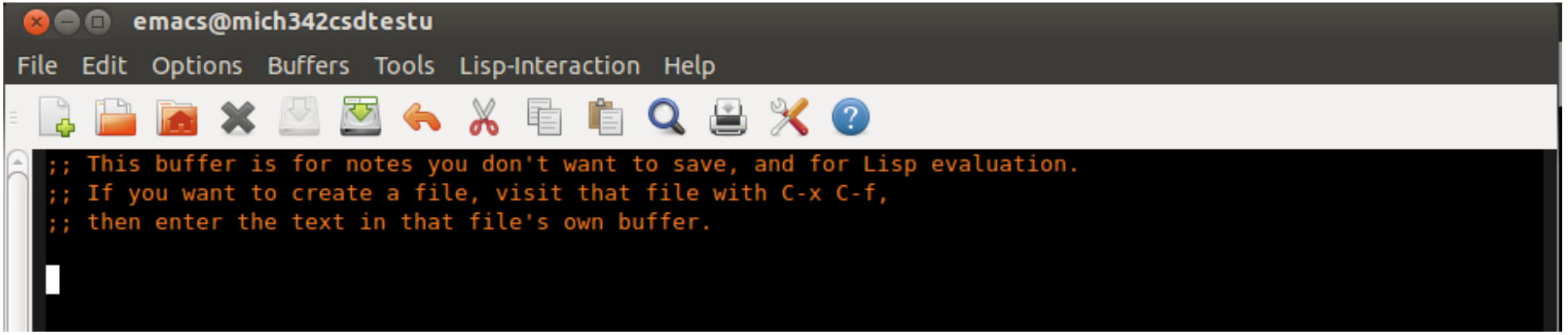
1. Convert the following permission states into an octet:
    a. User: read and write; Group: read; Global: read
    b. User: read and execute; Group execute; Global: read
2. Make the `runme` program executable and execute it. What is the output?
3. Work with a partner, copy the `runme` program to your *home* directory. Change the name of it to `runme-username` where `username` is replaced by *your* username (e.g., m16723).
    a. Set the program to the permission setting such that anyone can execute it. Ask your partner to try and execute it using the following command: `~username/runme-username` where `username` is replaced by your username. What is the output?
    b. Set the program to have the permission where only members of the same group can execute it. Run `ls -l` to verify you've done this properly and include that output in your worksheet. Can you partner run your program now? Why so or why not?
    c. Set the program such that only you can execute it, but anyone can read it. Run `ls -l` to verify that you've done this properly and include that output in your worksheet. Can your partner still run the program? But, could your partner *copy* the program to his/her home directory and then run it? Briefly describe that process in your worksheet.

Extra: Editing with Emacs

While you may use any editor to complete the lab worksheet, we'd like to suggest that you use emacs. It's a very powerful editor and a great tool to add to your tool box as a computer programmer. If you type `emacs` into the terminal like so with a file, it should open it up directly.

```
emacs worksheet.txt
```

The GUI window will look like below, and the button are self explanatory, if you want to use them.



While you can use all the buttons at the top, there are a number of really handy shortcuts. Throughout this class, we are going to recommend that you use emacs, but you may also any other editor of your choosing, such as `vim` or `gedit`, if that is what you feel most comfortable with.

`Emacs Keyboard Shortcut`

Terminology:

- Control Click: `C^` using the control key
- Meta Click: `M^` using the alt or esc key

Emacs Base Commands:

- PANIC: `C^g` or `esc esc esc`
- Save: `C^x C^s`
- Exit: `C^x C^c`
- Open: `C^x C^f`
- Alternate between open files: `C^x b`

Emacs Editing Commands:

- Highlight: `Cspc` then move cursor over highlight
- Copy: `M^w`
- Cut: `C^w`
- Past: `C^y`
- Cut Line: `C^k`