University of Texas at Austin
TAURUS 2017

Basic Programming & Python Workshop

Jackie Champagne
June 12, 2017

1 Command Line Tools

The most basic thing to learn when getting started with scientific research is navigating your way through the command line terminal. It will be a lot faster to change directories, create new directories, and run programs from the terminal. Many of these instructions are the same for Linux and the Mac OS, but these will specifically be the Mac OS commands. The terminal is an application found in the Utilities folder on the Mac.

1.1 Navigation

To find out where you are, type:

```
pwd
```

If you are in your home directory, the response will be:

```
/Users/yourcomputerusername
```

To create a new sub-directory called "research" in your current location, type:

```
mkdir research
```

To access this new directory, type:

```
cd research
```

Now your working directory is research. If you want to go up one directory (i.e. the host folder of where you are currently working), you can just type "`cd ..`" into the terminal. If you want to go back home, because don't we all, type `cd`. You can navigate to anywhere on your computer with `cd`, given the full path (`cd /Users/yourcomputer/Documents/UTAustin/research`, for example). Your computer knows your paths, so you can use tab complete when writing out a specific location; also, ~/ is a shortcut for the home directory, so the above can be written as `cd ~/Documents/UTAustin/research`.

If you have a file you would like to move to a different directory, type the following command; the first argument is the file to be moved, and the second is the location to drop it in.

```
mv file.txt research
```

If you're already in the research directory and you'd like to move the file here, you can also just type "`mv file.txt .`" in the terminal. The `mv` command also renames files, so if the second argument that you supply is not a directory that exists, the system will assume you'd like to rename the file. To rename a file:

```
mv file.txt differentfile.txt
```

When you are doing scientific research it is good to retain an original version of a file or image you are going to manipulate, so instead of moving a file from one directory to another, you can also copy it to a new location.

```
cp file.txt research
```

So now `file.txt` exists in both your home directory and the research directory.

To delete a file at any time, type:

```
rm file.txt
```

If you want to delete a directory and all its contents, you will need to add a recursive command:

```
rm -rf research
```

BE VERY VERY CAREFUL WITH RM -RF!

There are several ways to list and search for files in your directory. To list all the files and subdirectories in your current directory, type:

```
ls
```

To list the subdirectories and their contents, add `-R` to `ls`. You can also access some details about each file in your directory including the file size, who has permission to access it, and its last modified date by adding `-l`. (You can combine these at any time and type, for example, `ls -Rl`).

Another powerful symbol in the command line is the asterisk: * means "all" and can be used in several situations. If you would like to search your directory for just text files, you could type:

```
ls *.txt
```

You can also use * when you're not sure of the full name of your file but know part of it, or want to list multiple files with the same prefix/suffix combination. Say you're looking for `research_is_really_boring.txt` and `research_is_really_fun.txt`:

```
ls research*.txt
```

Your system will now list all files that begin with "research" and end with ".txt":

```
> research_is_really_boring.txt
  research_is_really_fun.txt
```

Perhaps now you know why you should be careful with `rm -rf`. Typing `rm -rf *` means get rid of absolutely everything in your current location. Maybe spring cleaning sounds attractive to you, but don't type this. Just don't. Especially not in your home directory. This is a good way to delete all your data. If you want to get rid of everything in a directory, go up one directory and do `rm -rf directory`.

1.2 Manipulating Files

There are a bunch of programs out there that let you edit basic text files, which can be used to write python or IDL scripts, create input files, or write out tabular data. I am an old fogey and prefer using vi, but you can use emacs and macvim too. To create a new text file, you can type:

```
vi file.txt
```

This will open the file and let you start typing. There are plenty of vi handbooks online for some of the fancier features, but the basic things you need to know if you're just writing a fast text file are the following:

- i - insert; you need to type this to start writing (`ESC` exits `INSERT` mode)
- o - insert and start writing on a new line
- r - replace this character with a new character. You can fix a typo by arrowing over to the character, typing "r" and then the replacement character.
- x - delete this character
- dd - delete this whole line
- :w & `ENTER` - save this document
- :q & `ENTER` - quit this document
- :wq & `ENTER` - save and quit
- / - typing / followed by anything will search the document for that sequence of characters

vi is especially useful when you need to edit your `.bashrc` or `bash_profile` files, say to change your python path or add an alias.

You can see the first couple of lines of a file by typing `more file.txt`.

Perhaps you have two files containing lists, and you want to compare what's in the two files. The `diff` command will help you! Say you have these files:

```
file1.txt: file2.txt:
bananas    bananas
apples     apples
oranges    oranges
blueberries strawberries
```

Now to find the difference:

```
diff file1.txt file2.txt
```

This will return `4c4 < blueberries --- > strawberries` (4c4 means the 4th line of the first file is different from the 4th line of the second file). It will print the results in the command line, but if you want to save the list of differences between the two files, you could add `> file3.txt` to your diff command. In fact, adding this sequence to any command will save the results to a new file.

Occasionally you will want to compress a group of files into a zipfile for the purpose of sharing with others. To create a zip file, do:

```
zip myzipfile.zip file1 file2 file3
```

To unzip it, just do `unzip myzipfile.zip`.

You can also use tar on Unix/Linux - tar.gz will give you a tighter compression. Many packages you download from the internet will come as .tar.gz files. To write:

```
tar -zcvf mytarfile.tar.gz file 1 file2 file3
```

To read:

```
tar -xzvf mytarfile.tar.gz
```

One more useful thing in the command line is opening programs (and writing shortcuts to open things). To run any program from the command line, simply type:

```
open -a Name\ of\ Program filename
```

The backslashes are necessary if there are spaces in the name; following the program name with a filename will open that file with that program. Sometimes this is really annoying to type out, so you can create an alias that does this a bit faster for you. You'll need a text editor to edit your .bashrc file. Let's create a shortcut for Adobe Acrobat Reader.

```
vi ~/.bashrc
(i for insert)
alias adobe = "open -a Adobe\ Acrobat\ Reader"
(ESC then :wq+ENTER for save and quit)
```

You'll have to restart your terminal for this to take effect, but now you can just type "adobe filename" to view a pdf. You can create an alias for any command.

1.3 Connecting to Another Computer

While you're working over the summer, you might need to connect to another computer in the department. To ssh to another computer, you'll need to have a username and know the full computer host name – in our department, it will be the computer name followed by .as.utexas.edu.

```
ssh username@computername.as.utexas.edu
```

It will then prompt you for a password. That terminal is now logged on to that computer, so any files you manipulate there will not have any effect on your computer. To log out, just type `exit`.

Sometimes you just want to copy something over from your advisor's computer, for example, without logging in. Then you can do secure copying, or scp:

```
scp [-r] username@computername.as.utexas.edu:/Users/username/path/to/file

    ~/Users/username/new/path/
```

Basically, you'll supply the path for the file on the host computer, and give it a location on your computer to deliver the file(s). If you want it in this directory, just type "." for that second argument. Remember the -r if you are copying a directory.

2 Getting Started with Python

2.1 Anaconda

The computer you're given will already have the Anaconda distribution of Python installed, so you're done with that step for now. However, it's still useful to know

some basic steps to Python installation. There is a version of Python installed to the Mac OS, and you could download Python from python.org, but it is generally advised to use Anaconda. This is an open-source distribution of Python that will allow you to easily keep track of your various Python packages and settings (and will also be immune to updates to the OS). Instructions for installation of Anaconda at are `anaconda.org`. For an astronomy-specific distribution that comes with almost every package an astronomer could want, I recommend Astroconda, maintained by the Space Telescope Science Institute.

There are two main versions of Python (with minor syntax differences between the two): 2.7 and 3.5. People have different preferences about the best version to use - 2.7 is older and its libraries better developed, but 3.5 has some fancy fixes for certain high-level processes. Almost everything you'd want to do in astronomy will be able to run on 2.7, and is typically the default installation on the OS. Very occasionally, however, there will be a package that is version-specific, and for that reason you might want to have side-by-side installations. This needs to be done using virtual environments to ensure that your two Python paths are set properly; otherwise, things like `pip install` often disappear into the void.

To set up different environments, you can use `conda create`:

```
conda create --name py35 python=3.5
```

So now if you type `conda list`, you will see your root environment as well as "py35". When creating environments you can also add a list of packages to install. To begin using this version of Python, you'll need to type `source activate py35` whenever you want to use Python 3.5; at the end, you can exit the terminal or type `source deactivate py35`. So, if you'd like, you could create a "py35" and a "py27" environment. If you want to get rid of the environment, just type `conda remove --name [name of environment] --all`. Again, this probably won't be immediately necessary in your career, but you never know when you might need this.

If you do not install additional environments, you won't need to `source activate` in the terminal - just type `iPython` to use Python in the terminal, or `jupyter notebook` to use a GUI version of Python, which brings us to our next section.

2.2 Jupyter Notebooks

Anaconda comes with the Jupyter notebook pre-installed. This is a GUI version of iPython, which will allow you to edit your code line by line interactively and

display plots inline. You may prefer to use python from the command line, but beginners (and even experts!) may find use in editing in a notebook. To open a notebook, simply type `jupyter notebook` into the terminal - no starting iPython required. This will open a new notebook from your working directory in your web browser (so you do need to be connected to the Internet for this to work). (Note: If you have virtual environments installed, you will need to call one before activating the notebook.) Our programming workshop will have accompanying Jupyter tutorials, and a link to these can be found HERE.

2.3 Running scripts from the command line

My general recommendation is to use a notebook to troubleshoot your code, but eventually put your final code in a .py file. You can edit .py files with any text editor discussed in section 1. The syntax between the notebook and the .py file is the same, but it's faster to run a script from the command line using keyword arguments. To insert places where your code will need inputs, you can use `sys`. Let's call this file code.py:

```
import sys
x = sys.argv[1]
y = sys.argv[2]
print(x, y)
```

This is an incredibly silly snippet of code that prints out exactly what you put in. When you use `sys.argv`, it tells Python to expect a value for that argument supplied by the user. To run this code, you'll type:

```
python code.py 1 2
```

Now you've supplied two arguments, so your code will spit out "1 2." This is how you will run any python script from the command line: "python" + "pyfile" + a value for each argument the code calls for.

2.4 Installing Packages

The main packages you'll use on a day-to-day basis in Python are things like numpy, scipy, and matplotlib, which are already installed. You will need to import them to use them, but no installation is required. If there is a package you want to add to

your Python distribution, you can usually download the source code and compile it yourself, but Python has taken care of that part for you with `pip`. Always keep in mind that you'll need to call your environment if you want something installed to a specific version of Python, but if you're still using the root environment, all that's required is:

```
pip install name_of_package
```

which you will type into the terminal, NOT in an iPython session. Python will automatically finish the installation for you, and you will be able to import it once you start a new Python session.

2.4.1 List of Useful Astronomy Packages

There are many Python packages for astronomy that will do data analysis, image analysis, cosmological calculations, plotting, etc. Included here is a list of the most useful packages you might want to install. A more complete list can be found at this link.

- astropy: a set of astronomical packages that incorporates cosmology, units & constants, image readers, and statistical fits
- aplpy: an image displayer which will read fits files & produce contour plots
- opencv: the open-source computer vision library; has interfaces that are sometimes necessary for other codes to run (this is a very optional package)
- emcee: runs Markov Chain Monte Carlo (MCMC) codes
- scipy: mostly used for statistics & calculus

All of these (except openCV) can be installed via pip install.

That's all for now, since there will be a lot more examples in Python in the actual Jupyter notebooks. I just wanted to set you up with some other Mac basics, but this is a living document and will probably expand in future versions!