

Thoughtful Solutions

Creatively Implemented and Communicated
<<http://very.thoughtful-solutions.info/>>

Apache 2 Training

Abstract

Systems training on Apache 2 for Microsoft Windows



Attribution-ShareAlike 2.0 United Kingdom

You are free:

- to copy, distribute, display, and perform the work
 - to make derivative works
 - to make commercial use of the work

Under the following conditions:



Attribution. You must give the original author credit.



Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under a licence identical to this one.

- For any reuse or distribution, you must make clear to others the licence terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

This is a human-readable summary of the [Legal Code \(the full licence\)](#).

[Disclaimer](#)

Apache 2 Review on Win32.....	7
Introductions.....	8
Systems Administration Philosophy.....	9
What We Believe.....	10
What We Do.....	11
Why We Do it.....	12
Automation.....	13
Naming Conventions.....	14
Centralised Management.....	15
Troubleshooting Best Practice.....	16
Objectives.....	17
Interview the Customer.....	18
Verify the Problem.....	19
Research the Problem.....	20
Isolate the Problem.....	21
Resolve the Problem.....	22
Cleanup.....	23
Understanding HTTP.....	24
HTTP.....	26
Uniform Resource Locators (URL).....	27
URL-encoding.....	28
More URL encoding.....	29
Basic HTTP - Client Request.....	30
Basic HTTP – Server Response.....	31
Beginning of Web Technologies.....	32
Limitations of Static Content.....	33
Why generate dynamic content?.....	34
Dynamic Content Technologies.....	35
Evolution of Dynamic Content.....	36
HTTP is stateless.....	37
LAB	38
Controlling the Apache processes.....	39
Controlling the Apache Processes.....	40
Overview of the Server Structure.....	41
The Apache Processes.....	42
Running Apache as a Service.....	43
Starting and Stopping Apache.....	44
Apache Service Monitor.....	45
Apache Online Documentation.....	46
The httpd.conf file.....	47
httpd.conf.....	48
Structure of httpd.conf.....	50
httpd.conf parameters.....	51
LAB.....	54
Configuring the server-status Screen.....	55
LAB.....	56
Modifying Server Locations	58
LAB.....	59
Filesystems & Webpace.....	60
DocumentRoot.....	61

Files Outside the DocumentRoot.....	62
File System Containers.....	63
What are directives ?.....	64
Webspace Containers.....	65
Wildcards and Regular Expressions.....	66
What to use when.....	67
How the sections are merged ?.....	68
A Merging Example.....	69
LAB.....	70
Apache Modules.....	71
Apache Modular Approach.....	72
Apache Module Types.....	73
How Modules Work.....	74
Important Apache Modules.....	75
Third Party Installation Methods.....	76
Compare Two Methods.....	77
Dynamically Linked Modules.....	78
Module Installation Steps.....	79
Another Installation Method.....	80
LAB.....	81
Virtual Hosting.....	82
Virtual hosting.....	83
IP Based Virtual hosting.....	84
Name based Virtual Hosting.....	86
ServerAlias.....	87
Dynamically configuring paths.....	88
Gotchas.....	89
Statically Configured Mass vhosting.....	90
Simple Dynamic Virtual Hosting.....	91
More efficient IP-based virtual hosting.....	92
Separate virtual hosting configs.....	93
Monitoring.....	94
Apache Logging.....	95
Error Logging.....	96
Error Log Settings – cont.....	97
Request Logging.....	98
Common Log Format (CLF).....	99
Conditional Logging.....	101
Other Logs.....	102
Tracking User Sessions.....	103
Cookies.....	104
Cookies (cont.).....	105
Session Tracking.....	106
Analyzing Logs.....	107
Rotating Logs.....	108
mod_proxy.....	109
What is a proxy ?.....	110
Example proxy configuration.....	111
Controlling access to your proxy.....	112
<Proxy> Directive.....	113

mod_rewrite.....	114
Voodoo for URL Manipulations.....	115
RewriteRule.....	116
Regular Expressions.....	117
Regular Expressions - cont.....	118
Searching and Regexp's.....	119
\ and ^ operators.....	120
\$ and [] Operators.....	121
Parenthesis ().....	122
Curly Braces {} and Repetition.....	123
Other Operators and Repetition.....	124
Greedy Evaluation.....	125
Optionality and Repetition.....	126
A Simple Exercise.....	127
RewriteRule Flags.....	133
RewriteRule Flags - cont.....	134
RewriteCond.....	135
Condition Patterns.....	136
Logging.....	137
Useful Rewrite Example.....	138
Useful Rewrite Example - cont.....	139
Useful Rewrite Example - cont.	140
http Authentication.....	141
Web Security Issues.....	142
User Authentication.....	143
Who Uses Flat Files?.....	145
How Authentication Works (1).....	146
How Authentication Works (2).....	147
How Authentication Works (3).....	148
How Authentication Works (4).....	149
Password Protection of Web Pages.....	150
Basic Authentication.....	151
How Basic Authentication Works.....	152
Problems with Basic Authentication.....	153
Digest (Challenge/Response) Auth.....	154
Challenge and Response.....	155
Advantages of Digest over Basic Auth.....	156
Availability of Digest Authentication.....	157
Dynamic Content with CGI & Perl.....	158
Creating Dynamic Content.....	159
The Common Gateway Interface (CGI).....	160
CGI (Common Gateway Interface).....	161
Server API for CGI.....	162
CGI Example (HTML form).....	163
CGI Example (GET).....	164
CGI Example (POST).....	165
CGI Environment Variables.....	166
LAB	167
Enabling CGI in Apache.....	168

LAB.....	170
Evaluation of CGI.....	172
Server-Side Scripts and Includes.....	173
Integrating Tomcat into Apache.....	174
What is Tomcat?.....	176
J2EE.....	177
Installing Tomcat Binaries.....	178
Tomcat Welcome Page.....	179
Integrating Tomcat with Apache.....	180
Using a Proxy.....	181
Using the mod_jk connector.....	183
LAB:.....	184
Dynamic Content servlets and JSP.....	185
Java Servlets - Introduction.....	186
Java Servlets – Introduction (cont.).....	187
A very simple Servlet.....	189
Advantages of servlets over CGI.....	190
Introduction to JSP.....	191
Benefits of JSP.....	192
JSP Vs Servlets.....	193
Simple JSP example.....	194
LAB:.....	195
Mod_php.....	196
What is PHP ?.....	197
What can PHP do ?.....	198
A Simple PHP Example.....	199
PHP Coding Style.....	200
PHP Dynamic Content.....	201
Accessing Form Variables.....	202
Strings, Variables & Literals.....	203
Identifiers.....	204
PHP Data Types.....	205
Type Strength.....	206
Type Casting.....	207
Variable Variables & Constants.....	208
Variable Scope.....	209
Operators.....	210
More Operators.....	211
Other Operators +.....	212
Variable Functions +.....	213
Conditional Control Structures.....	214
Iterations +.....	215
Installing PHP as an Apache Module.....	216
Basic Debugging.....	217

Apache 2 Review on Win32

Apache 2 Review on Win32



Edmund J. Sutcliffe

Thoughtful Solutions

Introductions

Introductions



- Who are you ?
 - What experience do you have ?
 - What are your expectations ?
 - Three things you like ?
-

Systems Administration Philosophy

Systems Administration Philosophy



"We will encourage you to develop the three great virtues of a programmer: *laziness, impatience, and hubris.*"

Larry Wall

Laziness

The quality that makes you go to great effort to reduce overall energy expenditure. It makes you write labor-saving programs that other people will find useful, and document what you wrote so you don't have to answer so many questions about it. Hence, the first great virtue of a programmer or admin

Impatience

The anger you feel when the computer is being lazy. This makes you write programs that don't just react to your needs, but actually anticipate them. Or at least pretend to. Hence, the second great virtue of a programmer or admin.

Hubris

Excessive pride, the sort of thing Zeus zaps you for. Also the quality that makes you write (and maintain) programs that other people won't want to say bad things about. Hence, the third great virtue of a programmer or admin.

What We Believe

What We Believe



- There to make our lives easier
 - Make them do the work
- Better at repetitive tasks than humans
 - Scripts automate boring tasks
- Managing systems should be easy
 - Less hassle is more uptime
 - More time for beer!

We strongly believe that Computers are there to make our lives easier. They are better at repetitive tasks than humans and that managing systems should be invested in to make our lives easier.

What We Do

What We Do



- Automation
 - Scripts to install OS
 - Scripts to install applications
 - Scripts to maintain system
 - Naming conventions
 - Separation of OS, applications and data
 - Centralised management
 - DNS
 - LDAP
-

Why We Do it

Why We Do it



- Consistency! Consistency! Consistency!
 - Hand installed operating systems and applications are never the same twice
 - Scripted builds and application installs will always be the same as the scripts
 - Scripted builds should be self documenting
 - Scripts can parse log files for problems far better than humans

Consistency is the last refuge
of the unimaginative

Oscar Wilde

Automation

Automation



- Automation provides repeatability
- Repeatability provides
 - Consistency
 - Faster recovery

One of my primary objects is to form the tools so the tools themselves shall fashion the work and give to every part its just proportion.

Eli Whitney

Naming Conventions

Naming Conventions

- Separate ownership and so, responsibility
 - Leave original copies of OS config files behind
 - Always know base config
 - /etc/hosts, /etc/hosts.Solaris9
 - Installed applications clearly separated from OS
 - Own directory, users, groups etc.
 - e.g. /domain/Oracle/10.1/etc...
 - Versions separated
 - /domain/Oracle/current -> /domain/Oracle/10.1

A place for everything and everything in its place.
Samuel Smiles

The separation of each application into being owned by a specific users allow delegation of its management to that specific users. Associate with this application owner we also impose a group. The reasons for this group associated with this application, is to impose access control to the application. Thus unless you are a member of this group; you are unable to access this application.

This is important for cost control of the licenses but also it can be extended to be part of the control procedure associated with various information sources, which eventually end up on storage.

The fact of encoding the application version into the path, and encouraging users to make of the “current” symbolic link or junction point, allows the central administration to move versions without breaking the users view of the application; keep semantic correctness. Additionally this allows for people in development to test and examine past and future versions while still being in the same groups and structures.

Centralised Management

Centralised Management



- DNS and LDAP provide centralised management
 - Keep data in one place
 - Make changes in one place
 - Use replication to remove single point of failure
 - DNS
 - Multiple machine names
 - Service starting by machine (SRV records)
 - LDAP
 - Access controls available everywhere, consistently
-

Troubleshooting Best Practice

Troubleshooting Best Practice



An undefined problem has an
infinite number of solutions.

Robert A. Humphrey

Objectives

Objectives



- Troubleshooting Best Practice
 - How to get free beer
-

Interview the Customer

Interview the Customer

- DO NOT give an immediate answer
- Find out what they are trying to do, as well as what doesn't work.
- Get details of any action already taken, and its effects.
- Ask "what has changed" - the single most important question.
- Ask isolation questions;
 - Does it happen every time?
 - Does it happen for all users?
 - etc



Verify the Problem

Verify the Problem

- Make sure the problem really exists
- Verify the information supplied by the customer
- Replicate the problem if possible / feasible



Research the Problem

Research the Problem

- Not many problems are NEW problems

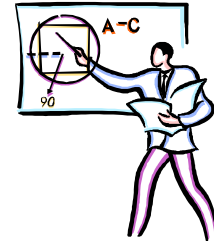
- Check sources of support information
 - Knowledge Bases
 - Forums
 - Vendor Tech Support
 - Your own support logs –
you do keep them don't you.



Isolate the Problem

Isolate the Problem

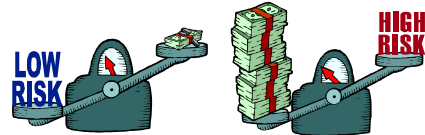
- Identify the components of the problem space.
- Isolate the problem by testing individual components
- Draw a picture
- Read the logs
- Use diagnostic tools
- Trial and Error:
 - Change one thing at a time
 - Put it back if it doesn't work !
 - Put it back if it does work.....



Resolve the Problem

Resolve the Problem

- Is solving the problem the right thing to do?
 - How long will it take
 - What will it cost
 - Is there any risk



- Are there alternative solutions and/or workarounds



Cleanup

Cleanup

- Test the Solution, Then Test the rest of the system.
- Get Customer Acceptance of the solution
- Follow-up on Outstanding or Arising Issues



Understanding HTTP

Understanding HTTP



"I just had to take the hypertext idea and connect it to the TCP and DNS ideas and -- ta-da! -- the World Wide Web."

Tim Berners-Lee

Understanding HTTP

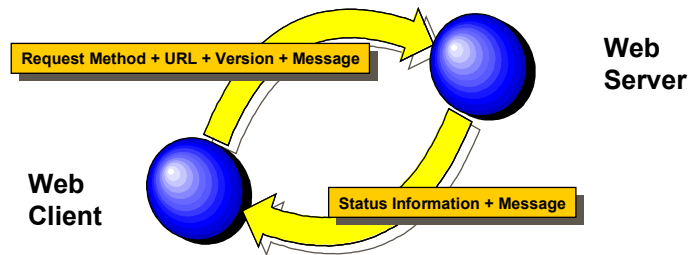


- URL structure
- URL encoding
- http requests and responses
- Static vs Dynamic content
- Keeping track of clients with http

HTTP

HTTP

- An 'extensible' protocol for client-server communication
 - Allows the client to specify the action to be performed by the server
 - Enables the negotiated transmission of varying types of data
 - e.g. Text, Graphics, Sound,



HTTP

HTTP (Hypertext Transfer Protocol) defines a request and response communication mechanism between Web client and server.

A client application submits a request of the following form to an HTTP server:

Request Method + URL + Version + Message

The Request Method specifies the purpose of the request; it informs the server how the following message is to be processed. The server deals with the request, responding with a message of the form:

Status Information + Message

The Status Information typically contains success or error codes, informing the client whether the request was successfully processed, or not. The status information is followed by a MIME header that specifies the type of data that the server is returning, followed by the data itself.

MIME encoding allows client and server to negotiate the data types to be transmitted between the two, enabling these to information other than plain ascii text, for example various graphics format, or sound. The display of this information need not be directly supported by the client itself - this can call on 'helper' applications to view (or play), the information transmitted.

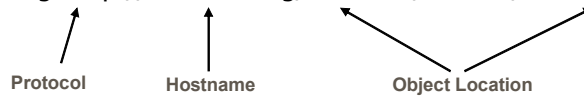
The use of the Request Method, together with MIME encoding to represent message content, combine to produce an open-ended protocol that is extensible in both the type of request that can be specified, and the type of data that can be transmitted. Like HTML, HTTP will also therefore be the subject of evolutionary development.

Uniform Resource Locators (URL)

Uniform Resource Locators (URL)

- Specifies the protocol, host and object location.

- eg: `http://www.w3.org/Protocols/rfc1945/rfc1945`



- Can also set the port to which we connect.
 - `http://jeeves.javasoft.com:8080/`
- URL can be made up of a combination of these elements.
 - `ftp://ftp.sun.com/pub/download`
 - `http://www.sun.com`
 - `http://www.sun.com:80/index.html`

Uniform Resource Locators

Also invented by Tim Berners-Lee and defined in RFC 1738, URL's allow someone on the Internet not allow to find an object, but tells them how to retrieve the object.

In the case of a normal HTML page, we could possibly use the following URL:
`http://java.sun.com/`

You notice that we have specified which protocol to use, HTTP and the hostname were the object resides. We have omitted the port number and the document name. In cases likes these there are sensible defaults. Web Servers reside on port 80 by default so the web browser will try there. If no object name is specified then the web server will have a default document name, normally `index.html`.

We can use a URL to specify objects residing on servers anywhere on the Internet and we can use any protocol even ones we have created ourselves.

`rmi://rmiserver.java.com/classLoader`

URL-encoding

URL-encoding

- Browsers use an encoding when sending query strings that include special characters.
 - Most non-alphanumeric characters are encoded as a '%' followed by 2 ASCII encoded hex digits.
 - '=' (which is hex 3D) becomes "%3D"
 - '&' becomes "%26"

URL-encoding

URL-encoding is a simple method of encoding which converts spaces and other characters which may confuse the program reading the form data. Since '=' and '&' are used for special purposes in the form data, the program wouldn't know the difference between the user's '&' [or '='] and the special '&' [or '=']. Therefore, all '&'s, '='s, spaces and some other characters are converted by URL-Encoding, however the majority of characters are left alone.

Spaces are converted to "+"; most alphabetical and numerical characters are left alone. Other characters are converted to their ASCII value preceded by a '%' character [ie: "%nn" where 'nn' is the ASCII value of the character in hexadecimal]. For instance, since <space> is converted to '+', a '+' in the user's input would be converted to '%2B', since 2B is the hexadecimal equivalent decimal 43, the ASCII value of the character '+'.

For instance, the string "2 + 2 = 4" would be URL-Encoded to read:

2+%2B+2+%3D+4since '+' is <space>, %2B is '+' and %3D is '='.

The URL decoding process is relatively straightforward, but it is usually more useful to use a pre-written library [such as 'cgi-lib.pl'] to do the conversion and decoding.

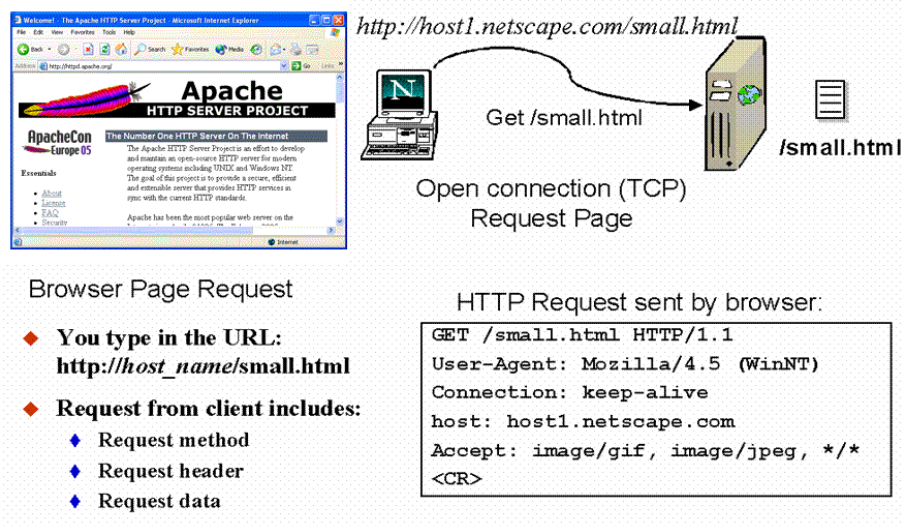
More URL encoding

More URL encoding

- The space character ` ` is replaced by `+`.
 - Why? (think about project 2 parsing...)
- The `+` character is replaced by "%2B".
 - Example:
"foo=6 + 7" becomes "foo%3D6+%2B+7"

Basic HTTP - Client Request

Basic HTTP - Client Request



part of [Hypertext Transfer Protocol -- HTTP/1.1](#)

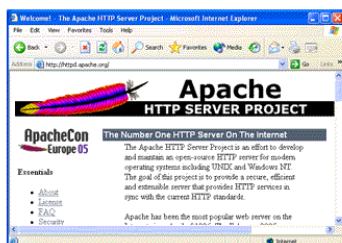
RFC 2616 Fielding, et al.

9.3 GET

The GET method means retrieve whatever information (in the form of an entity) is identified by the Request-URI. If the Request-URI refers to a data-producing process, it is the produced data which shall be returned as the entity in the response and not the source text of the process, unless that text happens to be the output of the process

Basic HTTP – Server Response

Basic HTTP – Server Response



- ◆ **Server evaluates the request**
- ◆ **Server response includes:**
 - ◆ **Status code**
 - ◆ **Response header**
 - ◆ **Response data**

```
HTTP/1.1 200 OK
Date: Tue, 28 Jun 2005 14:10:58 GMT
Server: Apache/2.0.53
Accept-Ranges: bytes
Cache-Control: max-age=2592000
Expires: Thu, 28 Jul 2005 14:10:58 GMT
Vary: Accept-Encoding
Content-Length: 4992
Connection: close
Content-Type: text/html; charset=UTF-8

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
```

Extracts from [Hypertext Transfer Protocol -- HTTP/1.1 RFC 2616](#) Fielding, et al.

6 Response

After receiving and interpreting a request message, a server responds with an HTTP response message.

Response = Status-Line ((general-header | response-header | entity-header) CRLF) CRLF [message-body]

6.1 Status-Line

The first line of a Response message is the Status-Line, consisting of the protocol version followed by a numeric status code and its associated textual phrase

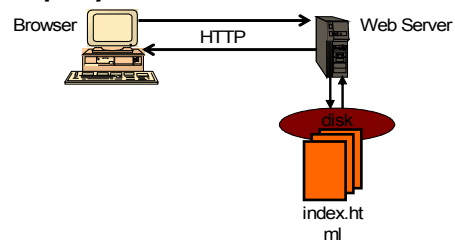
6.2 Response Header Fields

The response-header fields allow the server to pass additional information about the response which cannot be placed in the Status-Line. These header fields give information about the server and about further access to the resource identified by the Request-URI.

Beginning of Web Technologies

Beginning of Web Technologies

- Static pages stored on disk
- Client / Server request-response cycle using HTTP
- Web Server loads file from disk and sends over the network to be displayed in the browser



Limitations of Static Content

Limitations of Static Content



- Have to edit the static files to change the content
 - Impossible to personalize or easily customize content
 - Does not meet the requirements of delivering application services over the Web
 - Need a mechanism to generate dynamic content
-

Why generate dynamic content?

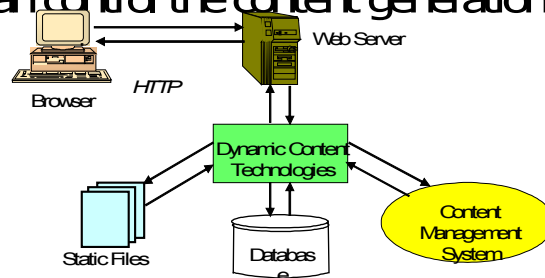
Why generate dynamic content?

- Static Content is not flexible enough for the needs of today's Web sites
- Dynamic Content provides a way to generate content as needed
 - User must be involved in the generation of content; personalization
- Many different ways of generating Dynamic Content
 - All have their own benefits and limitations

Dynamic Content Technologies

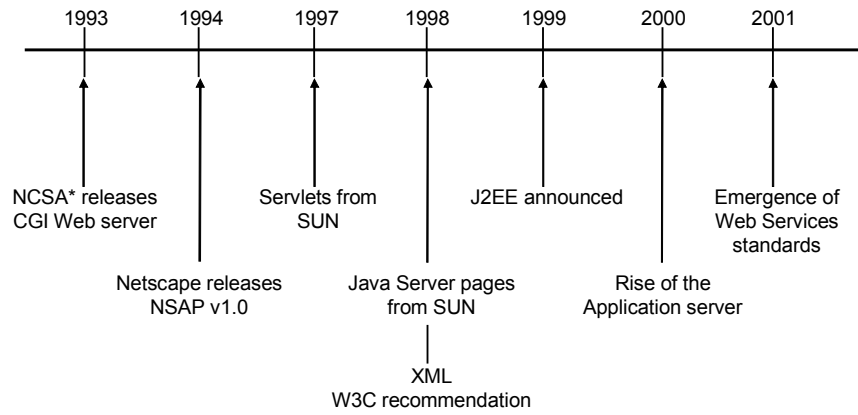
Dynamic Content Technologies

- Web Applications are often required to generate content which is a result of the ongoing conversation with the client
- Content generated during the life of the request
- The request can control the content generation process



Evolution of Dynamic Content

Evolution of Dynamic Content



*National Center for Super Computing Applications

HTTP is stateless

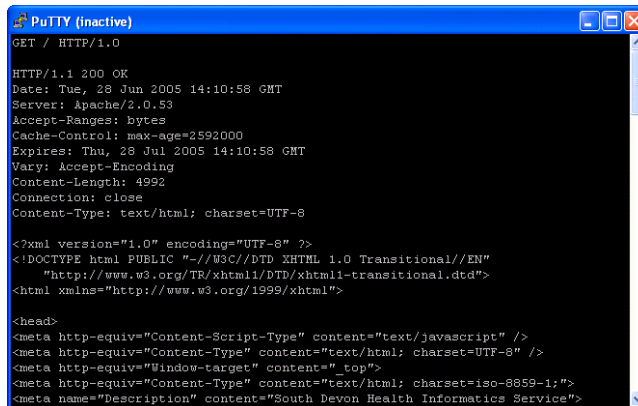
HTTP is stateless

- Problem: How do Web applications keep track of their clients if HTTP is stateless?
- Solution: Cookies
 - Unique token included in HTTP request
 - Identifies the client to the server
- Solution: URL rewriting
 - Unique token included in URL as a URL form parameter '?'
 - Identifies the client to the server

LAB

LAB

- Use Telnet to send a simple http request.



```
PUTTY (inactive)
GET / HTTP/1.0

HTTP/1.1 200 OK
Date: Tue, 28 Jun 2005 14:10:58 GMT
Server: Apache/2.0.53
Accept-Ranges: bytes
Cache-Control: max-age=2592000
Expires: Thu, 28 Jul 2005 14:10:58 GMT
Vary: Accept-Encoding
Content-Length: 4992
Connection: close
Content-Type: text/html; charset=UTF-8

<?xml version="1.0" encoding="UTF-8" ?>
<!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>
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<meta http-equiv="Window-target" content="top">
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1;">
<meta name="Description" content="South Devon Health Informatics Service">
```

Open a connection to localhost port 80 using a telnet client,
Enter the request “GET / HTTP/1.0”<CR><CR>

Controlling the Apache processes

Controlling the Apache processes



He who controls others may be
powerful, but he who has
mastered himself is mightier still

Lao Tzu

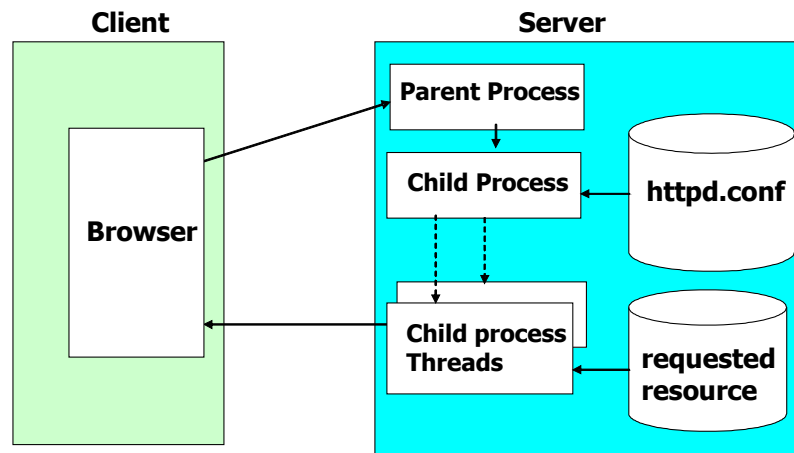
Controlling the Apache Processes

Controlling the Apache Processes

- Understand how the httpd deals with requests
 - Running Apache as a Service
 - Stop and Start Apache
 - Configure the httpd.conf file
 - Test the server
-

Overview of the Server Structure

Overview of the Server Structure



Student Notes

The controlling process in the web server system is the apache parent Process. This program listens to one or more TCP/IP ports for http requests from a client (usually through a web browser, although any socket based application can connect to the server).

When a connection is made, the request is passed to the child process, which generates a new thread for each request and returns the requested resource to the client.

httpd.conf is the configuration file for the apache, it is re-read each time a child process is started.

The Apache Processes

The Apache Processes

- Apache.exe starts Apache Processes
- Configured with httpd.conf
- By default a single child process handles all requests.
- Threads are created on demand

Because Apache for Windows is multithreaded, it does not use a separate process for each request, as Apache does on Unix. Instead there are usually only two Apache processes running: a parent process, and a child which handles the requests. Within the child process each request is handled by a separate thread. The process management directives are also different:

[MaxRequestsPerChild](#): Like the Unix directive, this controls how many requests a single child process will serve before exiting. However, unlike on Unix, a single process serves all the requests at once, not just one. If this is set, it is recommended that a very high number is used. The recommended default, *MaxRequestsPerChild 0*, causes the child process to never exit.

Warning: The server configuration file is reread when a new child process is started. If you have modified httpd.conf, the new child may not start or you may receive unexpected results.

[ThreadsPerChild](#): This directive is new. It tells the server how many threads it should use. This is the maximum number of connections the server can handle at once, so be sure to set this number high enough for your site if you get a lot of hits. The recommended default is *ThreadsPerChild 50*.

Running Apache as a Service

Running Apache as a Service

- can run as console application or Service
- Service recommended for NT, W2K
- Apache -k commands
 - Install
 - Uninstall
- Issues with windows 9x.

You can install Apache as a service automatically during the installation. If you chose to install for all users, the installation will create an Apache service for you. If you specify to install for yourself only, you can manually register Apache as a service after the installation. You have to be a member of the Administrators group for the service installation to succeed.

Apache comes with a utility called the Apache Service Monitor. With it you can see and manage the state of all installed Apache services on any machine on your network. To be able to manage an Apache service with the monitor, you have to first install the service (either automatically via the installation or manually).

You can install Apache as a Windows NT service as follows from the command prompt at the Apache bin subdirectory:

apache -k install

If you need to specify the name of the service you want to install, use the following command.

You have to do this if you have several different service installations of Apache on your computer.

apache -k install -n "MyServiceName"

If you need to have specifically named configuration files for different services, you must use this:

apache -k install -n "MyServiceName" -f "c:\files\my.conf"

If you use the first command without any special parameters except -k install, the service will be called Apache2 and the configuration will be assumed to be conf\httpd.conf.

Removing an Apache service is easy. Just use:

apache -k uninstall

The specific Apache service to be uninstalled can be specified by using:

apache -k uninstall -n "MyServiceName"

Starting and Stopping Apache

Starting and Stopping Apache

- Apache Service Monitor
- Control Panel Services / Net Start / Net Stop
- Apache -k start
- Apache -k stop or Apache -k shutdown
- Apache -k restart

- Apache -n "ServiceName" -t
- Test with http://localhost/

Normal starting, restarting and shutting down of an Apache service is usually done via the Apache Service Monitor, by using commands like NET START Apache2 and NET STOP Apache2 or via normal Windows service management. Before starting Apache as a service by any means, you should test the service's configuration file by using:

apache -n "MyServiceName" -t

You can control an Apache service by its command line switches, too. To start an installed Apache service you'll use this:

apache -k start

To stop an Apache service via the command line switches, use this:

apache -k stop

or

apache -k shutdown

You can also restart a running service and force it to reread its configuration file by using:

apache -k restart

Testing that Apache is running

Even if you don't have a network connection for your machine, you can always test your server by using the loop back address 127.0.0.1. On most systems, this is named localhost.

The loop back address allows you to send TCP/IP requests out through the Ethernet card and then back into the same machine. The computer treats loopback requests exactly as if they were coming from an external machine with the IP address of 127.0.0.1

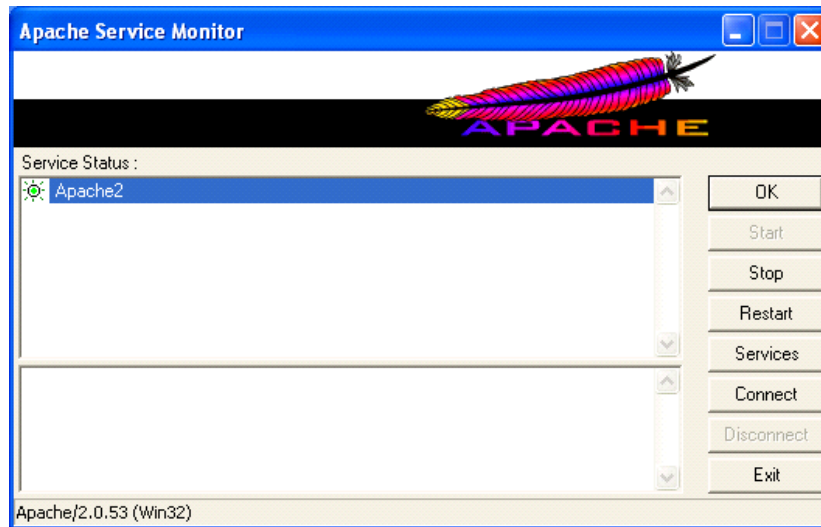
Open your browser and type

http://localhost/

This will instruct the browser to loop back on itself and try and access the default page.

Apache Service Monitor

Apache Service Monitor

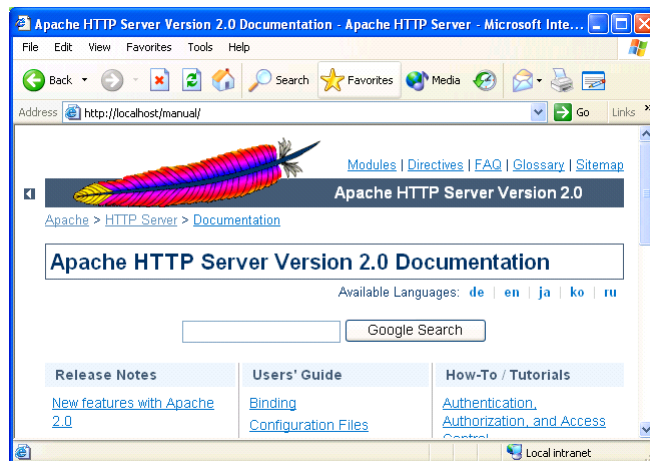


Apache comes with a utility called the Apache Service Monitor. With it you can see and manage the state of all installed Apache services on any machine on your network. To be able to manage an Apache service with the monitor, you have to first install the service (either automatically via the installation or manually).

Apachemonitor.exe

Apache Online Documentation

Apache Online Documentation



Apache Online Documentation

The Apache online documentation included with every distribution is very thorough, and although it is a little difficult to follow in places it is usually the best place to start looking when problems arise. Once you have the server running, the online documentation is accessible from the test start page. If the dummy start page is removed at any point, the documentation can be reached at <http://localhost/manual/>

The httpd.conf file

The httpd.conf file



httpd.conf

httpd.conf



- Structure and syntax of the httpd.conf file.
 - Activate and control the server status page.
 - Modify the run time linking of DSOs.
 - Change the server directory locations.
 - Set up virtual hosts
-

httpd.conf

- The main configuration file for the httpd server
- This provides information about:
 - The location of the server 'root' directory
 - The port on which the server listens for requests
 - The location of access control and resource configuration files
 - The location for log files and the type of logging to be performed

Student Notes

Don't Panic.

Apache is a highly configurable software system. Because the full range of configuration is made available using just one configuration file, the prospect of Apache configuration can be daunting at first. Fortunately, the configuration file is remarkably simple to use given its scope and with most administration tasks the most difficult part of the configuration task is locating the required keyword in the file.

Cautionary note:

As with any potentially risky administrative task, it is recommended that backups of httpd.conf be taken before any major changes are made. httpd.conf is a large file and even though changes that cause syntax errors will generate debugging information when you try to start the server, errors can be difficult to track down. Even worse, there is the prospect of accidentally making changes that allow the server to start smoothly, but leave gaping security holes in the system or prevent the site from working properly. In a production environment, stringent version control of httpd.conf is highly recommended (rcs is included in Red Hat by default, and would be adequate for such a task.)

This module will take a task-based approach to maintaining httpd.conf. There are many lists outlining the complete syntax of httpd.conf available for reference. However, there are fewer documents that connect individual changes together to form an introduction and guide for good practice.

A historical note about other configuration files.

Older versions of Apache were controlled by a set of three configuration files, Access.conf, httpd.conf and srm.conf. Since then the functionality of these three files has been integrated into the one httpd.conf file. The other two configuration files are still included with current distributions, and the server can be configured to read the old layout. There doesn't seem to be any reason to do this other than to upgrade a server whilst maintaining the existing configuration. The access.conf and srm.conf files supplied with the newest version of Apache are just stubs, they are ignored by the program.

Structure of httpd.conf

Structure of httpd.conf

- Global environment settings
- Main server settings
- Virtual server settings

Student Notes

Structure of httpd.conf

The httpd.conf file is made up of many small, (semi) independent configuration instructions, known as directives. Several lines of commentary precede each set of directives, briefly explaining the function of the directive that follows. Due to the extensive commentary the httpd.conf file is large, approaching 1000 lines, around 650 of those lines are commentary and commented out directives, 100 more are blank lines.

httpd.conf is split into 3 sections, global environment settings, 'main' server settings and virtual server settings.

The global environment section contains settings that will affect the whole server environment, such as which ports the daemon will listen to, how many child httpd processes will be started, etc.

The main server setting section describes a default server configuration. As we will see later on in this module, apache can pretend to be several web servers whilst only running one copy of httpd. This section has configurations for which port the main server will listen to, what user the server will run processes as, where the server documents are located, who has access permissions to which areas of the site, etc.

The virtual server section allows the specification of further 'virtual' servers. Virtual servers inherit the configuration of the main server unless they are explicitly overridden.

httpd.conf parameters

httpd.conf parameters

Parameter	Description	Default Value
ServerType	Started by inetd or standalone	standalone
Group	Groupid of children	<none>
TimeOut	Time to wait for client response	1200 seconds
PidFile	Name of file storing PID	logs/httpd/pid
TypesConfig	Name of mime types file	config/mime.types
ServerAdmin	Email Address of WebMaster	<none>
ErrorLog	Name of Error log file	conf/error.log

httpd.conf parameters

Parameter	Description	Default Value
AccessConf	Access configuration file	conf/access.conf
AccessConfig	Logging of Remote usernames	off
User	Server Userid	-1 <nobody>
ServerRoot	httpd 'root' directory	/usr/local/etc/httpd
ServerName	Hostname for the httpd	<none>
TransferLog	Client access log	logs/access.log

Student Notes

Structure of configuration directives.

Individual configuration directives come in a variety of formats.

The most simple is a toggled directive. If the directive is commented out with a hash, the setting is switched off, if the comment is removed it is switched on.

Example

```
#ExtendedStatus On
```

The extended status is switched off.

```
ExtendedStatus On
```

Now it is switched on.

Slightly more complicated are the directives requiring a value for them to make sense. Commenting these directives out could make the server unstable, or not run at all so it is important not to confuse them with the simple on/off directives.

Example

```
ServerRoot "/testapache"
```

The base directory for the server is set to /testapache.

Finally there is a directive format which allows the configuration of directory options. The structure of this kind of directive is as follows:

```
<Directory DIRECTORY_NAME>
```

```
    Option
```

```
    Option
```

```
    ...
```

```
    ...
```

```
</Directory>
```

With these directives, any number of options can be specified.

Example

```
<Directory />
```

```
    Options FollowSymLinks
```

```
    AllowOverride None
```

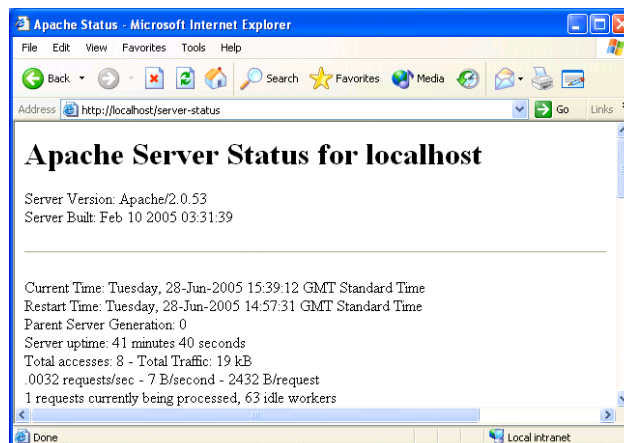
```
</Directory>
```

This sets the configuration for the root directory for the server. The permissions here are very restrictive - generally when specifying directory options the server will "fail-safe", i.e. if an option is not specified, it will default to the most secure. The first option given allows the server to access symbolic links, the second prevents any re-definition of the root directory permissions further on in the configuration file.

LAB

LAB

■ Configuring the **server-status** Screen



Student Notes

Example configuration exercise 1 - Configuring the server status screen

Apache has an extension module called `mod_status`, which allows the activation of a status page for the server, detailing the amount of up time, number of hits and other status information.

The directive that controls the server status option looks like this:

```
#<Location /server-status>  
#           SetHandler server-status  
#           Order deny,allow  
#           Allow from .your_domain.com  
#</Location>
```

Server status can be enabled by changing the directive to this:

```
<Location /server-status>  
           SetHandler server-status  
           Order deny,allow  
#           Allow from .your_domain.com  
           Allow from all  
</Location>
```

Uncommenting the lines activates the option. `Allow from .your_domain.com` is a dummy line that should be replaced by a valid domain. At present, there is no reason to restrict access to the page, and the "allow from all" option is very likely to get the directive working without complicating the issue with machine or network names.

Practical instructions

Make a backup copy of `httpd.conf`

Change the configuration file to allow the server status option.

Open Netscape and visit the URL "`http://localhost/server-status`" - what happens?

Stop the server and then start it again using `apache -k restart`. (See module 3)

Try the URL again - what happens?

Once the server status screen is working on your local machine try accessing the same

page on other machines in the classroom. ("`http://MACHINE_NAME/server-status`").

Change the "allow from all" line in the configuration file to "allow from localhost". Restart the server and try the local server status screen. Get your partner to try the status screen URL for your machine. What happens and why?

Try to make the status page accessible to your partner's machine by adding another allow from line to the server-status directive. Write down the option that was added.

Within `httpd.conf` there is an `ExtendedStatus` option which makes the status page more detailed. Switch it on and see what the extended page looks like.

The server status functionality is provided by the module `mod_status`. Comment out the `LoadModule` directives - what happens when the server is restarted?

Fix the configuration file so that the server starts without using `mod_status`. Write down the changes that you made.

Configuring the server-status Screen

Configuring the server-status Screen

- The server-status page can be used to verify that the server is running and to gain a measure of the server load.
- Directives activated/ de-activated with (#).
- Changes don't take effect until the server has been restarted.
- The "allow from" option controls access to a server feature.
- Many Apache features are contained within extension modules
- If the configuration file contains errors, a detailed error message is given when attempting to start the server.

Student Notes

Exercise summary

The server-status page can be used to verify that the server is running and to gain a measure of the server load.

Directives are activated by removing their comment and de-activated by commenting them out with a hash (#).

Changes don't take effect until the server has been restarted.

The "allow from" option controls access to a server feature. Several "allow from" statements can be added to a directive. Any machine that meets the criteria of any one "allow from" statement will be allowed access. "Allow from all" removes any access restrictions. "Allow from localhost" permits access from the local machine.

Many Apache features are contained within extension modules that are conditionally loaded at runtime. Removing the LoadModule and AddModule statements for a particular module will completely remove its functionality from the software. Active directives that relate to a disabled module may cause the httpd.conf file to be invalid.

If the configuration file contains errors, a detailed error message is given when attempting to start the server.

The search command ('/EXPRESSION') in vi is useful for finding directives

LAB

LAB

■ Modifying Server Locations

Student Notes

Example configuration exercise 2 - Configuring server locations

This exercise demonstrates configurations for the httpd.conf directives that define the location of documents. This type of server maintenance is essential for setting up test versions of a site and flipping between several test sites residing on the same machine.

Document Root

The document root directory is the internal directory on the file system that maps to / (root) for web access. For example, if a URL *http://localhost/* is requested and the document root is *c:\testapache\htdocs*, the server will look for an index page in *c:\testapache\htdocs*. Sub-directories in the URL map directly to sub-directories on the local file system, so in the example above, *http://localhost/manual* will map to *c:\testapache\htdocs\manual*.

Practical Instructions

Make a backup copy of http.conf.

Make a new directory on your system called \newroot. This will become the new default location for apache documents.

Copy the example index file (*****) into \newroot.

The DocumentRoot directive defines the default document directory. Change the directive to refer to the \newroot that was created. Write down the modified directive.

Restart the server and check that the new page is displayed by default.

It is possible to confuse the DocumentRoot setting, which sets the default directory for content, with the ServerRoot setting, which governs the location of the server's configuration and log files. Try changing the ServerRoot directive to \newroot and write down the consequences.

Restore DocumentRoot to its correct setting.

When a client requests a URL which ends in a directory name rather than a data file, the server searches for the default index page for that directory to display. The default name for an index page is index.html. The DirectoryIndex directive controls the name of the default index page. Edit httpd.conf and change the default index page to home.html.

Write down what happens when you try to access *http://localhost/*

Write down what happens when you try to access *http://localhost/index.html*

Modify the DirectoryIndex directive so that it contains entries for home.html and index.html.

DirectoryIndex home.html index.html

Try to view *http://localhost/* again. Write down what happens.

Use the sample home2.html and index2.html to determine whether apache searches from right to left or left to right when using the DirectoryIndex directive to search for an index page. Write down the answer.

If the server cannot find an index page for a directory it attempts to build its own index by listing the files in the directory. The indexing option must be explicitly enabled in the root Directory directive. Change the directive from:

```
<Directory />
```

```
Options FollowSymLinks  
AllowOverride None
```

```
</Directory>
```

to:

```
<Directory />
```

```
Options FollowSymLinks Indexes  
AllowOverride None
```

```
</Directory>
```

Rename the html files in the document root directory, so that no file will be recognized as a directory index. Try `http://localhost/` in the browser. Verify that a directory listing is displayed in the browser window. Make a subdirectory of `\newroot`. Place some files in the new directory. Can a directory listing be accessed for this new directory?
Restore the backup `httpd.conf` file.

Modifying Server Locations

Modifying Server Locations

- The base directory for server documents is defined with the DocumentRoot directive.
- The base directory for server executables is defined with the ServerRoot directive.
- Default index page file names are specified using the DirectoryIndex directive.
- If no index page is found for a page, the server attempts to generate a directory listing for the requested directory.
- Directory listings are forbidden by default.

Student Notes

The base directory for server documents is known as the document root and is defined with the DocumentRoot directive.

The base directory for server executables and configuration files is known as the server root and is defined with the ServerRoot directive.

When a client requests a directory name, the server looks for a default index page.

Default index page file names are specified using the DirectoryIndex directive.

Multiple file names are added using one DirectoryIndex directive, the file names are added as a space separated list.

If no index page is found for a page, the server attempts to generate a directory listing for the requested directory.

Directory listings are forbidden by default. Adding the Options Indexes attribute to a directory directive will enable listings. This is done on a directory by directory basis. Any sub-directory of an Index enabled directory will by default allow directory listings.

LAB

LAB

■ Configuring the Apache Network Port Settings

Note: lab assumes virtual host not configured

Example configuration exercise 3 - Configuring Network Ports

By default, Apache listens to port 80, the standard port for http requests.

Ports with a number less than 1024 are considered to be standard reserved ports.

Due to the restrictions on port allocation, a standard port is defined for http servers that cannot access port 80. when installing a second alternative server on a machine that is already using port 80, it is advisable to use port 8080.

Practical Instructions

The Listen directive in httpd.conf controls the port that Apache will listen to for http requests.

Change the port to 8080 and restart the server.

The server should now be running on port 8080 ignoring the default port of 80. Verify that this is the case by trying to access *http://localhost/*.

Accessing a non-standard port using a web browser is achieved by adding the port number to the URL at the end of the machine name. For example, to access *http://education.hp.com/* on port 8080 type:

http://education.hp.com:8080/

Check that your local server is running on port 8080. Check the server-status screen. Verify that the footer of the status screen shows that the server is now running on port 8080.

The Listen directive allows the specification of extra ports for the server to listen to.

Add a directive to the httpd.conf file to allow the server to listen to port 80 in addition to its default Port, which should currently be set to 8080. After restarting the server, verify that requests can now be handled on both port 80 and port 8080.

Display the server-status screen through port 80. What is the port number listed in the page footer?

Restore the original httpd.conf file.

Filesystems & Webpace

Filesystems & Webpace



For the wise man looks into
space and he knows there is no
limited dimensions.

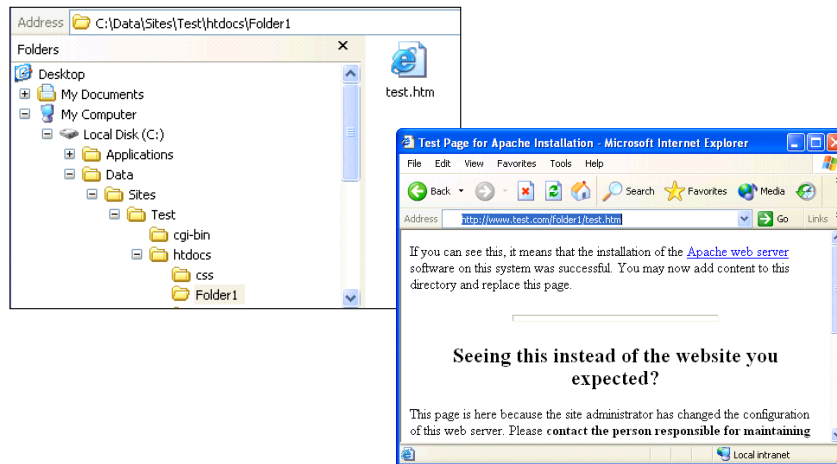
Lao Tzu

The most commonly used configuration section containers are the ones that change the configuration of particular places in the filesystem or webpace. First, it is important to understand the difference between the two. The filesystem is the view of your disks as seen by your operating system. For example, in a default install, Apache resides at `/usr/local/apache2` in the Unix filesystem or `"c:/Program Files/ Apache Group/ Apache2"` in the Windows filesystem. (Note that forward slashes should always be used as the path separator in Apache, even for Windows.) In contrast, the webpace is the view of your site as delivered by the web server and seen by the client. So the path `/dir/` in the webpace corresponds to the path `/usr/local/apache2/htdocs/dir/` in the filesystem of a default Apache install on Unix. The webpace need not map directly to the filesystem, since webpages may be generated dynamically from databases or other locations

DocumentRoot

DocumentRoot

DocumentRoot = /Data/Sites/Test/htdocs



In deciding what file to serve for a given request, Apache's default behavior is to take the URL-Path for the request (the part of the URL following the hostname and port) and add it to the end of the [DocumentRoot](#) specified in your configuration files. Therefore, the files and directories underneath the [DocumentRoot](#) make up the basic document tree which will be visible from the web.

Apache is also capable of [Virtual Hosting](#), where the server receives requests for more than one host. In this case, a different [DocumentRoot](#) can be specified for each virtual host, or alternatively, the directives provided by the module [mod_vhost_alias](#) can be used to dynamically determine the appropriate place from which to serve content based on the requested IP address or hostname.

Files Outside the DocumentRoot

Files Outside the DocumentRoot

- Symbolic links
- Alias
 - Alias /doc /data/web
- ScriptAlias
- AliasMatch and ScriptAliasMatch
 - ScriptAliasMatch `^/~([a-zA-Z0-9]+)/cgi-bin/(.+) /home/$1/cgi-bin/$2`

There are frequently circumstances where it is necessary to allow web access to parts of the filesystem that are not strictly underneath the [DocumentRoot](#). Apache offers several different ways to accomplish this. On Unix systems, symbolic links can bring other parts of the filesystem under the [DocumentRoot](#). For security reasons, Apache will follow symbolic links only if the [Options](#) setting for the relevant directory includes `FollowSymLinks` or `SymLinksIfOwnerMatch`.

Alternatively, the [Alias](#) directive will map any part of the filesystem into the web space. For example, with

```
Alias /docs /var/web
```

the URL `http://www.example.com/docs/dir/file.html` will be served from `/var/web/dir/file.html`. The [ScriptAlias](#) directive works the same way, with the additional effect that all content located at the target path is treated as CGI scripts. For situations where you require additional flexibility, you can use the [AliasMatch](#) and [ScriptAliasMatch](#) directives to do powerful regular-expression based matching and substitution. For example,

```
ScriptAliasMatch ^/~([a-zA-Z0-9]+)/cgi-bin/(.+) /home/$1/cgi-bin/$2
```

will map a request to `http://example.com/~user/cgi-bin/script.cgi` to the path `/home/user/cgi-bin/script.cgi` and will treat the resulting file as a CGI script.

File System Containers

File System Containers

- **<Directory> <DirectoryMatch>**
 - Applies directives to Directories/Folders
 - <Directory "C:/www/restricted">*
 - Order deny,allow*
 - Deny from all*
 - Allow from localhost*
 - </Directory>*

- **<Files> <FilesMatch>**
 - Applies directives to individual files

What are directives ?

What are directives ?

- Module Specific limits associated with
 - Directory
 - Server Configuration
 - Virtual host
 - .htaccess files

To find out what directives are allowed in what types of configuration sections, check the [Context](#) of the directive. Everything that is allowed in [<Directory>](#) sections is also syntactically allowed in [<DirectoryMatch>](#), [<Files>](#), [<FilesMatch>](#), [<Location>](#), [<LocationMatch>](#), [<Proxy>](#), and [<ProxyMatch>](#) sections. There are some exceptions, however:

The [AllowOverride](#) directive works only in [<Directory>](#) sections.

The FollowSymLinks and SymLinksIfOwnerMatch [Options](#) work only in [<Directory>](#) sections or .htaccess files.

The [Options](#) directive cannot be used in [<Files>](#) and [<FilesMatch>](#) sections

Webpace Containers

Webpace Containers

- **<Location> <LocationMatch>**
 - Applies Directives to the URL
 - need not have anything to do with the file system

```
<Location /server-info>
```

```
Order deny,allow
```

```
Deny from all
```

```
Allow from localhost
```

```
</Location>
```

The [<Location>](#) directive and its regex counterpart, on the other hand, change the configuration for content in the webpace

Wildcards and Regular Expressions

Wildcards and Regular Expressions

- * - any sequence of characters
- ? – single character
- [seq] – any characters in the sequence
 - [0-9]
- Note: "/" character NOT matched by wildcards
- Perl-compatible regular expressions with regex directives

Wildcards and Regular Expressions

The [<Directory>](#), [<Files>](#), and [<Location>](#) directives can each use shell-style wildcard characters as in `fnmatch` from the C standard library. The character "*" matches any sequence of characters, "?" matches any single character, and "[seq]" matches any character in `seq`. The "/" character will not be matched by any wildcard; it must be specified explicitly.

If even more flexible matching is required, each container has a regular-expression (regex) counterpart [<DirectoryMatch>](#), [<FilesMatch>](#), and [<LocationMatch>](#) that allow perl-compatible [regular expressions](#) to be used in choosing the matches. But see the section below on configuration merging to find out how using regex sections will change how directives are applied.

A non-regex wildcard section that changes the configuration of all user directories could look as follows:

```
<Directory /home/*/public_html>
```

```
Options Indexes
```

```
</Directory>
```

Using regex sections, we can deny access to many types of image files at once:

```
<FilesMatch \.(?i:gif|jpe?g|png)$>
```

```
Order allow,deny
```

```
Deny from all
```

```
</FilesMatch>
```

What to use when

What to use when

- **<Directory> <Files>**
 - To anything which eventually is on the filesystem
- **<Location>**
 - When applying to anything not on the filesystem
 - When applied to dynamic webpages
 - Never to anything on the filesystem

Choosing between filesystem containers and webspace containers is actually quite easy. When applying directives to objects that reside in the filesystem always use [<Directory>](#) or [<Files>](#). When applying directives to objects that do not reside in the filesystem (such as a webpage generated from a database), use [<Location>](#).

It is important to never use [<Location>](#) when trying to restrict access to objects in the filesystem. This is because many different webspace locations (URLs) could map to the same filesystem location, allowing your restrictions to be circumvented. For example, consider the following configuration:

```
<Location /dir/>  
Order allow,deny  
Deny from all  
</Location>
```

This works fine if the request is for `http://yoursite.example.com/dir/`. But what if you are on a case-insensitive filesystem? Then your restriction could be easily circumvented by requesting `http://yoursite.example.com/DIR/`. The [<Directory>](#) directive, in contrast, will apply to any content served from that location, regardless of how it is called. (An exception is filesystem links. The same directory can be placed in more than one part of the filesystem using symbolic links. The [<Directory>](#) directive will follow the symbolic link without resetting the pathname. Therefore, for the highest level of security, symbolic links should be disabled with the appropriate [Options](#) directive.)

If you are, perhaps, thinking that none of this applies to you because you use a case-sensitive filesystem, remember that there are many other ways to map multiple webspace locations to the same filesystem location. Therefore you should always use the filesystem containers when you can. There is, however, one exception to this rule. Putting configuration restrictions in a `<Location />` section is perfectly safe because this section will apply to all requests regardless of the specific URL.

How the sections are merged ?

How the sections are merged ?

1. <Directory> & .htaccess (if allowed)
2. <DirectoryMatch> & <Directory ~>
3. <FilesMatch> & <Files>
4. <LocationMatch> <Location>

The configuration sections are applied in a very particular order. Since this can have important effects on how configuration directives are interpreted, it is important to understand how this works.

The order of merging is:

[<Directory>](#) (except regular expressions) and .htaccess done simultaneously (with .htaccess, if allowed, overriding [<Directory>](#))

[<DirectoryMatch>](#) (and <Directory ~>)

[<Files>](#) and [<FilesMatch>](#) done simultaneously

[<Location>](#) and [<LocationMatch>](#) done simultaneously

Apart from [<Directory>](#), each group is processed in the order that they appear in the configuration files. [<Directory>](#) (group 1 above) is processed in the order shortest directory component to longest. So for example, <Directory /var/web/dir> will be processed before <Directory /var/web/dir/subdir>. If multiple [<Directory>](#) sections apply to the same directory they are processed in the configuration file order. Configurations included via the [Include](#) directive will be treated as if they were inside the including file at the location of the [Include](#) directive.

Sections inside [<VirtualHost>](#) sections are applied after the corresponding sections outside the virtual host definition. This allows virtual hosts to override the main server configuration.

Later sections override earlier ones.

Technical Note

There is actually a <Location>/<LocationMatch> sequence performed just before the name translation phase (where Aliases and DocumentRoots are used to map URLs to filenames). The results of this sequence are completely thrown away after the translation has completed.

A Merging Example

A Merging Example

```
<Location />
E
</Location>

<Files f.html>
D
</Files>

<VirtualHost *>
<Directory /a/b>
B
</Directory>
</VirtualHost>

<DirectoryMatch "^.*b$">
C
</DirectoryMatch>

<Directory /a/b>
A
</Directory>
```

Assuming they all apply to the request, the directives in this example will be applied in the order A > B > C > D > E.

LAB

LAB

- Create an Alias
- Use a File space container to
 - Allow directory browsing on the new alias
 - Restrict Access to Localhost

Create an Alias by adding the line

```
Alias /pics c:/data/sites/unknown/htdocs/images
```

To httpd.conf

Restart Apache, and use your browser to view the alias

```
http://localhost/pics
```

You will probably find access if forbidden, this is because there is not a suitable index file (such as index.html) in the location. So lets Enable Directory browsing at this location.

Create a filespace container for the directory and invoke the option to allow directory browsing;

```
<Directory "C:/data/sites/unknown/htdocs/images" >  
    Options Indexes  
</Directory>
```

You should now be able to see a list of files in the browser

We now want to restrict access to these files to localhost, so go back to the filespace container you created and add the following lines

```
Order deny,allow  
Deny from All  
Allow from localhost
```

Check that you can still access the files, now ask your neighbour to see if he/she can see the files

```
Http://{your ip address}/pics
```

Hint: you can find out your ip address by typing "ipconfig" at the command prompt.

Apache Modules

Apache Modules



Apache Modular Approach

Apache Modular Approach



- Apache core – basic, simple
 - Source Code available to all
 - Programmers write module for any function
 - Commercial and third party wrote modules
 - Most modules are free
-

Apache Module Types

Apache Module Types

- Core (httpd_core.c)
 - statically linked into kernel, cannot be removed
 - available in all Apache servers
- Standard
 - part of Apache distributions
 - maintained by Apache Software Foundation
 - can be removed for security or performance
- Third-party – not part of distributions

How Modules Work

How Modules Work



- Modules register callbacks with Apache
 - callbacks = function registered with Apache
 - Apache can call any function provided by modules at any time
 - callbacks = registered as handlers for specific function
 - Module functions part of HTTP request cycle
-

Important Apache Modules

Important Apache Modules

- **mod_perl** – perl.apache.org/guide/
 - before Perl, modules written in C
 - provides built-in handler for all 11 phases of Apache cycle
 - Perl interpreter is memory resident
- **mod_jk** - jakarta.apache.org/tomcat/
 - JSP on apache
 - Links to Tomcat Servlet engine

Third Party Installation Methods

Third Party Installation Methods

- Within Apache source tree – Supplied as part of the Apache distribution
- Outside – most third party installed as dynamic shared objects (DSOs)
 - use apxs to compile as "shared object" with .so extension
 - use with any version
 - upgrade w/o Apache recompile

Compare Two Methods

Compare Two Methods

■ Statically Linked

- APACI
- source inside tree
- Apache bigger
- Fastest loading
- Module always loaded
- Recommended for simple & fast Apache configuration

■ Linked as DSO

- apxs
- source outside tree
- Apache smaller
- Load time 20% more
- Loaded is specified in httpd.conf
- Recommended when server or modules change frequently

Dynamically Linked Modules

Dynamically Linked Modules



- Loaded at runtime
- No performance overhead
- All modules, even standards, can be DSOs
- mod_so must be loaded for DSOs to load
- First modules loaded, last processed

In Apache 1.x module loading required two directives: LoadModule & AddModule

Module Installation Steps

Module Installation Steps

- Download and extract to a working directory
 - Read instructions and contents of makefile
 - Compile & install module and configure Apache
 - Check if httpd.conf modified
 - Restart server & check server-info
 - Edit Apache configuration to use module
-

Another Installation Method

Another Installation Method



- Use the Makefile with Make utility
 - More convenient than previous example
 - Sometimes not available
 - Some settings may not be applicable to you
-

LAB

LAB



- Install mod_info
- Install the server-status module
- Show that you can report from them and view them
- Show access controls for modules

Open httpd.conf

Find the line
#LoadModule info_module modules/mod_info.so
Remove the #

Scroll down to find the server info section

```
#<Location /server-info>  
#  SetHandler server-info  
#  Order deny,allow  
#  Deny from all  
#  Allow from localhost  
#</Location>
```

Remove the #'s
Save your changes to httpd.info
Restart apache
Check the module if working by launching your browser and connection to <http://localhost/server-info>

Repeat the above process for the server status module mod_status.

Try viewing the server-info screen of your neighbours machine
<http://<their ip address>/server-info>

have them change the access controls on the server-info location so that you can view the page.

Virtual Hosting

Virtual Hosting



The term Virtual Host refers to the practice of running more than one web site (such as `www.company1.com` and `www.company2.com`) on a single machine. Virtual hosts can be "[IP-based](#)", meaning that you have a different IP address for every web site, or "[name-based](#)", meaning that you have multiple names running on each IP address. The fact that they are running on the same physical server is not apparent to the end user.

Apache was one of the first servers to support IP-based virtual hosts right out of the box. Versions 1.1 and later of Apache support both IP-based and name-based virtual hosts (vhosts). The latter variant of virtual hosts is sometimes also called host-based or non-IP virtual hosts.

Virtual hosting

Virtual hosting

- Running multiple Websites on single Server
- IP Based
 - Each host has an individual IP address
- Name based
 - Many hostnames using a single IP address
 - Relies on "hostname: " in HTTP conversation

IP-based virtual hosts use the IP address of the connection to determine the correct virtual host to serve. Therefore you need to have a separate IP address for each host. With name-based virtual hosting, the server relies on the client to report the hostname as part of the HTTP headers. Using this technique, many different hosts can share the same IP address.

Name-based virtual hosting is usually simpler, since you need only configure your DNS server to map each hostname to the correct IP address and then configure the Apache HTTP Server to recognize the different hostnames. Name-based virtual hosting also eases the demand for scarce IP addresses. Therefore you should use name-based virtual hosting unless there is a specific reason to choose IP-based virtual hosting. Some reasons why you might consider using IP-based virtual hosting:

Some ancient clients are not compatible with name-based virtual hosting. For name-based virtual hosting to work, the client must send the HTTP Host header. This is required by HTTP/1.1, and is implemented by all modern HTTP/1.0 browsers as an extension. If you need to support obsolete clients and still use name-based virtual hosting, a possible technique is discussed at the end of this document.

Name-based virtual hosting cannot be used with SSL secure servers because of the nature of the SSL protocol.

Some operating systems and network equipment implement bandwidth management techniques that cannot differentiate between hosts unless they are on separate IP addresses.

IP Based Virtual hosting

IP Based Virtual hosting

```
Listen 10.0.0.1:80
Listen 10.0.0.2:80
<VirtualHost 10.0.0.1>
    ServerAdmin webmaster@mail.smallco.com
    DocumentRoot /groups/smallco/www
    ServerName www.smallco.com
    ErrorLog /groups/smallco/logs/error_log
    TransferLog /groups/smallco/logs/access_log
</VirtualHost>
<VirtualHost 10.0.0.2>
    ServerAdmin webmaster@mail.baygroup.org
    DocumentRoot /groups/baygroup/www
    ServerName www.baygroup.org
    ErrorLog /groups/baygroup/logs/error_log
    TransferLog /groups/baygroup/logs/access_log
</VirtualHost>
```

As the term IP-based indicates, the server **must have a different IP address for each IP-based virtual host**. This can be achieved by the machine having several physical network connections, or by use of virtual interfaces which are supported by most modern operating systems (see system documentation for details, these are frequently called "IP aliases", and the "ifconfig" command is most commonly used to set them up).

There are two ways of configuring apache to support multiple hosts. Either by running a separate [httpd](#) daemon for each hostname, or by running a single daemon which supports all the virtual hosts.

Use multiple daemons when:

There are security partitioning issues, such as company1 does not want anyone at company2 to be able to read their data except via the web. In this case you would need two daemons, each running with different [User](#), [Group](#), [Listen](#), and [ServerRoot](#) settings.

You can afford the memory and [file descriptor requirements](#) of listening to every IP alias on the machine. It's only possible to [Listen](#) to the "wildcard" address, or to specific addresses. So if you have a need to listen to a specific address for whatever reason, then you will need to listen to all specific addresses. (Although one [httpd](#) could listen to N-1 of the addresses, and another could listen to the remaining address.)

Use a single daemon when:

Sharing of the httpd configuration between virtual hosts is acceptable.

The machine services a large number of requests, and so the performance loss in running separate daemons may be significant.

```
<VirtualHost www.abc.dom>
ServerAdmin webgirl@abc.dom
DocumentRoot /www/abc
</VirtualHost>
```

In order for Apache to function properly, it absolutely needs to have two pieces of information about each virtual host: the [ServerName](#) and at least one IP address that the server will bind and respond to. The above example does not include the IP address, so Apache must use DNS to find the address of www.abc.dom. If for some reason DNS is not available at the time your server is parsing its config file, then this virtual host **will not be configured**. It won't be able to respond to any hits to this virtual host (prior to Apache version 1.2 the server would not even boot).

Suppose that www.abc.dom has address 10.0.0.1. Then consider this configuration snippet:

```
<VirtualHost 10.0.0.1>
ServerAdmin webgirl@abc.dom
DocumentRoot /www/abc
</VirtualHost>
```

This time Apache needs to use reverse DNS to find the ServerName for this virtualhost. If that reverse lookup fails then it will partially disable the virtualhost (prior to Apache version 1.2 the server would not even boot). If the virtual host is name-based then it will effectively be totally disabled, but if it is IP-based then it will mostly work. However, if Apache should ever have to generate a full URL for the server which includes the server name, then it will fail to generate a valid URL.

Here is a snippet that avoids both of these problems:

```
<VirtualHost 10.0.0.1>
ServerName www.abc.dom
ServerAdmin webgirl@abc.dom
```

```
DocumentRoot /www/abc  
</VirtualHost>
```

use IP addresses in [VirtualHost](#)

use IP addresses in [Listen](#)

ensure all virtual hosts have an explicit [ServerName](#)

create a `<VirtualHost _default_*>` server that has no pages to serve

Name based Virtual Hosting

Name based Virtual Hosting

```
NameVirtualHost *:80
<VirtualHost *:80>
    ServerName www.domain.tld
    ServerAlias domain.tld *.domain.tld
    DocumentRoot /www/domain
</VirtualHost>
<VirtualHost *:80>
    ServerName www.otherdomain.tld
    DocumentRoot /www/otherdomain
</VirtualHost>
```

To use name-based virtual hosting, you must designate the IP address (and possibly port) on the server that will be accepting requests for the hosts. This is configured using the [NameVirtualHost](#) directive. In the normal case where any and all IP addresses on the server should be used, you can use * as the argument to [NameVirtualHost](#). If you're planning to use multiple ports (e.g. running SSL) you should add a Port to the argument, such as *:80. Note that mentioning an IP address in a [NameVirtualHost](#) directive does not automatically make the server listen to that IP address. See [Setting which addresses and ports Apache uses](#) for more details. In addition, any IP address specified here must be associated with a network interface on the server.

The next step is to create a [<VirtualHost>](#) block for each different host that you would like to serve. The argument to the [<VirtualHost>](#) directive should be the same as the argument to the [NameVirtualHost](#) directive (ie, an IP address, or * for all addresses). Inside each [<VirtualHost>](#) block, you will need at minimum a [ServerName](#) directive to designate which host is served and a [DocumentRoot](#) directive to show where in the filesystem the content for that host lives.

Main host goes away

If you are adding virtual hosts to an existing web server, you must also create a [<VirtualHost>](#) block for the existing host. The [ServerName](#) and [DocumentRoot](#) included in this virtual host should be the same as the global [ServerName](#) and [DocumentRoot](#). List this virtual host first in the configuration file so that it will act as the default host.

ServerAlias

ServerAlias

```
<VirtualHost *>  
  ServerName server.domain.com  
  ServerAlias server server2.domain.com server2  
  # ...  
</VirtualHost>
```

- Sets alternate names for a host, for use with name-based virtual hosts
- Allows server to be accessed by more than one name

Many servers want to be accessible by more than one name. This is possible using the [ServerAlias](#) directive, placed inside the [<VirtualHost>](#) section.

For example, in the [<VirtualHost>](#) block above, the [ServerAlias](#) directive indicates that the listed names are other names which people can use to see that same web site. The wildcard characters * and ? can be used to match names. Of course, you can't just make up names and place them in [ServerName](#) or [ServerAlias](#). You must first have your DNS server properly configured to map those names to an IP address associated with your server

Dynamically configuring paths

Dynamically configuring paths

```
VirtualDocumentRoot /usr/local/apache/vhosts/%0  
VirtualScriptAlias /Scripts/%0/cgi-bin/
```

- Allows determination of document root and script locations based on the value of the server name
- Companion directives allow interpolation based on IP address

The `VirtualDocumentRoot` directive allows you to determine where Apache will find your documents based on the value of the server name. The result of expanding interpolated-directory is used as the root of the document tree in a similar manner to the [DocumentRoot](#) directive's argument. If interpolated-directory is none then `VirtualDocumentRoot` is turned off.

This directive cannot be used in the same context as `VirtualDocumentRootIP`.

The `VirtualScriptAlias` directive allows you to determine where Apache will find CGI scripts in a similar manner to `VirtualDocumentRoot` does for other documents.

It matches requests for URIs starting `/cgi-bin/`, much like `ScriptAlias /cgi-bin/` would.

Gotchas

Gotchas

- Client connected Server IP address
 - No match
 - the `_default_vhost`
 - if no vhost `main_server`
 - Match then
 - vhost which matches IP
 - Host:
compared matches `ServerName` or `ServerAlias` found within vhost. First Match used.

set in the main server context (outside any `<VirtualHost>` container) will be used only if they are not overridden by the virtual host settings.

Now when a request arrives, the server will first check if it is using an IP address that matches the `NameVirtualHost`. If it is, then it will look at each `<VirtualHost>` section with a matching IP address and try to find one where the `ServerName` or `ServerAlias` matches the requested hostname. If it finds one, then it uses the configuration for that server. If no matching virtual host is found, then **the first listed virtual host** that matches the IP address will be used.

As a consequence, the first listed virtual host is the default virtual host. The `DocumentRoot` from the main server will **never** be used when an IP address matches the `NameVirtualHost` directive. If you would like to have a special configuration for requests that do not match any particular virtual host, simply put that configuration in a `<VirtualHost>` container and list it first in the configuration file.

Statically Configured Mass vhosting

Statically Configured Mass vhosting

```
NameVirtualHost 111.22.33.44
<VirtualHost 111.22.33.44>
ServerName www.customer-1.com
DocumentRoot /www/hosts/www.customer-1.com/docs
ScriptAlias /cgi-bin/ /www/hosts/www.customer-1.com/cgi-bin
</VirtualHost>
<VirtualHost 111.22.33.44>
ServerName www.customer-2.com
DocumentRoot /www/hosts/www.customer-2.com/docs
ScriptAlias /cgi-bin/ /www/hosts/www.customer-2.com/cgi-bin
</VirtualHost>
# ...
<VirtualHost 111.22.33.44>
ServerName www.customer-N.com
DocumentRoot /www/hosts/www.customer-N.com/docs
ScriptAlias /cgi-bin/ /www/hosts/www.customer-N.com/cgi-bin
</VirtualHost>
```

Disadvantages:

- Large configuration file
- Slow to start up
- Uses more memory

Simple Dynamic Virtual Hosting

Simple Dynamic Virtual Hosting

■ Use `mod_vhost_alias`

■ Sample config extract:

```
# get the server name from the Host: header
UseCanonicalName Off

# this log format can be split per-virtual-host
based on the first field
LogFormat "%V %h %l %u %t \"%r\" %s %b" vcommon
CustomLog logs/access_log vcommon

# include the server name in the filenames used to
satisfy requests
VirtualDocumentRoot /www/hosts/%0/docs
VirtualScriptAlias /www/hosts/%0/cgi-bin
```

The basic idea is to replace all of the static `<VirtualHost>` configuration with a mechanism that works it out dynamically. This has a number of advantages:

- Your configuration file is smaller so Apache starts faster and uses less memory.
- Adding virtual hosts is simply a matter of creating the appropriate directories in the filesystem and entries in the DNS - you don't need to reconfigure or restart Apache.

The main disadvantage is that you cannot have a different log file for each virtual host; however if you have very many virtual hosts then doing this is dubious anyway because it eats file descriptors. It is better to log to a pipe or a fifo and arrange for the process at the other end to distribute the logs to the customers (it can also accumulate statistics, etc.).

This configuration can be changed into an IP-based virtual hosting solution by just turning `UseCanonicalName Off` into `UseCanonicalName DNS`. The server name that is inserted into the filename is then derived from the IP address of the virtual host.

More efficient IP-based virtual hosting

More efficient IP-based virtual hosting

```
# get the server name from the reverse DNS of the IP address
UseCanonicalName DNS

# include the IP address in the logs so they may be split
LogFormat "%A %h %l %u %t \"%r\" %s %b" vcommon
CustomLog logs/access_log vcommon

# include the IP address in the filenames
VirtualDocumentRootIP /www/hosts/%0/docs
VirtualScriptAliasIP /www/hosts/%0/cgi-bin
```

- Filesystem laid out by IP address
- Logging tagged by IP address
- DNS lookup not required for every request

Unfortunately when the previous configuration is used for IP based hosting it is not very efficient because it requires a DNS lookup for every request. This can be avoided by laying out the filesystem according to the IP addresses themselves rather than the corresponding names and changing the logging similarly. Apache will then usually not need to work out the server name and so incur a DNS lookup.

Separate virtual hosting configs

Separate virtual hosting configs

■ Sample vhost.map

```
www.customer-1.com /www/customers/1
www.customer-2.com /www/customers/2
# ...
www.customer-N.com /www/customers/N
```

■ Sample http.conf

```
RewriteEngine on
RewriteMap lowercase int:tolower
# define the map file
RewriteMap vhost txt:/www/conf/vhost.map
# deal with aliases as above
RewriteCond %{REQUEST_URI} !^/icons/
RewriteCond %{REQUEST_URI} !^/cgi-bin/
RewriteCond ${lowercase:%{SERVER_NAME}} ^(.+)$
# this does the file-based remap
RewriteCond ${vhost:%1} ^(/.*)$
RewriteRule ^(/.*)$ %1/docs/$1
RewriteCond %{REQUEST_URI} ^/cgi-bin/
RewriteCond ${lowercase:%{SERVER_NAME}} ^(.+)$
RewriteCond ${vhost:%1} ^(/.*)$
RewriteRule ^(/.*)$ %1/cgi-bin/$1
```

This arrangement uses advanced `mod_rewrite` features to get the translation from virtual host to document root from a separate configuration file. This provides more flexibility but requires more complicated configuration.

Monitoring

Monitoring



To see what is in front of one's
nose needs a constant struggle.

George Orwell

Apache Logging

Apache Logging



- To track:
 - Unusual conditions
 - Problems
 - Client Usage
- Two types of logs
 - Request or transfer logs: access log
 - Error log
- Core module: `mod_log_config`

In order to effectively manage a web server, it is necessary to get feedback about the activity and performance of the server as well as any problems that may be occurring. The Apache HTTP Server provides very comprehensive and flexible logging capabilities.

Error Logging

Error Logging

- Malformed HTTP requests account for the majority of errors
- Less frequent but require administrator attention:
 - loading modules
 - implementing configuration directives
 - setting up virtual hosts
- Location and name configured by `ErrorLog` directive

The server error log, whose name and location is set by the [ErrorLog](#) directive, is the most important log file. This is the place where Apache httpd will send diagnostic information and record any errors that it encounters in processing requests.

It is the first place to look when a problem occurs with starting the server or with the operation of the server, since it will often contain details of what went wrong and how to fix it.

A very wide variety of different messages can appear in the error log. Most look similar to the example above. The error log will also contain debugging output from CGI scripts. Any information written to stderr by a CGI script will be copied directly to the error log.

Error Log Settings – cont.

Error Log Settings – cont.

- `LogLevel` directive controls verbosity
 - Error Logging Levels:
 - emerg, alert, crit, error, warn, notice, info, debug

`LogLevel error`

- Example error log message

```
[Wed Oct 11 14:32:52 2000] [error] [client 127.0.0.1] client
denied by server configuration:
/export/home/live/ap/htdocs/test
```

The [LogLevel](#) directive is used to control the types of errors that are sent to the error log by restricting the severity level.

The format of the error log is relatively free-form and descriptive. But there is certain information that is contained in most error log entries.

- The first item in the log entry is the date and time of the message.
- The second entry lists the severity of the error being reported.
- The third entry gives the IP address of the client that generated the error.

Beyond that is the message itself, which in this case indicates that the server has been configured to deny the client access. The server reports the file-system path (as opposed to the web path) of the requested document.

It is not possible to customize the error log by adding or removing information. However, error log entries dealing with particular requests have corresponding entries in the access log. For instance, the above example entry corresponds to an access log entry with status code 403. Since it is possible to customize the access log, you can obtain more information about error conditions using that log file.

Request Logging

Request Logging

- Provided by `mod_log_config`
- CustomLog directive sets filename and format
 - If not specified, Common Log Format is used
- LogFormat directive can be used to specify different logging formats

The server access log records all requests processed by the server. The location and content of the access log are controlled by the [CustomLog](#) directive. The [LogFormat](#) directive can be used to simplify the selection of the contents of the logs.

Common Log Format (CLF)

Common Log Format (CLF)

```
LogFormat "%h %l %u %t \"%r\" %>s %b" common
CustomLog logs/access_log common
```

- The above configuration will write log entries in a format known as the Common Log Format (CLF)
- This standard format can be produced by many different web servers and read by many log analysis programs
- The log file entries produced in CLF will look something like this:

```
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0" 200 2326
```

This defines the *nickname* common and associates it with a particular log format string. The format string consists of percent directives, each of which tell the server to log a particular piece of information. Literal characters may also be placed in the format string and will be copied directly into the log output. The quote character (") must be escaped by placing a back-slash before it to prevent it from being interpreted as the end of the format string. The format string may also contain the special control characters "\n" for new-line and "\t" for tab.

The [CustomLog](#) directive sets up a new log file using the defined *nickname*. The filename for the access log is relative to the [ServerRoot](#) unless it begins with a slash.

Each part of this log entry is described below.

127.0.0.1 (%h)

This is the IP address of the client (remote host) which made the request to the server. If [HostnameLookups](#) is set to On, then the server will try to determine the hostname and log it in place of the IP address. However, this configuration is not recommended since it can significantly slow the server. Instead, it is best to use a log post-processor such as [logresolve](#) to determine the hostnames. The IP address reported here is not necessarily the address of the machine at which the user is sitting. If a proxy server exists between the user and the server, this address will be the address of the proxy, rather than the originating machine.

- (%l)

The "hyphen" in the output indicates that the requested piece of information is not available. In this case, the information that is not available is the RFC 1413 identity of the client determined by identd on the client's machine. This information is highly unreliable and should almost never be used except on tightly controlled internal networks. Apache httpd will not even attempt to determine this information unless [IdentityCheck](#) is set to On.

frank (%u)

This is the userid of the person requesting the document as determined by HTTP authentication. The same value is typically provided to CGI scripts in the REMOTE_USER environment variable. If the status code for the request (see below) is 401, then this value should not be trusted because the user is not yet authenticated. If the document is not password protected, this entry will be "-" just like the previous one.

[10/Oct/2000:13:55:36 -0700] (%t)

The time that the server finished processing the request. The format is: [day/month/year:hour:minute:second zone]

day = 2*digit

month = 3*letter

year = 4*digit

hour = 2*digit

minute = 2*digit

second = 2*digit

zone = ('+' | '-') 4*digit

It is possible to have the time displayed in another format by specifying `%{format}t` in the log format string, where format is as in `strftime(3)` from the C standard library.

"GET /apache_pb.gif HTTP/1.0" ("%r")

The request line from the client is given in double quotes. The request line contains a great deal of useful information. First, the method used by the client is GET. Second, the client requested the resource /apache_pb.gif, and third, the client used the protocol HTTP/1.0. It is also possible to log one or more parts of the request line independently. For example, the

format string "%m %U%q %H" will log the method, path, query-string, and protocol, resulting in exactly the same output as "%r".

200 (%>s)

This is the status code that the server sends back to the client. This information is very valuable, because it reveals whether the request resulted in a successful response (codes beginning in 2), a redirection (codes beginning in 3), an error caused by the client (codes beginning in 4), or an error in the server (codes beginning in 5). The full list of possible status codes can be found in the [HTTP specification](#) (RFC2616 section 10).

2326 (%b)

The last entry indicates the size of the object returned to the client, not including the response headers. If no content was returned to the client, this value will be "-". To log "0" for no content, use %B instead.

Conditional Logging

Conditional Logging

- CustomLog can use conditional logging
- Use SetEnvIf directive

```
# Mark requests from the loop-back interface
SetEnvIf Remote_Addr "127\.0\.0\.1" dontlog
# Mark requests for the robots.txt file
SetEnvIf Request_URI "^/robots\.txt$" dontlog
# Log what remains
CustomLog logs/access_log common env=!dontlog
```

There are times when it is convenient to exclude certain entries from the access logs based on characteristics of the client request. This is easily accomplished with the help of [environment variables](#). First, an environment variable must be set to indicate that the request meets certain conditions. This is usually accomplished with [SetEnvIf](#). Then the env= clause of the [CustomLog](#) directive is used to include or exclude requests where the environment variable is set.

Other Logs

Other Logs



- PID file
- Script Log
- Rewrite Log
- Windows Event Logs

PID File

On startup, Apache httpd saves the process id of the parent httpd process to the file logs/httpd.pid. This filename can be changed with the [PidFile](#) directive. The process-id is for use by the administrator in restarting and terminating the daemon by sending signals to the parent process; on Windows, use the -k command line option instead. For more information see the [Stopping and Restarting](#) page.

Script Log

In order to aid in debugging, the [ScriptLog](#) directive allows you to record the input to and output from CGI scripts. This should only be used in testing - not for live servers. More information is available in the [mod_cgi](#) documentation.

Rewrite Log

When using the powerful and complex features of [mod_rewrite](#), it is almost always necessary to use the [RewriteLog](#) to help in debugging. This log file produces a detailed analysis of how the rewriting engine transforms requests. The level of detail is controlled by the [RewriteLogLevel](#) directive.

Tracking User Sessions

Tracking User Sessions

- Reveals behavior of typical user
- Combine with user provided demographics – determines buying habits
- Users from business are behind firewalls and does not provide unique info
- Use `mod_usertrack`

mod_usertrack

Previous releases of Apache have included a module which generates a 'clickstream' log of user activity on a site using cookies. This was called the "cookies" module, `mod_cookies`. In Apache 1.2 and later this module has been renamed the "user tracking" module, `mod_usertrack`. This module has been simplified and new directives added.

Cookies

Cookies

- Web server generated and stored by client browser
- Browser sends cookie with subsequent requests
- Server sends cookie with HTTP response

```
Set-Cookie: name=value; expires=date; path=path; domain=domain_name; secure
```

- Enabled by Apache module: `mod_usertrack`
- Logging performed by CustomLog

Previously, the cookies module (now the user tracking module) did its own logging, using the `CookieLog` directive. In this release, this module does no logging at all. Instead, a configurable log format file should be used to log user click-streams. This is possible because the logging module now allows multiple log files. The cookie itself is logged by using the text `%{cookie}n` in the log file format. For example:

```
CustomLog logs/clickstream "%{cookie}n %r %t"
```

Cookies (cont.)

Cookies (cont.)

- To set: `CookieTracking on`
- Enables user tracking in virtual hosts, directory contexts and `.htaccess` files but not in `<Files>` containers

When the user track module is compiled in, and "CookieTracking on" is set, Apache will start sending a user-tracking cookie for all new requests. This directive can be used to turn this behavior on or off on a per-server or per-directory basis.

By default, compiling `mod_usertrack` will not activate cookies.

Session Tracking

Session Tracking

- Cookies not accepted by many clients
- HTTP = connectionless protocol – no persistent connection
 - connection lost after server replies
- Must maintain state information for some web applications:
 - e.g. shopping cart
- Usually requires some external coding

Analyzing Logs

Analyzing Logs



- Logs fill up quickly if significant traffic
- Create scripts
- Better to use ready made analyzers:
 - Analog – www.analog.cx
 - Webalizer – www.mrunix.net/webalizer/
 - WebTrends – www.webtrends.com/

Rotating Logs

Rotating Logs



- Log files can be large and grow quickly
- Write scripts to “rotate” logs
 - Move or delete old file and start a new one
- Use scheduler to run automatically

On even a moderately busy server, the quantity of information stored in the log files is very large. The access log file typically grows 1 MB or more per 10,000 requests. It will consequently be necessary to periodically rotate the log files by moving or deleting the existing logs. This cannot be done while the server is running, because Apache will continue writing to the old log file as long as it holds the file open. Instead, the server must be [restarted](#) after the log files are moved or deleted so that it will open new log files.

By using a *graceful* restart, the server can be instructed to open new log files without losing any existing or pending connections from clients. However, in order to accomplish this, the server must continue to write to the old log files while it finishes serving old requests. It is therefore necessary to wait for some time after the restart before doing any processing on the log files.

mod_proxy

mod_proxy



What is a proxy ?

What is a proxy ?

- Forward proxy
 - Intermediate between client and outside
 - e.g.
 - forward proxy can use caching (as provided by `mod_cache`) to reduce network usage .
- Reverse proxy
 - Disguises where connections are going
 - e.g.
 - Reverse proxies can also be used to balance load among several back-end servers

An ordinary **forward proxy** is an intermediate server that sits between the client and the origin server. In order to get content from the origin server, the client sends a request to the proxy naming the origin server as the target and the proxy then requests the content from the origin server and returns it to the client. The client must be specially configured to use the forward proxy to access other sites.

A typical usage of a forward proxy is to provide Internet access to internal clients that are otherwise restricted by a firewall. The forward proxy can also use caching (as provided by [mod_cache](#)) to reduce network usage.

The forward proxy is activated using the `ProxyRequests` directive. Because forward proxies allow clients to access arbitrary sites through your server and to hide their true origin, it is essential that you secure your server so that only authorized clients can access the proxy before activating a forward proxy

A **reverse proxy**, by contrast, appears to the client just like an ordinary web server. No special configuration on the client is necessary. The client makes ordinary requests for content in the name-space of the reverse proxy. The reverse proxy then decides where to send those requests, and returns the content as if it was itself the origin.

A typical usage of a reverse proxy is to provide Internet users access to a server that is behind a firewall. Reverse proxies can also be used to balance load among several back-end servers, or to provide caching for a slower back-end server. In addition, reverse proxies can be used simply to bring several servers into the same URL space.

A reverse proxy is activated using the `ProxyPass` directive or the [P] flag to the [RewriteRule](#) directive. It is **not** necessary to turn `ProxyRequests` on in order to configure a reverse proxy.

Example proxy configuration

Example proxy configuration

■ Forward Proxy

```
ProxyRequests On
ProxyVia On
<Proxy *>
Order deny,allow
Deny from all
Allow from internal.example.com
</Proxy>
```

■ Reverse Proxy

```
ProxyRequests Off
<Proxy *>
Order deny,allow
Allow from all
</Proxy>
ProxyPass /foo http://foo.example.com/bar
ProxyPassReverse /foo http://foo.example.com/bar
```

Controlling access to your proxy

Controlling access to your proxy

```
<Proxy *>
  Order Deny,Allow
  Deny from all
  Allow from 192.168.0
</Proxy>
```

- Open proxy servers can be dangerous
 - Any client could access arbitrary hosts while hiding his or her true identity
- Access control directives are the same as for `mod_access`

For more information on access control directives, see [mod_access](#).

Strictly limiting access is essential if you are using a forward proxy (using the `ProxyRequests` directive). Otherwise, your server can be used by any client to access arbitrary hosts while hiding his or her true identity. This is dangerous both for your network and for the Internet at large. When using a reverse proxy (using the `ProxyPass` directive with `ProxyRequests Off`), access control is less critical because clients can only contact the hosts that you have specifically configured.

<Proxy> Directive

<Proxy> Directive

- Directives placed in <Proxy> sections apply only to matching proxied content.
 - Shell-style wildcards are allowed.
- The following example will
 - process all files in the `foo` directory of `example.com` through the `INCLUDES` filter when they are sent through the proxy server:

```
<Proxy http://example.com/foo/*>  
SetOutputFilter INCLUDES  
</Proxy>
```

mod_rewrite

mod_rewrite



"The great thing about mod_rewrite is it gives you all the configurability and flexibility of Sendmail."

The downside to mod_rewrite is that it gives you all the configurability and flexibility of Sendmail."

Brian Behlendorf
Apache Group

Voodoo for URL Manipulations

Voodoo for URL Manipulations

- Gives you the flexibility to manipulate URLs based on:
 - Server Variables
 - Environmental Variable
 - HTTP headers
 - Time stamps
 - Regular Expressions
 - etc., etc., etc.

"Despite the tons of examples and docs, `mod_rewrite` is voodoo. Damned cool voodoo, but still voodoo."
Brian Moore
bem@news.cmc.net

One can use Apache's **mod_rewrite** to solve typical URL based problems webmasters are usually confronted with in practice. The Apache module `mod_rewrite` is a module which provides a powerful way to do URL manipulations. With it you can nearly do all types of URL manipulations you ever dreamed about. The price you have to pay is to accept complexity, because **mod_rewrite** is not easy to understand and use for the beginner.

From a [search engine optimization](#) point of view, `mod_rewrite` is a great tool to help your sites get more pages indexed in more search engines. For example, although [search engine spiders are getting better](#) at it, some still have a hard time with dynamic pages.

Take this filename:

```
file.php?id=12&section=23&template=45&page=3
```

Forgetting the notion that it's not good to use the variable `id` as some spiders think it's a session `id`, many search engines have a hard time crawling these types of pages. As I said, they're getting better at it, but what if you could help them crawl your site in the meantime?

Enter `mod_rewrite`. With the module you can rewrite the above URL into something like:
`/id/12/section-23/template-45/page-3.html`

Or, even better:

```
/keyword/section-name/file.html
```

You can see where I'm going with this, I hope, and why it might [help your pages rank](#) in the [search engines](#).

The module is also useful for a variety of other tasks; managing a lot of virtual hosts, dealing with the trailing slash problem on URLs, and even stopping people from using images from your server on their site (running up your bandwidth).

RewriteRule

RewriteRule

RewriteRule Pattern Substitution Flags

- The RewriteRule directive is the real rewriting workhorse
- The directive can occur more than once
 - Each directive defines one single rewriting rule
- The **definition order** of these rules is **important**
 - This order is used when applying the rules at run-time
- *Pattern* is a perl compatible regular expression which gets applied to the current URL
 - Here "current" means value of the URL when rule is applied
 - May not be the originally requested URL, because other rules may already have matched and made alterations to it

Regular Expressions

Regular Expressions

■ Matching Text:

- . Any single character
- [chars] Any one of chars in []
- [^chars] Any char but those in []
- text1|text2 text1 or text2

■ Quantifiers:

- ? 0 or 1 of the preceding text
- * 0 or N of the preceding text (N > 0)
- + 1 or N of the preceding text (N > 1)

Regular Expressions - cont.

Regular Expressions - cont.

- Grouping:
 - (text) Grouping of text
- Anchors:
 - ^ Start of line anchor
 - \$ End of line anchor
- Escaping:
 - \char escape that particular character

Searching and Regexp's

Searching and Regexp's

- regular expressions (regexps) can search for patterns in strings.
- The syntax is:
 - `<expression>/`
 - `"Hello, World" =~ /Hello/ #matches`
 - `"Hello, World" =~ /World/ #also matches`
 - `"Hello, World" =~ /world/ #doesn't match`
 - `"Hello, World" =~ /world/i #matches.. i stands for case insensitivity`

\ and ^ operators

\ and ^ operators

- It is possible to use special symbols in patterns. They only have to be escaped using the '\'

```
\.\.\.|\.\.\.
```

- Search for ellipses (...) with or without spaces
- The carrot '^' can be used as an "anchor" to tell the regexp to affix the pattern to the beginning of a line.

```
^Ju|^Th i
```

- A string that begins with 'Ju' or 'Th' as in 'July' or 'Thursday'

\$ and [] Operators

\$ and [] Operators

- The dollar sign \$ is like the ^, except that it anchors an expression to the end of a line
 - e\$ # Matches time or waste
 - ^\$ # Only matches an empty string ""
- Character classes are denoted using the square brackets [], The regexp will match a class if any of the characters in it match
 - [abcdef] # Matches any letter a through f
 - [a-f] # Also matches any letter a through f
 - [^a-f] # Matches any character except for a through f
 - [^aeiou] # Matches the word if it contains a consonant
- Note in the above case that a ^ inside the character class negates the class, while outside it is used as an anchor for the beginning of a line.
- Also ,if the class does not begin with [^, then the ^ has the same meaning as if it were outside of the character class.

Parenthesis ()

Parenthesis ()

- `$var =~ /(ca[tk]{2})/`
 - Matches `ca[tk][tk]` and stores that in memory, that can be accessed with `$1`.
 - Subsequent storages can be stored with higher numbers. (e.g. `$2`, `$3`, . . . `$9`)
- `$var ~ /((ca[tk]){2})/`
 - Taking the previous example, assume we wanted to capture the entire matching string to regular expression memory

Curly Braces `{}` and Repetition

Curly Braces `{}` and Repetition

- `$var =~ /[aeiou]{2}/`
 - Matches if there are two vowels in a row
- `$var =~ /[aeiou]{2,5}/`
 - Matches if there are between three and than five vowels in a row
- `$var =~ /[aeiou]{3,}/`
 - Matches if there are at least three vowels in a row

Other Operators and Repetition

Other Operators and Repetition

- ?
 - causes the regexp to search for a pattern with the preceding character or pattern appearing 0 or 1 times.
 - a?ll # Matches 'all' or even 'll'
- *
 - Matches 0 or more occurrences of the preceding pattern
 - (new)* # Matches '' 'new' or 'newnewnew'
- +
 - Like * except that it requires at least one of the preceding letter or pattern.
 - n+ew # Matches 'new' or 'nnnnnnnew' but not 'ew'

Greedy Evaluation

Greedy Evaluation

- Greedy evaluation is done by default.
- Operators attempt to match as much of the string as possible.
- Problems may occur with multiple wildcards.
(. + .*)
- For matches to match a minimum instead of a maximum append a ? onto the character that is doing a repetitive match.

Optionality and Repetition

Optionality and Repetition

- `/[Ww]oodchucks?/`
 - matches woodchucks, Woodchucks, woodchuck, Woodchuck
- `/colou?r/`
 - matches color or colour
- `/he{3}/`
 - matches heee
- `/(he){3}/`
 - matches hehehe
- `/(he){3,}/`
 - matches a sequence of at least 3 he's

A Simple Exercise

A Simple Exercise

- Write a regular expression to find all instances of the determiner "the":

The recent attempt by the police to retain their current rates of pay has not gathered much favor with the southern factions.

A Simple Exercise



- Write a regular expression to find all instances of the determiner "the":

/the/

The recent attempt by the police to retain their current rates of pay has not gathered much favor with

A Simple Exercise

- Write a regular expression to find all instances of the determiner "the":

/the/

/[tT]he/

The recent attempt by the police to retain their current rates of pay has not gathered much favor with the southern factions.

A Simple Exercise

- Write a regular expression to find all instances of the determiner "the":

/the/

/[tT]he/

*The recent attempt by the police to retain **their** current rates of pay has not gathered much favor with the sou**the**rn factions.*

A Simple Exercise

- Write a regular expression to find all instances of the determiner "the":

/the/

/[tT]he/

/\b[tT]he\b/

The recent attempt by the police to retain their current rates of pay has not gathered much favor with the southern factions.

A Simple Exercise

- Write a regular expression to find all instances of the determiner "the":

/the/

/[tT]he/

/\b[tT]he\b/

/(^[^a-zA-Z])[tT]he[^a-zA-Z]/

The recent attempt by the police to retain their current rates of pay has not gathered much favor with the southern factions.

RewriteRule Flags

RewriteRule Flags

- [R] force Redirect
 - Redirect the URL to a external redirection. Send the HTTP response, 302 (MOVED TEMPORARILY).

- [F] force URL to be Forbidden
 - Forces the current URL to be forbidden. Send the HTTP response, 403 (FORBIDDEN).

- [G] force URL to be Gone
 - Forces the current URL to be gone. Send the HTTP response, 410 (GONE).

RewriteRule Flags - cont.

RewriteRule Flags - cont.

- [L] last rule
 - Forces the rewriting processing to stop here and don't apply any more rewriting rules.

- [P] force proxy
 - This flag forces the current URL as a proxy request and put through the proxy module **mod_proxy**.

RewriteCond

RewriteCond

- The `RewriteCond` directive defines a rule condition

```
RewriteCond TestString CondPattern flags
```

- Flags include
 - `[NC]` No Case
 - This makes the condition pattern case insensitive, no difference between 'A-Z' and 'a-z'.
 - `[OR]` OR next condition
 - Used to combine rule conditions with a OR

Precede a **RewriteRule** with one or more **RewriteCond** directives. The following rewriting rule is only used if its pattern matches the current state of the URI and if these additional conditions apply too.

You can set special flags for condition pattern by appending a third argument to the `RewriteCond` directive. `Flags` is a comma-separated list of the following flags:

[NC] (No Case)

This makes the condition pattern case insensitive, no difference between 'A-Z' and 'a-z'.

[OR] (OR next condition)

Used to combine rule conditions with a OR.

Condition Patterns

Condition Patterns

- **< Condition** is less than *Condition*
- **> Condition** is greater than *Condition*
- **= Condition** is equal to *Condition*
- **-d** is a directory
- **-f** is regular file
- **-s** is regular file with size greater than 0
- **-l** is symbolic link
- **-F** is existing file via sub request
- **-U** is existing URL via sub request
- NOTE: You can prefix the pattern string with a '!' character (exclamation mark) to specify a non-matching pattern

There are some special variants of CondPatterns. Instead of real regular expression strings you can also use one of the following:

< Condition (is lower than Condition)

Treats the Condition as a string and compares it to String. True if String is lower than Condition.

> Condition (is greater than Condition)

Treats the Condition as a string and compares it to String. True if String is greater than CondPattern.

= Condition (is equal to Condition)

Treats the Condition as a string and compares it to String. True if String is equal to CondPattern.

-d (is directory)

Treats the String as a pathname and tests if it exists and is a directory.

-f (is regular file)

Treats the String as a pathname and tests if it exists and is a regular file.

-s (is regular file with size)

Treats the String as a pathname and tests if it exists and is a regular file with size greater than zero.

-l (is symbolic link)

Treats the String as a pathname and tests if it exists and is a symbolic link.

-F (is existing file via sub request)

Checks if String is a valid file and accessible via all the server's currently configured access controls for that path. Use it with care because it decreases your servers performance!

-U (is existing URL via sub request)

Checks if String is a valid URL and accessible via all the server's currently configured access controls for that path. Use it with care because it decreases your servers performance! NOTE: You can prefix the pattern string with a '!' character (exclamation mark) to specify a non-matching pattern

Logging

Logging

- Logging is vital with `mod_rewrite`
 - It is of great help when you're not getting the expected result and you can't figure out why
- Add to `httpd.conf` or `.htaccess`

```
RewriteEngine On  
RewriteLog /var/log/apache/rewrite.log  
RewriteLogLevel 9
```

Which method (`.htaccess` or `httpd.conf`) is better? Most likely putting it in the `conf` file, but the difference may not be enough if you're really [looking for bottle-necks](#). You can put `rewrite` code in both places, but again be careful of what overrides what.

Useful Rewrite Example

Useful Rewrite Example

- Protecting your images from external linking

```
RewriteEngine On
RewriteCond %{HTTP_REFERER} !^$ [NC]
RewriteCond %{HTTP_REFERER} !^http://domain.com [NC]
RewriteCond %{HTTP_REFERER} !^http://www.domain.com [NC]
RewriteCond %{HTTP_REFERER} !^http://212.204.218.80 [NC]
RewriteRule ^.*$ http://www.domain.com/ [R,L]
```

DESCRIPTION: In some cases other webmasters are linking to your download files or using images, hosted on your server as inline-images on their pages.

EXPLAIN: In this case are the visitors redirect to **http://www.domain.com/** if the hyperlink has not arrived from **http://domain.com**, **http://www.domain.com** or **http://212.204.218.80**.

Useful Rewrite Example - cont.

Useful Rewrite Example - cont.

- Redirect visitor by domain name
 - Allow access to site by multiple names, but redirect to only use one

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

DESCRIPTION: In some cases the same web site is accessible by different addresses, like **domain.com**, **www.domain.com**, **www.domain2.com** and we want to redirect it to one address.

EXPLAIN: In this case the requested URL **http://domain.com/foo.html** would be redirected to the URL **http://www.domain.com/foo.html**.

Useful Rewrite Example - cont.

Useful Rewrite Example - cont.

- Redirect visitor by user agent
 - Different browsers have different 'features'
 - May want to show different content

```
# MS Internet Explorer - Mozilla v4
RewriteEngine On
RewriteCond %{HTTP_USER_AGENT} ^Mozilla/4(.*MSIE
RewriteRule ^index\.html$ /index.IE.html [L]

# Netscape v6.+ - Mozilla v5
RewriteCond %{HTTP_USER_AGENT} ^Mozilla/5(.*Gecko
RewriteRule ^index\.html$ /index.NS5.html [L]

# All other browsers
RewriteRule ^index\.html$ /index.32.html [L]
```

DESCRIPTION: For important top level pages it is sometimes necessary to provide pages dependent on the browser. One may want to provide a version for the latest Netscape, a version for the latest Internet Explorer, a version for the Lynx or old browsers and a average feature version for all others

EXPLAIN: In this case we have to act on the HTTP header **User-Agent**. If the **User-Agent** begins with **Mozilla/4** and is MS Internet Explorer (**MSIE**), the page index.html is rewritten to index.IE.html and the rewriting stops. If the **User-Agent** begins with **Mozilla/5** and is Netscape (**Gecko**), the page index.html is rewritten to index.NS5.html. If the **User-Agent** begins with **Lynx/** or **Mozilla/1,2**, the page index.html is rewritten to index.20.html. All other browsers receive page index.32.html

http Authentication

http Authentication



Web Security Issues

Web Security Issues



- Aware of the common Security risks
 - Access restrictions for site areas
 - Understand Apache log files
-

User Authentication

User Authentication



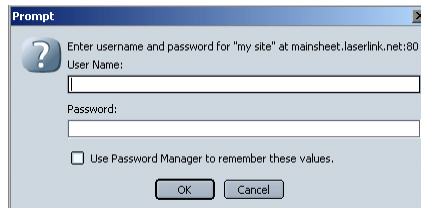
- Apache default authentication mechanism is mod_auth
 - Uses a password file generated using Apache's htpasswd utility

```
geoff:zzpEyL0tbgwwk
```

User Authentication

- Configuration can be placed in `.htaccess` file or `httpd.conf`

```
AuthUserFile access/.htpasswd  
AuthName "my site"  
AuthType Basic  
Require valid-user
```



Who Uses Flat Files?

Who Uses Flat Files?



- Flat files are
 - Limiting
 - hard to manage
 - difficult to integrate
 - and just plain boring
 - We can use the Apache API to replace flat files with other authentication mechanisms
-

How Authentication Works (1)

How Authentication Works (1)



■ Client requests a document

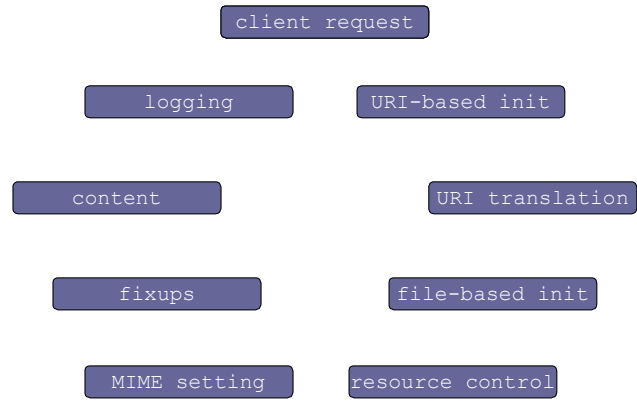
```
GET /perl-status HTTP/1.1
Accept: text/xml, image/png, image/jpeg, image/gif, text/plain
Accept-Charset: ISO-8859-1, utf-8;q=0.66, *;q=0.66
Accept-Encoding: gzip, deflate, compress;q=0.9
Accept-Language: en-us
Connection: keep-alive
Host: www.example.com
Keep-Alive: 300
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US)
```

■ Server denies request

```
HTTP/1.1 401 Authorization Required
WWW-Authenticate: Basic realm="my site"
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=iso-8859-1
```

How Authentication Works (2)

How Authentication Works (2)



How Authentication Works (3)

How Authentication Works (3)

client request

logging

URI-based init

URI translation

file-based init

resource control

HTTP/1.1 401 Authorization Required

How Authentication Works (4)

How Authentication Works (4)

■ Client sends a new request

```
GET /perl-status HTTP/1.1
Accept: text/xml, image/png, image/jpeg, image/gif, text/plain
Accept-Charset: ISO-8859-1, utf-8;q=0.66, *;q=0.66
Accept-Encoding: gzip, deflate, compress;q=0.9
Accept-Language: en-us
Authorization: Basic Z2VvZmY6YWZha2VwYXNzd29yZA==
Connection: keep-alive
Host: www.example.com
Keep-Alive: 300
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US)
```

■ Server sends document

```
HTTP/1.1 200 OK
Keep-Alive: timeout=15, max=99
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html
```

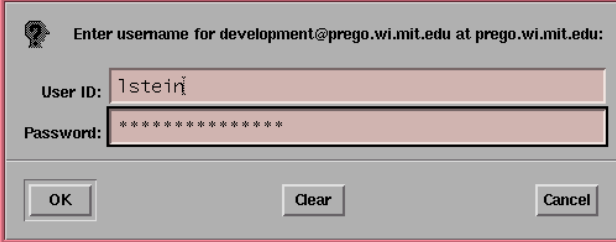
Password Protection of Web Pages

Password Protection of Web Pages

- Password protection requires
 - A list of users and passwords
 - A per-directory list of permissions
 - Overall Permissions in access.conf can be overridden by .htaccess files
 - Passwords stored in .htpasswd files
 - Use the htpasswd program to produce and maintain .htpasswd files
-

Basic Authentication

Basic Authentication



A dialog box for basic authentication. The title bar contains a question mark icon and the text "Enter username for development@prego.wi.mit.edu at prego.wi.mit.edu:". Below the title bar, there are two input fields. The first is labeled "User ID:" and contains the text "tstein". The second is labeled "Password:" and contains ten asterisks. At the bottom of the dialog box, there are three buttons: "OK", "Clear", and "Cancel".

Enter username for development@prego.wi.mit.edu at prego.wi.mit.edu:

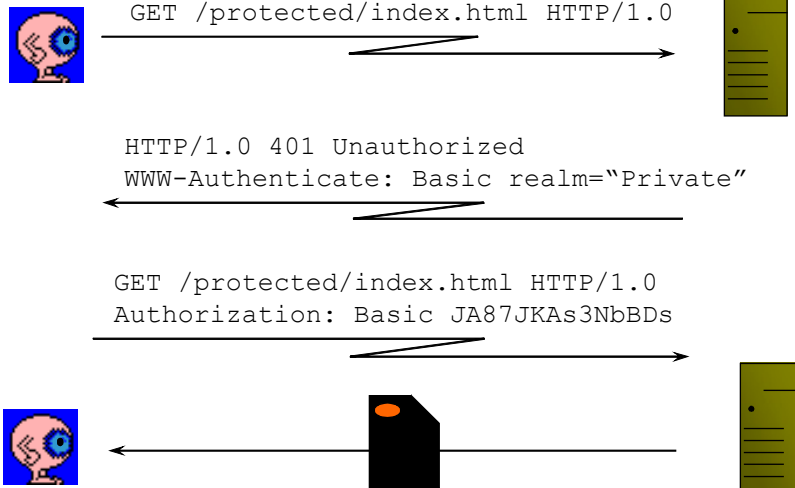
User ID: tstein

Password: *****

OK Clear Cancel

How Basic Authentication Works

How Basic Authentication Works



Problems with Basic Authentication

Problems with Basic Authentication

- Passwords easy to intercept
- Passwords easy to guess
- Passwords easy to share
- No server authentication
 - Easy to fool client into sending password to malicious server
- One intercepted password gives eavesdropper access to many documents

Digest (Challenge/Response) Auth

Digest (Challenge/Response) Auth



Challenge and Response

Challenge and Response

- Challenge ("nonce"): any changing string
 - e.g. MD5(IP address:timestamp:server secret)
- Response: challenge hashed with user's name & password
 - MD5(MD5(name:realm:password):nonce:MD5(request))
- Server-specific implementation options
 - One-time nonces
 - Time-stamped nonces
 - Method authentication digests

Advantages of Digest over Basic Auth

Advantages of Digest over Basic Auth

- Clear text password never transmitted across network
- Clear text password never stored on server
- Replay attacks difficult
- Intercepted response only valid for a single URL
- Shared disadvantages
 - Vulnerable to man-in-the-middle attacks
 - Document itself can be sniffed

Availability of Digest Authentication

Availability of Digest Authentication

- Part of HTTP/1.1 protocol
- Server support
 - Since Apache 1.2
- Browser support
 - All modern browsers

Dynamic Content with CGI & Perl

Dynamic Content with CGI & Perl



Things alter for the worse
spontaneously, if they be not
altered for the better designedly.

Francis Bacon

Creating Dynamic Content

Creating Dynamic Content



- Describe CGI's rich communication methods
 - Manage the rights to CGI execution
 - Use CGI to collect Site stats
 - Outline the built in functions for CGI in Perl
 - Use Server Side includes to call CGI scripts
-

The Common Gateway Interface (CGI)

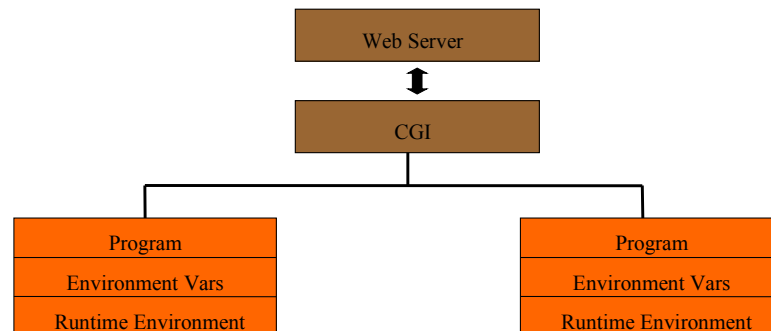
The Common Gateway Interface (CGI)

- CGI defines an interface between a Web server and an independent application program.
 - CGI are used to create “gateways” between the Web and an existing application.
 - CGI also serve as the interface for new applications designed for the Web, not integrated directly into a Web server (as in plug-ins).
-

CGI (Common Gateway Interface)

CGI (Common Gateway Interface)

- Platform independent interface that allows Apache to execute programs on the server, then send the output to the client's web browser



Apache provides support for the Common Gateway Interface (CGI). CGI is a platform independent interface that allows apache to execute programs on the server machine, then send the output to the client's web browser. The beauty of CGI is that the programs on the server don't need to know anything about networks, or which machine is executing the program - they just return output to standard out and the web server takes care of the rest.

Input parameters to CGI programs are passed through the standard input stream (as if it were passed in from the keyboard at the command line) as a string of name=value pairs. For example, the search program that is used to find sites at www.yahoo.com takes a single input parameter, the search string.

You can explicitly call the yahoo search engine at
<http://search.yahoo.com/bin/search?p=computers>

This URL is passing a variable called p to the CGI program search with the value "computers". Passing more than one variable can be achieved by separating name value pairs with the ampersand (&) character. e.g.

<http://www.mycompany.com/getdetails?name=john smith&age=45>

Server API for CGI

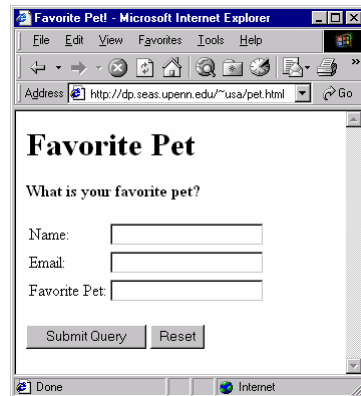
Server API for CGI



- Starting and stopping application
 - Passing data from the client to the application
 - Passing data from the application to the client
 - Status and error reporting
 - Passing configuration information to the application
 - Passing client and environment information to the application
-

CGI Example (HTML form)

CGI Example (HTML form)

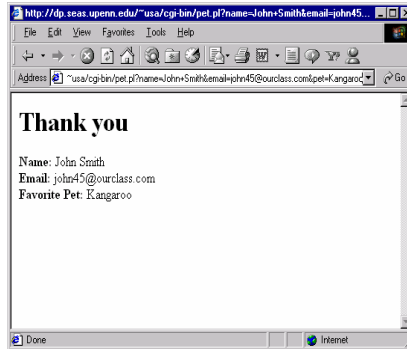


A screenshot of a Microsoft Internet Explorer browser window. The title bar reads "Favorite Pet! - Microsoft Internet Explorer". The address bar shows "http://dp.seas.upenn.edu/~usa/pet.html". The main content area displays a form titled "Favorite Pet" with the question "What is your favorite pet?". The form contains three text input fields labeled "Name:", "Email:", and "Favorite Pet:". Below the fields are two buttons: "Submit Query" and "Reset".

```
<HTML>
<HEAD>
<TITLE>Favorite Pet!</TITLE>
</HEAD>
<BODY BGCOLOR="white">
<H1>Favorite Pet</H1>
<B>What is your favorite pet?</B>
<FORM METHOD="GET" ACTION="cgi-bin/pet.pl">
<TABLE>
<TR>
<TD>Name:</TD>
<TD><INPUT TYPE="TEXT" NAME="name"></TD>
</TR>
<TR>
<TD>Email:</TD>
<TD><INPUT TYPE="TEXT" NAME="email"></TD>
</TR>
<TR>
<TD>Favorite Pet:</TD>
<TD><INPUT TYPE="TEXT" NAME="pet"></TD>
</TR>
</TABLE>
<P><INPUT TYPE="SUBMIT" VALUE="Submit Query">
<INPUT TYPE="RESET"></P>
</FORM>
</BODY>
</HTML>
```

CGI Example (GET)

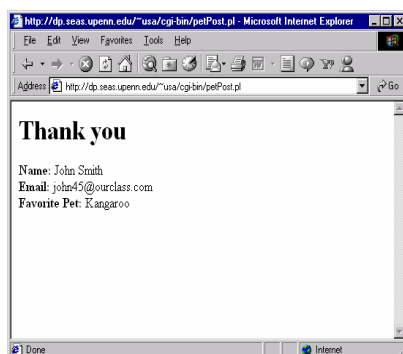
CGI Example (GET)



```
#!/usr/bin/perl -w
use CGI qw(:standard);
print "Content-type: text/html", "\n\n";
@pairs = split('&', $ENV{'QUERY_STRING'});
foreach $pair (@pairs) {
    ($name, $value) = split('=', $pair);
    $value =~ tr/+//;
    $value =~ s/%[a-fA-F0-9][a-fA-F0-9]/
        pack("C", hex($1))/eg;
    $info{$name} = $value;
}
print "<HTML>", "\n";
print "<BODY><H1>Thank you</H1>", "\n";
print "<B>Name:</B>", $info{name}, "<BR>", "\n";
print "<B>Email:</B>", $info{email}, "<BR>", "\n";
print "<B>Favorite Pet:</B>", $info{pet}, "<BR>", "\n";
print "</BODY></HTML>";
```

CGI Example (POST)

CGI Example (POST)



```
#!/usr/bin/perl -w
use CGI qw(:standard);

print "Content-type: text/html", "\n\n";

read(STDIN, $buffer,
     $ENV{'CONTENT_LENGTH'});
@pairs = split('&', $buffer);
foreach $pair (@pairs) {
    ($name, $value) = split('=', $pair);
    $value =~ tr/+//;
    $value =~ s/%([a-fA-F0-9][a-fA-F0-9])/
                pack("C", hex($1))/eg;
    $info{$name} = $value;
}

print "<HTML>","\n";
print "<BODY><H1>Thank you</H1>","\n";
print "<B>Name:</B> ", $info{name}, "<BR>","\n";
print "<B>Email:</B> ", $info{email}, "<BR>","\n";
print "<B>Favorite Pet:</B> ", $info{pet}, "<BR>","\n";
print "</BODY></HTML>";
```

Student Notes

Perl on Red Hat is distributed with an extension module for the purpose of CGI programming called `cgi.pm`. The CGI module can be included for use in programs using two different programming styles.

`use CGI;`

allows an object oriented style of program. A CGI object has to be initialized and any functions in the CGI module must be used by referencing the CGI object.

`use CGI qw(:standard/;`

Imports the set of standard CGI functions into the default namespace and initializes a default object. This method of inclusion allows direct reference to CGI functions, but has the negative side effect of only allowing one CGI object to be used in a program.

Generating Output for CGI

Although CGI takes care of the networking issues involved in web programming, it is not quite as simple to generate CGI output as output for the screen.

Firstly, the output from a CGI script is going to be displayed in a web browser, so it must be formatted as html.

Secondly, when returning the output from a CGI script the web server requires an extra header line to tell it what the content of the document is. This extra header information can be generated automatically by the server for documents that have standard extensions. The server assumes that `index.html` is an html document and therefore the content will be text. When writing CGI programs, the extension is always `.cgi` or `.pl`, whatever the content output of the program (it could be an image or an executable) so this extra heading information must be generated manually. The CGI module has a built in header function to do this.

CGI Environment Variables

CGI Environment Variables

Variable Name	Value
HTTP_HOST	The hostname of your server
HTTP_USER_AGENT	The browser type of the visitor
HTTPS	"on" if the script is being called through a secure server
QUERY_STRING	The query string
REMOTE_ADDR	The IP address of the visitor
REMOTE_HOST	The hostname of the visitor
REMOTE_PORT	The port the visitor is connected to on the web server
REQUEST_METHOD	GET or POST
SERVER_NAME	The server's domain name
SERVER_PORT	The port number the server is listening on
SERVER_SOFTWARE	The server software used (e.g. Apache 1.3.12)

Student Notes

In addition to explicitly defined variables, the CGI interface also sends a standard set of environment variables to the CGI program. Using these environment variables, it is possible to ascertain, amongst other things, the version and manufacturer of the client browser and the IP address of the client. Every time a browser connects to a CGI script, a package of variables is placed in the script's environment. These CGI variables are accessed in exactly the same way as standard Unix environment variables such as PATH and HISTORY.

Useful CGI environment variables

GATEWAY_INTERFACE	The version of CGI used by the server.
HTTP_USER_AGENT	The client browser name and version number.
QUERY_STRING	The user provided data from a form submission.
REMOTE_ADDR	The IP address of the client machine.
REMOTE_HOST	The host name of the client machine.
REMOTE_USER	If user authentication is enabled, this variable will contain the user id of the user accessing the script
REQUEST_METHOD	The method by which the script was called (most often, GET or POST).
SCRIPT_NAME	The name and path to the called script.
SERVER_NAME	The value set in the ServerName directive of httpd.conf.
SERVER_PORT	The value set in the Port directive of httpd.conf
SERVER_SOFTWARE	The name and version of the Web server.

The REMOTE_HOST variable will only display the client host name if Apache is configured to run host name lookups for every client that connects to the server. This feature will increase the load on the DNS server, increase network traffic and degrade server performance.

Host name lookups can be turned on by setting

```
HostnameLookups Off
```

to

```
HostnameLookups On
```

in httpd.conf. Unless it is really necessary to generate host names, this option should be turned off. It is still possible to monitor the hosts that have accessed the site through their IP addresses and host name lookups can be made at any time for the stored IP addresses.

LAB

LAB

■ Make hellocgi.pl example

Practical Exercise

Try the hellocgi.pl example for yourself.

hellocgi.pl

Use Notepad to create hellocgi.pl containing the following lines;

```
#!/wperl.exe -w
use CGI qw/:standard/;
print header;
print start_html;
print "Hello World!\n";
print end_html;
```

When running the hellocgi.pl script at the command line something strange happens.

./hellocgi.pl

(offline mode: enter name=value pairs on standard input)

The script generates no output and the command line fails to return.

This behavior is a feature of any perl program that has the CGI module enabled. The perl interpreter waits for CGI input variables before trying to execute the program, allowing a simulation of the CGI environment at the command line. For now, pressing ctrl-D will tell the interpreter that there are no input variables and allow execution to continue.

Content-Type: text/html

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML><HEAD><TITLE>Untitled Document</TITLE>
</HEAD><BODY>Hello World.
</BODY></HTML>
```

The

```
print header;
```

statement generates the output:

Content-Type: text/html

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
```

The

```
print start_html;
```

statement generates the output:

```
<HTML><HEAD><TITLE>Untitled Document</TITLE>
```

The

```
print end_html;
```

statement generates the output:

```
</BODY></HTML>
```

Enabling CGI in Apache

Enabling CGI in Apache

- There are two methods for enabling CGI execution
 - `AddHandler cgi-script`
 - add `ExecCGI` option to root directory
 - or
 - `cgi-bin` directory
 - `ScriptAlias /cgi-bin/ "/>`

There are two methods for enabling CGI execution in the Apache environment.

The `AddHandler cgi-script` Directive

This is the most straightforward method of enabling cgi scripts. Remove the comment from the line `#AddHandler cgi-script .cgi`

and add the `ExecCGI` option to the document root directory. For example:

```
<Directory />  
    Options FollowSymLinks ExecCGI  
    AllowOverride None  
    Allow from all  
</Directory>
```

Once the modifications have been made, the server will run any file ending with a `.cgi` extension as a cgi script. There are considerable security flaws to adopting the `.cgi` extension approach to CGI execution.

CGI scripts are stored in the same directory as html files. In large directories, it may be difficult to compile a definitive list of the scripts residing on the system.

Because the scripts are in the same file as html documents, a CGI script could inadvertently (or deliberately) damage web site content.

Because the permission to execute CGI scripts is automatically transferred to sub-directories unless it is specifically overridden and there are many directories that could possibly contain CGI content, it is unlikely that the CGI execution permissions will be configured with tight enough control.

The default set up for Apache is to allow access to the rest of the file system from document root. If the user nobody has access to a directory, a CGI script can access it.

For example:

```
.passwd.cgi  
#!/bin/sh  
echo "Content-Type: text/html" ""  
echo '<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">'  
cat /etc/passwd
```

This script will not be shown in a standard directory listing because its name begins with a period character. If this script is placed in any sub directory of the document root and the user nobody has access to the `/etc/passwd` file, it will list all the users on the system and display their encrypted passwords.

The `cgi-bin` Directory

The preferred option for CGI execution configuration is to place all the scripts in a separate directory called `cgi-bin`. The `cgi-bin` directory can be located on a completely different part of the Unix file system to the document root directory meaning that it is impossible for scripts to access the html content of a site. Because all the CGI scripts are in one directory, it is easy to audit and monitor the CGI usage on the server and the directory permissions for `cgi-bin` can be controlled very tightly.

The `cgi-bin` directory is set to be the root directory for CGI scripts. This means that no directory above `cgi-bin` in the directory tree is accessible to CGI scripts. The `.passwd.cgi` example shown above will not work when placed in the `cgi-bin` directory.

Enabling the `cgi-bin` Directory

The `cgi-bin` directory is enabled using the `ScriptAlias` directive. For example:
`ScriptAlias /cgi-bin/ "/>`

The cgi-bin directory should also have a Directory container to allow the specification of access permissions.

```
<Directory "/testapache/cgi-bin">  
    AllowOverride None  
    Order allow, deny  
    Allow from all
```

```
</Directory>
```

There is no need to specify the ExecCGI option for the directory, as the ScriptAlias directive allows any file with Unix execute permission stored in the cgi-bin directory to be executed.

LAB

LAB

- Enabling and Disabling CGI in apache

Make sure that the .cgi extension is disabled.

Place the hellocgi.pl file created earlier in the server document root directory.

Rename the file to hello.cgi.

Browse the URL *http://localhost/hello.cgi*.

What is displayed?

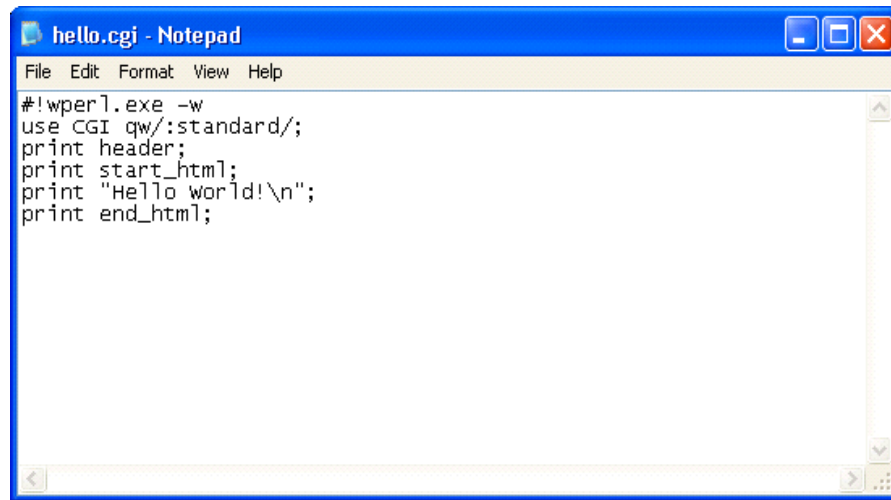
Ensure that the cgi-bin directory is enabled (it should be by default) and is set to /Applications/Apple2/cgi-bin.

Move hello.cgi to /Applications/Apple2/cgi-bin.

Browse the URL *http://localhost/cgi-bin/hello.cgi*.

What is displayed?

LAB



```
hello.cgi - Notepad
File Edit Format View Help
#!wperl.exe -w
use CGI qw/:standard/;
print header;
print start_html;
print "Hello world!\n";
print end_html;
```

Evaluation of CGI

Evaluation of CGI

■ Advantages of CGI

- General: the application is completely decoupled from the Web server
- Standard: works with every sever and browser
- Flexible: any language (C++, Perl, Java, ...) can be used

■ Disadvantages of CGI

- Inefficient: the application must be launched/forked independently for each request
 - Stateless: the application exits after a request, there is no place to remember state between Web requests
 - Security: CGI programmer is responsible for security. No automatic system or language support.
-

Server-Side Scripts and Includes

Server-Side Scripts and Includes

■ Includes

- Allow script output or environment variables to be included in Web pages. For example
 - Page text
 - Date, time
 - Author name and Email address

■ Scripts

- Allows the creation of simple gadgets, or more complex tools for such things as:
 - Forms processing
 - Interfaces to databases, graphics libraries

Server Side Scripting

Normally, HTML is sent back to the client exactly as it is on the disk, with no server intervention. However, sometimes you might find it useful to have the server parse these files and insert request-specific information or files into the document. You can do this through parsed HTML.

You should also choose a method for the server to determine which files should be parsed and which should not. The usual method is to use a filename extension of .shtml for server parsed HTML files. You can also use a filename of your choosing. The web server must be configured to parse the HTML files.

The server can also only look at files with the UNIX file permissions set such that the execute bit is on. This is often unreliable, though, as sometimes documents have the execute bit set even though they are not really executable.

The server can also look at every HTML file on the server. This incurs a large performance since the server must now look at every single HTML file it sends back from this directory.

Having the server parse documents is a double edged sword. It can be costly for heavily loaded servers. Furthermore it can be a security risk for a user to be able to execute commands on the server.

Integrating Tomcat into Apache

Integrating Tomcat into Apache



Integrating Tomcat into Apache



- What is Tomcat?
- J2EE
- Installing Tomcat Binaries
- Using Tomcat with Apache via mod_jk

What is Tomcat?

What is Tomcat?



- Servlet container
- Reference Implementation
- Tomcat 5.5 implements:
 - Servlet 2.4
 - Java Server Pages 2.0

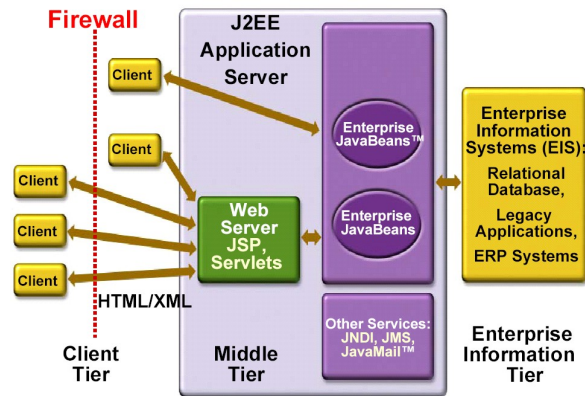
Apache Tomcat is the servlet container that is used in the official Reference Implementation for the [Java Servlet](#) and [JavaServer Pages](#) technologies. The Java Servlet and JavaServer Pages specifications are developed by Sun under the [Java Community Process](#).

Tomcat is developed in an open and participatory environment and released under the [Apache Software License](#). Tomcat is intended to be a collaboration of the best-of-breed developers from around the world

<http://tomcat.apache.org/>

J2EE

J2EE



J2EE technology and its component based model simplifies enterprise development and deployment. The J2EE platform manages the infrastructure and supports the Web services to enable development of secure, robust and interoperable business applications. The J2EE platform is the foundation technology of the Sun ONE platform and Sun's Web services strategy.

Java Servlets and Java Server Pages as implemented by Tomcat are part of the J2EE model.

Installing Tomcat Binaries

Installing Tomcat Binaries

- Download a suitable distribution from:

<http://tomcat.apache.org/>

- Simply download the distribution .exe on to your hard drive.
- Run the installation and configure options
- Test by pointing browser at

<http://localhost:8080/>

Tomcat Welcome Page

Tomcat Welcome Page



Integrating Tomcat with Apache

Integrating Tomcat with Apache

- Works stand alone on port 8080
- Can be used via a proxy from apache
- Using `mod_jk 1.2.x` or `mod_proxy`
 - An AJP connector, e.g. `mod_jk`, will provide faster performace than proxied HTTP
 - `mod_jk2` is no longer a supported option

Using a Proxy

Using a Proxy

- Enable the proxy connector in server.xml
- Set the proxy name to the current machine
- Turn on the proxy in apache
- Map calls to port 8080* with ProxyPass and ProxyPassReverse enteries
- * Or port specified in server.xml proxy connector

Tomcat is configured using the file server.xml, which may be found in the conf subdirectory. An appropriate entry for a proxy connector is already present, it just needs un-commenting and the proxyName entry updating to be the current machine name:

```
<Connector className="org.apache.catalina.connector.http.HttpConnector"
  port="8081" minProcessors="5" maxProcessors="75"
  enableLookups="true"
  acceptCount="10" debug="0" connectionTimeout="60000"
  proxyPort="80"
  proxyName="u607su61.aah.co.uk"/>
```

To use Apache as the front end, it must have both the proxy module and alias modules present (this is the case if the instructions above have been followed, 'modules=most' includes them), and the httpd.conf file requires additional entries. Alternatively, and this is the approach we recommend, the configuration for an application can be placed in a separate configuration file and included in the httpd.conf file (See Section 10).

The restrictions of using a straight proxy are that all directory paths between the front end server, Apache, and the back end server, Tomcat must be kept perfectly aligned. This is because when Tomcat generates cookies for session management it includes a path for the cookie that relates to the tomcat path. When the browser hits a web page it sees the Apache path, and checks that against the cookie path. If the paths do not match then the cookie will not be returned, and the session state will be lost.

To turn on proxy in Apache, edit the section shown below into the application's configuration file. The important lines are the ProxyPass and ProxyPassReverse. These lines should always be used in pairs. You may use as many pairs as you wish to map paths.

```
<IfModule mod_proxy.c>
ProxyRequests On
ProxyVia On
ProxyPass /myapp http://localhost:8081/myapp
ProxyPassReverse /myapp http://localhost:8081/myapp
ProxyPass /mapping/controller http://localhost:8081/mapping/controller
ProxyPassReverse /mapping/controller http://localhost:8081/mapping/controller
</IfModule>
```

NOTE: Be very careful with the last character of the path, you should either always use a trailing '/' or never use one. Since to keep the cookie sessions working correctly we have to do a one-to-one mapping of directories, this does not allow any easy and elegant way of separating different application static images and pages so they are served by Apache rather than Tomcat. If there are placed in the tree in an application specific manner, the easiest to manage from a development and deployment perspective, Apache will simply proxy requests for static content through to Tomcat.

To change this behaviour we suggest using the alias module, which allows us to map http request paths to alternative paths on the Apache web server machine. The example below from a configuration file changes requests received for '/mapping/images', which may be notionally part of a Tomcat aliased path to be served by Apache from the directory '/usr/local/tomcat/webapps/mapping/images/' on the Apache server. Notice that there is no requirement that the target of the alias be under the standard httpd documents directory.

The <Directory> directive controls permissions. In most cases it will not be required. The one shown here allows the images directory to be browsed by all users. See the Apache documentation for additional details.

```
<IfModule mod_alias.c>
Alias /mapping/images/ "/usr/local/tomcat/webapps/mapping/images/"
```

```
<Directory "/usr/local/tomcat/webapps/mapping/images">  
  Options Indexes MultiViews  
  AllowOverride None  
  Order allow,deny  
  Allow from all  
</Directory>  
</IfModule>
```

Using the mod_jk connector

Using the mod_jk connector

- Install mod_jk as DSO in apache
 - Configure in tomcat files
 - server.xml
 - Configure in mod_jk files
 - worker.properties
 - mod_jk.conf
 - Configure in apache file
 - httpd.conf
-

LAB:

LAB:

- Install Sun's JDK
- Install Tomcat from Binaries
- Install the mod_jk connector

Install JDK

go to Sun's Java download site and download J2SE for Windows:

<http://java.sun.com/j2se/1.4.2/download.html>. I prefer the SDK version, To install J2SDK, just execute the file you downloaded. Once the Installer is finished unpacking, you'll be presented with a standard license prompt. If you agree, choose the appropriate option and click **Next**.

The installer will display a standard installation dialog. Click **Next**. This will install J2SE in **C:\j2sdk1.4.2_01**, which is fine.

Register the browsers on your system, and click **Install**.

After the install has finished, click **Finish**.

Build/Install Tomcat

First, create a folder on your hard drive. Call it **Tomcat**, and put it in the C drive root, so that you end up with **C:\Tomcat**. **NOTE:** You can put Tomcat wherever you like, it is your choice.

Wherever you put it, try to keep spaces or other unusual characters out of the pathname.

Now, grab the Windows EXE distribution for Tomcat from a mirror site.

Once it is downloaded, execute the file.

Agree to the license, and click **Next**. On the next screen, check the box marked **NT Service**. Click **Next** again.

Next, change the directory where Tomcat will be installed. Change the default so that there are no spaces in the pathname.

When the install completes, you'll be presented with a final screen that will let you test your installation. Make sure to use a decent password for the **admin** user. Click **Next** to verify and complete the installation.

At this point, Tomcat should be running. Browse to <http://localhost:8080> to be sure. If everything is ready, you'll see the Tomcat Welcome page. Verify the Tomcat examples are available at <http://localhost:8080/examples>.

Dynamic Content servlets and JSP

Dynamic Content servlets and JSP

- What are Java Servlets
- What are the advantages of servlets over CGI
- What is JSP
- What are the benefits of JSP

Java Servlets - Introduction

- Extensions of the web server
 - Run inside the Servlet Container
 - The container can act as a web server
 - The container can interact with a web server via plugins (Apache and mod_jk for Tomcat for example)
- One of the front ends to the J2EE system
 - The servlet acts as the HTTP Receiver and calls EJBs, JMS and so on

Java Servlets – Introduction (cont.)

Java Servlets – Introduction (cont.)

- Good for implementing control logic
 - Captures that request and determines what to do
 - Initializes Beans, EJBs and other resources
 - Forward the request to a view (A Java Server Page)
- Support for sessions
 - A way around the stateless nature of the web
 - Stored on the server
 - Can contain anything
 - Private for a specific user
 - Identified with "jsessionId"
 - As a Cookie or as a request parameter

Java Servlets – Introduction (cont.)

- Multithreaded by nature
 - There is only one instance of each Servlet
 - The Servlet must be thread safe
 - Only read only Class variable
 - The Servlet can implement SingleThreadModel to force the container to only use one thread at a time
 - No concurrent access => bad performance
- Not so good for generation of nice layout

A very simple Servlet

A very simple Servlet

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class Servlet1 extends HttpServlet {

    public void init(ServletConfig config) throws
ServletException {
        super.init(config);
    }

    public void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<h1>Yo</h1>");
        out.close();
    }
}
```

Advantages of servlets over CGI

Advantages of servlets over CGI

- Efficiency
- Convenience
- More Powerful
- More Portable

Java servlets are more efficient, easier to use, more powerful, more portable, and cheaper than traditional CGI and than many alternative CGI-like technologies. (More importantly, servlet developers get paid more than Perl programmers :-).

Efficient. With traditional CGI, a new process is started for each HTTP request. If the CGI program does a relatively fast operation, the overhead of starting the process can dominate the execution time. With servlets, the Java Virtual Machine stays up, and each request is handled by a lightweight Java thread, not a heavyweight operating system process. Similarly, in traditional CGI, if there are N simultaneous request to the same CGI program, then the code for the CGI program is loaded into memory N times. With servlets, however, there are N threads but only a single copy of the servlet class. Servlets also have more alternatives than do regular CGI programs for optimizations such as caching previous computations, keeping database connections open, and the like.

Convenient. Hey, you already know Java. Why learn Perl too? Besides the convenience of being able to use a familiar language, servlets have an extensive infrastructure for automatically parsing and decoding HTML form data, reading and setting HTTP headers, handling cookies, tracking sessions, and many other such utilities.

Powerful. Java servlets let you easily do several things that are difficult or impossible with regular CGI. For one thing, servlets can talk directly to the Web server (regular CGI programs can't). This simplifies operations that need to look up images and other data stored in standard places. Servlets can also share data among each other, making useful things like database connection pools easy to implement. They can also maintain information from request to request, simplifying things like session tracking and caching of previous computations.

Portable. Servlets are written in Java and follow a well-standardized API. Consequently, servlets written for, say I-Planet Enterprise Server can run virtually unchanged on Apache, Microsoft IIS, or WebStar. Servlets are supported directly or via a plugin on almost every major Web server.

Inexpensive. There are a number of free or very inexpensive Web servers available that are good for "personal" use or low-volume Web sites. However, with the major exception of Apache, which is free, most commercial-quality Web servers are relatively expensive. Nevertheless, once you have a Web server, no matter the cost of that server, adding servlet support to it (if it doesn't come preconfigured to support servlets) is generally free or cheap

Introduction to JSP

Introduction to JSP



- Mixed content pages
- HTML with embedded Java
- Server side processing

Java Server Pages (JSP) is a technology that lets you mix regular, static HTML with dynamically-generated HTML. Many Web pages that are built by CGI programs are mostly static, with the dynamic part limited to a few small locations. But most CGI variations, including servlets, make you generate the entire page via your program, even though most of it is always the same. JSP lets you create the two parts separately

Benefits of JSP

Benefits of JSP



- Support a component model and software reuse through the use of components
 - Recompile automatically when changes are made to the source file
 - Simplify page development with JSP and custom tags
 - Ability to separate the Web content from the code
 - Platform-independent
 - Performance and scalability
 - Reliability
 - Integrate into enterprise as part of J2EE
-

JSP Vs Servlets

JSP Vs Servlets

- Recommended Uses of Servlets:
 - Extend the functionality of a Web server
 - Generate objects that do not contain HTML
 - Initialize a Web application

- Recommended Uses of Java Server Pages:
 - Access application logic separated from Web content and embedded in components
 - Present dynamic portions of content, which is tailored to a specific user.

Simple JSP example

Simple JSP example

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD><TITLE>Welcome to Our Store</TITLE></HEAD>
<BODY>
  <H1>Welcome to Our Store</H1>
  <SMALL>Welcome,
  <!-- User name is "New User" for first-time visitors -->
  <% out.println(Utils.getUserNameFromCookie(request)); %>
  To access your account settings, click
  <A HREF="Account-Settings.html">here.</A></SMALL>
  <P> Regular HTML for all the rest of the on-line store's Web page.
</BODY></HTML>
```

LAB:

LAB:

■ Simple JSP

Create a file "hello.jsp"

```
<%-- This JSP comment will not appear in the
generated html --%>
<%-- This is a JSP directive: --%>
<%@ page import="java.util.*" %>
<%-- These are declarations: --%>
<%!
    long loadTime= System.currentTimeMillis();
    Date loadDate = new Date();
    int hitCount = 0;
%>
<html><body>
<%-- The next several lines are the result of a
JSP expression inserted in the generated html;
the '=' indicates a JSP expression --%>
<H1>This page was loaded at <%= loadDate %> </H1>
<H1>Hello, world! It's <%= new Date() %></H1>
<H2>Here's an object: <%= new Object() %></H2>
<H2>This page has been up
<%= (System.currentTimeMillis()-loadTime)/1000 %>
seconds</H2>
<H3>Page has been accessed <%= ++hitCount %>
times since <%= loadDate %></H3>
<%-- A "scriptlet" that writes to the server
console and to the client page.
Note that the ';' is required: --%>
<%
    System.out.println("Goodbye");
    out.println("Cheerio");
%>
</body></html>
```

Mod_php

Mod_php



What is PHP ?

What is PHP ?



- PHP: Hypertext Preprocessor
 - PHP is a widely-used Open Source general-purpose scripting language that is especially suited for Web development and can be embedded into HTML
-

What can PHP do ?

What can PHP do ?



- Server Side Scripting
 - Command Line Scripting
 - Cross Platform Applications
 - Good Database Support
 - Faster than ASP on same hardware
-

A Simple PHP Example

A Simple PHP Example

```
<html>
  <head>
    <title>Example</title>
  </head>
  <body>
    <?php echo '<p>Hello World!</p>'; ?>
  </body>
</html>
```

Notice how this is different from a script written in other languages like Perl or C -- instead of writing a program with lots of commands to output HTML, you write an HTML script with some embedded code to do something (in this case, output some text). The PHP code is enclosed in special start and end tags that allow you to jump into and out of "PHP mode".

PHP Coding Style

PHP Coding Style

- Statements – separated by semicolon ';'
- Whitespace ignored by HTML & PHP, but used for readability
- Comments:
 - C-style: /* comments */
 - C++ style: // comments
 - Script style: # comments

PHP Dynamic Content

PHP Dynamic Content



- Call built-in functions executed at server
 - `date ()`
 - Passing data
 - `date ("H:I, jS F")`
 - H=24 hour format, i=minutes, j=day of month, S=ordinal suffix ("th"), F=four digit year
-

Accessing Form Variables

Accessing Form Variables

- Short style - PHP variable has same name in HTML form but with a '\$'
 - If register_globals directive is "on"
- Faster - retrieve from arrays stored in:
 - \$HTTP_POST
 - \$HTTP_GET_VARS

Strings, Variables & Literals

Strings, Variables & Literals

- String concatenation, use period (.)
- Variable starts with \$: \$name
- Strings – data themselves, use double quotes ("")
- Literals – raw data, you can use single quotes ('')

Identifiers

Identifiers

- Names of variables
- Any length of letters, numbers, underscores and dollar signs
- Cannot begin with a digit
- Case sensitive. PHP built-in functions are not
- Identifiers for variables can have same name as built-in function. BUT avoid it!

PHP Data Types

PHP Data Types



- Integer – use for whole numbers
 - Double – use for real numbers
 - String – use for text
 - Array – store multiple data of same type
 - Object – use for storing instances of classes
-

Type Strength

Type Strength

- A PHP variable holds only one data type
- Variable assumes type from data stored

```
$total = 0;           //integer
$sum = 0.00;         //double
$sum = "Hi";        //now string
```
- Variable can change type on-the-fly

Type Casting

Type Casting



```
$sum = 0;  
$total = (double)$sum;
```

- \$sum remains integer but value assigned to \$total is now double

Variable Variables & Constants

Variable Variables & Constants



- Change name of variable dynamically

```
$varname = "shoeqty";
```

```
$$varname = 5;
```

This is the same as \$shoeqty = 5;

- Use define function to set constants

```
define ("REGULAR", 1.50)
```

```
define ("SUPREME", 1.70)
```

- Constant name uppercase = convention from C

- Note: No dollar sign (\$) for constants

Variable Scope

Variable Scope

- Global – visible throughout script but not inside functions!
 - All variables global by default
- Variables used inside functions are local to the function
- Variables used inside functions that are declared as global refer to the global variable of the same name

Operators

Operators

- Arithmetic - + - * / %(modulus)
 - If operators applied to strings, PHP will convert to numbers
- String – period (.) for concatenation
- Assignment – equal sign (=)
- Combination - +=, -=, *=, /=, %=, .=
 - `$a += 10;` is the same as `$a = $a + 10;`

More Operators

More Operators

- Pre- and Post-Increment and Decrement

```
$a=5;  
echo ++$a;    //results in 6
```

- However:

```
$a=5;  
echo $a++;    //results in 4
```

Other Operators +

Other Operators +



- References
 - Comparison
 - Logical
 - Bitwise
 - Comma, new and ->
 - Array
 - Ternary, error suppression, execution
-

Variable Functions +

Variable Functions +



- Testing & Setting Variable Types
 - `gettype ()`
 - `settype ()`
 - Testing Variable Status
 - `isset ()`
 - `unset ()`
 - `empty ()`
 - Reinterpreting Variables
-

Conditional Control Structures

Conditional Control Structures



- if
 - code blocks – inside curly braces { }
 - indent codes
 - else
 - elseif
 - switch
-

Iterations +

Iterations +



- while
- for
- do . . while
- Breaking out of control structures
 - exit;

Installing PHP as an Apache Module

Installing PHP as an Apache Module

- **Httpd.conf:**
 - `LoadModule php5_module "c:/php/php5apache2.dll"`
 - `AddType application/x-httpd-php .php`
 - `# configure the path to php.ini`
`PHPIniDir "C:/php"`

Example: is for PHP5 installed at c:\php with Apache2

Basic Debugging

Basic Debugging

- Error reporting directives
 - *display_errors* – show errors in browser window
 - *log_errors* – send errors to the web server error log
 - *error_reporting* – sets which errors are reported
- Settings in php.ini can be overridden at runtime
- Add-on debugging tools available

