**Creating a configuration file.**
We can put aliases into a configuration file that is loaded automatically when you open a new terminal. This can store aliases. It can also set or modify some environment variables. The configuration file is in your home directory. Your home directory is available from
`echo $HOME`

In the bash shell, the login configuration file is called .bash_profile or .bash_login or .profile. If you have a configuration file already, you can look at it with `less` and add to it. If you do not, we will create a new one.

```
nano .bash_profile
```

Type in:
```
alias rm='rm -i'
```

If we type
```
touch hello
rm hello
```

We do not get a message. The file is deleted.

```
source .bash_profile
#or source ~/.bash_profile

touch hello
rm hello
```

We should now get a warning message with the `rm` command.

Because the command is in the configuration file, it should now work every time you login without using source. Try logging out and logging back in. Then type:

```
touch hello
rm hello
```

We should now get a warning message with the `rm` command.

Within the configuration file, you can add aliases, change environment variables, print welcome messages or specify a unix command or program to run. For example, add:

```
MYDATE=`date "+%H:%M:%S %m/%d/%y"`
echo "Welcome $USER. The current time is $MYDATE"
```

Note the command is enclosed in backtick characters. Now try ending and restarting your terminal.

**Making UNIX programs**

We may want to treat the file containing the bash instructions as if it was any other unix command. We want to simply type the file name to run it.

Let's make a simple unix program in nano, hello.sh
The program has two lines:

```
#a shell script
echo "hello"

lewis$ hello.sh
-bash: hello.sh: command not found
```

Why can't UNIX find this command?
-- you have to explicitly give permissions to files that are going to be run as programs. Solution: add the executable permission.
-- UNIX will only look in certain places for files that can be run as programs. Solution: have it look in the directory with your executable.

For file permissions, recall we have read/write/execute permissions for user/group/other
A new file has read, write, but not execute permissions by default.

```
-rw-r--r--  1 lewislukens  staff   20  2 Jan 12:17 hello.sh

chmod u+x hello.sh

-rwxr--r--  1 lewislukens  staff   20  2 Jan 12:17 hello.sh

hello.sh
-bash: hello.sh: command not found
```

This does not work, but…

```
./hello.sh #does work. The dot refers to the current directory
```

Once a script is executable, you can always run it by prefixing its name with a path. You cannot run a program without the path if you are in the same directory as the program.

Modifying your path
Anytime you try running a program, UNIX will check through a list of predefined directories to see if that program exists in those locations. A program is any file that has executable permission.

The $PATH environment variable has the list of where UNIX looks for programs.

```
echo $PATH
```

Here is at $PATH example:

```
/Library/Frameworks/Python.framework/Versions/3.7/bin:/Library/F
rameworks/Python.framework/Versions/3.7/bin:/usr/local/bin:/usr/
bin:/bin:/usr/sbin:/sbin:/opt/X11/bin
```

The order of directories within the path determines the order in which UNIX will search for programs.

We can modify the $PATH. Below <your_directory> is the path to the directory with the program hello.sh starting from your home directory, which is depicted as ~.

```
export
PATH=$PATH:~/Teaching_admin/Bioinfclasses/BINF6410/Unix_and_Perl
/Code
```

Check Now:
```
echo $PATH
```

What do you see? This should have added a new directory to the end of the list. Then…

```
hello.sh
hello
```

Success! But… modifying path is only good for that terminal session. If a new terminal is opened up, it is wiped away.

Move the command into the `.bash_profile` file. Now, every time you login, you should be able to issue the command `hello.sh`!