Programming For Biology 2022

programmingforbiology.org

Instructors
Simon Prochnik
Sofia Robb

Big Picture

Why?

Why is it important for **biologists** to learn to program?

You might already know the answer to this question since you are here.

We firmly believe that knowing how to program is just as essential as knowing how to run a gel or set up a PCR reaction. The data we now get from a single experiment can be overwhelming. This data often needs to be reformatted, filtered, and analyzed in unique ways. Programming allows you to perform these tasks in an **efficient** and **reproducible** way.

Helpful Tips

What are our tips for having a successful programming course?

1. Practice, practice, practice. Please spend as much time as possible actually coding.
2. Write only a line or two of code, then test it. If you write too many lines, it becomes more difficult to debug if there is an error.
3. Errors are not failures. Every error you get is a learning opportunity. Every single error you debug is a major success. Fixing errors is how you will cement what you have learned.
4. Don't spend too much time trying to figure out a problem. While it's a great learning experience to try to solve an issue on your own, it's not fun getting frustrated or spending a lot of time stuck. We are here to help you, so please ask us whenever you need help.
5. Lectures are important, but the practice is more important.
6. Review sessions are important, but practice is more important.
7. Our key goal is to slowly, but surely, teach you how to solve problems on your own.

Unix

Unix 1

Unix Overview

What is the Command-Line?

Underlying the pretty Mac OSX Graphical User Interface (GUI) is a powerful command-line operating system (OS). The command-line gives you access to the internals of the OS, and is also a convenient way to write custom software and scripts.

Many bioinformatics tools are written to run on the command-line and have no Graphical User Interface. In many cases, a command-line tool is more versatile than a graphical tool, because you can easily combine command-line tools into automated scripts that accomplish custom tasks without human intervention.

In this course, we will be writing Python scripts and running them exclusively from the command-line based.

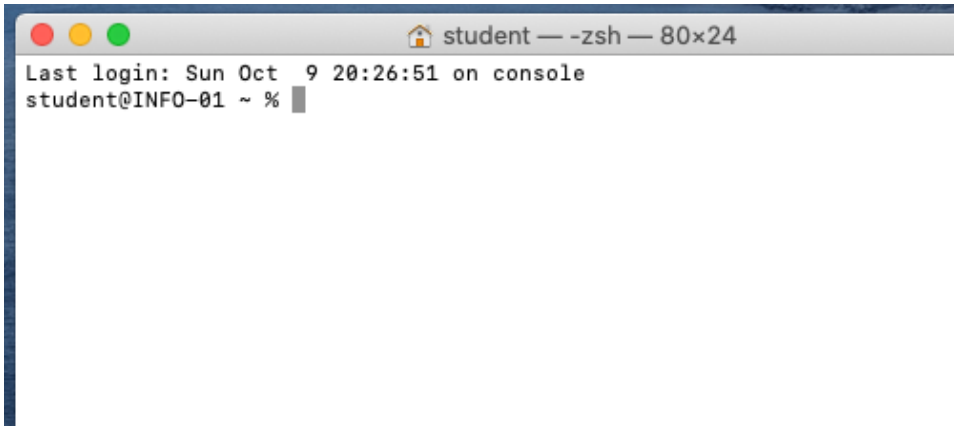
The Basics

Logging into Your Workstation

Your workstation is an iMac. To log into it, provide your user name and password. Your username is 'student' and the password is 'pfb2022'

Bringing up the Command-Line

To bring up the command-line, use the Finder to navigate to *Applications->Utilities* and double-click on the *Terminal* application. This will bring up a window like the following:



You can open several Terminal windows at once. This is often helpful.

You will be using the Terminal application a lot, so I suggest that you drag its icon into the shortcuts bar at the bottom of your screen.

OK. I've Logged in. What Now?

The terminal window is running **shell** called "zsh". (Mac recently changed from bash) The shell is a loop that:

1. Prints a prompt
2. Reads a line of input from the keyboard
3. Parses the line into one or more commands
4. Executes the commands (which usually print some output to the terminal)
5. Go back step 1.

There are many different shells with bizarre names like **bash**, **sh**, **csh**, **tcsh**, **ksh**, and **zsh**. The "sh" part means shell. Each shell has slightly different syntax and features. Your accounts are set up to use **zsh**. It's very similar to **bash** which is standard on linux systems. Stay with **zsh** and you'll get used to it, eventually.

Command-Line Prompt

Most of bioinformatics is done by running command-line software in a shell, so you should take some time to learn to use the shell effectively.

This is a command-line prompt:

```
bush202>
```

This is another:

```
(~) 51%
```

This is another:

```
srobb@bush202 1:12PM>
```

What you see depends on how the system administrator has customized your login. You can customize it yourself (but we won't go into that here)

The prompt tells you the shell is ready to accept a command. Most commands run almost instantly, but if you run a long command, the prompt will not reappear until it is finished and the system is ready to accept your next request.

6. Save `:w` and Exit `:q`
7. (Add) Stage your changes. `git add git_exercises.txt`
8. (Commit) Become sure you want your changes. `git commit -m 'added a line of text'`
9. (Push) Sync/Upload your changes to the **remote** repository. `git push origin master`

That is all there is to it! There are more complicated things you can do, but we won't get into those. You will know when you are ready to learn more about git when you figure out there is something you want to do but don't know how. There are thousands of online tutorials for you to search and follow.

Keeping track of differences between local and remote repositories

If you are ever wondering what do you need to add to your remote repository use the `git status` command. This will provide you with a list of files that have been modified, deleted, and those that are untracked. Untracked files are those that have never been added to the staging area with `git add`

command	description
`git status`	To see a list of files that have been modified, deleted, and those that are untracked

Deleting and moving files

command	description
`git rm`	Remove files from the index, or from the working tree and from the index
`git mv`	Move or rename a file, a directory, or a symlink

> these two commands will update your index as well as change your local files. If you use just `rm` or `mv` you will have to update the index with add/commit.

Get a copy of file on your remote

Sometimes you really really mess up a file, or you delete it by mistake. You have a small heart attack then you remember that you have a good copy in your remote github repo. How do you get it in your local repo?

```
git checkout <filename>
```

> Whew, what a life saver!

Tips

1. Adding files over 50M will break your git repo. Don't add large files. Don't blindly use `git add -A` when there might be large files present. You will be very sad if you do.
2. Don't clone a git repository into another git repository. This makes git really unhappy.
3. Don't be afraid to ask your questions on Google. git can be complicated and a lot of people ask a lot of

questions that get answered in online forums, or GitHub will have a tutorial

Cloning a Repository

Sometimes you want to download and use someone else's repository. This is different form above where we created our own repository. This is just a copy of someone else's repository

Let's clone the course material.

Let's do it!

1. Go to our PFB GitHub Repository
2. Click the 'Clone or Download' Button
3. Copy the URL
   ~Clone PFB2022
4. *Clone* the repository to your local machine
   `git clone https://github.com/prog4biol/pfb2022.git`

Now you have a copy of the course material on your computer!

Bringing Changes in from the Remote Repository to your Local Repository

If changes are made to any of these files in the online, remote repository, and you want to update your local copy, you can *pull* the changes.
`git pull`

command	description
`git pull`	To get changes from the remote into your local copy

Links to *slightly* less basic topics

You will KNOW if you need to use these features of git.

1. View Commit History
2. Resolving Merge Conflicts
3. Undoing Previous Commits
