

Welcome: to the Class and to the Command Line!

Jamie Saxon

Introduction to Programming for Public Policy

September 26, 2016

Why Learn to Code?

1. Technology powers the modern world.

- ▶ Good policy requires that we understand the system.
- ▶ Gain currency with technology so you can help govern it.
- ▶ What's at stake when services go wrong (VA, healthcare.gov).
- ▶ Understand the potential of algorithms to improve policy.

2. Expand your own toolset.

- ▶ Find, manipulate, and share data to get answers and promote solutions.
- ▶ Needn't authorize money or create programs: just solve problems.

This is an Amazing Moment to Learn

1. Software is easier and more powerful than ever.
 - ▶ Mapping, internet, etc.: making awesome stuff has never been easier.
2. Governments are getting on board...
 - ▶ 'One size fits all' bureaucracy doesn't cut it.
 - ▶ Modern interface for services.
 - ▶ Target interventions (money) where it's most needed.
3. And they're learning to share their data.
 - ▶ Most states have some data portal presence; many are very good.

Coding for Public Policy

Boston 311 System: Improve Constituent Services



New Report

Recent

My Reports

Favorites

Trash Alerts

Reporter

How can we help?



Location

Description

Has pole



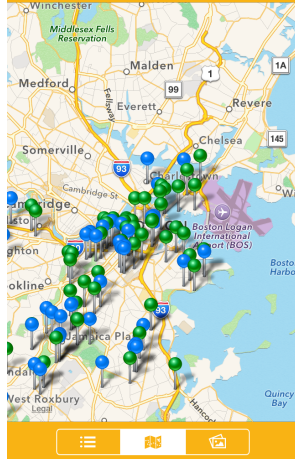
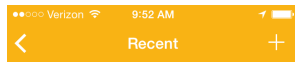
Share with public



Reporter

John Doe

Reporter info will not be shared with public

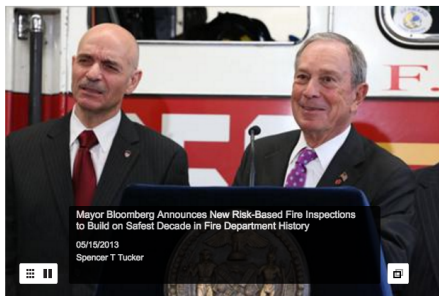


Mapping Trees in New York: Make Something Beautiful



jillhubley.com/project/nyctrees

Prioritizing Building Inspections: Public Safety



Mayor Bloomberg And Fire Commissioner Cassano Announce New Risk-based Fire Inspections Citywide Based On Data Mined From City Records

May 15, 2013

Mayor's Office of Data Analytics Validates and Improves Effectiveness of System

Mayor Michael R. Bloomberg, Deputy Mayor for Operations Cas Holloway, Fire Commissioner Salvatore J. Cassano, Chief Policy Advisor John Feinblatt, Chief Analytics Officer Michael Flowers and Chief Information and Innovation Officer Rahul N. Merchant today announced that firefighters citywide are now using new technology

Policing: Intervene with 'At-Risk' Officer

U. of C. researchers use data to predict police misconduct



Rayid Ghani, director of the Center for Data Science & Public Policy, said police departments can benefit from programs that use big data to, among other things, predict an officer's adverse reaction to a citizen. (Antonio Perez / Chicago Tribune)

By **Ted Gregory** · Contact Reporter
Chicago Tribune

AUGUST 18, 2016, 6:45 AM

In case you missed it



Judgment day for Chicago's police code of silence

AUG. 17, 2016



Timeline: Chicago police controversies during Mayor Rahm Emanuel's administration

SEP. 16, 2016



More stories: Chicago's Cop Crisis

AUG. 26, 2016

Poverty by Census Tract: Visualize the World

- ▶ See Jupyter Notebook, `census/Census.ipynb`.

How the Class is Structured

What We Will and Won't Cover

- ▶ Thinking algorithmically with python: coding.
 - ▶ Building blocks of code from the ground up.
 - ▶ First-pass of low-level tools: the command line.
 - ▶ Fundamentals of databases and the web.
-
- ▶ Higher-level analysis 'recipes.'
 - ▶ Build your own projects from large, free components.
 - ▶ Wrangle data to get and share information, and create solutions.
-
- ▶ However: not a management or policy course.

What We Will and Won't Cover

- ▶ Thinking algorithmically with python: coding.
 - ▶ Building blocks of code from the ground up.
 - ▶ First-pass of low-level tools: the command line.
 - ▶ Fundamentals of databases and the web.
-

- ▶ Higher-level analysis 'recipes.'
 - ▶ Build your own projects from large, free, common data sets.
 - ▶ Wrangle data to get and share information, and create solutions.
-

- ▶ However: not a management or policy course.

Assignments

- ▶ Weekly assignments posted on the class [Github site](#).
- ▶ You will also submit them through 'Github.'
 - ▶ Covered later today. It is the 'standard' for collaboratively developing and publishing code.
 - ▶ If you have trouble with git/Github, speak up fast!
- ▶ Collaborative, large scale final project for exam. I will provide templates, and you will propose the project.

Yes, some group work.

Real projects require collaboration.

- ▶ Developers use git to work collaboratively. Using git is a skill.
 - ▶ You don't need to be in the same place, or even talk to your partner.
- ▶ Even better: learn to contribute to the open source community.

Google

group assignments are



group assignments are **the worst**
why group assignments are **important**

Press Enter to search.

Additional Resources

- ▶ TAs to be finalized in the next 24 hours.
- ▶ We will host a 'clinic' (discussion session) for 2-3 hours, starting from 16h on Thursdays or Fridays. **Please come!**
- ▶ Use the class Piazza discussion board.
 - ▶ First item of business: vote for the tea time that fits your schedule!
- ▶ New class: **Please** tell me (anonymously, if you prefer) if a lecture or assignment is confusing, or if you're frustrated.
- ▶ And please ask questions as we go.

Welcome to the Command Line

This will be different: hang in there.

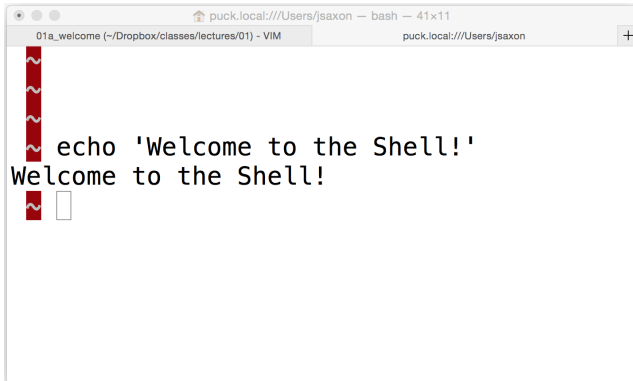
- ▶ This may be one of the hardest lectures in the entire class.
- ▶ We are doing it now, because everything else depends on it.
- ▶ We'll list and examine a bunch of 'commands' (programs/words).
- ▶ Just understand the 'meaning' – don't worry about memorizing these.
- ▶ Learning to string them together will take practice (HW!).

Why Use The Command Line?

- ▶ Making Graphical User Interfaces (GUIs) is tricky and unenlightening.
- ▶ GUIs are often not extensible or scriptable: doing something fifty times usually requires the user to actually click around, fifty times!
 - ▶ Insane! Computers are good at repetition!
- ▶ For computation, we're better off running programs from the command line.
- ▶ Immediately available on Mac OS X. On Windows, cygwin provides a Unix-like interface.

Approaching the Command Line

- ▶ Please open Terminal (Mac) or Cygwin (Windows).
- ▶ If you have not yet installed this (Windows), you can use tmpnb.org.
 - ▶ This is not the real McCoy, and you will need cygwin very soon!



```
puck.local:///Users/jsaxon — bash — 41x11
01a_welcome (~/.Dropbox/classes/lectures/01) - VIM    puck.local:///Users/jsaxon +
echo 'Welcome to the Shell!'
Welcome to the Shell!
█
```

The Fundamental Commands

- ▶ **pwd**: print working directory
- ▶ **ls**: list (files and folders)
- ▶ **cd**: change directory
- ▶ **mkdir**: create a directory
- ▶ **mv**: move or rename a file
- ▶ **touch**: create an empty file
- ▶ **cp**: copy a file or folder
- ▶ **rm(dir)**: remove a file (directory)
- ▶ **man**: read the 'manual'

- ▶ **ssh/scp**: secure connections (not in this course)

Notes on the Directory Structure

- ▶ `'.'`: means this directory
 - ▶ `'..'`: means one directory higher
 - ▶ `'~'`: means 'my home directory'
 - ▶ `'-'`: the last directory I was in.
 - ▶ `'/'`: is 'root'
-
- ▶ `*`: is a 'wildcard' (match anything or nothing)
-
- ▶ In **cygwin**, your 'normal' hard drive lives at `/cygdrive/c/`.

Going Deeper: The Commands

Tremendous power 'built in' to the command line: quickly compose programs to get answers.

- ▶ **echo**: parrot back some text
- ▶ **curl/wget**: retrieving web resources.
- ▶ **cat, head, tail**: 'concatenate' (dump) a file, or part of it
- ▶ **less**: page through a file
- ▶ **grep**: search for lines in a file
- ▶ **sed**: find and replace
- ▶ **wc**: count words or lines in file
- ▶ **sort**: sort a file
- ▶ **cut**: choose out specific columns
- ▶ **uniq**: with `-c`, count occurrences of a unique line.

Going Further: Piping and Scripting

- ▶ The power of the command line comes from the ability to quickly compose programs from these building blocks.
- ▶ There are two important 'connectors' to know:
 - | **pipe**: forward the output to the next command.
 - > **redirect output**: write to a file
- ▶ You may sometimes see these as well:
 - >> **redirect output**: append to a file.
 - < **redirect input**: feed in to command.
 - << **X read input**: read from the command line 'until X' (uncommon)

- ▶ echo just parrots everything that follows it:

```
■ echo hello world.  
hello world.
```

- ▶ You could easily use this to write to a file... not like this!

```
■ echo Hello (and happy birthday) > bd  
-bash: syntax error near unexpected token '('
```

- ▶ 'Special' characters (!, , \$, (,), etc.) need to be enclosed in quotes:

```
■ echo "Hello (and happy birthday)" > bd
```


- ▶ curl and wget retrieve a web-page or other net resource [\[link\]](#):

```
wget data.cityofchicago.org/api/views/xzkq-xp2w/rows.csv
```

```
curl data.cityofchicago.org/api/views/xzkq-xp2w/rows.csv  
-s -o salaries.csv
```

- ▶ The wget '-O' and curl '-o' 'options' allow you to specify and output file name for the download.
 - ▶ And -s stands for 'silent' – see the man pages.

Please do the curl command.

cat/head/tail/less

- ▶ cat dumps a file to the screen:

```
■ cat salaries.csv
```

- ▶ For very large files, better to 'page through it', or check the beginning or end:

```
■ less salaries.csv
```

```
■ head -42 salaries.csv # first 42 lines
```

```
■ tail -12 salaries.csv # last 12 lines
```

- ▶ With << X, one could write a small script ... uncommon.

- ▶ grep is a filter. It finds all matching lines in a file.

```
■ grep EMANUEL salaries.csv
```

- ▶ You can also 'reverse grep' with '-V.'
- ▶ **grep is my favorite command.** I hope you will enjoy it too!

grep [2 of 3]: Regular Expression Special Characters

Regular expressions (regex) is a shorthand, for complex pattern matches.

- Dramatically expands potential of grep.

<code>^</code>	Beginning of the line.
<code>\$</code>	End of line.
<code>\</code>	Turn off the next special character.
<code>[]</code>	Any <i>contained</i> characters; use 'x-y' for range.
<code>[^]</code>	None of contained characters.
<code>.</code>	Any single character.
<code>*</code>	The preceding character/expression, any number of times.
<code>\{x\}</code>	The preceding, x times.
<code>x y</code>	x OR y.

A bit quirky at first, but super useful!

grep [3 of 3]: Applying Regular Expressions

- ▶ How much does the mayor make?

```
■ grep '^\"EMA' salaries.csv
```

- ▶ Who makes more than \$200k?

```
■ grep '\$[2-9][0-9]\{5\}\.' salaries.csv
```

wc: word count

- ▶ wc allows you to count the number of bytes (-c), number of words (-w) or number of lines (-l) in a file:

```
■ wc -l salaries.csv # by far the most useful
```

- ▶ How many police officers are on the streets of Chicago?

```
■ grep -i "police officer" salaries.csv | wc -l
```

- ▶ How many of them are detectives?

```
■ grep "POLICE.*DETECTIVE" salaries.csv | wc -l
```

- ▶ sed allows for simple, regex find and replace
- ▶ If you learn vim it is the same syntax:

```
■ sed 's/find/replace/g' salaries.csv
```

- ▶ Here, the s means 'search' and the g means global/all occurrences in a line. For instance, remove the \$ signs:

```
■ grep '\$' salaries.csv | sed 's/\$//g'
```

- ▶ Technical warning: it's single quotes, here. With double quotes, you'd need to 'escape' the \$ twice: for the command line and regex.

- ▶ sort sorts your file, with many options.

Find the 10 highest salaries in the city.

man: `-k` for key field, `-t` for delimiter, `-r` for reverse, `-n` numeric.

- ▶ Modify the sort command here:

```
■ grep '\$' salaries.csv | sed 's/\$/g' | sort | head
```


cut (penultimate!)

- ▶ cut allows you to choose which columns to print.
- ▶ Really needs the `-d` (delimiter) and `-f` (field) options.
- ▶ To print only the first names in the city:

```
■ cut -f2 -d, salaries.csv
```

- ▶ i.e., the second field, delimited by commas.

uniq (last one!)

- ▶ uniq prints unique lines: subsequent duplicates are removed.
- ▶ It is most-often used after sort, and with -c option to count.
- ▶ To count the occurrences of each last name:

```
■ cut -d, -f1 salaries.csv | sort | uniq -c
```

That was a lot!

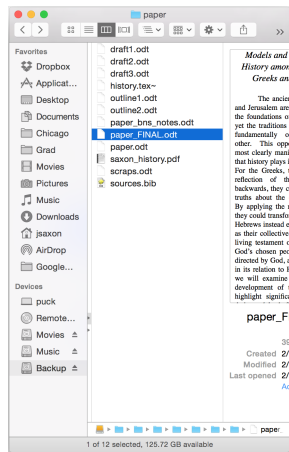
- ▶ You will need the navigation commands all the time.
- ▶ The text analysis commands are really useful for getting fast answers.
 - ▶ You will get more familiar with them in homework.
 - ▶ But the rest of the course will not depend on these.
- ▶ It should slow down a bit now.

Version Control: Git

What is version control? Why use it?

- ▶ Perhaps a familiar story, for paper drafts. \implies
- ▶ What if several people need to be able to edit simultaneously.
- ▶ What if there are many different files that depend on each other being at a specific version, all of which may be changed?

**Version Control Systems
maintain a history and
facilitate collaborative editing.**



What is git? GitHub?

- ▶ Git is the modern VCS, designed by Linus Torvalds (creator of Linux).
- ▶ Git maintains a history of meaningful 'commits.'
 - ▶ It is tremendously flexible ('branches').
- ▶ Git is distributed: everyone has a copy of the entire history.
- ▶ However, it is often useful to maintain a master copy on a server where anyone can access it or 'push' their changes: GitHub.
- ▶ GitHub is a nice **interface** for hosting repositories.



You'll use these regularly:

- ▶ **git init**: create a repository in this directory
- ▶ **git clone**: download repository
- ▶ **git add**: add a file to 'staging' area
- ▶ **git status**: view status of all files
- ▶ **git commit**: commit staged files to history
- ▶ **git push**: upload all changes to a remote server
- ▶ **git log**: show the history

Start with a single user and a single thread of edits:

1. Download your homework skeleton:
 - ▶ `git clone git@github.com:harris-ipp/01-welcome.git`
2. Make your edits with Atom or vim.
3. Add files to the 'staging' area, and commit them; check the status and log to see that it worked:
 - ▶ `git add q1.py`
 - ▶ `git status` # is everything there?
 - ▶ `git commit -m "started question 1"`
 - ▶ `git log` # now all part of the commit history?
4. Upload it to the server:
 - ▶ `git push`

Then repeat steps 2-4 as you go.

Start with a single user and a single thread of edits:

1. Download your homework skeleton:
 - ▶ `git clone git@github.com:harris-ipp/01-welcome.git`
2. Make your edits with Atom or vim.
3. Add files to the 'staging' area, and commit them; check the status and log to see that it worked:
 - ▶ `git add q1.py`
 - ▶ `git status` # is everything there?
 - ▶ `git commit -m "started question 1"`
 - ▶ `git log` # now all part of the commit history?
4. Upload it to the server:
 - ▶ `git push`

Then repeat steps 2-4 as you go.

This is what you'll use regularly.

Next time:



pythonTM