Welcome: to the Class and to the Command Line!

Jamie Saxon

Introduction to Programming for Public Policy

September 25, 2017

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.

3. And it's fun! Like learning a language.

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...
   - Target interventions (money) where it's most needed.
   - Modern interface for services.
   - 'One size fits all' bureaucracy doesn't cut it.
3. And they're learning to share their data.
   - Most states have some data portal presence; many are very good.

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...
   - Target interventions (money) where it's most needed.
   - Modern interface for services.
   - 'One size fits all' bureaucracy doesn't cut it.
3. And they're learning to share their data.
   - Most states have some data portal presence; a few are very good.

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...
   - Target interventions (money) where it's most needed.
   - Modern interface for services.
   - 'One size fits all' bureaucracy doesn't cut it.
3. And they're learning to share their data.
   - Most states have some data portal presence; a few are passable.

Coding for Public Policy

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

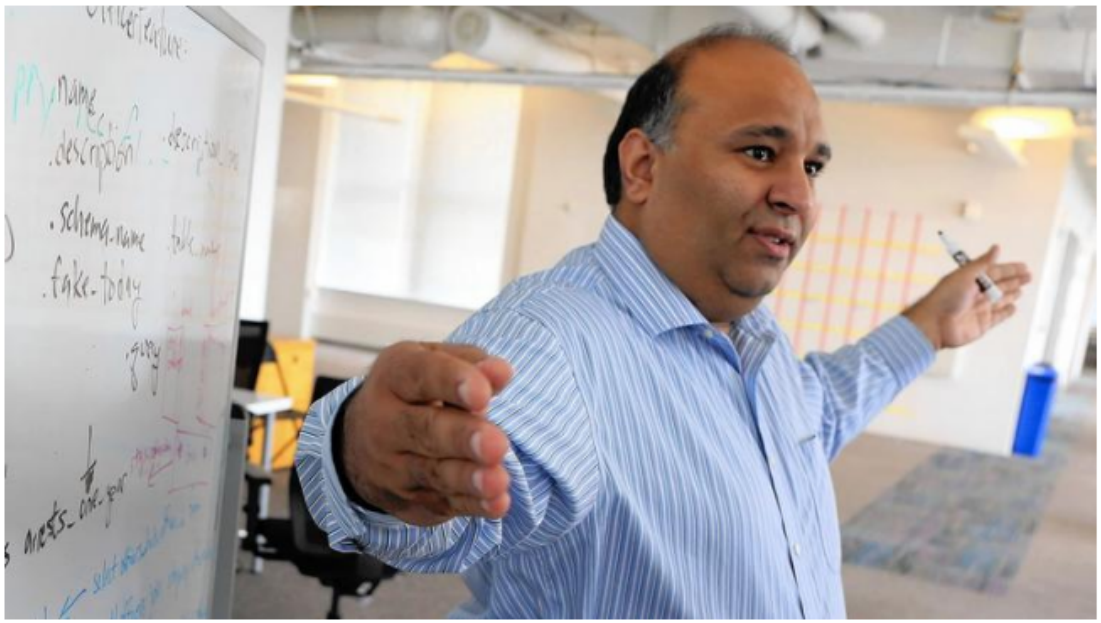
Mayor Michael R. Bloomberg, Deputy Mayor for Operations Cas Holloway, Fire Commissioner Salvatore J. Cassano, Chief Policy Advisor John Feinblatt, Chief Analytics Officer Michael Flowers and Chief Information and Innovation Officer Rahul N. Merchant today announced that firefighters citywide are now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

Policing: Intervene with 'At-Risk' Officer

SECTIONS SEARCH **Chicago Tribune** SUBSCRIBE 4 WEEKS FOR 99¢ LOG IN

SATURDAY SEP. 24, 2016 BREAKING SPORTS TRENDING OPINION SUBURBS ENTERTAINMENT BUSINESS ADVERTISING 73°

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

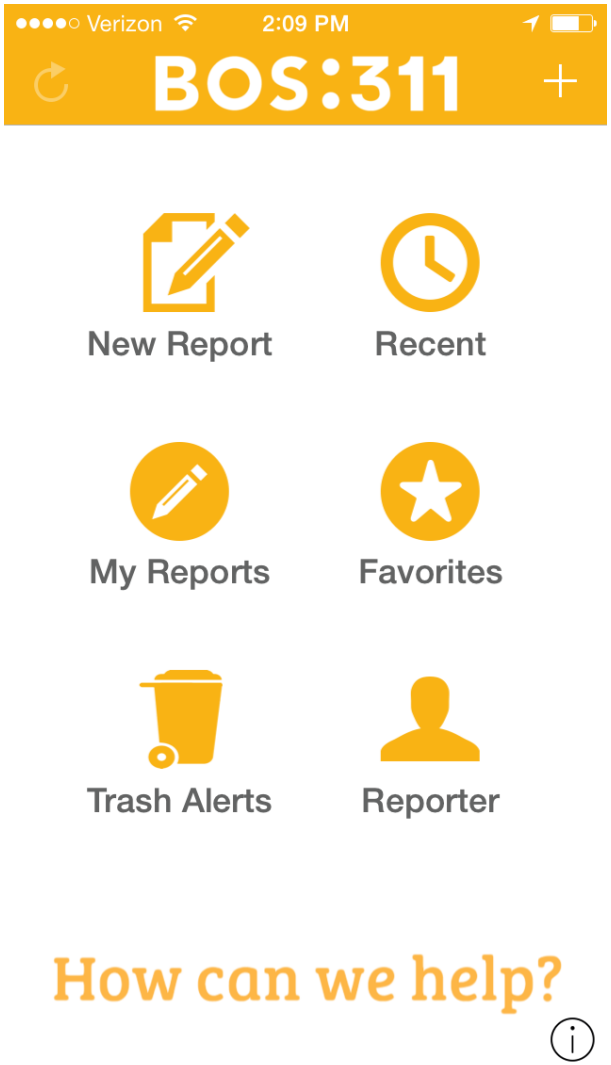
In case you missed it

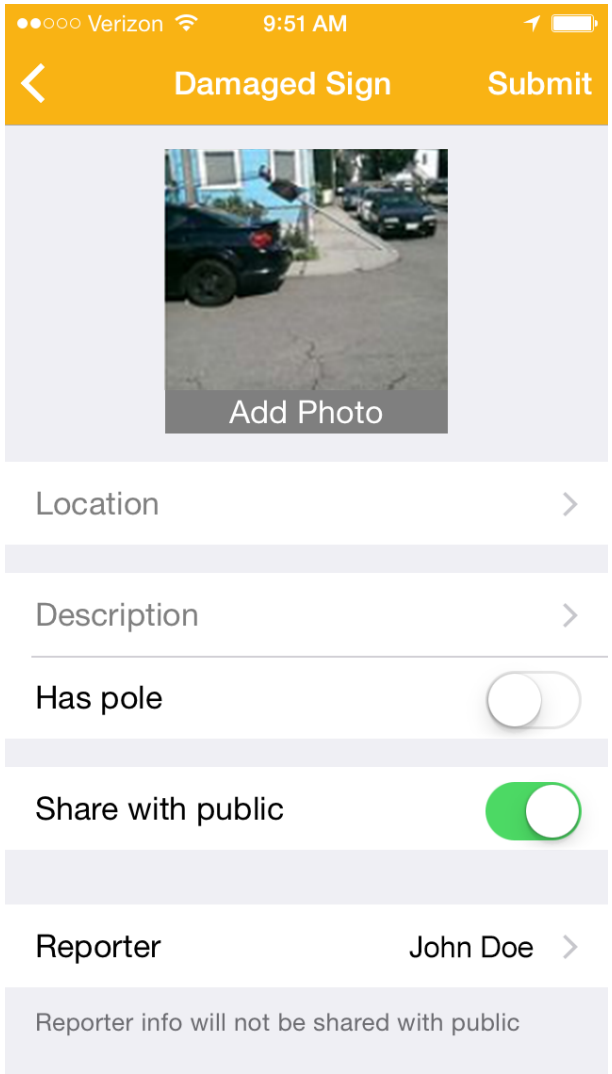
Judgment day for Chicago's police code of silence
AUG. 17, 2016

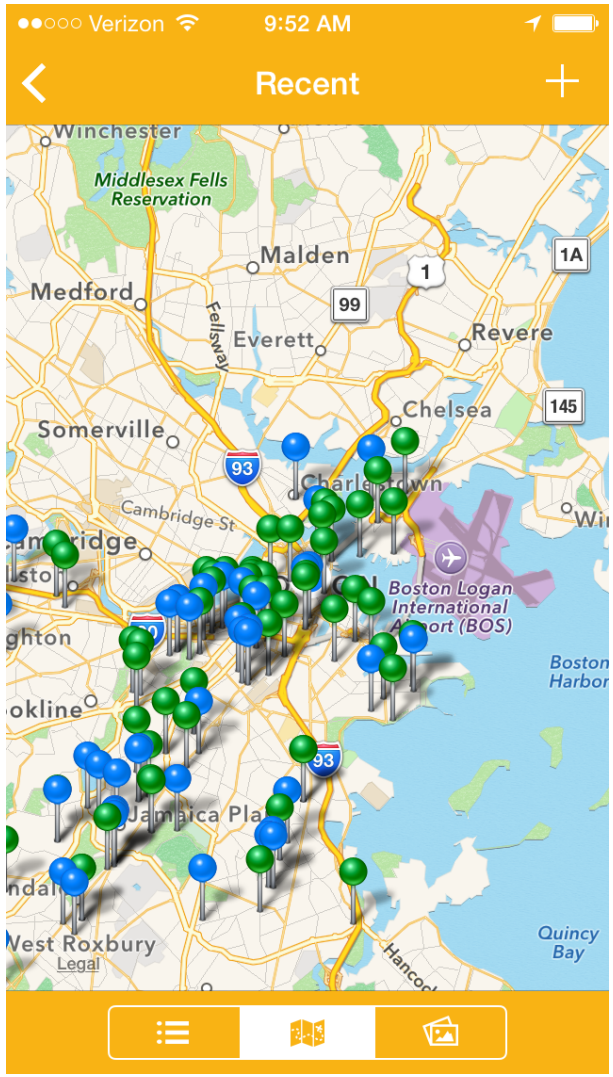
Timeline: Chicago police controversies during Mayor Rahm Emanuel's administration
SEP. 16, 2016

More stories: Chicago's Cop Crisis
AUG. 26, 2016

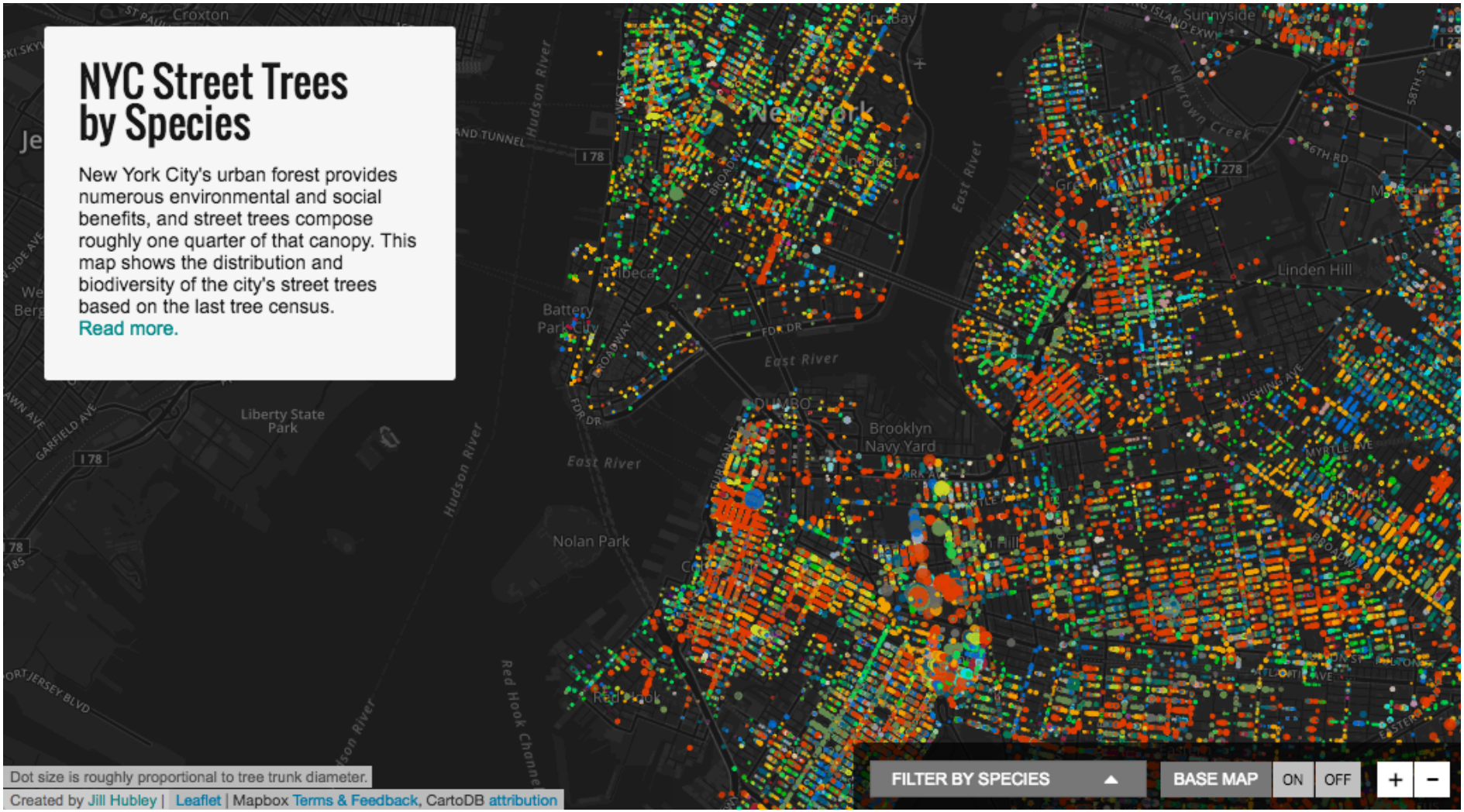
Boston 311 System: Improve Constituent Services







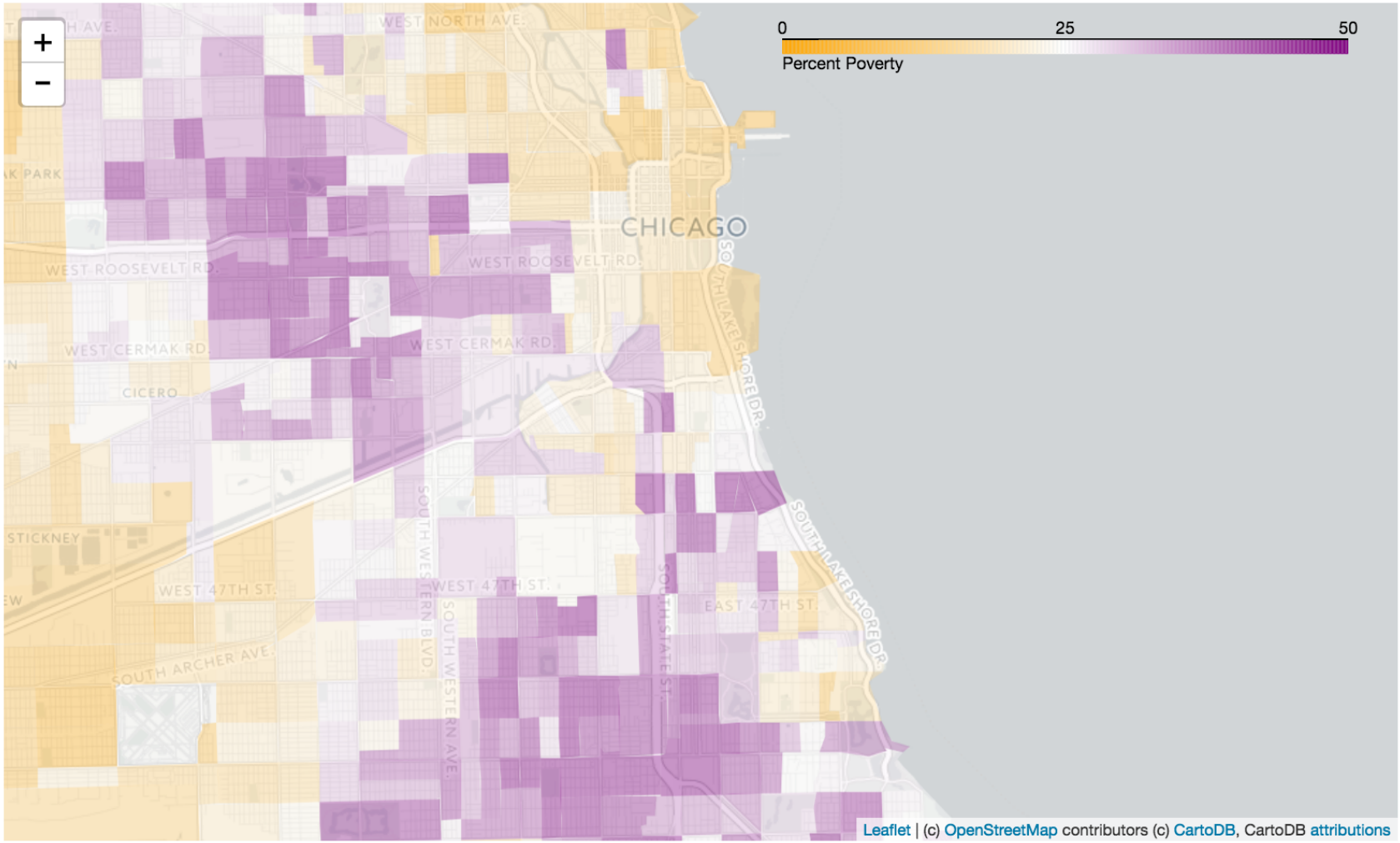
Mapping Trees in New York: Make Something Beautiful



jillhubley.com/project/nyctrees

Poverty by Census Tract: Visualize the World

See Jupyter Notebook, `ex/Census.ipynb`.



How the Class is Structured

What We Will and Won't Cover

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

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

≈ 5 weeks

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

≈ 5 weeks

- However: not a management or policy course.

Assignments

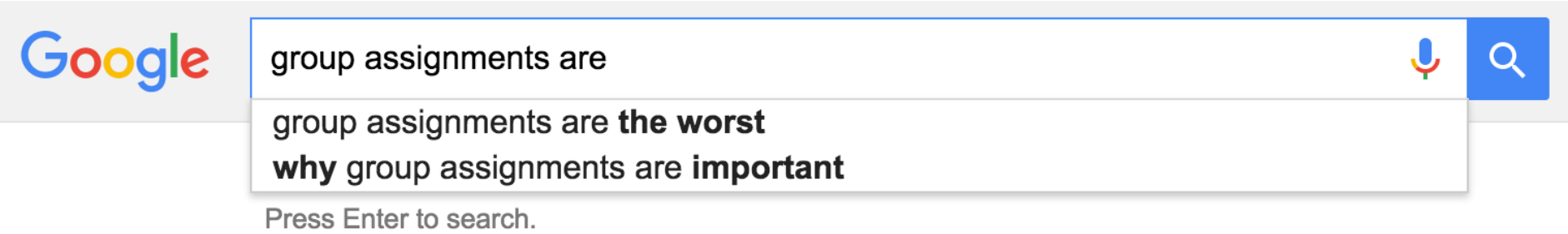
- Weekly assignments (70% of grade) posted on the class **Github site**.
- You will also submit them through 'Github.'
  - Covered Wednesday. It is the 'standard' for collaboratively developing and publishing code.
  - If you have trouble with git/Github, speak up fast!
- Collaborative (2 or 3 person) final project (25%). Start thinking about this now! Look for data on government and non-profit sites, in corporate APIs – anywhere!

- Discussion board & participation (5%): good questions and useful answers both get "points."

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.



Additional Resources

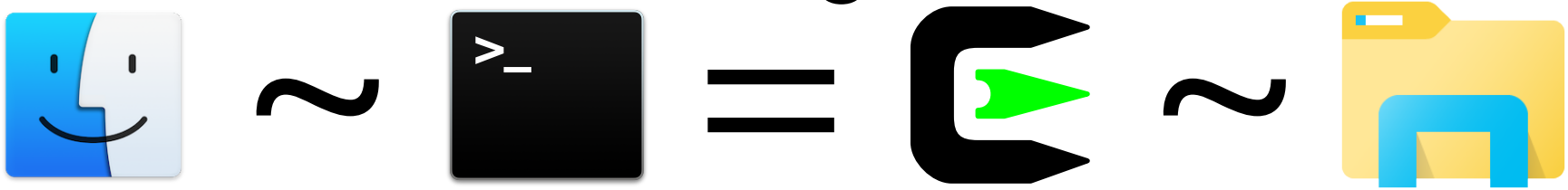
- Four fabulous TAs (2nd-year MSCAPP):
  - Victor Vilchis Tella
  - Emma Peterson
  - Andrew Yaspan
  - Yuxi Wu
- TA sessions in 289A on Thursday and Friday afternoons at 3:30-5pm (October 6 in 224).
- Use the class discussion threads on the **Canvas** page.
- All lecture slides and notebooks stored here – follow & bookmark!

  github.com/harris-ippp/lectures/

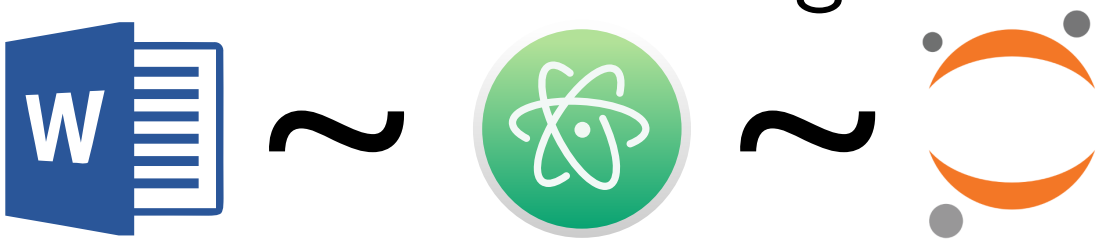
- And **please** (**!!**) ask questions as we go.

Software

File Navigation



Text Editing



Language Interpreters

Welcome to the Command Line

This will be different: hang in there.

- This may be one of the harder lectures – a steep learning curve.
- We are doing it now, because everything else depends on it.

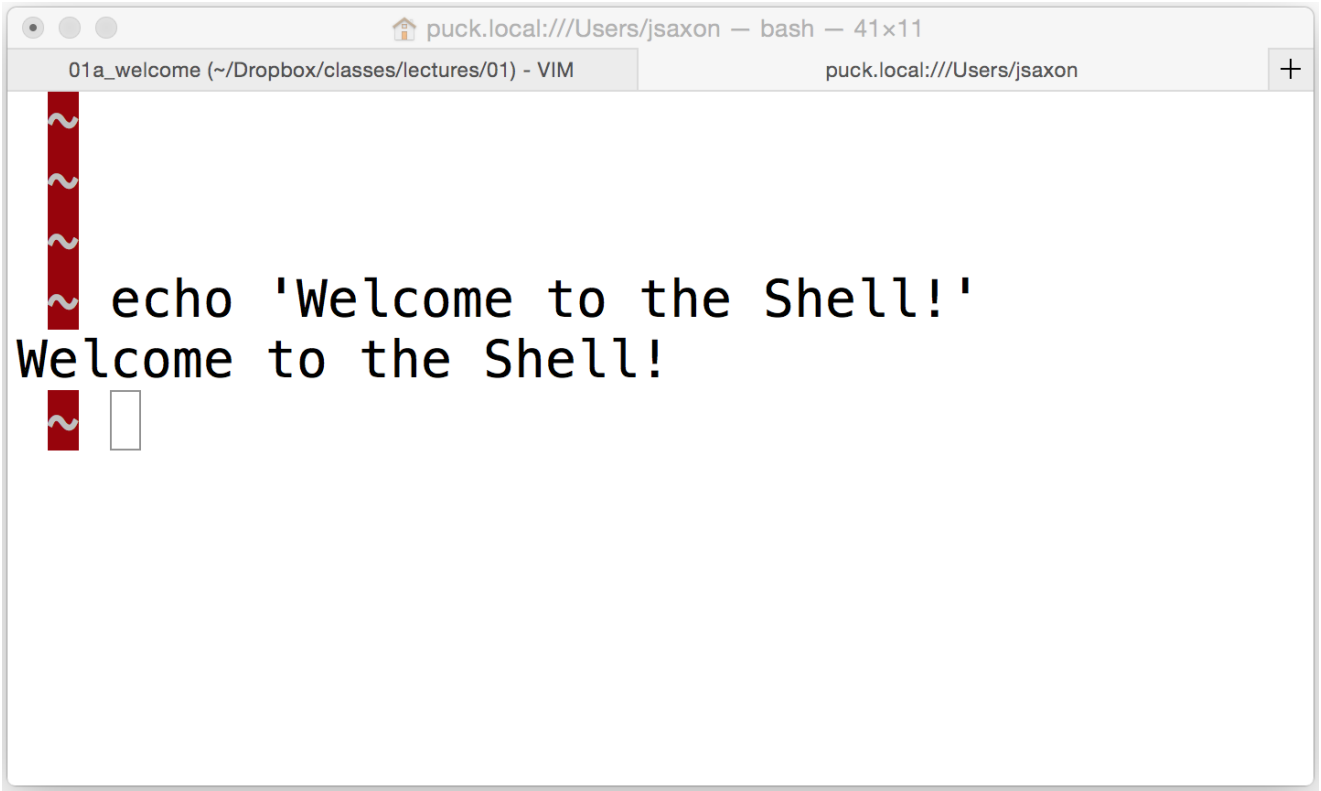
- We'll list and examine a 'dictionary' of commands (programs/words):
  1. Navigation
  2. File and text manipulation
- Think of today as a 'dictionary': understand the 'meaning,' and look them up later.
- Learning to string them together will take practice (HW!).

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!
- So we're better off running programs from the command line.
- Immediately available on Mac OS X. On Windows, cygwin provides a Unix-like interface.
  - You should already have installed it!!

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!



Navigation Commands

- **pwd**: print working directory
- **ls**: list (files and folders)
- **cd**: change directory
- **mkdir**: create a directory
- **mv**: move or rename a file
- **cp**: copy a file or folder
- **rm(dir)**: remove a file (directory) – **¡¡Careful!!**
- **man**: read the 'manual'
- **chmod**: change the 'permissions' of a file.

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

Notes on the Directory Structure

In cygwin, your C drive is at /cygdrive/c/.

- '.': 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)

Exercise: find, list, and open music or a doc.

Manipulating Output

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

- **echo**: parrot back some text
- **curl**: 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 occurences of a unique line.
- **python**: much on this, later!

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 import '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 ithe command line 'until X' (uncommon)

echo

- `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 (or wget)

- `curl` retrieves a web-page or other net resource **[link]**:

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

- The '-o' option allows 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 [1 of 3]

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

```
grep EMANUEL salaries.csv
```

- Use the `man` pages to figure how to reverse grep, or ignore case.
- **grep is my favorite command.** I hope you will enjoy it too!

grep [2 of 3]: Regular Expresion Special Characters

Regular expressions (regex) are shorthand for complex matching.

- Dramatically expands potential of grep.

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

- 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

- 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
```

- The **answer** – don't peak!

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 use the navigation commands all the time.
- The text analysis commands are super useful for 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 dwn a bit now.

Next time: scripts.