Module 1a:

Coding for Biologists

Michelle Franc Ragsac (She/Hers/Siya)
4th Year BISB Student, Emma Farley Lab
mragsac@eng.ucsd.edu

September 13, 2021

Learning Goals of Module 1a

1. Navigating with Command Line Interfaces (CLI) using a Terminal Application
2. Package & Environment Management with `conda`
   a. Learning about the `conda` package manager
   b. Configuring the `bioconda` channel for bioinformatics package installation
   c. Creating, Saving, and Loading new `conda` environments
   d. Reviewing commonly-used Python packages for bioinformatics (e.g., `jupyterlab`, `numpy`, etc.)
3. Brief Introduction to Programming in Python
   a. The 5 Basic Concepts found in most programming languages
   b. Examples on common programming tasks for bioinformatics work
      i. Printing & Manipulating Text: **Determining Fragments after a Restriction Enzyme Digest**
      ii. Reading & Writing Files: **Evaluating a FASTQ File for Unique Sequence**

Checking in about the summer homework...

There are some tasks we gave you to do over the summer, including a lot of installations!

Has everybody installed everything before we continue with the module?

github.com/mragsac/BISB-Bootcamp-2021/tree/main/module0_welcome-

BISB-Bootcamp-2021/module0_welcome-to-bootcamp at main · mragsac/BISB-Bootcamp-2021

README.md

How should I prepare for the first day of bootcamp?

Action Items

- [ ] **If you have a Windows computer**, please make sure that Linux is available to you! While there are several ways to do this, we suggest the **Windows Subsystem for Linux**: https://docs.microsoft.com/en-us/windows/wsl/.
- [ ] Install `bash` on your computer and learn the basics if you don't already know them!
- [ ] Install `miniconda` on your computer to get the `conda` package manager.
- [ ] Sign up for **GitHub** on your computer and **star** this repository!
- [ ] Install `git` on your computer and learn the basic syntax.
- [ ] Familiarize yourself with the basic syntax for the `Python` programming language if you don't already know it!

Long Answer

Bootcamp moves pretty fast, so it's important to install some things ahead of time so that we can hit the ground running on September 13! During bootcamp, we'll have some hands-on, practical programming sessions, so please have the following things set up:

1. **A terminal to use for the sessions**: We will learn how to navigate your computer (and a remote computer!) using `bash` in a *Command Line Interface* (CLI). If you don't know what that means, we've added some tutorials in this folder!

2. **A package manager to set up software environments**: We'll go over how to create coding environments for more reproducible analyses using the `conda` package manager.

3. **A way to version your code and collaborate with others**: Finally, we'll also cover how to track changes you (or your teammates) make to software using `git`!

Please do not arrive on the first day without the appropriate software installed because it'll be easy to fall behind! We expect that you are all self-directed, enthusiastic learners, so as you address the above list of requirements, please feel free to ask us to clarify or assist, but please prioritize!

If you haven’t installed things yet, I have a few hidden slides for installation instructions for those on a **macOS**/**Linux** computer and those on a **Windows** computer!

Local Computer
Set-Up & Installation

macOS & Linux Computers

Luckily, you have a terminal application already!

One of the main things we wanted you to set up was a **terminal** application so that you're able to interact with your computer using **command-line input** (e.g., text)

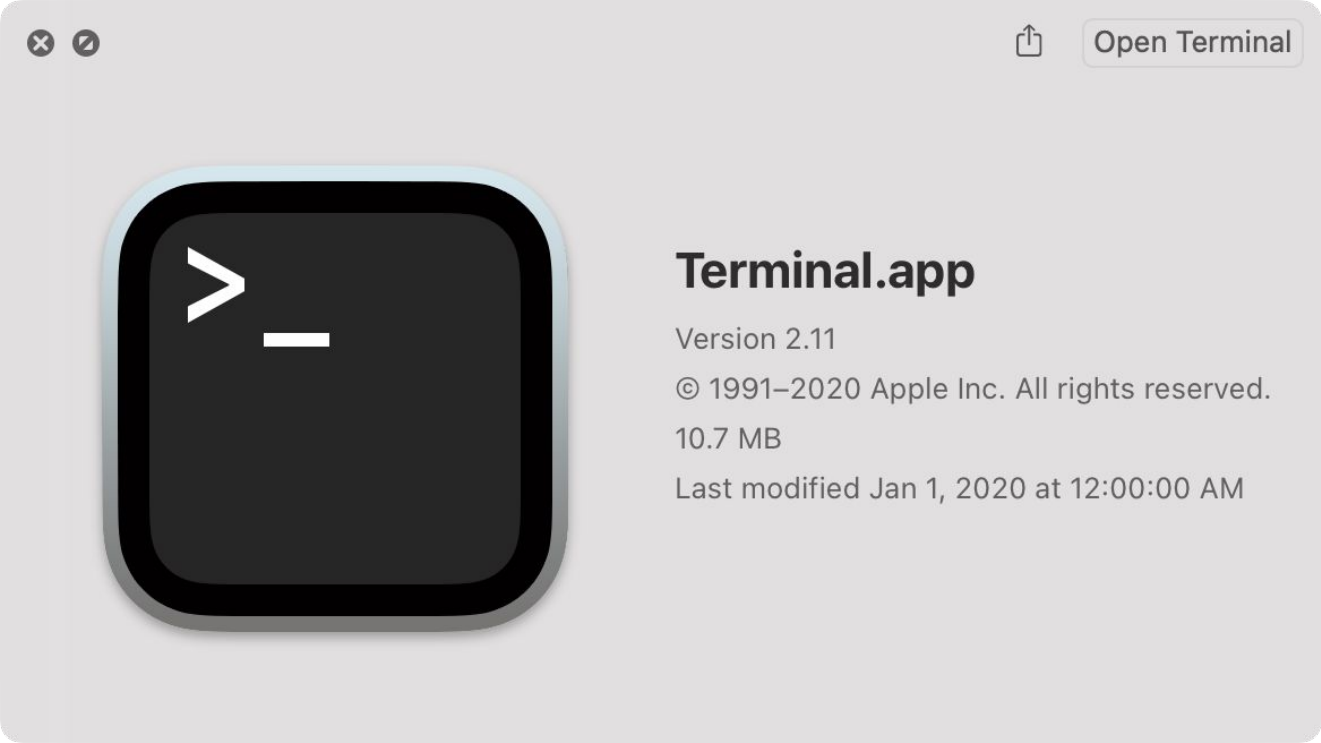
Luckily, that application is already pre-configured the way we want it to be on Macintosh and Linux-based operating systems! The application is called **Terminal**!

If you have a **Mac**, you can find the application under the following path:

`Applications > Utilities > Terminal`

If you have a **Linux** system, you can find the application under the following path:

`Applications > Terminal`



The terminal application icon on a macOS computer

Installing Homebrew (`brew`) for macOS computers

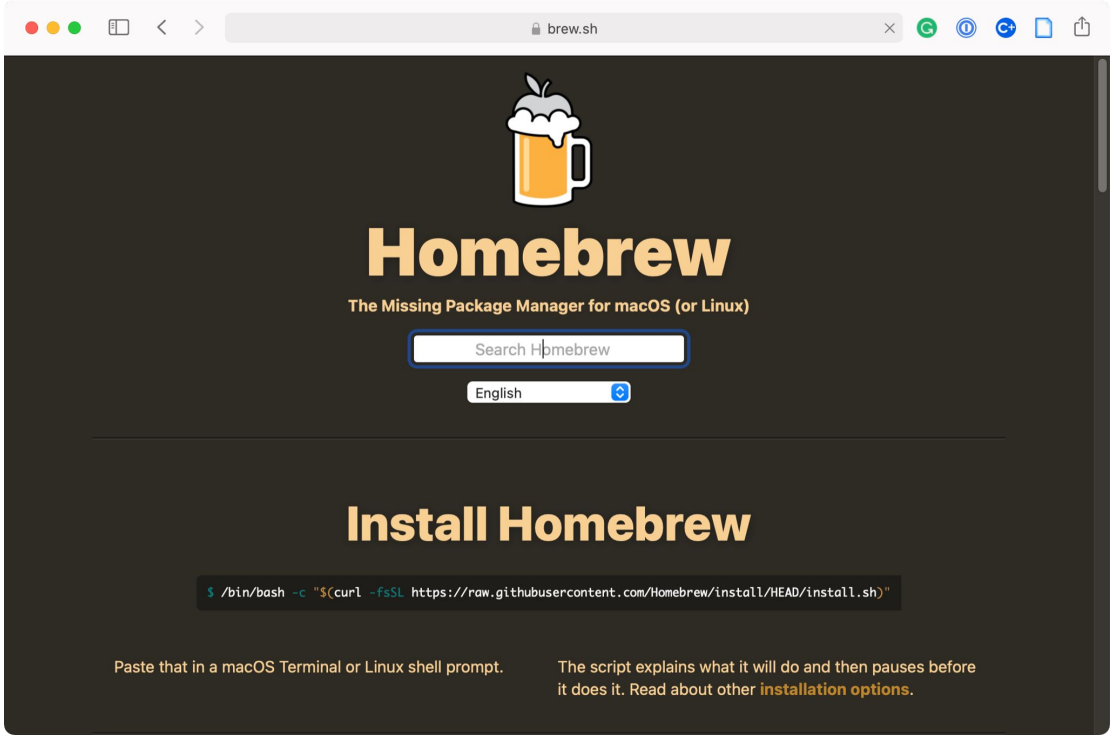
The macOS operating system doesn't have a package manager of its own like other UNIX-based operating systems (e.g., Ubuntu Linux has **apt**)

Homebrew (https://brew.sh) solves this issue by acting as a third-party package manager that can install commonly-used UNIX commands that are missing from macOS computers

If you have a **Mac**, you can install Homebrew by copying the following command into your terminal, then pressing **ENTER** to run the command:

```
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

Note: This command was taken directly from the Homebrew site! What this command does is run the Homebrew installation bash script found at the specified URL to install `brew` *on your computer.*



The Homebrew website with the installation command

Installing `wget` for macOS computers

`wget` is a commonly-used command-line application for downloading URL-specified resources that also works when connections are poor

---

If you have a **Mac** with **`brew`** installed, you can install **`wget`** with the following command:

```
brew install wget
```

*Note: Generally, to install packages using brew, you use the format **`brew`** `install $PACKAGE_NAME`.*
*You can find more information on how to install and uninstall packages with **`brew`** on the Homebrew site.*

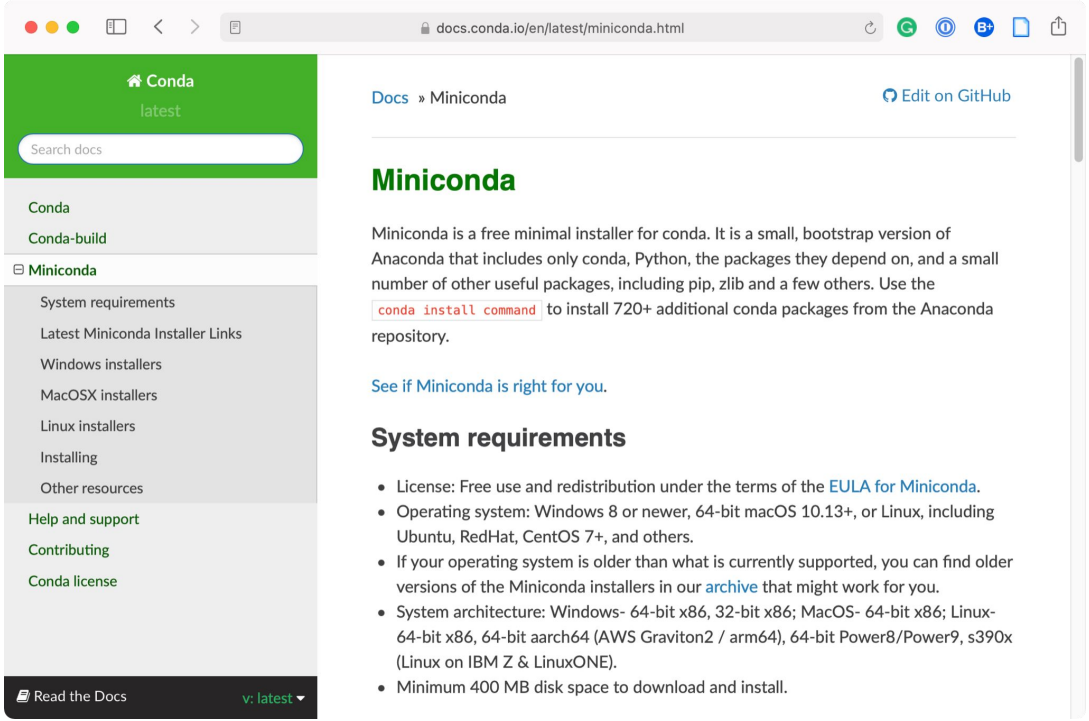
Using `miniconda3` to install the `conda` package manager

`conda` is an open-source, cross-platform, language-agnostic package manager that is commonly used with Python. We will be using **`conda`** to manage the installation of packages commonly used in data science and bioinformatics!

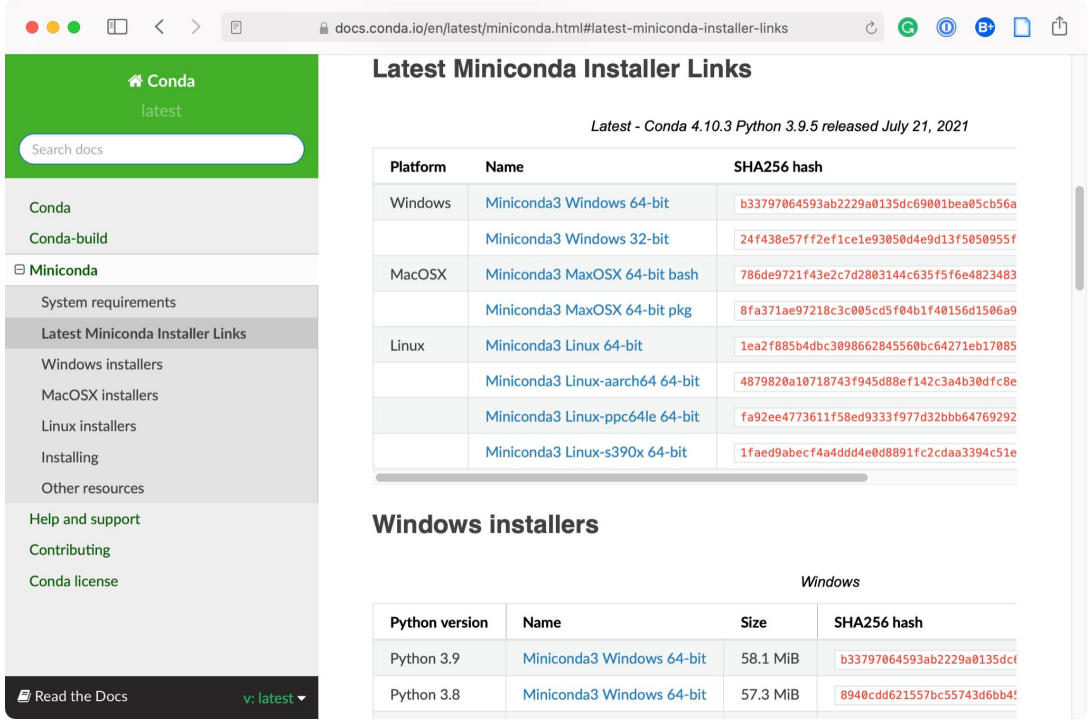
`miniconda3` is a minimal installer for **`conda`** that only contains the essential packages for **`conda`** to function--it's a lot faster and easier to use compared to other installers (e.g., **`anaconda3`**)

There are installation scripts for your specific operating system on the **`miniconda3`** site:

https://docs.conda.io/en/latest/miniconda.html#latest-miniconda-installer-links



The Miniconda website with information on the installer



Location of installers on the Miniconda website

Installing `conda` using `miniconda3` installation scripts, pt. 1

We will be installing **`conda`** by downloading the **`miniconda3`** installation scripts to your local computer with the **`wget`** command

1. Download the installation script to your computer with wget
   a. If you have a **Mac**:
      `$ wget` https://repo.anaconda.com/miniconda/Miniconda3-latest-MacOSX-x86_64.sh
   b. If you have a **Linux** system (64-bit systems):
      `$ wget` https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
2. Run the installation script on your computer in the directory where it downloaded to
   a. If you have a **Mac**: `$ bash Miniconda3-latest-MacOSX-x86_64.sh`
   b. If you have a **Linux** system (64-bit systems): `$ bash Miniconda3-latest-Linux-x86_64.sh`

```
-bash — mragsac@MFR.local:/Users/mragsac/Downloads
(module1a) mragsac@MFR:~$ pwd
/Users/mragsac
(module1a) mragsac@MFR:~$ cd Downloads/
(module1a) mragsac@MFR:~/Downloads$ wget https://repo.anaconda.com/miniconda/Miniconda3-latest-MacOSX-x86_64.sh
--2021-08-15 15:22:48--  https://repo.anaconda.com/miniconda/Miniconda3-latest-MacOSX-x86_64.sh
Resolving repo.anaconda.com (repo.anaconda.com)... 104.16.131.3, 104.16.130.3
Connecting to repo.anaconda.com (repo.anaconda.com)|104.16.131.3|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 44393084 (42M) [application/x-sh]
Saving to: ‘Miniconda3-latest-MacOSX-x86_64.sh’

Miniconda3-latest-MacOSX-x86_64.s  82%[=================================================>           ]  34.89M  68.2KB/s    in 3m 22s

2021-08-15 15:26:10 (177 KB/s) - Read error at byte 36583872/44393084 (Connection reset by peer). Retrying.

--2021-08-15 15:26:11--  (try: 2)  https://repo.anaconda.com/miniconda/Miniconda3-latest-MacOSX-x86_64.sh
Connecting to repo.anaconda.com (repo.anaconda.com)|104.16.131.3|:443... connected.
HTTP request sent, awaiting response... 206 Partial Content
Length: 44393084 (42M), 7809212 (7.4M) remaining [application/x-sh]
Saving to: ‘Miniconda3-latest-MacOSX-x86_64.sh’

Miniconda3-latest-MacOSX-x86_64.s 100%[++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++>]  42.34M   134KB/s    in 49s

2021-08-15 15:27:01 (156 KB/s) - ‘Miniconda3-latest-MacOSX-x86_64.sh’ saved [44393084/44393084]

(module1a) mragsac@MFR:~/Downloads$ bash Miniconda3-latest-MacOSX-x86_64.sh
```

Downloading the macOS installer with **`wget`** *and typing the installation command*

Installing `conda` using `miniconda3` installation scripts, pt. 2

3. Follow the instructions in the installation script
   a. You can quickly scroll through the terms and conditions by pressing **SPACE**
   b. Use the **default installation location** when prompted by pressing **ENTER**
   c. Opt-into **initializing conda** on the terminal when prompted by typing **YES**

4. Test if conda was properly installed and initialized on your terminal
   a. If you have a **Mac** or a **Linux** system: **$ conda**
   b. If **conda** successfully installed, then you should see the usage information for the package--this is a sampling of all of the commands that you can use conda for, including installing other packages

5. Congratulations! You have successfully installed **conda** on your local computer!

Luckily, you *should* have `git` pre-installed on your computer!

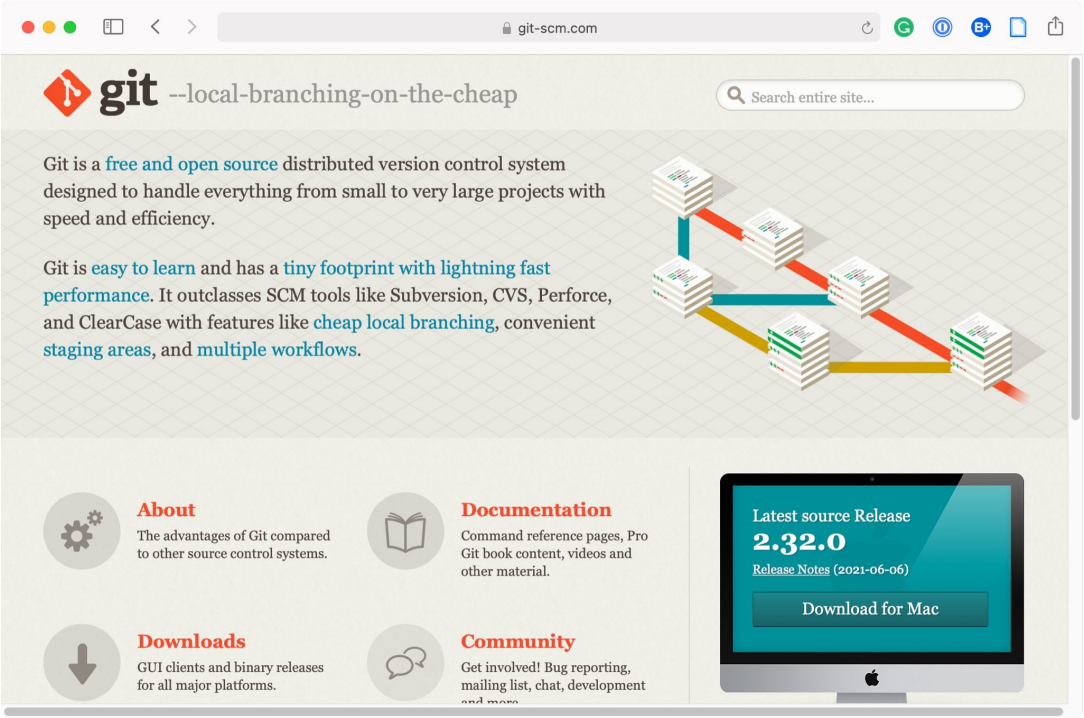
Version Control Systems are software tools that help record changes that are made to files by keeping a track of the modifications done to your code--similar to how Google Docs tracks changes made by different people!

`git` is a free and open-source version control system that is commonly used in software engineering, data science, and bioinformatics

You can check if `git` is installed on your computer with the following command:

```
$ git
```

If `git` *is* installed, then you should see the usage information for the package!
If `git` *is not* installed, you can install it with your operating system's package manager (e.g., `brew` on macOS).



The git website with learning resources on the homepage

Local Computer
Set-Up & Installation

Windows Computers

Unfortunately, there's quite a bit of set-up for Windows…

Many bioinformaticians use software environments that run GNU/Linux/UNIX-based environments. While macOS computers have this environment natively, Windows computers do not :-(

Luckily, Microsoft developed a subsystem for Windows computers that provide this functionality called the **Windows Subsystem for Linux (WSL)** (https://docs.microsoft.com/en-us/windows/wsl/about) !

We'll be spending some time to install **WSL** on your computer so that you can perform command-line operations in the same manner as your peers that use macOS or Linux computers! :-)

Installing the Windows Subsystem for Linux (WSL), pt. 1

We'll be installing quite a bit of software and configuring a lot of files, so make sure you *follow each step carefully* to make sure you properly set up your computer

1. Ensure that your computer is running the latest version of **Windows 10**
   a. **WSL2** (Windows Subsystem for Linux, Version 2) is *only* available in Windows 10, Version 2004, Build 19041 or higher

2. Enable the **WSL** Feature in Windows by configuring the Windows Features menu
   a. In the Start Menu search box, search for **Turn Windows Features On Or Off**
   b. A window will pop up with a list of folders with checkboxes next to them. Please scroll down and make sure that the following options are *enabled*:
      i. Virtual Machine Platform
      ii. Windows Hypervisor Platform
      iii. Windows Subsystem for Linux
   c. **RESTART** your computer to initialize the settings you selected

Installing the Windows Subsystem for Linux (WSL), pt. 2

3. Install the **Windows Terminal** from the Microsoft App Store
   a. You should be able to find the App Store from your Applications folder

4. Configure the **Windows Terminal** to use **WSL 2** as the default subsystem instead of the default **Powershell** environment
   a. Open the Windows Terminal and type in the following command to set the system default version of the Windows Subsystem for Linux to Version 2: `$ wsl --set-default-version 2`
   b. This command may ask you to install additional software to change the version of **WSL** from Version 1 to Version 2; please follow the instructions given carefully!

5. Install the **Ubuntu** application from the Microsoft App Store
   a. Once again, you should be able to find the App Store from your Applications folder
   b. After installing **Ubuntu**, it will prompt you to open the application. Open the application by pressing the **LAUNCH** button to initiate the installation of **Ubuntu** on your computer!

Installing the Windows Subsystem for Linux (WSL), pt. 3

6. Configure the **Ubuntu** application with your desired user credentials
   a. After installing **Ubuntu**, the installation screen will prompt you for a username and password to use with the Windows Subsystem for Linux; please enter a username and password that you will remember is associated with WSL!
   b. To protect your password, **nothing will appear on the screen** when you are typing it in; do not be alarmed and be sure you type in your desired password correctly! This sort of interface is common when typing in passwords on the command-line :-)

7. After configuring your **Ubuntu** user account, confirm that you are in the root directory of the Linux filesystem that we installed on your computer
   a. Type in the following command in the Ubuntu application: `$ pwd`
   b. The result of the command *should* read as `/home/$USERNAME`

Installing the Windows Subsystem for Linux (WSL), pt. 4

7. Verify that you are running **WSL 2** on your computer using **Command Prompt**
   a. Open the **Command Prompt** application by typing **Command Prompt** in the Windows Start Menu
   b. Run the following command to confirm that the version of WSL currently running is Version 2:
      ```
      $ wsl -l -v
      ```

8. Configure **Ubuntu** as the default subsystem to use when opening the **Windows Terminal** application through the terminal's settings menu
   a. Open the **Windows Terminal** application then open the Settings menu through the drop-down arrow
   b. For the default profile, select Ubuntu
   c. Save the settings by clicking **SAVE** then closing the **Windows Terminal** application window

9. Congratulations! You have successfully installed **WSL** on your Windows computer!
   a. Whenever you are given command-line inputs to run, remember to run the commands under **Ubuntu** by selecting the **Ubuntu** terminal from the **Windows Terminal** application drop-down arrow menu
   b. *DO NOT* use the Powershell option for the commands given to you!

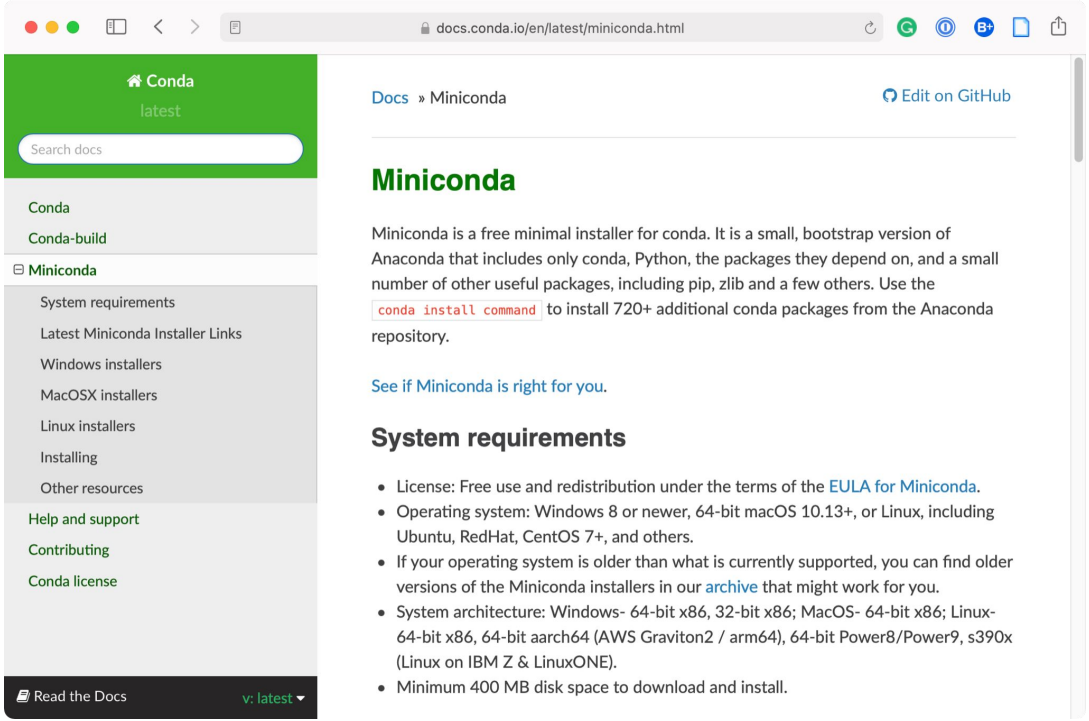
Using `miniconda3` to install the `conda` package manager

`conda` is an open-source, cross-platform, language-agnostic package manager that is commonly used with Python. We will be using **`conda`** to manage the installation of packages commonly used in data science and bioinformatics!

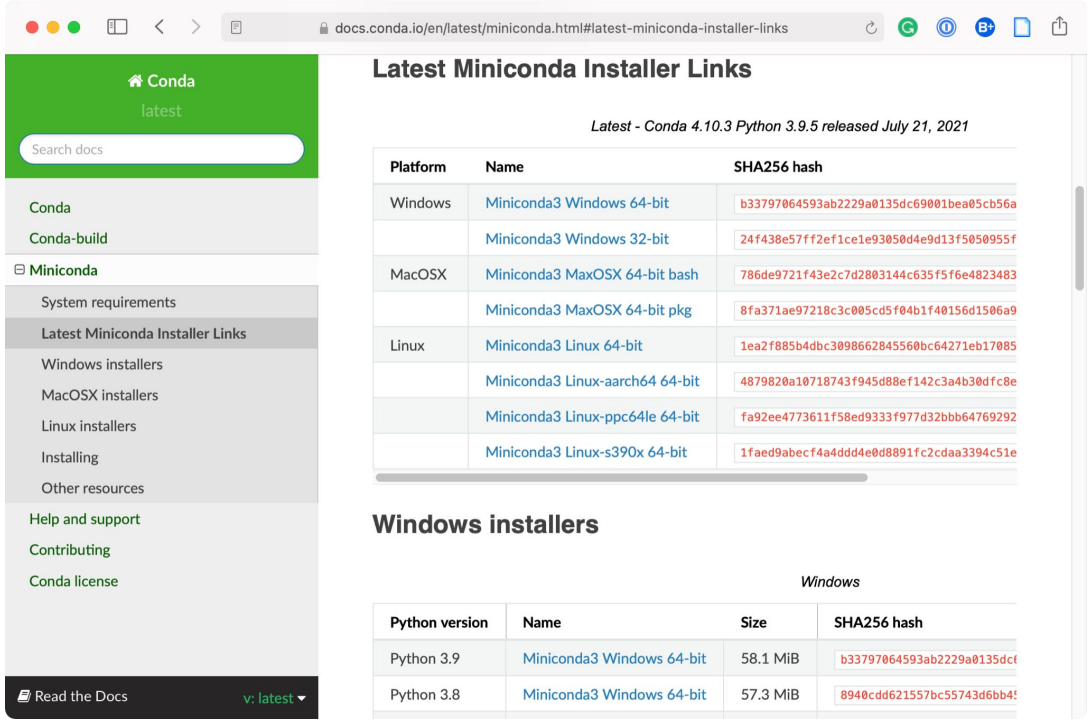
`miniconda3` is a minimal installer for **`conda`** that only contains the essential packages for **`conda`** to function--it's a lot faster and easier to use compared to other installers (e.g., **`anaconda3`**)

There are installation scripts for your specific operating system on the **`miniconda3`** site:

https://docs.conda.io/en/latest/miniconda.html#latest-miniconda-installer-links



The Miniconda website with information on the installer



Location of installers on the Miniconda website

Installing `conda` using `miniconda3` installation scripts, pt. 1

We will be installing **`conda`** by downloading the **`miniconda3`** installation scripts to your local computer with the **`wget`** command

1. Download the installation script to your computer with wget
   a. Since we have the Linux-based file subsystem, we can use the Linux installation link:
      `$ wget` https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
2. Run the installation script on your computer in the directory where it downloaded to
   a. `$ bash Miniconda3-latest-Linux-x86_64.sh`
3. Follow the instructions in the installation script
   a. You can quickly scroll through the terms and conditions by pressing **SPACE**
   b. Use the **default installation location** when prompted by pressing **ENTER**
   c. Opt-into **initializing conda** on the terminal when prompted by typing **YES**

Installing `conda` using `miniconda3` installation scripts, pt. 2

4. Test if conda was properly installed and initialized on your terminal
   a. **`$ conda`**
   b. If **`conda`** successfully installed, then you should see the usage information for the package--this is a sampling of all of the commands that you can use conda for, including installing other packages

5. Congratulations! You have successfully installed **`conda`** on your local computer!

Luckily, you *should* have `git` pre-installed on your computer!

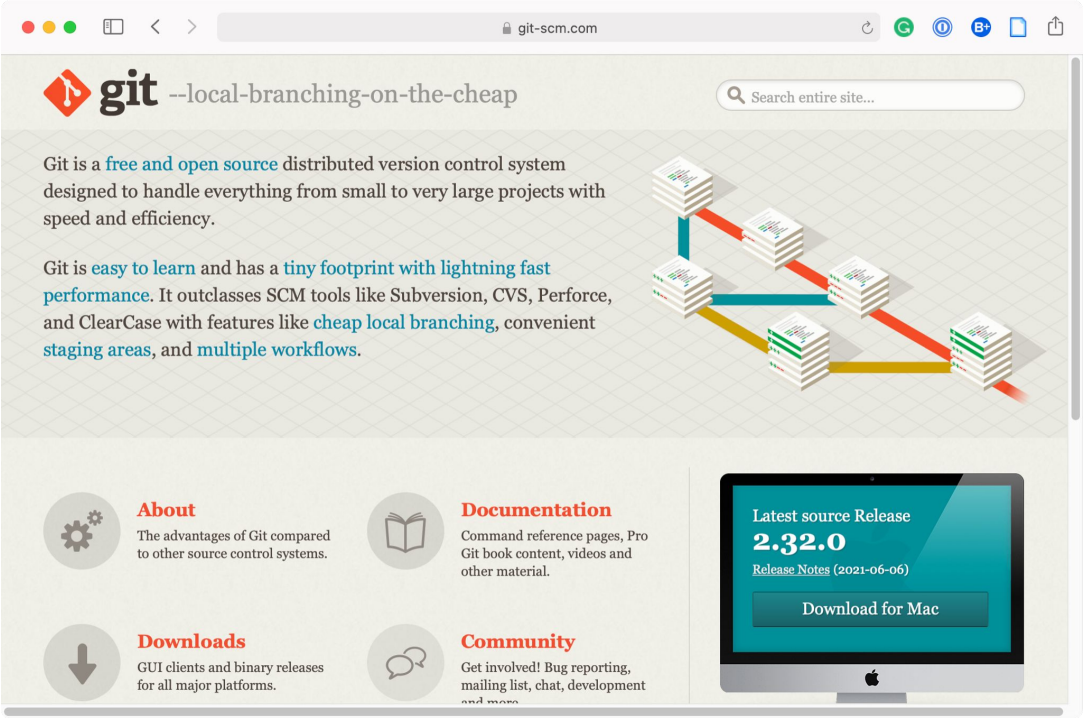
Version Control Systems are software tools that help record changes that are made to files by keeping a track of the modifications done to your code--similar to how Google Docs tracks changes made by different people!

`git` is a free and open-source version control system that is commonly used in software engineering, data science, and bioinformatics

You can check if `git` is installed on your computer with the following command:

```
$ git
```

If `git` *is* installed, then you should see the usage information for the package!
If `git` *is not* installed, you can install it with your operating system's package manager (e.g., `brew` on macOS).



The git website with learning resources on the homepage

I hope those installation resources helped!
Let’s get back to the main presentation :-)

Command-Line Basics

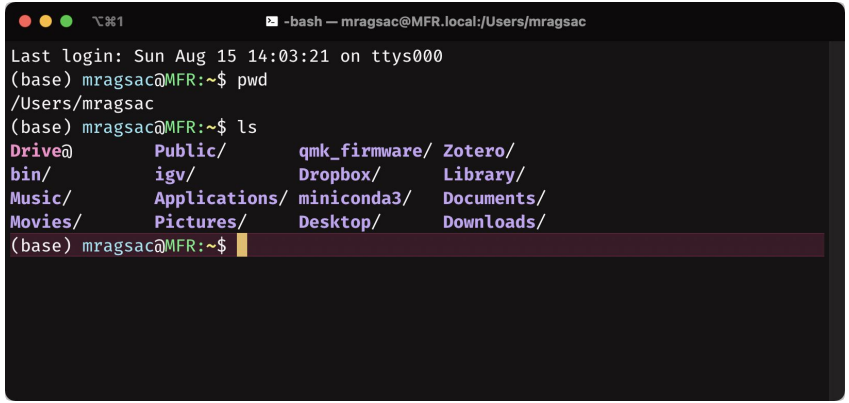
Navigating your computer with *only* text-based commands

What exactly are we looking at today?

The main things that we'll be configuring are a **terminal** application and a **package manager**.

What is a **Terminal** (or terminal emulator)?

- Versus a **graphical user interface (GUI)**, provides text-based access to an operating system, or **command-line interface**



What is a **Package Manager**?

- Collection of software tools that automate the process of installing, upgrading, configuring, and removing programs
- **`conda`** is an open-source, cross-platform, language-agnostic package manager that is commonly used with Python
- *Other notable package managers*:
  - **`brew`** - for macOS
  - **`CRAN`** - for R Packages
  - **`Bioconductor`** - for bio-related R Packages

Let’s practice using the terminal!

Open up your terminal application
to follow along with the class :-)

?

Does anybody have any questions after the demo?

Python Package Management

Compartmentalizing software environments with **conda** for reproducible analyses

What are modules and packages in Python?

While you can solve computational problems in Python without using any additional resources (i.e. “vanilla” Python), there are many additional software tools you can install to help make your life easier!

Any Python file can be considered a **module** that is named according to its filename: anything before the `.py` file extension is the module’s name

On the other hand, **packages** in Python are a *collection* of **modules**--so a collection of Python scripts that usually have a general function (e.g., `matplotlib` for plotting)

As mentioned previously, we’ll be using **`conda`** to install useful **packages** that are used in bioinformatics for data cleaning, data analysis & processing, as well as data visualization!

What are conda environments and when would I use them?

Here's a situation:

> Say you're working with a collaborator to analyze some biological data. They know a little about programming and want to try running your code on their system! But after sending them your scripts and other files, **they're not able to run your code on their computer, but you're able to run things on your computer!** What gives?!

Think of all of the packages required for a specific type of analysis as an individual piece of paper. **conda environments** are then a way to compartmentalize and organize those pieces of paper into a place that's easy to find and reference, like a binder!

You can then share the configuration for your **conda environment** with your collaborator!

Wait, so what did `miniconda3` end up installing?

`miniconda3` set up two things for you when you installed it: `conda`, the package management tool, and something called the `root` environment!

The `root` environment contains some version of Python, along with some basic packages to run simple things if you wanted to.

You can then create as many additional requirements as you want (and your computer allows) to be used for various cases:

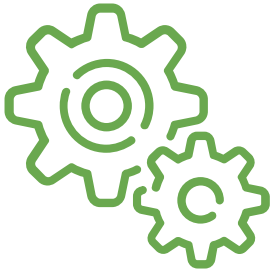
- Developing applications with different Python or package version requirements
- Using applications with different Python or package version requirements
- Collaborating with other developers or scientific collaborators
- Creating Python applications

Let’s practice using the terminal!

Open up your terminal application
to follow along with the class :-)

?

Does anybody have any questions after the demo?



Programming Basics

Brief introduction to basic concepts shared across most programming languages

Breaking down programming into 5 components

Like how we break up languages into basic structures (e.g., part of speech, verb tense, etc.), we can also deconstruct programming languages into five basic concepts:

① VARIABLES

② CONDITIONAL STATEMENTS

③ LOOPING & ITERATION

④ DATA TYPES & DATA STRUCTURES

⑤ FUNCTIONS

*I won't be going too deeply into these concepts in this presentation (this is a **very brief overview**), but they're some terms to look out for if you're reading books on how to code!*

Concept ① : Programming Variables

Variables are a symbolic name or reference to some sort of information.

EXAMPLE

Terry has 5 apples, **Cameron** has 4 apples, and **Michelle** has 3 apples.
Calculate the total number of apples they have.

terry	5
cameron	4
michelle	3

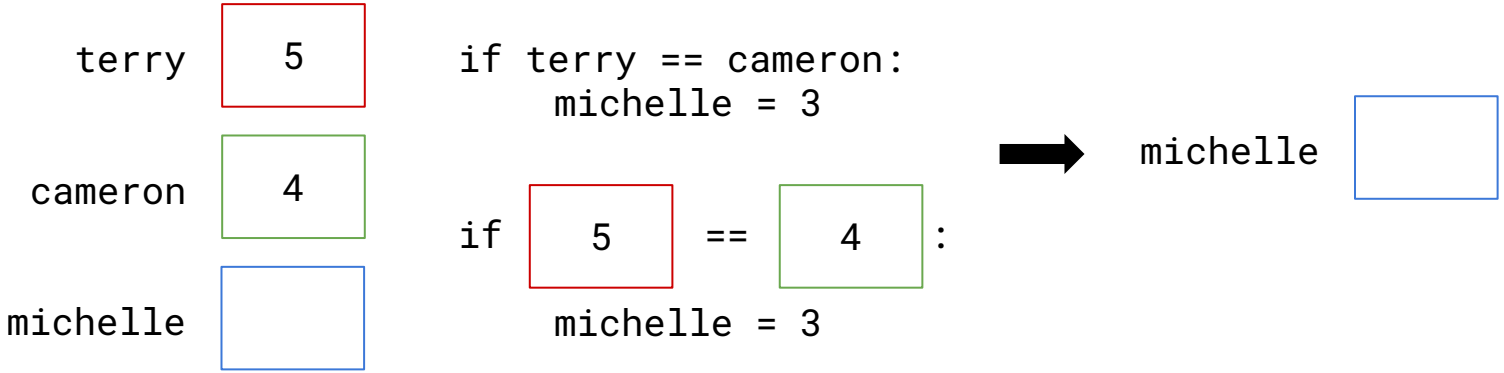
```
total_apples =  terry   + cameron + michelle
total_apples =  5 +  4 +  3
total_apples = 12
```

Concept ② : Conditional Statements

Conditional statements are expressions that determine if a variable is true or false.

EXAMPLE

Terry has 5 apples, **Cameron** has 4 apples.
If Terry and Cameron have the **same number of apples**, **then** give Michelle 3 apples.



Concept ③ : Looping & Iteration

An **iteration** is any time a program *repeats* a process or sequence.

Loops are a common type of iteration where a program repeats a process or sequence under certain conditions that are specified at the *beginning* of the loop.

EXAMPLE

```
for i in range(10):
    print(i)
```

for-loops continue
"FOR A SPECIFIED AMOUNT OF TIME"
(like a range of numbers)

```
i = 0
while i < 10:
    i += 1
    print(i)
```

while-loops continue
"WHILE A CONDITION IS MET"
(like while a variable meets a criteria)

Concept ④ : Data Types & Data Structures, pt. 1

Data Types classify the *type* of information a variable can represent and the types of operations that can be performed on the variable (e.g., **adding** two variables of type **`int`**)

Python Data Types

Name	Type	Description
Integers	int	Whole numbers, such as: 3 300 200
Floating point	float	Numbers with a decimal point: 2.3 4.6 100.0
Strings	str	Ordered sequence of characters: "hello" 'Sammy' "2000" "楽しい"
Lists	list	Ordered sequence of objects: [10,"hello",200.3]
Dictionaries	dict	Unordered Key:Value pairs: {"mykey" : "value" , "name" : "Frankie"}
Tuples	tup	Ordered immutable sequence of objects: (10,"hello",200.3)
Sets	set	Unordered collection of unique objects: {"a","b"}
Booleans	bool	Logical value indicating **True** or **False**

https://medium.com/@shawnren527/learn-about-python-3-data-types-numbers-and-strings-76c75a917c9b

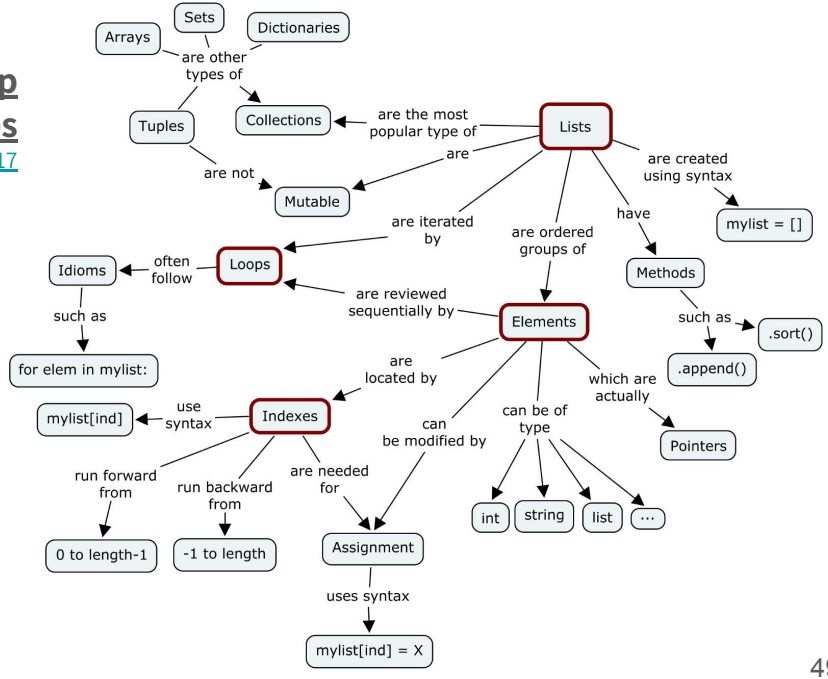
Concept ④ : Data Types & Data Structures, pt. 2

Data Structures are containers that hold data in a certain way.

Data Structures also have certain hard-coded rules to how you can manipulate them.

Python Lists and their Relationship to Other Data Structures

https://devopedia.org/python-data-structures#Mohan-2017



Some common data structures:

1. Arrays
2. Linked Lists
3. Hash Tables
4. Trees
5. Graphs

Concept ⑤ : Functions

Functions are self-contained modules of code that accomplish a particular task; once they're written, they can be called upon and reused within your code.

Generally, if you're performing an operation over and over again, it can be a function.

EXAMPLE

```
def add_numbers(numberA,numberB):
    result = numberA + numberB
    return result
```

➡

```
add_numbers(1,1)
add_numbers(1,5)
add_numbers(1,7)
add_numbers(8,7)
```

Practice Problems

Programming in Python to solve bioinformatics-inspired problems

Let's apply our basic programming concepts!

Now that we know the basics of programming, let's tackle some bioinformatics problems!

Problem ①: Determining Fragments after a Restriction Enzyme Digest

We have a DNA plasmid that we're using for a transformation, but we want to check that our restriction enzymes will cut the fragment *only* at the places we've specified. How can we use Python programming to check what the restriction digest should look like?

Problem ②: Evaluating a FASTQ File for Unique Sequence

We have a FASTQ file of sequencing reads from a MPRA experiment. We want to determine the set of unique sequences that appear in the file and how many times they appear.
How can we use Python programming to gather the set of unique sequences from a file?