Computer Science 75
Spring 2009
Scribe Notes

Lecture 1: February 2, 2009
Andrew Sellergren

Contents

Computer Science 75 Lecture 1: February 2, 2009
Spring 2009 Andrew Sellergren
Scribe Notes

1 Announcements (0:00–2:00)

- O hai! Welcome back to Computer Science E-75: Building Dynamic, Scalable Websites!
- 2 new handouts this week, available online.
- Enrollment is up to 142 students! That means we'll be having more like 6 or 7 TFs as opposed to 2. Whew. Say hello to Jesse Cohen, our newest addition, a junior at Harvard College.

2 Project 0 (2:00–10:00)

- As we mentioned last time, for Project 0 you'll be getting yourself set up on teh interwebs by buying a domain name and configuring it to use the course's web server. To do so on GoDaddy, you'll type in your chosen domain name, click through a series of attempted upsells by selecting "No thanks,"[1] and checking out. Once you've purchased your domain name, you'll need to login to GoDaddy, click on the Nameservers icon, and enter in `ns1.cs75.net` and `ns2.cs75.net` as custom nameservers.
- Note that you should resist the temptation to navigate to your domain name until you've completed the above steps. If you try to access your domain name before you've told the world where it's hosted, you'll be directed to a GoDaddy IP address which will then be cached by your web browser or by DNS servers or both. Then you might have to wait up to a day for the true IP address to propagate.
- There's one more step to this set up process that we've neglected so far. You'll need to tell us, by completing the Project 0 survey, what your domain name as well as your FAS username is. Once you give us this information, we'll give you login credentials for `cs75.net`.
- After you've been assigned a username and password, you can login to `panel.cs75.net`. Interestingly, you'll notice when you first attempt to access this URL, you'll be warned of a possible security violation. This is because the course's SSL (Secure Sockets Layer) certificate has expired. For the most part, SSL certificates pay lip service to security without actually heightening security. These days, almost anyone can purchase an SSL certificate and thus it has no real correlation to how trustworthy a website actually is.
- In Firefox, to get through to the login forms, you'll need to click "Or you can add an exception..." followed by the Add Exception button, the Get Certificate button, and finally the Confirm Security Exception button. Make sure the checkbox which says "Permanently store this exception"

[1] Or, if you're what we might call a "sucker," by adding these features to your cart.

is checked so that you never have to go through this process again (or at least until you clear your browser cache). In Google Chrome, this process is as simple as clicking "Proceed anyway."

- Once you login to DirectAdmin and click DNS Management, you'll see a lot of familiar DNS records which we reviewed last lecture. If you decide to tinker with these, be sure to make a note of all the changes you make so that you can undo them if you need to.
- One of your tasks for Project 0 will be to create a subdomain which will map to a directory that contains all of Project 0's files. You will do the same for the rest of your projects so that they will all be housed on a single domain in an organized fashion.

3 Apache and Web Server Configuration (10:00–50:00)

- 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.
- One of the features which Apache (and also IIS) provides is virtual hosting. Recall from last time that this enables a web server to host more than one domain name. Tucked into the HTTP header is a line of text which begins with "Host:" and represents what the user actually typed into his address bar. In this way, the web server that hosts multiple domain names can know *which* webpage to spit back out when it receives a request.
- One of the gotchas of SSL certificates is that you need a dedicated IP address for an SSL-based website. The reason is that some of the header information is encrypted and can only be decrypted by the appropriate host. But how could a web server that hosts multiple domain names know *which* domain name is the correct one until it has decrypted this information? Here we have a chicken and the egg problem, clearly. As a result, if you want an SSL certificate for your site, you're going to have to pay an additional monthly fee to a commercial host so that it will provide you with a unique, dedicated IP address.
- The configuration file for Apache is `httpd.conf`. Below you'll see an excerpt from the course server's configuration file:

```
<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. 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`.
- `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).
- What's the deal with the suPHP lines? 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.

- A few more details about DirectAdmin (in answer to student's questions):
  - It places each user in his own user group, thus `malan` in the user group `malan`.
  - It hardcodes the port 80 into the `VirtualHost` tag, but this is actually not necessary, but simply to be explicit.
  - It runs its own web server which isn't meant to be accessed except by itself, so any other web server actually has to live at a port other than 80. In this case, it is port 2222, which you'll notice you are automatically redirected to when you visit `panel.cs75.net` thanks to a PHP script.
  - Virtualmin is similar in spirit and functionality to DirectAdmin.
- If you recall from last week, one of our goals was to make sure that `malanrouge.com` and `www.malanrouge.com` were both functional. David issued a challenge to find a website which didn't accomplish this and mentioned that you shouldn't look very far. Sure enough, a few students discovered that our very own Harvard requires that you type in `www.harvard.edu`, not just `harvard.edu`. For shame!
- Next unofficial homework assignment: figure out *why* Harvard's website doesn't have this functionality. Is it because there's no DNS record for `harvard.edu`? Is it a virtual hosting problem such that the config file doesn't have a server alias listed? Try messing around with `ping` and `nslookup` and see what you can find out.
- Let's make sure that `malanrouge.com` doesn't have the same problem. In fact, we want to make sure `malanrouge.com` actually gets remapped as `www.malanrouge.com`, if only for the sake of branding. How do we accomplish this? As ugly as it seems, we can do this by pasting the following lines of code into the `.htaccess` file on our Apache build:

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

Because we turned on all the options for you in the `httpd.conf` file, you are now empowered to create a `.htaccess` file in your `public_html` directory which will be accessed by Apache everytime a request is made for any file in that directory or any subdirectory therein.

- 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 m̂eans "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."

- When in doubt, clear your cache! After we've done this, we notice that the `www` gets added in now when we navigate to `malanrouge.com`. Why is this good? Firstly, there's the branding issue. Secondly, search engines like Google will respect these response codes and cache the address which you redirect all your traffic to. Why might this be bad though? Well, your servers are going to be hit twice by people who type in the URL which you are redirecting. Then again, Google does this, so it can't be but so bad in terms of load.

- We began our story by adding DNS records for both `www.malanrouge.com` and `malanrouge.com`, so at this point in our story, both requests will have been directed here. Only this `.htaccess` file will then cause them both to be funneled to the same URL.

- Would it be possible to exploit the fact that `harvard.edu` does not resolve? A disgruntled employee with access to Harvard's DNS servers could, of course, add an A record to redirect that traffic elsewhere. Beyond that, anyone who has control over DNS responses between you and the rest of the internet could do likewise. For example, some ISPs have actually gotten in trouble for redirecting unregistered domain names to websites containing advertisements.

- It's worth noting that we could've, in fact, accomplished the same redirect using a simple PHP script, but this isn't really an optimal solution.

- A quick note about XAMPP: as we mentioned last time, this 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.

4 PHP (50:00–97:00)

- 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.

- Unlike C or C++ or other compiled languages, PHP is an interpreted language. This means it is not as high-performing, but for the purposes of rendering web content and accessing databases, it is perfectly well-suited. With add-ons like memcached, PHP's performance can even be improved.

- 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) and ends with the tag `?>`.

- What do we mean when we say PHP is installed on a server? That means that any request for a file ending in `.php` is not served up as static content but is rather passed to a program on the local server (e.g. `/usr/bin/php` or `php.exe`) which *interprets* the code it contains. After interpreting the code, the content which is output by the PHP program is passed back to the user's browser.

- Debugging PHP is not necessarily as straightforward as it might be. To help you out, we've installed some modules that will spit out debugging info like line numbers in the event of an error.

- Recall from last time our introduction to forms. Let's apply this to the real world and 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!

- So what will the HTTP header look like when we enter a query of foo into Google and click Google Search? It will look something like the below:

  ```
  GET /search?q=foo HTTP/1.1
  ```

Recall from last time the GET and HTTP/1.1. Something new, however, is the search?q=foo after the forward slash. All this means it that we're requesting not the root directory, but rather a file called search. More than that, we're passing it a parameter called q which in this case holds the value of our search query. If there were other parameters being passed via the GET method, they would be separated by the & symbol.

- Another type of HTTP request is the POST method. One of the problems with the GET method, as you might have guessed, is that it must be tacked onto the end of the URL. This means that the length of the URL imposes limitations on the number and type of variables that may be passed. In addition, any variables that are passed are not obscured. Passwords, for example, should definitely not be transmitted via the GET method. The POST method, on the other hand, provides at least some measure of security and imposes no limitations on the amount of data that can be transmitted. Data transmitted via the POST method is transmitted in the *body* of the HTTP request as opposed to the headers.

- So how might we fake an HTTP request like the above? Check out the lines of code below:

```
<form action="/search" method="get">
  <input type="hidden" name="q" value="foo" />
  <input type="submit" value="Fake Query" />
</form>
```

  When we click the Fake Query submit button, the value of foo will be passed as the variable q to the file search, just like the HTTP header above implied.

- If we change the lines above like so, we could actually implement our own Fake Google:

```
<div align="center>
<h1>Fake Google</h1>
<form action="http://www.google.com/search" method="get">
  <input type="text" name="q" />
  <br />
  <input type="submit" value="Fake Google Search" />
</form>
</div>
```

  Notice that we've changed the action attribute to be the hardcoded web address of Google Search. That's, of course, because if we upload this to our own web directory, we won't have a search file as Google does on their servers.

- So what if we needed to transmit login information rather than a simple search query? As mentioned before, we'd want to use the `POST` method rather than the `GET` method. It's worth noting that even if you change the type of HTML input to be a `password` field, the information will still be transmitted over the wire in plain view. Not ideal, especially considering that it will be cached and saved in the browser history, etc.

- What's the final step to implementing Fake Google? If we want to recreate the logo, we can go to http://googlefont.com/, type in Fake Google, and download a GIF. Then, we need to SFTP the file to our web directory (perhaps using a client such as SecureFX) and change the permissions on it so that it's world-readable. Finally, we include the following line of code:

  ```
  <img src="logo.gif />
  ```

  That's all there is to it!

- Still, we haven't accomplished very much. After all, we didn't *actually* implement the search functionality of Google. But perhaps we can begin to at least implement some server-side dynamic code. Finally, some PHP:

  ```
  <?php
    print("hello world);
  ?>
  ```

  If we place this code in a file called `search.php`, which we don't, in fact, need to change the permissions on, we see that we will get "hello, world" outputted to the browser. Not very impressive, but, hey, at least it works, right?

- What if we include the PHP code above in our original XHTML implementation of Google. Wouldn't you know it, it still works! That's the beauty of PHP is that it can be streamlined with static XHTML content. In fact, a file which contains *only* XHTML content but which has the file extension `.php` can still be considered a perfectly functioning PHP program.

- Now, let's spice up the code a little bit. Take a look at the below:

  ```
  <?php

    print("Your query was: " . $_GET["q"]);

  ?>
  ```

Note that `.` is the concatenation operator and that `$_GET` is a superglobal variable which automatically is loaded up with the data that has been passed to it via the `GET` method. In fact, it will be an associative array indexed by the names of the variables we have been passed. If we type in "something random" into our search box, we will get "Your query was something random." outputted to the browser.

- We can also change the HTTP request to `POST` and access the same variable using the superglobal `$_POST` as opposed to `$_GET`.
- Even though we have comingled PHP with XHTML, the PHP is not actually outputted to the browser. This goes back to its nature as an interpreted language.
- What if we want to view all the contents of the `$_GET` variable? We can use the code below:

```
<pre>
<?php

  print_r($_GET);

?>
</pre>
```

  This will simply display the entirety of the superglobal array. The `<pre>` tags imply that their content is pre-formatted text, so it will display a little more cleanly. This is a useful debugging trick since the `print_r` function will recursively print out the contents of *any* variable, including objects.

- 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

  PHP is very loosely typed. None of these types are declared explicitly, and there's a lot of implicit conversion which goes on. If you add `'1'` and the integer `2`, you'll get back the integer `3`.

- This is arguably an advantage of PHP because, at the end of the day, all the variables which are being passed to PHP via HTTP requests are simply strings.
- Nonetheless, data types *are* distinguishable. The manual will always specify what data types are returned. Still, when you declare a variable, you don't specify its type.
- In PHP, there exists not only the assignment operator `=` and the equality operator `==`, but also the identity operator `===`. This means that not only the value but also the type of the two operands will be checked.
- One of the advantages of PHP is that functions can return several different data types. Herein lies the usefulness of the identity operator.
- Here's some reading material: `http://us2.php.net/manual/en/language.references.php`.
- Aaaaand a list of the superglobals:
  - `$_COOKIE`
  - `$_ENV`
  - `$_FILES`
  - `$_GET`
  - `$_POST`
  - `$_REQUEST`
  - `$_SERVER`
  - `$_SESSION`

  Note that the `$_REQUEST` variable will contain all the data passed by either the `GET` method or the `POST` method, but its use is discouraged because we think it's important that you exercise control over your code as far as possible.
- If you wanna see a whole ton of information that is always accessible by the server, try printing out the contents of the `$_SERVER` variable. Remember how?
- Know that there's a whole slew of array functions available in PHP!