DAY 2!

Logs, Aliases, Redirects, Rewrites, and More! Oh My!

VIRTUAL HOSTING

OVERVIEW

- Virtual Hosting is an extremely popular feature of the Apache webserver.
- Virtual Hosting allows Apache to serve up more than one website, based on:
  - Destination IP address (IP-based virtual hosting)
  - Destination Port (Port-based virtual hosting)
  - Destination Name (Name-based virtual hosting)

VIRTUALHOST

- The <VirtualHost> directive defines a new Virtual Host in the Apache configuration.
- The directive takes one argument, which defines how Apache will identify requests for the virtual host.
- Valid values include:
  - IP address, Hostname, `*`, `_default_`

NAMEVIRTUALHOST

- Named-based virtual hosting requires special discussion.
- In a normal scenario, Apache identifies and routes requests to the filesystem based off of standard TCP information, such as IP address and port.
- By definition, name based hosting means Apache has to identify requests with a name instead of an IP address. The `NameVirtualHost` directive tells Apache which IP address/port combinations should use Name matching.

APACHE CONFIGURATION

- You can find this example Apache VirtualHost definition at the bottom of `httpd.conf`:

```
<VirtualHost ____________>

    ServerName name

    ServerAlias alias

    DocumentRoot path

    CustomLog /path/to/access_log combined

    ErrorLog /path/to/error_log

</VirtualHost>
```

LAB

1. Configure two websites on your server. "X" represents your station #.

2. `wwwX.example.com` should be served from "`/var/www/html/wwwX`" and should also respond to requests for the short hostname `wwwX`.

3. `vhostX.example.com` should be served from "`/var/www/html/vhostX`". `vhostX` and should also respond to requests for the short hostname `vhostX`.

4. Both should be listening on your primary ip address, but `wwwX.example.com` should be the default site.

   Hint: Read the documentation closely, especially the Virtual Hosts user's guide

LAB

1. Create a third vhost that uses port-based hosting to run on port 81. This site should serve content from "`/var/www/html/port81`". Note there are several steps to get this working!

LOGS

- As discussed previously, Apache has a significant logging engine built in.
  - Error Logs provide details about the status of operations, such as unsuccessful requests, configuration errors and warnings.
  - Access Logs provide details on every single request made to the server, regardless of the status of the response. Mostly used to track server utilization and feed statistics tools.

LOGS

- In the lab environment, these files are located:
  - `/var/log/httpd/error_log`
  - `/var/log/httpd/access_log`
- Recall that the locations can easily be changed via the **`ErrorLog`** and **`CustomLog`** directives, and the directives can be used repeatedly in different contexts.

LAB

1. Spend a few minutes looking over the existing log files. Try various valid and invalid requests to your server and observe the log entries generated.

2. Modify your configuration such that each of your Virtual Hosts is logging to it’s own access and error logs. Come up with a logical directory layout or naming scheme for the log files.

3. Create a new log format for the default server access log and use it in place of the combined format. Examine the resulting logs.

URL HANDLING

- As discussed in the configuration file walkthrough, Apache *normally* maps a url path into a filesystem path starting from the DocumentRoot, but there are various tools available to modify this behavior, most importantly:
    - Aliases
    - Redirects
    - Rewrites
- `http://httpd.apache.org/docs/2.2/urlmapping.html`

ALIASES

- Aliases are the simplest configuration directive used to add filesystem locations outside the DocumentRoot to the web space served by Apache. Consider:

- **`DocumentRoot /var/www/html`**

- A request for:

  `http://www.site.com/downloads/myfile.zip`

  results in Apache returning

  `/var/www/html/downloads/myfile.zip`

ALIASES

- Now consider the same server:

  ```
  DocumentRoot /var/www/html
  ```

- But, throw in an Alias for /downloads:

  ```
  Alias /downloads /var/downloads
  ```

- A request for:

  ```
  http://www.site.com/downloads/myfile.zip
  ```

  now results in Apache returning

  ```
  /var/downloads/myfile.zip
  ```

LAB

1. Add a folder, `/var/www/html/products`. Under this folder, create subfolders with names matching each of your three vhosts (`wwwX, vhostX, port81`). In each of these folders, create a couple of dummy product files.

2. Extend your configuration such that when a user requests a page using a url beginning with “`/products`”, they are sent to the appropriate `/var/www/html/products` folder for the site they are browsing.

REDIRECTS

- Redirects are a useful way to send a visitor to a different URL, either temporarily or permanently. Redirects differ from Aliases because the server does not respond with any content. Instead, the response tells the user to request a new URL.
- For example:
- `Redirect permanent /old http://www.site.com/new`
- Forces users requesting anything under `/old` to be permanently redirected to `/new`
- `www.site.com/old/boxes/one`   is redirected to:

  `www.site.com/new/boxes/one`

LAB

1. Create a new file, `/var/www/html/wwwX/whizbang-2000.html` and briefly describe your whizbang 2000 invention. Verify the page through firefox.

2. Invent whizbang-3000! Create a file for your amazing new invention! Create a “See other” redirect from the 2000 page to the 3000 page and see how it works.

REWRITES

- URL Rewriting is a very powerful feature of Apache, provided by the mod_rewrite module.
- With rewriting, it is possible to interpret and manipulate an incoming request URL in almost any manner.
- It is also possible to make yourself insane. :)
- We will focus on the basics and leave the padded cells for your own exploration.

REGULAR EXPRESSIONS

- Before any meaningful conversation about mod_rewrite can occur, another topic must be covered: **regular expressions**.
- Bum bum bummmmmmm!
- We will briefly introduce regular expressions. For more information, consult one of the many *tomes* that have been written on the topic.

REGULAR EXPRESSIONS

- Apache recognizes the Perl Compatible Regular Expression syntax.

- A regular expression is a *pattern* used to describe a particular input that the user is interested in. For example:

  ```
  .*firefly.*
  ```

  When interpreted as a regular expression, will match any input with the word “`firefly`” somewhere in it.

- What?!?!??

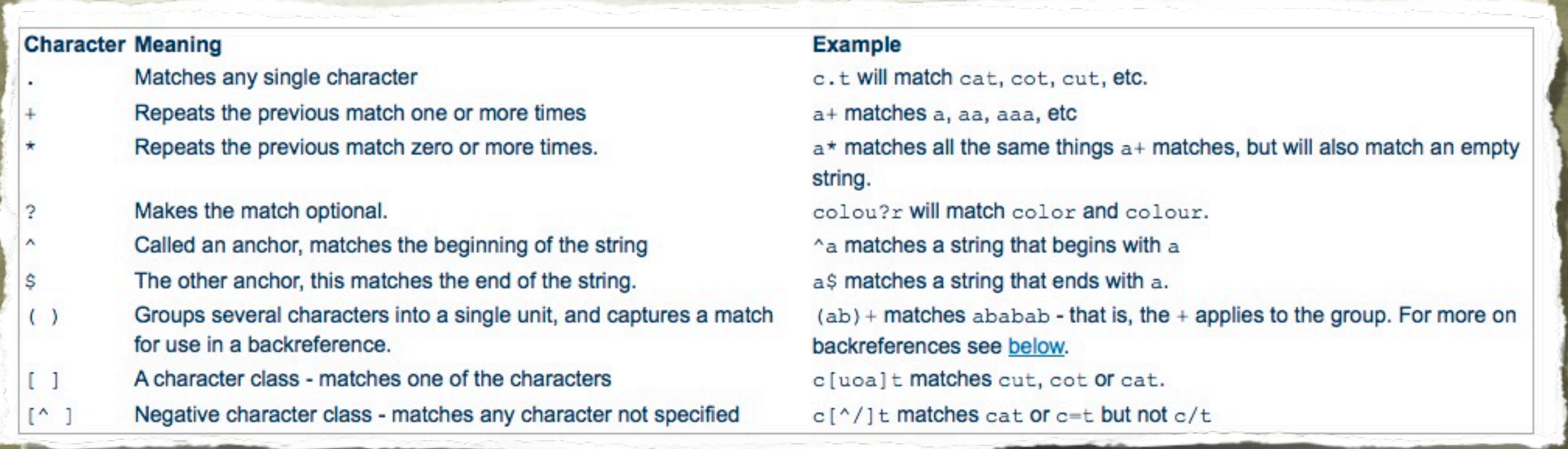
REGULAR EXPRESSIONS

- `.*firefly.*`   What's going on here, you ask?
- The period is a special character in RE syntax. The period will match any <u>single</u> character.
- The asterisk is also a special character. It allows the previous match to repeat <u>zero or more</u> times.
- So, "`.*`" means to match zero or more characters, any characters!
- Put that around the word firefly, which does not contain any special characters, and you match zero or more characters before and after the word "firefly". Or more simply, anything with "firefly" in it!

REGULAR EXPRESSIONS

- A fantastic RE guide, ripped straight from the Apache docs:

  `http://httpd.apache.org/docs/2.2/rewrite/intro.html`



Character	Meaning	Example
`.`	Matches any single character	`c.t` will match `cat`, `cot`, `cut`, etc.
`+`	Repeats the previous match one or more times	`a+` matches `a`, `aa`, `aaa`, etc
`*`	Repeats the previous match zero or more times.	`a*` matches all the same things `a+` matches, but will also match an empty string.
`?`	Makes the match optional.	`colou?r` will match `color` and `colour`.
`^`	Called an anchor, matches the beginning of the string	`^a` matches a string that begins with `a`
`$`	The other anchor, this matches the end of the string.	`a$` matches a string that ends with `a`.
`( )`	Groups several characters into a single unit, and captures a match for use in a backreference.	`(ab)+` matches `ababab` - that is, the `+` applies to the group. For more on backreferences see below.
`[ ]`	A character class - matches one of the characters	`c[uoa]t` matches `cut`, `cot` or `cat`.
`[^ ]`	Negative character class - matches any character not specified	`c[^/]t` matches `cat` or `c=t` but not `c/t`

REWRITING

- So what, you ask, does regular expressions have to do with URL rewriting in Apache? Everything!
- The main directive used for URL rewriting is:

```
RewriteRule pattern substitution [flags]
```

- Where `pattern` is a regular expression to match against the request URL-path (the part after the host/port), `substitution` is where the request should go if the pattern matches, and `flags` is an optional list of rewrite engine modifiers.

REWRITING

- As you have probably heard through the grapevine, it’s quite complex. Regular expressions are bad enough, but then consider:
- The `substitution` parameter can be:
  - An absolute filesystem path
  - A web-path
  - An absolute URL [possibly] to another site altogether!

EXAMPLES!

- OK, enough with the painful syntax.. Let's see some examples!
- `RewriteRule ^/current$ /2011`
  - `site.com/current` internally rewrites to `site.com/2011`

-
  ```
  RewriteRule ^/partners/acme-co$
    http://www.acme-co.com/site.com-referal [R]
  ```
  - This interesting rule rewrites

    `www.site.com/partners/acme-co` to

    `www.acme-co.com/site.com-referal`

HOMEWORK

- Time for some homework!
- Ok, not really.. But you can see that URL rewriting is extremely complex, and we've only just scratched the surface..
- Documentation is your friend, here.. But don't expect to get it the first or thirtieth time you've read the docs..
- Rewrite proficiency is only gained through long, hard experience and practice. Check out:
- `http://httpd.apache.org/docs/2.2/rewrite/rewrite_guide.html`

LAB

1. Use a RewriteRule to send requests to "`vhostX/whizbang-2000.html`" over to the correct `wwwX` location.

2. Look up the `AliasMatch` and `ScriptAliasMatch` directives. Create a new folder `/var/www/html/produce`. Create an `AliasMatch` configuration which will send any request for "`/apple`", "`/banana`", or "`/pear`" on the port81 vhost over to the produce folder. Do this with one `AliasMatch`.

```
stationX:81/apple -> /var/www/html/produce
```

SCRIPTING

- Scripting involves making Apache *execute* a file and return it's *output*, as opposed to simply returning the file itself.
- There is an entire framework for facilitating this operation, and allowing the webserver to communicate basic information to script through the use of environment variables, and sometimes input.
- This is known as CGI scripting, or Common Gateway Interface scripting.

BASIC SCRIPTING

- Some of the simplest scripting requires only a shell script. Consider:

```
#!/bin/bash

echo -e “Content-type: text/html\n”

echo “<h1>Hello world!</h1>”
```

BASIC SCRIPTING

- If we put the appropriate execute permissions on the script, then we can see it output the expected content at the command line:

```
# chmod +x myscript

# ./myscript

Content-type: text/html



<h1>Hello world!</h1>
```

BASIC SCRIPTING

- If this file is placed in a location identified to Apache as supporting executables (CGI scripts), then we have a working CGI!

LAB

1. Look up the `ScriptAlias` directive. Use this directive and your simple shell script to create a simple, dynamic website. Maybe have it report the current date and time with the `date` command.

SSL

- SSL, or Secure Sockets Layer, allows for a complete end to end encryption of the information sent between client and server.
- In order to implement SSL, Apache needs:
  - mod_ssl
  - A private key and a paired certificate
  - Various additional configuration directives controlling the SSL engine and encrypted stream (see `ssl.conf`)

SSL

- Public key cryptography 101:
    - Server generates a private and public key pair
    - Server generates a "certificate signing request" which includes the public key, information about the site, and is signed by the private key. This CSR is sent to a "certificate authority" for verification.
    - Once the CA certifies the identity of the site, commonly via phone, letterhead or snail mail, the CA returns a "certificate", which is signed by the private key of the CA.
- This creates a ring of trust, since a web user "trusts" the certificate authorities, and the CA's have "trusted" a website, you can trust the website! Amazing, I know!

CERTIFICATES

- So continuing, how does this certificate identify and encrypt the communication?
  - Client connects to server, requesting secure URL (https://)
  - Server responds with certificate (includes public key)
  - Client responds with random encryption key encrypted with public key, as well as details of web request.
  - Server decrypts request with private key, and uses the random encryption key for encrypting the response. It's magic!

SAY WHAT?!

- Ok, so you can see there is some pretty serious complexity going on here. Take a look at `ssl.conf` for even more fun, detailing protocol support, ciphers, caches, pseudo random numbers and more! Yee-haw!
- 95% of the time, the main interest for server admins lies in pointing Apache to the appropriate certificate and key files, via:
    - `SSLCertificateFile`
    - `SSLCertificateKeyFile`

LAB

1. Complex lab alert. Work together and ask questions!

2. Use the `genkey` command with the `--test` flag to create a test certificate for use with your Apache server. Create the cert for `wwwX.example.com`. Use defaults for everything.

3. Once the key and certificate are created, locate them buried under the `/etc/pki` folder. Plug them into your configuration for `wwwX` and test. Use `conf.d/ssl.conf` as a reference. Minimum directives: `SSLEngine`, `SSLCertificateFile`, `SSLCertificateKeyFile`.

```
slideshow.end();
```