Computer Science 75 Lecture 1: September 14, 2009
Fall 2009 Andrew Sellergren
Scribe Notes

Contents

Computer Science 75
Fall 2009
Scribe Notes

Lecture 1: September 14, 2009
Andrew Sellergren

1 HTTP (0:00–30:00)

- What is HTTP, exactly? Not to be confused with `httpd`, the HTTP daemon, the Linux software which implements it, HTTP is the actual protocol or set of standards which dictates how web browsers communicate with web servers and vice versa. You can think it like a conversation between two people which begins with a handshake, just as, in fact, a HTTP conversation does.
- Recall from last time our discussion of the virtual envelope which is sent from browser to server. Inside the envelope is the actual HTTP request, which might look like the following:

  ```
  GET /index.html HTTP/1.1
  ```

  Additionally, we can send parameters using this `GET` syntax if the data is short and not sensitive. If we want to send longer or more private data, we use `POST`. For example, if we want to return a dynamic page, we might structure our request like so:

  ```
  GET /foo.php?x=y&z=w HTTP/1.1
  ```

  In this way, we can influence the behavior of `foo.php` based on those variables which we passed.
- Note that you can pass an array via `GET` using the following syntax:

  ```
  GET /foo.php?foo[]=x&foo[]=y HTTP/1.1
  ```

  It's ugly, but it works. For the purposes of this class, we'll be mostly working with simple `GET` strings.
- The speed of the transaction is influenced by the number of parameters which are passed, but only minimally given how much other data is passed via the headers.
- There's no standard maximum length for URLs, but 1024 characters is a good rule of thumb.
- To view what's actually being passed across the wire, we can use Live HTTP Headers, a plugin for Firefox. Check out our other recommendations on the Software page.
- As Live HTTP Headers will show, even a visit to the course website spawns dozens of requests. This large number of requests, however, can be considered conservative compared to sites like CNN. The increasing desire for dynamic websites can prove costly when a large portion of users are now browsing the web on mobile devices that tend to be much slower in serving up these requests.

- Accessing `google.com` while Live HTTP Headers is open will capture the following:

  ```
  GET / HTTP/1.1
  Host: google.com
  User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9.1.3)...
  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
  Cookie: PREF=ID=44056...
  ```

  The first `GET` string is, as we mentioned previously, the core of the request. We'll come back to the Host line. The User-Agent string can be useful, but shouldn't be relied upon. It can be forged or scrubbed, so it's not a good idea to have functionality which depends upon it in order to execute. There's a feature of Safari, for example, which allows one to specify which browser will be used to fill in this header field.

- The Accept field is a comma-separated list of file types that the browser is willing to receive in response to the request. The other Accept-Language field is pretty self-explanatory. Accept-Encoding here specifies that the browser is willing to use compression when talking to the webserver. This is a great way to save bytes! The rest of the fields we'll gloss over, with the exception of the Cookie field, which we'll come back to.

- So what was the server's response? Google spit out a HTTP 301 message, meaning "Moved Permanently":

  ```
  HTTP/1.x 301 Moved Permanently
  Location: http://www.google.com/
  Content-Type: text/html; charset=UTF-8
  Date: Wed, 14 Oct 2009 04:41:16 GMT
  Expires: Fri, 13 Nov 2009 04:41:16 GMT
  Cache-Control: public, max-age=2592000
  Server: gws
  Content-Length: 219
  X-XSS-Protection: 0
  ```

  We can see that via the Location field, Google has actually caused the browser to redirect to a different URL, one that has been prepended with `www` and appended with a trailing slash.

- Now that we're talking about dynamic websites, it will be important to pay attention to the Cache-Control field, which specifies how long a page can be cached. After all, we want to make sure that the user has the most recent version of our site. Unfortunately, caching is one of the first issues which causes cross-browser difficulties. We'll have to turn to copying and pasting a few lines of code which control caching on multiple browsers since not all of them agree on the syntax for doing so.

- If we scroll down, we can see that the second request was met with this response:

  ```
  HTTP/1.x 200 OK
  Date: Wed, 14 Oct 2009 04:41:16 GMT
  Expires: -1
  Cache-Control: private, max-age=0
  Content-Type: text/html; charset=UTF-8
  Content-Encoding: gzip
  Server: gws
  Content-Length: 3511
  X-XSS-Protection: 0
  ```

  HTTP 200 is an invisible error code of sorts. We've all seen HTTP 404, which means "File Not Found." HTTP 401 means "Unauthorized," and points to a probable file permissions problem. HTTP 500 is "Internal Server Error." You'll see this error most often with a misconfigured `.htaccess` file. HTTP 200, however, means "OK."

- What was actually returned by the server was the content—the default page for this directory. After that, several more requests were spawned to retrieve the Google logo, some JavaScript and CSS, etc. Google is actually pretty minimalistic in this regard—many other websites have hundreds of requests which cause mobile browsers, among others, to grind to a crawl.

- Question: with Live HTTP Headers, how can you differentiate between requests and responses? A thin dashed line separates each request-response pair.

- The Keep-Alive header field is an improvement which allows multiple requests to go over the same TCP/IP socket connection. A value of 300, for example, specifies that the socket should remain open for 300 seconds. Toggling configuration options like this can have huge performance gains, especially in the context of database communication, if you know what you're doing.

- Question: does sourcing images in CSS files save bytes? Not really. There are tricks whereby developers can aggregate all images into a single image file which can be parsed or selectively displayed using JavaScript and CSS. This allows all images to be downloaded in a single request, which should save download time.

- Question: if a single background image is being repeated over and over again, it will only be downloaded once.

2 Apache and Web Server Configuration (30:00–45:00)

- It's time we learned a little more about the brand of web server we'll be using most frequently in this course. The advantages of Apache begin with its being both free and open-source. While IIS (Internet Information Services), the Microsoft-brand web server, offers similar functionality, it simply ain't cheap. We're going to abstain from most of the religious debates surrounding Microsoft vs. non-Microsoft products, but suffice it to say that we're voting with our wallets at least in this case.

- For the most part, `cs75.net` comes pre-packaged for you, so you won't need to do much configuration. Occasionally, however, you'll find it useful to consult the Apache manual so that you can tweak certain configuration options (which we've enabled you to do) in `.htaccess` files. In a `.htaccess` file, for example, you can accomplish what Google did with its URL rewriting—namely redirecting `google.com` to `www.google.com`.

- If all of your websites are being hosted on our course server, then all requests will contain the same IP address. How will it know which website to spit out? Thanks to virtual hosting, a feature offered by Apache (and also IIS), the request will contain not only an IP address, but also a single line of text in the HTTP header that represents what the user actually typed into the address bar. It might say, for example, Host: foo.com. We saw this earlier while using Live HTTP Headers.

- The configuration file for Apache is `httpd.conf`. Its location on servers varies, but it is commonly found in the `/etc/` directory. Below you'll see an excerpt from the course server's configuration file which configures virtual hosting:

```
<VirtualHost 64.131.79.130:80>
     ServerName www.malanrouge.com
     ServerAlias www.malanrouge.com malanrouge.com
     ServerAdmin webmaster@malanrouge.com
     DocumentRoot /home/malan/domains/malanrouge.com/public_html

     SuexecUserGroup malan malan
     CustomLog /var/log/httpd/domains/malanrouge.com.bytes bytes
     CustomLog /var/log/httpd/domains/malanrouge.com.log combined
     ErrorLog /var/log/httpd/domains/malanrouge.com.error.log

     <Directory /home/malan/domains/malanrouge.com/public_html>
          Options All
          suPHP_Engine ON
          suPHP_UserGroup malan malan
     </Directory>

</VirtualHost>
```

This is an excerpt from a very large text file which is actually composed of multiple smaller files. Notice that it's somewhat similar to an XML file in that it has open and close tags, simply by convention. The IP address at the top is that of the course's server. What's the significance of the 80 after the IP address at the top? This is a port number which corresponds to web traffic. We're saying to the server: listen for web traffic at this IP address on this specific port number.

- Notice that next to `ServerAlias` we're specifying that the web server should respond the same to requests for `www.malanrouge.com` and `malanrouge.com`. The other piece of the puzzle is that both these hostnames need DNS entries that point them to the same IP address.
- `ServerAdmin` isn't so important except in the case that the webpage fails completely to load, in which case the webmaster's e-mail address will be displayed by default.
- The `DocumentRoot` line is important to dissect. This is where the actual mapping of a domain name to a file directory takes place. In this case, we're telling the web server that the files for this domain name are located in the `public_html` subdirectory of the `malanrouge.com` directory. In fact, this naming convention for the directories was actually generated automatically by DirectAdmin, which makes it easy to administer multiple domains from a single account.
- The lines beginning with `CustomLog` and `ErrorLog` are, as you might expect, setting various logging options but aren't too important to examine.
- Within the `Directory` tag, you can see that we've waved our hands at the options by simply turning them all on by default (some examples include following symbolic links or viewing the contents of a web directory). This is what allows you to override default options in your `.htaccess` files.
- Question: if you wanted to keep an extra directory in the URL and house all your website's content in that subdirectory, you need only have a simple `RewriteRule` in your `.htaccess` file to redirect all traffic from the root directory to that subdirectory. Via the `.htaccess` file, you can also hide file extensions in URLs so that if you want to change the language you implement your site in, you won't have to fuss over changing all the URLs. The Panel, in fact, uses a `RewriteRule` to redirect you to port 2222 so that you don't have to remember it.
- If you need to view the server logs, you can do so by logging into the Panel and clicking on Site Summary.

- Question: almost all web hosting companies utilize virtual hosting. If you happen to want multiple domain names for the same account, you can pay a little extra for it.

3 Administrative Details (45:00–50:00)

- Sid will be hosting the first live section after class in Room 103. We realize this makes for a long night, but at least it cuts down your commuting requirements. There will also be a weekly online section held using the Elluminate software.
- Know that there's a course bulletin board run using the Simple Machines Forum software, which happens to be written in PHP and MySQL. This allows us to anonymize your names so that you can post any question without fear of looking silly! Please reserve questions which require sharing a large chunk of code for the e-mail list, but feel free to post questions of a general nature. You can also subscribe to the bulletin board via RSS so that you don't have to constantly check for new posts.

4 More with Apache (50:00–70:00)

- Let's see what we can do via the `.htaccess` file. If we navigate to `malanrouge.com`, we can see that the URL gets rewritten to have the `www` prepended. We do this with the following lines:

  ```
  RewriteEngine On
  RewriteCond %{HTTP_HOST} !^www\.malanrouge\.com$ [NC]
  RewriteRule (.*) http://www.malanrouge.com/$1 [R=301,L]
  ```

  What do these lines of code actually do? The first line is going to turn on the rewrite engine, as you might have guessed. The second line is a condition for rewriting: "If the host is not equal to `www.malanrouge.com`..." The ! or bang means "not," the ^ means "starts with," and the $ means "ends with." The NC means "not case," or rather case-insensitive. Finally, the third line: "...put everything after `http://www.malanrouge.com`." Here, the (.*) stands for "everything after the first forward slash," which by convention is stored in a variable called `$1`. Then, it will tack this on the end of the *full* web address and spit out a 301 "Moved Permanently" header (hence the `R=301`). The `,L` means "this is the last rewrite rule."

- The `%{HTTP_HOST}` is Apache's syntax for variable names, a list of which is available in the Apache documentation.
- HTTP 301 is "Moved Permanently" while HTTP 302 is "Moved Temporarily"—the former will be remembered by the browser while the latter won't. This was a distinction which became important when we briefly housed the course website at a temporary subdomain and used HTTP 301 instead

of HTTP 302. Google happened to store this result and from then on we were forced to keep this temporary subdomain alive so that it could redirect to the real website.

- To go the other direction,—to remove the `www` from a URL—we could remove the ! from the `RewriteCond` and the `www` from the `RewriteRule`. Walk through the logic yourself to see how this would work.
- The backslashes in front of the dots are the escape characters, which, in the context of these regular expressions, signals the engine to treat the dots as literal dots. Otherwise, by convention of regular expression syntax, the dots are placeholders for *any* character.
- We can even redirect users to a website that's not on our domain, if we so desire. The Apache module which makes all this possible is `mod_rewrite`.
- On the course website, we redirect users to the SSL-protected areas of the website using the Apache variable `%{HTTPS}`.
- Multiple `RewriteCond` lines will be combined together with the `AND` operator so that they must all be met for a given `RewriteRule` to be executed.
- A lot of URL rewriting is done by the browser these days. The `http://` is added by most browsers for example. Even ISPs have a hand in it. They've gotten a lot of flak in the geek community for redirecting users that make typos in URLs to sites that have advertisements on them.
- Question: we'll come back to this, but know that SSL requires a unique IP address, so virtual hosting with SSL isn't a possibility.
- How can you run Apache on your own computer? As we mentioned last time, XAMPP is an extremely useful software bundle that will allow you to develop on your home machine, even without an internet connection. Check out the installation directions here which were created by Keito Uchiyama, a former teaching fellow. Later in the semester, we will also offer a virtual machine package configured like the CS 75 server.

5 PHP (70:00–104:00)

5.1 Forms

- Forms are what allow us to take user input—via `GET` or `POST` strings—and generate dynamic content—in fact, this is what all user-drive websites boil down to. Here are some of the types of forms we'll be implementing this semester:
  - Text Fields

    ```
    <input name="email" type="text" />
    ```

- Password Fields

  ```
  <input name="password" type="password" />
  ```

- Hidden Fields

  ```
  <input name="id" value="123" />
  ```

- Checkboxes

  ```
  <input checked="checked" name=remember" type="checkbox" />
  ```

- Radio Buttons

  ```
  <input name="gender" type="radio" value="F" />
  <input name="gender" type="radio" value="M" />
  ```

- Drop-Down Menus

  ```
  <select name=state">
    <option value=""></option>
    <option value=MA"></option>
    <option value=NY"></option>
  </select>
  ```

- Text Areas

  ```
  <textarea name=comments"></textarea>
  ```

Text fields are pretty self-explanatory. Password fields are the same except they show text as bullets instead of actual characters (but don't really provide much more security). Hidden fields allow data to be sent without being easily visible to or changeable by the user. Don't mistake this for a security measure, however, because anyone with a modicum of technical savvy will be able to manipulate the values of these fields. It wouldn't be a good idea to store the prices of items in hidden fields, for example.

- Checkboxes are fairly straightforward. Radio buttons are the same except mutually exclusive. Drop-down menus and text areas, we'll mention just briefly.

- Let's examine the source code for Google's home page. If we boil it down to just the form elements, we are left with the following:

```
<form action="/search" name=f>
<input name=hl type=hidden value=en>
<input autocomplete="off" maxlength=2048
       name=q size=55 title="Google Search" value="">
<br>
<input name=btnG type=submit value="Google Search">
<input name=btnI type=submit value="I'm Feeling Lucky">
</form>
```

For all practical purposes, this is all it takes to implement Google search on the front end! Notice that this isn't valid XHTML because they don't have quotation marks surrounding a lot of their attribute tags. However, consider that those extra few quotation marks would probably cost them billions of bytes per day.

- The only input that's really worth mentioning is the text field that contains the user's query. The name of this input field is simply `q`.
- So let's re-implement Google on our own site. All we need are the following lines:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Fake Google</title>
</head>
<body>
    <form action="/search" method="get">
          <input type="text" name="q" value="" />
          <br />
          <input type="submit" value="Fake Google Search" />
          <input type="submit" value="I'm Feeling Lucky" />
    </form>
</body>
</html>
```

  If we save this as `google.html` and then try to access it on our own server, we encounter a problem. We get an HTTP 403 error. That's because we didn't make our file world-readable. We need to execute the following command:

```
chmod 644 google.html
```

  If you use DirectAdmin to upload your files, you can set the permissions via their GUI as well.

- Now let's dress it up a bit:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
```

```
    <title>Fake Google</title>
</head>
<body>
    <div align="center">
    <h1>Fake Google</h1>
        <form action="/search" method="get">
              <input type="text" name="q" value="" />
              <br />
              <input type="submit" value="Fake Google Search" />
              <input type="submit" value="I'm Feeling Lucky" />
        </form>
    </div>
</body>
</html>
```

Unfortunately, it's still broken. When you click Submit, we get an HTTP 404 error. What we need to do is change the `action` attribute of our form to the following:

```
 <form action="http://www.google.com/search" method="get">
```

Google seems to accept requests that don't originate from their own domain. If we change the `method` attribute to `POST`, however, we get an error since Google has disallowed this, for whatever technical reason.

5.2 About PHP

- Unlike C or C++ or other compiled languages, PHP is an interpreted language. This means that the server will pass your PHP files to the PHP interpreter, which will execute your scripts and serve the output to the browser.

- Our study of PHP will focus not on the quirks of its syntax, but moreso on the conceptual underpinnings of the language itself. Know that the online documentation for PHP is your best friend! For functions, it provides not only information on return values and arguments, but also examples of their usage.

- On our course server, we have also installed suPHP for security reasons. suPHP stands for Superuser PHP, which is simply an allusion to the Linux notion of a user who has root permissions. What suPHP requires is that when a user executes code on the server, he does so as the *owner* of that file. The alternative is that every user executes code as the same user like `nobody` or `apache`. This alternative has two major problems associated with it: first, if the web server is shared, then anyone else who has access to it will be able to read your code and thus potentially steal your intellectual property since your files will have to be world-readable; second, if a user uploads personal files whose location is discovered by a malicious user, this malicious user actually has permissions to read and delete those files. suPHP addresses these problems by isolating users.

- For those who have prior programming experience, the PHP Language Reference may prove a useful tool for introducing the syntax conventions of PHP. For those with no prior programming experience, it can also be useful as an in-depth look at some fundamental programming concepts.

- Variables in PHP are loosely typed. This means that you don't have to specify a variable's type when declaring it. However, variables *do* have types associated with them in case you want to do an explicit conversion or cast between types. For example, all user input from forms is passed as strings, so you can cast it to an integer if you so desire.

- In PHP, there's a trade-off between the cleanliness of code and the speed with which it can be implemented. You may find yourself writing somewhat sloppy code (although PHP does provide you with the ability to define classes and organize your programs very rigidly), but realize that this might be a sacrifice worth making in the interest of getting something up and running very quickly and easily.

- Variables in PHP begin with $, as has already become obvious. The following types exist in PHP:
  - boolean
  - integer
  - float
  - string

  - array
  - object

  - resource
  - NULL

  - mixed
  - number
  - callback

- One other useful piece of reading material is the Explication of References. References are a way of passing variables using a construct somewhat similar to pointers. If you have very little prior programming experience, don't worry so much about this.

- PHP code is stored simply in a text file which begins with the tag `<?php` or `<?` (if the server is configured to interpret short tags via the `php.ini` file) and ends with the tag `?>`.
- PHP has a special set of variables called superglobals which allow you to access user input which has been submitted via forms as well as a whole slew of other useful data. Here's a list of superglobals:
  - `$_COOKIE`
  - `$_ENV`
  - `$_FILES`
  - `$_GET`
  - `$_POST`
  - `$_REQUEST`
  - `$_SERVER`
  - `$_SESSION`

5.3 Reimplementing Google (Again)

- Let's write a file called `search.php` and put the following lines of code in it:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Fake Google</title>
</head>
<body>

    hello, world

</body>
</html>
```

  Notice that all of this code is just XHTML—none of it is PHP. PHP files, however, can have XHTML within them.

- Now we'll set the `action` attribute of our Fake Google search page to be `search.php`. When we click Submit, we'll see "hello, world" displayed in our browser.
- Now let's see if we can access the user's query:

```
<?php

    print($_GET["q"]);

?>
```

We have to make sure, of course, that the `method` attribute of our form is specified as `GET`. Now the user's query will be printed out to the browser window.

- Let's add another parameter to the search page:

  ```
  <input type="checkbox" name="random" value="yes" />Click me for random
  ```

  We can see that this second field will be appended to the URL when we click Submit. We can print out this second form value like so:

  ```
  <?php

      print($_GET["q"]);
      print("<br />";
      print($_GET["random"]);

  ?>
  ```

  We can accomplish the same thing by interspersing XHTML and PHP:

  ```
  <?php print($_GET["q"]); ?>
  <br />
  <?php print($_GET["random"]); ?>
  ```

- If we want to see all of the contents of `$_GET` or even `$_SERVER`, we can use the `print_r` function.
- PHP also has arrays, an extremely useful data structure which pairs keys with values, and loops, a construct which allows chunks of code to be repeated multiple times.