
labibi Documentation

Release 0.1

C. Titus Brown

April 06, 2013

CONTENTS

1	Software Installation	3
1.1	Quick guide	3
1.2	Links and more details	3
1.3	Technical details	4
2	Day 1 / Morning	5
2.1	Introductions!	5
2.2	Using this Web site	5
2.3	Intro technology, and Python basics	5
2.4	Applying Python basics to data analysis	6
3	Day 1 / Afternoon	7
3.1	Doing data analysis	7
3.2	A bit more IPython Notebook and matplotlib	7
3.3	Testing	7
3.4	Installing Python packages; useful Python packages	8
4	Day 2 / Morning	9
4.1	Scripting with Python, and running scripts at command line	9
4.2	How computers work; online editing at github	9
5	Day 2 / Afternoon	11
5.1	Using the shell effectively	11
5.2	Version control with git and github	11
6	Wrapping up - Day 2, afternoon	13
6.1	Everything is on the Web site	13
6.2	Posting IPython Notebooks	13
6.3	Moving around directories in the shell	13
6.4	Shell scripts and pipelines	14
7	Indices and tables	15

Instructors: Titus Brown <ctb@msu.edu>, Karen Cranston <karen.cranston@nescent.org>, and Rich Enbody <enbody@msu.edu>.

The [Google Docs](#) link.

resources

9:30am-4:30pm, April 4th and April 5th, 2013.

SOFTWARE INSTALLATION

Please follow the instructions below. If you have trouble installing things, that's OK – we can help. Just be sure to download the files, because some of them are quite large.

1.1 Quick guide

1. Download and install [VirtualBox](#)
2. Download [this BIG file](#) and import it into VirtualBox.
3. Download and install [Anaconda CE](#).
4. If on Windows, [install Git Bash](#)
5. If on Mac OS X, either [install XCode](#) and the command line tools, OR (easier!) [just install Git](#).
5. Choose a code editor, below, and install that.

1.2 Links and more details

1.2.1 Virtual Machine

To give everyone a consistent, pre-configured environment we provide a Linux virtual machine. Install [VirtualBox](#) and download [this virtual machine image](#).

Load the VM into VirtualBox by doing Import Appliance and loading the .ova file.

1.2.2 Other Options than the virtual machine

To complete the entire workshop you need several things: a Bash shell, git, a code editor (though any plain text editor will work in a pinch), and a scientific Python installation that includes the IPython Notebook, NumPy, and matplotlib.

Even if you plan to install things yourself please download VirtualBox and the virtual machine as a backup. It can be extremely difficult to get everything set up correctly, and the virtual machine almost always works.

1.2.3 Bash

Mac:

No installation needed; the default shell in Mac OS X is bash.

Windows:

Install [Git Bash](#) following the instructions [here](#).

Linux:

The default shell is usually bash but if not you can get to bash by opening a terminal and typing “bash”.

1.2.4 Git

Mac:

Install [Xcode](#) and the command line tools (from the Download preferences pane) or install [just git](#).

Windows:

Install [Git Bash](#) following the instructions [here](#).

Linux:

If git is not already available on your machine you can try to install it via your distro’s package manager (e.g. apt-get). We can help you with this at the workshop.

1.2.5 Code Editor

Mac:

We recommend [Text Wrangler](#), [Sublime Text](#), or [Text Mate 2](#).

Windows:

[Notepad++](#) is a popular free code editor for Windows.

Linux:

[Kate](#) and [gedit](#) are options for Linux users.

1.2.6 Python

We recommend the all-in-one scientific Python installer [Anaconda CE](#). Installation on Mac and Linux requires using the shell and if you aren’t comfortable doing the installation yourself just download the installer and we’ll help you at the boot camp.

For other options check the Python4Astronomers page on [installing scientific Python](#).

1.3 Technical details

See: <http://software-carpentry.org/setup/>

DAY 1 / MORNING

2.1 Introductions!

Introductory talk

For more on efficiency, reproducibility, and correctness, see [this talk](#) Titus gave in Colorado yesterday,, as well as the paper [Best Practices in Scientific Computing](#).

You might also be interested in [this blog post](#) and the associated paper on version control.

2.2 Using this Web site

Reload to get the latest links!

Commenting

Here's a [Google Doc](#) that we'll paste stuff into:.

2.3 Intro technology, and Python basics

(Titus Brown)

Git quick guide:

- to retrieve the materials for the class, do:

```
git clone https://github.com/swcarpentry/2013-04-az.git
cd 2013-04-az/notebooks
```

at the shell prompt. On Windows, you may need to use Git Bash to create the directory, and then cmd.exe to run ipython notebook (see below).

- to update the materials, do:

```
git pull origin master
```

IPython Notebook quick guide:

- run

```
ipython notebook --pylab=inline
```

in the directory containing your notebook files. A Web view of the notebook should pop up.

If you're using Anaconda, you can type:

```
c:\anaconda\scripts\ipython notebook --pylab=inline
```

The trick will be running this in the correct directory, but you have a little bit of time to figure this out because we won't start with a preexisting notebook.

- executing code

```
shift-ENTER -- to execute current "code cell" and move to next  
ctrl-ENTER -- to execute current "code cell" and stay
```

Python installing guide:

- run, variously, one of

```
pip install <packagename>
```

or

```
~/anaconda/bin/pip install <packagename>
```

or

```
sudo pip install <packagename>
```

2.4 Applying Python basics to data analysis

(Rich Enbody)

[Counting birds \(full of code\)](#)

[Generating the raw data – for the interested](#)

[Rich's final notebook](#)

DAY 1 / AFTERNOON

3.1 Doing data analysis

(Karen Cranston)

See the IPython Notebook, [numpy-and-graphing.ipynb](#).

3.2 A bit more IPython Notebook and matplotlib

The whole IPython Notebook gallery

See: <http://ged.msu.edu/papers/2012-diginorm/> for an example of a paper where all the figures were generated with ipynb.

Matplotlib gallery: <http://matplotlib.org/gallery.html>

Note that you can use

```
%loadpy http://matplotlib.org/mpl_examples/api/bbox_intersect.py
```

on the source code in the gallery links to load the code into ipynb and execute it.

3.3 Testing

(Titus Brown)

See the IPython Notebook, [testing-with-nose.ipynb](#).

When we say “testing” we really mean *automated testing*. The central problems addressed by testing are *correctness* and *reproducibility*. (While these are linked, they are not the same!)

I don’t want to depart from the *efficiency* argument, though – being able to make sure your old code *stays* working is very important to moving forward with

There are two basic kinds of tests that I’d like to briefly discuss. One kind of test is a *unit test*. The other kind of test is a *regression test*. (There are also many more.)

Unit tests address *small units of code*, like functions. They are used to isolate and nail down and prove the functionality of potentially complicated little functions.

Regression tests address *the overall function of code*, and they are used to make sure that your code is doing the same thing *today* as it was *yesterday*.

I'll show you examples of both.

3.3.1 Writing tests

We're going to be using the nose testing framework, which is just a framework that makes it easy to find and execute tests.

Basically, 'nose' creates a command 'nosetests' that finds and runs tests. The idea is that you won't need to register new tests.

A test function looks like this:

```
def test_something():
    # run some code
    # fail loudly or succeed silently
```

3.3.2 More reading

For more reading, see:

http://software-carpentry.org/4_0/test/

and

<http://ivory.idyll.org/articles/nose-intro.html>

and also

<http://ivory.idyll.org/blog/software-quality-death-spiral.html>

3.4 Installing Python packages; useful Python packages

Using pip, which is an interface to PyPi

See `day1-python-packages`

Also see [the IPython Notebook](#), using `screeed.ipynb`.

Done!

DAY 2 / MORNING

4.1 Scripting with Python, and running scripts at command line

(Rich Enbody)

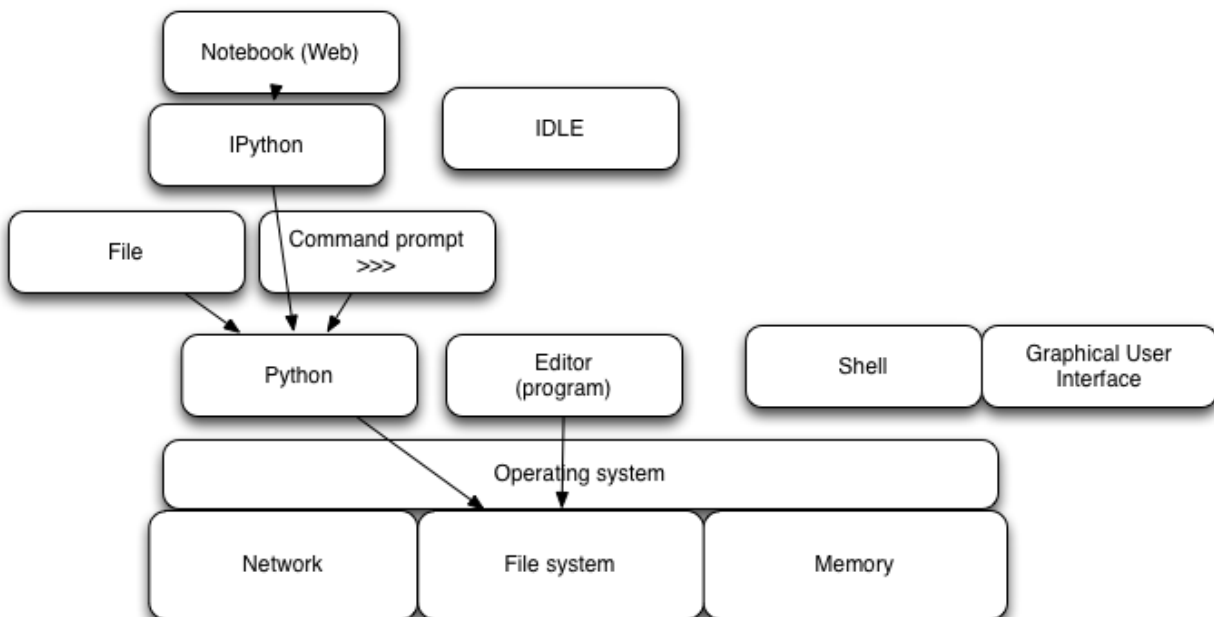
The final script from this section is available as:

<https://github.com/swcarpentry/2013-04-az/blob/master/scripts/birds.py>

or as 'scripts/birds.py'. Note that the data is in 'data/birds.csv'.

4.2 How computers work; online editing at github

(Titus Brown)



4.2.1 Github

Please sign up for an account at <http://github.com/>, and then log in.

We will:

1. Go to <https://github.com/swcarpentry/2013-04-az>
2. Click on the “Fork” button to make a local copy into YOUR account.
3. Go into the ‘example’ directory in your own account, which should be [https://github.com/\(ACCOUNT\)/2013-04-az/tree/master/example](https://github.com/(ACCOUNT)/2013-04-az/tree/master/example)
4. Click on *either* CHANGEME.txt *or* CHANGEME-2.txt
5. Click on Edit.
6. Make some changes.
7. Save and provide a commit message.

DAY 2 / AFTERNOON

5.1 Using the shell effectively

Navigating the shell; transferring files around; connecting scripts.

5.2 Version control with git and github

WRAPPING UP - DAY 2, AFTERNOON

6.1 Everything is on the Web site

We've tried to put everything up on the Web site. Please ask us if you can't find it!

Additional resources include those in `resources`, as well as using Google to look for answers, and checking out both `StackOverflow` and `BioStars`.

You might also consider `attending office hours`.

6.2 Posting IPython Notebooks

You can post IPython Notebooks on github, and view them statically via `nbviewer.ipython.org` (i.e. without loading them into a running IPython Notebook instance). For example, see:

`https://github.com/swcarpentry/2013-04-az/blob/master/notebooks/10-introducing-bird-counting-FULL.ipynb`

which can be viewed in “raw” form here:

`https://github.com/swcarpentry/2013-04-az/raw/master/notebooks/10-introducing-bird-counting-FULL.ipynb`

and which can be viewed in rendered form by pasting the ‘raw’ URL into `http://nbviewer.ipython.org`:

`http://nbviewer.ipython.org/urls/github.com/swcarpentry/2013-04-az/raw/master/notebooks/10-introducing-bird-counting-FULL.ipynb`

6.3 Moving around directories in the shell

Paths, directories, and file locations are a source of great confusing. Here are a few simple rules:

1. Treat everything as a relative location.

For example, if you're in the `2013-04-az/scripts` folder and you want to reference a file in the `2013-04-az/data` folder, use:

```
../data/filename
```

where the `‘..’` means “go up one level” and the `‘/data/’` means “go down into the data directory from there.”

2. `pwd` will tell you what directory you're in.
3. `cd` will change your current directory.

4. `ls` will list files in your current directory.
5. TAB completion is your friend. TAB completion does two things: it makes you less typo-prone, AND it makes sure that the file actually exists (because you can only tab-complete on files that are actually there).

6.4 Shell scripts and pipelines

In the `scripts` subdirectory, we have a number of Python scripts. Turn your attention to three of them, please:

`make-big-birdlist.py` – creates a bunch of fake bird data. Run by giving it the name of the file that you want it to output, e.g.:

```
python make-big-birdlist.py counts.csv
```

`make-birdcounts.py` – parses the `counts.csv` file and converts dates into day-of-year, then produced a histogram of bird counts by day. This is then saved into a `.dat` file that can be loaded by the next script. Usage:

```
python make-birdcounts.py counts.csv counts.dat
```

`plot-birdcounts.py` – loads in the `counts.dat` file, produces a plot, and then saves the plot.

```
python plot-birdcounts.py counts.dat counts.pdf
```

These are three automated scripts that each do some smaller part of an overall larger set of tasks - essentially, a data analysis pipeline.

You could run all three scripts above by hand, but that's error prone. You could also combine all three scripts above into one – there's no reason why not – but that's inflexible, especially if there are multiple ways to use the scripts (not so in the above case, but frequently true in real cases) or if some of the steps take a long time.

So what to do?

You can write a shell script, as in `make-plot.sh`. This can be run by typing:

```
bash make-plot.sh
```

All this is is a list of the commands you want run at the shell – simplicity itself, right? No hidden tricks here. It's that simple.

Note that a good way to get a shell script started is to run a series of commands at the shell, then – when done – type 'history' to get a list of the commands you've run. Copy/paste from that history into a file, and you've got a shell script of sorts!

Remember to version control your shell scripts :)

Writing shell scripts is a good way to keep track of what it is you've actually run, as opposed to what you think you've run. And once they're version controlled, well, now you're really in good shape for your methods section...

Final point: if you look at the shell script, you'll see that every time you run it, it runs all three commands. For large data sets, this can get time consuming, especially if you're working to optimize one step, or doing parameter explanation. There's a program called 'make' that can keep track of what has changed and what needs to be run again based on those changes, but you have to configure it a bit with a file called 'Makefile'.

For an example from our own pipeline for the diginorm paper, see:

<https://github.com/ged-lab/2012-paper-diginorm/blob/master/pipeline/Makefile>

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*