Vanderbilt - Starting Grad School Documentation

Release 0.1

Victor Calderon

Apr 11, 2019

Contents:

In order to facilitate your time as a graduate student, here are some suggestions and tips to make your time in graduate school much easier.

In case you have any questions, please contact me via victor.calderon@vanderbilt.edu

CHAPTER 1

MAC 101: An Introduction

This is an introduction to the things that you should know about your new Mac.

In this section we discuss how to properly setup your Mac, how to SSH onto a remote computer, and more.

Table of Contents

1.1 What your MAC should have

Your MAC already comes with pre-installed packages, but some of the packages that you may need are not *entirely* ready to be used. The first packages that you may need to install are the following:

1.1.1 Xcode

Xcode is an editor that you can use to edit and run your scripts.

Note: In my opinion, Sublime Text is a much better editor than Xcode is, because it has a lot more integrations and feautures than Xcode does.

After having downloaded and installed Xcode, there are a few commands that need to be run in order to let other programs run, i.e. Xcode Command-line Tools.

You should run the following commands from the Terminal:

and verify that you've installed Xcode Command Line Tools successfully:

Just to be certain, verify that **gcc** is installed:

```
$ gcc --version
gcc-7 (Homebrew GCC 7.2.0) 7.2.0
Copyright (C) 2017 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

For more information, read this to know more about **Xcode Command Line Tools**.

1.1.2 LaTeX

Another important package that you need to install whenever you get a new MAC is **LaTeX**. LaTeX is a *document preparation system* that lets you present your work or documents in a presentable format. LaTeX is also used a lot by scientific journals, i.e. Astrophysical Journal, Monthly Notices of the Royal Astronomical Society.

To download LaTeX on a MAC, you need to download **MacTEX**. This can be found at https://www.tug.org/mactex/.

After downloading the necessary `.pkg` file, you will be able to execute and run LaTeX on your computer.

For a tutorial on how to use LaTeX, go to the links in *LaTeX*.

1.1.3 Vanderbilt VPN

Sometimes you will need to access remote servers from outside Vanderbilt. To do this you will need to set up a **VPN** connections. In order to do this, you will need to install the Pulse Secure VPN client. This will will let you connect remotely to servers hosted at Vanderbilt .

1.1.4 MAC Time Machine & Backup

When you get a MAC, you have the option to set up Time Machine. Time Machine lets you back up your entire computer, so that you don't loose any important file.

I **strongly** suggest buying a hard drive (>1TB) to back up all of your files. I find the hard drives from *Western Digital* quite useful. In case you're interested, see here.

Note: It is **extremely** important that you back up your computer at least a few times a week. If not, you may end up loosing a substantial amount of files (and your work!) if your computer fails. So this should be one of the **first things** that you do when having a Mac.

1.2 SSH

For most of the research being done in graduate school, one needs to access a remote computer that have more processors, more disk space than your computer. For this, you can **SSH** onto a remote computer via the terminal.

In order to do that, you first need to do the following in the terminal

- **SSH folder**

```
$ cd $HOME
$ mkdir .ssh
$ chmod 700 .ssh
```

- **SSH Configuration file**

```
$ cd ~/.ssh
$ touch config
$ chmod 600 config
```

- **Authorized_keys file**

```
$ cd ~/.ssh
$ touch authorized_keys
$ chmod 700 authorized_keys
```

- **Connections Folder**

```
cd ~/.ssh
mkdir connections
chmod 700 connections
```

- **SSH-Keys Folder**

```
cd ~/.ssh
mkdir ssh_keys
chmod 700 ssh_keys
```

- **Public Keys Folder**

```
cd ~/.ssh
mkdir pub_keys
chmod 700 pub_keys
```

At this point, your `~/.ssh` folder should look like this:

```
$ ls -lah ~/.ssh

drwx------    8 user  staff   256B Jan 21 18:37 ./
drwxr-xr-x@ 161 user  staff   5.0K Jan 21 20:24 ../
-rw-------@   9 user  staff   288B Jan 21 18:37 authorized_keys
-rw-------@   1 user  staff   1.4K Jan 21 19:03 config
drwx------    2 user  staff    64B Jan 22 16:37 connections/
drwx------    2 user  staff    64B Jan 22 16:37 pub_keys/
drwx------    2 user  staff    64B Jan 22 16:37 ssh_keys/
```

Now you can access a remote computer by logging in from the terminal:

```
ssh username@123456.server.io
```

The tedious thing about this is that it will prompt you for your password anytime that you want to access the remote server.

This can be solved by using `SSH keys`.

1.2.1 SSH-Keys

Some servers are configured to accept encryption keys in addition to (or instead of) requiring a password. This can be more secure since the account cannot be compromised by someone guessing passwords!

SSH keys are comprised of a **public** and a **private** key. The public key can be given to anyone (hence the name). If you connect to a server that has your public key and you can provide your private key, it will let you in. (Consequently, if your private key is **stolen**, someone else can log into your account!)

In order to **generate SSH keys**, you need to run the following:

```
$ cd ~/.ssh
$ ssh-keygen -t rsa -b 4096
$ Generating public/private rsa key pair.
Enter file in which to save the key (/Users/calder/.ssh/id_rsa): id_rsa_4096
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
$ ls
id_rsa_4096
id_rsa_4096.pub
$ chmod 600 id_rsa*
$ mv id_rsa_4096 ssh_keys/
$ mv id_rsa_4096.pub pub_keys
```

Now you can add your **SSH-Keys** by typing the following:

```
ssh-add -K ~/.ssh/ssh_keys/*
```

Note: The argument `-K` in `ssh-add` for adding the key to your **Keychain** if you are on a MacOSX system. If not, just have the command `ssh-add ~/.ssh/ssh_keys/*` to add all of the SSH-KEYS that you have created.

From now on, you should add the **private** keys and their respective **public** keys to the `ssh_keys` and `pub_keys` folders, and then run the commands `chmod 600 key` and `chmod 600 key.pub` command, replacing `key` with the name of the actual SSH-key.

Note: If you enter a passpharase, you will need to type that password every time you use the ssh keys (e.g. when connecting to a server). It's common to not create a password, but know that if the private key is lost, anyone can use them. (But they would have to know which server to connect to, which "config" file will provide!)

1.2.2 SSH Config file

This file acts as the file with predefined options for how you connect to numerous SSH servers.

After having created the `config` file in the `~/.ssh` directory, you must add the information to each of the servers that you connect to.

First, you must execute

```
open ~/.ssh/config
```

in order to open the `~/.ssh/config` file. After having opened the file, you can add **global** settings for how each SSH sessions executes. Add these lines to your `config` file:

```
Host *
ControlMaster auto
ControlPath ~/.ssh/connections/%C
ControlPersist 1m
ServerAliveInterval 30
ServerAliveCountMax 10
```

If you're on a **MAC** and would like to use X11 as well, add **these extra lines** beneath `ServerAliveCountMax`:

```
XAuthLocation /opt/X11/bin/xauth
AddKeysToAgent yes
UseKeychain yes
```

This will ensure that your connections don't die, forward X11, and save those keys to your **Keychain** (if applicable).

Connecting to Github

Once you have your `~/.ssh/config` file setup, you can add your **Github** information to it.

You would just need to add this below the code from above:

```
Host github.com
HostName github.com
User git
IdentityFile ~/.ssh/ssh_keys/github_key
IdentitiesOnly yes
PreferredAuthentications publickey
```

This will make **git** to use the public key `github_key`, which you should have created already. If not, follow these instructions here.

Forwarding X11 (MACs only)

If you happen to plot on remote servers, you might want to use XQuarts (X11) if you're on a Mac in order to plot. If so, you will need to add the following line to the `~/.ssh/config` file below the **Host** information for the server.

```
ForwardX11 yes
```

And make sure that the `XAuthLocation` setting is pointing to the correct path of `xauth`. This will guarantee that you don't a problem with rerouting your plots to X11. For more information, see XQuartz.

Your `~/.ssh/config` file should look something like this now:

```
Host *
ControlMaster auto
ControlPath ~/.ssh/connections/%C
ControlPersist 1m
ServerAliveInterval 30
ServerAliveCountMax 10
```

(continues on next page)

(continued from previous page)

```
XAuthLocation /opt/X11/bin/xauth
AddKeysToAgent yes
UseKeychain yes

## Connects to Github
Host github.com
HostName github.com
User git
IdentityFile ~/.ssh/ssh_keys/github_key
IdentitiesOnly yes
PreferredAuthentications publickey

## Connects to a remote Server via SSH
Host server_name
HostName path.to.server
User username
IdentityFile ~/.ssh/ssh_keys/server_key
IdentitiesOnly yes
PreferredAuthentications publickey
ForwardX11 yes
```

where `server` is the name of the *server* to which you want to connect, and `path.to.server` is the URL to the server. This will use the `~/.ssh/ssh_keys/server_key` SSH key to access the server with your credentials for username `username`.

CHAPTER 2

Python 101 - Anaconda, environments, and more

This is a small introduction of what *I* think you should have installed in order to property use python in your projects.

Table of Contents

2.1 Anaconda

The very first thing for you to have is Anaconda installed. From their website:

> With over 4.5 million users, Anaconda is the world's most popular Python data science platform. Anaconda, Inc. continues to lead open source projects like Anaconda, NumPy and SciPy that form the foundation of modern data science. Anaconda's flagship product, Anaconda Enterprise, allows organizations to secure, govern, scale and extend Anaconda to deliver actionable insights that drive businesses and industries forward.

Having **Anaconda** installed on your computer is important, since it allows you to not worry about installing missing dependencies, creates environments for you projects, etc.

2.1.1 Installing Anaconda

The first thing is to download Anaconda. If you're starting with Python from scratch, it is better to start with **Python 3**.

You can download Anaconda from here: https://www.anaconda.com/download, and make sure to download the Python 3 version.

Note: If you're downloading it from the terminal, you can download the executable from by typing:

```
>>> wget https://repo.continuum.io/archive/Anaconda3-5.0.1-MacOSX-x86_64.pkg /path/to/
↪download/to/
```

```
>>> bash /path/to/download/to/Anaconda3*.sh
```

For more information on how to download it, go to https://docs.anaconda.com/anaconda/install/#detailed-installation-information

Once you have downloaded Anaconda, you should be able to start using *python* and *iPython*. You can try this by typing the following on the terminal:

```
>>> which python
    /home/username/anaconda/bin/python

>>> which ipython
    /home/username/anaconda/bin/ipython

>>> ipython
    Python 3.6.3 |Anaconda custom (64-bit)| (default, Oct  6 2017, 12:04:38)
    Type 'copyright', 'credits' or 'license' for more information
    IPython 6.1.0 -- An enhanced Interactive Python. Type '?' for help.
```

Note: If you're using a separate machine, to which you *ssh*, you can install *Anaconda* to a specified location other than your home directory. This is important if you are limited by *the number of files in your **home directory**, e.g. a computer hosted by ACCRE.

2.1.2 Managing environments

When working on a project, it is really important to keep **reproducibility** in mind. For example, if you were to hand me you code, I should be able to read the documentation and understand it, as well as **running the code**.

This is why **creating an environment** for your project is extremely important. This is where Ananconda helps a lot. Anaconda let's you have your own defined environment for your project, and you can **specify which packages** to include in your project.

All of the packages can be specified in an **environment.yml** file. An example for such file would look like (taken from Conda Manage Environments):

```
name: example-environment

channels:
  - defaults

dependencies:
  - python=3
  - anaconda
  - astropy
  - h5py
  - numpy
  - pandas
```

(continues on next page)

(continued from previous page)

```
  - scipy
  - seaborn
  - pip
  - pip:
    - GitPython
    - progressbar2
    - halotools
    - sphinx_rtd_theme
```

You can install the desired environment *example-environment* by running the command on the terminal:

```
>>> conda env create -f environment.yml
```

For more information, see Creating an environment from an environment.yml file.

Note: A helpful package to use is conda-env-auto which allows you to automatically create and *activate* the project environment once you are in the directory. For more information on how to install it and use it, see https://github.com/chdoig/conda-auto-env.

For more on environments and how to integrate them in your project, see *Structuring your Project* and *Editing your environment*.

CHAPTER 3

Structuring your Project

Now that your have a *working* version of **python** on your computer, you can start doing research.

One of the key elements of a project is for it to be **reproducible** by others. Having this in mind when you're structuring your project will allow others to look at your code, understand it well enough to be able to **recreate** your results.

This is a short guid on 2 ways to structure your code, without having to do much creating of documets, etc.

Table of Contents

3.1 Cookiecutter and Folder structure

Cookiecutter is a command-line utility that creates projects from cookiecutters (project templates), e.g. Python package projects, LaTeX documents, etc.

Cookiecutter has been widely used for many projects, and each team and organization can create their own *template*. For more information, visit the cookiecutter documentation.

As the famous say goes:

> Don't reinvent the wheel!

You can always create your own folder and file structures, and organize your documents the old-fashioned way. The problem with this is that it may **vary** from project to project, and it will be more difficult to be consistent and effective throught your projects.

For this reason, I rely on `cookiecutter` templates to create the file and folder structure of a project.

There are many different `cookiecutter` templates out there, but after trying to find the best one that suits my needs in **research** and **programming**, I found one that works great! And after some modifications, I came up with a *version* of this template.

These two templates are shown in *Data Science - Cookiecutter* and *Personal version - Cookiecutter*.

3.1.1 Data Science - Cookiecutter

Cookiecutter Data Science is best described as

> A logical, reasonably standardized, but flexible project structure for doing and sharing data science work.

This folder structure allows everyone looking at your code to understand it right away. It also provides many different functions (as part of a `Makefile`) that simplify the workflow of your project.

In a nutshell, this cookiecutter includes:

- A **Makefile** file with **useful functions.
- **Documentation** to make your project easily accessible and readable
- And more!

In order to use this template, you follow the documentation in Cookiecutter Data Science.

3.1.2 Personal version - Cookiecutter

If you need more than the *normal* Data Science Cookiecutter template, you can use my version. Some of the differences are:

- It includes and easy-to-use `environment.yml` file that makes it easy to install dependencies.
- Extra functions in the `Makefile`.
- Choice of what kind of documenation to use. One has the option choose from *traditional* Read The Docs style or the Astropy Sphinx Theme.

You can check how these two styles look like:

- - **Read The Docs Version**
- - **Astropy Version**

Next, you can create your own Project based on this *cookiecutter* version

Createing your own Project using Cookiecutter

The first thing to do is to install cookiecutter

```
$ pip install cookiecutter
```

or

```
$ conda config --add channels conda-forge
$ conda install cookiecutter
```

To start a new project

To start a new project, type the following:

```
$ cookiecutter https://github.com/vcalderon2009/cookiecutter-data-science
```

If you want the **default** project scheme from *DrivenData* (see above), run:

```
cookiecutter https://github.com/drivendata/cookiecutter-data-science
```

Depending on what kind of folder structure you want, you might want to choose from the different types.

After running this command, **you will be prompted some questions** regarding the parameters for the project.

The resulting directory structure

The directory structure of your new project looks like this:

```
├── LICENSE
├── Makefile           <- Makefile with commands like `make data` or `make train`
├── README.md          <- The top-level README for developers using this project.
├── data
│   ├── external       <- Data from third party sources.
│   ├── interim        <- Intermediate data that has been transformed.
│   ├── processed      <- The final, canonical data sets for modeling.
│   └── raw            <- The original, immutable data dump.
│
├── docs               <- A default Sphinx project; see sphinx-doc.org for details
│
├── models             <- Trained and serialized models, model predictions, or model␣
↪summaries
│
├── notebooks          <- Jupyter notebooks. Naming convention is a number (for␣
↪ordering),
│                         the creator's initials, and a short `-` delimited␣
↪description, e.g.
│                         `1.0-jqp-initial-data-exploration`.
│
├── references         <- Data dictionaries, manuals, and all other explanatory␣
↪materials.
│
├── reports            <- Generated analysis as HTML, PDF, LaTeX, etc.
│   └── figures        <- Generated graphics and figures to be used in reporting
│
```

(continues on next page)

(continued from previous page)

```
├── requirements.txt   <- The requirements file for reproducing the analysis␣
↪environment, e.g.
│                         generated with `pip freeze > requirements.txt`
│
├── environment.yml    <- The Anaconda environment requirements file for reproducing␣
↪the analysis environment.
│                         This file is used by Anaconda to create the project␣
↪environment.
│
├── src                <- Source code for use in this project.
│   ├── __init__.py    <- Makes src a Python module
│   │
│   ├── data           <- Scripts to download or generate data
│   │   │
│   │   └── make_dataset.py
│   │
│   ├── features       <- Scripts to turn raw data into features for modeling
│   │   └── build_features.py
│   │
│   ├── models         <- Scripts to train models and then use trained models to make
│   │   │                 predictions
│   │   ├── predict_model.py
│   │   └── train_model.py
│   │
│   └── visualization  <- Scripts to create exploratory and results oriented␣
↪visualizations
│       └── visualize.py
│
└── tox.ini            <- tox file with settings for running tox; see tox.testrun.org
```

Editing your environment

Now that you have a working proect from **cookiecutter**, you can start by editing the *environment* of your project.

If you downloaded **my version of cookiecutter**, you should be able to edit the `environment.yml` file. This file states which packages need to be installed by Anaconda and `pip` in order to run the scripts of the package.

The `environment.yml` file looks like the following:

```
name: name_of_environment

channels:
  - defaults

dependencies:
  - python>=3.6
  - ipython
  - anaconda
  - astropy
  - h5py
  - numpy
  - pandas
  - scipy
  - seaborn
  - pip
  - pip:
```

(continues on next page)

(continued from previous page)

```
    - GitPython
    - progressbar2
```

You can edit the `environment.yml` file to include/exclude packages needed by your project.

After having edited the list of packages needed by your project, you can execute the command

```
$ make environment
```

to **create the environment**.

If you have done this step before, and you want to **update the environment**, you need to run

```
$ make update_environment
```

instead.

Adding your Project repository to Github

If you follow the instructions from above, you should have

- Downloaded the repository
- Created your own project with the desired file and folder structure
- Created your working **environment** for you project

The next step is to add it to Github and make it accessible.

To do this, your should do the following:

1. Create a Github repository with the **same name** as the repository.
2. Type `git add remote origin git@github.com:<username>/<project_name>.git`. In here you need to **replace** `<username>` and `project_name` with your details.
3. `git push origin master` - This will push your project to Github.

To check that you did this correctly, type

```
git remote -v
```

and you should get something that looks like this:

```
origin  https://github.com/<username>/<project_name>.git (fetch)
origin  https://github.com/<username>/<project_name>.git (push)
```

where `username` and `project_name` pertain to your repository on Github.

Now all of the files are online on Github, and should be ready to integrate them with Read The Docs.

Documentation for your new project

Now that you have both a working local and online copy of your code, the next step is to create the documentation for the project.

For this, you can easily use Read The Docs (RTD).

You need to do the following:

- Create an account on "Read the Docs"
- Go to your `Profile` and select `My Projects`
- From there, you should import the repository *manually* (it's easier). Click on `Import a Project` and follow the instructions.
- You should add the project with the **same name** as the Github Repo if possible. Otherwise, you might need to **change** the links to the *badges* on the `README.md` files in the project, among others.
- Make sure that the repository was correctly built by looking at the `Builds` and see that it compiled correctly. If not, it should tell you if there was an error and what the error was.
- Now you go and change the documentation depending on the project's needs.

Continuous Integration for your Project

Continuous integration deals with testing your code for possible errors, and making sure that everything is working as expected. Depending on your project's needs.

This template includes a `.travis.yml`, which the files used by Travis CI. Travis CI is a *Continuous integration* platform for testing your code, and checking the functionality of your project.

More to come!

3.2 Links and Resources

For more information on, you can take a look at *Code Structure* for links and resources on how to structure your code and more.

CHAPTER 4

Python Packaging - Building your own module and package

Table of Contents

4.1 Introduction

Each project requires its own different functions, definitions, etc. But most of the times, you will be recycling over old code, over and over, and you will have many duplicates of the same code laying around.

This is why it's important to know how to **build your own python module**.

This is a small tutorial on how to property setup your module, and about some of the **very useful** tools that you can use in order to

- Test your module for errors (Travis CI)
- Keep your documentation up to date (ReadTheDocs)
- and more.

4.2 Initial Setup

In order to construct your package, you first need to setup your package with the following folder structure:

```
your_package/
    LICENSE.txt
    README.txt
    setup.py
    package_name/
        __init__.py
```

As you can tell, there's a file named `__index__.py`. This file tells Python to treat this directory as a *module/package*. Without this file, Python will not understand that your building a package, and will not link the files properly.

There are many different ways to create a Python module. I prefer to use cookiecutter. Cookiecutter is a command-line utility that creates projects from cookiecutters (project templates), e.g. creating Python package project from a Python package project template. For more information on cookiecutter, see https://github.com/audreyr/cookiecutter .

For creating a Python project, there are different methods. But I prefer these two methods:

- Astropy Template
- PyPackage

Depending on what kind of package you intend to produce, the Astropy Template or PyPackage should be enough for creating a new Python package.

4.2.1 Astropy Template

This template is powered by the Astropy Project. After answering a few questions, this project will create the folder structure for your new Python Package.

Installing Astropy Package

To first install the Astropy Template, you need to run:

```
conda install -c conda-forge cookiecutter gitpython
```

or:

```
pip install cookiecutter gitpython
```

This will instal the necessary dependencies of `cookiecutter`. Now you can go ahead and run the following commands to create the structure of your Python package:

```
cd /path/to/new/Python/Package
cookiecutter -c cookiecutter gh:astropy/package-template
```

This will prompt you with a series of questions about your project. For a full list of the different options during Setup, see Options during Setup.

Setting up Continuos Integration

After having created the folder structure of the new Python structure, it is advisable to use continous integration to ensure that all of the modules and functions are behaving the way they are supposed to.

The `Astropy Template` comes with easy-to-use files that you can modify to use with Travis CI and other CI clients.

For more information on CIs and how to use them, see Setting up Continous Integration .

Documentation and Read the Docs

4.2.2 PyPackage Template

4.3 Resources and further reading

This is a small list for further reading

- The Hitchhiker's Guide to Packaging
- Python Packaging User Guide
- Astropy Package Template

CHAPTER 5

Machine Learning

Table of Contents

5.1 Why machine learning?

The idea of machine learning (ML) is not new, but ML has become a tool that many astronomers are using at their analysis.

It has been used to:

- Predict the *photometric redshifts* of distant galaxies
- Determine HI richness of galaxies
- Classify galaxy morphologies, i.e. *ellipticals*, *disky*, *barred*, etc.
- and many other applications.

Machine learning has definitely started to change the field of astronomy and astrophysics. This is the reason for why it is important to properly understand how to get the best results for a ML project.

5.2 ML algorithms

There are two main types of ML algorithms, i.e. **supervised** and **unsupervised**. The first type is mainly used for *classification* problems, while the second one is mainly used for *clustering* problems.

5.2.1 Supervised Machine Learning

Supervised machine learning refers to the *prediction* of labels (`classification`) or values (`regression`) based on a *previous* knowledge of the data, i.e. the user know the *truth* of each observation, and tries to make an educated guess of for unobserved data based on a **training** process with data.

5.2.2 Unsupervised Machine Learning

5.3 Papers that have used ML

This is a list of a few papers that have used machine learning and deep learning to study astrophysical phenomena.

5.3.1 Cosmology and Galactic astronomy

5.3.2 Stellar Astrophysics

5.3.3 Observational Astronomy

CHAPTER 6

Vanderbilt PhD Thesis - Template

Whenever you're getting close to finishing your Ph.D., you will eventually have to start writing up your dissertation. There are many templates out there, but there has been one template passed around between Astronomy graduate students at Vanderbilt *from generation to generation*.

Now, it's even easier to start writing your dissertation if you follow the following steps:

6.1 Steps to take to write your dissertation

The dissertation can be found at: Vanderbilt Astro PhD Template

This template is easy to use, and you only need to answer some questions.

6.1.1 Downloading Vanderbilt PhD Thesis

You first need to run:

```
cd /path/to/where/main/thesis/will/be/
pip install cookiecutter
cookiecutter https://github.com/VandyAstroML/Vanderbilt_Astro_PhD_Template
```

This will install the necessary packages and directories for the PhD Thesis.

Note: Make sure you `cd` into the **correct path**. Otherwise, you will be downloading the repository wherever.

6.1.2 Downloading Vanderbilt PhD Thesis

Next, it will prompt you for some answers. The different prompts are:

Question	Description
`thesis_title`	Title of the thesis. Should not have '_' symbols in it. Examples: • Understanding Exoplanets and Other Variable Sources • The Clustering of Galaxies on the Smallest Scales Across Cosmic Time
`first_name`	Author's first name. `first_name` will used for the *title page* of the dissertation. Examples: • Adam • Rose
`last_name`	Author's **last** name. `last_name` will used for the *title page* of the dissertation. Examples: • Calderon • Piscionere
`repo_name`	Name of the directory/repository, in which the thesis will be saved.
 This name is selected by default, but can be changed.
 This field **should not contain spaces**. Examples: • Calderon_Victor_Astro_PhD_Thesis • Szewciw_Adam_Astro_PhD_Thesis
`add_signatures`	Option for adding signatures to the thesis. Options: 1. "y" … Add signatures 2. "n" … Do not add signatures
`department_name`	Name of the department. Default: **Physics and Astronomy**. Should **not** contain '_' (underscores) symbols. Examples: • Physics and Astronomy • Name of another department
`dissertation_date`	Date of the Dissertation presentation. Format: `Month Year`. Examples: • May 2019 • August 2020
`name_committee_1`	First and last name of the committee member 1. Should not have '_' symbols in it. Examples: • Keivan Stassun • Andreas Berlind
`name_committee_2`	First and last name of the committee member 2. Should not have '_' symbols in it. Examples: • Keivan Stassun • Andreas Berlind
`name_committee_3`	First and last name of the committee member 3. Should not have '_' symbols in it.

6.1.3 Writing the Thesis

Once you've downloaded the repository and answered all of the questions, you can start writing your thesis.

My advice would be to follow these steps to guarantee that you're doing it correctly:

1. Create a new repository on Github. This will be the repository for your newly created local repository.
2. `git init` your local repository.
3. Follow the instructions to upload the files of your dissertation to Github.
4. Write your dissertation.

After having downloaded and answered the questions, the repository should look like this:

```
Calderon_Victor_Vanderbilt_Astro_PhD_Thesis/
├── Bibliography
│   └── bibliography.bib
├── Chapters
│   ├── acknowledgments.tex
│   ├── appendix_A.tex
│   ├── chapter_1.tex
│   ├── chapter_2.tex
│   ├── chapter_3.tex
│   ├── chapter_4.tex
│   ├── dedication.tex
│   ├── future_work.tex
│   ├── introduction.tex
│   └── titlepage.tex
├── Extras
│   ├── commands.tex
│   ├── headings_settings.tex
│   └── packages.tex
├── Figures
│   ├── project_1
│   ├── project_2
│   └── project_3
├── Makefile
├── README.md
├── Thesis
│   └── thesis.tex
└── requirements.txt

8 directories, 18 files
```

This is the file structure after downloaing the repository.

The main file of the repository is: `Thesis/thesis.tex`. This is the file that will get compiled by LaTeX, and will produce a PDF version.

The only files that you will need to **edit** (aside from `thesis.tex`) are located in the `Chapters` directory. These are the ones that you need to edit.

6.1.4 Compiling your Thesis

This repository includes a `Makefile`. This file serves as the file that will make the *cleaning*, *compiling*, and *opening the pdf* of the `thesis.tex` file.

To show all of the options of the Makefile, write:

```
make show-help
```

This will show you a list of options:

```
./Calderon_Victor_Vanderbilt_Astro_PhD_Thesis: make show-help
Available rules:

all                 Perform all tasks
clean               Clean all unnecessary latex-related files
open_pdf            List all unnecessary files
thesis.tex          Compiles Main Thesis file
```

To compile your thesis, you will need to run the following commands:

```
make all
```

or

```
make thesis.tex
```

This will create all of the necessary files for compiling your thesis.

To open the PDF version of the thesis, run:

```
make open_pdf
```

and a PDF version of the `thesis.tex` file will pop up.

Note: In order to properly use the Makefile and compile `thesis.tex`, you will need `latexmk` installed. If you're on a MAC, you want to check out the Latexmk documentation, and make sure to have MacTex installed on your computer.

An example of the resulting PDF can be found in:

CHAPTER 7

Tools you might want to check out

Table of Contents

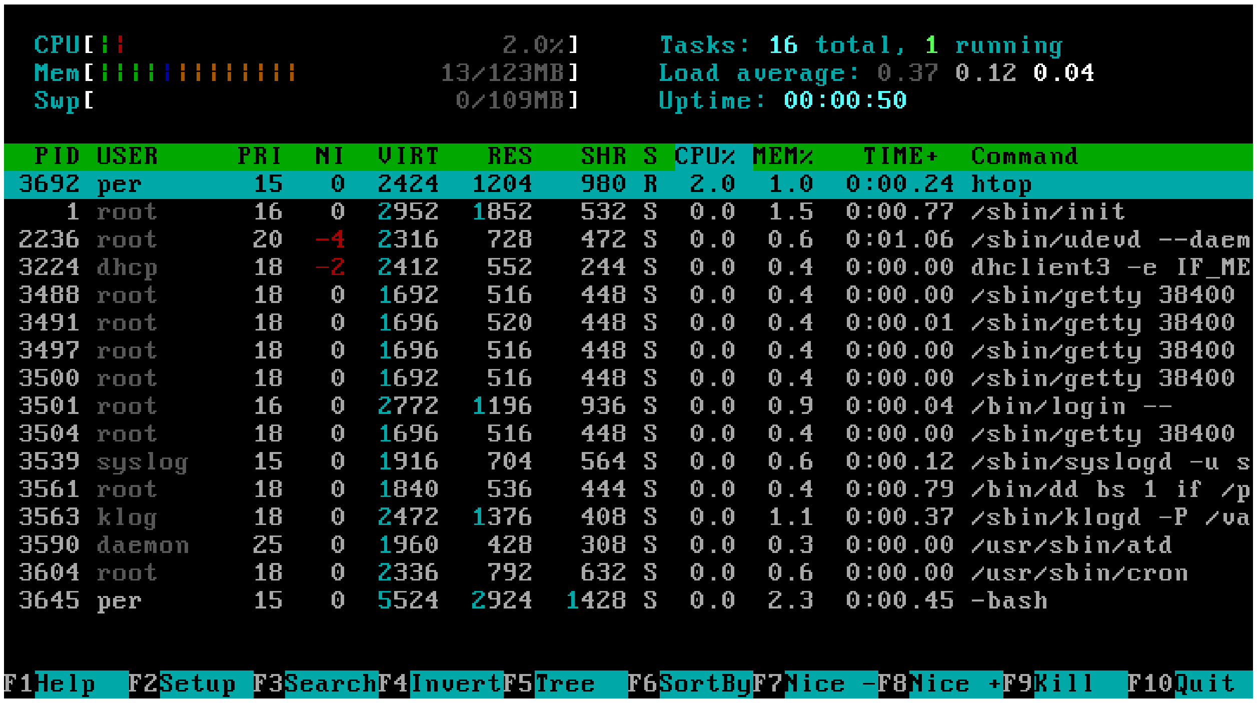
7.1 Introduction

There are tools out there that can make your workflow **much, much smoother**. This is a small list of some of the tools that I've found useful throughout my stay at Vanderbilt. They have significantly improved my workflow, and have made the projects much easier to understand.

7.2 htop - An interactive viewer for Unix

For viewing which processes are running on your computer and how much memory is left, I like to use htop.

This application, as shown in



```
  CPU[||                        2.0%]   Tasks: 16 total, 1 running
  Mem[|||||||||||||           13/123MB]   Load average: 0.37 0.12 0.04
  Swp[                         0/109MB]   Uptime: 00:00:50

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 3692 per        15   0  2424  1204   980 R  2.0  1.0  0:00.24 htop
    1 root       16   0  2952  1852   532 S  0.0  1.5  0:00.77 /sbin/init
 2236 root       20  -4  2316   728   472 S  0.0  0.6  0:01.06 /sbin/udevd --daem
 3224 dhcp       18  -2  2412   552   244 S  0.0  0.4  0:00.00 dhclient3 -e IF_ME
 3488 root       18   0  1692   516   448 S  0.0  0.4  0:00.00 /sbin/getty 38400
 3491 root       18   0  1696   520   448 S  0.0  0.4  0:00.01 /sbin/getty 38400
 3497 root       18   0  1696   516   448 S  0.0  0.4  0:00.00 /sbin/getty 38400
 3500 root       18   0  1692   516   448 S  0.0  0.4  0:00.00 /sbin/getty 38400
 3501 root       16   0  2772  1196   936 S  0.0  0.9  0:00.04 /bin/login --
 3504 root       18   0  1696   516   448 S  0.0  0.4  0:00.00 /sbin/getty 38400
 3539 syslog     15   0  1916   704   564 S  0.0  0.6  0:00.12 /sbin/syslogd -u s
 3561 root       18   0  1840   536   444 S  0.0  0.4  0:00.79 /bin/dd bs 1 if /p
 3563 klog       18   0  2472  1376   408 S  0.0  1.1  0:00.37 /sbin/klogd -P /va
 3590 daemon     25   0  1960   428   308 S  0.0  0.3  0:00.00 /usr/sbin/atd
 3604 root       18   0  2336   792   632 S  0.0  0.6  0:00.00 /usr/sbin/cron
 3645 per        15   0  5524  2924  1428 S  0.0  2.3  0:00.45 -bash

F1Help  F2Setup  F3SearchF4InvertF5Tree   F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

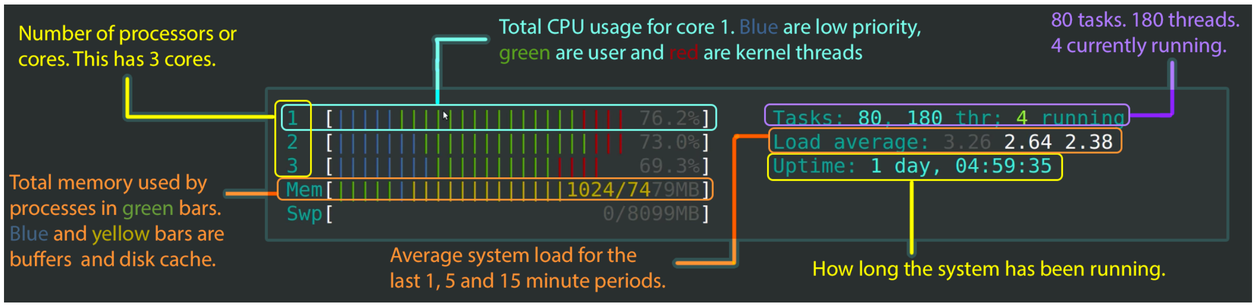
Fig. 1: Figure 1.1: htop running in the terminal. Credit: Wikipedia

This tool is extremely useful when running multiple jobs, since it lets you see which *jobs* are running, for how long they've been running, and more. Figure `Fig-1.1` how *htop* looks whenever you run this from the terminal:

```
htop
```

7.2.1 htop Explained

htop has different things to offer. Figures `Fig-1.2` and `Fig-1.3` explain what each column means.



```
1 [||||||||||||||||||||||||||||||    76.2%]   Tasks: 80, 180 thr; 4 running
2 [|||||||||||||||||||||||||||||     73.0%]   Load average: 3.26 2.64 2.38
3 [|||||||||||||||||||||||||         69.3%]   Uptime: 1 day, 04:59:35
Mem[||||||||||||||||||||||||||1024/7479MB]
Swp[                            0/8099MB]
```

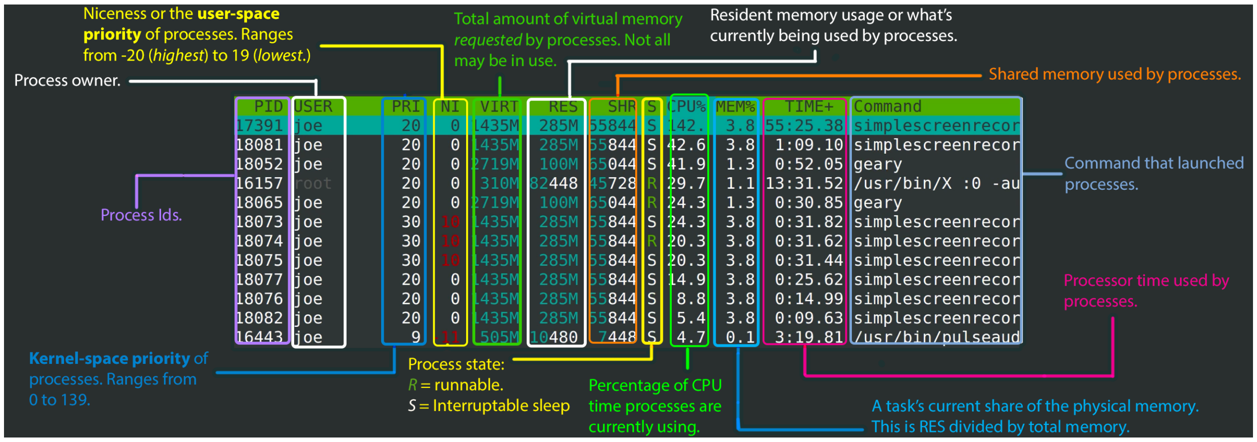
Number of processors or cores. This has 3 cores.
Total CPU usage for core 1. Blue are low priority, green are user and red are kernel threads
80 tasks. 180 threads. 4 currently running.
Total memory used by processes in green bars. Blue and yellow bars are buffers and disk cache.
Average system load for the last 1, 5 and 15 minute periods.
How long the system has been running.

Fig. 2: Figure 1.2: Top of *htop*. This figure shows the different components in the upper part of *htop*. Credit: CodeAhoy

And the bottom part . . .

7.2.2 Further Reading

For a **more in depth discussion** of the different sections of `htop`, see:


Niceness or the **user-space priority** of processes. Ranges from -20 (*highest*) to 19 (*lowest.*)
Total amount of virtual memory *requested* by processes. Not all may be in use.
Resident memory usage or what's currently being used by processes.
Process owner.
Shared memory used by processes.
Process Ids.
Command that launched processes.
Processor time used by processes.
Kernel-space priority of processes. Ranges from 0 to 139.
Process state:
R = runnable.
S = Interruptable sleep
Percentage of CPU time processes are currently using.
A task's current share of the physical memory.
This is RES divided by total memory.

```
  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
17391 joe        20   0 1435M  285M 55844 S 142.  3.8 55:25.38 simplescreenrecor
18081 joe        20   0 1435M  285M 55844 S 42.6  3.8  1:09.10 simplescreenrecor
18052 joe        20   0 2719M  100M 65044 S 41.9  1.3  0:52.05 geary
16157 root       20   0  310M 82448 45728 R 29.7  1.1 13:31.52 /usr/bin/X :0 -au
18065 joe        20   0 2719M  100M 65044 R 24.3  1.3  0:30.85 geary
18073 joe        30  10 1435M  285M 55844 S 24.3  3.8  0:31.82 simplescreenrecor
18074 joe        30  10 1435M  285M 55844 R 20.3  3.8  0:31.62 simplescreenrecor
18075 joe        30  10 1435M  285M 55844 S 20.3  3.8  0:31.44 simplescreenrecor
18077 joe        20   0 1435M  285M 55844 S 14.9  3.8  0:25.62 simplescreenrecor
18076 joe        20   0 1435M  285M 55844 S  8.8  3.8  0:14.99 simplescreenrecor
18082 joe        20   0 1435M  285M 55844 S  5.4  3.8  0:09.63 simplescreenrecor
16443 joe         9 -11  505M 10480  7448 S  4.7  0.1  3:19.81 /usr/bin/pulseaud
```

- htop Explained Visually
- Understanding and using htop monitor system resources
- htop Explained

7.3 tmux and screen - Terminal Multiplexers

Two other great tools that you should get familiar with are:

- `screen` - GNU Screen
- `tmux` - Github Tmux

These two tools are essential when working on the terminal for a long time. Assume you have a script that takes a long time to complete. If you dedice to go for dinner, or leave school to go home, you would have to stop the script since it wouldn't be running anymore.

7.3.1 Screen

The `screen` program allows you to multiple virtual windows in **Unix**.

Some of the features of `screen` are (from this page):

- **If your local computer crashes or you lose the connection, the processes or login sessions you establish through screen don**
  - You can resume your screen session with the command: `screen -r`
  - In some cases you may have to manually *detach* your screen session before resuming it.
- The `screen` program creates multiple processes instead of multiple Unix login sessions, which means that it is resource-efficient.
- You can cut and paste between different screens without using a mouse. Thus, you don't need to be on a computer with a windowing environment such as macOS, Windows, or the X Window System.
- It has a block copy feature which is similar to the kill rectangle feature of Emacs.
- You can copy and paste more than one page at a time, which you cannot do with some clients. You can scroll up more than one page, depending on how many scrolling lines you have set with the `-h` option.

- Using the detach feature, you can save screen processes when logging out and resume where you left off, saving the trouble of restarting them.

A useful set of commands for using `screen` are:

Command	Purpose
`Ctrl-a c`	Create new window (shell)
`Ctrl-a k`	Kill the current window
`Ctrl-a w`	List all windows (the current window is marked with "*")
`Ctrl-a 0-9`	Go to a window numbered 0-9
`Ctrl-a n`	Go to the next window
`Ctrl-a Ctrl-a`	Toggle between the current and previous window
`Ctrl-a [`	Start copy mode
`Ctrl-a ]`	Paste copied text
`Ctrl-a ?`	Help (display a list of commands)
`Ctrl-a D`	Power detach and logout
`Ctrl-a d`	Detach but keep shell window open

For some useful tutorials, see:

- In Unix, what is screen, and how do I use it?
- Learn to use screen, a terminal multiplexer

7.3.2 Tmux

For further reading and tutorials, see these:

- A Gentle introduction to tmux

7.4 Crontab

7.5 Whole Tale

CHAPTER 8

Useful links and Resources

This is a set of useful links and resources that might make your days as a graduate student much more comfortable. It includes links related to science, coding, machine learning, etc.

Table of Contents

8.1 General

- Vanderbilt University
- Vanderbilt Astronomy Group
- Journal Club - Schedule
- AstroBrew Schedule
- Vandy Astronomy - Workshops and Seminars - Main Website for the Astronomy Workshops and Seminars at Vanderbilt

8.2 MAC-related (if applicable)

- Setting up a MAC
- Apps for MACs

8.3 Reading Papers

- ADS
- ADS 2.0 Beta
- arxiv astro-ph
- Vanderbilt VoxCharta - ArXiv discussion of the latest astronomy papers
- Arxiver - Bringing you the latest astronomy papers uploaded to astro-ph
- Astrobites - The astro-ph reader's digest
- Astrobitos - Spanish version of **Astrobites**
- Daily Brew - Selected astro-ph abstracts for Astro
- Benty Fields - Selected papers to use. It uses ML to suggest you new papers.
- AAS Nova - Research highlights from the journals of the American Astronomical Society

8.4 Giving Presentations

- Speak your science: How to give a better conference talk, Part 1
- Speak your science: How to give a better conference talk, Part 2
- Giving a Scientific Talk

8.5 Paper Management

- Mendeley

8.6 Taking Notes

- Evernote
- OneNote

8.7 Collaborations

- Slack

8.8 Conferences

- Python in Astronomy
- Canadian Astronomy Data Centre
- Conference Management Software for Astronomy

8.9 General Learning

- The Tech Savvy Mission - Empowering Researchers
- Coursera - Free Online Courses
- EdX - Free Online Courses
- World Science U
- World Science Festival
- TED Talks
- DataGeekette - Walkthrough for Aspiring Data Scientists

8.10 Python & Anaconda

- Python Course by Codecademy
- Learn Python the Hard way - Free online course on Python
- Anaconda and Python
- Scientific Python Lectures
- The Hitchhiker's Guide to Python - Useful guide to use when using Python. Both for novice and experty Python developers.
- The Hitchhiker's Guide to Packaging - Useful guide to know how to package your Python modules.
- Astro ML - Machine Learning and Data Mining for Astronomy
- Cython Tutorial
- iPython in Depth Tutorial
- Python for Scientists Tutorial - Set of lecture notes by Thomas Robitaille

8.11 Interesting Python Packages

- Halotools
- Astropy
- Pandas and Tutorial - Another tutorial
- Scikit-Learn
- Astropy - Affiliated Packages
- Corrfunc - Blazing fast correlation functions on the CPU

8.12 Python Packaging

- Astropy Template for Python Packages - Github and its documentation
- How to create and maintain a Python package using the Astropy template
- Cookiecutter template for a Python package - I actually prefer the Astropy Template because of how easy it is to set up.
- The Python Package Index (PyPi)
- Read The Docs - Create custom documentation for your package and project for **free**. See Astropy_Template_rtd_subsec for more info.
- Travis CI (Continuous Integration) - Used for Continous Integration. For more information, see Astropy_Template_continous_integrations_subsec.

8.13 Code Editors

- Sublime Text 3
- Sublime Text Unofficial Documentation
- PyCharm
- Atom Editor - Similar to Sublime Text.
- VIM for Beginners

8.14 Code Structure

- How to structure your code property
- Markdown CheatSheet
- Restructured Text (reST) CheatSheet
- Create Documentation with RST, Sphinx, Sublime, and GitHub
- An introduction to Sphinx and Read the Docs for Technical Writers

8.15 Version Control

- Github
- Github Guides and Tutorials
- Git tutorials and training by Atlassian
- Bitbucket

8.16 SSH Keys

- How to Set up SSH Keys
- Connecting to Github with SSH

8.17 LaTeX

- Installing LaTeX
- Learn LaTeX in 30 minutes

8.18 Writing Papers

- Overleaf - Collaborative Writing and Publishing
- ShareLatex - Another tool for Collaborative Writing and Publishing
- Acknowledgment Generator - Easy way to write the __Acknowledgement section of a paper

8.19 Online Presentations

- Frank van den Bosch Lectures or here
- Frank van den Bosch - Video Lectures
- Galaxy Formation Conferences - Lectures and Videos

8.20 Data Science

- Kaggle Learn - Faster Data Science Education
- Astronomy and Data Science Toolkin - Connecting Astronomers and Data Science
- 12 things I wish I'd known before starting as a Data Scientist

8.21 Machine Learning

- Vanderbilt Astro Machine Learning group
- AstroML
- AstroML Book or on Amazon
- Machine Learning Course by Andrew Ng (Stanford)
- Victor Lavrenko's playlist on machine learning tutorials
- Introductory Applied Machine Learning
- Udacity's Deep Learning course
- Intro to Random Forests - Good introduction to the topic of Random Forests in machine learning
- Data Science Learning Resources - Curated list of resources to learn *machine learning*
- Essentials of Machine Learning Algorithms (with Python and R Codes) - Nice explanation of ML concepts and algorithms
- TensorFlow tutorial - Non-traditional TensorFlow tutorial by Daniel Foreman-Mackey
- Essential libraries for Machine Learning in Python
- VIP Cheatsheets for Stanford CS 230 Deep Learning and in other languages
- Best Machine Learning resources

8.22 Papers

- Good Enough Practices in Scientific Computing by by Greg Wilson et al. (2016)
- "Ten Simple Rules for Making Research Software More Robust" by Morgan Taschuk et al. (2017)
- Interactive Notebooks: Sharing the Code by Helen Shen

8.23 Interesting Books

- The Cosmic Web by Gott
- Statistics Books by astronomers and physicists
- Practical Statistics for Astronomers
- Galaxy Formation and Evolution by Mo, van den Bosch, and White
- Extragalactic Astronomy and Cosmology by Peter Schneider
- Introduction to Cosmology by Barbara Ryden or the PDF version
- An Introduction to Modern Astrophysics by Bradley Carroll

8.24 Jobs and Applications

- Astronomy - Rumor Mill
- The Grad Cafe - Grad School Admissions Results, Tips, Forums, etc.
- AAS Job Register - Find and post astronomy related jobs
- Benti-Fields Job Market

8.25 Miscellaneous Links

- Badge Creator - Tool to create badges for your Github repositories.
- Guide to Science Policy - Astrobites