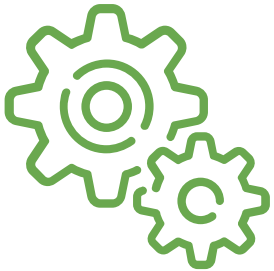
Bench to Terminal: Getting Set Up for Bioinformatics Work!

Michelle Franc Ragsac
3rd Year BISB | PI: Emma Farley, PhD
mragsac@eng.ucsd.edu



What's the point of this module?

1. **Help you set-up your computer for accessing bioinformatics tools**
   We'll be installing things *together* and configuring our computers for local programming and accessing the supercomputer cluster

2. **Provide a safe space to get set-up and configuration questions answered**
   Getting a computer set up is daunting, especially if you haven't done it before

3. **Learn about basic programming concepts that you might encounter**
   I'll be going over some basic terminology and pointing out helpful resources that you can refer to in the future if you're ever stuck

How is this module organized?

SETUP

We'll start off by installing some things and briefly learning about them!

DEMO

After (or while) getting through installations, we'll also demo some of the things we've installed!

CONCEPTS

If there are things that take a while to install, we'll simultaneously go over programming concepts.

Set-up & Installation

Within this module, we'll be installing some common things you might encounter for Bioinformatics work during your rotation and/or research!

However, this is not a comprehensive guide of everything you might need…

What are we installing and/or configuring?

1. Terminal Application
   a. Accessing your ***local*** file system using the terminal on your local computer
   b. Accessing a ***remote*** file system (e.g. cluster) using the terminal on your local computer
2. **`conda`** Package Manager via **`Miniconda`**
3. **`Bioconda`** Channel
4. Several commonly-used Python Packages
   a. **`jupyterlab`**
   b. **`numpy`**
   c. **`pandas`**
   d. **`scikit-learn`**
   e. **`matplotlib`**
5. R Language Jupyter Notebook environment

What is the operating system (OS) of your computer?

macOS? Windows?
Ubuntu or other Linux distribution?

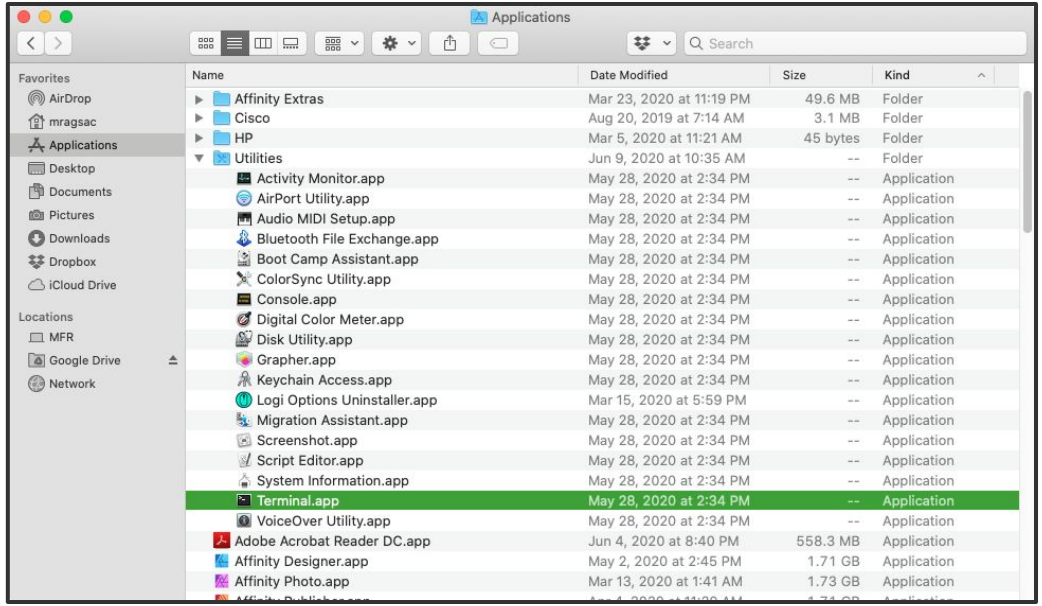
Depending on what OS you have, the instructions for installing software will be different from other OS'.

Installing a Terminal Application on macOS

For people using **Apple** computers (macOS), you already have a built-in terminal application on your computer that works super well!

It's called the **Terminal** application.

Linux users also have a pre-installed Terminal application.



`Applications > Utilities > Terminal.app`

Installing a Terminal Application on Windows 10

In one of the largest updates to Windows 10, Microsoft included a full Ubuntu command line that is able to run natively within the Windows OS that mimics a UNIX-based shell!

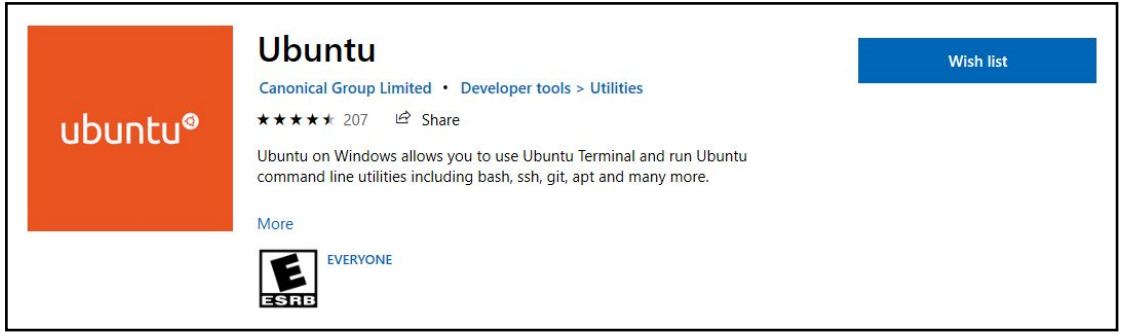
However, it requires some set-up…

What applications are we installing?

1. `Ubuntu` application from the Microsoft Store
2. `Windows Terminal` application from the Microsoft Store
3. Michelle's handy-dandy, pre-written `settings.json` file

Windows 10 Terminal Installation:
Part 1, Ubuntu Installation

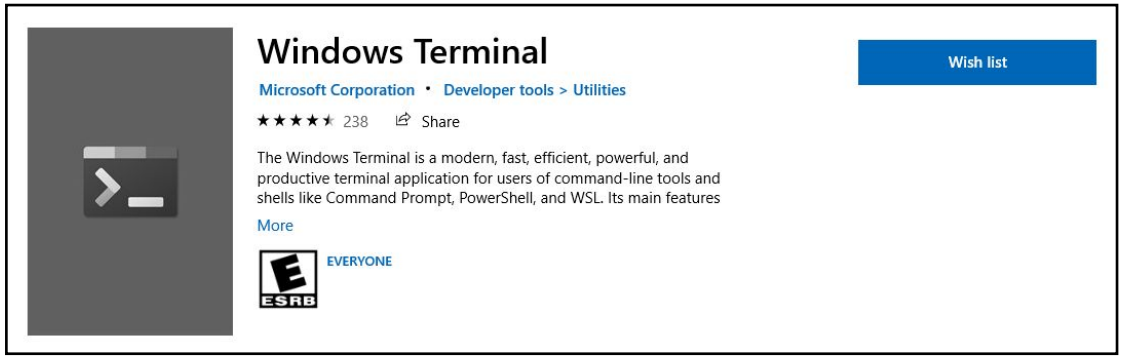
Search for "Ubuntu" on the Microsoft Store on your computer.
The application you search for should have the following information card:



This application will give us the ability to access the Windows file system using UNIX-based commands (since Ubuntu is a Linux-based OS); this program also installs some basic command-line software that we'll need to access a compute cluster

Windows 10 Terminal Installation:
Part 2, `Windows Terminal` Installation

Search for "`Windows Terminal`" on the Microsoft Store on your computer. The application you search for should have the following information card:



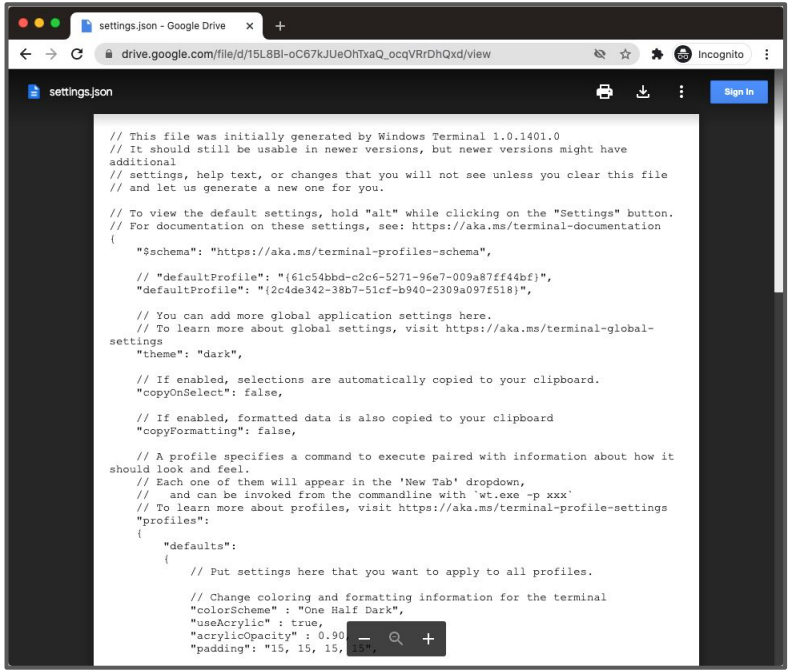
This application will give us a terminal environment that mirrors the terminal you'd see on a macOS or Linux (i.e., native UNIX-based operating systems) computer

Windows 10 Terminal Installation: Part 3a, `settings.json` Download

Download the `settings.json` file I configured at the following link: settings.json.

This file contains configuration information for:

1. *the Windows Terminal application to add to the Ubuntu application you downloaded,*
2. *make it the default shell for browsing the Windows file system, and then*
3. *make the terminal interface aesthetically pleasing*



Windows 10 Terminal Installation:
Step 3b, `settings.json` Configuration

- Open the `settings.json` file you just downloaded with the `Notepad` application on your computer
  - Or any plain text editor (e.g., Sublime Text, Notepad++)
- **[COPY]** **all of the contents** from the downloaded file to your clipboard
- Open the `Windows Terminal` application that you just installed
- Open the `Windows Terminal` "Settings" item
  - You can access the settings by pressing the ∨ symbol next to the + symbol in the tab bar
- **[PASTE]** **all of the contents** from your clipboard to overwrite the "Settings" file for the `Windows Terminal` application
- **[SAVE]** the "Settings" file for the `Windows Terminal` application
- Exit the `Windows Terminal` application

With the terminal configured, we can finally feel like programmers!

The terminal is a text-based way of interacting with your computer (alternative to using a “Graphic User Interface”, or GUI)!

However, we have to learn the language to navigate the terminal.

What is a command-line interface?

- **Commands** are directives given to a computer to perform specific tasks
- In the previous section, we installed (or discovered) the **Terminal**, which is a **command-line interface** to process commands given to your computer

Next, we'll briefly go over some basic UNIX commands!

Additional Resources:

- *Introduction to the Command-Line Interface*
  https://www.codecademy.com/learn/learn-the-command-line
  https://carlalexander.ca/introduction-command-line-interface/
- *UNIX Command Line Cheat Sheets*
  https://files.fosswire.com/2007/08/fwunixref.pdf
  http://cheatsheetworld.com/programming/unix-linux-cheat-sheet/

Demo Time!

Make sure your Terminal Application is open!

We'll be going through some UNIX commands before installing the next set of items!

What is a package?

You can think of packages as a collection of computer programs designed to perform a specific task.

For example, Python has packages that are used for generating plots, accessing web pages, or perform math calculations!

What is a package manager?

Package managers are a collection of software tools that automate installing, upgrading, configuring, and removing computer programs!

We'll be installing the **conda** package manager, as its popular for installing bioinformatics tools, Python packages, and even R packages!

Installing `conda` via `Miniconda`

- `conda` is an open-source package management and environment management system that runs on Windows, macOS, and Linux
- `miniconda` is a minimal installer for `conda` that only includes the essentials
  - To install `miniconda`, first run the following command to download the installer:

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
```

  - Then run the installer with the following command:

```
bash Miniconda3-latest-Linux-x86_64.sh
```

Additional Resources:

🔗 *`conda` User Guide* https://docs.conda.io/projects/conda/en/latest/#

🔗 *`Miniconda` Installer for `conda`* https://docs.conda.io/en/latest/miniconda.html

What are conda channels?

Channels are locations where packages are hosted online for you to install.

By default, **conda** installs packages from a default list of channels, but we can modify this default list to include additional channels.

Configuring the Bioconda Channel

- Bioconda is a channel for conda that specializes in bioinformatics software!
  - https://anaconda.org/bioconda/
- We can configure our installation to access the channel with the commands:

```
conda config --add channels defaults
conda config --add channels bioconda
conda config --add channels conda-forge
```

- Now we can install bioinformatics software using conda!

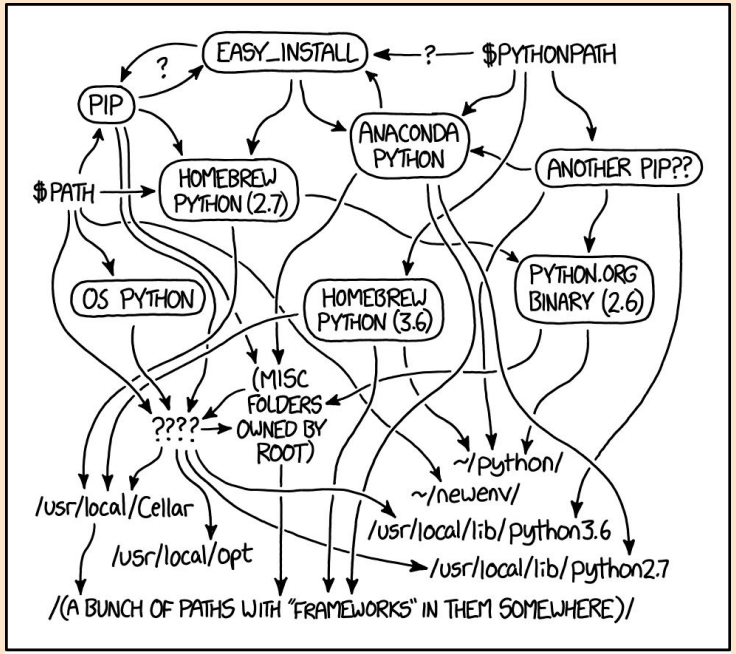
Additional Resources:

🔗 *Bioconda Information Page* https://bioconda.github.io/

What are conda environments?

A **conda** environment is a directory that contains a specific collection of **conda** packages that you have installed.

We'll be creating a **conda** environment for this module, then installing several commonly-used bioinformatics Python packages that we'll play around with after!



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

https://xkcd.com/1987/

Why are conda environments important?

A **conda** environment can organize what *exact* packages were used for a specific analysis.

You can make debugging easier by encapsulating specific analyses into **conda** environments, while making your code more reproducible!

Basics on Configuring conda Environments

How do I create a conda environment?

```
conda create --name $ENVIRONMENT_NAME_GOES_HERE
```

How do I change into the new environment I've created?

```
conda activate $ENVIRONMENT_NAME_GOES_HERE
```

And how do I get out of the environment?

```
conda deactivate
```

How do I see what packages are in my environment?

```
conda list
```

How do I install packages with conda into my environment?

```
conda install $PACKAGE_NAME_GOES_HERE
```

How do I delete an environment?

```
conda env remove --name $ENVIRONMENT_NAME_GOES_HERE
```

Demo Time!

Make sure your Terminal Application is open!

We'll be going through some commands to set up a new channel for this module and install some packages!

DEMO

Creating conda Environments:

Part 1, Creating a Python Environment for this Module

We'll be creating an environment with some commonly-used data analysis packages, and then practicing exporting it as a yml file!

Run the following command to create a new environment called **`module1a`**, and after it's created, activate the environment:

```
$ conda create --name module1a
$ conda activate module1a
```

DEMO

Creating conda Environments:

Part 2, Installing Commonly-Used Analysis Packages into our New Environment

With our new environment activated, we can now install some analysis packages in Python that people use:

```
(module1a) $ conda install -c anaconda numpy pandas matplotlib scikit-learn
```

- `NumPy` adds support for large, multi-dimensional arrays and optimized linear algebra in Python
- `pandas` adds support for Excel-like table operations in Python (i.e., R Data Frames)
- `Matplotlib` is a Python plotting library
- `scikit-learn` is a Python library used for machine learning

```
(module1a) $ conda install -c conda-forge jupyterlab
```

- `JupyterLab` is a web-based user interface for Jupyter Notebooks

While waiting for things to install, let's go over...

Essential Programming Concepts

The Backbone of Programming Languages

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: **Variables**

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

EXAMPLE

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

`owen`	5
`cameron`	4
`michelle`	3

```
total_apples =  owen   + cameron + michelle
```

```
total_apples =  5  +  4  +  3
```

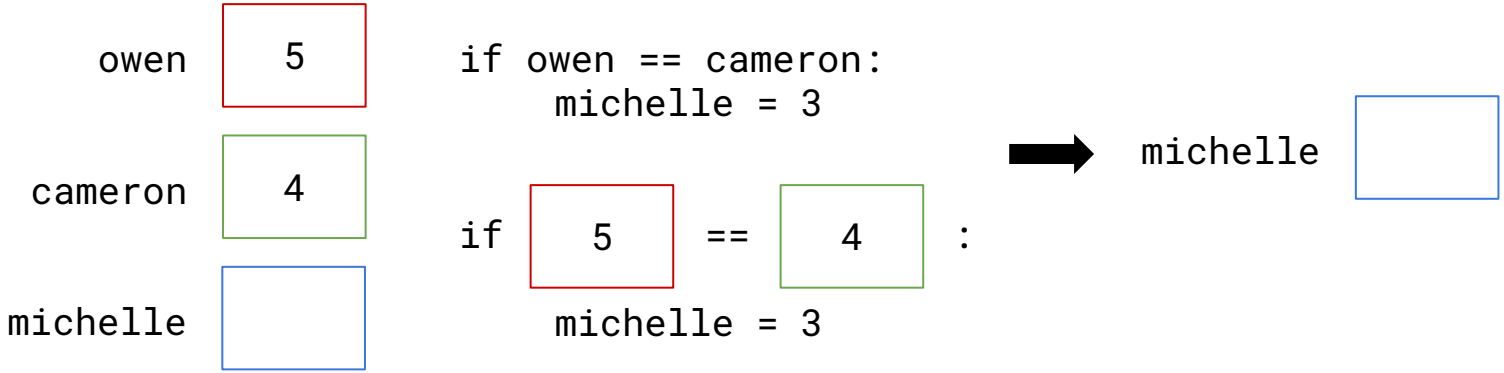
```
total_apples = 12
```

Concept 2: **Conditional Statements**

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

EXAMPLE

Owen has 5 apples, **Cameron** has 4 apples.
If Owen and Cameron have the same number of apples, 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.

EXAMPLE

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

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

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

while-loops continue
"WHILE A CERTAIN CONDITION IS MET"
(like while a certain variable meets a criteria in this example)

Concept 4a: **Data Types & Data Structures**

Data types classify what type of information a variable can hold and what other manipulations can be done with it.

Python Data Types

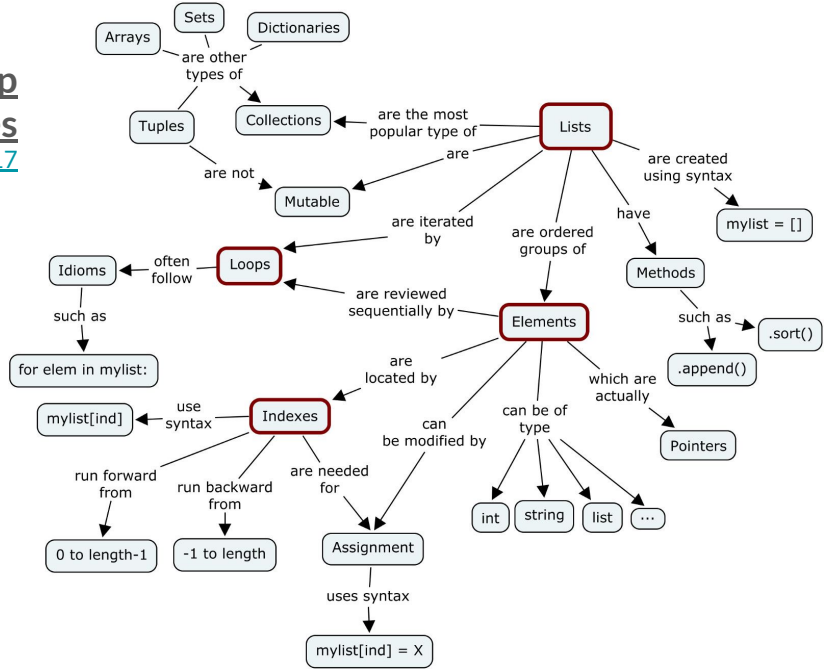
Name	Type	Description
Integers	int	Whole numbers, such as: 3 300 200
Floating point	float	Numbers with a decimal point: 2.3 4.6 100.0
Strings	str	Ordered sequence of characters: "hello" 'Sammy' "2000" "楽しい"
Lists	list	Ordered sequence of objects: [10,"hello",200.3]
Dictionaries	dict	Unordered Key:Value 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 4b: **Data Types & Data Structures**

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

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.

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)
```

Some Key Resources that Helped Me, Part 1

Online Forums & Q/A Websites

- **Do you have general programming questions?**
  Somebody's probably already solved the problem!
  - StackOverflow: https://stackoverflow.com/
    - Yahoo! Answers, but for programming questions

- **Do you have bioinformatics questions?**
  There's a discussion board for that!
  - BioStars: https://www.biostars.org/
    - Online forum for people doing bioinformatics analyses

Some Key Resources that Helped Me, Part 2

Learning to Program

- **Do you want to learn how to program in Python/R/Java/C++/etc.?**
  - The best way that I found for learning to program was to **do projects or solve problems with the language you want to learn**!
    - ROSALIND: http://rosalind.info/problems/locations/
      - Has bioinformatics-oriented problem sets that can be done in any language
    - **@BISB/BMI Students**, you'll be assigned questions from the **Bioinformatics Textbook Track** for the course requirement **Bioinformatics II: Introduction to Bioinformatics Algorithms (BENG 202/CSE 282)** taught in winter
- **What if I need a general resource, and not problem sets?**
  - CodeAcademy: https://www.codecademy.com/
  - freeCodeCamp: https://www.freecodecamp.org/
  - DataQuest: https://www.dataquest.io/
  - *Towards Data Science* Medium Blog: https://towardsdatascience.com/

Essential Programming Concepts

(end)

DEMO

Sharing conda Environments

- Sometimes, you may want to share your environment with somebody else so they can re-run your code in the same programming environment you developed on
- You can export your environment to an `environment.yml` file to share

Exporting the Newly-Created `module1a` Environment as a YML File

1. Activate the environment
2. Export your active environment to a new file
3. Email or copy the exported file to the other person

```
conda activate module1a
(module1a) $ conda env export > module1a.yml
```

Creating an Environment from the `module1a` YML File (not running this, but included as an example)

Create the environment from the file
(*where the first line of the yml file sets the env name*)

```
conda env create -f module1a.yml
```

DEMO

Creating conda Environments:

BONUS, Creating R Language Jupyter Notebooks

Conda also has support for R Language packages and Jupyter Notebooks (versus using the R Studio interface). We can set up a new R environment called **module1a_R** with the following commands:

```
# create the new R environment with conda then activate it
$ conda create --name module1a_R
$ conda activate module1a_R

# install the jupyterlab environment, then
# install all of the essential R packages
(module1a_R) $ conda install -c conda-forge jupyterlab
(module1a_R) $ conda install r-essentials r-base
```

How do people use supercomputer clusters?

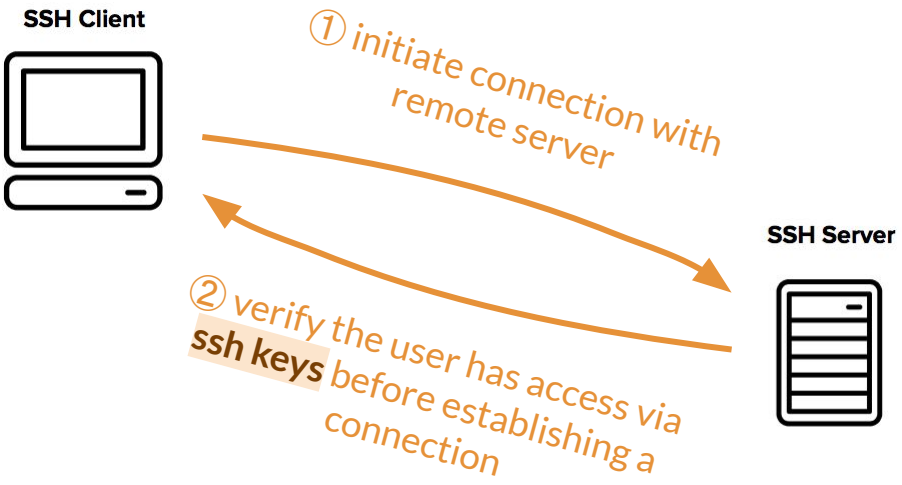
Using the Secure Shell Protocol (SSH) to log-in to Remote Systems

We'll be generating ssh keys to access some training accounts on the Triton Shared Compute Cluster!

What is Secure Shell Protocol (SSH)?

The **SSH protocol** encrypts a secure connection between a client and host, and is most commonly used for:

1. Providing secure access for users and automated processes
2. Interactive and automated file transfers
3. Issuing remote commands
4. Managing network infrastructure



What are SSH Keys? How do I use them?

To access the Triton Shared Compute Cluster, we'll be using **SSH Keys**

your computer

① GENERATE SSH KEYS

generates two files in a hidden `~/.ssh/` folder:

public key `id_rsa.pub`

a lock you can place in places you want to access from your local computer

private key `id_rsa`

*a **master key** you can use to access places where your public key (lock) is located*

do not share, keep this file on your computer

② SHARE PUBLIC KEY WITH *REMOTE* ACCOUNTS YOU WANT TO ACCESS

SSH Server

Demo Time!

Make sure your Terminal Application is open!

We'll be making sure that you can log into your TSCC training account for our HPC module!

DEMO

Generating SSH Key Pairs

- SSH Keys are relatively easy to generate, and you only have to generate them once for a computer!
  - Then the public key can be used with any account you want to access with that computer

To generate SSH Keys, follow the command below:

```
ssh-keygen
```

Example of Generating SSH Key Pairs

```
klar (11:39) ~>ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ylo/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ylo/.ssh/id_rsa.
Your public key has been saved in /home/ylo/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:Up6KjbnEV4Hgfo75YM393QdQsK3Z0aTNBz0DoirrW+c ylo@klar
The key's randomart image is:
+---[RSA 2048]----+
|     .      ..oo..|
|   . . .  . .o.X.|
|    . . o.  ..+ B|
|   .   o.o  .+ ..|
|    ..o.S   o..  |
|   . %o=      .  |
|    @.B...     . |
|   o.=. o. . .  .|
|    .oo  E. . .. |
+----[SHA256]-----+
klar (11:40) ~>
```

https://www.ssh.com/ssh/keygen/

DEMO

Setting Up Your TSCC Training Account

- The Triton Shared Compute Cluster (TSCC) is one of the high-performance computing (HPC) clusters that labs on campus analyze their data on
  - We've gotten training accounts for bootcamp so you all can get some experience with accessing the cluster and running programs on it before you go into your rotations :-)

- To access your training account, please **email me your public key file** that you generated using the ssh-keygen command!
  - The public and private keys can be found in the **hidden** folder `~/.ssh/`
  - The public key file usually appears as `id_rsa.pub`
    (where `id_rsa` is your private key, and the `.pub` dicates that the file is your public key)
  - You can email me at mragsac@eng.ucsd.edu :-)

Thanks for listening! :-)

Any questions?