JMS 302: HACKING THE MEDIA

CLASS 1: 8/19/2014

AGENDA

- Administrivia, Overview
- How the Web Works
- Setup, Your First Programs

Roll, handout syllabus, warmup exercise

NORMS

STANDARDS TO HELP EACH OTHER SUCCEED

DEVICES CLOSED

WRITE THESE IN YOUR OWN WORDS

1. **Be fully present**
2. Be prepared for class
3. Open/use electronics only when explicitly allowed (but login and setup before class)
4. Arrive on time
5. Don't be afraid to ask questions—expect some frustration and failure, as well as triumph and success
6. Experiment, guess and test
7. Be precise
8. Make comprehension/learning your goal
9. The lab is for JMS work only. **No food or drink is allowed in the lab**. If you login to personal accounts (gmail, etc.), be sure to logout when you leave. Never configure software (Mac Mail, iTunes, etc.) with your login unless explicitly authorized (e.g., GitHub app).

WHAT IS HACKING?

NOT THIS

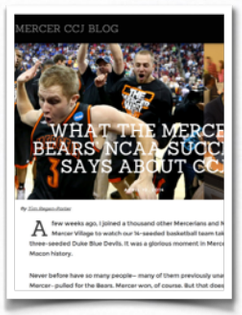
CLOSER TO THIS

Definition
- One definition of hacking: Use a computer to gain unauthorized access to data in a system.
- Our definition: to “manage and cope” and “to cut with rough or heavy blows”—specifically, to explore & delight in the workings of computers and stretch the capabilities of available software and hardware

Hierarchy of related studies
- Computer Science / Software Engineering - solving problems efficiently and effectively with the limited resources found in a computer is the real goal. CS: algorithms, languages, hardware architecture, systems. SE: quality and system for creating, software lifecycle
- Programming: Writing programs to accomplish some goal. More likely to be self-taught. Practical (but may be inefficient).
- Hacking: Applying creativity and brute force to existing programs to accomplish some goal. Efficiency is not the goal. Writing whole new programs is not the goal. A hacker could be a newbie or a computer scientist (outside of her expertise or unconcerned with efficiency for some reason).
    - BUT, hacking requires understanding some base level of understanding of programming.

In all three cases, you tell your computer to how do something—something that it doesn’t know how to do, using what it does know how to do. Thus we must start by (partially) answering, “What does a computer know how to do?” [Writing and language analogy]

EXAMPLES

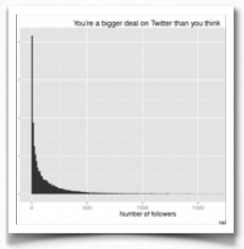


CCJ BLOG

Add bylines

TELEGRAPH

Grab e-edition, data



O'REILLY

Twitter Analysis

NYT

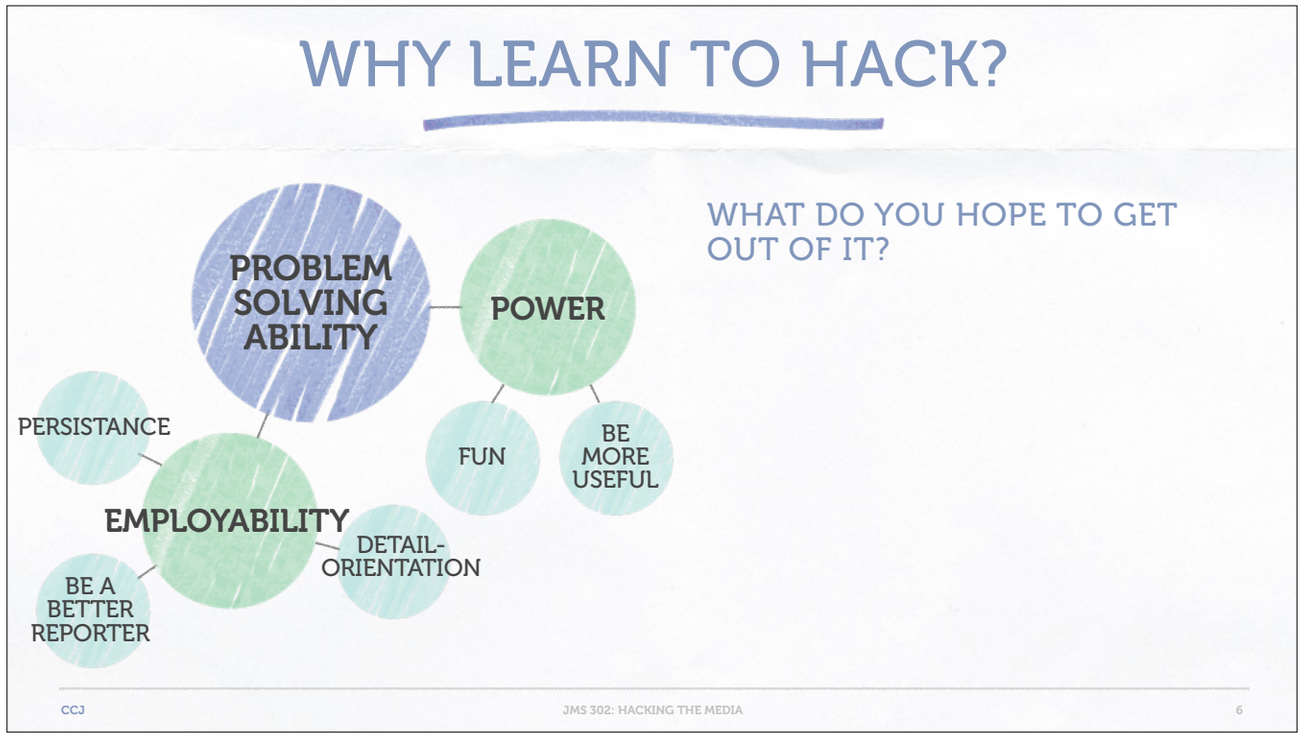
Dialect Quiz

NYT

Snowfall

1. The WordPress theme that CCJ bought did not have bylines—it was intended for a single author. That's bad for a group blog. The simple solution (but not ideal) was to manually add using HTML & CSS. Demo
2. Utility scripts to help company or make your life easier—e.g., a script to pull the e-edition from the website at 2 a.m. and post it to Facebook (maybe multiple times) or a script to pull police reports or election results and analyze them.
3. Scripts to do complex analysis: Jon Bruner sampled 400,000 Twitter accounts to look at follows and followers. Alexis Madrigal looked at how Netflix uses almost 77,000 genres to give you recommendations, which also nabbed him an interview with a Netflix VP. These scripts wouldn't have been possible without coding.
    - See also (from "Scoped by code")
        - A story from Al Jazeera America analyzing stolen user accounts looking for common insecure passwords, written by a recent MIT grad.
        - A story about online retailers that secretly offer different prices to people based on where they live, reported by a team that includes two programmers.
        - A story — and interactive database — that involved writing software to detect self-censorship on a popular Chinese social networking service, and then showing readers images that were deleted.
4. NYT dialog quiz
    - Most-visited NYT story of 2013, in just 11 days
    - Created by an intern
    - Used quiz to verify and update data from Harvard Dialect Survey
        - "Finally, the examples I've given show how compelling, fascinating stories can be created by one or two journalists coding scripts and building databases, including automating the work of data collection or cleaning…. There's one reason (among many) that ProPublica can win Pulitzers prizes without employing hundreds of staff."
5. Eye candy / Storytelling / Interactive Guides
    - http://www.google.com/doodles/john-venns-180th-birthday
    - http://projects.propublica.org/nursing-homes/state/GA
    - http://apps.npr.org/fire-forecast/

See "Of scripts, scraping and quizzes: how data journalism creates scoops and audiences"



Problem solving: try different approaches, don't fear the unknown, examine the feedback you get, make use of available resources

Programming / Hacking = creative problem solving

Detail orientation: every little thing may matter (quotation marks, commas, capitalization, sometimes white space). Read last bullet from "Class Preparation" section of syllabus.

OUR APPROACH

- Languages
- Exploratory
- Collaborative

Language: Ruby, HTML, CSS & JavaScript.
- From that list, only Ruby & JavaScript are “true” programming languages.
- In general, which language you learn is not paramount; it’s most important to learn a language.
- See “Learning Outcomes” in syllabus

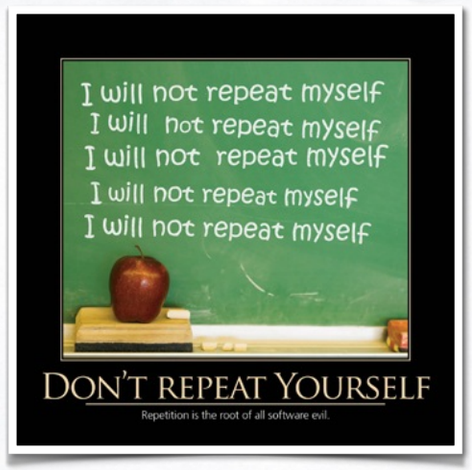
Exploratory:
- Read 2nd & 3rd bullets in “Class Preparation” section of syllabus
- Even the best never master everything. You need to have a foundational knowledge and then know how to quickly either find help or try solutions using your intuition (something Ruby is great at enabling).

Collaborative
- Tests are “summative”—to demonstrate how much **you** know. Use everything else to learn. [A word on projects.]
- Peer program. Use the Internet. Use your books (one of which contains the answers to its exercises). Use the class Facebook group. Use your professor. But struggle on your own first—that’s where the value is. YOU are responsible for understanding what’s going on and being able to do it on your own.

PROGRAMS ARE GROWN

A program is grown. Like a good feature, it doesn't just pop into your head, intact. It must be planned, prototyped, tested, extended, rewritten, etc. Start small—do one thing, test, add another step, test, etc.

GOOD CODE IS DRY



DRY: Don’t Repeat Yourself

Cultivated laziness: spend one hour doing a 10-minute task, if you’ll do that task 10 more times

COMPUTERS ARE DUMB

ROBOT DIRECTIONS

TO THE ROOKERY

YOUR TOOLKIT

- Editor
- Terminal
- Code Repository
- Browser

Editor
You can edit most code with any text editor, but special editors can make you more productive.
- Basic code editors add things like project organization and syntax highlighting [show].
- IDEs (Integrated Development Environments) contain a host of tools to run, debug and profile code. They usually have code completion.

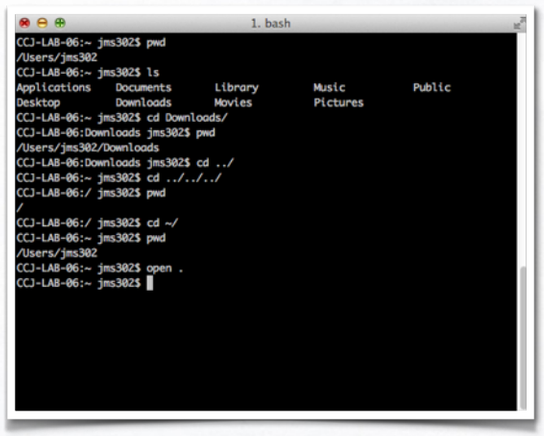
- We will be using Brackets most of the time. It's somewhere between the two, and free. Other options you might want to look at down the road: TextMate, TextWrangler, Sublime Text, CodeAnywhere.net, etc. Professionals with a Linux bent often use vim or Emacs.
- Later, we may use Aptanta Studio 3 or RubyMine for full-fledged IDE functionality. Other IDEs you might want to look into down the road: Eclipse, Xcode, Netbeans.

Terminal
- Although our editor can sometimes remove the need to use the command line, you need to know how to get around the command line.
- You can always use the built-in Terminal app (in Applications > Utilities).
- We'll use iTerm (in your dock and in Applications). The differences are minor. One feature you might want to use is "Step Back in Time" (notice how I use Help to find menu options).
- If you want a good CLI (command line interface) for Windows, install Cygwin (which can be tricky due to all of the packages).

Code Repository
- Good programmers use a code repository to manage their code. It provides backup (if external), revision control (you can look at what you did in the past, undo, etc.), allow teams to work on the same files, fork and merge branches (to work on special features without disrupting the team), etc.
- We'll use GitHub(.com)—based on git—as a service, and GitHub [the app] to communicate with it. We might need to use the git commands on the command line later.

Browser
- We'll use Google Chrome, primarily—including its Developer Tools (under View).
- Good programmers test their work in multiple browsers (Safari—which should be very similar to Chrome, Firefox, IE) on multiple OSes and devices.

THE COMMAND LINE



GETTING AROUND THE TERMINAL

- Launch iTerm (in dock, also under Applications)
  - Get use to Finder
- Notice everything before cursor
  - computer name, directory, user, command prompt
- Follow me…
  - Notice where we start

Command line may appear different on different computers.

Key commands
Note: capitalization matters

- `pwd`: **p**rint **w**orking **d**irectory
- `ls`: **l**i**s**t files in a directory (current directory by default
  - flags: add letters, and sometimes words, after commands to get more information or different behavior. Order doesn't usually matter.
    - `ls -l`: detai**l**ed listing
    - `ls -al`: detai**l**ed listing of **a**ll files (including "hidden" files)
    - `ls -lSr`: detai**l**ed listing, by **S**ize, in **r**everse order
- `man`: get **man**ual page on a command (e.g., `man ls`)
- locations:
  - `.` (or `./`): current directory (also used to run an executable file in current directory)
  - `..` (or `../`): up one directory
  - `~`: your directory (also `~username/`)
- `cd`: **c**hange **d**irectory
  - type the following and observe:
    - `cd Downloads`
    - `ls ~/`
    - `pwd`
    - `cd ..`
    - `pwd`
    - `cd ../../`
    - `pwd`
    - `cd /Library`
    - `pwd`
    - `cd`
    - `pwd`
  - note: tab completion
- `mkdir`: **m**a**k**e **dir**ectory
- `more`: display contents of a file (paginated for you)
- `open`: opens file/directory (context specific)
- `!!`: redo previous command
- `history`: see your command history
- `!<number>`: execute command <number>
- `Ctrl + U`: Clear the line

- other useful commands: `less`, `cat`, `nano`, `cp`, `mv`, `rm` (warning), `|`, `>`, `grep`, `find`, `locate`, `ack`, `top`, `kill` (warning), `ps aux`