Computer Science 50
Fall 2010
Scribe Notes

Week 8 Monday: October 25, 2010
Andrew Sellergren

Contents

Computer Science 50
Fall 2010
Scribe Notes

Week 8 Monday: October 25, 2010
Andrew Sellergren

1 Announcements and Demos (0:00–6:00)

- This is CS50.
- 2 new handouts.
- Soon, thanks to our instruction in web programming, you'll be empowered to make horrendous (read: awesome) websites like Hamster Dance!
- If you're interested in joining the Harvard Digital Media Group, sign up here or e-mail them.
- If you're interested in taking your final project to the next level (once you're done with it of course), HackHarvard is available to you as a way of possibly recruiting funding, programmers, campus space, etc. to build out your project. Not to be confused with the Hackathon!
- The Final Project specification is now available so that you can begin thinking about it.
- Problem Set 6 endeavors to give you some exposure to text editors other than Nano and VIm. One handy feature of TextWranger for the Mac, which we'll evangelize in addition to Notepad++ for the PC, is the ability to connect to an SFTP server within it so that you actually sync your work to the cloud.
- We apologize if you've found long lines at office hours. One way to mitigate this is to come earlier in the week rather than Wednesday and Thursday when many students will be scrambling to meet the problem set deadline. We do try to have more office hours concentrated at the end of the week, but we too are human and we have our own scheduling constraints to respect.

2 The Internet (6:00–30:00)

2.1 The Ethernet and TCP/IP Layers

- In the coming weeks, we'll be exposing you to web development which, we dare say, might be more appealing to you than command-line programs. This is not to say, however, that learning C was for nought, since it undoubtedly introduced you to the basics of programming and also enforced a higher standard of elegance in your code than many modern programming languages will. Frankly, even the convenience alone of languages like PHP makes them compelling. Whereas we had to implement our own hash table in C, we'll soon see that hash tables are built in to PHP.
- So far, we've been connecting to the CS50 Cloud via SSH and SFTP. As we begin our foray into web development, we'll be connecting to it via HTTP, a protocol that defines the conventions which web servers and clients must adhere to in order to communicate with one another.

- Check out this trailer for Warriors of the Net, which is a half-fun, half-serious look at how the internet really works.
- In our high-level look at the internet, we'll take for granted that data can be transmitted over ethernet cables. Suffice it to say that zeroes and ones can be encoded using electricity, as we saw in the case of hard drives.
- On top of the ethernet layer of the internet is the IP layer. This it the protocol which requires entities to have unique IP addresses, a number of the form w.x.y.z, where w, x, y, and z are numbers between 0 and 255. If you do the math, that means each of the four numbers can be represented by 1 byte, so the full IP address can be represented by 4 bytes. This implies that there are 4 billion possible IP addresses just as we can store numbers up to 4 billion using a 4-byte unsigned integer. As you can imagine, we actually are running out of IP addresses since there are only 4 billion possible, but thankfully IPv6, the next version of IP, will support many more possible addresses.
- TCP is a layer on top of IP which guarantees delivery of data. If data is lost during transfer, TCP notifies the sender so that it can resend it.
- Data on the internet is passed through routers, or gateways, whose sole purpose is to direct data from source IP address to destination IP address. Typically, data will pass through 30 or fewer stops along its way between point A and point B. We can see this in action using a command-line program called `traceroute` like so:

  ```
  traceroute www.stanford.edu
  ```

  This command will spit out a list of the domain names of the routers which our data passes through on its way to Stanford. Alongside each of these routers will be three independent measurements of the time it took our data to reach this router. The first few lines list routers that belong to Harvard with values on the order of 2-3 milliseconds next to them. Next, the domain names begin to contain the prefix `nox`, which stands for Northern Crossroads, a large datacenter in the Northeast. A few steps later we see the prefixes `kans`, `hous`, and `lax`, which we can guess stand for Kansas, Houston, and Los Angeles. You'll notice that the time measurements are somewhat proportional to the physical distances between these routers. For the rows that show `* * *`, the corresponding gateways didn't respond to us, possibly for privacy reasons.
- Similarly, we can run `traceroute` on `gmail.com` or `cnn.com` to find out their latency and the location of their nearest datacenter.[1] What's perhaps

[1] Alternatively, you can use `traceroute` to discover someone's IP address and connection speed!!!!1111 Okay, actually, please please know that this video is just plain wrong. I don't want to have to give points back to someone on a quiz.

more interesting is running `traceroute cnn.co.jp`, which will connect us to the Japanese version of CNN. When we do so, we'll see that the request takes much longer, on the order of hundreds of milliseconds, presumably because it actually has to traverse the ocean!

2.2 HTTP and The Client-Server Model

- When you go to a restaurant and order a meal, your food is brought to you by your server. In this scenario, you are the client. In the context of the internet, the browser that you open to surf the web is the client and the computer that answers your request with a web page is the server. The relationship is essentially the same, however.
- When you make a request for the homepage of CNN, the first step in the process is called a DNS lookup. DNS, or Domain Name System, is the protocol which maps a human readable URL like `cnn.com` to a machine-readable IP address of the form w.x.y.z.
- Once the DNS lookup is complete, your request is routed to the IP address of CNN's servers via the gateways we looked at using `traceroute`. The request looks something like the following:

  ```
  GET / HTTP/1.0
  ```

  The first `/` denotes that we want the root page of the server. `HTTP/1.0` indicates that we're using HTTP, or Hypertext Transfer Protocol, version 1.0.
- Embedded in the response given by the web server will be HTML, which indicates how the page will be displayed to us, including any graphics or special markup. To view the HTML that is returned by the web server, we can simply right click on the page in our browser window and choose View Source. If we do this on The Crimson's web site, we can see, among other things, lines that look like the following:

  ```
  <a class="small" href="/tag/Game%20Recaps/">Game Recaps</a>
  ```

  This is the syntax which gives us a hyperlink. The `href` attribute tells the web server where this page is located. Since in this case it's a page on the same web server, it is a relative URL. If it were an external page, it would begin with `http://`, just like any fully formed URL.
- This ability to view any website's source code will prove invaluable as you learn web programming. HTML is not generally considered intellectual property, so if you see something cool on a website, you can probably replicate it on your own by borrowing from their source code (possibly with attribution).

- HTML is an interpreted language. The response from the web server contains the plaintext source code. The browser is responsible for interpreting that source code and displaying it properly. This is also true of JavaScript source code. PHP is also an interpreted language, but it will be interpreted on the server before anything is returned to the client.

- As we begin our foray into web programming, we'll be relying on a multitude of development tools. One that we'll evangelize is Firebug, a free plugin for Firefox that installs a toolbar on the bottom of your browser window. When you click on the bug icon in the bottom right corner, this toolbar will pop up and enable you to examine the structure of the web page you're on in a cleaner, more structured way. You can click the Net tab, for example, and watch all of the HTTP requests that are made when you visit a page. On webhamster.com, for example, 6 requests are made, including 4 to grab the different GIFs of hamsters. When we click on the first of these 6 requests and click to show the Request Headers, we see that the first line indeed looks like what we mentioned earlier:

```
GET / HTTP/1.0
Host: webhamster.com
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9.1.10) Gecko/2
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
If-Modified-Since: Fri, 19 Dec 2008 00:52:02 GMT
If-None-Match: "160656-2526-45e5bb796f080"
Cache-Control: max-age=0
```

  Most of these so-called HTTP headers are uninteresting. However, we'll touch upon a few. The Host header specifies the full domain of the web page we're requesting. This is useful on shared servers which host more than one web site. User-Agent sends the version of your browser and operating system to the web server.

3 HTML (32:00–75:00)

- If we click on the Response tab in Firebug for this same request, we'll see the actual HTML source code of the web page that was returned by the server. HTML is very pedantic in that each element on your page begins with an opening tag, e.g. `<head>`, and ends with a corresponding closing tag, e.g. `</head>`.

- The HTML of a web page is divided into two main sections. The `head`

element contains the title of the page and (usually) any script declarations. The `body` elements contains the actual content of the page.

- HTML elements, somewhat like functions in C, can have parameters of *attributes* provided to them within the tag declaration. For example:

  ```
  <body bgcolor="#FFFFFF">
  ```

  `bgcolor` is the background color attribute and it is defined as either a common color (e.g. red) or a hexadecimal RGB triple (e.g. #FFFFFF). Incidentally, #FFFFFF specifies white.

- We won't focus too much on `webhamster.com` since it actually uses an older version of HTML. The newest version, which we'll be teaching you, is HTML 5. You may have heard that iPhones don't support Flash but they do support HTML 5. HTML 5 hasn't seen widespread adoption yet, but we feel it's worth teaching because it's the latest and greatest and within a few years, it will be the industry standard. HTML 5 is also extremely powerful. Using it, we were able to develop a version of the CS50 video interface that can be used on browsers that don't have Flash installed.

- We've configured your CS50 Cloud accounts so that you'll be able to use them for web development and hosting. If you so choose, you can pay a nominal fee to a domain registrar like GoDaddy and register your very own domain such as `isawyouharvard.com`. If you do so, you can simply point that domain to the CS50 Cloud and thus host all your files on our servers so that you don't have to pay someone else to.

- On Linux systems, web hosting is as simple as creating a directory named `public_html` which is accessible for executing by anyone. Once we've done so, we'll open a file called `hello.html` and write the following:

  ```
  <!DOCTYPE html>
  <html>
      <head>
          <title>hello, world!</title>
      </head>
      <body>
          hello, world!  welcome to my homepage!
      </body>
  </html>
  ```

  This is the doctype declaration or DTD. Because we haven't specified a version of HTML, the browser is going to assume that we mean HTML 5. The `html` tags tell the browser that some HTML source code is on its way. Even though we have nothing between them, it's good practice to

write a closing tag whenever you write an open tag (just as you would write a closing curly brace whenever you write an open curly brace for a function) so that you don't forget.

- Interestingly, this is all that's required to make a web page. Unfortunately, when we try to visit our newly created page on the web by navigating to `http://cloud.cs50.net/~cs50/index.html`, we initially get an error telling us it is forbidden. The forbidden error gives us one interesting piece of information, namely that we tried to access this page on port 80. Just as SSH and SFTP use port 22 by convention, HTTP uses port 80 by convention. Port numbers allow servers to handle multiple different types of requests on the same IP address, as they can funnel the SSH traffic to one port and the HTTP traffic to another, for example.

- Port number 80 is implied within URLs, but we can actually run web services on different ports and connect to them like so:

  ```
  http://cloud.cs50.net:8000/~cs50/hello.html
  ```

  Here, we're using port 8000 instead of the standard port 80.

- The forbidden error occurred because we didn't execute the `chmod` command on our file to make it readable by anyone. If we run `ls -l hello.html`, we can see that it's readable and writable only by the owner. We fix this by running the command `chmod a+r hello.html`, which makes the file readable by all. Once we do this, we can actually access the web page and we see that the title "hello, world!" is displayed above the address bar in the browser.

- Let's change the `body` tag to look like the following:

  ```
  <body bgcolor="#ff0000">
  ```

  This makes the background of the entire webpage red. Now let's make use of the header tags, which range in size from the largest `h1` to the smallest `h6`:

  ```
  <h1>hello, world! welcome to my homepage!</h1>
  <h5>goodbye, world!</h5>
  ```

- Apropos to Problem Set 5, check out this video that introduces content-aware resizing of images.

- More in line with what we're doing today are animated GIFs, such as those of hamsters. These are essentially small, simple movies. To add an image of a hamster, animated GIF or otherwise, we'll use the `img` tag and specify the `src` attribute.

- Now that we've added more content, we'll want to center some or all of it, which we can do by placing it within a `div` tag. By placing the content within a separate tag, we can add different stylings to it. For example, we might specify the `style` attribute like so:

  ```
  <div style="background-color: yellow;">
  ```

  `bgcolor` is actually a deprecated way of specifying colors. The preferred method is via the `style` attribute. What we've specified here is a CSS (cascading stylesheet) property. We can actually specify multiple within the style attribute like so:

  ```
  <div style="background-color: yellow; text-align: center">
  ```

- One annoying consideration of web development is standardizing the appearance of web pages across multiple browsers. It turns out that different browsers interpret HTML in different ways, so we have to alter our source code to handle each of their idiosyncrasies if we're hoping for a uniform look and feel.

- Frankly, it would be tedious to cover all the different syntax that HTML uses, so we'll point you to the resources page which has a full reference.

- Let's turn the "goodbye, world!" message into a hyperlink:

  ```
  <a href="http://www.disney.com/">goodbye, world!</a>
  <br>
  ```

  The value of the `href` attribute specifies where we want the link to lead, in this case, Disney's website. After the anchor tag, we write `<br />` to tell the browser to insert a line break so that our hamster picture and hyperlink won't be on the same line.

- If we want to change the font that is used for text on a web page, we specify the `font-family` CSS property under the `style` attribute. Because not all browsers recognize all fonts, we can provide a list of fonts that the browser will attempt to render in order, stopping when it finds one it recognizes.

- One of the most useful aspects of websites is HTML forms, which allow you to gather input from the user. Using forms, we can re-implement Google like so:

  ```
  <!DOCTYPE html>

  <html>
    <head>
  ```

```
    <title>Fake Google</title>
  </head>
  <body>
    <div style="text-align: center">
      <h1>Fake Google</h1>
      <form action="http://www.google.com/search">
        <input name="q" type="text">
        <br>
        <input type="submit" value="Fake Google Search">
      </form>
    </div>
  </body>
</html>
```

Notice how simple it is to implement Google! The `form` tag takes as the `action` attribute the URL of a page that will handle the user's input. In this case, we're passing to Google's search page. The `input` tag specifies some information that should be gathered from the user. In this case, the name of the parameter is `q`, for query. A special `input` tag with the `type` of "submit" creates the familiar button that sends the form data to the back end.

- When we actually type in a query like "hamster," and click the submit button, we get whisked away to the following URL:

  `http://www.google.com/search?q=hamster`

  This is the value of the form's `action` attribute with our query appended to the end.

- If you examine the source code of Google's homepage, you'll notice that it's packed together very tightly with almost no whitespace. Consider that every space sent between server and client is another byte that Google has to pay for. Multiply that one byte by 1 billion requests per day and that's an additional 1 billion bytes of data transfer that Google incurs. Still, you can examine the source code of sites like Google using Firebug, which we'll organize the HTML cleanly even if there is no whitespace in the original.

- As a teaser for next week, check out David's first website, an effort to digitize the process of registering for freshman IMs.