



# HTTP: Apache

*This material is based on work supported by the  
National Science Foundation under Grant No. 0802551*



*Any opinions, findings, and conclusions or recommendations expressed in this material are those of  
the author (s) and do not necessarily reflect the views of the National Science Foundation*

# Lesson Overview

As a system administrator, you will most likely be responsible for installing or administering a web server in addition to your regular duties.

The most popular web server software is Apache. Statistically, Apache is used more than all other web servers combined. Chances are very high you will end up with an Apache web server at some point in your career.

This lesson will introduce students to the popular Apache Web Server application. You will explore Apache history, HTTP protocol, Apache installation, configuration, and security.

The final lab activity will allow you to download, install and configure a working Apache server.

Select **PLAY** below to review the lesson introduction:



View Video  
VideoLesson8HTTPApachen  
troduction(C3L8S2).mp4

# Objective

You should know what will be expected of you when you complete this lesson. These expectations are presented as objectives. Objectives are short statements of expectations that tell you what you must be able to do, perform, learn, or adjust after reviewing the lesson.

## **Lesson Objective:**

Given a need for a web server, a student will be able to defend the use of free alternatives such as Apache and will install and configure an appropriate web server as per industry standards.



# Lesson Outline

In this lesson, you will explore:

- ❖ Apache Webserver
  - Purpose & Benefits
  - HTTP
- ❖ Apache Implementation
  - Installation
  - Configuration
  - Directives
  - Security

# Purpose of Webservers

A web server is probably the most over looked and yet most important tool computer users work with on a daily basis. Webservers perform their tasks outside users' immediate operating system. Consequently, users have no need to worry about it. They trust that it works and remains reliable for a long time.

The Internet works on a client/server basis. For example, when you surf the Internet, you may use your local machine's browser to access Yahoo at <http://www.yahoo.com>. The browser and the operating system work together to send the request for Yahoo out to the Internet and locate the web server you are attempting to access. The web server's main job is to supply the prewritten content (web page) back to the user who requested it. Your web browser then receives the HTML text, and converts it in to the page that you view.



During the infancy of the Internet, web users used to use an application called [Gopher](#) that retrieved content from web servers. In those days, most of the web content consisted of text files and was not very large in size compared to today's documents. In the mid '90s, web developers started looking for ways to add multimedia (pictures, videos, and sound) to web pages. Html 1.0 was developed, and the term "surfing the Internet" took on new meaning. Developers also needed to create a way to provide these multimedia pages to users when requested, and so Apache was created in 1995 to meet this need.

Select **PLAY** below to learn about Apache:

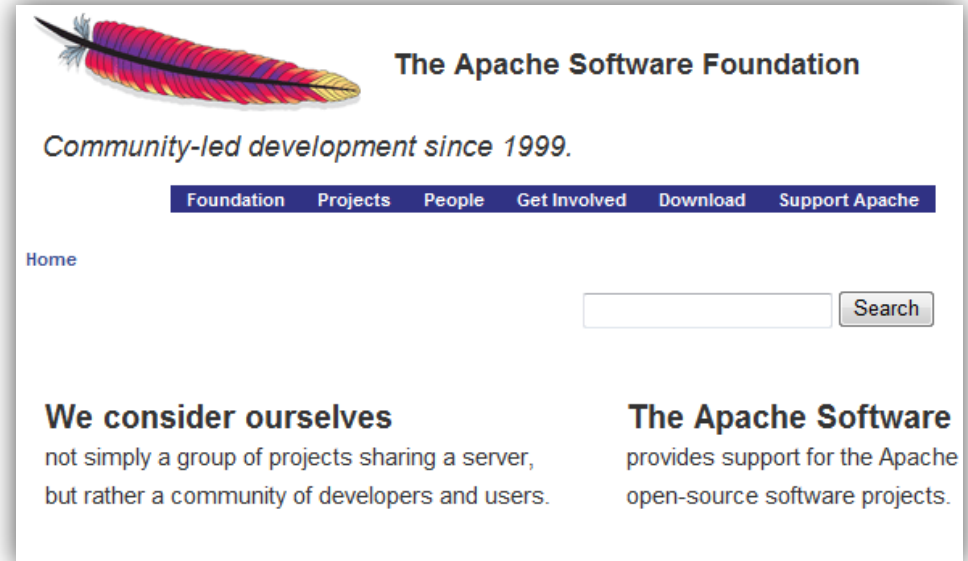
View Video  
VideoLesson8ApachePt2(C  
3L8S6).mp4

# Apache is Free!

Apache is a full-featured, powerful Web server available absolutely free. The Apache Software Foundation that develops Apache does not derive revenue from the Apache server software, so it cannot afford to offer robust technical support.

Consequently, amenities such as phone or online support are not included with Apache. However, abundant documentation and support is available, although such support may not match the offerings from commercial software.

Apache is also free in the sense that the source code is publicly available. Computer savvy programmers can modify the source code to make a better product.



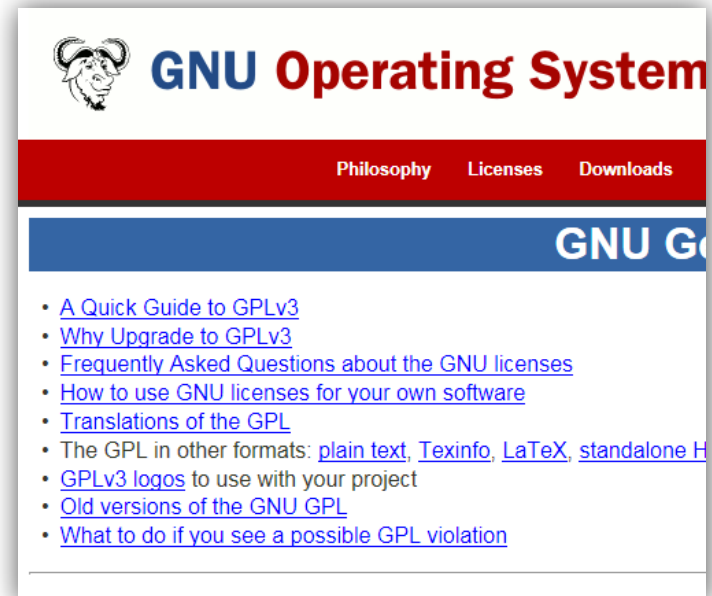
## For Review:

- [Apache Software Foundation](#)
- [Apache Wiki](#)

# Apache is Open-source

You can get the [source code](#) for Apache and modify it to your heart's content. Most people do not use the source code to modify the default operations of Apache. Instead, they modify the way in which the software is compiled. In other words, they modify which pieces are added or used in their installed version.

If you need a mean, lean server, you can recompile the source code to create a custom server with only the options you need. That said, if you ever find a problem or need to make a rudimentary change to the Apache source code, you can.



## Required Reading:

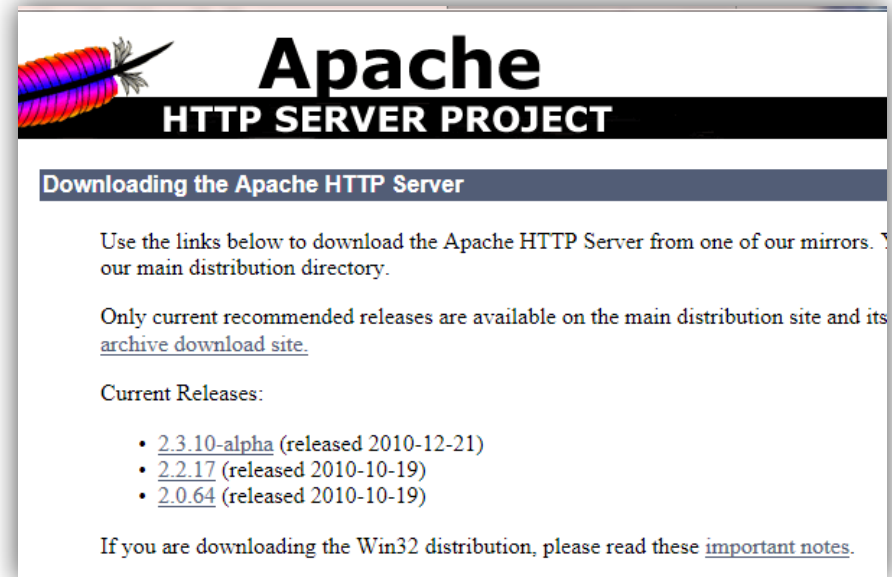
- [GNU Licenses](#)

# Apache is Cross Platform

Apache is available for [multiple platforms](#), including the following:

- Unix
- Linux
- Windows (9x through Win7, although server versions—NT/2000/XP—are preferred)
- Novell NetWare
- Mac OS X (BSD under the GUI)

Besides a few small details, such as the placement of its files in the file system, Apache operates in the same manner on the platforms mentioned above.



## Required Reading:

- [Apache Server Project](#)



# Apache Updates

Apache is maintained by the Apache Software Foundation and is under continual development and improvement. Bug and security fixes take only days to find and correct, which makes Apache the most stable and secure web server available.

## Tech Tip!

*The relative stability and security of any web server depends on the system administrator as much as, if not more than, the underlying software.*

Another advantage of [rapid development](#) and releases is the robust feature set. New Internet technologies can be deployed in Apache much more quickly than in other web servers.

## Development

There is a core group of contributors, formed in 1995, who are granted access to the source code control repository. The HTTP Project Management Committee (PMC), formed in 1996, manages the overall management. The terms "The Apache Group" and "The Apache Project" are used interchangeably.

The project is a meritocracy -- the more work you do, the more you are considered for the active PMC members. There is a group of people who are granted access to the source code repositories. Changes to the code are proposed on the mailing list, and then a committer to commit a code change during a release cycle.

Our primary method of communication is our mailing list. We use it to discuss features to add, bug fixes, user problems, development plans, and other matters. Proposed changes are communicated to the mailing list by one of the committers. Anyone on the mailing list can propose a change to the server are counted towards the requirements for becoming a committer.

New members of the Apache HTTP Project Management Committee are elected by the voting members. In most cases, this "new" member is a committer.

The project guidelines continuously evolve under the influence of the community.

## Required Reading:

- [Apache Development](#)

# The Apache Name

The name 'Apache' was chosen from respect for the various Native American nations collectively referred to as [Apache](#), well-known for their superior skills in warfare strategy and their inexhaustible endurance.

The name *Apache* also makes a cute pun on "a patchy web server" —a server made from a series of patches—but this was not its origin.

The group of developers who released this new software soon started to call themselves the "Apache Group."

## How Apache Came to Be

In February of 1995, the most popular server Supercomputing Applications, University of Illinois webmasters had developed their own extensions. They gathered together for the purpose of coordinating space, and logins for the core developers on a list that formed the foundation of the original Apache project.

Brian Behlendorf Roy T. Fielding Robert  
David Robinson Cliff Skolnick Ran  
Robert S. Thau Andrew Wilson

with additional contributions from

Eric Hagberg Frank Peters Nicolas Pio

## Required Reading:

- [Apache HTTP Server](#)
- [HTTP Server Project](#)

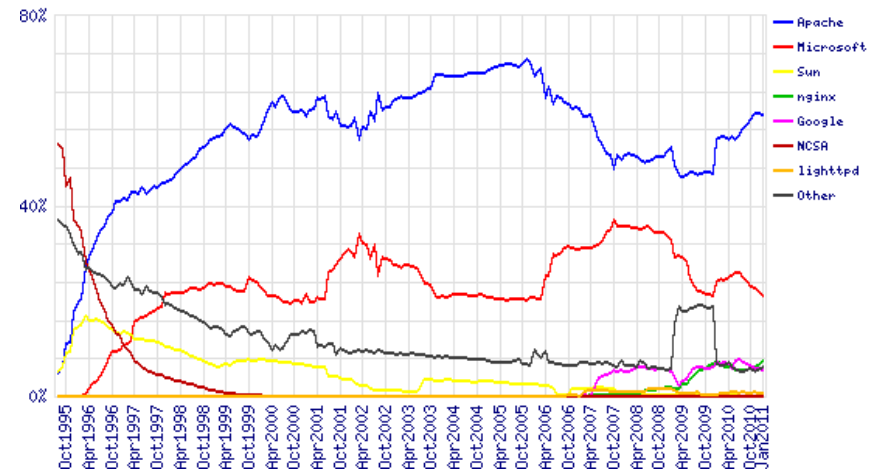
# Apache Capabilities

Apache continues to implement its features with distinct pieces, or [modules](#). Utilizing a modular approach to feature implementation enables Apache to be deployed with only the amount of overhead necessary for the features desired. Modularity also allows third parties to develop their own modules to support their technologies.

Apache supports almost all Internet Web technologies, including proprietary solutions such as Microsoft's FrontPage Extensions. Apache supports various HTTP protocols, scripting, authentication, and platform integration.

Visit the [Apache module Web site](#) for information on the modules included with Apache and the registered third-party modules. For our purposes, we care about the following capabilities:

- Robust [HTTP](#) delivery
- Configurable, reliable security
- Integration with [PHP](#) and [MySQL](#)
- [CGI](#) and other scripting integration



## For Review:

- [Web server survey](#)

# HTTP Protocol



# HTTP Protocol

**HTTP** (Hypertext Transfer Protocol) is a fairly simple protocol. A client requests data from a server, the data is parsed against a list of content types, and the server sends the data to the client via the system required by the content type.

Such content types are referred to as Multipurpose Internet Mail Extensions (*MIME*). These extensions enable non-ASCII data to be sent over the Internet. The HTTP server uses **MIME** types to determine how to send specific files.

Today's HTTP specification is quite complex. Interested readers should visit the World Wide Web Consortium Web site [www.w3.org](http://www.w3.org) for more information.

Select **PLAY** below to learn about HTTP:



View Video  
VideoLesson8HTTP(C3L8S14).mp4

## Required Reading:

- [What is HTTP?](#)

# HTTP Over TCP

Most web traffic is conducted over TCP port 80. However, as the web has matured, several other ports are used to serve data to clients. Port 443, for example, provides secure web connections via SSL. Other protocols are also available via a standard browser thanks to Netscape's innovative plug-ins. These plug-ins (now used by most browsers) enable other applications to send and receive data through the browser— not always via HTTP on TCP port 80.

You can configure Apache to operate on any port. Alternative port options include TCP ports 81, 85, and 8080. Most administrators choose to operate Apache on an alternative port only for the following reasons:

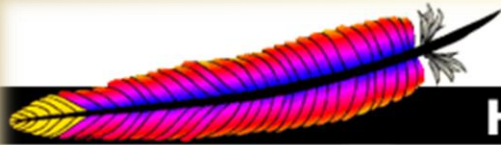
- The server needs to be hidden from casual hackers (who look for it on port 80).
- More than one server is running on a given machine.
- Some other port conflicts or an application needs the alternative port.

## **Note:**

Some applications and services (like plugins) that rely on the browser interface for communication do not use the configured port—80 or otherwise—for their communication.

When changing the default Apache port, you should pick a port above 1023 to avoid conflicts with existing Internet services.

# Apache



## Apache HTTP SERVER PROJECT

### Essentials

- [About](#)
- [License](#)
- [FAQ](#)
- [Security Reports](#)

### Download!

### The Number One HTTP Server On The Internet

The Apache HTTP Server Project is an effort to develop and maintain an open-source, efficient and extensible server that provides HTTP services in sync with the current H

Apache httpd has been the most popular web server on the Internet since April 1996.

The Apache HTTP Server ("httpd") is a project of [The Apache Software Foundation](#).

### Apache HTTP Server 2.2.17 Released

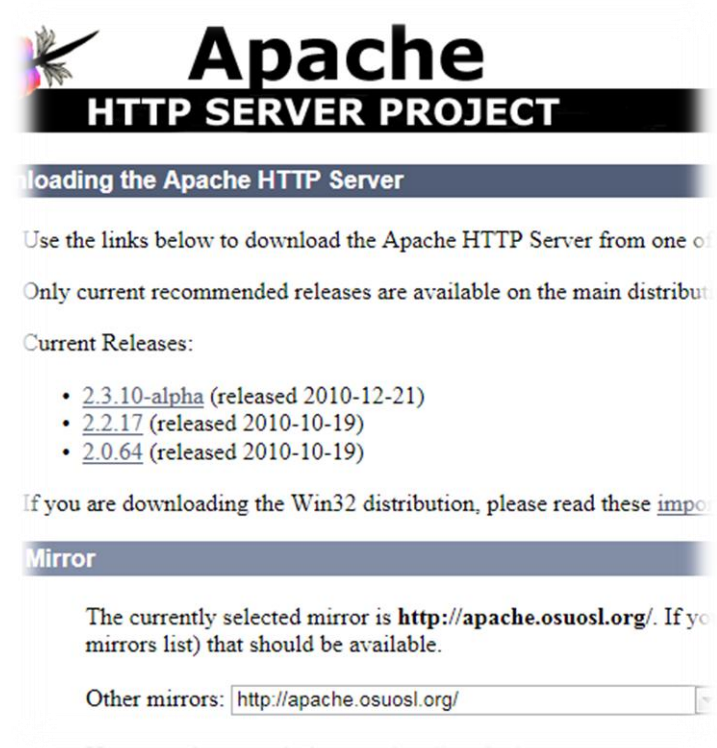


# Download and Install Apache

Apache is a full-featured HTTP server, yet, it is surprisingly easy to install. Apache is server software, so you must consider the many security risks and implications on your system after Apache is installed.

Students are advised to be cautious in installing and running the Apache server because it will provide services that could allow system access to remote users if improperly configured.

Navigate to [Apache's server project and download](#) the latest version of the software from one of the many mirrors provided. On the next few slides you will learn to install Apache.



The screenshot shows the Apache HTTP Server Project website. At the top is the Apache logo (a stylized bird) and the text "Apache HTTP SERVER PROJECT". Below this is a section titled "Downloading the Apache HTTP Server". The text says: "Use the links below to download the Apache HTTP Server from one of the mirrors. Only current recommended releases are available on the main distribution." Under "Current Releases:", there is a list of versions: 

- [2.3.10-alpha](#) (released 2010-12-21)
- [2.2.17](#) (released 2010-10-19)
- [2.0.64](#) (released 2010-10-19)

Below this, it says: "If you are downloading the Win32 distribution, please read these [important notes](#)." There is a section titled "Mirror" which says: "The currently selected mirror is <http://apache.osuosl.org/>. If you have a better mirror, please let us know (see the mirrors list) that should be available." At the bottom, there is a text input field labeled "Other mirrors:" with the value "http://apache.osuosl.org/" entered.



# Installing Apache from Package Mgr

Most Linux distributions have their own packaging scheme that allows programs to be distributed and installed on systems. Red Hat distributions, for example, have the *RPM* (Red Hat Package Manager) format. The advantages to using packages are as follows:

- ❖ The programs can typically be located and installed very easily. For example, you can easily find packaged programs on the Red Hat network, and you can download and install them by using the Red Hat Update Agent. Ubuntu uses a Debian package manager and can be utilized in a similar manner.
- ❖ Packages typically handle dependency issues for you. That is, if you need other tools or programs to use a particular program, the package has those dependencies encoded in it and will alert you of this requirement before you use the new program.
- ❖ The packages expand themselves, installing their components into the correct directories—typically with one command. Applications installed from packages also follow the base distribution's conventions for locations of binaries and configuration files.

Select **PLAY** below for help installing Apache:



View Video  
VideoLesson8InstallApache(C3L8S18).mp4

# Installing Apache (Contd)

Redhat and Fedora currently ship with Apache Version 2 as part of the installation and can be selected from the package manager. You can also use the console **rpm** command to install a package. The form of the command for RPM installation is as follows:

**rpm -i <rpm file name>**

The rpm application inspects the package, tests the system for any dependencies, and installs the package. If you use the RPM package to install Apache, the various pieces are installed according to Red Hat's file location scheme. The binary is placed in **/usr/sbin** and the configuration files in **/etc/httpd**.

You can test whether the server was installed by running it with the **-v** parameter, as follows:

**/usr/sbin/httpd -v**

The server should respond with its version and build date.

Start the server by using the **apachectl** script with the following command:  
**/usr/sbin/apachectl start**

## Note:

Verify your  
downloads  
using PGP or  
MD5 Signatures

## Required Reading:

- [Verification](#)

# Testing the Installation

After Apache is installed and running, you can test it by pointing a browser at the machine running the server. On the server machine itself, you can point a browser to the following address:

**`http://localhost`**

If you are using another machine to connect to the server, replace **localhost** with the server's fully qualified name or its IP address. To stop the server, use the following command:

**`/usr/sbin/apachectl stop`**

On Linux machines, you can use the process status command, **ps**, to determine whether the server is running. Most Linux distributions report the Apache server processes as **httpd**, but some report it as **apache**. Using one of the two following commands should display the Apache processes currently running:

**`ps -A | grep "httpd"`**

or

**`ps -A | grep "apache"`**

## Note:

An unmonitored Web server can present a security hazard to the system running it and to attached network(s).

You should immediately implement security measures or stop the server whenever you do not need it.

# Locating Apache Files

Knowing the location of various Apache files is important. You need to edit the configuration files, write scripts to access the executables, and view the log files. Apache installs in a variety of directories, depending on the operating system you use.

Depending on your installation, you will find the key files by:

- Searching for the file **apache** or **httpd** (main application)
- Doing a configuration search for the file **httpd.conf** (main configuration file)



Select **PLAY** below for help locating Apache files:

View Video  
VideoLesson8ApacheFiles  
(C3L8S21).mp4

## Other key files:

- Search in modules for the file **mod\_access.so**
- Search for the file **access.log** or **error.log**. (Key log files)

# Configure Apache

The following sections detail the various parts and parameters in the **httpd.conf** file. On Linux, you can usually find this file in the following location:

**/etc/httpd/conf/httpd.conf**

However, you might also find the configuration file in these locations:

**/usr/local/apache2/conf/httpd.conf**

## Note:

If you have trouble finding the *httpd.conf* file, use your operating system's search feature to locate it.



## Required Reading:

- [Platform specific help](#)

# Configure Apache

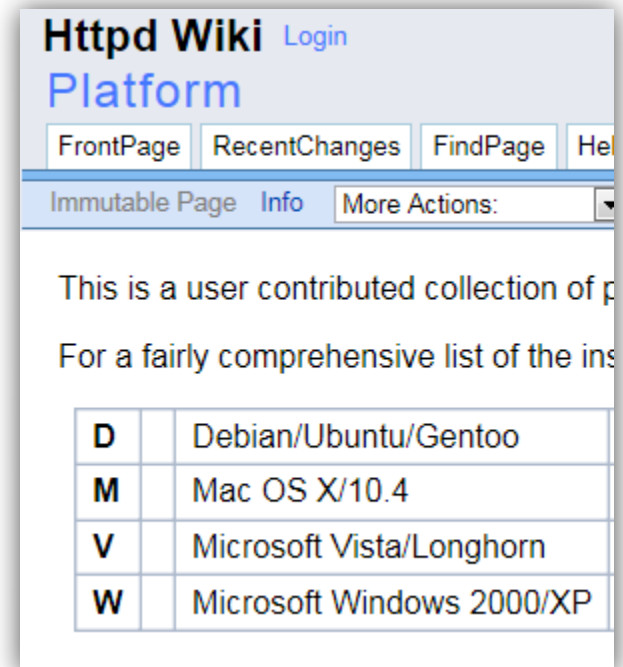
While you are learning about the **httpd.conf** file, it is wise to take two precautions:

1. Back up the original file to a secure location.
2. When making changes to the file, *comment out* the old settings instead of removing them or typing over them.

You can change the default configuration file by starting Apache with the **-f** configuration option, followed by the full path to an alternative configuration file, as shown in the following example:

```
/usr/local/apache2/bin/apachectl start -f  
/usr/local/apache2/conf/alt-httpd.conf
```

Open the file with your favorite text editor and follow along through the rest of this lesson.



# Apache Directives

httpd.conf is a plaintext file filled with settings commonly known as directives. These directives are unique keywords followed by the setting for the directive. For example, one of the directives in the configuration file follows:

**DocumentRoot** “/var/www/html”

This directive tells Apache that the Document Root, the main directory for content, is **/var/www/html**.

Note that all directives observe the following syntax:

*Directive Setting1 Setting2 Setting3...*

## Note

Directives and their options should be contained on one line. If you need to split a directive over two or more lines, use a backslash (\) at the end of the incomplete lines to signal Apache that the directive is continued on the following line.

Most directives are preceded by a comment line (or several) detailing the setting. For example, the *DocumentRoot* directive is preceded by the following comments:

```
#  
# DocumentRoot: The directory from which documents will be served. By default, all requests are taken  
# from this directory, but symbolic links and aliases may be used to point to other locations.  
#
```

# Common Apache Directives

**ServerRoot** The root directory in which Apache files (binaries, modules, and config files) are stored. (Argument: **directoryname**)

**PidFile** The file where the server should store its process ID when it starts. (Argument: **filename**)

**KeepAlive** Indicates whether the server should allow persistent connections (allow clients to send more than one request per connection). Setting this directive to *On* allows for better performance for connected clients. (Argument: *On|Off*)

**Listen** The address and port on which the server should listen for connections. This directive enables you to specify the port and particular IP on which the server should listen. If you have only one server (physical or virtual) and you want it to listen to all incoming traffic, specify the port only. (Argument: *ipaddress:port*)

**LoadModule** Loads the specified module. (Arguments: *status\_module modulepath/modulename*)

**ServerAdmin** The address for the Webmaster/administrator. This address is used on all server-generated pages. (Argument: *emailaddress*)

## Note:

All directive directory name arguments should be full paths to the specified directory or a relative path from the directory specified in the *ServerRoot* directive.

Likewise, all file name arguments need to specify the full path to the file or a relative path from the directory specified in the *ServerRoot* directive.



# Common Apache Directives (Contd)

**ServerName** The server's fully qualified host name. (Argument: fullyqualifieddomainname)

**DocumentRoot** The root directory for server content. (Argument: directoryname)

**UserDir** The directory to add to user directories when they are requested. For example, on Windows, the default *My Documents/My Website* is added to the user directory *C:\Documents and Settings\username\*. (Argument: directoryname)

**AccessFileName** The filename that the server should look for in each directory for additional access rules. (Argument: filename, usually .htaccess)

**TypesConfig** The file the server should use to determine MIME types. (Argument: filename)

**DefaultType** The default MIME type the server should use for a file if it is unable to determine otherwise. (Argument: MIMEType)

**HostnameLookups** Indicates whether the server should look up the hostname of each client or only log its IP address. Note that turning this option on can significantly increase the server load. (Argument: On|Off)

**ErrorLog** The location of the default error log. (Argument: filename)

# Common Apache Directives (Contd)

**LogLevel** The amount of information the server records to log files. The default value, `warn`, provides the mid-range of information. (Argument: `emerg|alert|crit|error|warn|notice|info|debug`)

**LogFormat** The format the server should use to record information in the log files. This directive's argument uses several placeholders and variables. See the documentation on the Apache Web site for explanations of each. (Argument: `logfileformat`)

**CustomLog** The main log file and the information to be stored in the file. This directive takes two arguments: the first is the file in which to store the log information and the second is the type of information to store. If the second argument is `common`, all access info (including virtual host files) will be logged here. The second argument can also be a format string for the log file to follow. (Arguments: `filename common|logfileformat`)

For more information on directives and their use, read the [Apache manual](#) online.

Review the Apache manual online for more information about directives.

## For Review

- [Directive reference](#)

# Access Directives

Apache has a very flexible access scheme, the details of which are covered in a future lesson. However, it is important to cover the rudimentary Apache access scheme while reviewing the **httpd.conf** file.

Apache has two basic access schemes: **allow** and **deny**. The allow scheme provides access to anything specified. Deny denies access to anything specified.

Typically, you specify which rules you want to run first, allow or deny. In most cases, you want to allow everyone and make exceptions to deny or deny everyone and make exceptions to allow. *Allow everyone* access scheme works well for a simple public server but does not create the most secure site because you must think of every possible instance you want to deny.

The *deny everyone* scheme is a very restrictive security model because you must think of every possible instance you want to allow.

Review the Apache manual online for more information about directives.

## For Review

- [Directive reference](#)

# Access Directives

The **Order** directive controls the basic model. The directive's syntax is as follows:

**Order allow,deny or deny,allow**

Individual **Allow** and **Deny** directives specify what to allow and what to deny. The syntax for the **Allow** and **Deny** directives is as follows (where **location** is one of those in the list):

Allow|Deny from location

Fully Qualified Domain Name (FQDN)

For example, **apache.org**

IP address

For example, **192.168.1.1**

A partial IP address (the first few octets)

For example, **192.168**

An IP address and subnet mask

For example, **192.168.1.0/255.255.255.0**

The text **all**

# Apache Security Configurations

As installed, the basic Apache security model is as follows:

```
Order allow,deny  
Allow from all
```

This security model means process the **Allow** rules first (**Allow from all**), and then the **Deny** rules. Because **Allow** is specified as **all**, only those hosts that match subsequent **Deny** directives are actually denied access. For example, later in the **httpd.conf** file, you have the following directives:

```
<Files ~ “^\.ht”>  
Order allow,deny  
Deny from all  
</Files>
```

These directives prohibit anyone from viewing files that begin with **.ht** (such files are typically Apache access or password files). In this case, the **Order** directive is not needed because of the earlier **Order** directive. However, it is good form to specify the order before every **Deny** or **Allow** directive to ensure that the **Deny** or **Allow** directive functions as intended.

# Apache Security Configurations

Unless your Apache server is behind a firewall or is otherwise inaccessible from the Internet, you probably want to lock it down with appropriate directives.

If you access the server from only one subnet, specify that subnet as the only access point. An example of this scheme would be the following:

**Order deny,allow**  
**Deny from all**  
**Allow from ip/netmask**

Simply change the default permissions for the document root and any other areas you want open or restricted.

## Required Reading

- [Apache Module Host](#)

# Apache Log Files

By default, Apache maintains two log files, **access.log** and **error.log**. These logs record successful client access and errors reported by the Apache server, respectively. The *LogLevel* and *LogFormat* directives govern the amount of information stored in each log file.

A typical access.log entry resembles the following:

```
192.168.10.25 - - [20/Apr/2008:22:01:25 -0500] "GET / HTTP/1.1" 200 1494  
192.168.10.25 - - [20/Apr/2008:22:01:25 -0500] "GET /apache_pb.gif HTTP/1.1" 200 2326
```

In the preceding case, a client from IP address 192.168.10.25 successfully made two requests on April 20 around 10:00 p.m. Both requests show status 200 (OK). The latter request was for a graphic, the Powered By Apache logo. (The request was for the Apache status page.)

A typical error.log entry resembles the following:

```
[Sat Apr 20 22:00:45 2003] [error] [client 192.168.10.16] client denied by server  
configuration: C:/Program Files/Apache Group/Apache2/htdocs/
```

In the preceding case, a client was denied access from IP address 192.168.10.16 because of a server configuration. Reviewing log files regularly can help you spot system errors, misconfigurations, and hacker attempts to compromise your server. Because each log contains separate information, you should analyze both logs concurrently.

# Apache Web Server Security

**Do not implement doubtful scripts.** Verify every script you put into place for vulnerabilities and security holes—especially if the script(s) accesses the raw file system.

**Ensure that scripts do not run as the root or administrator user.** If a security issue exists with a script, it would be disastrous to allow someone exploiting that issue access to the administrator account!

**Do not allow users of your web server to implement scripts without your express permission** (and check it yourself). Many scripts are available on the Internet and can be installed by simply copying them to the appropriate **cgi-bin** directory. However, most scripts that are freely available were not written with strict security in mind. Be sure to check for security holes before allowing users to implement their own scripts.

**Remove unneeded script directories from the file system and the Apache configuration file.** Most administrators recommend removing the **ScriptAlias cgi-bin** from the Apache configuration file and naming your script directories something other than **cgi-bin**, or at least not placing them so prominently at the root of every domain.





# Apache Security (Contd)

**Ensure that dangerous scripts have restricted access.** Using the methods described earlier in this session, protect script use by unwanted users by restricting access to the directories or to individual files.

**Keep your copies of the OS and Apache up-to-date.** Most operating systems have useful auto-update features—Windows has the Windows Update service, Red Hat Linux has the Red Hat Network. An operating system and Web server that are kept up-to-date are inherently more secure.

**Add only one feature to your site at a time, securing each new feature before adding another.** New web developers and system administrators are quick to add multiple features, lights, buzzers, and whistles to their site. However, each new feature has the potential to add another security vulnerability. It serves you well to ensure that the features you have are secure before adding more.



# Apache Security (Contd)

**Add only features you really need—and survey each carefully before adding.**

Many scripts, programs, and server add-ons are available for you to add features to your web server. However, each has the potential of adding more security holes. Make sure the features or programs you add come from a reliable source that updates the software frequently. Read reviews and newsgroup messages about the software before implementing it. As they say, an ounce of prevention is worth a pound of cure.

**Monitor your logs.** The easiest and cheapest thing to keeping your system safe is to properly and regularly check you logs. Look for entries that do not seem to fit the normal patterns, and make sure to follow up on the smallest of changes in your system operation and performance.



# Lesson Summary

In this lesson, you were to the Apache Web Server. You learned to install and configure a basic web server in Linux. Additionally, you learned to install Debian by performing a *netinstall*, configuring the web server, reviewing the http protocol, and discussing a few web services and their associated ports.

With a nearly 60% market share, it is almost certain you will encounter an Apache web server at some point in the future. Whether in a professional environment or on at home network, students should have the necessary skill sets to install and maintain an Apache web server after completing this lesson.



## Apache HTTP SERVER PROJECT

### The Number One HTTP Server On The Internet

The Apache HTTP Server Project is an effort to develop an efficient and extensible server that provides HTTP services

Apache httpd has been the most popular web server on the

The Apache HTTP Server ("httpd") is a project of [The Apache](#)

### Apache HTTP Server 2.2.17 Released

The Apache HTTP Server Project is proud to [announce](#) the

This version of httpd is a major release of the stable branch. Balancing, Graceful Shutdown support, Large File Support