Everyday Scripting

Agustin Armendariz
aarmendariz@cironline.org
510.809.3173

This tutorial is intended for CAR reporters interested in adding some programming chops to their toolbox. I am no programmer, but I do some scripting every day to get work done faster and better.

Real programmers have done the hard work, so I just have to pull the levers correctly and have a good idea of what I'm doing.

Of course, if a real programmer looked at my code their face would melt. But who cares? It works for my purposes and I'm getting better. Slowly. Plus my efficiency rises 10 fold.

It's worth putting in the week of time to script a hard task instead of the 4 days it might take to do by hand. After you script it, then you can reproduce your work on demand at the drop of a hat. You can go back and fix things you might have messed up instead of starting over or using some sort of hack or work around. And as you script things more you will get better and be able to do it quicker.

So you've heard about python or ruby or perl and are curious. But you're not sure how it really fits into your toolbox. I'd argue that CAR reporters should utilize some programming chops to work more efficiently and effectively, delegating the boring, repetitive and time intensive work to computers.

Sample scripting tasks:

1) Cycle through folders of shapefiles and write out the missing projection files.
2) Cycle through foreclosed properties and query the voter registration database for phone numbers of the owners.
3) Turn a bunch of PDFs into text and search through them all.
4) Bust up all the worksheets in a folder full of Excel files.

All of the above tasks would take a lot of time to do by hand on a large scale, but using a computer program makes short work of the tasks. Not to mention it reduces the errors that creep in when doing boring, repetitive tasks by hand.

Computer Set-up

You'll need Python and SQLite installed on your computer to work through the exercises. If you are using a Mac, then they are already installed.

I strongly encourage you to use a package manager to install the software that's part of your CAR toolbox. On the Mac, Homebrew or MacPorts work fine. Installing either of those package managers will make your life so much easier. There appears to be a windows package manager too called Npackd (pronounced "unpacked"). I have no idea if it works, and would also encourage you to not use Windows.

Unix-like operating systems like Mac's OS X and Ubuntu come with an arsenal of powerful tools under the hood. Ubuntu is extremely easy to install on most windows laptops now. Use it. Stay with it. You will be richly rewarded. My parents love it and cannot break it.

If you simply must use Windows or need some time to ease into using a Unix-like system you can try using Cygwin. It will create a Linux-like environment for your Windows box.

You can also use a program like virtualbox to install Ubuntu and play around with it. It creates a virtual computer that you can set-up and destroy at will without fear of messing up your main system.

The command line AKA terminal

I don't use Windows anymore so I'll be talking about "terminal" commands. On unix-like systems the terminal is the command line. I recommend getting a copy of the "Linux Pocket Guide," by Daniel J. Barrett. It's a great desk reference and beginner's guide to navigating the terminal.

Couple quick conventions.

- When you see a dollar sign it indicates a terminal prompt. So a command to see what directory you are in on your computer would be noted like so:

  $ pwd

  The "$" indicates that at the command prompt, you enter "pwd" and hit enter.

- When you see a pound sign it indicates a comment. I will try to comment as much as I can on terminal commands and python code so that you know what's going on. So, taking the above example, here's how I will write out comments from now on.

  $ pwd # print your current working directory so you know where you are in the file system.

Here are the commands you'll use a lot:

- cd directory_name # move into a folder called directory_name
- wc -l file_name # return the number of lines in a file called file_name
- pwd # print your current working directory
- head -n 100 file_name # show the beginning 100 lines of file_name
- tail -n 100 file_name # show the last 100 lines of file_name.

But get the guide. It will help lower your frustration level as you're starting out.

Perhaps the single best gateway drug for using the terminal is csvkit. For anyone that works with data, this suite of tools created by Christopher Groskopf and other contributors are invaluable. They will force you to learn how to use the terminal as it was intended to be used, and show you the utter power of unix-like systems for data munging.

Why python

I started out using Perl for programming tasks, but now I pretty much exclusively use Python. I find python is perfect for my needs since tools like Django make web development and serious GIS work more accessible to me. There's also a statistical computing library for python called pandas.

Finally, Ipython makes it really easy to play around with data and perform other programming tasks. The main author of the pandas stats module wrote a book about using python for data analysis, and there is a whole chapter on using Ipython for data exploration. I highly recommend getting the book.

Getting Started

I'm going to install Ubuntu 12.04.1 on my Mac using VirtualBox and set it up for this tutorial. If you decide to use a virtual machine, you can fire up your virtual machine and follow right along.

Once I installed Ubuntu using VirtualBox, I fired up my new virtual machine and popped open the terminal and did the following commands.

The “which” command tells you where and if a terminal program is installed. Let's check the default Ubuntu install to see what's already available to us.

$ which python # check that python is installed and it is.
$ which easy_install # check that python setuptools is installed. It is not.

Ubuntu has an amazingly simple to use package manager that installs software. You can use the package manager from the command line or from the Software Center program.

$ sudo apt-get install python-setuptools # install pyhton setuptools
$ sudo easy_install virtualenv # use the easy_install program to install virtualenv
$ sudo apt-get install sqlite # install sqlite
$ sudo apt-get install mercurial # install the mercurial versioning system to install the nameparser module. I would use MacPorts or Homebrew to do this on a Mac.
$ sudo apt-get install python-tk # for Ubuntu users, this allows us to cut and paste into Ipython

The python virtualenv module lets us set-up a self-contained python programming environment that's isolated from the rest of the system. Let's create one now.

$ virtualenv –no-site-packages nicar_project
$ cd nicar_project # move into the new directory created by virtualenv
$ ls # check out the contents of the directory
$ source bin/activate # activate the virtual environment. Notice the parenthesis at the far left.

You always have to activate your virtual environment when you want to use it, and you can tell that it is active when you see the name in parenthesis at the far left of the terminal prompt.

$ mkdir project # make a folder or directory to hold our work and keep it separate from the other stuff
$ cd project # move into the project folder just created.

$ sudo apt-get install git-core # install cool-kid version control system Git
$ git clone https://github.com/armendariz/everyday_scripting.git # check out the code used for this tutorial

$ ls # list the contents of the current directory and see the new folder we just cloned from github.
$ cd everyday_scripting # move into the new folder

Inside the folder we cloned from github you'll see a file called “requirements.txt” which has a list of python packages we want to install in order to do this tutorial.

$ pip install -r requirements.txt # install the requirements.

Now we are ready to rock and roll. We now have the computer all set-up and ready to do some work

$ ipython # start-up ipython

```
In [1]: from db_audit_trail import * # load the code from this script
In [2]: do_it_do_it() # run the do_it_do_it function
```

If you aren't sure what something is you can always check in Ipython by putting a question mark at the end and hitting return. If, for example, I wanted to know what the “csv” thing is that I'm importing at the top of the script I would do this:

```
In [2]: csv?
```

Then I would see this:

```
Type:       module
Base Class: <type 'module'>
String Form:<module 'csv' from
'/opt/local/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/csv.pyc'
>
Namespace:  Interactive
File:
/opt/local/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/csv.py
Docstring:
CSV parsing and writing.

This module provides classes that assist in the reading and writing
of Comma Separated Value (CSV) files, and implements the interface
described by PEP 305.  Although many CSV files are simple to parse,
the format is not formally defined by a stable specification and
is subtle enough that parsing lines of a CSV file with something
like line.split(",") is bound to fail.  The module supports three
basic APIs: reading, writing, and registration of dialects.


DIALECT REGISTRATION:

Readers and writers support a dialect argument, which is a convenient
handle on a group of settings.  When the dialect argument is a string,
it identifies one of the dialects previously registered with the module.
If it is a class or instance, the attributes of the argument are used as
the settings for the reader or writer:

    class excel:
        delimiter = ','
        quotechar = '"'
        escapechar = None
        doublequote = True
        skipinitialspace = False
        lineterminator = '\r\n'
        quoting = QUOTE_MINIMAL
```

```
SETTINGS:

    * quotechar - specifies a one-character string to use as the
        quoting character.  It defaults to '"'.
    * delimiter - specifies a one-character string to use as the
        field separator.  It defaults to ','.
    * skipinitialspace - specifies how to interpret whitespace which
        immediately follows a delimiter.  It defaults to False, which
        means that whitespace immediately following a delimiter is part
        of the following field.
    * lineterminator -  specifies the character sequence which should
        terminate rows.
    * quoting - controls when quotes should be generated by the writer.
        It can take on any of the following module constants:

        csv.QUOTE_MINIMAL means only when required, for example, when a
            field contains either the quotechar or the delimiter
        csv.QUOTE_ALL means that quotes are always placed around fields.
```

When you're done reading it in Ipython it “q” on your keyboard to quit.

You can also cut and paste code from the script into ipython and play around with it. If you want to see how the name parsing works, just cut and paste each line in and see what it does.

```
In [3]: sql_select_command = ''' SELECT rowid, name_of_contributor
FROM `%s_raw_contribs` WHERE employer <> '' AND occupation <> '' '''
% date_stamp

In [4]: execute_query = c.execute(sql_select_command)

In [5]: rows = execute_query.fetchall()
```

Check out what's in the rows variable now. It's a list of rows. You can check how many rows are there using the len() function and can index into the rows like this: rows[0]

Every list starts with 0, not 1.

```
In [6]: len(rows)
Out[6]: 43
```

So we have 43 rows returned by our query and contained in this python list. Each row in our list is a “tuple” which is basically just a list itself.

Let's dissect the name parsing a bit.

```
In [7]: row = rows[0]

In [8]: row
out[8]: (1, u'MR. GEORGE JOSEPH')

In [9]: row[0]
out[9]: 1

In [10]: row[1]
out[10]: u'MR. GEORGE JOSEPH'

In [11]: name = row[1]
```

```
In [12]: name
out[12]: u'MR. GEORGE JOSEPH'

In [13]: parsed_name = HumanName(name)


In [14]: parsed_name?
Type:       HumanName
Base Class: <class 'nameparser.nameparser.HumanName'>
String Form:MR. GEORGE JOSEPH
Namespace:  Interactive
Length:     3
File:
/Users/agustin/Sites/virtual_environments2.7/playground/lib/python2.7/site-
packages/nameparser/nameparser.py
Docstring:
Parse a person's name into individual components

Usage::

    >>> name = HumanName("Dr. Juan Q. Xavier de la Vega III")
    >>> name.title
    'Dr.'
    >>> name.first
    'Juan'
    >>> name.middle
    'Q. Xavier'
    >>> name.last
    'de la Vega'
    >>> name.suffix
    'III'
    >>> name2 = HumanName("de la Vega, Dr. Juan Q. Xavier III")
    >>> name == name2
    True
    >>> len(name)
    5
    >>> list(name)
    ['Dr.', 'Juan', 'Q. Xavier', 'de la Vega', 'III']
    >>> name[1:-1]
    [u'Juan', u'Q. Xavier', u'de la Vega']

In [15]: parsed_name.title
Out[15]:  u'MR.'

In [16]: parsed_name.first
Out[16]: u'GEORGE'

In [17]: parsed_name.middle
Out[17]: u''

In [18]: parsed_name.last
Out[18]: u'JOSEPH'

In [18]: parsed_name.suffix
Out[18]: u''
```

So you see, you can cut and past and play around with the code in the scripts so see how things work and hopefully adapt them to your needs. I crafted them as examples of things I do every day at work so I hope the examples will be of immediate use and provide concrete examples of the kinds of things scripts are great for.

Here are some resources to learn more and help you understand how python works. This is just what helped me out. There are many other good books out there.

Couple of books:

Just working through the first three chapters of this book will help a ton as you get started.
"Think Python. How to Think Like a Computer Scientist," by Allen B. Downey
http://www.greenteapress.com/thinkpython/

"Think Stats. Probability and Statistics for Programmers," by Allen B. Downey, published by O'Reilly Media.
http://www.greenteapress.com/thinkstats/

Browse the python code on GitHub and play around with it. I think I've solved most if not all of my problems by using something I found on GitHub, in a book or through a random Google search.

In particular, Ben Welsh's page is amazingly useful. Both his personal and professional one.

https://github.com/palewire
https://github.com/datadesk

There are a ton of smart and helpful people on GitHub and this is just a quick sampling of who I pay attention to.

https://github.com/lukerosiak
https://github.com/sunlightlabs
https://github.com/aplayford

Stack Overflow is a great resource for answeres to most problems you'll run into. Search it if you hit a wall. http://stackoverflow.com/