

Homework #1A (Unix and Perl)

Submit in the inbox outside Clark S260 before 5:00 PM.

This first problem set covers two of the tools which we will use this quarter: Unix and Perl. HW #1B will cover R and review basic bioinformatics. Note that some of the text below is linked to web pages, which will not be present if you print it; in any case it is probably best to view the pdf on your computer and click on the links rather than printing it and typing in the URLs.

1. *Reading and Literature*

The following chapters might be useful to look over if you just need a refresher in Unix, Perl, and/or R. The main purpose of the three coding books is as references. They're also really great for “interstitial” reading; whenever you have a spare 5-10 minutes or so when waiting for a meeting or standing in line for lunch, you can make that time into productive time by going over a bite-size chapter from the Perl Cookbook or Unix Power Tools. Over time this is how you build up an arsenal of tricks.

- (a) Unix Power Tools: Chapters 1, 2, 19.1-19.8 (see also below for full Emacs tutorial), 28 (skim as necessary)
- (b) Perl Cookbook: Chapters 1, 4-7 (read chapter intros, then skim as necessary)
- (c) Data Analysis and Graphics Using R: Chapters 1, 2, 14 (skim chapters as necessary)

As something that falls vaguely under the “reading” category, I also suggest that you get Firefox (if you don't already have it) and install the following plugins: Biobar, Colorful Tabs, Download Statusbar, DownThemAll, Fasterfox, FlashGot, Google Notebook, Greasemonkey, Linkification, RSiteSearch, Tab Mix Plus, Viamatix foXpose. Tab Mix Plus in particular is tremendously customizable; in conjunction with the Biobar and the literature reading tips outlined [here](#), it allows you to quickly go through dozens of papers. Tab Mix Plus allows you to have an arbitrary number of tabs open, which is very useful. While we won't be doing much literature reading just yet, I suggest getting the above utilities so that you can open each of the links during the installation procedures in a tab.

2. *Installing Unix*

The very first task at hand is to set up your own Linux box. If you already run a Linux box, you can skip this particular subproblem. The instructions below are fairly detailed. If you have problems, especially platform specific problems, the first thing to do is try to use Google solve the problem on your own. Googling the error message is often the best way to do this. In real research, there is often no “technical support” other than your own ingenuity and search skills, so this is a good skill to develop. Of

course, if you still have problems after making a good faith effort, then don't hesitate to post a question on the blog or ask us by email.

In order to allow yourself to Google if necessary, before you start installing Linux, you will want to have at least one other computer with internet access (such as a cluster computer or a laptop with a wireless card) in order to deal with problems as they arise.

- (a) The first thing to do is to get access to a system on which you can install Linux. You will want access to a system which has space for at least 50GB or so of data¹. This can be a dual boot of your current PC.
- (b) If you have a Mac², you can get essentially complete Unix functionality by obtaining [DarwinPorts](#) and installing all the important GNU programs; of these, the text utilities like `grep` and `cut` are some of the most useful. I have put together a list of all of my [installed Darwin Ports](#); you won't go wrong if you just install everything on that list.

The GNU versions of Unix programs are the most powerful and feature rich available and far better than the BSD versions which come standard with OS X. One problem is that the BSD programs by default take precedence at the command line, and that the Darwinports binaries for GNU commands have a `g` in front of them (e.g. `gcut` rather than `cut`). You can get around this with the following steps:

- i. First, symbolically link the GNU binaries in the `/opt/local/bin` directory to their standard names with the `ln` command (e.g. running `ln gcut cut` from within the `/opt/local/bin` directory). You can run the following commands to accomplish this; make sure you look at each stage of the piped command to see what these commands are doing:

```
cd /opt/local/bin
ls gnu* | sed 's/^gnu//g' | awk '{print "ln gnu"$1" "$1}' | bash
ls g* | grep -v '^gnu' | sed 's/^g//g' | awk '{print "ln g"$1" "$1}' | bash
```

- ii. Then change the `$PATH` variable in the `.bashrc` to put `/opt/local/bin` first (e.g. including the line `PATH=/opt/local/bin:$PATH` in your `.bashrc` file). See the sample `.bashrc` file on the webpage.
- (c) If you have a PC, you will want to install Ubuntu Linux if you've never used Linux before. Note that if you already have some other flavor of Linux, it will be perfectly fine for this course and you can skip this step. I chose Ubuntu as it's probably the easiest Linux distribution to install and use.
 - i. In order to install Ubuntu, you will need to [download](#) the ISO CD for version 6.10.

¹You might be able to get by with less, but it will be easier if you don't need to keep zipping and unzipping files to do things. File sizes and directories in bioinformatics can be huge.

²The Mac is, in my opinion, the platform of choice for a bioinformatics frontend. The ideal situation is a Mac frontend (such as a laptop) and a powerful Linux cluster or server backend which you `ssh` into to do big jobs. This provides the perfect mix of Unix power tools like `emacs` and `grep` with office productivity (Word, Pages, Keynote, etcetera), graphics (Photoshop, OmniGraffle, Illustrator), and scientific software (TeXShop, LaTeXIt, Preview).

- ii. An ISO is a disk image which you need to burn to CD³ in a special way to ensure that you can install from it. You can't just burn it as data. Instructions and tools for burning ISO CDs can be found at <http://isorecorder.alexfeinman.com/isorecorder.htm> and <http://iso.snoekonline.com/iso.htm>.
- iii. Once you have the ISO CD, place it in the CD drive and reboot your PC. The next steps are fairly self explanatory, though you can see this [installation with screenshots](#). Make sure that you have some free space to set up a a partition for Linux. The number one thing you *don't* want to do is erase your old data. You may want to install server packages, if you want to `ssh` into your machine remotely, set up a web page, or allow other people to access your files via `ftp`. You won't need to do these things for this course, but often machines get repurposed for other things and it's often easier to just install everything you think you might need at the beginning (assuming you have enough space). See [here](#) for directions on a LAMP installation.
- iv. You now want to install some useful packages. Many packages come standard with Ubuntu; many others that you will want can be obtained through Scibuntu <http://scibuntu.sourceforge.net/> and Easy Ubuntu <http://easyubuntu.freecontrib.org>, and you might also want to check out http://ubuntuguide.org/wiki/Ubuntu_Edgy.
- v. Arbitrary packages can be installed with `aptitude`; for example, the invocation


```
sudo aptitude install lyx
```

 will “do as superuser” (`sudo`) the `aptitude` based installation of the `lyx` L^AT_EX editing software. For the most part we are not going to be using tons of Unix software outside the basic GNU programming tools and text utilities, most of which come standard; however, in the process of installing new software one often has to install new libraries and this can be a hassle. We will cross these bridges as we come to them during the quarter. Blog posts are the best way to alert me to unforeseen dependency issues.
- vi. Finally, you will want to ensure that your Unix box is connected to the internet by running `ping www.stanford.edu` at the command line; if you get packets back, then you're good. Otherwise you will want to contact your local network administrator to help you get online.

3. *Intro to Unix*

Now that you have a box of your own up and running, we will learn the basic Unix commands and familiarize yourself with a variety of powerful text processing and programming tools. To help you get started, I've included several sample configuration files for `bash`, `emacs`, and `screen` in the [code](#) section of the website.

- (a) Basic Commands: The best way to get exposure to the basic Unix commands is through the interactive tutorial `learn`.
 - i. The first thing to do at your new Unix box is to open up a terminal and make a secure shell (`ssh`) connection. Type in something similar to:

³Make sure you're burning to a CD and not a DVD!

```
ssh -l balajis elaine.stanford.edu
```

...but put your leland username after the login (-l) flag. You will then be asked for your password, after which you will be at the prompt for a remote machine. If you have any problems with this,

<http://www.stanford.edu/services/unix/loggingin.html> and

<http://www.stanford.edu/services/unixcomputing/> will be of use.

- ii. Now run the `learn files` command. Once you're done with that, run the `learn morefiles` command. This interactive learning program was written many years ago, and it's slightly out of date, but as an overall introduction to Unix it can't be beat.
- iii. You can also check out this Unix tutorial:
<http://www.ee.surrey.ac.uk/Teaching/Unix/>
this list of 15 essential Unix commands:
<http://www.petefreitag.com/item/426.cfm>
and this quick guide:
<http://freeengineer.org/learnUNIXin10minutes.html>.
The Unix Power Tools book will also be helpful in looking up any activity which you want to perform, as it's organized by topic.
- iv. **To confirm that you finished the tutorial, please write down and include on your homework the last question that is asked by `learn morefiles`.**

(b) The `bash` shell and GNU TextUtils

- i. `bash`, which stands for the "Bourne-again shell", is the underlying program which processes commands at the command line. There are a number of preferences for the shell which you can set up using the `.bashrc` file from the website as a starting place.
- ii. You might think that the command line is a pretty prosaic thing, but there's actually quite a few tricks that `bash` has up its sleeve, including searchable command histories, autocompletion, and (most important of all) pipes. Pipes allow you to redirect the output of one command into the input of the next. For example, consider the command:

```
head -n 5 blah.txt | tail -n 3
```

This would first extract take the first 5 lines of `blah.txt`, and then extract the last 3 lines with `tail`.
- iii. Note the use of the `-n` flags after the commands. These flags modify the default behavior of a program. Whenever using a new program for the first time, it is important to look at the list of all possible flags and see which ones might be useful. You can usually get this list by using the `-help` or `-h` flag; for example, try running `head -help` and see what is returned.
- iv. To get some idea of `bash`'s capabilities, read through this tutorial:
http://www.hypexr.org/bash_tutorial.php
Pay particular attention to the section on pipes, as you will need it for the text processing section below.
- v. The GNU command line text parsing utilities are invaluable. A full list is here:
<http://www.gnu.org/software/textutils/textutils.html>

I've selected some of the most important ones below; we used `head` and `tail` above. It is amazing how useful the Unix text utilities are for bioinformatics applications. The next problem will give you some flavor for this.

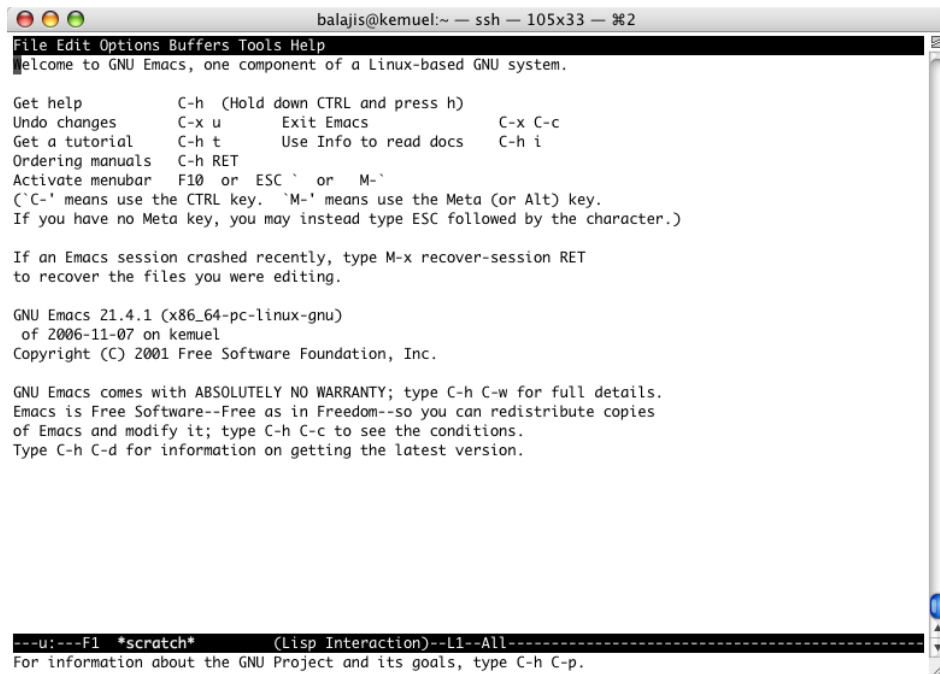
- A. `cat`: concatenate files and print to the standard output
 - B. `csplit`: split a file into sections determined by context lines
 - C. `cut`: remove sections from each line of files
 - D. `join`: join lines of two files on a common field
 - E. `nl`: number lines of files
 - F. `paste`: merge lines of files
 - G. `sort`: sort lines of text files
 - H. `split`: split a file into pieces
 - I. `sum`: checksum and count the blocks in a file
 - J. `tail`: output the last part of files
 - K. `tr`: translate or delete characters
 - L. `uniq`: remove duplicate lines from a sorted file
 - M. `wc`: print the number of bytes, words, and lines in files
 - N. `head`: output the first part of files
- vi. Now we will run some commands on real data to get the flavor of how text processing is used in bioinformatics. **Please do the following session and explain what each command is doing.** One line will be sufficient. If a command flag is used, such as `-r`, use the `-help` flag as described above and write down what the flag is doing. If a symbol other than the alphanumeric symbols is used, such as `*`, explain what it's doing. The first one is done as an example. It will often be easier to type in the commands and then figure out what they're doing.
- `cd ~` (This command changes to your home directory and is equivalent to `cd $HOME`; we do this so we can have write access.)
 - `mkdir stat366_hw1`
 - `wget -r ftp://ftp.ncbi.nih.gov/genomes/Bacteria/Escherichia_coli_K12`
 - `mv ftp.ncbi.nih.gov/genomes/Bacteria/Escherichia_coli_K12/ .`
 - `rmdir -p ftp.ncbi.nih.gov/genomes/Bacteria/`
 - `cd Escherichia_coli_K12/`
 - `ls`
 - `ls -l`
 - `ls -alrt`
 - `ls -alrS`
 - `file *`
 - `mkdir binaries`
 - `file * | cut -f1 -d':'`
 - `file * | grep data | cut -f1 -d':' | xargs mv --target-directory=binaries/`
 - `head *ptt`
 - `head *rnt`
 - `tail *rnt`

- `ls *.*`
- `file *.*`
- `less *ptt` (use Emacs keybindings to move around and `q` to exit)
- `grep 'DNA' *ptt`
- `grep --color 'DNA' *ptt`
- `grep --color 'division' *ptt`
- `head -n 5 *ptt`
- `wc *ptt` (Compare the number of lines in the file to the number of proteins reported in the header with the previous command)
- `nl *ptt`
- `tail -n +4 *ptt` (A number of lines will scroll down your screen)
- `tail -n +4 *ptt | head`
- `nl *ptt | tail -n+4 | head`
- `tail -n +4 *ptt | awk '{if ($3 < 100 && $3 > 95) { print $0;}}'`
- `cut -f8 *ptt | head`
- `cut -f8 *ptt | sort | uniq -c`
- `cut -f8 *ptt | sort | uniq -c | sort -k1 -rn | less` (Which COG category is the most frequent in *E. coli*?)
- The following should all be typed on one line.


```
tail -n +4 *ptt | cut -f8 | sort | uniq -c |
sort -k1 -rn | awk '{print $1"\t"$2}' > coli_cogs.txt
```
- `cat coli_cogs.txt`

(c) Emacs

- i. Emacs is an extraordinarily powerful text editing tool which you will find useful throughout your scientific career. It is the tool of choice for a statistically minded computational biologist, as it allows you to run R and Matlab from within Emacs (which we will see next week). The best way to learn Emacs is through the online tutorial. If you type in `emacs` at the command line, you will obtain the following screen:



```
balajis@kemuel:~ — ssh — 105x33 — %2
File Edit Options Buffers Tools Help
Welcome to GNU Emacs, one component of a Linux-based GNU system.

Get help          C-h (Hold down CTRL and press h)
Undo changes     C-x u      Exit Emacs          C-x C-c
Get a tutorial   C-h t          Use Info to read docs  C-h i
Ordering manuals C-h RET
Activate menubar F10 or ESC ` or M-`
(`C-' means use the CTRL key. `M-' means use the Meta (or Alt) key.
If you have no Meta key, you may instead type ESC followed by the character.)

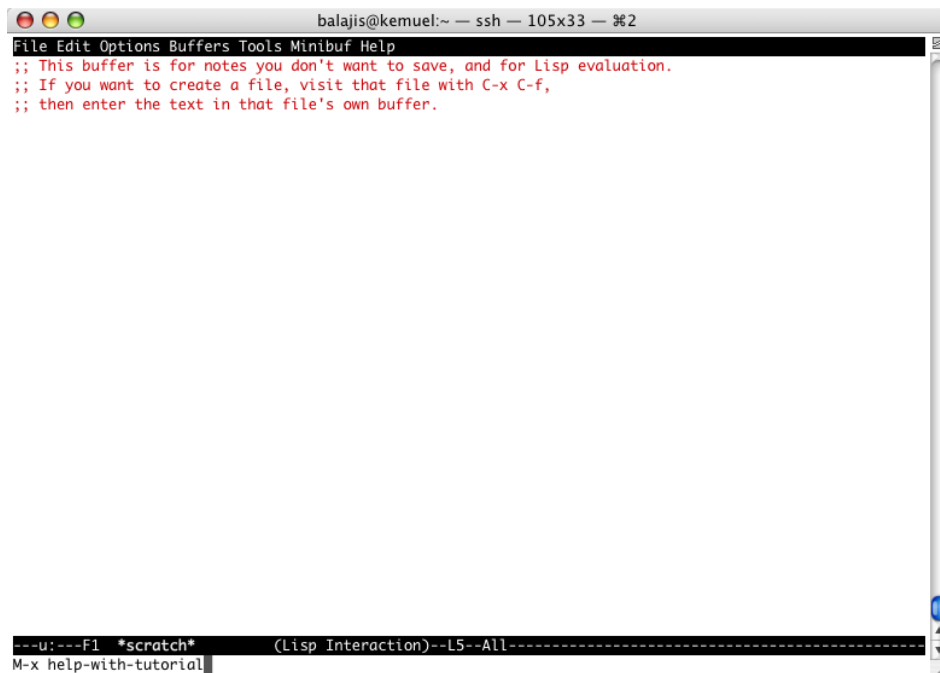
If an Emacs session crashed recently, type M-x recover-session RET
to recover the files you were editing.

GNU Emacs 21.4.1 (x86_64-pc-linux-gnu)
of 2006-11-07 on kemuel
Copyright (C) 2001 Free Software Foundation, Inc.

GNU Emacs comes with ABSOLUTELY NO WARRANTY; type C-h C-w for full details.
Emacs is Free Software--Free as in Freedom--so you can redistribute copies
of Emacs and modify it; type C-h C-c to see the conditions.
Type C-h C-d for information on getting the latest version.

---u:--F1 *scratch* (Lisp Interaction)--L1--All-----
For information about the GNU Project and its goals, type C-h C-p.
```

- ii. If you then type M-x, which means “hold meta and push x at the same time” (where Meta is usually the Alt or possibly the Escape key), you will see a cursor appear in the lower left corner of the screen. Type in `help-with-tutorial` as shown and press enter:



```
balajis@kemuel:~ — ssh — 105x33 — %2
File Edit Options Buffers Tools Minibuf Help
;; This buffer is for notes you don't want to save, and for Lisp evaluation.
;; If you want to create a file, visit that file with C-x C-f,
;; then enter the text in that file's own buffer.

---u:--F1 *scratch* (Lisp Interaction)--L5--All-----
M-x help-with-tutorial
```

- iii. You will now be at the emacs tutorial. It's fairly self explanatory. Please go through the whole thing and then answer the questions below:

```

balajis@kemuel:~ -- ssh -- 105x33 -- #2
File Edit Options Buffers Tools Help
You are looking at the Emacs tutorial. See end for copying conditions.
Copyright (c) 1985, 1996, 1998, 2001, 2002 Free Software Foundation.

Emacs commands generally involve the CONTROL key (sometimes labeled
CTRL or CTL) or the META key (sometimes labeled EDIT or ALT). Rather than
write that in full each time, we'll use the following abbreviations:

C-<chr> means hold the CONTROL key while typing the character <chr>
Thus, C-f would be: hold the CONTROL key and type f.
M-<chr> means hold the META or EDIT or ALT key down while typing <chr>.
If there is no META, EDIT or ALT key, instead press and release the
ESC key and then type <chr>. We write <ESC> for the ESC key.

Important note: to end the Emacs session, type C-x C-c. (Two characters.)
The characters ">>" at the left margin indicate directions for you to
try using a command. For instance:

[Middle of page left blank for didactic purposes. Text continues below]

>> Now type C-v (View next screen) to move to the next screen.
(Go ahead, do it by holding down the CONTROL key while typing v).
From now on, you should do this again whenever you finish
reading the screen.
-----F1 TUTORIAL (Fundamental)--L1--Top-----
You can run the command 'help-with-tutorial' with C-h t

```

- iv. The main thing to remember about the emacs keybindings is that they are a little counterintuitive at first (why hit C-n, meaning “hold control and push n”, to go down when there is a perfectly good arrow key?), but they make a tremendous amount of sense in the context of the qwerty keyboard. You will be amazed how much faster you can knock out code when you never have to move your hands from the touch typing position. Moreover, the effort you put into learning Emacs pays off: every OS X application, the Bash command shell, and Firefox all have emacs keybindings on by default. You can also run R and Matlab, debug Perl, and compile C++ from within Emacs. It’s really well worth learning.
- v. **Here are some review questions to hand in with your HW to confirm that you’ve learned the tutorial.**
 - A. How do you move up? down? left? right?
 - B. How do you move to the front of the line? to the back of the line?
 - C. What does the command sequence M-< do? What about M->?
 - D. What does the command sequence C-x 0 do? What about C-x 1? C-x 2? C-x 3?
 - E. How do you kill text? How do you paste text? What happens if you keep hitting M-y after you paste text (this is called the “kill ring” – it’s a buffer in memory that holds several past kill regions)

(d) The **screen** window multiplexer

- i. **screen** is a utility that does for windowing and tabs what emacs does for typing. Think of it like a tab emulator. You will frequently want to run multiple simultaneous **ssh** connections to the same machine, for example to switch back and forth between writing files in one directory and looking at the progress of a download in another.
- ii. The **.screenrc** file available on the website configures **screen** so that it plays well with **emacs** and **bash**

- iii. After copying the `.screenrc` file into your home directory, take a look at the tutorials here: <http://librenix.com/?inode=4202>
- iv. Pay particular attention to the commands for creating new window detachment/reattachment. One of the huge advantages of `screen` is that it lets you detach screens, when you're going home or logging off a remote machine for example, and then reattach them later to pick up your work where you left off⁴

4. Perl Basics

- (a) The first task here is to download and run `hello.pl`. The following command should just work on any Unix system:

```
perl hello.pl
```

- (b) The following tutorial will introduce you to the syntax of Perl.

<http://perldoc.perl.org/perlintro.html>

Note that Perl is an extremely good language for scripting the web, stacking together a bunch of Unix commands which are too complicated to enter one by one at the command line⁵, and doing all kinds of text processing. The practical point for a statistically/computationally inclined biologist is that you will usually use Unix to organize your data at a high level (moving it into directories, setting up hierarchies, viewing it, sharing it), Perl to massage it into shape for data analysis, and then R or Matlab or possibly C++ for the actual data analysis.

- (c) **To solidify your understanding of perl, I would like you to write a simple program.** This program should be called `cog2faas.pl`. The purpose is to put the protein sequences from each COG category in *E. coli* K12 into individual files in preparation for calculating multiple alignments for each COG protein family. The inputs will be the `NC_000913.ptt` and `NC_000913.faa` files in the `Escherichia_coli_K12` directory which we downloaded in the text processing section above. The output will be a number of files, one for each COG, which you should place in the subdirectory `cog_faas`. Please title each file by the category followed by an `.faa`. For example, all the sequences for COG0083E should go into `COG0083E.faa` in **FASTA format**, which is the same format as `NC_000913.faa` with a one line header followed by the protein sequence. The invocation should be specified on the command line as follows:

```
perl cog2faas.pl NC_000913.ptt NC_000913.faa cog_faas
```

Within the program, the three arguments after the `cog2faas.pl` script will be placed into the `@ARGV` variable, as described on the Perl tutorial page:

<http://perldoc.perl.org/perlintro.html>

⁴One issue is that the `$DISPLAY` environmental variable often gets unset when you log back into screen with an `ssh -X` connection with X11 forwarding. This is somewhat annoying as there are processes, like R, which crash or behave badly if the X11 forwarding is changed in the middle of the run. I haven't yet figured out an elegant way around this short of hardcoding the IP address for the display variable into the bash config file.

⁵There are lots of people who write `.sh` files for shell scripting, but I always use Perl for such tasks. It's a more full featured programming language and its capabilities are a superset of anything that shell scripting allows.

(d) Here is a suggested way to write this program. You don't have to follow this structure if you don't want to.

- i. First get the arguments from the command line and store them in variables. You don't need to do any error checking to make sure that the inputs are valid right now – just assume the user knows what he's inputting and in the right order.
- ii. Next, open up a filehandle to the `.ptt` file. Iterate over the `.ptt` file and make a hash of arrays (See Perl Cookbook Section 11.2), which maps each GI number to an array of COG categories for that GI. You need an array in order to deal with GIs with multiple COG categories. Neglect those with dashes as their COG category, and make sure to skip the three line header of the `.ptt` file. The following code may help you get started:

```
#assume $pttfile contains the name of the ptt file
#and that all variables have been defined above
open(PTT,$pttfile);

#skip the three header lines
<PTT>;
<PTT>;
<PTT>;

#iterate over the lines of the PTT file
my($line,@tokens);
while(defined($line = <PTT>)) {
    chomp($line);
    @tokens = split(/\t/, $line);

    #now do stuff with these tokens
}
```

- iii. Now iterate over the `.faa` file. For each sequence, parse out the GI number from the header using Perl's text processing capabilities. Look up the corresponding COG categories and write out the sequence to the corresponding COG file in the cog subdirectory. You will want to use the append to file syntax (see Perl Cookbook 249), but make sure that in between debugging runs that you clear out the extra `.faa` files after each run.
- iv. Note: it's kind of a pain to have a variable number of rows per sequence record. You can definitely handle this in Perl with a while loop and temp variables, but it adds complexity to the parsing. Therefore, you may wish to use the below command line invocation to transform your `.faa` file to a `.faa.oneline` file (it goes all one one line):

```
sed 's/^>/@>/g' NC_000913.faa | sed 's/]$/]@/g' | tr -d "\n" |
sed 's/@/\n/g' | tail -n +2 > NC_000913.faa.oneline
```

You should look at each pipe stagewise to see the tricks that I'm using there.

I'm making some assumptions about the `.faa` file format which are not necessarily generally satisfied in order to do a quick command line hack to get one line per sequence. It is also possible to use a one line perl script to delete newlines at the ends of lines which lack a leading `>`; this is what you'd use in general. In any case, if you do decide to use this command line invocation, you can modify your code to accept a `.faa.oneline` file rather than a `.faa` file.

- (e) When writing this program, you will want to use the perl debugger. This will pay for itself in the long run. You can run the debugger at the command line with

```
perl -d cog2faas.pl NC_000913.ptt NC_000913.faa cog_faas
```

However, the best way to run it is from within emacs. See the following screenshot:

See here for tutorials:

<http://www.sunsite.ualberta.ca/Documentation/Misc/perl-5.6.1/pod/perldebtut.html>

<http://www.ddj.com/184404744>

and see here for how to run this from within emacs:

<http://lug.umbc.edu/tutorials/adv-emacs.html>