JavaScript on the Command Line

Learning Objectives

- Differentiate between the Internet and the World Wide Web.
- Summarize the client-server model and explain how a DNS Lookup works.
- Use the most common commands to navigate and modify files / directories via the terminal window.
- Run basic JavaScript script on the command line using Node.

Previously on JavaScript Development:

- Installed and configured Node.js, git, other tools
- Wrote pseudocode and to think programmatically
- Before class, should be able to use HTML and CSS to create static web pages. If you need a refresher, try Dash (https://dash.generalassemb.ly).

JavaScript!

- Current uses include:
  - Front-end web programming (in browsers)
  - Back-end web programming (on the Node.js platform and others)
  - Various miscellaneous non-web uses (scripting of Adobe Photoshop, for example)
  - We're going to concentrate on web, mostly front end but a little Node

Web Development

- **Web Development:** The process of building sites and applications for the web.
- **Front-End Web Development:** The development of code that runs on the client, i.e. in the browser (HTML, CSS, JS). What the user sees and interacts with.
- **Back-End Web Development:** The development of server-side code that handles things like routing, data handling and databases (Ruby, Python, Java, Clojure, and now JavaScript). The stuff "behind the scenes" that makes web applications work.
- With back end programming you have a choice of languages. With front end programming you can only use JavaScript

A Little History

- Written in 10 days
- Almost nothing to do with Java
- Was part of the Netscape/Microsoft throwdown, fought over competing versions
- Features and bugs locked in by existing user base
- Standardized by the European Computer Manufacturer's Association, so now its official name is ECMAScript, hence ES3, ES5, ES6 version names

A Little More History

- **First 10 years:** it got no respect, because of
    - inconsistent competing versions
    - the browser DOM (and its own inconsistent competing versions)
    - bugs and design flaws in JS itself
- **Next 10 years:** it took off.
    - jQuery was written to smooth over differences,
    - JS was improved, browsers became more uniform
    - amazing apps came out like Google Maps

What can JS do for you today?

- Historically, JavaScript has been used mostly to add interactive elements--like animations, buttons and user input forms--on the front-end.

- JavaScript is very easy to implement. All you need to do is put your code in the HTML document and tell the browser that it is JavaScript.

- JavaScript works on web users' computers — even when they are offline!

- JavaScript allows you to create highly responsive interfaces that improve the user experience and provide dynamic functionality, without having to wait for the server to react and show another page.

- JavaScript can load content into the document if and when the user needs it, without reloading the entire page — this is commonly referred to as Ajax (stands for "Asynchronous JavaScript and XML").

- JavaScript can test for what is possible in your browser and react accordingly — this is sometimes called defensive scripting and it supports something called progressive enhancement, or graceful degradation.

- Last but not least, JavaScript, through its Node.js interpreter (or runtime environment), can now run servers, talk to databases, and do other back end tasks.

Downsides of JavaScript

- **Environment**. In the browser, you have no control over the environment your code will be running in.
    - You don't know what computer is on the receiving end of your web page and you don't know how busy the person's browser and their computer is with other things.
    - It might not even run at all because the person turned off JS for some reason (security concerns, annoyed by popups)
- **Bugs and design flaws**. They remain from the initial days of the language, that can't be fixed because that would break backwards compatibility.
    - Some aspects of JS that seem difficult to grasp at first are really elegant concepts that are important to learn for programming in general
    - Some are just weird features of JS whose confusingness is not fundamentally interesting
    - Some are straight-up bugs. You just have to learn them
    - We'll try to point out all these things in this class

Node.js

- Allows JS to be used on the back end

- JS was already popular and necessary for browser work — Node allows people to use one language for their code up and down the “stack”

- This is very helpful for engineers because it results in less context switching. Switching between JavaScript, HTML, CSS and a back-end language (Ruby, Python, PHP, Java, etc) is cognitively draining. By using a single language, we can avoid some of that back and forth.

Asynchronicity in Node.js

- **Asynchronous**. On top of that, one of the big differences is that Node.js is fundamentally designed to be event-driven and asynchronous. The default mode of many platforms and languages is to do one thing at a time — doing otherwise is these languages is convoluted if it's even possible.
    - Imagine someone doing a paper delivery riding on their bike delivering papers every morning. Imagine they stop at each house, throw the paper on your doorstep, and wait to make sure you come out & pick it up before moving on to the next house. That would be what we'd call blocking – each line of code finishes before moving on to the next line of code.
    - Now imagine they throw the newspaper on your porch but never stop pedalling. They just keep throwing papers on porches, so that by the time you pick it up they'll be 3 or 4 houses down. That would be non-blocking, or asynchronous. And that's what makes Node so powerful.

What we'll be using Node for

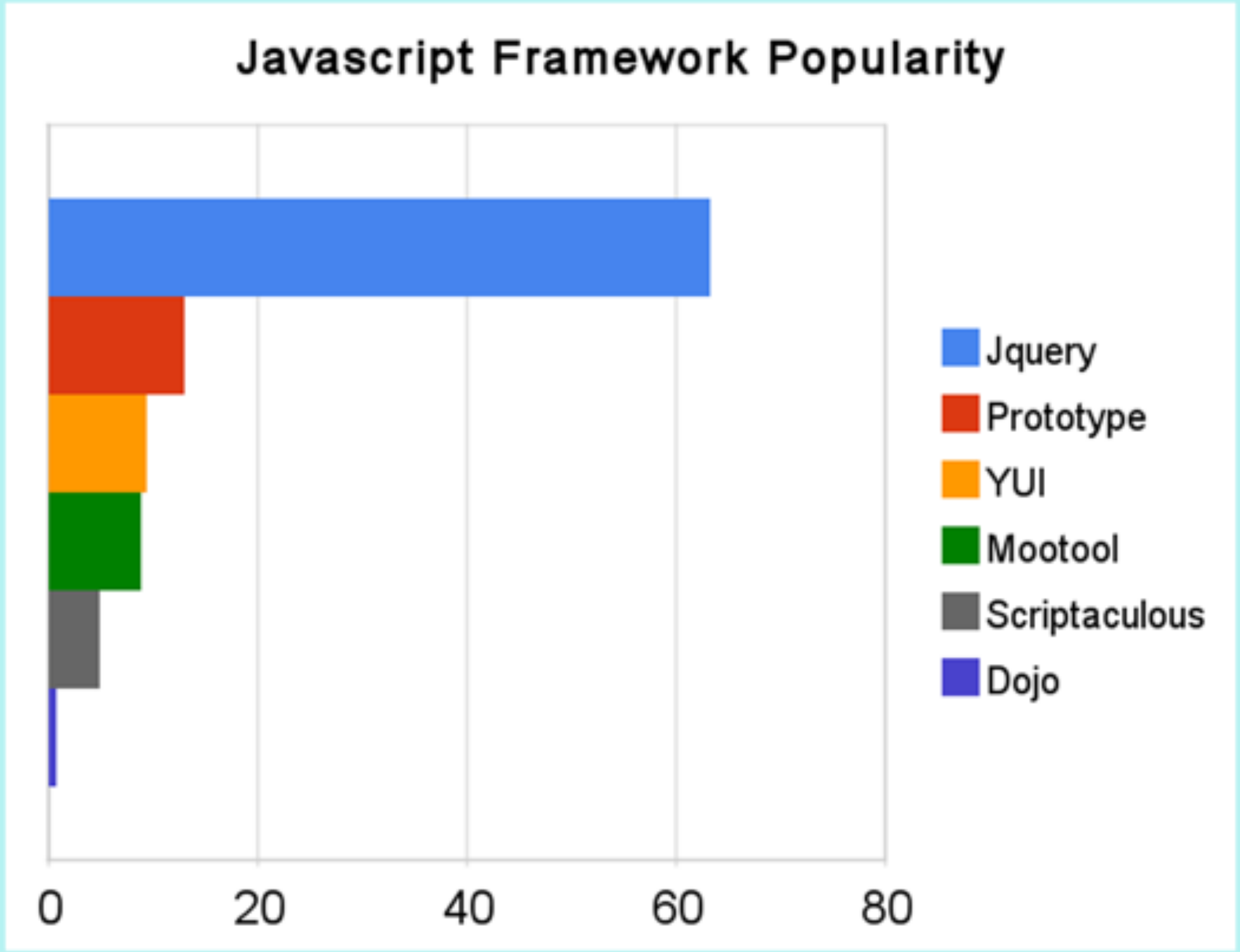
- We won't be delving into using Node to run back end servers in this class
- Very useful for running basic programs on your computer without having to worry about browsers and html pages

Frameworks

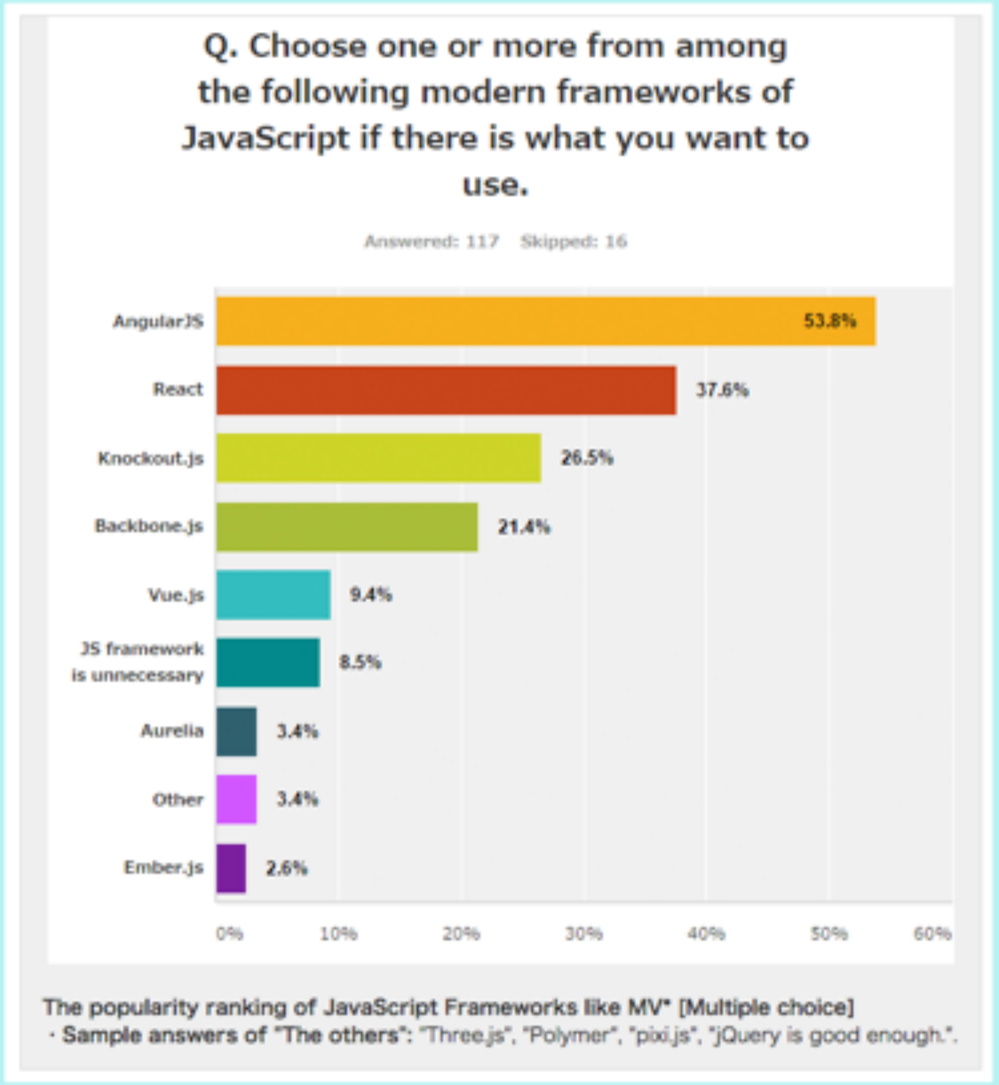
- Difference between frameworks and libraries
- Libraries are collections of code for you to use how you see fit, like a swiss army knife
  - Examples of JS libraries we'll be using are jQuery and Handlebars, another very good one is Underscore.
- The word "Framework" is ill-defined.
  - One definition: a complex library with tools for managing everything you need to run a web application, that relies on you following certain conventions in organizing all your code to best take advantage of the power of the framework.
  - Often comes with a lot of its own concepts you have to learn, many based on the MVC (Model-View-Controller) pattern, or at least MV*
  - You need a solid grasp on Vanilla JS first.
  - Popular front-end frameworks include Ember (MVC), Angular (MVVM). Backbone (MV?) is kind of a framework, React is pretty much just a view renderer, so (V).

Where are they now?

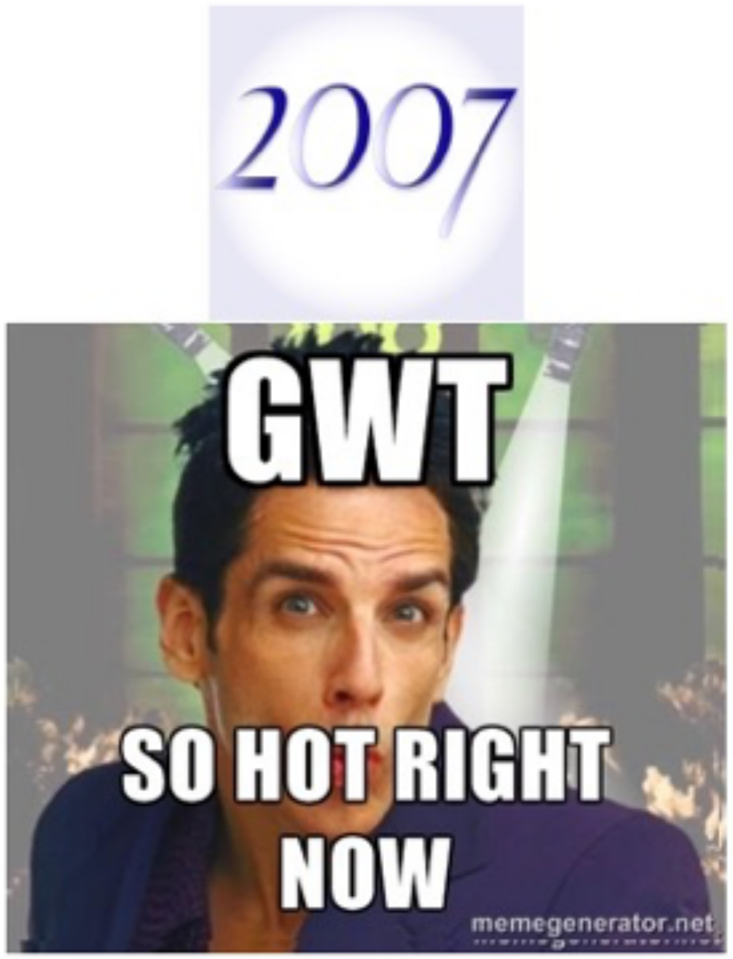
2010

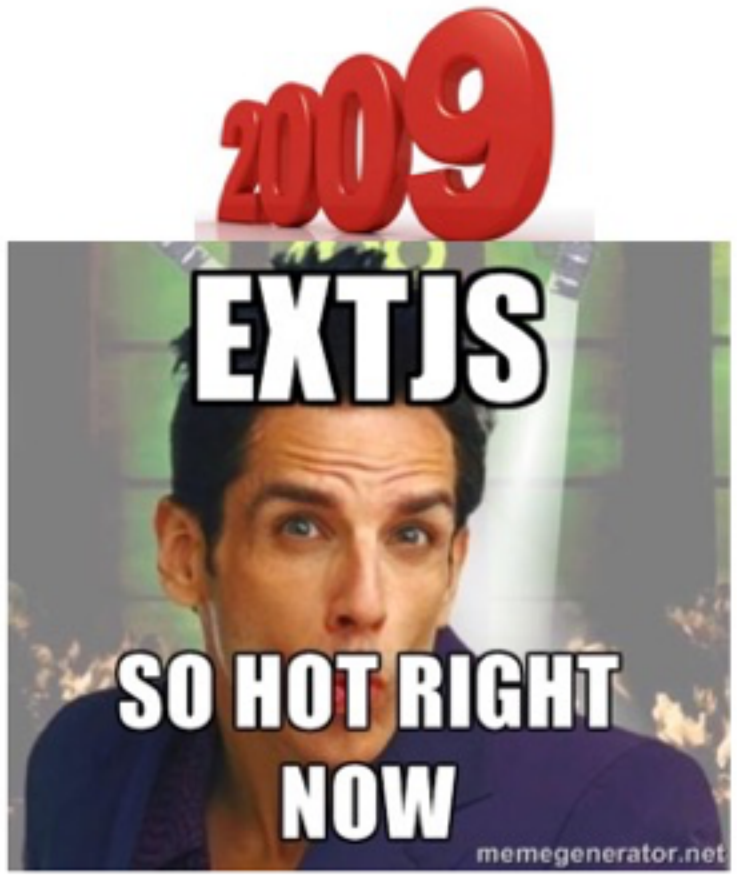


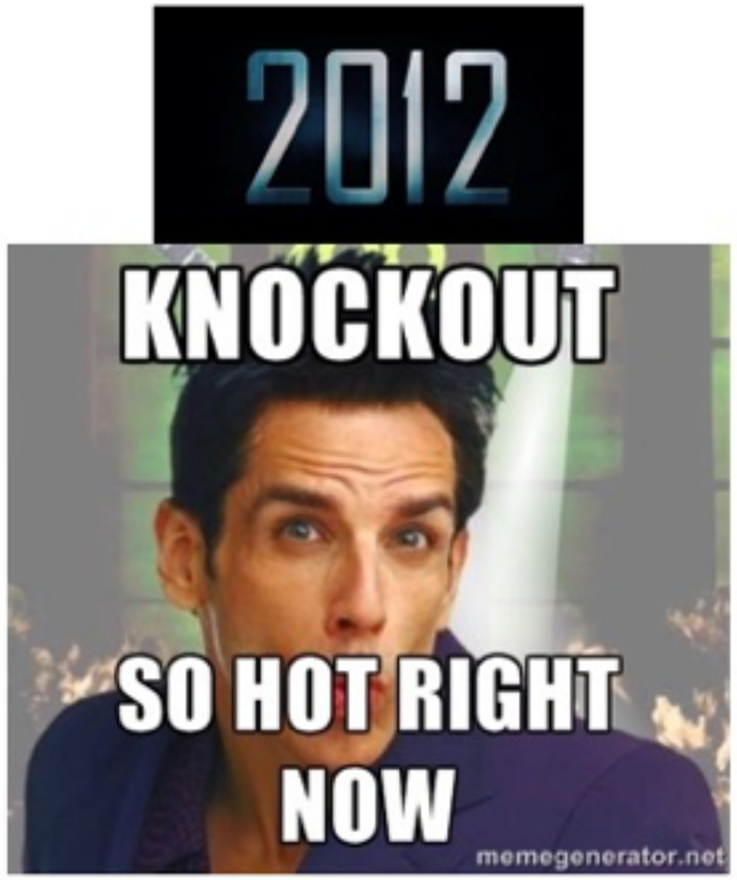
2015

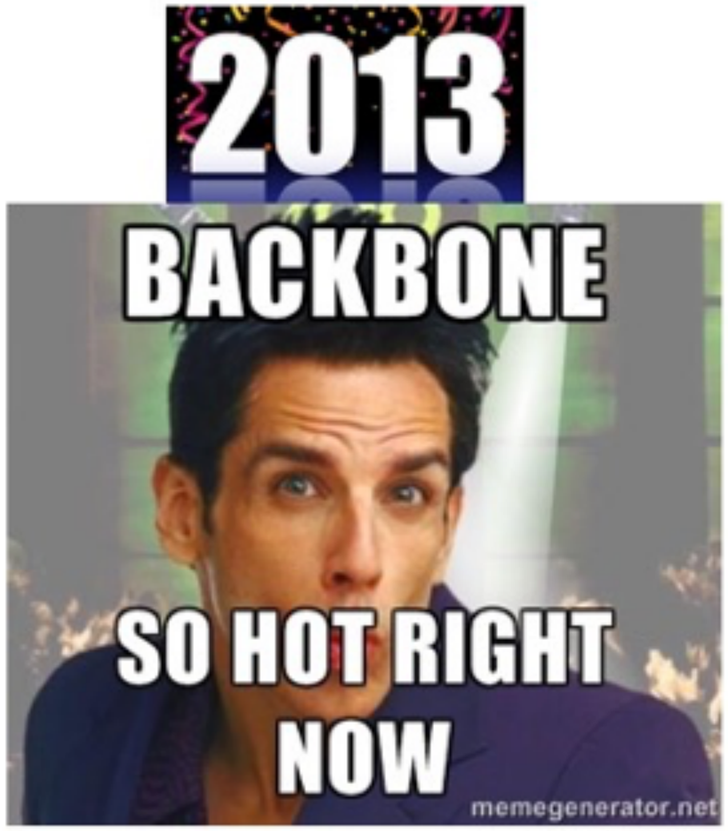


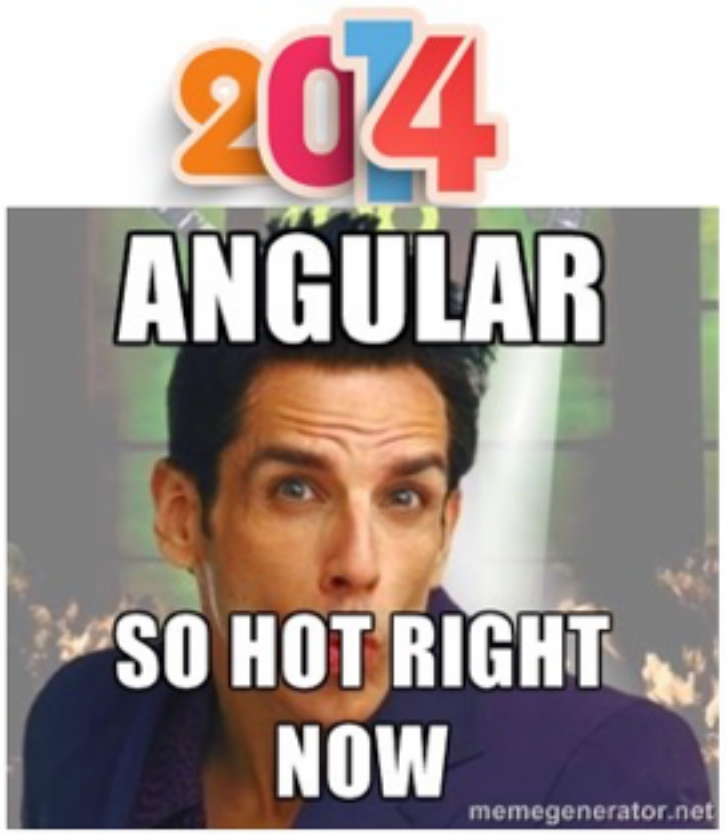
The popularity ranking of JavaScript Frameworks like MV* [Multiple choice]
· Sample answers of "The others": "Three.js", "Polymer", "pixi.js", "jQuery is good enough.".

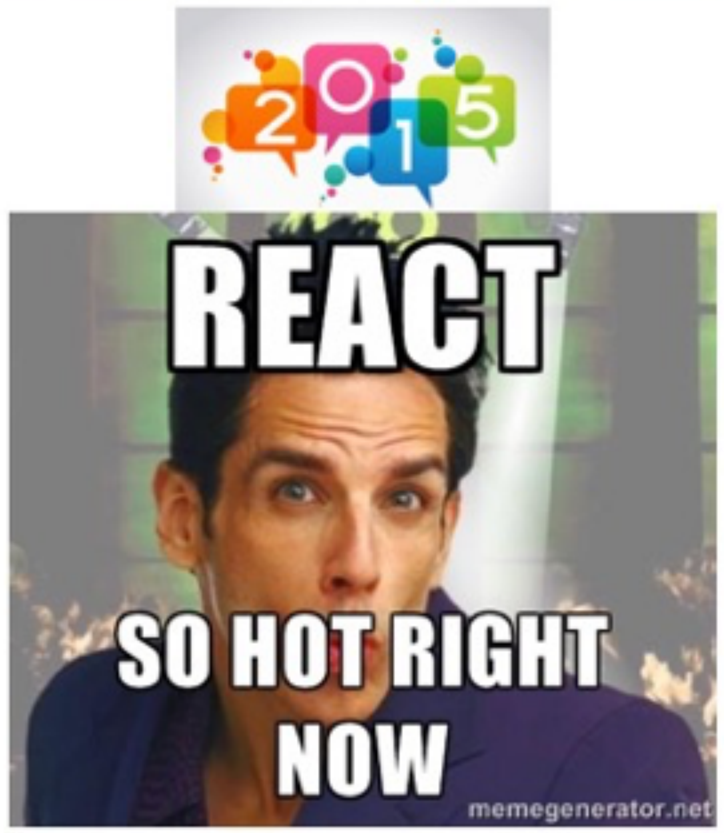

2007
GWT
SO HOT RIGHT NOW
memegenerator.net


2009
EXTJS
SO HOT RIGHT NOW
memegenerator.net


2012
KNOCKOUT
SO HOT RIGHT NOW
memegenerator.net


2013
BACKBONE
SO HOT RIGHT NOW
memegenerator.net


2014
ANGULAR
SO HOT RIGHT NOW
memegenerator.net


2015
REACT
SO HOT RIGHT NOW
memegenerator.net

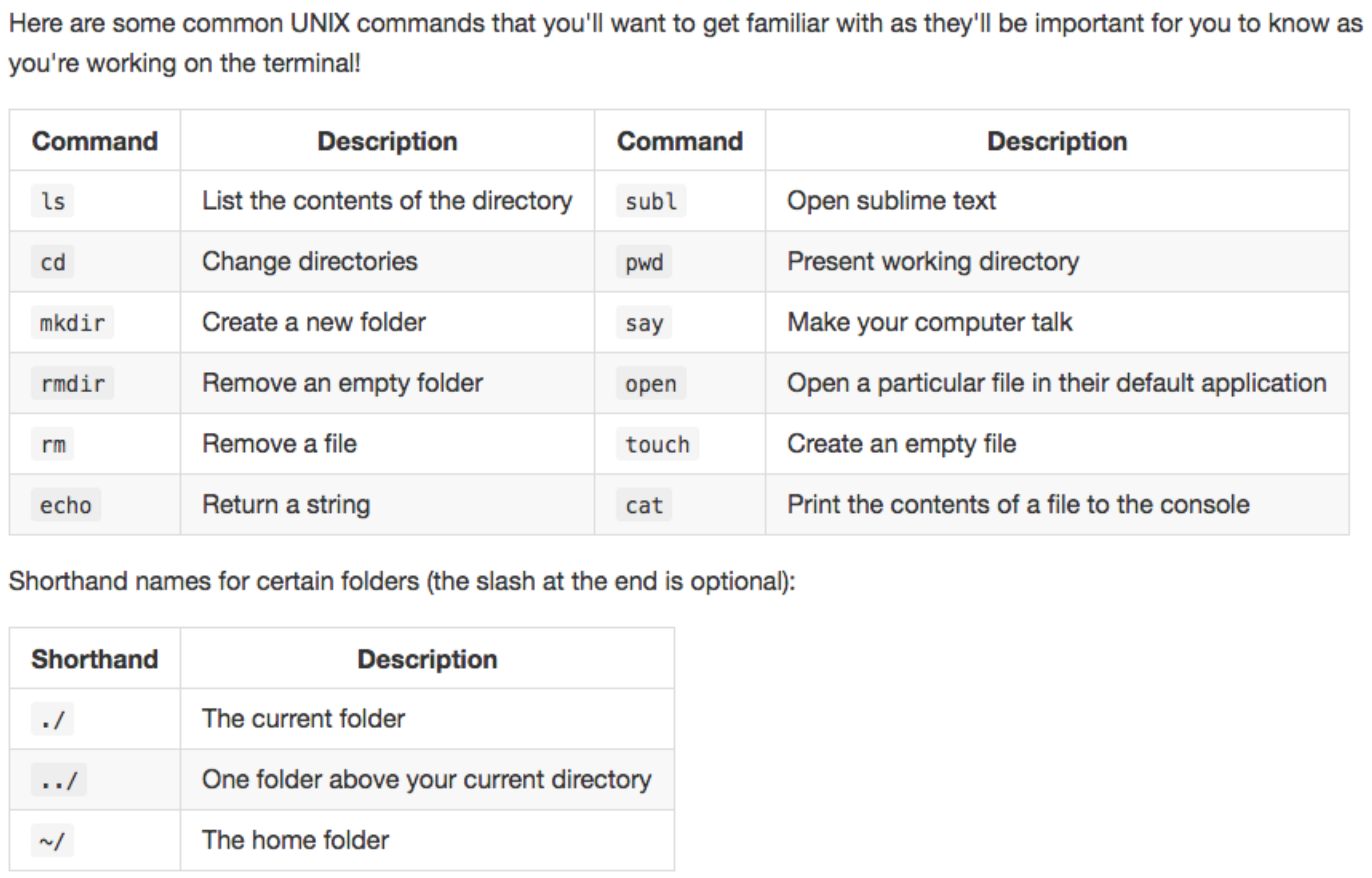
Intro to the command line

- A control panel for the Unix operating system
  - released in 1970
  - open-source version (GNU/Linux) released in 1991
  - incorporated into Mac with OS X in 1999
  - So ubiquitous now you'll never hear people say the word "Unix"
- Command line accessed from a terminal program
- Everything you can do with the UI you can do from the command line
- We'll need it to push projects to Github

Which terminal program to use?

- Mac: Terminal
  - Better alternative: iTerm (www.iterm2.com)
- Windows: “Command Prompt”
  - Alternatives: The “Git Bash” program which came with Git for Windows is supposed to be pretty good, but if you want you could also check out: Console (http://sourceforge.net/projects/console/)

Basics



Here are some common UNIX commands that you'll want to get familiar with as they'll be important for you to know as you're working on the terminal!

Command	Description	Command	Description
`ls`	List the contents of the directory	`subl`	Open sublime text
`cd`	Change directories	`pwd`	Present working directory
`mkdir`	Create a new folder	`say`	Make your computer talk
`rmdir`	Remove an empty folder	`open`	Open a particular file in their default application
`rm`	Remove a file	`touch`	Create an empty file
`echo`	Return a string	`cat`	Print the contents of a file to the console

Shorthand names for certain folders (the slash at the end is optional):

Shorthand	Description
`./`	The current folder
`../`	One folder above your current directory
`~/`	The home folder

Codealong

Make a page

- Complete the following instructions below in the Terminal application.
  - Create a folder in your ~/GA-JS/ folder titled goals.
  - Once inside that folder, create three empty files:

    goals.html
    worries.html
    index.html

  - Open Sublime Text, click File > Open and navigate to your newly created ~/Sites/goals folder. In the respective files (goals.html and worries.html), write your top 3 goals and top 3 worries for this course.
  - Paste the following contents into index.html:

```
<html>
   <head></head>
   <body>
      <a href="goals.html"><img src="http://i.imgur.com/dosK05U.gif" /></a>
      <br>
      <a href="worries.html"><img src="http://i.imgur.com/2s0HwpM.gif" /></a>
   </body>
</html>
```

  - Open index.html with your browser and make sure you see your goals and worries.

Internet vs. World Wide Web

- What do you think the World Wide Web is?
- How is it different from the Internet?
- Can you think of an analogy to describe the difference?
- Who controls the Internet?

The Internet

- Stands for “interconnection of computer networks”
- Combination of:
  - **The physical infrastructure.** Big collection of computers and cables — millions of personal, business, and governmental computers, all connected like roads and highways.
  - the agreed-upon protocols (rules) used to transfer the information
- Modern form of the Internet coalesced into existence in the 1960s and 1970s
- The Internet in action boils down to a series of requests and responses — you send information out to someone else on the Internet, and based on the info you send, you get information back.
  - Any device you have that makes a request or is capable of receiving requests, is physically part of the Internet.

Who owns the Internet?

- No single person owns the Internet. No single government has authority over its operations. Some technical rules and hardware/software standards enforce how people plug into the Internet, but for the most part, the Internet is a free and open broadcast medium.

The World Wide Web

- Invented in 1989 at CERN by Tim Berners-Lee, who made 2 things:
    - the language for describing web pages — HTML (hypertext markup language) and
    - the protocol for transferring HTML pages across the Internet — HTTP (hypertext transfer protocol)
- The “Web” consists of all the HTML documents sitting on servers around the world, ready to be requested and then displayed as web pages in people’s browsers.
- Clicking a link in each page make a request for a new page — this was what was meant by “hypertext”
- The Web is the way most of us access information over the Internet — over 65 billion web pages today.

Client-Server model

- Create diagram of the client-server model on the whiteboard
- Vocab: server, client, HTTP, request, response

Client Server model in-depth

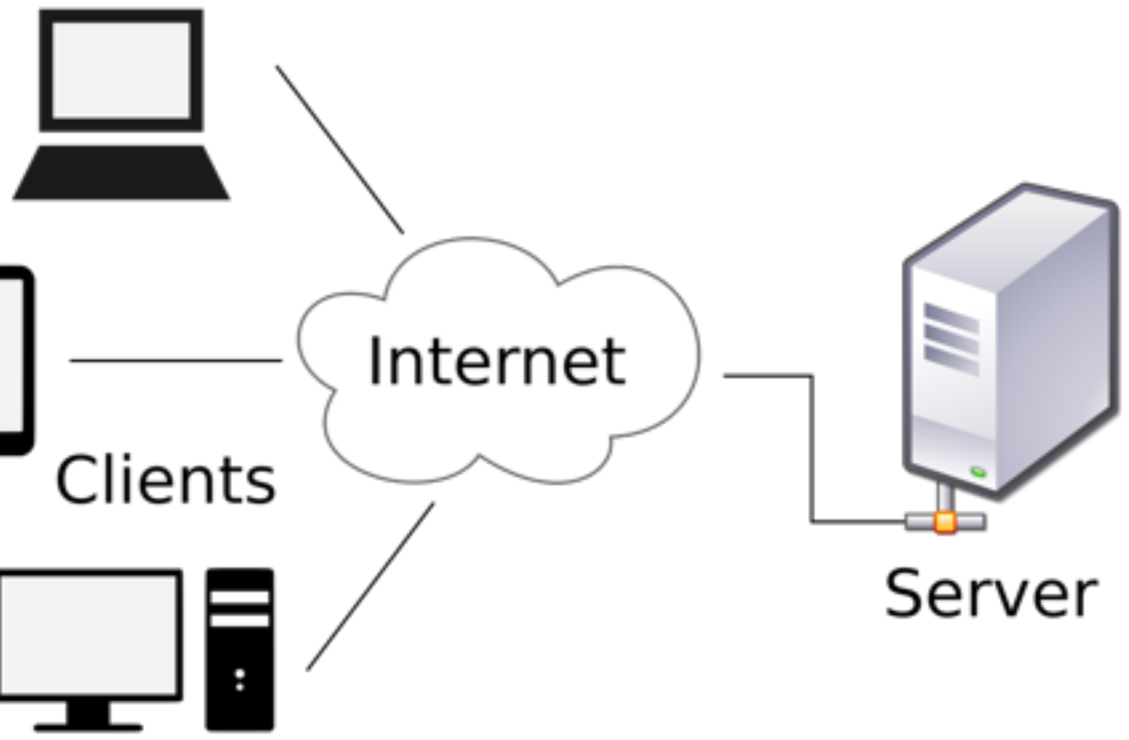
- Communication between a server and a client occurs via a request/response pair. The client initiates an HTTP (Hyper Text Transfer Protocol) request message, and the server responds with an HTTP response containing HTML for a web page. In short, server receives requests from clients and makes responses to them, over and over and over again.

- The terms “server” and “client” can refer both to the devices themselves, and to the applications running on them (examples include a web server app on the server device, and a web browser or a terminal program on the client device)

- A server may receive requests from many different clients in a very short period of time. Because the computer can perform a limited number of tasks at any moment, it relies on a scheduling system to prioritize incoming requests from clients in order to accommodate them all in turn. To prevent abuse and maximize uptime, the server's software limits how a client can use the server's resources. Even so, a server is not immune from abuse. A denial of service attack exploits a server's obligation to process requests by bombarding it with requests incessantly. This inhibits the server's ability to respond to legitimate requests.

- Of course, this is a somewhat abstract explanation of how this all works — there are a ton of intricacies that go into getting your request message to the right place and delivering the information you requested.

Client Server recap



- **Server** - A computer or other device connected to a computer network; the server may offer information resources, services, and applications (via computer code!) to users or other computers on the network. **Ex:** servers and web services
- **Client** - The requesting program in the client/server relationship; the client initiates an HTTP request message, which is serviced through a HTTP response message in return. **Ex:** browsers, terminals, mobile devices
- **HTTP** - Stands for "Hyper Text Transfer Protocol" - because it's a protocol, it allows for communication between a variety of different computers and supports a ton of different network configurations. To make this possible, it assumes very little about a particular system.

Domain Names

- All computers on the Internet have numeric addresses called IP (“Internet Protocol”) addresses, like 123.123.123.123
- Obviously we don’t want to use these:
    - “For more information about our amazing dog-washing products, find us on the World Wide Web at 124.45.32.122.”
- So we use names instead, like “www.dogwash.com”

DNS (Domain Name System)

- When you type a website domain into your web browser, what happens is a "DNS Lookup" of that particular domain's IP address, so your computer can actually connect to it.

- When you type in “www.dogwash.com”, a request (the DNS Lookup) is first made to a DNS server, which tells you the numeric IP address.

- In real world terms, it's like how we use people’s names these days to call them on the phone, rather than having to know their phone number. The contacts software in your phone (or in the old days, the phone book) is like the DNS server.

- apple.com -> 17.172.224.47
  facebook.com -> 31.13.65.1
  google.com -> 216.58.192.46

JS in the terminal

- Mostly we’ll be running JS in the browser, but in order to get comfortable with using the terminal and working like programmer, we'll be focusing on using JS on the command line during the first unit of this class.

- We’ll be using Node to make scripts run in the terminal. Node.js can do a lot of things and is often used to run servers, but for the purposes of this class, we'll only be using Node as a command line JS interpreter.

- Starting next class, we'll be creating our own JavaScript files and building things out. But before we dive in, we want to explore the power of the terminal and run JavaScript inside of it.

- The Terminal is like your universal Swiss Army knife. It will always be with us moving forward and for now it helps us focus our learning. Computers have always had a text-only interface. Some of you may remember DOS or the early text only games before complex graphical interfaces. The terminal is a tool from that era that professional developers still use every day.

Conclusion

- Review
    - The client-server model.
    - Explain the difference between the Internet and the world wide web.
    - Understand how we will be using Node in the class (as a JS interpreter, not necessarily a framework).
- **The Developer Mentality.** Here are some tips that you'll want to keep in mind as your continue coding!
    - Choose the right OS, editors and tools for your projects. Remember to do your research.
    - Leverage the online community's vast libraries and documentation.
    - Be efficient: Use the keyboard as much as possible instead of the mouse.
    - If you find yourself doing the same thing repeatedly, automate it.