# Apache Web Server Administration

EVALUATION COPY

**International Technology Solutions, Inc.**
**Wake Forest, North Carolina**

# Welcome

## Welcome to Apache Web Server Administration

Apache Web Server Administration introduces you to the concepts and strategies necessary to use effectively use and program the Apache web server. Presented as lecture and hands-on labs, this class concentrates on the practical application of Apache server administration, including configuring secure sites, virtual hosts, and writing Apache extensions.

The text provides material for in-class discussions and may also be used as an invaluable Apache administration reference.

## Course Objectives

Apache Web Server Administration will teach you:

- basic and advanced configuration directives.
- how to effectively work with and monitor the Apache server.
- how to implement Apache modules.

After completing this course, you will be able to apply your Apache administration knowledge to configure a fully functional and robust Apache server and diagnose a variety of access and performance problems.

EVALUATION COPY

# Course Structure

This course is a three-day, lecture and lab intensive, fast track curriculum. Lectures follow the structure of the class's text, with labs and question and answer sessions woven in after each chapter.

# About International Technology Solutions

Since 1994, International Technology Solutions Inc. (ITS) has been providing training and consulting services to Fortune 500 companies such as Alcatel, Blue Cross Blue Shield NC, Cisco Systems, Duke Power, Ericsson Inc, Fujitsu, Lucent Technologies, Nortel Networks, Sprint, and many more.

Our corporate mission is to provide high-quality cost effective technology solutions that increase efficiency and productivity, resulting in a return on investment for our clients.

ITS is committed to providing superior corporate education programs and related services. Our main goal is to increase the productivity of those we educate and show our clients a return on investment.

ITS offers an entire curriculum of Linux courses for the user, programmer, or administrator. These include:

- Linux Fundamentals
- Linux bash Shell Programming
- Linux System Administration
- Linux Network Administration
- Linux and Windows Integration with Samba
- Apache Web Server Administration
- Introduction to Linux Development
- Linux Systems Programming
- Linux Kernel Programming
- Linux Device Driver Programming

For these courses, plus many more, please visit us on the Internet at http://www.itsinc-us.com/.

EVALUATION COPY

# Table of Contents


**WELCOME** 1

**WELCOME TO APACHE WEB SERVER ADMINISTRATION** 1
**COURSE OBJECTIVES** 1
**COURSE STRUCTURE** 2
**ABOUT INTERNATIONAL TECHNOLOGY SOLUTIONS** 2
**TABLE OF CONTENTS** 3

**CHAPTER 1: INTRODUCTION** 7

**CHAPTER OVERVIEW** 7
**CHAPTER OBJECTIVES** 7
**OVERVIEW** 8
APACHE'S STRENGTH WORLD-WIDE 8
APACHE'S OPERATING SYSTEMS 8
**FEATURES** 9
**COMPARISON TO OTHER SERVERS** 10
**CHAPTER SUMMARY** 11

**CHAPTER 2: APACHE INSTALLATION** 13

**CHAPTER OVERVIEW** 13
**CHAPTER OBJECTIVES** 13
**PLACING YOUR WEB SERVERS** 14
UNTRUSTED USERS 14
**OBTAINING APACHE** 15
OBTAINING APACHE 15
**COMPILING AND INSTALLING APACHE** 16
COMPILING APACHE 16
APACHE BINARY INSTALLATION 16
EXECUTABLE AND CONFIGURATION FILE LOCATIONS 17
MODULES 18
**STARTING AND TESTING APACHE** 23
STARTING THE SERVER 23
TESTING THE SERVER 24
**CHAPTER SUMMARY** 25

**CHAPTER 3: APACHE CONFIGURATION** 27

**CHAPTER OVERVIEW** 27
**CHAPTER OBJECTIVES** 27
**APACHE DIRECTIVES** 28
SIMPLE DIRECTIVES 28
BLOCK DIRECTIVES 28
DIRECTORY LEVEL CONFIGURATION 30
**SERVER CONFIGURATION** 31
SELECTING A SERVER TYPE 31
CHOOSING THE HTTP PORT NUMBER 31
HOSTNAME LOOKUPS 32


EVALUATION COPY

CHOOSING THE SERVER'S USER AND GROUP 32
SETTING THE SERVER'S MAIN DIRECTORY 33
SELECTING SERVER INFORMATION FILES 33
SETTING THE DOCUMENT CONTENT DIRECTORY 34
SPECIFYING THE DEFAULT DIRECTORY FILENAMES 34
SETTING LOCK FILES 34
DEFINING HOSTNAMES 35
CACHE CONFIGURATION 35
SELECTING CONNECTION VALUES 36
NUMBER OF SERVER PROCESSES 37
SPECIFIC ADDRESS BINDING 38
CUSTOMIZING ERROR RESPONSES 38
**USER-SPECIFIC WEB PAGES 39**
DISABLING AND ENABLING USERS 39
DIRECTORY SPECIFICATION 40
**CGI PROGRAMS 41**
**SERVER SIDE INCLUDES 41**
**CHAPTER SUMMARY 42**

## CHAPTER 4: EFFECTIVELY WORKING WITH APACHE 43

**CHAPTER INTRODUCTION 43**
**CHAPTER OBJECTIVES 43**
**CONTROLLING APACHE 44**
APACHECTL 44
SYSTEM V SCRIPT 46
APACHE COMMAND-LINE PARAMETERS 47
**WORKING WITH THE APACHE LOGS 48**
THE ERROR LOG 48
THE ACCESS LOG 49
**CHAPTER SUMMARY 52**

## CHAPTER 5: VIRTUAL HOSTS 53

**CHAPTER OVERVIEW 53**
**CHAPTER OBJECTIVES 53**
**IP ADDRESS VIRTUAL HOSTS 54**
HOW TO SET UP APACHE 54
SETTING UP MULTIPLE DAEMONS 55
SETTING UP A SINGLE DAEMON 56
**NAME-BASED VIRTUAL HOSTS 57**
**DYNAMICALLY-NAMED VIRTUAL HOSTS 58**
SETTING UP THE CONFIGURATION FILE 58
SIMPLE DYNAMIC VIRTUAL HOSTS 59
COMBINING VIRTUAL HOSTING METHODS 60
MORE EFFICIENT IP ADDRESS-BASED VIRTUAL HOSTING 61
**SYSTEM LIMITATIONS 62**
FILE DESCRIPTOR LIMITS 62
IP ADDRESS LIMITS 63
**CHAPTER SUMMARY 64**

## CHAPTER 6: ADVANCED CONFIGURATION 65

**CHAPTER OVERVIEW 65**

EVALUATION COPY

**CHAPTER OBJECTIVES** **65**
**CONDITIONAL DIRECTIVES** **66**
TESTING FOR CONDITIONS 66
TESTING FOR MODULES 67
**MODIFYING THE ENVIRONMENT** **68**
BROWSER MATCHING 68
PASSING THE ENVIRONMENT ON 69
**APACHE HANDLERS** **70**
HANDLERS 70
ASSOCIATING WITH FILES 71
CREATING HANDLERS 72
**REDIRECTING CONTENT** **73**
SIMPLE ALIASES 73
PATTERN ALIASES 73
REDIRECTS 74
**FANCY INDEXING** **75**
ASSOCIATING ICONS WITH FILES 75
ASSOCIATING DESCRIPTIONS WITH FILES 76
SPECIAL DIRECTORY FILES 76
EXCLUDING FILES 76
**DELIVERING BROWSER-SENSITIVE CONTENT** **77**
ENCODING 77
LANGUAGE 77
MEDIA TYPE 79
**CHAPTER SUMMARY** **80**

## CHAPTER 7: PERFORMANCE AND SECURITY 81

**CHAPTER OVERVIEW** **81**
**CHAPTER OBJECTIVES** **81**
**APACHE'S SECURITY AND PERFORMANCE GOALS** **82**
HARDWARE AND PLATFORM CONSIDERATIONS 82
**PERFORMANCE TUNING** **84**
RUN-TIME TUNING 84
**SECURITY** **87**
RESTRICTING ACCESS 87
SETTING ACCESS OPTIONS 88
ENABLING ACCESS TO LOCAL DOCUMENTS 90
SERVERROOT DIRECTORY PERMISSIONS 90
SAFE CGI 91
**CHAPTER SUMMARY** **92**

## CHAPTER 8: URL REWRITING 93

**CHAPTER OVERVIEW** **93**
**CHAPTER OBJECTIVES** **93**
**THE URL REWRITING ENGINE** **94**
REWRITING FUNDAMENTALS 94
**COMMON REWRITING NEEDS** **98**
TRAILING SLASHES 98
USERS ON ANOTHER SERVER 99
REDIRECT INVALID URLS 99
TIME IS IMPORTANT 100
FAKING STATIC PAGES 100
**CHAPTER SUMMARY** **101**

EVALUATION COPY

**APPENDICES** **103**

**LAB 1: INTRODUCTION** **104**
PART A (5 MINUTES) 104
**LAB 2: APACHE INSTALLATION** **105**
PART A (10 MINUTES) 105
PART B (30-45 MINUTES) 105
**LAB 3: APACHE CONFIGURATION** **107**
PART A (5 MINUTES) 107
PART B (40 MINUTES) 107
**LAB 4: EFFECTIVELY WORKING WITH APACHE** **109**
PART A (5 MINUTES) 109
PART B (15 MINUTES) 109
PART C (30 MINUTES) 109
**LAB 5: VIRTUAL HOSTS** **110**
PART A (10 MINUTES) 110
PART B (45 MINUTES) 110
PART C (15 MINUTES) 111
**LAB 6: ADVANCED CONFIGURATION** **112**
PART A (5 MINUTES) 112
PART B (15 MINUTES) 112
PART C (15 MINUTES) 112
**LAB 7: PERFORMANCE AND SECURITY** **113**
PART A (5 MINUTES) 113
PART B (45 MINUTES) 113
PART C (30 MINUTES) 114
**LAB 8: URL REWRITING AND CUMULATIVE LAB** **115**
PART A (5 MINUTES) 115
PART B (90 MINUTES) 115
CHALLENGE 1 (90 MINUTES) 115
**REFERENCES** **116**

EVALUATION COPY

# Chapter 1: Introduction

## Chapter Overview

Before using Apache, it is sensible to review the features it offers and how it compares to other servers. In this chapter, you'll see the benefits Apache gives administrators, and you'll see how Apache compares to other web servers.

## Chapter Objectives

After completing this chapter, you will be able to:

- describe the Apache web server.
- list Apache's features.
- compare Apache with other Web servers.

EVALUATION COPY

# Overview

The Apache web server began simply: to provide an open-source Web server for Linux and other open-source operating systems. Originally developed by the Apache Group, the Apache web server met that goal.

Today, Apache has grown far beyond its original scope. Currently funded by the Apache Software Foundation (`http://www.apache.org/`), the Apache web server is just one piece of a larger suite of many Internet-oriented, open-source projects.

## Apache's strength world-wide

Apache is a commercial-grade server actively designed, developed, and debugged by volunteers worldwide. Apache serves (i.e. provides the content for browsers to view) more Internet sites than any other web server on the market does. With this kind of coverage, you can imagine Apache is a strong and stable web server.

## Apache's operating systems

Apache runs on many operating systems. Frequently, Apache runs on Linux, but the Apache source code builds and runs perfectly well on:

- FreeBSD, OpenBSD, and NetBSD
- Solaris and SunOS
- HP-UX
- AIX
- IRIX
- Digital UNIX
- Windows NT/2000 and 9x
- Netware 5.x
- OS/2
- Macintosh
- BeOS
- SCO

EVALUATION COPY

# Features

There are numerous reasons to use Apache. Apache is:

- a powerful, flexible, HTTP/1.1-compliant web server.
- a modern server, implementing the latest protocols, including HTTP/1.1 (RFC2616).
- highly configurable and extensible with third-party modules.
- very customizable with 'modules' conforming to the Apache module API.
- free, provides full source code, and comes with an unrestrictive license.
- actively developed by dedicated volunteers worldwide.
- robust because it encourages user feedback through new ideas, bug reports, and patches.
- powerful as it implements:
    - DBM databases for authentication.
    - customized error messages.
    - different directory index views.
    - unlimited and flexible URL rewriting and aliasing.
    - content negotiation.
    - virtual hosts.
    - reliable logging.

EVALUATION COPY

# Comparison to Other Servers

The overwhelming majority of Internet sites use Apache. That statistic alone speaks for Apache's strength over other web servers. As The Apache Software Foundation says:

> *"Apache has been shown to be substantially faster, more stable, and more feature-full than many other web servers. Although certain commercial servers have claimed to surpass Apache's speed (it has not been demonstrated that any of these "benchmarks" are a good way of measuring WWW server speed at any rate), we feel that it is better to have a mostly-fast free server than an extremely-fast server that costs thousands of dollars. Apache is run on sites that get millions of hits per day, and they have experienced no performance difficulties."*

Independent third-party evaluations have shown that Apache excels in:

- CGI execution.
- configuration capability.
- security.

However, Apache uses an expensive process-oriented model that, for static pages and some architectures, makes it a poor performer. Fortunately, the Apache Software Foundation recognizes these performance barriers and always works to improve them.

EVALUATION COPY

## Chapter Summary

Apache is a widely used, stable, and robust Web server. After five years of development, Apache evolved a rich set of configuration and performance features that make it a top choice for high-volume web sites around the world.

Apache excels in CGI script execution and security, but lacks some performance because of its process-oriented model. Because volunteer developers worldwide care about Apache's success on a daily basis, these performance barriers are rapidly being removed in favor of better models.

EVALUATION COPY

*This page intentionally left blank*

EVALUATION COPY

# Chapter 2:
# Apache Installation

EVALUATION COPY

## Chapter Overview

Installing Apache can be very simple or extremely complex. The range of configuration possibilities that Apache offers is staggering, but the default Apache installation is sufficient for many sites. This chapter will illustrate the installation procedure and point out many of the configuration parameters you can use to change the standard behavior.

## Chapter Objectives

After completing this chapter, you will be able to:

- describe what factors influence web server placement on a network.
- install Apache from either tar or rpm archives.
- configure your system to start Apache at boot.
- test Apache's configuration.

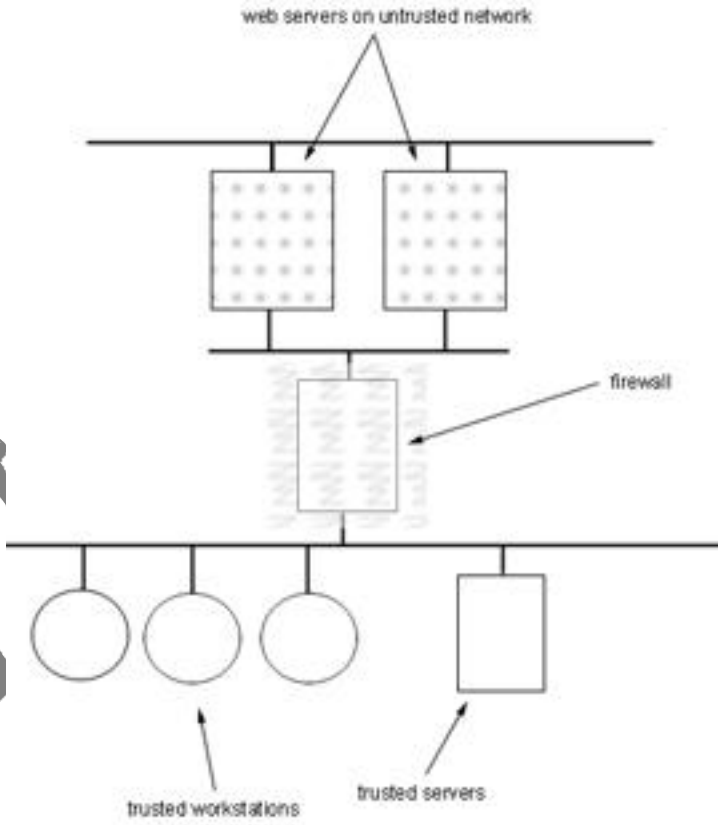
# Placing your Web Servers

Your Apache web server will provide information to a base set of users. In most cases, you will not trust the users accessing your web site, such as when you're serving pages to the Internet. In some cases, however, you will trust some (maybe all) of the users connecting to your site, such as for an Intranet.

## Untrusted users

When you will serve pages to any untrusted users, you'll need to take several precautions to prevent unauthorized access to your server.

The general architecture for sites with untrusted users is:




EVAL COPY

You should secure your web server by:

- turning off unneeded services (for example, telnet).
- ensuring that Apache is correctly setup *before* placing the server on the untrusted network.

Should a cracker defeat your security measures on one or more web servers, your firewall will prevent the damage from immediately flooding into your trusted network.

# Obtaining Apache

## Obtaining Apache

You can download Apache from the World Wide Web, or you can find it on your Linux operating system CD. For Red Hat Linux users, Apache is automatically installed with the "server" install, but you can add it manually by selecting the "Web Server" option during a custom install.

Apache's web site, `http://httpd.apache.org/`, holds the latest version for the Apache web server. This site provides the current release, more recent beta-test releases (if available), and anonymous ftp sites.

EVALUATION COPY

# Compiling and Installing Apache

Before you can use the Apache web server, you will need to install the server software. If you've downloaded the source code, you'll need to compile that; otherwise, you can simply install the server executables and configuration files.

## Compiling Apache

The Apache web site distributes the Apache source code in a compressed "tarball" format. After unpacking the archive, you must configure and build the software for your system. The example below shows the recommended procedure; it requires no intervention because the server software is highly portable:

```
$ tar -zxf apache*.gz
$ cd apache*
$ ./configure --prefix=PREFIX
$ make
$ make install
```

In this example, you supplied a compile-time configuration parameter to Apache. Specifically, the "`PREFIX`" above is a path, such as `/usr/local/bin/httpd/`, where you want the server binaries to reside; you don't have to supply this option, but you can. There are many other compile-time configuration parameters, given in the `README` file that comes with Apache distribution.

This creates a binary, `src/httpd`. You will need to copy this file to a common server directory, such as `/usr/sbin`. Also, you will need to copy the default configuration files, which end with `-dist` in the `conf/` directory, to `/etc/httpd`, removing the `-dist` during the copy.

## Apache binary installation

Your Linux distribution's CD comes with the Apache binaries conveniently packaged. You can also download these binaries from the Apache web site.

For example, on a Red Hat Linux system, the following is appropriate:

```
$ mount /mnt/cdrom
$ cd /mnt/cdrom/RedHat/RPMS
$ rpm –ivh apache*
```

The distribution will put the binary (`httpd`) and the standard configuration files in your system-specific directories.

EVALUATION COPY

## Executable and configuration file locations

The table below shows the standard Red Hat directories for Apache and its files. The paths leading to these directories vary with distribution, but the overall structure remains the same.

Although it is possible to move any of the files to other directories, it is not normally advised. There may be many other files that will have to be modified to search for a new location.

***Web site directories***

| Directory | Description |
|---|---|
| `/home/httpd` | Directory for Apache Web site files |
| `/home/httpd/html` | Web site Web files |
| `/home/httpd/cgi-bin` | CGI program files |
| `/home/httpd/html/manual` | Apache Web server manual |

***Configuration files***

| Directory | Description |
|---|---|
| `.htaccess` | Directory-based configuration files. A `.htaccess` file holds directives to control access to files within the directory in which it is located |
| `/etc/httpd/conf` | Directory for Apache Web server configuration |
| `/etc/httpd/conf/httpd.conf` | Primary apache Web server configuration file |

***Application files***

| Directory | Description |
|---|---|
| `/usr/sbin` | Location of the Apache Web server program file and utilities |
| `/usr/doc` | Apache Web server documentation |
| `/var/log/http` | Location of Apache log files |

EVALUATION COPY

## Modules

You can have particular "modules," which are simply extensions to Apache's base code, dynamically linked at run-time. These modules have already been compiled, but they're not actually part of the Apache executable. Instead, you must explicitly load them into a running server with the `LoadModule` directive, as shown below:

```
LoadModule mod_name modules/mod_name.so
```

The listing below (`httpd.conf`) shows the default modules that will be loaded. Lines starting with a "#" are comments and are ignored:

```
# LoadModule foo_module modules/mod_foo.so
#LoadModule mmap_static_module modules/mod_mmap_static.so
LoadModule vhost_alias_module modules/mod_vhost_alias.so
LoadModule env_module         modules/mod_env.so
LoadModule config_log_module  modules/mod_log_config.so
LoadModule agent_log_module   modules/mod_log_agent.so
LoadModule referer_log_module modules/mod_log_referer.so
#LoadModule mime_magic_module modules/mod_mime_magic.so
LoadModule mime_module        modules/mod_mime.so
LoadModule negotiation_module modules/mod_negotiation.so
LoadModule status_module      modules/mod_status.so
LoadModule info_module        modules/mod_info.so
LoadModule includes_module    modules/mod_include.so
LoadModule autoindex_module   modules/mod_autoindex.so
LoadModule dir_module         modules/mod_dir.so
LoadModule cgi_module         modules/mod_cgi.so
LoadModule asis_module        modules/mod_asis.so
LoadModule imap_module        modules/mod_imap.so
LoadModule action_module      modules/mod_actions.so
#LoadModule speling_module    modules/mod_speling.so
LoadModule userdir_module     modules/mod_userdir.so
LoadModule alias_module       modules/mod_alias.so
LoadModule rewrite_module     modules/mod_rewrite.so
LoadModule access_module      modules/mod_access.so
LoadModule auth_module        modules/mod_auth.so
LoadModule anon_auth_module   modules/mod_auth_anon.so
LoadModule db_auth_module     modules/mod_auth_db.so
LoadModule digest_module      modules/mod_digest.so
LoadModule proxy_module       modules/libproxy.so
#LoadModule cern_meta_module  modules/mod_cern_meta.so
LoadModule expires_module     modules/mod_expires.so
LoadModule headers_module     modules/mod_headers.so
LoadModule usertrack_module   modules/mod_usertrack.so
#LoadModule example_module    modules/mod_example.so
#LoadModule unique_id_module  modules/mod_unique_id.so
LoadModule setenvif_module    modules/mod_setenvif.so
#LoadModule bandwidth_module  modules/mod_bandwidth.so
#LoadModule put_module        modules/mod_put.so

# Extra Modules
#LoadModule perl_module        modules/libperl.so
#LoadModule php_module         modules/mod_php.so
#LoadModule php3_module        modules/libphp3.so
```

EVALUATION COPY

The server can have modules compiled in but not in use. To actually use these modules, specify them with the `AddModule` directive. The defaults, shown below, are acceptable for many sites.

```
#AddModule mod_mmap_static.c
AddModule mod_vhost_alias.c
AddModule mod_env.c
AddModule mod_log_config.c
AddModule mod_log_agent.c
AddModule mod_log_referer.c
#AddModule mod_mime_magic.c
AddModule mod_mime.c
AddModule mod_negotiation.c
AddModule mod_status.c
AddModule mod_info.c
AddModule mod_include.c
AddModule mod_autoindex.c
AddModule mod_dir.c
AddModule mod_cgi.c
AddModule mod_asis.c
AddModule mod_imap.c
AddModule mod_actions.c
#AddModule mod_speling.c
AddModule mod_userdir.c
AddModule mod_alias.c
AddModule mod_rewrite.c
AddModule mod_access.c
AddModule mod_auth.c
AddModule mod_auth_anon.c
AddModule mod_auth_db.c
AddModule mod_digest.c
AddModule mod_proxy.c
#AddModule mod_cern_meta.c
AddModule mod_expires.c
AddModule mod_headers.c
AddModule mod_usertrack.c
#AddModule mod_example.c
#AddModule mod_unique_id.c
AddModule mod_so.c
AddModule mod_setenvif.c
#AddModule mod_bandwidth.c
#AddModule mod_put.c
# Extra Modules
#AddModule mod_perl.c
#AddModule mod_php.c
#AddModule mod_php3.c
```

You should maintain synchronization between the `LoadModule` and `AddModule` sections. Specifically, if you don't need a module, comment it out in both sections.

EVALUATION COPY

### *Standard modules*

The table below describes each of the standard modules:

| Module | Description |
|---|---|
| `http_core` | One of two modules that must be statically linked, which implements the Apache's basic core |
| `mod_access` | Provides access control based on originating hostname or IP address |
| `mod_actions` | Conditionally executes CGI scripts based on the file's MIME type of the request method |
| `mod_alias` | Allows for redirection and mapping part of the physical file system into logical entities accessible through the Web server |
| `mod_asis` | Enables files to be transferred without adding any HTTP headers, such as `Status`, `Location` and `Content-Type` header fields |
| `mod_auth` | Provides access control based on username/password pairs. This authentication information is stored in plain text, although the password is encrypted using the `crpyt()` system call |
| `mod_auth_anon` | Similar to anonymous FTP, enabling predefined usernames access to authenticated areas using a valid e-mail address as a password |
| `mod_auth_db` | Provides access control based on username/password pairs. The authentication information is stored in a Berkeley DB binary database file, with encrypted passwords |
| `mod_auth_dbm` | Provides access control based on username/password pairs. The authentication information is stored in a DBM binary database file, with encrypted passwords |
| `mod_authoindex` | Implements automatically generated directory indexes |
| `mod_cern_meta` | Emulates Meta files, which contain HTTP header information, as found in the original CERN httpd |
| `mod_cgi` | Controls the execution of files that are parsed by the CGI script handler or that have a MIME type of `x-httpd-cgi` |

EVALUATION COPY

| | |
|---|---|
| mod_digest | Provides access control based on username/password pairs. The authentication is MD5-encrypted and stored in a plain text file |
| mod_dir | Set the list of filenames that may be used if no explicit filename is selected in a URL that references a directory |
| mod_env | Controls environment variables passed to CGI scripts |
| mod_example | Illustrates how the server handles module references |
| mod_expires | Implements time limits on cached documents by using the Expires HTTP header |
| mod_headers | Enables custom HTTP header's creation and generation |
| mod_imap | Control inline image map files, which have a x-httpd-imap MIME type or are parsed by the imap handler |
| mod_include | Implements Server-Side Includes (SSI), which are HTML documents that include conditional statements parsed by the server prior to being sent to a client |
| mod_info | Provides a detailed summary of the server's configuration, including a list of actively loaded modules and the current settings of every directive defined within each module |
| mod_log_agent | Enables UserAgent field logging from the incoming client requests' HTTP header |
| mod_log_config | Enables a customized format for log file information |
| mod_log_referer | Enables Referer field's logging from the incoming client requests' HTTP header |
| mod_mime | Alters the handling of documents based on predefined values or the file's MIME type |
| mod_mime_magic | Similar to the UNIX file command, this module attempts to determine the file's MIME type based on a few bytes of the file's contents |

EVALUATION COPY

| | |
|---|---|
| mod_negotiation | Provides for the conditional display of documents based upon the Content-Encoding, Content-Language, Content-Length, and Content-Type HTTP header fields |
| mod_proxy | Implements a caching proxy server |
| mod_rewrite | Provides a flexible and extensible method for redirecting client requests and mapping incoming URLs to other locations in the file system |
| mod_setenvif | Conditionally sets environment variables based on the various HTTP header fields' contents |
| mod_so | The only module other than http_core that must be statically compiled in the server, this module contains the directives necessary to implement loading dynamic shared objects |
| mod_speling | Attempt to automatically correct misspellings in requested URLs |
| mod_status | Provides activities' summary of each individual httpd server processes, including CPU and bandwidth usage levels |
| mod_userdir | Specifies locations that can contain individual users' HTML documents |
| mod_usertrack | Uses cookies to track the progress of users through a Web site |
| mod_unique_id | Attempts to assign each client request a token that is unique across all server processes on all machines within a cluster |

EVALUATION COPY

# Starting and Testing Apache

Having the server installed is not enough; you must test the server and configure your system to start Apache at boot.

## Starting the server

There are several ways to start the Apache server at boot.

### *System V style*

For Red Hat Linux, which uses a System V-style interface to start services at boot, you can configure Apache to start at boot with:

```
$ chkconfig -add httpd
```

This command presumes that the `/etc/rc.d/init.d/httpd` file exists. If you installed Apache with your distribution's recommended method (for example, an RPM with Red Hat), then this file is placed automatically. Otherwise, you'll have to retrieve it from an archive site.

Once configured to start at boot, you can start Apache without rebooting with:

```
$ /etc/rc.d/init.d/httpd start
```

### *BSD style*

With other distributions, such as Slackware, you'll need to manually add the Apache server to the system start-up scripts. For example, assume you installed the server in `/usr/sbin/httpd`, then you'd put the following at the bottom of `/etc/rc.d/rc.local`:

```
# /etc/rc.d/rc.local

/usr/sbin/httpd &
```

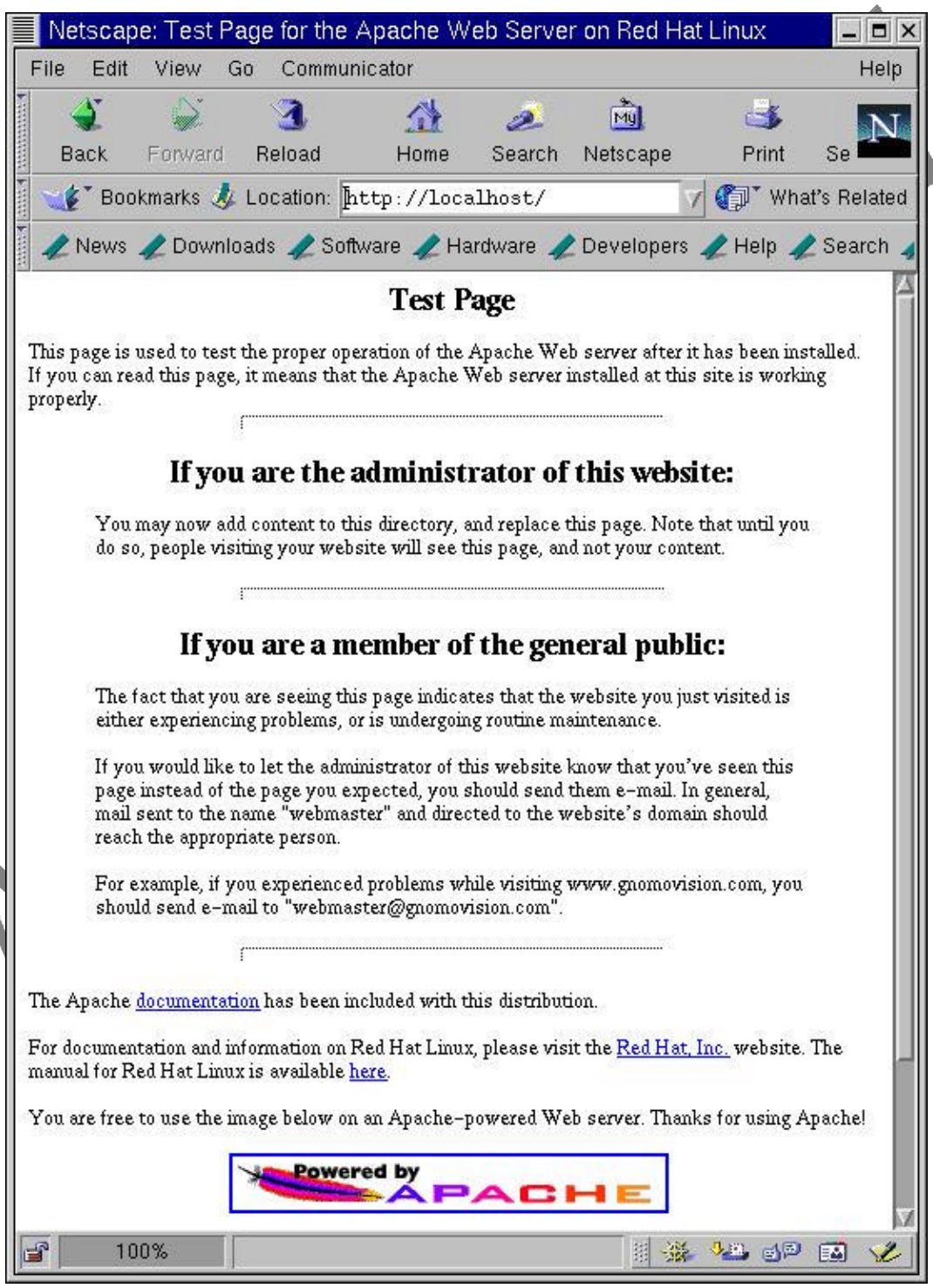
Then, you can start Apache without rebooting with:

```
$ httpd &
```

EVALUATION COPY

## Testing the server

Open a browser and load your site's main page; if the screenshot below appears, then your server is working:



## Chapter Summary

In this chapter, you learned how to obtain, compile, install, start, and test the Apache distribution. These steps only get the standard server running; additional configuration is possible through the run-time extensions provided by modules. The `LoadModule` and `AddModule` directives, held in Apache's configuration file `httpd.conf`, allow you to alter the run-time capabilities of the Apache server easily.

EVALUATION COPY

*This page intentionally left blank.*

EVALUATION COPY

# Chapter 3:
# Apache Configuration

EVALUATION COPY

## Chapter Overview

In this chapter, you will see a large collection of Apache's more popular configuration parameters, and how they affect the operation of an Apache-served web site. Understanding these parameters will allow you to tune your Apache configuration to your sites' specific requirements.

## Chapter Objectives

After completing this chapter, you will be able to:

- explain the difference between simple and block directives.
- list and describe the use of common Apache directives.
- enable CGI and SSI extensions.

# Apache Directives

The Apache configuration file, `httpd.conf`, is comprised of directives that hold the Apache configuration operations. Directives allow you to enter basic configuration information, such as your server name, or perform more complex operations, such as implementing virtual hosts.

Since all directives and most of the options are case sensitive, it is best to always use the exact format given to reduce syntax errors. A "#" at the beginning of line denotes a comment, and you may continue a directive to the next line by using a "\".

## Simple directives

Simple directives have global scope in Apache's `httpd.conf` file and take the form of the directive name followed by options. The syntax for a simple directive is:

```
Directive Option Option . . .
```

For example, to set the server administrator's email address, you would have the simple `ServerAdmin` directive set such as below:

```
ServerAdmin webmaster@company.com
```

## Block directives

Block directives hold configuration parameters that apply to specific components. Block directives are entered in pairs; specifically, there is a beginning and terminating directive.

The beginning block directive takes an argument that specifies the particular component to which the directives apply, and the terminating directive consists of a slash and the directive name designating the block's end. This syntax, which is very much like HTML containers, has the following syntax:

```
<BlockDirective Argument . . .>
      Directive Option . .
      Directive Option . .
</BlockDirective>
```

EVALUATION COPY

A couple of the more common block directives are listed below:

| **Block Directive** | **Description** |
|---|---|
| `<Directory myDir>` | Used to hold directives that apply to the specified directory |
| `<VirtualHost hostaddress>` | Used to configure a specific virtual host Web server, where `hostname` is either the IP address of the domain name |
| `<Files file(s)>` | Applies directives to one or more files |

EVALUATION COPY

## Directory level configuration

Directory configuration can be specified by either the block `Directory` directive (shown in the table above) or by placing a `.htaccess` file within the directory you wish to configure.

### *The .htaccess file*

To establish directory configuration using the `.htacess` file, simply create this file in the directory you want to configure and include all the pertinent directives.

> **TIP:**
> The `.htaccess` file inherits the configuration parameters of its parent directory and any special configuration applied in the `httpd.conf` file.

`Disabling .htaccess use`

The simple directive `AllowOverride` specifies whether per-directory overrides apply. A directory governed by an `AllowOverride None` directive will not allow `.htaccess` use, but one governed by `AllowOverride All` will.

The following example allows `.htaccess` files in the `/home/httpd/` directory (and consequently all subdirectories of `/home/httpd/`), but disables `.htaccess` files in user home directories:

```
<Directory /home/httpd>
    AllowOverride All
</Directory>

<Directory /home/*/public_html>
    AllowOverride None
</Directory>
```

> **TIP:**
> You can change the directory access control filename from `.htaccess` with the `AccessFileName` directive. For example, `AccessFileName .access` sets the filename to `.access`.

EVALUATION COPY

# Server Configuration

The `httpd.conf` file holds most of Apache's configuration, and for a typical Apache installation, many of the directives' defaults can be left as-is.

Older versions of Apache separated configuration into three files: `access.conf`, `httpd.conf`, and `srm.conf`. Apache no longer recommends this separation, and insists on keeping all configuration information within `httpd.conf`.

## Selecting a server type

Apache allows you to choose how server daemons are started to handle HTTP requests, as seen below:

```
ServerType standalone
# ServerType inetd   # not recommended
```

A `standalone` server type starts one master httpd daemon, which is then responsible for starting other daemons as necessary. Apache employs an algorithmic scheme to match the system use against the demand. For this reason, you should always set your server to `standalone`.

If you choose the `inetd` server type, then your system's inetd superserver, which all Linux systems have on by default, will start a new httpd daemon each time a HTTP request comes in. You should not use the `inetd` server type, because HTTP requests can come very rapidly and because a new daemon must be loaded and configured for each new request.

## Choosing the HTTP port number

The Internet standard HTTP port is 80, meaning that most computers on the Internet run Web servers that listen on port 80. You can alter or add other ports the Apache server listens on with the Port directive, seen below:

```
Port 80
Port 8080  # also listen on port 8080
```

You can use any number below 65535, as long as no other server is using that port. The `/etc/services` file lists the ports normally associated with particular servers, and you should check this file before randomly adding a new port.

EVALUATION COPY

## Hostname lookups

The `HostnameLookup` directive allows you to log clients by either IP address or hostname. If you enable this directive, every incoming connection will generate a DNS lookup to translate the IP address into the corresponding hostname. For example, `204.62.129.132` will be changed into `www.apache.org` before writing information into the log files.

Enabling this feature greatly reduces the server's response time, so unless you have no other way to resolve hostnames that may be required for certain analysis or statistical programs, you should leave it set to the default of `Off`:

```
HostnameLookups Off  # Set to On to enable
```

## Choosing the server's user and group

Apache doesn't have to run as the root user. Instead, you can use the `User` and `Group` directives to specify another user and group, respectively, to run the server as.

You should change the server's user and group for two reasons:

1. Running the web server as a different user allows you to separate the function of the web server (which is servicing HTTP requests) from the function of the root account (which is system maintenance).
2. Should someone discover a bug in Apache, your Apache wouldn't provide root access to your system via Apache's bug.

***The user and group method***

When the system boots, Apache starts (assuming you're using the `standalone` server type). This first server runs as `root.root` (root user and group), which is necessary in order to bind the server to port 80 and to switch to the specified user and group. Other servers started by this first server will run as the user and group you set, such as below:

```
User www
Group www
```

EVALUATION COPY

## Setting the server's main directory

The `ServerRoot` directive specifies the directory that contains the configuration files, log files, and the modules. The default for Red Hat systems, shown below, normally shouldn't be modified:

```
ServerRoot /etc/httpd
```

Should you decide to modify this directive, you must specify the parent directory that holds the configuration, log, and module files. Within this parent directory, there should be a directory named `conf` that holds configuration information, `logs` that holds log information, and `modules` that holds module files. On most systems, the `logs` and `modules` directories don't reside in the parent directory; instead, they're symbolic links to other directories in the filesystem.

## Selecting server information files

Several files hold Apache server information.

### *Process identifier (PID) file*

The `PidFile` directive identifies the file in which the server should record its process identification number. Apache uses the `PidFile` directive to store the master daemon's process ID. System maintenance scripts, such as Red Hat's `/etc/rc.d/init.d/httpd` script, use this file to find the server's ID, and these scripts might not be clever enough to check this directive to locate the file. Therefore, you should not modify this directive's default (below) without first checking your system scripts:

```
PidFile /var/run/httpd.pid
```

### *Server statistics file*

The `ScoreBoardFile` directive specifies the file that stores internal server process information. Linux doesn't require this file, but other architectures do. This file will be created if needed, so it's safe to leave the default (below) alone:

```
ScoreBoardFile /var/run/httpd.scoreboard
```

EVALUATION COPY

## Setting the document content directory

The `DocumentRoot` directory specifies the directory tree from which you will serve your documents. By default, all requests are taken from this directory, but symbolic links and aliases may be used to point to other locations:

```
DocumentRoot /home/httpd/html
```

## Specifying the default directory filenames

The `DirectoryIndex` directive specifies the filename(s) to use as a pre-written HTML directory index. Separate multiple entries with spaces:

```
DirectoryIndex index.html index.htm \
   index.shtml index.cgi
```

Apache looks for these files when a browser requests a directory and not a specific file. The first file found in the directory that matches an entry in the `DirectoryIndex` list is used. If none of the files exists and the `Indexes` option is in effect for the directory, Apache generates a directory file index; otherwise, an error message is shown.

## Setting lock files

The `LockFile` directive sets the path to the Apache's lock-file. Apache only uses this directive when compiled with either:

- `USE_FCNTL_SERIALIZED_ACCEPT`
- `USE_FLOCK_SERIALIZED_ACCEPT`

Normally, the `configure` script doesn't set these compilation flags for Linux. Unless you manually forced these compilation flags for your Apache server, you can ignore this directive. If you compiled with these flags, then the default directory is safe to leave unmodified.

```
LockFile /var/lock/httpd.lock
```

**TIP:**
The lock-file must reside on a local disk;
it can't be on a remote (e.g., NFS) filesystem.

EVALUATION COPY

## Defining hostnames

Apache can send browsers a different hostname than the one they requested.

### *Returning a different hostname*

The `ServerName` directive specifies the hostname to return to all browsers. You cannot just invent host names; you must have a valid DNS name. In the case where your server doesn't have a registered DNS name, you should set the ServerName directive to your server's IP address.

```
ServerName localhost
```

### *Canonical hostnames*

The `UseCanonicalName` directive (shown below) allows your server to enforce name consistency. When set to `On`, Apache will always use the `ServerName` and `Port` directives to create an explicit URL that uniquely refers back to your server. This name, known as the canonical name, enforces a consistent naming, which might be important for CGI scripts that validate by hostname.

```
UseCanonicalName On
```

## Cache configuration

By default, Apache sends a `Pragma: no-cache` header with each content-negotiated document. This header asks proxy servers to not cache the document, so that future requests to the document will force content renegotiation.

Un-commenting the `CacheNegotiatedDocs` directive line disables this behavior, which will allow proxies to cache documents:

```
#CacheNegotiatedDocs # uncomment to enable
```

EVALUATION COPY

## Selecting connection values

The `Timeout` directive specifies the number of seconds that Apache will hold a connection open between the receipt of a `PUT` or `POST` HTTP request, the acknowledgement of sent messages, or while receiving an incoming request. The default, shown below, can be reduced if you find an excessive number of open idle connections:

```
Timeout 300  # seconds before timeout
```

The `KeepAlive` directive instructs Apache to hold a connection open for a period of time after a request has been handled. This enables subsequent requests from the same client to be processed faster as a new connection doesn't need to be created for each request; therefore, this should be left at the default value:

```
KeepAlive On
```

The `MaxKeepAliveRequests` directive sets the maximum number of requests to allow during a persistent connection. A setting of `0` allows an unlimited amount. For maximum performance, it is recommended you leave this number high.

```
MaxKeepAliveRequests 100
```

The `KeepAliveTimeout` directive sets the number of seconds to wait for the next request from the same connection client. The time it might take a client to scan your average page and select a link from it will determine if you need to increase the 15-second default:

```
KeepAliveTimeout 15
```

EVALUATION COPY

## Number of server processes

Apache dynamically changes the number of server processes to compensate for demand. Apache samples the number of servers and load on each periodically, then algorithmically determines if more or less servers are needed.

The `MinSpareServers` and `MaxSpareServer` directives can limit the minimum and maximum number of servers. For average sites (those hit no more than 100,000 times per hour), the defaults are reasonable:

```
MinSpareServers 5
MaxSpareServers 20
```

At startup, and when operating in standalone mode, Apache will start one master server, then start more servers as given by the `StartServers` directive. Again, for average sites, the default is reasonable:

```
StartServers 8
```

Using the values specified above, when the daemon is started, the server processes will run, waiting for connections. As more requests arrive, Apache will ensure that at least 5 servers are ready to request connections. When a request has been fulfilled and no new connections arrive, Apache will begin killing processes until the number of idle Web server processes is less than 20.

### *Safety nets*

Apache can limit the total number of simultaneous server processes with the `MaxClient` directive. The `MaxClient` directive should be sufficiently high for your site's normal load. The default of 150 is almost always large enough for most sites:

```
MaxClients 150
```

The `MaxRequestsPerChild` directive sets the number of requests each child server is allowed to process before the child dies. The child will exit to avoid any problems with bugs in the Apache server or the system libraries Apache uses. Linux doesn't suffer from any known bugs, but other notable systems (such as Solaris) do, and this directive should be set for these systems:

```
MaxRequestsPerChild 100
```

EVALUATION COPY

## Specific address binding

The `Listen` directive allows you to bind Apache to specific IP addresses and, optionally, ports. The `Listen` directive is more powerful than the `Port` directive, as it allows you to specify both the IP addresses and ports you want Apache to monitor.

You will use this directive primarily when you have multiple network cards and want Apache to listen on different ports for each network card. The Port directive, or the Listen directive with just a port number, instructs Apache to listen on that port for all network cards. You can narrow that scope by supplying an IP address and port, as shown below:

```
Listen 8888  # all network interfaces use 8888
Listen 192.168.0.1:3000  # only the interface
                         # 192.168.0.1 will
                         # listen on port 3000
```

## Customizing error responses

For different error conditions that occur, you can define specific responses. The responses can be in plain text, redirected to local server pages, or external redirects.

The `ErrorDocument` directive allows you to configure specific error messages. The example below shows some customized error responses.

```
# 1) plain text
ErrorDocument 500 "The server made a boo boo.

# 2) local redirects
#    redirect to local URL /missing.html
ErrorDocument 404 /missing.html

#    redirect to a script or a
#    document using server-side-includes.
ErrorDocument 404 /cgi-bin/missing_handler.plx

# 3) external redirects
ErrorDocument 402 \
    http://www.remote.com/error.html
```

EVALUATION COPY

# User-Specific Web Pages

Apache allows you to specify which users can have their own web pages, accessible with conventional tilde (~) notation; for example, a user named "john" could access his particular user directory with the URL `http://www.company.com/~john/`.

## Disabling and enabling users

The `UserDir` directive can explicitly allow or deny username-to-path name translation for particular users by using the keyword `enabled` and `disabled`.

The keyword `disabled` without a user listing will turn off all username-to-path translations *except* those explicitly named with the `enabled` keyword. The following directive will turn off all translations, requiring you to specifically enable the users who should have access:

```
UserDir disabled
```

If you use the `disabled` keyword followed by a space-delimited username list, those listed usernames will never have directory translation performed, even if they appear in an `enabled` clause.

For example, the following directive will completely disable the root user from access, which should be done to avoid publishing data that shouldn't be made public:

```
UserDir disabled root
```

If you have disabled all users, you can use the `enabled` keyword followed by a space-delimited username list to allow these users access. These usernames will have directory translation performed even if a global disable is in effect, but *not* if they also appear in a `disabled` clause.

The following directive disables all users except "john":

```
UserDir disabled
UserDir enabled john mike
UserDir disabled mike
```

EVALUATION COPY

## Directory specification

If neither the `enabled` nor the `disabled` keyword appears in the `UserDir` directive, the argument is treated as a filename pattern. This filename specifies the directory within a user's home directory to find web content.

There are two ways that the `UserDir` directive can handle incoming request that include a tilde expansion:

1. Identify the physical pathname of the individual user's publicly accessible directories.
2. Specify a URL to which the request is redirected.

***Example***

Suppose a browser requests the URL:
`http://www.company.com/~john/`

The `UserDir` directive affects how this URL is expanded, as shown in the following table[1]:

| **Directive** | **Location** |
|---|---|
| `UserDir www` | `/home/john/www/` |
| `UserDir /usr/web` | `/usr/web/john/` |
| `UserDir /home/*/www` | `/home/john/www/` |
| `UserDir http://www.home.com/` | `http://www.home.com/john/` |
| `UserDir http://www.home.com/users/` | `http://www.home.com/users/john/` |
| `UserDir http://www.home.com/~*/` | `http://www.home.com/~john/` |

[1] The table assumes that user directories exist under `/home` in the local filesystem.

EVALUATION COPY

# CGI Programs

Common Gateway Interface (CGI) files are programs that browsers can request the server to execute.

***CGI by directory***

Traditionally, these files were placed in the `cgi-bin` directory and could only be executed if they resided in that special directory. Typically, a Web site will only have one CGI directory.

Red Hat Linux sets the CGI directory, by default, to `/home/httpd/cgi-bin`. You can set the `ScriptAlias` directive to alter this default, as shown below:

```
ScriptAlias /cgi-bin/ /home/httpd/cgi-bin/
```

***CGI by file***

It is also possible to configure Apache to consider any files ending in a particular extension as CGI programs. The `AddHandler` directive allows you to map a filename extension to some behavior within Apache. For example, the directive below maps all files that end in `.cgi` as CGI programs:

```
AddHandler cgi-script .cgi
```

# Server Side Includes

Server Side Includes (SSI) provide refined web page control. Pages that use SSI can easily and dynamically alter their content by including a few simple lines. When Apache serves a SSI page, Apache will replace the SSI commands with the appropriate data.

To use SSI, you will need to associate the parsing behavior of Apache with filename extensions, likewhat was done for CGI by file:

```
AddHandler server-parsed .shtml
```

Additionally, you'll need to instruct Apache that `.shtml` extensions are still HTML files, as in:

```
AddType text/html .shtml
```

EVALUATION COPY

## Chapter Summary

Configuring Apache to meet your site's specific requirements is a critical piece of a high-quality web site. In addition to understanding the syntax of the Apache configuration file, `httpd.conf`, you'll need to understand how the directives affect Apache's behavior. Of key importance to many administrators is Apache's performance and security features, and to adequately address these issues, an administrator must understand the directives available in the Apache configuration file.

EVALUATION COPY

# Chapter 4: Effectively Working with Apache

EVALUATION COPY

## Chapter Introduction

When you installed Apache, you configured it to start at system boot. Though this is the usual way of starting Apache, you might encounter situations where you need to restart or even stop Apache. At other times, you might need to start Apache with a different set of start-up flags. Once you've started Apache, you'll need to routinely monitor the error and access logs for odd behavior.

This chapter will explain the various ways to start Apache, the meanings of Apache's command-line flags, and how to examine the Apache logs.

## Chapter Objectives

After completing this chapter, you'll be able to:

- use the `apachectl` script to control Apache.
- use the System V style script `httpd` to control Apache.
- list and explain Apache's command-line parameters.
- describe Apache's logs and how to read them.

# Controlling Apache

Normally, you'll configure Apache to start at system boot and run until the system is shut down. However, if you are testing or modifying Apache's configuration, you will probably want to stop, start, or restart Apache without rebooting the system.

There are a couple ways to control Apache, including the command-line approach using the `apachectl` command or using the System V script.

## apachectl

Apache (post version 1.3) comes with a command to control the Apache server. In the source distribution, this file is found in `src/support/apachectl`, but binary distributions will install the file in `/usr/sbin/apachectl`.

### *Configuring apachectl*

At the top of the `apachectl` script is a configuration section, shown below:

```
# the path to your PID file
PIDFILE=/usr/local/apache/logs/httpd.pid
#
# the path to your httpd binary, including
# options if necessary
HTTPD='/usr/local/apache/src/httpd'
#
# a command that outputs a formatted text
# version of the HTML at the url given on the
# command-line.  Designed for lynx, however
# other programs may work.
LYNX="lynx -dump"
#
# the URL to your server's mod_status status
# page.  If you do not have one, then status
# and fullstatus will not work.
STATUSURL="http://localhost/server-status"
```

If you built Apache from the source code and modified the default Apache installation directories, you'll need to update this configuration section to reflect your changes.

EVALUATION COPY

### *Using apachectl*

The apachectl script accepts one of several parameters that control Apache's behavior. The table below summarizes the parameters:

| Parameter | Description |
|---|---|
| START | Starts the Apache server as given by the HTTPD configuration variable. If you need to pass any command-line flags to Apache, put those in the HTTPD configuration variable |
| stop | Stop the Apache server |
| restart | Start the server, if it's not running. Otherwise, check the Apache server's configuration file for syntax errors and then send a HUP signal to the Apache server |
| graceful | The same as restart, except send the USR1 signal. Apache closes all connections gracefully when it receives the USR1 signal; with the HUP signal, it brutally closes all connections |
| status | Use the browser given by the LYNX variable to retrieve the server status information at the STATUSURL location, and then print only server process information |
| fullstatus | Same as status, but show all the server's information |
| configtest | Test Apache's configuration file for syntax errors |

For example, to restart Apache, you would type `apachectl restart`.

EVALUATION COPY

## System V script

Some systems, such as Red Hat Linux, provide an Apache System V-like control script at `/etc/rc.d/init.d/httpd`. This script is similar to the Apache control script, though not as configurable.

The following table describes the five parameters that the `/etc/rc.d/init.d/httpd` script accepts:

| Parameter | Description |
|---|---|
| START | Start the Apache server. The Red Hat Linux version turns off core dumps, which will prevent you from performing adequate debugging should Apache have a major startup problem |
| `stop` | Stop the Apache server by sending it a TERM signal |
| `restart` | Simply executes a `stop` and then a `start` |
| `reload` | Send the HUP signal to the server, causing it to reload its configuration file and restart all connections |
| `status` | Report the process ID for all Apache servers |

EVALUATION COPY

## Apache command-line parameters

The Apache server binary, `httpd`, accepts several command-line options, explained in the table below:

| Option | Description |
|---|---|
| `-C DIRECTIVE` | Read the configuration files and then process the directive. This may supersede a definition for the directive within the configuration files |
| `-C directive` | Process the directive and then read the configuration files. The directive may alter the evaluation of the configuration file, but it may also be superseded by another definition within the configuration file |
| `-d directory` | Use "directory" as the `ServerRoot` directive, overriding the configuration file's specification |
| `-D parameter` | Define "parameter" to be used for conditional evaluation within the `IfDefine` directive |
| `-f file` | Use "file" as the Apache configuration file, rather than the default |
| `-h` | Display a list of possible command-line arguments |
| `-l` | List the modules linked into the executable at compile-time |
| `-L` | Print a verbose list of directives that can be used in the configuration files, along with a short description and the module that contains each directive |
| `-S` | List the configured setting for virtual hosts |
| `-t` | Perform a syntax check on the configuration file |

EVALUATION COPY

# Working with the Apache Logs

By default, Apache stores its log files in a directory called "logs" in the `ServerRoot` directory. For example, the Red Hat Linux default server root is `/etc/httpd`, so the log directory is `/etc/httpd/logs`. For Red Hat Linux, and for many other distributions, the `logs` directory in the server root is actually a symbolic link to another location; commonly, the log files are actually held in `/var/log/httpd/`.

Within the logs directory, Apache usually keeps two logs:

- `error_log`, which holds any errors the server generates.
- `access_log`, which holds browser connection information, such as browser IP address and version.

## The error log

When you look at the `error_log` file, you'll see a format similar to:

```
[Fri Dec  8 18:08:07 2000] [notice] Apache/1.3.12
(Unix)  (Red Hat/Linux) mod_perl/1.21 configured --
resuming normal operations
```

The first information, held within the brackets (`[]`), is the date and time of the error, as reported by the system clock. The second information, also within brackets, shows the severity of the error. The remainder is error specific, but usually provides clues as to the error's nature.

***Example error***

Often times, administrators will see the following error:

```
[Fri Jun 16 09:54:37 2000] [error] [client
192.168.0.1] File does not exist:
/home/httpd/htdocs/favicon.ico
```

In this error, Apache is complaining that the "favicon.ico" file doesn't exist. Many sites don't have a "favicon.ico" file, so administrators will wonder if someone's trying to hack their site.

This error is actually benign. When an Internet Explorer (version 4.0 or higher) user sets a bookmark on a page, Internet Explorer tries to associate a "favorite icon" with the bookmark. Internet Explorer looks for a file called "favicon.ico" in the same directory as the bookmark, and if it finds the file, puts that image in the Internet Explorer menu.

You can use this "error" to track how often your page is bookmarked, which is a good statistic to have if you need to demonstrate a server's popularity.

EVALUATION COPY

## The access log

The `access_log` file has a different format than the `error_log` file. The example below illustrates a typical entry from the `access_log` file:

```
192.168.0.1 - - [12/Jun/2000:08:19:22 -0400] "GET
/graphics/tpixel.gif HTTP/1.1" 200 61
```

***Formats***

The access log, and in fact all logs within Apache, are governed by a *format*. The format specifies what each entry in the log file should look like. For example, the format might state if the log entryshould contain the timestamp, and if so, where should it be placed relative to the other information.

When you configure Apache, you can specify a different log format with the `LogFormat` directive. The LogFormat directive has the following syntax:

```
LogFormat format handle
```

EVALUATION COPY

The format is a string, enclosed in double quotation marks ("), which is built from special format specification characters. The table below shows the defined specification characters:

| Format Character | Description |
|---|---|
| %b | Bytes sent, excluding HTTP headers |
| %f | The log filename |
| %{Var}e | The contents of the environment variable VAR |
| %h | The remote host name |
| %{Head}i | The contents of the "Head" header line(s) in the HTTP request |
| %l | The remote login name, obtained from identd, if available |
| %{Head}o | The contents of the "Head" header line(s) in the HTTP reply |
| %p | The port the request was served to |
| %P | The Apache server PID that serviced the request |
| %r | First line of request |
| %s | HTTP status information |
| %t | Time, in common log format |
| %T | The time taken to serve the request, in seconds |
| %u | The remote user, obtained from auth |
| %U | The URL path requested |
| %v | The name of the server (i.e. the virtual host) |

The "handle" parameter specifies a name to associate the format with. That name can then be used in place of the entire format string. For example, the standard log format, given a handle of "common", is declared as:

```
LogFormat "%h %l %u %t \"%r\" %>s %b" common
```

EVALUATION COPY

`Multiple logs`

The `CustomLog` directive allows you bind a log filename with a format that applies to the log file. For example, `CustomLog logs/standard_log common` would log information to `standard_log` using the "common" format seen above.

Whenever log information is available, Apache scans the custom log formats. If any of the formats contain the information that's available, those entries are written immediately. If some of the information is available, but not all, Apache writes as much as possible, filling in the non-available fields with a hyphen (`-`).

EVALUATION COPY

## Chapter Summary

Occasionally, you'll find need to stop or restart the Apache server; perhaps for diagnostic purposes or configuration changes. Rather than rebooting your entire system to restart Apache, you can use the Apache-supplied `apachectl` script or a script provided by your operating system. These scripts make it easy for you to control and retrieve status information on your Apache server.

Commonly, though, you'll look through Apache's logs. Monitoring security and access statistics are vital for a healthy server, so understanding the Apache log files is a necessary administrative duty.

Apache allows you to specify a custom log format with the `CustomLog` and `LogFormat` directives. Setting these allows you to fine-tune your logs to meet your precise requirements.

EVALUATION COPY

# Chapter 5: Virtual Hosts

## Chapter Overview

Virtual hosting refers to maintaining more than one server on a machine, differentiated by host name or IP address. For example, companies sharing a web server want to have their own domains and allow web server accessibility by `www.company1.com` and `www.company2.com`, without requiring any extra path information from the user. Apache supports several types of virtual hosting: IP address-based, name-based, and dynamically-named.

## Chapter Objectives

After completing this chapter, you will be able to:

- implement IP address-based virtual hosts.
- implement name-based virtual hosts.
- implement dynamically-named virtual hosts.
- describe limitations with virtual hosts and appropriate remedies.

EVALUATION COPY

# IP Address Virtual Hosts

When using the IP address method, each host must have its own valid IP address and your machine must be set up to support multiple IP addresses. Typically, you'll have multiple, physical network connections, but you can also configure a single network connection to listen for several IP addresses.

You must have a separate daemon running for each virtual host that separately listens for an IP address or a single daemon running that listens for requests on all virtual hosts.

## How to set up Apache

Supporting multiple hosts can be configured in two ways:

- running a *separate* Apache server for each hostname.
- running a *single* server that supports all the virtual hosts.

***Using separate servers***

You will want to use separate servers when:

- you want to divide administration between sites to several administrators, including the Apache server management.
- you can afford the memory and file descriptor requirements of listening to all the machine's IP aliases.

***Using a single server***

You will want to use a single server when:

- sharing the `httpd` configuration between virtual hosts is acceptable.
- the machine services a large number of requests, and running separate daemons may result in significant performance loss.

EVALUATION COPY

## Setting up multiple daemons

Each server will need its own configuration file that specifies specific `User`, `Group`, `Listen`, `DocumentRoot`, and `ServerRoot` directives. The `Listen` directive will specify which IP address the server will listen on.

**TIP:**
Because you're specifying configuration parameters for two separate Apache servers, all the directives are available. You will need to tailor these appropriately for each of the individual sites.

For example, suppose your Linux system hosts two web sites:

- `www.company1.com`, with an IP address of 192.168.0.1
- `www.company2.com`, with an IP address of 192.168.0.2

Then, the configuration file for `www.company1.com` would look like:

```
# httpd configuration for www.company1.com
User www
Group company1
Listen 192.168.0.1:80
ServerRoot /etc/httpd/company1/
DocumentRoot /home/httpd/htdocs/company1/
```

The configuration file for `www.company2.com` would look like:

```
# http configuration for www.company2.com
User www
Group company2
Listen 192.168.0.2:80
ServerRoot /etc/httpd/company2/
DocumentRoot /home/httpd/htdocs/company2/
```

At system boot, start an http server using the configuration file for company1, and an http server using the configuration file for company2 and you've achieved IP address virtual hosting.

EVALUATION COPY

## Setting up a single daemon

To set up a single server to manage all virtual hosts, use the `VirtualHost` block directive. Within the `VirtualHost` directive, specify the parameters for that particular host. These should include `ServerAdmin`, `ServerName`, `DocumentRoot`, and `TransferLog` directives.

**TIP:**
You can place all of Apache's directives within a `VirtualHost` block except for: `ServerType`, `StartServers`, `MaxSpareServers`, `MinSpareServers`, `MaxRequestsPerChild`, `BindAddress`, `Listen`, `PidFile`, `TypesConfig`, `ServerRoot`, and `NameVirtualHost`.

For example, suppose your Linux system hosts two web sites:

- `www.company1.com`, with an IP address of 192.168.0.1
- `www.company2.com`, with an IP address of 192.168.0.2

You can set these up with a single Apache server with IP address-based virtual hosts with:

```
<VirtualHost 192.168.0.1>
    ServerName www.company1.com
    User www
    Group company1
    DocumentRoot /home/httpd/htdocs/company1/
    ErrorLog company1/logs/error_log
    CustomLog company1/logs/access_log common
</VirtualHost>

<VirtualHost 192.168.0.2>
    ServerName www.company2.com
    User www
    Group company2
    DocumentRoot /home/httpd/htdocs/company2/
    ErrorLog company2/logs/error_log
    CustomLog company2/logs/access_log common
</VirtualHost>
```

**TIP:**
Though you could specify the DNS name instead of the IP address in the `VirtualHost` block, doing so isn't recommended. Apache has to perform a DNS lookup before allowing access, which slows down response time.

EVALUATION COPY

# Name-Based Virtual Hosts

IP address-based virtual hosting imposes a limit on the number of sites your system can support; you can only support a limited number of separate, physical network connections. However, name-based virtual hosting allows an unlimited number of virtual hosts without additional IP addresses.

You'll also use the `VirtualHost` directive to specify a name-based virtual host, but the additional `NameVirtualHost` directive binds a particular IP address to the hosts you want to service.

The `VirtualHost` directives each take the same IP address specified in the `NameVirtualHost` directive as its argument. Use the Apace directives within the `VirtualHosts` block to configure each host separately. Name-based virtual hosting uses the header address to determine the virtual host to use. If no such information exists, the first host is used as the default. The following example implements two name-based virtual hosts: `maple` and `elm`.

For example, suppose your Linux system hosts two web sites `www.company1.com` and `www.company2.com`, and the system has a single IP address of 192.168.0.1. The configuration below would set up these two sites:

```
NameVirtualHost 192.168.0.1

<VirtualHost 192.168.0.1>
    ServerName www.company1.com
    User www
    Group company1
    DocumentRoot /home/httpd/htdocs/company1/
    ErrorLog logs/error_log.company1
    CustomLog logs/access_log.company1 common
</VirtualHost>

<VirtualHost 192.168.0.1>
    ServerName www.company2.com
    User www
    Group company2
    DocumentRoot /home/httpd/htdocs/company2/
    ErrorLog logs/error_log.company2
    CustomLog logs/access_log.company2 common
</VirtualHost>
```

**TIP:**

Apache looks up the server to access from the HTTP headers. If this information isn't available (such as with very old browsers), Apache will use the first defined virtual host.

EVALUATION COPY

# Dynamically-Named Virtual Hosts

If your `httpd.conf` contains many `VirtualHost` block directives that are similar, you will want to use dynamically-named virtual hosts. The basic idea is replacing all static `VirtualHost` block directive configurations with a dynamic mechanism.

This method has a number of advantages including:

1. Apache starts faster and uses less memory, since your configuration file is smaller.
2. Adding virtual hosts is simply a matter of creating the appropriate directories and DNS entries and doesn't require reconfiguring or restarting Apache.

Apache's virtual host mechanism works by binding the IP address the browser connects to and the contents of the HTTP request's `Host:` header. This behavior is built directly into Apache. However, the dynamically-named virtual hosting method uses the `mod_vhost_alias` module, which obviously must be included as part of a `LoadModule` directive.

## Setting up the configuration file

To use dynamically-named virtual hosts, you'll need to set the following directives appropriately:

- `ServerName` must reflect your server's actual DNS name. Apache will use the defined `ServerName` should a dynamically-named host fail to find a real host name.
- `UseCanonicalName` must be set to either `Off` or `DNS`. If it is set to `Off`, then Apache uses the server name in the HTTP request's `Host:` header. If it is set to `DNS`, then Apache looks up the IP address the browser connected to and finds the host name. In the event that Apache can't find the server name, it will use the value given by `ServerName`.
- `DocumentRoot` and `ScriptAlias` should not be set *unless* you want these to apply to *all* hosts. Dynamically-named virtual hosts use a different syntax.

EVALUATION COPY

## Simple dynamic virtual hosts

The example below implements dynamically-named virtual hosts, relying on the contents of the HTTP request's `Host:` header:

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

# the first field, %V, holds the virtual host
# Apache uses.  Notice the use of the vcommon
# handle on the end
LogFormat "%V %h %l %u %t \"%r\" %s %b" vcommon
CustomLog logs/access_log vcommon

# include the virtual host name in the paths
# (notice the %0)
VirtualDocumentRoot /home/httpd/htdocs/%0/
VirtualScriptAlias  /home/httpd/%0/cgi-bin/
```

EVALUATION COPY

## Combining virtual hosting methods

You can combine the virtual hosting provided by the `VirtualHost` directive with that provided by the `VirtualScriptAlias` and `VirtualDocumentRoot` directives. This allows you to have path name expansion bound to a particular IP or host name.

For example, suppose you have two network cards in your web server. One (192.168.0.1) is connected to a high bandwidth backbone, and the other (192.168.0.2) is connected to a slower network. You want all your corporate clients on the backbone, and all your personal web sites on the slower network. You could configure this easily with the following configuration:

```
# get the server name from the Host: header
# and use logging that contains the virtual
# host name
UseCanonicalName Off
LogFormat "%V %h %l %u %t \"%r\" %s %b" vcommon

# configure directory permissions for corporate
# and personal web spaces
<Directory /home/httpd/htdocs/corp/>
    Options FollowSymLinks
    AllowOverride All
</Directory>

<Directory /home/httpd/htdocs/pers/>
    Options FollowSymLinks
    AllowOverride None
</Directory>

<VirtualHost 192.168.0.1>
    ServerName www.corp.isp.com
    CustomLog logs/corp/access_log vcommon
    VirtualDocumentRoot /home/httpd/htdocs/corp/%0
    VirtualScriptAlias  /home/httpd/cgi-bin/%0
</VirtualHost>

<VirtualHost 192.168.0.2>
    ServerName www.hom.isp.com
    CustomLog logs/access_log.hom vcommon
    VirtualDocumentRoot /home/httpd/htdocs/pers/%0
    ScriptAlias /cgi-bin/ /home/httpd/cgi-bin/
</VirtualHost>
```

EVALUATION COPY

## More efficient IP address-based virtual hosting

When Apache expands the `%0` variable, it's actually filling in the host name the browser wants. This requires Apache to perform a DNS lookup, which can take some time, especially if the network is down.

Generally speaking, Apache doesn't need to worry about the host name. If you're using IP address-based virtual hosting, which implies every host has a separate IP address, then you can ignore the lookup step and simply index by IP address, as shown below:

```
UseCanonicalName Off

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

# include the IP address in the filenames
VirtualDocumentRootIP /home/httpd/htdocs/%0/
VirtualScriptAliasIP  /home/httpd/cgi-bin/%0/
```

EVALUATION COPY

# System Limitations

## File Descriptor Limits

When using a large number of virtual hosts, Apache may run out of available file descriptors if each `VirtualHost` block specifies different log files. The total number of file descriptors used by Apache is one for each distinct error log file, one for every other log file directive, plus 10 or 20 for internal use.

Most multi-tasking, multi-user operating systems, including Linux, limit the number of file descriptors that a process may use. The limit is typically 64, and usually may be increased up to a large hard limit.

Although Apache attempts to increase the limit as required, this may not work if:

1. Your system does not provide the `setrlimit()` system call.
2. The `setrlimit(RLIMIT_NOFILE)` call does not function on your system.
3. The number of file descriptors required exceeds the hard limit.
4. Your system imposes other file descriptor limits, such as a limit on `stdio` streams only using file descriptors below 256.

In the event of problems you can:

- reduce the number of log files by not specifying log files in the `VirtualHost` blocks, but only server-wide.
- increase the file descriptor limit (if your system falls under 1 or 2 above) before starting Apache, using a script like:

```
#!/bin/sh
ulimit -S -n 100
exec /usr/sbin/httpd
```

EVALUATION COPY

## IP address limits

If your system has only one IP address, then implementing virtual hosts prevents access to your main server using that address. You can no longer use your main server as a Web server directly, only indirectly to manage your virtual hosts.

You could configure a virtual host to manage your main server's Web pages. Then you could use your main server to support virtual hosts that function as Web sites, rather than the main server operating as one site directly.

If your machine has two or more IP addresses, one can be used for the main server and the other for the virtual hosts. Mixing IP-based and name-base virtual hosts is also allowed and so is using separate IP addresses to support different virtual hosts' sets.

Several domain addresses can access the same virtual host by placing a `ServerAlias` directive listing the domain names within the selected `VirtualHost` block:

```
ServerAlias www.company1.com www.alias.com
```

Requests sent to your virtual hosts' IP address have to match a configured virtual domain name. Requests not matching one of these can be caught by setting up a default virtual host using `__default:*`, causing unmatched requests to be handled by the default virtual host.

```
<VirtualHost __default:*>
```

EVALUATION COPY

## Chapter Summary

Virtual hosting provides a method for maintaining more than one server on a computer by differentiating between servers by host name. The virtual hosting method you choose depends on your system's and user's needs. With several IP addresses, virtual hosting by IP address is efficient and sensible.

With a single IP address, however, it makes sense to use name-based virtual hosting. Finally, if you have a large number of hosts or would like to repeat additional performance benefits, dynamically-named virtual hosts are the best solution.

EVALUATION COPY

# Chapter 6:
# Advanced Configuration

## Chapter Overview

Apache supports an extensive set of configuration directives. We have previously only touched on the major ones. In this chapter, you'll see that Apache can have conditional configuration, attach handlers to particular types of files, and change how it renders information.

## Chapter Objectives

After completing this chapter, you will be able to:

- use conditional directives to alter Apache's configuration.
- test and set Apache environment variables.
- recognize and associate handlers with files.
- redirect content.
- enable and modify Apache's fancy indexing.
- configure Apache's content negotiation.

EVALUATION COPY

# Conditional Directives

Apache provides two block directives, `IfDefine` and `IfModule`, that allow you to alter Apache's configuration conditionally. These directives let you section off configuration that should only be included when special conditions exist.

## Testing for conditions

The `IfDefine` block directive, shown below, alters Apache's configuration behavior:

```
# log tracking data if in paranoid mode
<IfDefine PARANOID>
    LogFormat "[%t][%a.%i]%H%s %f" paranoid
    CustomLog logs/paranoid_log paranoid
</IfDefine>
```

The configuration between the `<IfDefine>` and `</IfDefine>` is included *only if* you define the parameter (`PARANOID`, in the example) when you start Apache.

To define the parameter, use Apache's `-D` command-line flag:

```
$ httpd -DPARANOID &
```

**TIP:**
Parameter names are case-sensitive.

***Reversing the condition***

If you want to include configuration when a conditional is *not* defined, you can still use `IfDefine`. Simply prefix the parameter name with an exclamation mark, as shown below:

```
# include proxying only when not debugging the
# server
<IfDefine !DEBUG>
  LoadModule rewrite_module modules/mod_rewrite.so
  LoadModule proxy_module   modules/libproxy.so
</IfDefine>
```

**TIP:**
You can nest `IfDefine` directives for simple multi-parameter tests.

EVALUATION COPY

## Testing for modules

You can test a module's presence with the `IfModule` block directive. This directive is syntactically similar to that of `IfDefine`, as shown below:

```
<IfDefine USEIMAP>
    LoadModule imap_module modules/mod_imap.so
</IfDefine>

# if the imagemap module is loaded, then
# configure Apache's imagemap handling
<IfModule mod_imap.c>
    # imagemaps end with .map
    AddHandler imap-file map
    # display a menu instead of default action
    ImapMenu formatted
</IfModule>
```

The `IfModule` directive expects the parameter to be the module's source code name, so the parameter will usually end in `.c`. As with `IfDefine`, placing an exclamation point (`!`) in front of the module name reverses the condition.

EVALUATION COPY

# Modifying the Environment

Apache, with the `SetEnvIf` directive, has the ability to scan browsers' HTTP requests for certain patterns and set an environment variable if the pattern is found. The `SetEnvIf` directive has the following syntax:

```
SetEnvIf attr regex variable[=value]
```

The attribute, "attr" can be:

- `Remote_Host`, which is the client's hostname (if available).
- `Remote_Addr`, which is the client's IP address.
- `Remote_User`, which is the authenticated username (if available).
- `Request_Method`, which is the retrieval method's name (e.g., "GET" or "POST").
- `Request_Protocol`, which is the name and version of the protocol (e.g., "HTTP/1.1").
- `Request_URI`, which is the URL following the protocol and host specification.
- Any header sent in the request, including `User-Agent`.

You can use these environment variables either to modify Apache's behavior or pass them along to the scripts. For example, to detect the kind of script a client requests, you could include:

```
SetEnvIf Request_URI "\.pl$" script="perl"
SetEnvIf Request_URI ".\sh$" script="shell"
SetEnvIf Request_URI "\.cgi$" script="generic"
```

**TIP:**
`SetEnvIf` is case-sensitive; `SetEnvIfNoCase` is not.

## Browser matching

A special case of the `SetEnvIf` directive is the `BrowserMatch` (and `BrowserMatchNoCase`) directive. This directive only checks the browser's type, so you can use this as a quick way to set environment variables describing the client's browser:

```
# unset the javascript variable if the client's
# Internet Explorer (IE uses jscript)
BrowserMatch MSIE !javascript
```

EVALUATION COPY

## Passing the environment on

Though Apache might use the environment variables, you can arrange to have Apache pass the environment variables set with `SetEnvIf` and `SetEnvIfNoCase` to all called CGI scripts.

The `PassEnv` directive passes one or more environment variables on to all CGI scripts:

```
# pass the javascript and shell environment
# variables down to all CGI scripts
PassEnv javascript shell
```

EVALUATION COPY

# Apache Handlers

Browsers instruct Apache to load files via URLs. Most often, these files are simply HTML files that should simply be sent back to the browser. Sometimes, however, the file is more complicated than a simple text file. For example, Apache needs to execute CGI scripts and send the *results* back to the browser; sending the CGI script itself could cause a security compromise.

## Handlers

Many handlers are compiled into Apache or are available in a module. The table below lists the handlers available either by Apache directly or through a module:

| Handler | Description | Module |
|---|---|---|
| `default-handler` | Simply send the file as-is to the browser, adding HTTP headers | `core` |
| `send-as-is` | Send file as-is | `mod_asis` |
| `cgi-script` | Treat the file as a CGI script | `mod_cgi` |
| `imap-file` | Treat the file as an image map definition file | `mod_imap` |
| `server-info` | Treat the file as server information | `mod_info` |
| `server-parsed` | Scan the file for server-side includes, replacing the includes as necessary | `mod_include` |
| `server-status` | Treat the file as a server status file | `mod-status` |
| `type-map` | Treat the file as a content negotiation map file | `mod_negotiation` |

EVALUATION COPY

## Associating with files

Apache allows you to define a "handler" for:

- files with certain extensions.
- files that reside in a certain location.

### *By file extension*

You can add a handler based on a file's extension with the `AddHandler` directive:

```
AddHandler cgi-script .cgi
```

**TIP:**
You can specify more than one extension and you do not need the leading dot with the `AddHandler` directive. For example, `AddHandler cgi-script .cgi pl` causes Apache to treat all files ending in ".cgi" and ".pl" as CGI scripts.

### *By file location*

You can instruct Apache to use the same handler for all files in a certain location with the `SetHandler` directive:

```
# all files in the users' cgi-bin directories
# are treated as CGI files
<Directory /home/*/public_html/cgi-bin/>
    SetHandler cgi-script
</Directory>

# the /status file holds server status
<Location http://www.company.com/status>
    SetHandler server-status
</Location>
```

When used within the `Directory` block directive, `SetHandler` applies to all files that match the directory specification. When used inside of a `Location` block directive, `SetHandler` applies to the supplied URL.

EVALUATION COPY

## Creating handlers

You can create new handlers with the `Action` directive:

```
#      handler name   script
Action add-footer     /cgi-bin/footer.pl
```

`Action` executes the script whenever the associated handler is called. The script is responsible for modifying the file pointed to by the `PATH_TRANSLATED` environment variable and outputting it for the server to send back to the client.

### *Example*

Suppose you want to check users' CGI scripts before they're run. You can set up a handler that is called whenever a user's CGI script runs; this handler will check the user's code for any security breaches:

```
Action check-user-cgi /cgi-bin/sanitize.pl
<Directory /home/*/public_html/cgi-bin/>
    SetHandler check-user-cgi
</Directory>
```

EVALUATION COPY

# Redirecting Content

You can have Apache redirect users from one location to another with an alias.

## Simple aliases

The `Alias` directive allows you to associate an alias with a real file's name, as shown below:

```
#       alias    real location
Alias   /icons/  /home/httpd/icons
```

Whenever a browser requests the alias, Apache actually goes to the real location, which must be in the local filesystem, and retrieves the content from there.

When using aliases, the alias must match exactly. This means that a trailing slash on an alias, as above with `/icons/`, must be present in the request for the alias to work. In the example above, a browser requesting `/icons/` would go to the aliased location, but one requesting `/icons` would not.

**TIP:**
You can't use the `Directory` or `Location` block directives on an alias. You must use them on the real location.

## Pattern aliases

Rather than specifying an exact alias, you can specify an alias by regular expression. The `AliasMatch` directive, shown below, is more powerful than the `Alias` directive alone:

```
# redirect any requests to the icons/ directory
# to the particular corporation's icon directory
AliasMatch (.*)/icons/ /home/httpd/corp/$1/icons
```

EVALUATION COPY

## Redirects

You can redirect one URL to another with the `Redirect` directive:

```
# redirect all requests to the foo.html file
# to the web site at www.foo.com
Redirect /help.html http://www.help.org/
```

Redirection with the `Redirect` directive has several benefits over the `Alias` directive:

1. The redirected location doesn't have to be in the local filesystem; it can be anywhere on the web.
2. You can send a status indicator along with the redirection. Browsers conforming to HTTP 1.1 will use this status code as an indication of the redirection's status.

### *Sending a status*

You can send a status along with the redirection by supplying an additional parameter:

```
Redirect permanent /help.html http://www.help.org/
Redirect gone /intranet.html
```

The statuses you can send are:

- `gone`, indicating the resource is no longer maintained.
- `permanent`, indicating the resource permanently moved.
- `seeother`, indicating the resource was replaced.
- `temp`, indicating the resource moved temporarily.
- any numeric value defined by the HTTP 1.1 protocol.

**TIP:**
Use the `RedirectMatch` directive to exert more control over the resource redirection.

EVALUATION COPY

# Fancy Indexing

When a browser requests a directory for which there is no index file, Apache will display the directory's contents (assuming `Allow Indexes` is enabled for the directory). By default, Apache shows the directory's contents as a simple list, from which you can click on an item within the directory to view the contents.

However, Apache can display an icon beside the file's name that visually describes the file's type; Apache can also show a text description for the file's type. To enable this, turn on fancy indexing:

```
IndexOptions FancyIndexing
```

## Associating icons with files

You can associate an icon with a file using the `AddIcon` directive. This directive matches a file's type by extension with a given icon:

```
#       icon file          extensions
AddIcon /icons/binary.gif .bin .exe
AddIcon /icons/binhex.gif .hqx
AddIcon /icons/tar.gif .tar
AddIcon /icons/compressed.gif .Z .z .tgz .gz .zip
AddIcon /icons/a.gif .ps .ai .eps
AddIcon /icons/layout.gif .html .shtml .htm .pdf
AddIcon /icons/text.gif .txt
AddIcon /icons/c.gif .c
AddIcon /icons/p.gif .pl .py
AddIcon /icons/f.gif .for
AddIcon /icons/dvi.gif .dvi
AddIcon /icons/uuencoded.gif .uu
AddIcon /icons/script.gif .sh .csh .ksh .tcl
AddIcon /icons/tex.gif .tex
AddIcon /icons/bomb.gif core
# special files
AddIcon /icons/back.gif ..
AddIcon /icons/hand.right.gif README
AddIcon /icons/folder.gif ^^DIRECTORY^^
AddIcon /icons/blank.gif ^^BLANKICON^^
```

### *Default icon*

The `DefaultIcon` directive specifies the image to display when no previous directive has associated an icon with a particular file:

```
DefaultIcon /icons/unknown.gif
```

EVALUATION COPY

## Associating descriptions with files

The `AddDescription` directive allows you to provide a short description for files with a given extension:

```
AddDescription "GZIP compressed document" .gz
AddDescription "tar archive" .tar
AddDescription "GZIP compressed tar archive" .tgz
```

## Special directory files

The `HeaderName` and `ReadmeName` directives specify two special files that Apache uses to enhance directory indexes.

### *Index header*

The `HeaderName` directive specifies the file that Apache should display prior to the fancy indexes:

```
HeaderName header.html
```

If the `HeaderName` directive isn't specified, then Apache looks for `name.html`. If `name.html` doesn't exist, Apache looks for `name.txt`.

### *Directory description*

The `ReadmeName` directive specifies the file that Apache should display after the fancy indexes:

```
ReadmeName README
```

## Excluding files

The `IndexIgnore` directive specifies files that shouldn't be included in an auto-generated index:

```
IndexIgnore .??* *~ *# HEADER* README* RCS
```

The default, shown above, will exclude:

- all files that start with a dot and have at least two characters.
- all files ending with ~ and #.
- all files beginning with `HEADER` or `README`.
- RCS directories (created by the rcs system).

EVALUATION COPY

# Delivering Browser-sensitive Content

Apache supports the content negotiation standard declared in the HTTP 1.1 protocol specification. Apache can automatically select a resource's best representation based on the browser-supplied preferences of:

- Encoding.
- Language and character set.
- Media type.

## Encoding

The `AddEncoding` directive maps a file extension to a MIME (Multi-purpose Internet Mail Extensions) encoding type:

```
#           MIME encoding   extension
AddEncoding x-zip           .zip
```

Some browsers use documents' MIME encoding to de-encode the documents immediately. For example, a browser can unzip a document that has the x-zip MIME encoding using an external application; this saves the user from having to do so manually.

## Language

The `AddLanguage` directive associates a file extension with a language code:

```
#           code  extension
AddLanguage en    .en
AddLanguage de    .de
AddLanguage it    .it
AddLanguage ja    .ja
```

When a browser imposes a language request on Apache, Apache searches for files that match the requested language. For example, if a browser requests `index.html` but has a preference of German (language code `de`), Apache will return the document `index.html.de`, if it exists, or `index.html` otherwise.

EVALUATION COPY

### *Language priorities*

The `LanguagePriority` directive allows you to give precedence to some languages in case the browser doesn't ask for a particular language. For example, a mostly-English site might specify:

```
# assume English first, then German and Italian
LanguagePriority en de it
```

A mostly-German site might specify:

```
# Erster Deutscher, dann Englisch und Italiener
LanguagePriority de en it
```

### *Character sets*

With different languages also comes the possibility of different character sets. For example, a Japanese encoding will not use the Western character set (ISO-8660-0), because it doesn't contain the Japanese alphabet.

You can map a document extension to a character set using the `AddCharset` directive:

```
# support three Japanese character sets
AddCharset EUC-JP .euc
AddCharset ISO-2022-JP .jis
AddCharset SHIFT_JIS .sjis
```

Then, when a browser requests a particular encoding, Apache will look for a file ending in the mapped extension.

For example, suppose a browser wanted `index.html` in the Japanese language (language code `ja`) and in the ISO standard Japanese character set (character set code `ISO-2022-JP`). Apache, configured as shown above, would look for the following files in order:

1. `index.html.ja.jis`
2. `index.html.jis.ja`
3. `index.html.ja`
4. `index.html.jis`
5. `index.html`

**TIP:**

The Apache documentation for the `mod_negotiation` module explains the content negotiation algorithm in more detail.

EVALUATION COPY

## Media type

The `AddType` directive maps a file extension to a MIME (Multi-purpose Internet Mail Extensions) type:

```
#         MIME type     extension
AddType   image/gif    .gif
AddType   image/jpg    .jpg
AddType   audio/mpeg   mpga mp2 mp3
```

Some browsers use documents' MIME types to correctly render the documents. Sometimes the browser can render the document itself (in the case of `text/html`, `image/gif`, `image/jpg`, and other MIME types), but other times a browser must call an external application (in the case of `audio/mpeg` and many others).

EVALUATION COPY

# Chapter Summary

Apache is a highly configurable web server. In addition to the everyday, mundane characteristics, you can configure Apache conditionally or have it take different action based on incoming requests.

Apache also supports handling, which makes it easy for an administrator to define his or her own special processing for documents. Combined with Apache's content negotiation features, which allows it to deliver different content based on browsers' requests, handling can grow to cover every conceivable configuration a site would need.

Finally, Apache supports two other useful features. First, fancy indexing allows Apache to dynamically build a listing of files within a directory, complete with descriptive text and icons. Second, alias and redirection allows Apache to send clients to resources even if they move.

EVALUATION COPY

# Chapter 7: Performance and Security

## Chapter Overview

Administrators typically want to increase their servers' performance and maximize their servers' security. Apache tailors to both these needs, as well as to more fundamental needs, including correct operation.

## Chapter Objectives

After completing this chapter, you will be able to:

- explain Apache's security and performance goals.
- describe hardware performance issues.
- tune Apache's performance.
- secure an Apache run site.

EVALUATION COPY

# Apache's Security and Performance Goals

Apache is built for the general population of administrators who need a web server. To that end, Apache is designed to be correct first, and fast only second. Much work has gone into making Apache fast, and that work shows in Apache's benchmarks.

## Hardware and platform considerations

To meet these goals, Apache contains general code that works well in the most common situation. Specifically, most web sites have a relatively low bandwidth connection (less than 1.5Mb, which is approximately T1 speed), which Apache can handle with ease on a low-end Pentium. Higher bandwidth sites, such as those with more than 10Mb of outgoing bandwidth, need more than a single computer running a single Apache server.

### *Memory*

The single most important characteristic for a web server machine is available RAM. Whenever Apache server processes swap (transfers recently unused data from disk to memory or vice versa), users notice a delay that exceeds their usual tolerance.[2]

To help reduce the potential for swapping, set the `MaxClients` directive so that your system doesn't need to swap server processes. This will dramatically improve your system's performance.

**TIP:**
For Linux, you can inspect your swap space usage with the `swapon -s` command.

EVALUATION COPY

[2] See `http://httpd.apache.org/docs/misc/perf-tuning.html`.

### *Platform*

The operating system you choose to run Apache is a site-specific issue. Generally speaking, you should choose an operating system that fits in with the rest of your network. For example, if your network is largely Windows-based, don't run Apache under Linux; your administration staff will be taxed more with learning the operating system than administering Apache.

Regardless of the operating system you choose, make certain that you've applied the latest patches, especially network patches.

**TIP:**
Apache is not yet seasoned for the Windows NT environment. The programming model NT employs differs from the one Apache uses. Hence, the performance of Apache under NT is significantly different than that for a Unix-like system.

EVALUATION COPY

# Performance Tuning

You can tune Apache both at run-time and at compile-time. The configuration script supplied with Apache chooses the best compile-time configuration for your system, so you won't need to modify these settings.

## Run-time tuning

Tuning Apache at run-time is a matter of configuring several key directives appropriately.

### *AllowOverride*

When you allow directories to have overrides (those provided by the `.htaccess` file), you impose a significant search burden on Apache. Consider the following configuration:

```
DocumentRoot /home/httpd/htdocs
<Directory />
    AllowOverride All
</Directory>
```

A request to the site's homepage will cause Apache to check for and process each of the following files:

- `/.htaccess`
- `/home/.htaccess`
- `/home/httpd/.htaccess/`
- `/home/httpd/htdocs/.htaccess`

These files' contents aren't cached, so every request will cause this processing.

### *FollowSymLinks and SymLinksIfOwnerMatch*

Similar to the problem caused by the `AllowOverride` directive, `FollowSymLinks` and `SymLinksIfOwnerMatch` also cause Apache to check extra file information. Specifically, Apache has to check each file in directory pathnames to see if:

1. the file is a link, and if so, follow it.
2. the file's owner matches the requesting process's owner.

For maximum performance, engage `FollowSymLinks` everywhere and disable `SymLinksIfOwnerMatch`. This reduces security by allowing symbolic links created by other users, but increases Apache's performance.

EVALUATION COPY

### *HostnameLookups*

With `HostnameLookups On`, every request requires a DNS lookup to complete the request. If the DNS is slow or, worse, down, the time to complete requests will lag. For maximum performance, set `HostnameLookups Off`.

It's possible to scope DNS lookups, so that only when Apache accesses certain files will the lookup commence:

```
# disable DNS lookup
HostnameLookups Off

# lookup hostnames only when CGI programs
# requested
<Files ~ "\.cgi">
    HostnameLookups On
</Files>
```

### *Keepalives*

You should keep connections open with the `KeepAliveTimeout` directive. This allows the client-server connection to remain open and reduce the penalties of accepting new connections. However, keeping Apache processes around too long means they'll sit in a busy (idle) loop, which just wastes resources.

The default `KeepAliveTimeout` of 15 seconds attempts to minimize this effect. However, the tradeoff between network bandwidth and server resources remains regardless of the value. You should never raise this value above 60 seconds, as the benefits are lost.

### *Negotiation*

Don't disable content negotiation. The benefits far outweigh the performance hit. The one scenario where it makes sense to restrict content negotiation comes with directory indexing. Rather than using a directory index wildcard, explicitly name the allowed index files:

```
# using index as a wildcard, like below which
# matches index*, is a performance no-no
# DirectoryIndex index

# explicitly name all valid index files
DirectoryIndex index.cgi index.pl \
               index.shtml index.html
```

EVALUATION COPY

***Server creation***

When Apache experiences an increased load, it starts enough servers to meet the load and maintain the `MinSpareServers` setting. When the load spikes (in other words, increases rapidly), Apache has to start servers to meet the load. Unfortunately, this tends to swamp servers, making them swap to disk.

To minimize swapping, Apache starts one server, waits a second, starts two, waits another second, starts four; this continues exponentially until it is spawning 32 servers per second and the `MinSpareServers` setting is reached.

Experimental results show that it is usually unnecessary to adjust the `MinSpareServers`, `MaxSpareServers`, and `StartServers` settings. However, when Apache spawns more than 4 children per second, a diagnostic message goes to the error log. Lots of these errors indicate you should tune the values.

***Server death***

The `MaxRequestsPerChild` directive restricts the number of request a child server will handle. Usually this value is 0, which means that there is no limit to the number of requests handled per child. Your configuration should not set this to a low number, like 50, as that's far too few, and it typically causes swapping.

For operating systems where this parameter is important, like SunOS or an old version of Solaris, limit this to approximately 10,000. This allows the child to process enough requests to prevent swapping and also limits the absorption of system memory through memory leaks.

EVALUATION COPY

# Security

Apache has several security features that every administrator should know.

## Restricting access

Apache allows you to specify hosts that can and hosts that cannot access your web sites. The `Allow` and `Deny` directives specify the hosts that can and cannot, respectively, access your sites. These directives typically appear within a `Directory` block directive.

```
<Directory />
    Allow from company.com friend.org
    Deny from foe.com
    Allow from ally.foe.com
</Directory>
```

When Apache checks a host for access, it scans the configuration file from top to bottom and uses the first match encountered. In the example above, Apache would *deny* `ally.foe.com` even though it's declared `Allow`.

To enforce a particular order, use the `Order` directive. Revisiting the previous example:

```
<Directory />
    Order allow, deny
    Allow from company.com friend.org
    Deny from foe.com
    Allow from ally.foe.com
</Directory>
```

Because the order is declared `Allow` first, all the `Allow` directives would be checked before the `Deny` directives.

***Pedantic access***

You can allow and deny from a special class of hosts: `All`. When you use `All` in either the `Allow` or `Deny` directive, Apache will match all hosts. For example:

```
# access only from company.com & friend.org
<Directory />
    Order deny, allow
    Allow from company.com friend.org
    Deny from All
</Directory>
```

EVALUATION COPY

## Setting access options

You can further control the access options with the `Options` and `AllowOverride` directives. The `Options` directive specifies what security-related options Apache maintains for a given directory. The `AllowOverride` directive specifies what security-related parameters can be overridden by users' `.htaccess` files.

### *The Options directive*

The table below describes the parameters available with the `Options` directive:

| Parameter | Description |
|---|---|
| `All` | Enable all features, with the exception of `MultiViews`, which must be explicitly enabled |
| `ExecCGI` | Permit CGI script execution |
| `FollowSymLinks` | Instruct the server to follow symbolic links |
| `Includes` | Permit all server-side includes |
| `IncludesNOEXEC` | Permit all server-side includes except `#exec` and `#include` |
| `Indexes` | If none of the files specified in the `DirectoryIndex` directive exist, generate a directory index |
| `MultiViews` | Instruct the server to perform a best-guess when the client requests a resource that's not available; the server finds the best match based on a client's settings |
| `None` | Enable no features |

EVALUATION COPY

### *The AllowOverride directive*

The table below describes the parameters available with the `AllowOverride` directive:

| Parameter | Description |
|---|---|
| `All` | Allow a `.htaccess` file to override all security options |
| `AuthConfig` | Enable authentication-related directives (`AuthName`, `AuthType`, `AuthUserFile`, `AuthGroupFile`, and `Require`) |
| `FileInfo` | Enable MIME-related directives (`Addtype`, `AddEncoding`, `AddLanguage`, `LanguagePriority`, etc.) |
| `Indexes` | Enable directives related to directory indexing (`FancyIndexing`, `DirectoryIndex`, `IndexOptions`, `IndexIgnore`, `HeaderName`, `ReadmeName`, `AddIcon`, `AddDescription`, etc.) |
| `Limit` | Enable directives controlling host access (`Allow`, `Deny`, and `Order`) |
| `None` | A `.htaccess` file cannot override any security option |
| `Options` | Enable the `Options` directive |

### *Example*

Considering performance and maximum security, a typical site will have the following configuration for user directories (notoriously the most insecure area):

```
# set the security on users' directories
<Directory /home/*/public_html>
    Options FollowSymLinks IncludesNOEXEC \
            -FollowSymLinksIfOwnerMatch
    AllowOverride AuthConfig Limit
</Directory>
```

**TIP:**

You can remove a previously set option by placing a hyphen (`-`) in front of the option parameter.

EVALUATION COPY

## Enabling access to local documents

Allowing access to documents stored outside of your normal `DocumentRoot` is sometimes advantageous. For example, Linux systems come with a large selection of documents, stored in HTML format at `/usr/doc/HTML`.

You can give your trusted users access to this information easily:

```
# create an alias so web visible clients can
# access an area outside of DocumentRoot
Alias /doc/ /usr/doc/HTML/
<Location /doc/>
    Order allow, deny
    Allow from friend.org
    Deny from all
    Options Indexes FollowSymLinks
</Location>
```

## ServerRoot directory permissions

The first Apache server starts as user root, and then switches to the user given by the `User` directive. Any command that root runs must be safe from arbitrary user modification. Not only must the files themselves be writeable only by root, but so must the directories and directories' parents.

For example, if `ServerRoot` is `/etc/httpd/`, then you should protect the directory with:

```
$ mkdir /etc/httpd && cd /etc/httpd
$ mkdir bin conf logs
$ chown root.root . bin conf logs
$ chmod 755 . bin conf logs
```

Of course, this assumes that only root can modify `/etc`.

The `httpd` executable itself should be protected:

```
$ cp httpd /etc/httpd/bin
$ chown root /etc/httpd/bin/httpd
$ chgrp 0 /etc/httpd/bin/httpd
$ chmod 511 /etc/httpd/bin/httpd
```

EVALUATION COPY

## Safe CGI

CGI programs are a major security concern. When you give your users the ability to execute CGI scripts, you're giving them the ability to execute programs as the same user as the Apache server. Because all the users' scripts run as the same user (that of the Apache server), one user's CGI script can overwrite another user's. This is problematic, but fortunately Apache has a solution.

### *suEXEC*

suEXEC allows the Web server to run CGI scripts as a different user than the Apache server. suEXEC is not configured by default, so you will have to go back to the Apache source to enable this.

When you configure Apache, pass the Apache configuration script `configure` the `--enable-suexec` flag and the `--suexec-safepath` flag. These will apply safe settings for suEXEC for most installations.

**TIP:**
After configuring but before compiling, check the suEXEC setup by typing `configure --layout` and inspecting the values.

When Apache starts up, a properly configured suEXEC system will log the following message:

```
[notice] suEXEC mechanism enabled
```

From that point on, CGI scripts will run as either:

- the `User` and `Group` defined for a particular virtual host, as long as these differ between the virtual host and the main server.
- the user specified by in a tilde (~) expanded URL.

EVALUATION COPY

# Chapter Summary

Administrators are rightly concerned about performance and security. Apache is written to be a generally powerful server, and in all cases, Apache sacrifices speed for correct and secure operation.

Generally speaking, Apache is fast enough for most sites. Those sites that have very large bandwidth connections demonstrate a need to increase the number of Apache servers and computers, but moderate sites do not.

Administrators should maximize their systems' performance by increasing the amount of system RAM. Also, administrators need to set appropriate values in the Apache configuration file, keeping in mind the goal: reduce the amount of time the server swaps.

Finally, Apache has a large array of security facilities that make it ideal for sites with a large, untrusted user base. With Apache, you can fine-tune the configuration for directories and files, tailoring them to your exact needs. With the suEXEC mechanism, you can even make CGI scripts secure by reducing their execution scope.

EVALUATION COPY

# Chapter 8:
# URL Rewriting

## Chapter Overview

The most complicated Apache module, `mod_rewrite`, is also the most powerful. With it, you can translate any URL into another, incorporating a wide array of conditions, variables, and patterns. Administrators wishing to take their site to the "next level" will need to make heavy use of this module.

## Chapter Objectives

After completing this chapter, you will be able to:

- explain the syntax of common Rewriting rules.
- list useful scenarios where Rewriting adds benefit.
- implement Rewriting rules for common Rewriting scenarios.

EVALUATION COPY

# The URL Rewriting Engine

Often, administrators need to translate a requested URL into another on the fly. There are several options:

- The `Alias` directive, but `Alias` is limited to static translations.
- The `AliasMatch` directive, which provides regular expression-driven translation.
- The `mod_rewrite` module, which provides rule-based, regular expression-driven, URL rewriting.

The subtle difference between URL *translation* and URL *rewriting* is important. Translation is a simple mapping between URLs, governed by at most one regular-expression. Rewriting involves applying one or more rules, taking into consideration any number of conditions and variables, to produce a final result. This difference shows that rewriting is far more complex and powerful than translation.

## Rewriting fundamentals

The `mod_rewrite` module can operate in two contexts:

- *per-server context*, which applies to all the configuration, overrides, aliases, and so on for an entire Apache web server. Typically, per-server context applies to everything within the `httpd.conf` file.
- *per-directory context*, which applies to the per-server context *and* all the configuration, overrides, aliases, and so on for a directory. Typically, per-directory context augments the per-server context via `.htaccess` files.

EVALUATION COPY

### *Rewriting within Apache*

Apache processes URL requests in phases. The `mod_rewrite` module augments two of these phases:

1. URL-to-filename translation
2. URL fix-up

#### `URL-to-filename translation phase`

When Apache receives a URL request, it finds the appropriate server (which might require looking up a virtual host) and starts the `mod_rewrite` module. The `mod_rewrite` module doctors the URL according to its *per-server* rules, and sends the URL back to Apache.

#### `URL fix-up phase`

Apache then processes the URL further, eventually finding the correct data directory. Apache calls the `mod_rewrite` module again, this time with a *directory* path, and not a URL[3]. At this point, the `mod_rewrite` module applies any per-directory rules by converting the directory back into the URL (via the `RewriteBase` directive) and restarting the phases with the URL.

EVALUATION COPY

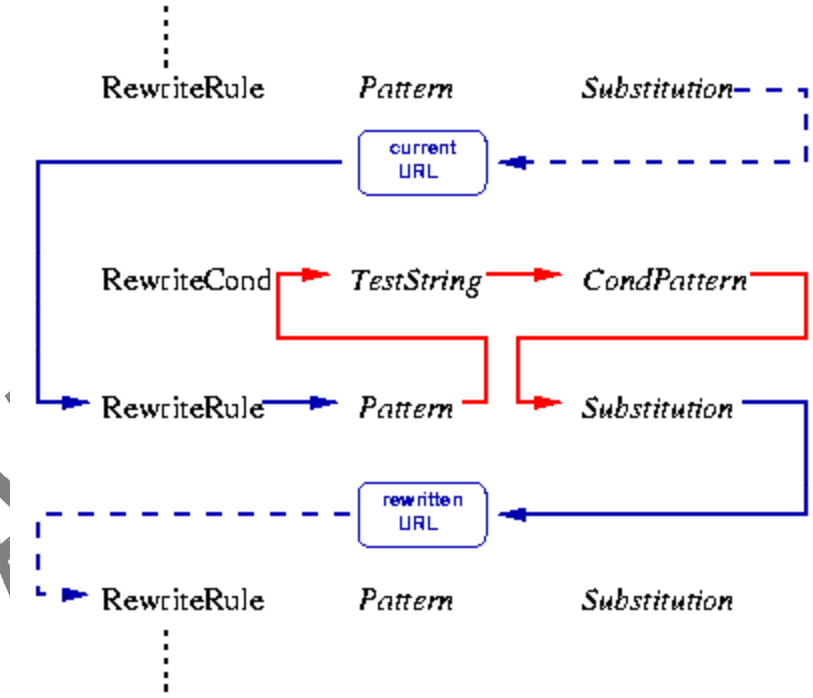
[3] There is no obvious distinction between a URL and a directory; both are ways of expressing data's location, though URLs have a more general and robust syntax.

### *Rules and conditions*

When invoked, the `mod_rewrite` module steps over each rule (read from top to bottom in the configuration file). When any rule matches, the `mod_rewrite` module checks all corresponding conditions.

To accomplish this, the `mod_rewrite` module introduces two important directives:

- `RewriteRule` ***`pattern replacement`***, which maps a regular expression ***`pattern`*** to a specific ***`replacement`***.
- `RewriteCond` ***`test condition`***, which expands variables (among other things) from the rule's ***`pattern`*** and checks that against ***`condition`***.




The `RewriteCond` conditions immediately precede the rules to which they apply; in other words, there are no interleaving `RewriteRule` directives between associated `RewriteCond` directives.

EVA COPY

### Options

Both the `RewriteCond` and `RewriteRule` directives support an additional, and optional, third parameter. This parameter specifies one or more options that govern the condition or rule, and they are usually written with:

- square brackets (`[]`).
- a hyphen (`-`).
- a logical operator (e.g., `!` for not).

For example, suppose your site's main page had several versions based on a browser's capability. You could select the appropriate page with the following rules and conditions:

```
# enable the engine (this is required, because
# the engine is NOT on by default)
RewriteEngine on

# condition: does the browser (in the variable
# %{HTTP_USER_AGENT}) contain "Mozilla"
RewriteCond  %{HTTP_USER_AGENT}  ^Mozilla.*

# if the URL is / (ie, the main page), and the
# Netscape condition above applies, replace the
# URL with index.NS.html.  Finally, stop the
# rewriting process (indicated by the [L]
# option).
RewriteRule  ^/$ /index.NS.html  [L]

# condition: is this the text-based Lynx
# browser?
RewriteCond  %{HTTP_USER_AGENT}  ^Lynx.*

# if so, use the text-only version page and
# stop rewriting.
RewriteRule  ^/$ /index.TO.html  [L]

# otherwise, rewrite the URL to use the
# standard page.
RewriteRule  ^/$ /index.html
```

EVALUATION COPY

# Common Rewriting Needs

The `mod_rewrite` module has virtually unlimited possibilities. The case studies below illustrate a few of the common rewriting tasks.

## Trailing slashes

When you request a URL that's a directory, you mustn't forget the trailing slash. For example, suppose you try to access:
`http://www.mycompany.com/~user/subdir`.
Apache tries to locate the file "subdir", which probably doesn't exist. What you wanted the URL to say was:
`http://www.mycompany.com/~user/subdir/`.

To accomplish this, use the following configuration:

```
RewriteEngine  on

# set the base to all directory-to-URL converts
# to ~user/
RewriteBase    /~user/

# match from the right ([R])
RewriteRule    ^subdir$  subdir/  [R]
```

Generally speaking, Apache actually tries to fix this trailing slash problem on its own. Sometimes it fails, though (for example, when you've already done a lot of complicated rewritings). Hence, the method above is hard-coded for those particular cases.

Users could, of course, put a general configuration (shown below) in their top-level `.htaccess` files:

```
RewriteEngine  on
RewriteBase    /~user/

# if the file is a directory (-d)
RewriteCond    %{REQUEST_FILENAME}  -d

# append the slash onto the end
RewriteRule    ^(.+[^/])$           $1/  [R]
```

***Note:*** *This general method imposes a small performance penalty.*

EVALUATION COPY

## Users on another server

Suppose you want your main web server (`http://www.mycompany.com/`, for example) to service user pages (by the tilde syntax), but all the user data is stored on `login.mycompany.com`.

Using the following configuration, you can reroute all user directory requests to another server:

```
RewriteEngine on

# rewrite all user account requests to the
# login machine.  Note that two options are
# specified [R,L] - "Right" and "Left".
RewriteRule ^/~(.+) \
          http://login.mycompany.com/~$1 [R,L]
```

## Redirect invalid URLs

The `Redirect` directive, seen earlier, allows you to specify explicit resources that move. If resources move often, it can be cumbersome to continually update the configuration.

With `mod_rewrite`, you can explicitly send all failed URLs to a new location, potentially the new location or an error document:

```
RewriteEngine on

# condition: the requested resource (given by
# the URI) doesn't exist (! -U)
RewriteCond   %{REQUEST_URI} !-U
RewriteRule   ^(.+) http://other.com/$1
```

EVALUATION COPY

## Time is important

Suppose you want to redirect traffic based on the time of day. This might be important for pages that service time information, where they might want to show one page for AM and another for PM. This technique might also apply for web sites that go down for maintenance routinely.

Again, the `mod_rewrite` module makes this trivial:

```
RewriteEngine on

# condition: the hour and minute (taken from
# variables) is between 8a AND 6p.  Note that
# sequential conditions are, by default, ANDed
# together.  Use the [O] option to OR them.
RewriteCond   %{TIME_HOUR}%{TIME_MIN} >0800
RewriteCond   %{TIME_HOUR}%{TIME_MIN} <1800

# show the normal, business page.
RewriteRule   ^/$  index.html

# otherwise, show the after hours page.
RewriteRule   ^/$  afterhours.html
```

## Faking static pages

Suppose you wanted users to access a CGI script named `app.pl`, but, to prevent any potential security attacks, you don't want the users to know they're accessing a script. For example, you want them to access app.html but a CGI script creates that page.

Here's how to do it:

```
RewriteEngine on
RewriteBase /serveroot/cgi-bin/
# transform app.html into app.pl, AND set the type to a
# cgi script.
RewriteRule ^app\.html$  app.pl [T=application/x-httpd-cgi]
```

EVALUATION COPY

# Chapter Summary

The `mod_rewrite` module is perhaps the most powerful module Apache has to offer. Without a doubt, it's also the most complicated. Once you've mastered its syntax, you'll be able to rewrite URLs according to your own specifications, taking into account server and environment variables and other conditions.

The Rewriter works by hooking into Apache's multi-phase API. In either per-server or per-directory context, the `mod_rewrite` module takes a URL and transforms it into another URL using conditions and rules. You specify conditions with the `RewriteCond` directive and rules with the `RewriteRule` URL.

EVALUATION COPY

*This page intentionally left blank.*

EVALUATION COPY

# Appendices


**APPENDICES** **A-1**

**LAB 1: INTRODUCTION** **A-2**
**LAB 2: APACHE INSTALLATION** **A-3**
**LAB 3: APACHE CONFIGURATION** **A-5**
**LAB 4: EFFECTIVELY WORKING WITH APACHE** **A-7**
**LAB 5: VIRTUAL HOSTS** **A-8**
**LAB 6: ADVANCED CONFIGURATION** **A-10**
**LAB 7: PERFORMANCE AND SECURITY** **A-11**
**LAB 8: URL REWRITING AND CUMULATIVE LAB** **A-13**
**REFERENCES** **A-14**


EVALUATION COPY

# Lab 1: Introduction

## Part A (5 minutes)

Answer the following questions:

1. What is Apache, and who provides enhancements and fixes?

2. What are some of Apache's features?

3. How does Apache compare to other servers?

EVALUATION COPY

# Lab 2: Apache Installation

## Part A (10 minutes)

Answer the following questions:

1. Where can you find the latest enhancements, fixes, and documentation for the Apache web server?

2. Suppose you want to set up a corporate Intranet site, where only users on your company's internal network will be able to access the site. Where should you place the Apache web server?

3. What is the name of Apache's configuration script? When do you need to use it?

4. What is Apache's configuration file? What is the name of the Apache binary?

5. What is an Apache module?

## Part B (30-45 minutes)

This part asks you to install the Apache server binary.

1. Download (or obtain from your Instructor) the Apache server binary. If your architecture doesn't supply binaries, obtain the source code.
2. If you're working with the source code (and not a packaged, pre-built binary), uncompress, configure using the standard defaults, and build Apache.
3. Install the built binaries, either from the pre-built package or from your built source code, overwriting any preexisting copies. Where are `httpd` and `httpd.conf` placed?
4. Review the `httpd.conf` and verify that the every `LoadModule` has a corresponding `AddModule` directive.

EVALUATION COPY

5. Make a copy of the `httpd.conf` file and store it in a safe location. You will use this saved copy in a later lab.
6. Configure the server to start at boot. For example, use `chkconfig` on Red Hat Linux systems.
7. Start the server without rebooting the computer. Open a browser to `http://localhost/` and verify the server is working correctly.
8. Review the log files. What information do you see?
9. Reboot your computer to verify that the server starts after reboot.

EVALUATION COPY

# Lab 3: Apache Configuration

## Part A (5 minutes)

Answer the following questions:

1. What's the difference, both conceptually and syntactically, between simple and block directives?

2. Name two ways to configure a directory's parameters.

3. What are two ways Apache can provide "safety nets" for runaway or buggy Apache servers?

## Part B (40 minutes)

This part asks you to configure several fundamental Apache parameters.

1. Create a new user called `www`. For this exercise, the user can have any available user ID. Also create a group called `www`.
2. The server should run as user and group `www`. Do you need to change any files to be readable or writeable by this user or group?
3. Configure Apache as a standalone server that *listens* on ports 80 and 8080.
4. Verify that the server's `ServerRoot` and `DocumentRoot` directives point to the correct location.
5. Specify that only `index.html` and `index.shtml`, in that order, should be allowed index files.
6. Disable the user-specific web pages of `root` and `www`, and set the default directory for all other users to "web".

EVALUATION COPY

7. Restart the server and verify that:
    - the server listens on both ports 80 and 8080 (e.g., `http://localhost:8080`).
    - the server doesn't allow you to view the `root` and `www` user's web pages.
    - the server allows you to view the contents of users' home pages under their `web` directory.
    - the server only shows indexes when the files are named either `index.html` or `index.shtml`, and always `index.html` first.

EVALUATION COPY

# Lab 4: Effectively Working with Apache

## Part A (5 minutes)

Answer the following questions:

1. What are two ways to control Apache?

2. Which two logs does Apache write to by default? Can you customize these logs?

## Part B (15 minutes)

This part asks you to work with the Apache control script.

1. Locate the `apachectl` script. **Hint:** start by checking the same directory that `httpd` is in, and if that fails, try `find / -name apachectl`. If you do not have `apachectl`, do you have a system-dependent control script, such as `/etc/rc.d/init.d/httpd`?
2. In a browser, go the server's homepage `http://localhost/`. Stop the server and hit reload. What happens?
3. Start the server, reload the page, and shut the server down gracefully. Is there any message? What does this say about the HTTP protocol?
4. Show the server status. Start the server and show the status again. What differences does the status information show?

## Part C (30 minutes)

This part asks you to configure logging.

1. Create a new custom log format called `everything` that includes every format available. For example, the format string could be `"%b %f %h %l %p %P %r %s %t %T %u %U %v"`. Format the message so that the information is clearly recognizable.
2. Log the `everything` format to `logs/everything_log`.
3. Restart the server and access the server's homepage. How does the information in this log differ from the standard logs?

EVALUATION COPY

# Lab 5: Virtual Hosts

## Part A (10 minutes)

Answer the following questions:

1. What are the differences between IP address, name, and dynamically-named virtual hosts?

2. What limitations can systems encounter while using virtual hosts?

## Part B (45 minutes)

This part asks you to configure several name-based virtual hosts.

1. Add the following entries to your system's `/etc/hosts` files:

```
127.0.0.1   company1    www.company1.com
127.0.0.1   company2    www.company2.com
127.0.0.1   company3    www.company3.com
```

2. Create three directories under your document root directory named `company1`, `company2`, and `company3`. For example, assuming your `DocumentRoot` directory is `/home/httpd/htdocs`, you'd have `/home/httpd/htdocs/company1`, etc.

3. Within these directories, put simple index.html files that will allow you to distinguish between directories. For example:

```
<HEAD>
<TITLE>Company 1</TITLE>
</HEAD>
<BODY>This is company 1's page</BODY>
```

4. Configure Apache to use name-based virtual hosts. Your IP address should be 127.0.0.1.

5. Create three `VirtualHost` blocks that contain the following directives: `ServerName`, `DocumentRoot`, `ErrorLog`, and `CustomLog`. These directives should be set appropriately for each virtual host.

6. Restart the server and verify you get separate pages for www.company1.com, www.company2.com, and www.company3.com.

EVALUATION COPY

## Part C (15 minutes)

This part asks you to replace all your work from Part B with dynamically-named hosts.

1. Configure Apache to use the `Host:` header to lookup canonical names.
2. Use a custom log format that shows the virtual host in the virtual common (`vcommon`) format.
3. Replace all the statically-declared `VirtualHost` blocks from Part C with a single `VirtualDocumentRoot` directive.

EVALUATION COPY

# Lab 6: Advanced Configuration

## Part A (5 minutes)

Answer the following questions:

1. What is an Apache handler?

2. What is content negotiation?

## Part B (15 minutes)

This part asks you to work with conditional directives.

1. Configure Apache to use your `everything` log format (from Lab 4, Part C) only when:
   - `PARANOID` is defined
   - The `mod_log_config` module is available
2. Update your Apache control script (e.g., `/etc/rc.d/init.d/httpd`) to define `PARANOID`.

## Part C (15 minutes)

This part asks you to work with redirection and aliases.

1. Configure Apache to *temporarily redirect* `/company1/` to `http://www.company1.com/` and redirect `/company2/` to `http://www.comapny2.com/`.
2. Configure Apache to *alias* `/company3/` to the `DocumentRoot` for company3.
3. Compare access to `http://localhost/company3/` and `http://localhost/company1/`. Is there any difference?
4. Compare access to `http://localhost/company3/` and `http://localhost/company3`. Is there any difference?

EVALUATION COPY

# Lab 7: Performance and Security

## Part A (5 minutes)

Answer the following questions:

1. What is the single biggest physical requirement for a web server? Why?

2. Why does Apache not start spare servers immediately, but instead starts some, waits a second, starts more, etc?

3. In what order does Apache apply `Allow` and `Deny` directives?

## Part B (45 minutes)

This part asks you to experiment with Apache's performance settings. As you change values, comment out the original values so you can easily restore them.

1. Partner with one neighbor.
2. Set `StartServers` to 1, `MinSpareServers` to 20, and `MaxSpareServers` to 20. Restart Apache. Do you notice any delay in Apache's start?
3. Increase `MinSpareServers` to 200 and `MaxSpareServers` to 200. Restart Apache. Do you notice any delay?
4. Set the `MaxRequestsPerChild` to 1. Restart Apache. Open company1's homepage and reload the page rapidly. Do you notice any performance drain on the server? Do you hear the computer's hard drive working more than normal?
5. Drop `KeepAliveTimeout` to 1. Restart Apache and rapidly reload the page. Is Apache's performance critically bad?
6. Experiment with these values to try and improve performance. Are you able to distinguish between adequate and superior performance?
7. Restore the values to their original settings.
8. Remove the entries you added to `/etc/hosts` in Lab 5, Part B.
9. Configure Apache to lookup host names.

EVALUATION COPY

10. Have your partner change the entries in his or her `/etc/hosts` file to your IP address. For example, if your server's IP address is 192.168.0.1, your partner's `/etc/hosts` file would contain:

```
192.168.0.1 company1    www.company1.com
192.168.0.1 company2    www.company2.com
192.168.0.1 company3    www.company3.com
```

11. Open www.company1.com in your partner's browser. Does Apache have a difficult time serving the page? Why or why not.
12. Restrict name lookups to only files in the `http://www.company2.com/` location. **Hint:** Use a `<Location>` block directive.
13. Repeat step 11. Is there any improvement? Why or why not.

## Part C (30 minutes)

This part asks you to configure Apache's security. In this part, you will partner with the same neighbor from Part B.

1. Configure your server to deny all access from your partner's IP address.
2. Have your partner verify he or she can't connect to your server.
3. Alias `/doc/` to `/usr/doc/HTML/`.
4. Configure your server to allow access to this directory from your partner's IP address.
5. Can your partner connect? Why or why not?
6. Explicitly set the access order for your server (configured in step 1) to `Order allow,deny`. Is your partner able to access the `doc/` directory now? Why or why not?
7. Move the configuration performed in step 4 to the configuration performed in step 1. Is your partner able to access the `doc/` directory now? Why or why not?

EVALUATION COPY

# Lab 8: URL Rewriting and Cumulative Lab

## Part A (5 minutes)

Answer the following questions:

1. How does the `mod_rewrite` module process the configuration given in the example on page 8-5? Draw a schematic.

## Part B (90 minutes)

In this part, you will complete a lab that encompasses everything you've learned so far.

1. Restore the `httpd.conf` file you saved in Lab 2, Part B, Step 5.
2. Limiting your review, implement the following:
   - Paranoid error logging that logs all events. You should only enable this type of logging when the PARANOID variable is set.
   - Two virtual hosts, as described in Lab 5, Part B, Steps 1-5. In addition, the virtual hosts should each run as separate users and groups. So, for example, company1 could run as user `com1` and company2 could run as user `com2`.
3. Ensure that you're loading the `mod_rewrite` module.
4. Implement the five common rewriting rules seen on pages 8-6 through 8-8. Verify that each of these work.
5. Disable the virtual hosts and translate the URLs using just the `mod_rewrite` functionality. Can you do everything with `mod_rewrite` that the `VirtualHost` functionality gives you?

## Challenge 1 (90 minutes)

1. Revisit the web server location schematic presented in Chapter 1. Using just `mod_rewrite`, have the main web server, which is the "external" server on the untrusted network, retrieve data from an internal host *behind the firewall.* **Hint:** the retrieval should happen via proxy.
2. For the purposes of this part, it's only necessary to actually configure the system; you don't need to test it -- ask your instructor to verify your solution.

EVALUATION COPY

# References

The materials below provide valuable Apache administrative help and should be close at hand for Apache administrators:

- **Maximum Linux Security**. Sams Publishing. 2000.
- Killelea, Patrick. **Web Performance Tuning**. O'Reilley Press. 1998.
- Laurie, Ben; Laurie, Peter. **Apache: The Definitive Guide**. 2nd Edition. O'Reilly Press. 1999.
- LeBlanc, Dee-Ann. **Linux System Administrator, Black Book**. CoriolisOpen Press. 2000.
- `http://httpd.apache.org/`. Apache Project. The Apache Web Server's primary page.
- `http://www.zdnet.com/pcmag/stories/reviews/0,6755,2551188,00.html`. ZDNet: Benchmark Tests: Web Platforms. A benchmark of Apache on several operating systems.
- `http://webcompare.internet.com/cgi-bin/quickcompare.pl`. Web Server Quick Compare. An overview comparison of most web servers on the market today; metrics include supported operating system and cost.

EVALUATION COPY