26. Apache (httpd): Setting Up a Web Server

In This Chapter

Objectives

After reading this chapter you should be able to:

- Explain the purpose of Apache and describe the use of several available modules
- Configure an Apache server to listen on the network and provide simple content
- Customize names, addresses, ports, and the location of content, logs, and scripts of an Apache server
- Allow users to publish content from their home directories
- Configure virtual hosts to display different content
- Configure password-protected directory hierarchies, host-based access control, directory options, and SELinux Booleans to secure access to Web content

The World Wide Web (WWW or Web for short), is a collection of servers that hold material, called*content*, that (Web) browsers can display. Each of the servers on the Web is connected to the Internet, a network of networks (an *internetwork*). Much of the content on the Web is coded in HTML (Hypertext Markup Language, page 1254). *Hypertext*, the links you click on a Web page, allows browsers to display and react to links that point to other Web pages on the Internet.

Apache is the most popular Web server on the Internet. It is both robust and extensible. The ease with which you can install, configure, and run it in the Linux

environment makes it an obvious choice for publishing content on the World Wide Web. The Apache server and related projects are developed and maintained by the Apache Software Foundation (ASF), a not-for-profit corporation formed in June 1999. The ASF grew out of the Apache Group, which was established in 1995 to develop the Apache server.

This chapter starts by providing introductory information about Apache. Following this information is the JumpStart section, which describes the minimal steps needed to get Apache up and running. Next is "Filesystem Layout," which details where the various Apache files are located.

Configuration directives are a key part of Apache and are discussed starting on page 939. This section includes coverage of contexts and containers, two features/concepts that are critical to understanding Apache. The next section, which starts on page 962, covers advanced Apache configuration: redirects, content negotiation, directory indexing, virtual hosts, troubleshooting, and modules you can use with Apache, including CGI and SSL.

Introduction

Apache is a server that responds to requests from Web browsers, or *clients*, such as Firefox, Netscape,lynx, elinks, and Internet Explorer. When you enter the address of a Web page (a *URI*, page 1279) in a Web browser's location bar, the browser sends a request over the Internet to the (Apache) server at that address. In response, the server sends (serves) the requested content back to the browser. The browser then displays or plays the content, which might be a textual document, song, picture, video clip, or other information.

Content

Aside from add-on modules that can interact with the content, Apache looks only at the type of data it is sending so that it can specify the correct *MIME* (page 1261) type; otherwise it remains oblivious to the content itself. Server administration and content creation are two different aspects of bringing up a Web site. This chapter concentrates on setting up and running an Apache server; it spends little time discussing content creation.

Modules

Apache, like the Linux kernel, uses external modules to increase load-time flexibility and to allow parts of its code to be recompiled without recompiling the whole program. Rather than being part of the Apache binary, modules are stored as separate files that can be loaded when Apache is started.

Apache uses external modules, called dynamic shared objects (DSOs), for basic and advanced functions; there is not much to Apache without these modules. Apache also uses modules to extend its functionality. For example, modules can process scripts written in Perl, PHP, Python, and other languages; use several different methods to authenticate users; facilitate publishing content; and process nontextual content, such as audio. The list of modules written by the ASF and third-party developers is constantly growing. For more information refer to "Modules" on page 968.

Graphical configuration

You can use the HTTP Server Configuration window (**system-config-httpd** package) to configure Apache graphically. Open this window by giving the command **system-config-httpd** from an Enter a Command window (ALT-F2) or a terminal emulator. Click **Help** in this window for more information.

More Information

Local

Apache documentation: With Apache running and the **httpd-manual** package installed, point a browser at ***server*/manual**. On the server system, substitute **localhost** for ***server*;** from a remote system, substitute the name or IP address of the Apache server for ***server***. If Apache is not running, point a browser on the server system only at **/usr/share/httpd/manual/index.html**.

apachectl and httpd man pages

Apache directives: ***server*/manual/mod/directives.html**

SSI directives: ***server*/manual/howto/ssi.html**

Web

Apache documentation: httpd.apache.org/docs/2.4

Apache directives: httpd.apache.org/docs/2.4/mod/directives.html

Apache Software Foundation (newsletters, mailing lists, projects, module registry, and more):www.apache.org

webalizer: www.webalizer.org

awstats: awstats.sourceforge.net

mod_perl: perl.apache.org

mod_php: www.php.net

mod_python: www.modpython.org; see **mod_wsgi**

mod_ssl: www.modssl.org

mod_wsgi: wsgi.readthedocs.org/en/latest, code.google.com/p/modwsgi

MRTG: oss.oetiker.ch/mrtg

SNMP: net-snmp.sourceforge.net

SSI directives: httpd.apache.org/docs/2.4/howto/ssi.html

Notes

Terms: Apache and **httpd**

Apache is the name of a server that serves HTTP and other content. The Apache daemon is named**httpd** because it is an HTTP server daemon. This chapter uses the terms *Apache* and ***httpd***interchangeably.

Terms: server and process

An Apache *server* is the same thing as an Apache *process.* An Apache child process exists to handle incoming client requests; hence it is referred to as a server.

Firewall

An Apache server normally uses TCP port 80; a secure server uses TCP port 443. If the server is running a firewall or is behind a firewall, you need to open one or both of these ports. Give the following commands to open the ports each time the system boots (permanently) and on the running system; see page 906 for information on firewall-cmd.

Click here to view code image

```
# firewall-cmd --add-port=80/tcp
# firewall-cmd --permanent --add-port=80/tcp
# firewall-cmd --add-port=443/tcp
# firewall-cmd --permanent --add-port=443/tcp
```

SELinux

When SELinux is set to use a targeted policy, **httpd** is protected by SELinux. You can disable this protection if necessary. For more information refer to "Setting the Targeted Policy with system-config-selinux" on page 475.

Running with **root** privileges

Because Apache serves content on privileged ports, you must start it running with **root** privileges. For security reasons, Fedora/RHEL sets up Apache to spawn processes that run as the user and group**apache**.

Locale

The **httpd** daemon starts using the C locale by default. You can modify this behavior—for example, to use the configured system locale—by setting the **HTTPD_LANG** variable in the**/etc/sysconfig/httpd** file.

Document root

The root of the directory hierarchy that Apache serves content from is called the *document root* and is controlled by the DocumentRoot directive (page 942). This directive defines a directory on the server that maps to /. This directory appears to users who are browsing a Web site as the root directory. As distributed by Fedora/RHEL, the document root is **/var/www/html**.

Modifying content

With the default Fedora/RHEL configuration of Apache, only a user working with **root** privileges can add or modify content in **/var/www/html**. To avoid having people work with **root** privileges when they are manipulating content, create a group (**webwork**, for example), put people who need to work with Web content in this group, and make the directory hierarchy starting at **/var/www/html** (or another document root) writable by that group. In addition, if you give the directory hierarchy setgid permission, all new files created within this hierarchy will belong to the group, which facilitates sharing files. The first three commands below add the new group, change the mode of the document root to setgid, and change the group that the document root belongs to. The last command adds ***username*** to the **webwork** group; you must repeat this command for each user you want to add to the group.

Click here to view code image

```
# groupadd webwork
# chmod g+rws /var/www/html
# chown :webwork /var/www/html

# usermod -aG webwork username
```

See page 600 for more information about working with groups.

Version

Fedora/RHEL runs Apache version 2.4.

Running an Apache Web Server

This section explains how to install, test, and configure a basic Web server.

Prerequisites

Group installation

Give the following command to install the Web Server group of packages. In addition to **httpd**, this command installs support for PHP, Perl, and Python; Apache documentation; encrypted content using HTTPS protocol; and database-driven Web sites.

Click here to view code image

```
# yum groupinstall "Web Server"
```

Minimal installation

Install the following packages:

- **httpd**
- **apr** (Apache portable runtime; installed with **httpd**)
- **apr-util** (installed with **httpd**)

Tip: Before you set up an Apache server

Before you configure an Apache server, set up a static IP address for the server system. See "Configuring a Static IP Address for a Wired NIC" on page 641 for instructions.

Enable and start **httpd**

Run systemctl to cause the **httpd** service to start each time the system enters multiuser mode and then start the **httpd** service. Use the systemctl **status** command to make sure the service is running.

Click here to view code image

```
# systemctl enable httpd.service
# systemctl start httpd.service
```

After modifying Apache configuration files, give the second command again, replacing **start** with**reload** to cause Apache to reread those files.

Optional packages

You might want to install the following optional packages:

• **httpd-manual**—The Apache manual; see "Local" on page 933 for more information

• **webalizer**—Web server log analyzer (page 975)

• **awstats**—Web server log analyzer

• **mod_perl**—Embedded Perl scripting language

• **mod_wsgi**—Embedded Python scripting language

• **mod_ssl**—Secure Sockets Layer extension (page 970)

• **php**—Includes embedded PHP scripting language, with IMAP and LDAP support

• **mrtg**—MRTG traffic monitor (page 975)

• **net-snmp** and **net-snmp-utils**—SNMP, required for MRTG (page 975)

JumpStart: Getting Apache Up and Running

To get Apache up and running, modify the **/etc/httpd/conf/httpd.conf** configuration file as described in this section. "Directives You Might Want to Modify as You Get Started" on page 940explains more about this file and explores other changes you might want to make to it.

Modifying the httpd.conf Configuration File

Apache runs as installed, but it is a good idea to add/modify the three lines of the**/etc/httpd/conf/httpd.conf** configuration file described in this section before starting Apache. If you do not add/modify these lines, Apache will assign values that might not work on the server.

ServerName

The ServerName line establishes a name for the server. Add one of the following lines to **httpd.conf** to set the name of the server to the domain name of the server or, if you do not have a domain name, to the IP address of the server. Add the line just below the commented-out ServerName line in the**httpd.conf** file. The commented-out line specifies port 80 (**:80** at the end of the line); port 80 is the default so you do not need to specify it.

ServerName **example.com**

or

ServerName **IP_address**

where ***example.com*** is the domain name of the server and ***IP_address*** is the IP address of the server. If you are not connected to a network, you can use the **localhost** address, 127.0.0.1, so you can start the server and experiment with it using a browser on the server system. See page 942 for more information on the ServerName directive.

ServerAdmin and ServerSignature

When a client has trouble getting information from a server, the server typically displays an error page that identifies the problem. For example, when Apache cannot find a requested page, it displays an error page that says **Error 404: Not Found**. The ServerSignature directive (page 956) enables you to add a signature line to server-generated pages, including error pages. You can turn the signature line on or off using this directive. You can also specify that the signature line include a **mailto:** link that the user can click to send mail to the server's administrator. This link appears as the domain name the user called in the browser. The ServerAdmin (page 941) directive specifies the email address that the server sends mail to when a user clicks the link on an error page. Change the first of these two lines and add the second in **httpd.conf**.

ServerAdmin **email_address**

ServerSignature EMail

where ***email_address*** is the email address of the person who needs to know when people are having trouble using the server. Make sure that someone checks this email account frequently. But also see the tip "ServerAdmin attracts spam" on page 942.

It can make system administration much easier if you use a role alias (e.g.,**webmaster@example.com**) instead of a specific username (e.g., **max@example.com**) as an***email_address***. See the discussion of email aliases on page 746.

After making changes to **httpd.conf**, start or restart **httpd** as explained on page 935.

Testing Apache

Once you start the **httpd** daemon, you can confirm that Apache is working correctly by pointing a browser on the local (server) system to **http://localhost/**. From a remote system, point a browser to**http://** followed by the ServerName you specified in the previous section. If you are displaying a page from a system other than the server, the local system must know how to resolve a domain name you

enter (e.g., by using DNS or the **/etc/hosts** file). For example, you might use either of these URI formats: **http://192.168.0.16** or **http://example.org**.

The browser should display the Fedora/RHEL test page, which is actually an error page that says there is no content (next). If the server is behind a firewall, open TCP port 80 (page 934). If Apache is not working, see "Troubleshooting" on page 967.

Fedora/RHEL test page

When you install Apache, there is no **index.html** file in **/var/www/html**; when you point a browser at the local Web server, Apache generates Error 403 (Forbidden), which returns the Fedora/RHEL test page. The mechanism by which this page is returned is convoluted: The Fedora/RHEL **httpd.conf** file holds an Include directive that includes all files with a filename extension of **.conf** that reside in the**conf.d** directory, that is in the ServerRoot directory (**/etc/httpd**; page 955). The**/etc/httpd/conf.d/welcome.conf** file contains an ErrorDocument 403 directive (page 953) that redirects users to **/usr/share/httpd/noindex/index.html**. This file is the Fedora/RHEL test page that confirms the server is working but there is no content to display. Apache will no longer display this page when you put an **index.html** file in **/var/www/html**.

Putting Content in Place

Place the content you want Apache to serve in **/var/www/html**. Apache automatically displays the file named **index.html** in this directory. Working with **root** privileges (or as a member of the group you set up for this purpose [e.g., **webwork**]), create such a page:

Click here to view code image

```
# cat /var/www/html/index.html
<html><body><p>This is <i>my</i> test
page.</p></body></html>
```

After you create this file, either refresh the page on the browser or restart the browser and point it at the server. The browser should display the page you created.

Filesystem Layout

This section lists the locations and uses of files you can work with to configure Apache and serve Web pages.

Binaries, scripts, and modules

The Apache server and related binary files are kept in several directories:

/usr/sbin/httpd—The Apache server (daemon).

/usr/sbin/apachectl—Starts and stops Apache.

/usr/bin/htpasswd—Creates and maintains the password files used by the Apache authentication module (page 972).

/usr/sbin/rotatelogs—Rotates Apache log files so that these files do not get too large. See logrotate (page 618) for more information.

/etc/httpd/modules—Holds module binaries. Two of the most frequently used module binary files are **mod_perl.so** (**mod_perl** package) and **mod_wsgi.so** (**mod_wsgi** package). This directory is a symbolic link to **/usr/lib64/httpd/modules** (page 968).

Configuration files

By default, **/etc/httpd** is the ServerRoot (page 955); all Apache configuration files are kept in this directory hierarchy, specifically in the **/etc/httpd/conf** and **/etc/httpd/conf.d** directories:

/etc/httpd/conf/httpd.conf—Holds configuration directives. This file is the main Apache configuration file. The discussion of configuration directives starts on the next page.

/etc/httpd/conf/magic—Provides *MIME* (page 1261) file type identification (the MIME hints file). It is not normally changed. See *magic number* (page 1259) for more information.

/etc/httpd/conf.d—Holds configuration files.

/etc/pki/tls/certs—Holds files and directories used by **mod_ssl** (page 970).

Logs

Log files are kept in **/var/log/httpd** (there is a symbolic link at **/etc/httpd/logs**):

/var/log/httpd/access_log—Logs requests made to the server.

/var/log/httpd/error_log—Logs request and runtime server errors.

/var/log/httpd/ssl_*_log—Holds **mod_ssl** logs.

Web documents

Web documents (including the Web pages displayed by client browsers), custom error messages, and CGI scripts are kept in **/var/www** by default:

/usr/share/httpd/error—Holds error documents in several languages. By default, Fedora/RHEL displays hardcoded error messages. See ErrorDocument (page 953).

/usr/share/httpd/icons—Holds icons used to display directory entries. This directory is aliased to**/icons/** in the **autoindex.conf** file.

/usr/share/httpd/manual—Holds the Apache Server Manual. Present only if the **httpd-manual**package is installed. This directory is aliased to **/manual/**.

/var/www/cgi-bin—Holds CGI scripts (page 969).

Document root

By default, the document root (page 934) is **/var/www/html**. You can change this location using the DocumentRoot directive (page 942).

.htaccess files

A **.htaccess** file contains configuration directives and can appear in any directory in the document root hierarchy. The location of a **.htaccess** file is critical: The directives in a **.htaccess** file apply to all files in the hierarchy rooted at the directory that holds the **.htaccess** file. You must use an AllowOverride directive to cause Apache to examine **.htaccess** files and process directives in those files. This protection is duplicated and enhanced in the **httpd.conf** file distributed by Fedora/RHEL, where a directive instructs Apache not to serve files whose names start with **.ht**. Because of this directive, Apache does not serve **.htaccess** files (nor does it serve **.htpassword** files).

Configuration Directives

Configuration directives, or simply *directives*, are lines in a configuration file that control some aspect of how Apache functions. A configuration directive is composed of a keyword followed by one or more arguments separated by SPACEs. For example, the following configuration directive sets **Timeout** to 300 (seconds):

```
Timeout 300
```

You must enclose arguments that contain SPACEs within double quotation marks. Keywords are not case sensitive, but arguments (pathnames, filenames, and so on) often are.

httpd.conf

The main file that holds Apache configuration directives is, by default, **/etc/httpd/conf/httpd.conf**. This file holds global directives that affect all content served by Apache. An Include directive (page 957) within **httpd.conf** can incorporate the contents of another file as though it were part of **httpd.conf**.

.htaccess

Local directives can appear in **.htaccess** files. A **.htaccess** file can appear in any directory within the document root hierarchy; it affects files in the directory hierarchy rooted at the directory it appears in.

Pathnames

When you specify an absolute pathname in a configuration directive, the directive uses that pathname without modifying it. When you specify a relative pathname, such as a simple filename or the name of a directory, Apache prepends to that name the value specified by the ServerRoot (page 955) directive (**/etc/httpd** by default).

Prefork MPM

The **/etc/httpd/conf.modules.d/00-mpm.conf** file loads the **prefork** multiprocessing module (MPM, page 974). You can edit this file to load the **worker** or **event** module instead. (The **worker**module does not work with PHP.) This chapter assumes you have loaded the **prefork** module.

Directives You Might Want to Modify as You Get Started

When it starts, Apache reads the **/etc/httpd/conf/httpd.conf** configuration file (by default) for instructions governing every aspect of how Apache runs and serves content. The **httpd.conf** file shipped with Fedora/RHEL is more than 350 lines long.

This section details some directives you might want to change as you start using Apache. You can add/modify each of the following directives in **httpd.conf**. The **Context** line in each explanation tells you which locations the directives can appear in; contexts are explained on page 945. The section titled "Advanced Configuration Directives" on page 949 describes more directives.

Listen

Specifies the IP address and port that Apache listens for requests on.

*Listen [***IP-address***:]***portnumber**

where ***IP-address*** is the IP address that Apache listens on and ***portnumber*** is the number of the port that Apache listens on for the given ***IP-address***. When ***IP-address*** is absent or is set to 0.0.0.0, Apache listens on all network interfaces. At least one Listen directive must appear in the configuration files or Apache will not work.

The following minimal directive from the **httpd.conf** file listens for requests on all network interfaces on port 80:

```
Listen 80
```

The next directive changes the port from the default value of 80 to 8080:

```
Listen 8080
```

When you specify a port other than 80, each request to the server must include a port number (as in**www.example.org:8080**) or the kernel will return a **Connection Refused** message. Use multiple Listen directives to have Apache listen on multiple IP addresses and ports. For example:

```
Listen 80
Listen 192.168.1.1:8080
Listen 192.168.1.2:443
```

accepts connections on all network interfaces on port 80, on 192.168.1.1 on port 8080, and on 192.168.1.2 on port 443.

Context: **server config**
Default: none (Apache will not start without this directive.)
Fedora/RHEL: Listen 80

Redirect

Tells the client to fetch a requested resource from a different, specified location.

*Redirect [***status***]* **requested-path** *[***new-URI***]*

where ***status*** is the status that Apache returns along with the redirect. If you omit ***status***, Apache assumes **temp**. The ***status*** can be an Apache error code in the range 300–399 or one of the following:

permanent	Returns status 301 (the resource has moved permanently)
temp	Returns status 302 (the resource has moved temporarily)
seeother	Returns status 303 (the resource has been replaced)
gone	Returns status 410 (the resource has been removed—does not take a ***new-URI*** argument)

The ***requested-path*** is the absolute pathname of the ordinary file or directory that Apache is to redirect requests for. Apache redirects all requests that start with the absolute pathname specified by***requested-path*** (see the example below). Use RedirectMatch (discussed next) if you want to use a regular expression in this argument.

The ***new-URI*** is the URI that Apache redirects requests to. If the ***new-URI*** starts with a slash (/) and not **http://**, **ftp://**, or a similar prefix, Apache uses the same prefix it was called with. Most Redirect directives require a ***new-URI*** argument.

A request must match all segments of the ***requested-path*** argument. Assume the following directive:

Click here to view code image

```
Redirect /www.example.com/pictures
http://pictures.example.com/
```

Apache will redirect a request for **http://www.example.com/pictures/mom.jpg** to**http://pictures.example.com/mom.jpg** but, because the final segment does not match, it will not redirect a request for **http://www.example.com/pictures_mom.jpg**.

Contexts: **server config**, **virtual host**, **directory**, **.htaccess**
Default: none
Fedora/RHEL: none

RedirectMatch

Tells the client to fetch a requested resource from a different location specified by a regular expression.

*RedirectMatch [***status***]* **requested-path-re** *[***new-URI***]*

This directive is the same as Redirect (discussed on the previous page), except you can use a regular expression (Appendix A) in ***requested-path-re***.

Contexts: **server config**, **virtual host**, **directory**, **.htaccess**
Default: none
Fedora/RHEL: none

ServerAdmin

Sets the email address used in **mailto** *links on error pages.*

ServerAdmin **email-address**

where ***email-address*** is the email address of the person who is responsible for managing the Web content. Apache includes this address as a link on Apache-generated error pages. However, Fedora/RHEL sets ServerSignature (page 956) to **On**, which causes Apache to display information about the server—rather than a link to an email address—on error pages. If you want Apache to display the link on

error pages, set ServerSignature (page 956) to **EMail**. Make sure ***email-address*** points to an email account that someone checks frequently. Users can use this address to get help with the Web site or to inform the administrator of problems. There is no default value for ServerAdmin; if you do not use this directive, and ServerSignature is set to **EMail**, the **mailto:** link on error pages displays**[no address given]**.

Security: ServerAdmin attracts spam

The email address you put in ServerAdmin often attracts spam. Use a spam-guarded address such as **"mgs at sobell dot com"** (you must use the quotation marks) or use a custom error page to point to a Web page with a form for sending mail to the right person.

You can use a role alias such as **webmaster** at your domain and use a mail alias to forward mail that is sent to **webmaster** to the person who is responsible for maintaining the Web site. See the discussion of mail aliases on page 746.

Contexts: **server config**, **virtual host**
Default: none
Fedora/RHEL: **root@localhost**

ServerName

Specifies the server's name and the port it listens on.

ServerName **FQDN** *[:***port***]*

where ***FQDN*** is the fully qualified domain name or IP address of the server and ***port*** is the optional port number Apache listens on. The domain name of the server must be able to be resolved (by DNS or**/etc/hosts**) and might differ from the hostname of the system running the server. If you do not specify a ServerName, Apache performs a DNS reverse name resolution (page 862) on the system's IP address and assigns that value to ServerName. If the reverse lookup fails, Apache assigns the system's IP address to ServerName.

Fedora/RHEL provides the following ServerName template in the **httpd**.**conf** file:

```
#ServerName www.example.com:80
```

Copy this line, remove the **#**, and substitute the FQDN or IP address of the server for**www.example.com**. You can omit the **:80** because it specifies the default port. Change the **80** to the port number Apache listens on if it does not listen on port 80.

The ports specified by ServerName and Listen (page 940) must be the same if you want the FQDN specified by ServerName to be tied to the IP address specified by the Listen directive.

Apache uses ServerName to construct a URI when it redirects a client (page 962). See also UseCanonicalName (page 951).

Contexts: **server config**, **virtual host**
Default: none
Fedora/RHEL: none

DocumentRoot

Points to the root of the directory hierarchy that holds the server's content.

DocumentRoot **dirname**

where ***dirname*** is the absolute pathname of the directory at the root of the directory hierarchy that holds the content Apache serves. Do not use a trailing slash. The FHS (page 189) specifies **/srv** as the top-level directory for this purpose. You can put the document root wherever you like, as long as the user **apache** has read access to the ordinary files and execute access to the directory files in the directory hierarchy. Access control includes both discretionary access controls (chmod, including ACLs for the user **apache**) and mandatory access controls (selinux for **httpd**). The following directive puts the document root at **/srv/www**:

```
DocumentRoot /srv/www
```

Contexts: **server config**, **virtual host**
Default: **/usr/local/apache/htdocs**
Fedora/RHEL: **/var/www/html**

Tip: You might want to change the location of the document root

By design, the **/var** directory hierarchy (page 41) is designed to hold data that changes frequently, which is not usually the case with static Web site data. Data in such a partition is more likely to become corrupted than data in a partition that holds static data. Thus you might want to place Web site data in a different partition. Good candidates for the document root include **/usr/local** (holds site-specific data), **/home/www** (the home directory is frequently backed up while other directories might not be), and **/srv/www** (specified by FHS).

UserDir

Allows users to publish content from their home directories.

UserDir **dirname** | *disabled* | *enabled* **user-list**

where ***dirname*** is the name of a directory that, if it appears in a local user's home directory, Apache publishes to the Web. When you do not specify

a ***dirname***, Apache publishes content in**~/public_html**. The *disabled* keyword prevents content from being published from users' home directories; *enabled* causes content to be published from the home directories of users specified in theSPACE-separated ***user-list***.

Apache can combine the effects of multiple UserDir directives. For example, assume you use the following directives:

Click here to view code image

```
UserDir disabled
UserDir enabled user1 user2 user3
UserDir web
```

The first directive turns off user publishing for all users. The second directive enables user publishing for three users. The third directive makes **web** the name of the directory that, if it appears in one of the specified users' home directories, Apache publishes to the Web.

To cause a browser to display the content published by a user, specify in the location bar the name of the Web site followed by a /~ and the user's username. For example, if Sam published content in the**public_html** directory in his home directory and the URI of the Web site was **www.example.com**, you would enter **http://www.example.com/~sam** to display Sam's Web page. To display a user's Web page, Apache must have execute permission (as user **apache**) for the user's home directory and the directory holding the content, and read permission for the content files.

Fedora/RHEL provides the following ServerName directive and template in the**/etc/httpd/conf.d/userdir**.**conf** file. Because the statements are in an IfModule container (page947), they are executed only if the **mod_userdir** module is loaded, which it is by default. These directives are surrounded by many comments.

```
<IfModule mod_userdir.c>
    UserDir disabled
    #UserDir public_html
</IfModule>
```

Put a hashmark (#) in front of the first line and remove the hashmark from the second line to allow users to publish content from directories named **public_html** in their home directories.

Contexts: **server config**, **virtual host**
Default: none
Fedora/RHEL: disabled

DirectoryIndex

Specifies which file Apache serves when a user requests a directory.

DirectoryIndex **filename** *[***filename** *...]*

where ***filename*** is the name of the file that Apache serves.

This directive specifies a list of filenames. When a client requests a directory, Apache attempts to find a file in the specified directory whose name matches a file in the list. When Apache finds a match, it returns that file. When this directive is absent or when none of the files specified by this directive exists in the specified directory, Apache displays a directory listing as specified by the IndexOptions directive (page 954).

Fedora/RHEL provides the following DirectoryIndex directive in the **httpd.conf** file:

```
DirectoryIndex index.html
```

This directive causes Apache to search the specified directory and return the file named **index.html**, which is the name of the standard, default HTML document. If you supply CGI documents, you might want to add the **index.cgi** value to this directive. The name **index** is standard but arbitrary.

A **.var** filename extension denotes a content-negotiated document that allows Apache to serve documents in one of several languages as specified by the client. If you are providing content in different languages, you can add this filename extension to the DirectoryIndex directive. For more information refer to "Type Maps" on page 963.

Contexts: **server config**, **virtual host**
Default: **index.html**
Fedora/RHEL: **index.html**

Contexts and Containers

To make it flexible and easy to customize, Apache uses configuration directives, contexts, and containers. Some configuration directives are covered in the previous section. This section discusses contexts and containers, which are critical to managing an Apache server.

Contexts

Four locations, called *contexts*, define where configuration directives can appear. This chapter marks each configuration directive to indicate which context(s) it can appear in. Table 26-1 describes each of these contexts.

Context	Location(s) directives can appear
server config	In the **httpd.conf** file or included files only, but not inside <VirtualHost> or <Directory> containers (next section) unless so marked
virtual host	Inside <VirtualHost> containers in the **httpd.conf** file or included files only
directory	Inside <Directory>, <Location>, and <Files> containers in the **httpd.conf** file or included files only
.htaccess	In **.htaccess** files (page 939) only

Table 26-1 Contexts

Directives in files incorporated by means of an Include directive (page 957) are part of the context they are included in and must be allowed in that context.

Putting a directive in the wrong context generates a configuration error and can cause Apache not to serve content correctly or not to start.

Containers

Containers, or *special directives*, are directives that group other directives. Containers are delimited by XML-style tags. Three examples are shown here:

```
<Directory> ... </Directory>

<Location> ... </Location>

<VirtualHost> ... </VirtualHost>
```

Look in **httpd.conf** for examples of containers. Like other directives, containers are limited to use within specified contexts. This section describes some of the more frequently used containers.

<Directory>

Applies directives to all directories within the specified directory hierarchy.

<Directory **directory***> ... </Directory>*

where ***directory*** is an absolute pathname specifying the root of the directory hierarchy that holds the directories the directives in the container apply to. The ***directory*** can include wildcards; a ***** does not match a /.

A <Directory> container provides the same functionality as a **.htaccess** file (page 939). While an administrator can use a <Directory> container in Apache configuration files, regular users cannot. Regular users must use **.htaccess** files to control access to their own directories.

The directives in the <Directory> container shown in the following example apply to the**/var/www/html/corp** directory hierarchy. The Deny directive denies access to all clients, the Allow directive grants clients from the 192.168.10. subnet access, and the AllowOverride directive (page 960) enables Apache to process directives in **.htaccess** files in the hierarchy:

Click here to view code image

```
<Directory /var/www/html/corp>
    Deny from all
    Allow from 192.168.10.
    AllowOverride All
</Directory>
```

The following <Directory> container appears in the **httpd.conf** file and sets up a restrictive environment for the entire server filesystem (specified by /):

```
<Directory />
    AllowOverride None
    Require all denied
</Directory>
```

The AllowOverride directive causes Apache not to process directives in **.htaccess** files. The Require directive (page 961) denies all users access to the entire server filesystem directory. You must explicitly enable less restrictive options if you want them, but doing so can expose the server filesystem and compromise system security. This restrictive <Directory> container is followed immediately in**httpd.conf** by one that sets up less restrictive options for **/var/www**.

```
<Directory "/var/www">
    AllowOverride None
    Require all granted
</Directory>
```

The previous <Directory> container is followed by one that sets up options for the DocumentRoot (**/var/www/html**); this container affects all content. The code in **httpd.conf** is interspersed with many comments. Without the comments it looks like this:

Click here to view code image

```
<Directory "/var/www/html">
    Options Indexes FollowSymLinks
    AllowOverride None
    Require all granted
</Directory>
```

The Indexes option in the Options directive allows Apache to display directory listings and the FollowSymLinks option causes Apache to follow symbolic links. This container is less restrictive than the preceding one, although it still does not allow Apache to follow directives in **.htaccess** files.

Contexts: **server config**, **virtual host**

<Files>

Applies directives to specified ordinary files.

<Files **directory***> ... </Files>*

where ***directory*** is an absolute pathname specifying the root of the directory hierarchy that holds the ordinary files the directives in the container apply to. The ***directory*** can include wildcards; a ***** does not match a /. This container is similar to <Directory> but applies to ordinary files rather than to directories.

The following directive, from the Fedora/RHEL **httpd.conf** file, denies access to all files whose filenames start with **.ht**, meaning that Apache will not serve the **.htaccess** and **.htpasswd** files.

```
<Files ".ht*">
    Require all denied
</Files>
```

Contexts: **server config**, **virtual host**, **directory**, **.htaccess**

<IfModule>

Applies directives if a specified module is loaded.

*<IfModule [!]***module-name***> ... </IfModule>*

where ***module-name*** is the name of the module (page 968) Apache tests for. Apache executes the directives in this container if ***module-name*** is loaded or with **!** if ***module-name*** is not loaded.

Apache will not start if you specify a configuration directive that is specific to a module that is not loaded.

The following <IfModule> container, which is located in the Fedora/RHEL **httpd.conf** file, depends on the **mime_magic_module** module being loaded. If this module is loaded, Apache runs the MIMEMagicFile directive, which tells the **mime_magic_module** where its hints file is located.

```
<IfModule mime_magic_module>
    MIMEMagicFile conf/magic
</IfModule>
```

Contexts: **server config**, **virtual host**, **directory**, **.htaccess**

<Limit>

Limits access-control directives to specified HTTP methods.

<Limit **method** *[***method***] ... > ... </Limit>*

where ***method*** is an HTTP method. An HTTP method specifies which action is to be performed on a URI. The most frequently used methods are GET, PUT, POST, and OPTIONS; method names are case sensitive. GET (the default method) sends any data indicated by the URI. PUT stores data from the body section of the communication at the specified URI. POST creates a new document containing the body of the request at the specified URI. OPTIONS requests information about the capabilities of the server.

The <Limit> container binds a group of access-control directives to specified HTTP methods: Only methods named by this container are affected by this group of directives.

The following example disables HTTP uploads (PUTs) from systems that are not in a subdomain of**example.com**:

```
<Limit PUT>
    order deny,allow
    deny from all
    allow from .example.com
</Limit>
```

Contexts: **server config**, **virtual host**, **directory**, **.htaccess**

Caution: Use <LimitExcept> instead of <Limit>

It is safer to use the <LimitExcept> container than to use the <Limit> container, as the former protects against arbitrary methods. When you use <Limit>, you must be careful to name explicitly all possible methods that the group of directives could affect.

<LimitExcept>

Limits access-control directives to all except specified HTTP methods.

<LimitExcept **method** *[***method***] ... > ... </LimitExcept>*

where ***method*** is an HTTP method. See <Limit> for a discussion of methods.

This container causes a group of access-control directives *not* to be bound to specified HTTP methods. Thus methods *not* named in <LimitExcept> are affected by this group of directives.

The access-control directives within the following <LimitExcept> container affect HTTP methods other than GET, POST, and OPTIONS. You could put this container in a <Directory> container to limit its scope:

```
<LimitExcept GET POST OPTIONS>
    Order deny,allow
    Deny from all
</LimitExcept>
```

Contexts: **server config**, **virtual host**, **directory**, **.htaccess**

<Location>

Applies directives to specified URIs.

<Location **URI***> ... </Location>*

where ***URI*** points to content; it specifies a file or the root of the directory hierarchy that the directives in the container apply to. While the <Directory> container points within the local filesystem, <Location> points outside the local filesystem.
The ***URI*** can include wildcards; a ***** does not match a /.

The following <Location> container limits access to **http://*server*/pop** to clients from the**example.net** domain, where ***server*** is the FQDN of the server:

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

```
    Allow from .example.net
</Location>
```

Contexts: **server config**, **virtual host**

Caution: Use <Location> with care

Use this powerful container with care. Do not use it to replace the <Directory> container: When several URIs point to the same location in a filesystem, a client might be able to circumvent the desired access control by using a URI not specified by this container.

<LocationMatch>

Applies directives to URIs specified by a regular expression.

<LocationMatch **regexp***> ... </LocationMatch>*

where ***regexp*** is a regular expression that matches one or more URIs. This container works the same way as <Location>, except that it applies to any URIs that ***regexp*** matches.

Contexts: **server config**, **virtual host**

<VirtualHost>

Applies directives to a specified virtual host.

<VirtualHost **addr***[:***port***] [***addr***[:***port***]] ... > ... </VirtualHost>*

where ***addr*** is the IP address (or FQDN, although it is not recommended) of the virtual host (or ***** to represent all addresses) and ***port*** is the port that Apache listens on for the virtual host. This directive does not control which addresses and ports Apache listens on; use a Listen directive (page 940) for that purpose. This container holds commands that Apache applies to a virtual host. For more information refer to "Virtual Hosts" on page 965.

Context: **server config**

Advanced Configuration Directives

This section discusses configuration directives that you might add to **httpd.conf** after you gain some experience using Apache. The Fedora/RHEL configuration files include the directives in this section that specify a value for Fedora/RHEL in addition to a default value.

Directives That Control Processes

MaxRequestWorkers (was MaxClients)

Specifies the maximum number of child processes.

MaxRequestWorkers **num**

where ***num*** is the maximum number of child processes (servers) Apache runs at one time, including idle processes and processes serving requests. When Apache is running ***num*** processes and there are no idle processes, Apache issues **Server too busy** errors to new connections; it does not start new child processes.

Context: **server config**
Default: 256

MaxConnectionsPerChild (was MaxRequestsPerChild)

Specifies the maximum number of requests a child process can serve.

MaxConnectionsPerChild **num**

where ***num*** is the maximum number of requests a child process (server) can serve during its lifetime. After a child process serves ***num*** requests, it does not process any more requests but dies after it finishes processing its current requests. Apache can start another child process to replace the one that dies. Additional requests are processed by other processes from the server pool.

Set ***num*** to 0 to not set a limit on the number of requests a child can process, except for the effects of MinSpareServers. By limiting the lives of processes, this directive can prevent memory leaks from consuming too much system memory. However, setting MaxConnectionsPerChild to a too-small value can hurt performance by causing Apache to create new child servers constantly.

Context: **server config**
Default: 0

MaxSpareServers

Specifies the maximum number of idle processes.

MaxSpareServers **num**

where ***num*** is the maximum number of idle processes (servers) Apache keeps running to serve requests as they come in. Do not set this number too high, as each process consumes system resources.

Context: **server config**
Default: 10

MinSpareServers

Specifies the minimum number of idle processes.

MinSpareServers **num**

where ***num*** is the minimum number of idle processes (servers) Apache keeps running to serve requests as they come in. More idle processes occupy more computer resources; increase this value for busy sites only.

Context: **server config**
Default: 5

StartServers

Specifies the number of child processes that Apache starts with.

StartServers **num**

where ***num*** is the number of child processes (servers) that Apache starts when it is brought up. This value is significant only when Apache starts; MinSpareServers and MaxSpareServers control the number of idle processes once Apache is up and running. Starting Apache with multiple servers ensures that a pool of servers is waiting to serve requests immediately.

Context: **server config**
Default: dynamically controlled based on load

Networking Directives

HostnameLookups

Specifies whether Apache puts a client's hostname or its IP address in the logs.

HostnameLookups On | Off | Double

On: Performs DNS reverse name resolution (page 862) to determine the hostname of each client for logging purposes.

Off: Logs each client's IP address.

Double: To provide greater security, performs DNS reverse name resolution (page 862) to determine the hostname of each client, performs a forward DNS lookup to verify the original IP address, and logs the hostname. Denies access if it cannot verify the original IP address.

Contexts: **server config**, **virtual host**, **directory**
Default: Off

Tip: Lookups can consume a lot of system resources

Use the **On** and **Double** options with caution: They can consume a lot of resources on a busy system. You can use a program such as logresolve to perform reverse name resolution offline for statistical purposes.

If you perform hostname resolution offline, you run the risk that the name might have changed; you usually want the name that was current at the time of the request. To minimize this problem, perform the hostname resolution as soon as possible after Apache writes the log.

Timeout

Specifies the amount of time Apache waits for network operations to complete.

Timeout **num**

where ***num*** is the number of seconds that Apache waits for network operations to finish.

Context: **server config**
Default: 60

UseCanonicalName

Specifies the method the server uses to identify itself.

UseCanonicalName On | Off | DNS

On*:* Apache uses the value of the ServerName directive (page 942) as its identity.

Off*:* Apache uses the name and port from the incoming request as its identity.

DNS*:* Apache performs a DNS reverse name resolution (page 862) on the IP address from the incoming request and uses the result as its identity. Rarely used.

This directive is important when a server has more than one name and needs to perform a redirect. Fedora/RHEL does not set this directive because it does not set the ServerName directive (page 942). Once you set ServerName, change UseCanonicalName to **On**. See page 962 for a discussion of redirects and this directive.

Contexts: **server config**, **virtual host**, **directory**
Default: Off

Logging Directives

ErrorLog

Specifies where Apache sends error messages.

ErrorLog **filename** | *syslog[:***facility***]*

where ***filename*** specifies the name of the file, relative to ServerRoot (page 955), that Apache sends error messages to. The *syslog* keyword specifies that Apache send errors to **syslogd** (page 620);***facility*** specifies which **syslogd** facility to use. The default facility is **local7**.

Contexts: **server config**, **virtual host**
Default: **logs/error_log**
Fedora/RHEL: **logs/error_log**

LogLevel

Specifies the level of error messages that Apache logs.

LogLevel **level**

where ***level*** specifies that Apache log errors of that level and higher (more urgent). Choose ***level*** from the following list, which is presented here in order of decreasing urgency and increasing verbosity:

emerg	System unusable messages
alert	Need for immediate action messages
crit	Critical condition messages
error	Error condition messages
warn	Nonfatal warning messages
notice	Normal but significant messages
info	Operational messages and recommendations
debug	Messages for finding and solving problems

Contexts: **server config**, **virtual host**
Default: warn
Fedora/RHEL: warn

Directives That Control Content

AddHandler

Creates a mapping between filename extensions and a builtin Apache handler.

AddHandler **handler extension** *[***extension***]* ...

where ***handler*** is the name of a builtin handler and ***extension*** is a filename extension that maps to the ***handler***. Handlers are actions that are built into Apache

and are directly related to loaded modules. Apache uses a handler when a client requests a file with a specified filename extension.

For example, the following AddHandler directive causes Apache to process files that have a filename extension of **.cgi** with the **cgi-script** handler:

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

See "Type Maps" on page 963 for another example of an AddHandler directive.

Contexts: **server config**, **virtual host**, **directory**, **.htaccess**
Default: none

Alias

Maps a URI to a directory or file.

Alias **alias pathname**

where ***alias*** must match part of the URI that the client requested in order to invoke the alias. The***pathname*** is the absolute pathname of the target of the alias, usually a directory.

For example, the following alias causes Apache to serve **/usr/local/pix/milk.jpg** when a client requests **http://www.example.com/pix/milk.jpg**:

```
Alias /pix /usr/local/pix
```

In some cases, you need to use a <Directory> container (page 945) to grant access to aliased content.

Contexts: **server config**, **virtual host**
Default: none
Fedora/RHEL: **/icons/ /usr/share/httpd/icons/** (in **autoindex.conf**)

ErrorDocument

Specifies the action Apache takes when the specified error occurs.

ErrorDocument **code action**

where ***code*** is the error code (page 975) this directive defines a response for and ***action*** is one of the following:

string—Defines the message that Apache returns to the client.

absolute pathname—Points to a local script or other content that Apache redirects the client to.

URI—Points to an external script or other content that Apache redirects the client to.

When you do not specify this directive for a given error code, Apache returns a hardcoded error message when that error occurs. See page 937 for an explanation of how an ErrorDocument directive returns the Fedora/RHEL test page when Apache is installed (before you add content).

Some examples of ErrorDocument directives follow:

Click here to view code image

```
ErrorDocument 403 "Sorry, access is forbidden."
ErrorDocument 403 /cgi-bin/uh-uh.pl
ErrorDocument 403
http://errors.example.com/not_allowed.html
```

Contexts: **server config**, **virtual host**, **directory**, **.htaccess**
Default: none; Apache returns hardcoded error messages
Fedora/RHEL: none; Apache returns hardcoded error messages, but refer to "Fedora/RHEL test page" on page 937

IndexOptions

Specifies how Apache displays directory listings.

*IndexOptions [±]***option** *[[±]***option***]* ...

where ***option*** can be any combination of the following:

DescriptionWidth=*n*—Sets the width of the description column to ***n*** characters. Use ***** in place of ***n***to accommodate the widest description.

FancyIndexing—In directory listings, displays column headers that are links. When you click one of these links, Apache sorts the display based on the content of the column. Clicking the link a second time reverses the order.

FoldersFirst—Sorts the listing so that directories come before plain files. Use only with FancyIndexing.

HTMLTable—Displays a directory listing in a table.

IconsAreLinks—Makes the icons clickable. Use only with FancyIndexing.

IconHeight=*n*—Sets the height of icons to ***n*** pixels. Use only with IconWidth.

IconWidth=*n*—Sets the width of icons to ***n*** pixels. Use only with IconHeight.

IgnoreCase—Ignores case when sorting names.

IgnoreClient—Ignores options the client supplied in the URI.

NameWidth=*n*—Sets the width of the filename column to ***n*** characters. Use ***** in place of ***n*** to accommodate the widest filename.

ScanHTMLTitles—Extracts and displays titles from HTML documents. Use only with FancyIndexing. Not normally used because it is CPU and disk intensive.

SuppressColumnSorting—Suppresses clickable column headings that can be used for sorting columns. Use only with FancyIndexing.

SuppressDescription—Suppresses file descriptions. Use only with FancyIndexing.

SuppressHTMLPreamble—Suppresses the contents of the file specified by the HeaderName directive, even if that file exists.

SuppressIcon—Suppresses icons. Use only with FancyIndexing.

SuppressLastModified—Suppresses the modification date. Use only with FancyIndexing.

SuppressRules—Suppresses horizontal lines. Use only with FancyIndexing.

SuppressSize—Suppresses file sizes. Use only with FancyIndexing.

VersionSort—Sorts version numbers (in filenames) in a natural way; character strings, except for substrings of digits, are not affected.

As an example of the use of IndexOptions, suppose a client requests a URI that points to a directory (such as **http://www.example.com/support/**) and none of the files specified by the DirectoryIndex directive (page 944) is present in that directory. If the directory hierarchy is controlled by a Directory container or **.htaccess** file, and AllowOverride (page 960) has been set to allow indexes, Apache displays a directory listing according to the options specified by this directive.

When an IndexOptions directive appears more than once within a directory, Apache merges the options from the directives. Use + and – to merge IndexOptions options with options from higher-level directories. (Unless you use + or – with all options, Apache discards any options set in higher-level directories.) For example, the following directives and containers set the options for/**custsup/download** to VersionSort; Apache discards FancyIndexing and IgnoreCase in the**download** directory because there is no + or – before VersionSort in the second <Directory> container:

Click here to view code image

```
<Directory /custsup>
    IndexOptions FancyIndexing
    IndexOptions IgnoreCase
```

```
</Directory

<Directory /custsup/download>
    IndexOptions VersionSort
</Directory>
```

Because + appears before VersionSort, the following directives and containers set the options for/**custsup/download** to FancyIndexing, IgnoreCase, and VersionSort:

Click here to view code image

```
<Directory /custsup>
    IndexOptions FancyIndexing
    IndexOptions IgnoreCase
</Directory

<Directory /custsup/download>
    IndexOptions +VersionSort
</Directory>
```

Contexts: **server config**, **virtual host**, **directory**, **.htaccess**
Default: none; lists only filenames
Fedora/RHEL: FancyIndexing HTMLTable VersionSort (in **autoindex.conf**)

ServerRoot

Specifies the root directory for server files (not content).

ServerRoot **directory**

where ***directory*** specifies the pathname of the root directory for the files that make up the server. Apache prepends ***directory*** to relative pathnames in **httpd.conf**. This directive does not specify the location of the content that Apache serves; the DocumentRoot directive (page 942) performs that function. Do not change this value unless you move the server files.

Context: **server config**
Default: **/usr/local/apache**
Fedora/RHEL: **/etc/httpd**

ServerTokens

Specifies the server information that Apache returns to a client.

ServerTokens Prod | Major | Minor | Min | OS | Full

Prod: Returns the product name: **Apache**; also ***ProductOnly***.

Major: Returns the major release number of the server: **Apache/2**.

Minor: Returns the major and minor release numbers of the server: **Apache/2.4**.

Min: Returns the complete version: **Apache/2.4.4**; also ***Minimal***.

OS: Returns the name of the operating system and the complete version: **Apache/2.4.4 (Fedora)**. Provides less information that might help a malicious user than ***Full*** does.

Full: Same as ***OS***, except that ***Full*** also sends the names and versions of non-ASF modules:**Apache/2.4.4 (Fedora) PHP/5.5.0**.

Unless you want clients to know the details of the software you are running, set ServerTokens to reveal as little as possible.

Context: **server config**
Default: Full

ServerSignature

Adds a line to server-generated pages.

ServerSignature On | Off | EMail

On: Turns the signature line on. The signature line contains the server version as specified by the ServerTokens directive (above) and the name specified by the <VirtualHost> container (page 949).

Off: Turns the signature line off.

EMail: To the signature line, adds a **mailto**: link to the server email address. This option produces output that can attract spam. See ServerAdmin (page 941) for information on specifying an email address.

Contexts: **server config**, **virtual host**, **directory**, **.htaccess**
Default: Off

Configuration Directives

Group

Sets the GID of the processes that run the servers.

Group #**groupid** | **groupname**

where ***groupid*** is a GID value, preceded by **#**, and ***groupname*** is the name of a group. The processes (servers) that Apache spawns are run as the group specified by this directive. See the User directive (page 959) for more information.

Context: **server config**
Default: #–1
Fedora/RHEL: apache

Include IncludeOptional

Loads directives from files.

Include | *IncludeOptional* **filename** | **directory**

where ***filename*** is the relative pathname of a file that contains directives. Apache prepends ServerRoot (page 955) to ***filename***. The directives in ***filename*** are included in the file holding this directive at the location of the directive. Because ***filename*** can include wildcards, it can specify more than one file.

The ***directory*** is the relative pathname that specifies the root of a directory hierarchy that holds files containing directives. Apache prepends ServerRoot to ***directory***. The directives in ordinary files in this hierarchy are included in the file holding this directive at the location of the directive. The***directory*** can include wildcards.

Whereas Include displays an error when wildcards do not match a file or directory, IncludeOptional does not display an error when there is no match for a wildcard.

The Apache installer puts configuration files for modules, which have a filename extension of **conf**, in the **conf.d** and **conf.modules.d** directories within the ServerRoot directory. The Include directives in the Fedora/RHEL **httpd.conf** file incorporate module configuration files for whichever modules are installed.

Contexts: **server config**, **virtual host**, **directory**
Default: none
Fedora/RHEL: **conf.d/*.conf conf.modules.d/*.conf**

LoadModule

Loads a module.

LoadModule **module filename**

where ***module*** is the name of an external DSO module and ***filename*** is the relative pathname of the named module. Apache prepends ServerRoot (page 955)

to ***filename*** and loads the external module specified by this directive. For more information refer to "Modules" on page 968.

Context: **server config**
Default: none; nothing is loaded by default if this directive is omitted
Fedora/RHEL: loads many modules; refer to the configuration files

Options

Controls server features by directory.

*Options [±]***option** *[[±]***option** *...]*

This directive controls which server features are enabled for a directory hierarchy. The directory hierarchy is specified by the container this directive appears in. A **+** or the absence of a **–** turns an option on, and a **–** turns it off.

The ***option*** might be one of the following:

None—None of the features this directive can control are enabled.

All—All of the features this directive can control are enabled, except for MultiViews, which you must explicitly enable.

ExecCGI—Apache can execute CGI scripts (page 969).

FollowSymLinks—Apache follows symbolic links.

Includes—Permits SSIs (server-side includes). SSIs are containers embedded in HTML pages that are evaluated on the server before the content is passed to the client.

IncludesNOEXEC—The same as Includes but disables the **#exec** and **#exec cgi** commands that are part of SSIs. Does *not* prevent the **#include** command from referencing CGI scripts.

Indexes—Generates a directory listing if DirectoryIndex (page 944) is not set.

MultiViews—Allows MultiViews (page 963).

SymLinksIfOwnerMatch—The same as FollowSymLinks but follows the link only if the file or directory being pointed to has the same owner as the link.

The following Options directive from the Fedora/RHEL **httpd.conf** file sets the Indexes and FollowSymLinks options and, because the <Directory> container specifies the **/var/www/html**directory hierarchy (the document root), affects all content. The AllowOverride None directive causes Apache to ignore directives in **.htaccess** files (page 960).

Click here to view code image

```
<Directory "/var/www/html">
    Options Indexes FollowSymLinks
    AllowOverride None
</Directory>
```

Context: **directory**
Default: All
Fedora/RHEL: Indexes, FollowSymLinks

ScriptAlias

Maps a URI to a directory or file and declares the target to be a server (CGI) script.

ScriptAlias **alias pathname**

where ***alias*** must match part of the URI the client requested to invoke the ScriptAlias. The***pathname*** is the absolute pathname of the target of the alias, usually a directory. Similar to the Alias directive, this directive specifies that the target is a CGI script (page 969).

The following ScriptAlias directive from the Fedora/RHEL **httpd.conf** file maps client requests that include **/cgi-bin/** to the **/var/www/cgi-bin** directory (and indicates that these requests will be treated as CGI requests):

Click here to view code image

```
ScriptAlias /cgi-bin/ "/var/www/cgi-bin/"
```

Contexts: **server config**, **virtual host**
Default: none
Fedora/RHEL: /cgi-bin/ "/var/www/cgi-bin"

User

Sets the UID of the processes that run the servers.

User #**userid** | **username**

where ***userid*** is a UID value, preceded by **#**, and ***username*** is the name of a local user. The processes that Apache spawns are run as the user specified by this directive.

Apache must start with **root** privileges to listen on a privileged port. For reasons of security, Apache's child processes (servers) run as nonprivileged users. The default UID of –1 does not map to a user under Fedora/RHEL. Instead, the Fedora/RHEL **httpd** package creates a user named **apache** during installation and sets User to that user.

Context: **server config**
Default: #–1
Fedora/RHEL: **apache**

Security: Do not set User to root or 0

For a more secure system, do not set User to **root** or **0** (zero) and do not allow the **apache**user to have write access to the DocumentRoot directory hierarchy (except as needed for storing data), especially not to configuration files.

Security Directives

Allow

Specifies which clients can access specified content.

Allow from All | **host** *[***host** *...] | env=***var** *[env=***var** *...]*

This directive, which must be written as **Allow from**, grants access to a directory hierarchy to the specified clients. The directory hierarchy is specified by the container or **.htaccess** file this directive appears in. See the Order directive (page 961) for an example.

All—Serves content to any client.

host*:* Serves content to the client(s) specified by ***host***, which can take several forms: an FQDN, a partial domain name (such as **example.com**), an IP address, a partial IP address, or a network/netmask pair.

var*:* Serves content when the environment variable named ***var*** is set. You can set a variable with the SetEnvIf directive.

Contexts: **directory**, **.htaccess**
Default: none; default behavior depends on the Order directive
Fedora/RHEL: none

AllowOverride

Specifies whether Apache examines ***.htaccess*** *files and which classes of directives in those files it processes.*

AllowOverride All | None | **directive-class** *[***directive-class** *...]*

This directive specifies whether Apache examines **.htaccess** files in the directory hierarchy specified by its container. If Apache does examine **.htaccess** files, this directive specifies which classes of directives within **.htaccess** files Apache processes. See the tip on page 973 for performance considerations.

All—Processes all classes of directives in **.htaccess** files.

None—Ignores directives in **.htaccess** files. However, Apache will still serve the content of **.htaccess**files, possibly exposing sensitive information. This choice does not affect **.htpasswd** files. The example in the description of the <Files> container (page 947) shows how to prevent Apache from serving the content of files whose names begin with **.ht**.

The ***directive-class*** is one of the following directive class identifiers:

AuthConfig—Class of directives that control authorization (AuthName, AuthType, Require, and so on). This class is used mostly in **.htaccess** files to require a username and password to access the content. For more information refer to "Authentication Modules and **.htaccess** Files" on page 972.

FileInfo—Class of directives that controls document types (DefaultType, ErrorDocument, SetHandler, and so on).

Indexes—Class of directives relating to directory indexing (DirectoryIndex, FancyIndexing, IndexOptions, and so on).

Limit—Class of client-access directives (Allow, Deny, and Order).

Options—Class of directives controlling directory features.

Context: **directory**
Default: All
Fedora/RHEL: None

Deny

Specifies which clients are not allowed to access specified content.

Deny from All | **host** *[***host** *...] | env=***var** *[env=***var** *...]*

This directive, which must be written as **Deny from**, denies access to a directory hierarchy to the specified clients. The directory hierarchy is specified by the container or **.htaccess** file this directive appears in. See the Order directive (next) for an example.

All—Denies content to all clients.

host**:* Denies content to the client(s) specified by ***host, which can take several forms: an FQDN, a partial domain name (such as **example.com**), an IP address, a partial IP address, or a network/netmask pair.

var**:* Denies content when the environment variable named ***var is set. You can set a variable using the SetEnvIf directive.

Contexts: **directory**, **.htaccess**
Default: none
Fedora/RHEL: none

Order

Specifies the default access and the order in which Allow and Deny directives are evaluated.

Order Deny,Allow | Allow,Deny

Deny,Allow—Allows access by default; denies access only to clients specified in Deny directives (first evaluates Deny directives and then evaluates Allow directives).

Allow,Deny—Denies access by default; allows access only to clients specified in Allow directives (first evaluates Allow directives and then evaluates Deny directives).

There must not be SPACEs on either side of the comma. Access defaults to the second entry in the pair (Deny,Allow defaults to Allow) if there is no Allow from or Deny from directive that matches the client. If a single Allow from or Deny from directive matches the client, that directive overrides the default. If multiple Allow from and Deny from directives match the client, Apache evaluates the directives in the order specified by the Order directive; the last match takes precedence.

Access granted or denied by this directive applies to the directory hierarchy specified by the container or **.htaccess** file this directive appears in. In the next directive, **Allow from all** grants access to all clients:

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

You can restrict access by specifying Deny,Allow to deny all access and then specifying only those clients you want to grant access to in an Allow directive. The following directives grant access to clients from the **example.net** domain only and would typically appear within a <Directory> container (page 945):

```
Order deny,allow
Deny from all
Allow from .example.net
```

Contexts: **directory**, **.htaccess**
Default: Deny,Allow

Require

Tests/specifies whether a user is authorized to access a directory hierarchy.

Require all granted | all denied | ip **addr**

where ***addr*** is one or more SPACE-separated IP addresses of systems that are allowed to access the directory hierarchy.

all granted*:* Any user can access the directory hierarchy.

all denied*:* No user can access the directory hierarchy.

The **httpd.conf** file contains several Require directives. The first, because it is in a <Directory> container that specifies the root directory, denies access to the entire filesystem:

```
<Directory />
    AllowOverride none
    Require all denied
</Directory>
```

However, that restriction is immediately relaxed for the **www** child directory.

```
<Directory "/var/www">
    AllowOverride None
    Require all granted
</Directory>
```

Context: **directory**, **.htaccess**
Default: none

Advanced Configuration

This section describes how to configure some advanced features of Apache.

Redirects

Apache can respond to a request for a URI by asking the client to request a different URI. This response is called a *redirect.* A redirect works because

redirection is part of the HTTP implementation: Apache sends the appropriate response code and the new URI, and a compliant browser requests the new location.

The Redirect directive can establish an explicit redirect that sends a client to a different page when a Web site is moved. Or, when a user enters the URI of a directory in a browser but leaves off the trailing slash, Apache can automatically redirect the client to the same URI terminated with a slash.

UseCanonicalName

The ServerName directive (page 942), which establishes the name of the server, and the UseCanonicalName directive (page 951) are both important when a server has more than one name and needs to perform an automatic redirect. For example, assume the server with the name**zach.example.com** and the alias **www.example.com** has ServerName set to**www.example.com**. When a client specifies a URI of a directory but leaves off the trailing slash (**zach.example.com/dir**), Apache has to perform a redirect to determine the URI of the requested directory. When UseCanonicalName is set to On, Apache uses the value of ServerName and returns**www.example.com/dir/**. With UseCanonicalName set to Off, Apache uses the name from the incoming request and returns **zach.example.com/dir/**.

Content Negotiation

Apache can serve multiple versions of the same page, using a client's preference to determine which version to send. The process Apache uses to determine which version of a page (file) to send is called*content negotiation.* Apache supports two methods of content negotiation: type maps and MultiViews search, which can work together.

Type Maps

The following AddHandler directive from **httpd.conf** would tell Apache to use any filename ending in**.var** as a type map (if it were not commented out):

```
# AddHandler type-map var
```

To see how type maps work, create the following files in **/var/www/html**:

Click here to view code image

```
$ cat /var/www/html/index.html.en
<html><body><h1>Hello</h1></body></html>
```

```
$ cat /var/www/html/index.html.fr
<html><body><h1>Bonjour</h1><body></html>

$ cat /var/www/html/index.html.var
URI: index.html.en
Content-Language: en
Content-type: text/html; charset=ISO-8859-1

URI: index.html.fr
Content-Language: fr
Content-type: text/html; charset=ISO-8859-1
```

If your browser's preferred language is set to English (**en**), it will display the **Hello** page when you browse to **http://localhost/index.html.var**. If your browser's preferred language is set to French (**fr**), it will display the **Bonjour** page. (With the MultiViews option turned on, as it is by default, the browser displays the correct page when you browse to **http://localhost**. See the next section.) You can change the default language in Firefox by selecting **Edit⇨Preferences** from the menubar, clicking the Content icon, and finally clicking **Choose** from the Languages frame. Select a language from the**Select a language to add** combo box, if necessary, and then move the preferred language to the top of the list. In the example, the **charset** assignments are not necessary. However, they would be helpful if you were sending pages using different encodings such as English, Russian, and Korean.

Type maps are used for more than selecting among different languages. Instead of matching **Content-Language** as in the preceding example, the map could match **Content-type** and send **jpeg** or **png**images depending on how the browser's preferences are set.

MultiViews

When you set the MultiViews option on a directory, Apache attempts to deliver the correct page when a requested resource does not exist. In **httpd.conf**, add **MultiViews** to the Options line in the <Directory> container as follows:

Click here to view code image

```
<Directory "/var/www/html">
...
    Options Indexes FollowSymLinks MultiViews
...
```

To see how MultiViews work, remove the **/var/www/html/index.html.var** type map file you created in the preceding section. Now browse to **http://localhost**. The proper language page is displayed, but why?

When a browser sends Apache a request for a directory, Apache looks for a file named **index.html** in that directory. In the example, Apache does not find the file. If MultiViews is enabled, Apache looks for files named **index.html.***. In the example it finds **index.html.en** and **index.html.fr**. Apache effectively creates a type map on the fly, mapping the **index.html.*** files to various languages, and sends its best guess as to the page you want.

Caution: Which to use: MultiViews or type maps?

MultiViews provides an easy way to serve multiple versions of the same file without having to create a type map. However, MultiViews is slower than type maps. Also, if you require finer-grained control over which version of a resource should be sent, type maps are a better solution.

Server-Generated Directory Listings (Indexing)

When a client requests a directory, the Apache configuration determines what is returned to the client. Apache can return a file as specified by the DirectoryIndex directive (page 944), a directory listing if no file matches DirectoryIndex and the Options Indexes directive (page 957) is set, or an error message if no file matches DirectoryIndex and Options Indexes is not set. Figure 26-1 shows a server-generated directory listing.

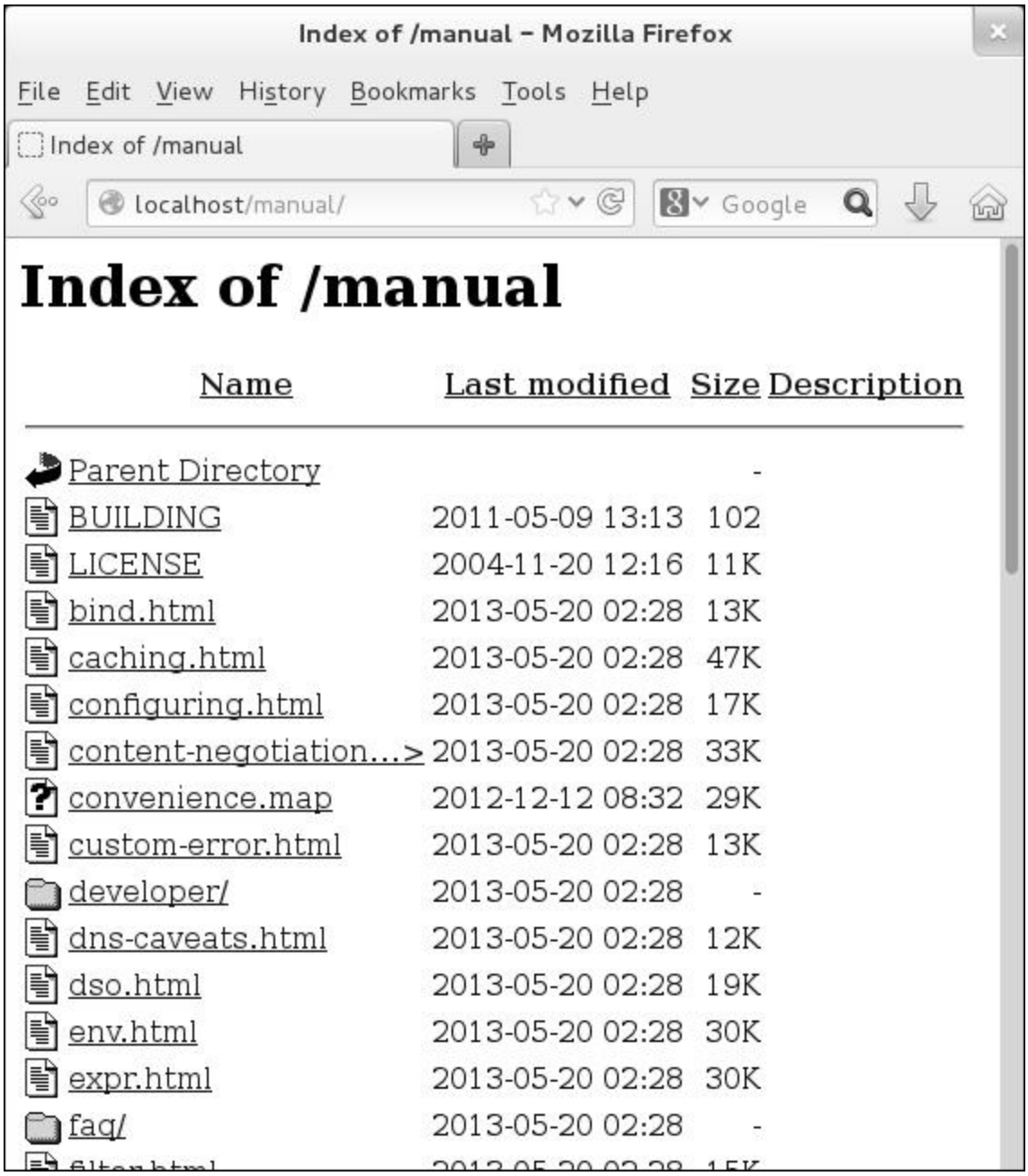


Figure 26-1 A server-generated directory listing

Virtual Hosts

Apache supports *virtual hosts*, which means that one instance of Apache can respond to requests directed to multiple IP addresses or hostnames as though it were multiple servers. Each IP address or hostname can then provide different content and be configured differently.

There are two types of virtual hosts: *host-by-name* (also called *host-based*) and *host-by-IP*. Host-by-name relies on the FQDN the client uses in its request to Apache—for example, **www.example.com**versus **www2.example.com**. Host-by-IP examines the IP address the host resolves as and responds according to that match.

Host-by-name is handy if there is only one IP address, but Apache must support multiple FQDNs. Although you can use host-by-IP if a given Web server has aliases, Apache should serve the same content regardless of which name is used.

The NameVirtualHost directive specifies which IP address supports host-by-name virtual hosting; the ServerName (or ServerAlias) directive must match the client request to match that virtual host. Without a NameVirtualHost directive, the virtual host is a host-by-IP virtual host; the ServerName directive specifies the name the server uses to identify itself.

You can specify many virtual hosts for a single instance of Apache.

Examples

The following examples of host-by-name virtual hosting use wildcards (*****) to remain as flexible as possible. You might want to replace the wildcards with the IP address of the server for more precise control when Apache is serving multiple virtual hosts.

The first <VirtualHost> container sets up host-by-name for the site named **example.com**. This virtual host handles requests that are directed to **example.com**. The ServerAlias directive allows it to also process requests directed to **www.example.com**.

Click here to view code image

```
<VirtualHost *>
    ServerName      example.com
    ServerAlias     www.example.com
    ServerAdmin     webmaster@example.com
    DocumentRoot    /var/www/html/example.com
    CustomLog       /var/log/httpd/example.com.log
combined
    ErrorLog        /var/log/httpd/example.com.err
</VirtualHost>
```

The next example is similar to the previous one. It adds a Directory directive that prevents remote users (users not coming from the 192.168. subnet) from accessing the Web site.

Click here to view code image

```
<VirtualHost *>
    ServerName      intranet.example.com
    ServerAdmin     webmaster@example.com
    DocumentRoot    /var/www/html
    ErrorLog        /var/log/httpd/intra.error_log
    CustomLog       /var/log/httpd/example.com.log
combined
    <Directory      /var/www/html>
        Order deny,allow
        Deny from all
        Allow from 192.168.  # allow from private
subnet only
    </Directory>
</VirtualHost>
```

The next example sets up two virtual hosts. The VirtualHost containers accept all traffic directed to the server by specifying *****. The ServerName directives accept traffic for **sam.example.com** (or the alias**www.example.com/sam**) and **mail.example.com**. The first virtual host serves documents from Sam's **public_html** directory; the second is a Webmail server with its content at**/var/www/html/squirrelmail**. This example works because all three addresses resolve to the IP address of the server.

Click here to view code image

```
NameVirtualHost *:
<VirtualHost *>
    ServerName      sam.example.com
    ServerAlias     www.example.com/sam
    ServerAdmin     webmaster@example.com
    DocumentRoot    /home/sam/public_html
</VirtualHost>

<VirtualHost *:>
    ServerName      mail.example.com
    ServerAdmin     webmaster2@example.com
    DocumentRoot    /var/www/html/squirrelmail
</VirtualHost>
```

If the user specifies an IP address and not a URI, that address might match more than one of the virtual hosts, as in the example. In this case, Apache serves the virtual host that best matches. If none of the virtual host addresses matches the IP address better than another, Apache serves the first virtual host. In the preceding example, both virtual hosts match an IP address the same way; neither is a better match, so Apache serves the first virtual host (**sam.example.com**).
If **mail.example.com**was defined as <VirtualHost 192.168.1.102> and a user specified that IP address, Apache would serve**mail.example.com** because it is a better match for the IP address than the wildcard that the other virtual host specifies.

The next example shows VirtualHost containers for a host-by-IP server. The example assumes that 111.111.0.0 and 111.111.0.1 point to the local server. Here each virtual host has its own IP/port combination. The third virtual host is distinguished from the first by the port that a request comes in on.

Click here to view code image

```
<VirtualHost 111.111.0.0:80>
    DocumentRoot /var/www/html/www0
</VirtualHost>

<VirtualHost 111.111.0.1:80>
    DocumentRoot /var/www/html/www1
</VirtualHost>

<VirtualHost 111.111.0.0:8080>
    DocumentRoot /var/www/html/www2
    Listen 8080
</VirtualHost>
```

The final example sets up a virtual server for Webmail that can be accessed only over SSL. To use this example you must create an SSL certificate (page 970).

Click here to view code image

```
<VirtualHost mail.example.com:80>
    Redirect permanent / https://mail.example.com/
</VirtualHost>
<VirtualHost mail.example.com:443>
    ServerName     mail.example.com
    ServerAdmin    postmaster@example.com
```

```
    DocumentRoot    /var/www/html/mail.example.com
    ErrorLog        /var/log/httpd/mail.example.com.err
    CustomLog       /var/log/httpd/mail.example.com.log
combined
    SSLEngine On
    SSLCertificateFile /etc/pki/tls/certs/apache.pem
</VirtualHost>
```

Troubleshooting

If the browser does not display the information you expect, make sure it is not displaying a cached version of the page. Try clearing the browser's cache and reloading the page. Also make sure that Apache has permission to read the files it is serving and the scripts it is running.

The **httpd** service checks the syntax of the Apache configuration files and logs an error if there is a problem. You can also call apachectl directly to check the syntax:

```
$ apachectl configtest
Syntax OK
```

Once you start the **httpd** daemon, you can confirm Apache is working correctly by pointing a browser on the local system at **http://localhost/**. From a remote system, use **http://*server*/**, substituting the hostname or IP address of the server for ***server***. In response, Apache displays the Fedora/RHEL test page (page 937) unless you have added an index file or changed the default virtual host.

If the browser does not display the test page, it will display one of two errors: **Connection refused** or an error page. If it displays **Connection refused**, make sure port 80 is not blocked by a firewall (page934), check that SELinux (page 934) is set up properly, and check that the server is running:

Click here to view code image

```
# systemctl status httpd.service
httpd.service - The Apache HTTP Server
   Loaded: loaded
(/usr/lib/systemd/system/httpd.service; enabled)
   Active: active (running) since ...
```

If the server is running, confirm that you did not specify a port other than 80 in a Listen directive. If you did, the URI you specify in the browser must reflect this

port number (**http://localhost:*port***specifies port ***port***). Otherwise, check the error log (**/var/log/httpd/error_log**) for information about what is not working.

To verify the browser is not at fault, use telnet to connect to port 80 of the server:

Click here to view code image

```
$ telnet www.example.com 80
Trying 192.0.34.166...
Connected to www.example.com.
Escape character is '^]'.
CONTROL-]
telnet> quit
Connection closed.
```

If telnet displays **Connection refused**, it means the local system cannot connect to the server.

Modules

Apache is a skeletal program that relies on external modules, called DSOs (dynamic shared objects), to provide most of its functionality. In addition to the modules included with Fedora/RHEL, many other modules are available. For more information see httpd.apache.org/modules and httpd.apache.org/docs/2.4/mod.

Following is a list of some of the modules that are available under Apache.

actions (mod_actions.so) Allows execution of CGI scripts based on the request method.

alias (mod_alias.so) Allows outside directories to be mapped to DocumentRoot.

asis (mod_asis.so) Allows sending files that contain their own headers.

auth_basic (mod_auth_basic.so) Provides user authentication via **.htaccess**.

auth_digest (mod_auth_digest.so) Uses MD5 digest for authentication.

authn_anon (mod_authn_anon.so) Provides anonymous user access to restricted areas.

authn_dbd (mod_authn_dbd.so) Uses SQL files for authentication.

autoindex (mod_autoindex.so) Allows directory indexes to be generated.

cgi (mod_cgi.so) Allows the execution of CGI scripts.

dav (mod_dav.so) Allows Distributed Authoring and Versioning.

dav_fs (mod_dav_fs.so) Provides a filesystem for mod_dav.

dir (mod_dir.so) Allows directory redirects and listings as index files.

env (mod_env.so) Allows CGI scripts to access environment variables.

expires (mod_expires.so) Allows generation of Expires HTTP headers.

headers (mod_headers.so) Allows customization of request and response headers.

include (mod_include.so) Provides server-side includes (SSIs).

info (mod_info.so) Allows the server configuration to be viewed.

log_config (mod_log_config.so) Allows logging of requests made to the server.

mime (mod_mime.so) Allows association of file extensions with content.

mime_magic (mod_mime_magic.so) Determines MIME types of files.

negotiation (mod_negotiation.so) Allows content negotiation.

proxy (mod_proxy.so) Allows Apache to act as a proxy server.

proxy_connect (mod_proxy_connect.so) Allows connect request handling.

proxy_ftp (mod_proxy_ftp.so) Provides an FTP extension proxy.

proxy_http (mod_proxy_http.so) Provides an HTTP extension proxy.

rewrite (mod_rewrite.so) Allows on-the-fly URI rewriting based on rules.

setenvif (mod_setenvif.so) Sets environment variables based on a request.

speling (mod_speling.so) Auto-corrects spelling if the requested URI has incorrect capitalization and one spelling mistake.

status (mod_status.so) Allows the server status to be queried and viewed.

unique_id (mod_unique_id.so) Generates a unique ID for each request.

userdir (mod_userdir.so) Allows users to have content directories (public_html).

usertrack (mod_usertrack.so) Allows tracking of user activity on a site.

vhost_alias (mod_vhost_alias.so) Allows the configuration of virtual hosting.

mod_cgi **and CGI Scripts**

The CGI (Common Gateway Interface) allows external application programs to interface with Web servers. Any program can be a CGI program if it runs in real time (at the time of the request) and relays its output to the requesting client. Various kinds of scripts, including shell, Perl, Python, and PHP, are the most commonly encountered CGI programs because a script can call a program and reformat its output in HTML for a client.

Apache can handle requests for CGI programs in several different ways. The most common method is to put a CGI program in the **cgi-bin** directory and then enable its execution from that directory only. The location of the **cgi-bin** directory, as specified by the ScriptAlias directive (page 958), is**/var/www/cgi-bin**. Alternately,

an AddHandler directive (page 952) can identify the filename extensions of scripts, such as **.cgi** or **.pl**, within the regular content (e.g., **AddHandler cgi-script .cgi**). If you use AddHandler, you must also specify the ExecCGI option in an Options directive within the appropriate <Directory> container. The **mod_cgi** module must be loaded to access and execute CGI scripts.

The following Perl CGI script displays the Apache environment. This script should be used for debugging only because it presents a security risk if remote clients can access it:

Click here to view code image

```
#!/usr/bin/perl
##
##  printenv -- demo CGI program that prints its
environment
##

print "Content-type: text/plain\n\n";
foreach $var (sort(keys(%ENV))) {
    $val = $ENV{$var};
    $val =~ s|\n|\\n|g;
    $val =~ s|"|\\"|g;
    print "${var}=\"${val}\"\n";
}
```

mod_ssl

SSL (Secure Sockets Layer), which is implemented by the **mod_ssl** module, has two functions: It allows a client to verify the identity of a server and it enables secure two-way communication between a client and a server. SSL is used on Web pages in conjunction with forms that require passwords, credit card numbers, or other sensitive data.

Apache uses the HTTPS protocol—not HTTP—for SSL communication. When Apache uses SSL, it listens on a second port (443 by default) for a connection and performs a handshaking sequence before sending the requested content to the client.

Server verification is critical for financial transactions: You do not want to give your credit card number to a fraudulent Web site posing as a known company. SSL uses a certificate to positively identify a server. Over a public network such as the Internet, the identification is reliable only if the certificate contains a digital

signature from an authoritative source such as VeriSign or Thawte. SSL Web pages are denoted by a URI beginning with **https://**.

Data encryption prevents malicious users from eavesdropping on Internet connections and copying personal information. To encrypt communication, SSL sits between the network and an application and encrypts communication between the server and the client.

Setting Up mod_ssl

You must install the **mod_ssl** package to run SSL.
The **/etc/httpd/conf.d/ssl.conf** file configures and loads **mod_ssl**. The first few directives in this file instruct Apache to listen on port 443, and set various parameters for SSL operation. About a third of the way through the file is a section labeled **SSL Virtual Host Context** that sets up virtual hosts (page 965).

As with any virtual host, a virtual host for SSL holds directives such as ServerName and ServerAdmin that need to be configured. In addition, it holds some SSL-related directives. See the example on page967.

Using a Self-Signed Certificate for Encryption

If you require SSL for encryption and not verification—that is, if the client already trusts the server—you can generate and use a self-signed certificate, bypassing the time and expense involved in obtaining a digitally signed certificate. Self-signed certificates generate a warning when you connect to the server: Most browsers display a dialog box that allows you to examine and accept the certificate. The**sendmail** daemon also uses certificates (page 765).

The self-signed certificate depends on two files: a private key and the certificate. The location of each file is specified in **/etc/httpd/conf.d/ssl.conf**:

Click here to view code image

```
# grep '^SSLCertificate' /etc/httpd/conf.d/ssl.conf
SSLCertificateFile /etc/pki/tls/certs/localhost.crt
SSLCertificateKeyFile
/etc/pki/tls/private/localhost.key
```

The following example creates a self-signed certificate. To generate the private key that the encryption relies on, cd to **/etc/pki/tls/certs** and enter a make command:

Click here to view code image

```
# cd /etc/pki/tls/certs
# make localhost.key
```

```
umask 77 ; \
/usr/bin/openssl genrsa -aes128 2048 > localhost.key
Generating RSA private key, 2048 bit long modulus
.+++
.....................+++
e is 65537 (0x10001)
Enter pass phrase:
Verifying - Enter pass phrase:
```

The preceding command generates a file named **localhost.key** that is protected by the passphrase you entered: *You will need this passphrase to start the server.* Keep the **localhost.key** file secret.

The next command generates the certificate. This process uses the private key you just created. You need to supply the same passphrase you entered when you created the private key.

Click here to view code image

```
# make localhost.crt
umask 77 ; \
/usr/bin/openssl req -utf8 -new -key localhost.key -
x509 -days 365 -out
localhost.crt -set_serial 0
Enter pass phrase for localhost.key:
You are about to be asked to enter information that
will be incorporated
into your certificate request.
What you are about to enter is what is called a
Distinguished Name or a DN.
There are quite a few fields but you can leave some
blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [GB]:US
State or Province Name (full name)
[Berkshire]:California
Locality Name (eg, city) [Newbury]:San Francisco
Organization Name (eg, company) [My Company Ltd]:Sobell
Associates Inc.
```

```
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname)
[]:www.sobell.com
Email Address []:mgs@sobell.com
```

The answers to the first five questions are arbitrary: They can help clients identify a site when they examine the certificate. The answer to the sixth question (**Common Name**) is critical. Because certificates are tied to the name of the server, you must enter the server's FQDN accurately. If you mistype this information, the server name and that of the certificate will not match. The browser will then generate a warning message each time a connection is made.

As specified by **ssl.conf**, Apache looks for the files in the directory that you created them in. Do not move these files. After you restart Apache, the new certificate will be in use.

Notes on Certificates

• As long as the server is identified by the name for which the certificate was issued, you can use the certificate on another server or IP address.

• A root certificate is the certificate that identifies the root certificate authority (root CA). Every browser contains a database of the public keys for the root certificates of the major signing authorities, including VeriSign and Thawte.

• It is possible to generate a root certificate and sign all your server certificates with this root CA. Regular clients can import the public key of the root CA so that they recognize every certificate signed by that root CA. This setup is convenient for a server with multiple SSL-enabled virtual hosts and no commercial certificates. For more information see www.modssl.org/docs/2.8/ssl_faq.html#ToC29.

• A self-signed certificate does not enable clients to verify the identity of the server.

Authentication Modules and .htaccess Files

To restrict access to a Web page, Apache and third parties provide authentication modules and methods that can verify a user's credentials, such as a username and password. Some modules support authentication against various databases including NIS (page 770) and LDAP (page 786).

User authentication directives are commonly placed in a **.htaccess** file. A basic **.htaccess** file that uses the Apache default authentication module (**mod_auth**) follows. When placed in **/var/www/html**, this file causes Apache to request a password before a browser can access content in

the **/var/www/html** directory hierarchy. Substitute appropriate values for the local server.

Click here to view code image

```
# cat .htaccess
AuthUserFile /var/www/html/.htpasswd
AuthGroupFile /dev/null
AuthName "Browser dialog box query"
AuthType Basic
require valid-user
```

The **/var/www/html/.htpasswd** is a typical absolute pathname of a **.htpasswd** file and **Browser dialog box query** is the string that the user will see as part of the dialog box that requests a username and password.

The second line of the preceding **.htaccess** file turns off the group function. The fourth line specifies the user authentication type **Basic**, which is implemented by the default **mod_auth** module. The last line tells Apache which users can access the protected directory. The entry **valid-user** grants access to the directory to any user whose username appears in the Apache password file and who enters the correct password.

You can put the Apache password file anywhere on the system, as long as Apache can read it. It is safe to put this file in the same directory as the **.htaccess** file because, by default, Apache will not answer any requests for files whose names start with **.ht**. However, be sure you have not changed the**httpd.conf** configuration file to allow Apache to answer requests for files whose names begin with **.ht**. See page 939 for more information on **.htaccess** files.

The following command creates (**–c**) a **.htpasswd** file with an entry for Sam in the working directory. Omit the **–c** option to add a user or to change a password in an existing **.htpasswd** file.

Click here to view code image

```
$ htpasswd -c .htpasswd sam
New password:
Re-type new password:
Adding password for user sam
```

The **default httpd.conf** file includes an **AllowOverride None** directive (page 960) for**/var/www/html**. You must change this directive to at least **AllowOverride AuthConfig** or remove it to enable Apache to process user authentication directives (such as reading an **.htaccess** file).

Tip: .htaccess files can degrade performance

With Apache configured to process **.htaccess** files, when it receives a request for a file, it has to traverse the directory hierarchy from the requested file upward, toward the root, looking for a **.htacess** file to determine whether it can serve the requested file. This search can affect performance. Typically performance degradation is not severe, but this issue can become important if performance is critical.

Scripting Modules

Apache can process content before serving it to a client. In earlier versions of Apache, only CGI scripts could process content. In the current version, *scripting modules* can work with scripts embedded in HTML documents. The downside is that major bugs in scripting modules can affect the stability of Apache processes.

Scripting modules manipulate content before Apache serves it to a client. Because they are built into Apache, scripting modules are fast. Scripting modules are especially efficient at working with external data sources such as relational databases. Clients can pass data to a scripting module that modifies the information that Apache serves.

Scripting modules stand in contrast to CGI scripts that are run externally to Apache. In particular, CGI scripts do not allow client interaction and are slow because they must make external calls.

You can embed Perl (**mod_perl**), Python, and PHP code in HTML content. Perl and Python are general-purpose scripting languages and are implemented in the **mod_perl** and **mod_wsgi** modules.

PHP, which was developed for manipulating Web content, outputs HTML by default. It is encapsulated for use directly in Apache, is easy to set up, has a syntax similar to that of Perl and C, and comes with a large number of Web-related functions.

Multiprocessing Modules (MPMs)

If Apache were to execute in only one process, then each time a client requested a page, Apache would have to ignore other requests while it read that page from disk (or waited for a CGI script to generate it). After it read the page, it could send the page to the client and respond to the next request. With this setup, Apache could serve only one client at a time.

prefork MPM

Apache 1.3 and earlier forked servers to respond to multiple clients. Apache 2 moved the forking behavior to the **prefork** MPM (multiprocessing module).

MPMs introduced the ability to switch between various multiprocessing techniques.

The **prefork** MPM uses the **fork**() system call to create an exact copy of the running Apache process to serve each request. The MaxServers, MaxRequestWorkers, and similar directives control how many copies of Apache run at the same time. Because the operating system has to spend time context switching between Apache processes, and because each process has its own memory, the **prefork**MPM generates considerable overhead on a busy server. Fedora/RHEL runs the **prefork** MPM by default.

worker MPM

The **worker** MPM reduces this overhead by using **threads**. A thread is similar to a process in that it can execute independently of other threads or processes. Waiting for a read to complete in one thread does not stop (block) other threads from executing. The difference between threads and processes is that all the threads running under one process share the same memory, and the program—rather than the operating system—is responsible for managing the threads. The **worker** MPM maintains a pool of threads it can use to serve each request. Instead of the parent Apache process forking a child to serve each request for content as in **prefork**, the **worker** MPM uses threads to serve requests for content.

Threads

Because all these threads run under the same process, they share the same memory. Code that is not*thread safe* (see *reentrant* on page 1269) can return inconsistent results. For example, some PHP library functions use the **strtok**() C function to convert a string to tokens. This function maintains internal variables. If it is called by multiple threads sharing the same memory, **strtok**()'s internal variables are put in an inconsistent state.

PHP

If you want to use PHP, either you must use the **prefork** MPM or, if you want to use the **worker**MPM and PHP, you must remove **mod_php** and run PHP as a CGI script (page 969).

To run the **worker** MPM, in **/etc/httpd/conf.modules.d/00-mpm.conf**, move the hash sign from the **worker module** line to the **prefork module** line as follows:

Click here to view code image

```
#LoadModule mpm_prefork_module
modules/mod_mpm_prefork.so
LoadModule mpm_worker_module modules/mod_mpm_worker.so
```

After making this change, restart the **httpd** daemon (page 935).

webalizer: Analyzes Web Traffic

The **webalizer** package creates a directory at **/var/www/usage**, a cron file (page 607) at**/etc/cron.daily/00webalizer**, a webalizer configuration file at **/etc/webalizer.conf**, and an Apache configuration file at **/etc/httpd/conf.d/webalizer.conf**. After installing the package, restart the **httpd** daemon (page 935). Once a day, the cron file generates usage data and puts it in the **usage**directory; you can view this data by pointing a browser at **http://*server*/usage/**, where ***server*** is the name or IP address of the server. However, the cron script must have run at least once so there is data in the **usage** directory before Apache will display results and not display an error.

The **/etc/webalizer.conf** file controls the behavior of the webalizer utility. If you want information on virtual hosts (which by definition change the location of the DocumentRoot and often change the location of log files), you must modify the **/etc/webalizer.conf** file. The**/etc/httpd/conf.d/webalizer.conf** file includes a Require local directive. If you want to view statistics from a remote browser, you must add an appropriate Require directive (page 961) to this file. For more information on webalizer, refer to the webalizer man page and the sites listed under "More Information" on page 933.

MRTG: Monitors Traffic Loads

Multi Router Traffic Grapher (MRTG; **mrtg** package) is an open-source application that graphs statistics available through SNMP (Simple Network Management Protocol). SNMP information is available on all high-end routers and switches as well as on some other networked equipment, such as printers and wireless access points. You can use the **net-snmp** and **net-snmp-utils** packages supplied by Fedora/RHEL to install SNMP on a system. You also need to install the **mrtg** package. The**/etc/httpd/conf.d/mrtg.conf** file includes Require local directive. If you want to view statistics from a remote browser, you must add an appropriate Require directive (page 961) to this file.

Once MRTG and SNMP are installed and running, you can view the reports at **http://*server*/mrtg**, where ***server*** is the hostname of the server. For more

information see the mrtg man page and the sites listed under "More Information" on page 933. See also "Introduction to Cacti" on page 645 for information on capturing and displaying SNMP information.

Error Codes

Following is a list of Apache error codes:

100 Continue
101 Switching Protocols
200 OK
204 No Content
205 Reset Content
206 Partial Content
300 Multiple Choices
301 Moved Permanently
302 Moved Temporarily
303 See Other
304 Not Modified
305 Use Proxy
400 Bad Request
401 Unauthorized
402 Payment Required
403 Forbidden
404 Not Found
405 Method Not Allowed
406 Not Acceptable
201 Created
202 Accepted
203 Non-Authoritative Info...
407 Proxy Authentication Required
408 Request Timed Out
409 Conflict
410 Gone
411 Length Required
412 Precondition Failed
413 Request Entity Too Large
414 Request-URI Too Large
415 Unsupported Media Type
500 Internal Server Error
501 Not Implemented
502 Bad Gateway
503 Service Unavailable
504 Gateway Time-out
505 HTTP Version Not Supported

Chapter Summary

Apache is the most popular Web server on the Internet. It is both robust and extensible. The **/etc/httpd/conf/httpd.conf** configuration file controls many aspects of how Apache runs. The Fedora/RHEL **httpd.conf** file, which is based on the **httpd.conf** file distributed by Apache, is heavily commented.

Content to be served is typically placed in **/var/www/html**, called the *document root*. Apache automatically displays the file named **index.html** in this directory.

Configuration directives, or simply directives, are lines in a configuration file that control some aspect of how Apache functions. Four locations, called *contexts*, define where a configuration directive can appear: **server config**, **virtual host**, **directory**, and **.htaccess**. Containers, or special directives, are directives that group other directives.

To restrict access to a Web page, Apache and third parties provide authentication modules and methods that can verify a user's credentials, such as a username and password. Some modules enable authentication against various databases, including LDAP and NIS.

Apache can respond to a request for a URI by asking the client to request a different URI. This response is called a *redirect.* Apache can also process content before serving it to a client using scripting modules that work with scripts embedded in HTML documents.

Apache supports virtual hosts, which means that one instance of Apache can respond to requests directed to multiple IP addresses or hostnames as though it were multiple servers. Each IP address or hostname can provide different content and be configured differently.

The CGI (Common Gateway Interface) allows external application programs to interface with Web servers. Any program can be a CGI program if it runs in real time and relays its output to the requesting client.

SSL (Secure Sockets Layer) has two functions: It allows a client to verify the identity of a server and it enables secure two-way communication between a client and server.

Exercises

1. What is Apache? Very basically, how does it work (i.e., what happens when you point a browser at a Web page)?

2. What is the function of the document root? Where is it located by default? How would you change the location of the document root?

3. How would you tell Apache that the content is in **/usr/local/www**?

4. How would you instruct an Apache server to listen on port 81 instead of port 80?

5. How would you enable Sam to publish Web pages from his **~/website** directory but not allow anyone else to publish to the Web?

6. Apache must be started with **root** privileges. Why? Why does this action not present a security risk?

7. What is the relationship between Apache and **httpd**?

8. What does the ServerName directive do? Which value can you use as a ServerName if you want to experiment with an Apache server locally (on the server system)?

Advanced Exercises

9. If you are running Apache on a firewall system, perhaps to display a Web front end for firewall configuration, how would you make sure that it is accessible only from inside the local network?

10. Why is it more efficient to run scripts using **mod_perl** than to run them through CGI?

11. What two things does SSL provide, and how does this situation differ if the certificate is self-signed?

12. Some Web sites generate content by retrieving data from a database and inserting it into a template using PHP or CGI each time the site is accessed. Why is this practice often a poor idea?

13. Assume you want to provide Webmail access for employees on the same server that hosts the corporate Web site. The Web site address is example.com, you want to use mail.example.com for Webmail, and the Webmail application is located in **/var/www/webmail**. Describe two ways you can set up this configuration.

14. Part of a Web site is a private intranet. Describe how you would prevent people outside the company's internal 192.168.0.0/16 network from accessing this site. The site is defined as follows:

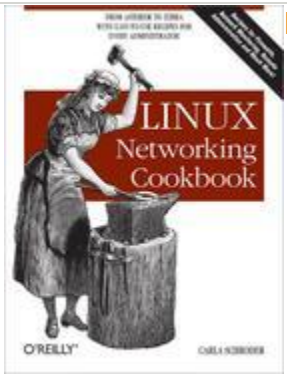
Click here to view code image

```
<VirtualHost *>
ServerName example.com
DocumentRoot /var/www
<Directory /var/www/intranet>
  AllowOverride AuthConfig
</Directory>
</VirtualHost>
```

15. What is a container? Give an example of container syntax.

16. What does CGI do? What are the characteristics of a CGI program?

17. What are the functions of SSL?

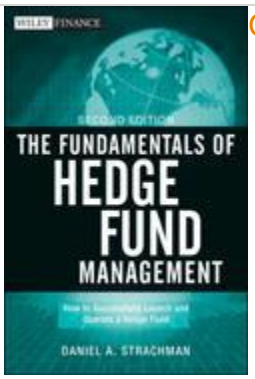


Network Monitoring with MRTGfrom: Linux Networking Cookbook by Carla Schroder*Released: November 2007*

39 MINS

- Information Technology/Operations
- Networking

BOOK SECTION

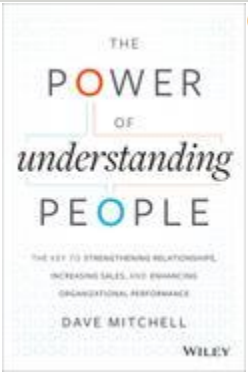


Chapter 2: The Service Providersfrom: The Fundamentals of Hedge Fund Management, 2nd Edition by Daniel A. Strachman*Released: July 2012*

36 MINS

- Personal & Professional Development

BOOK SECTION



Chapter 2: Understanding Experts and Masterminds: Tried and True Contrasted with Possibilitiesfrom: The Power of Understanding People: The Key to Strengthening Relationships, Increasing Sales, and Enhancing Organizational Performance by Dave Mitchell*Released: December 2013*

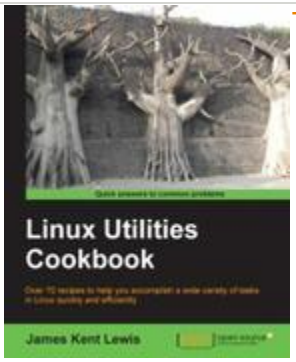
25 MINS

- Personal & Professional Development
  - BOOK SECTION

Building JavaScript Programsfrom: Microsoft® Start Here!™ Learn JavaScriptby Steve Suehring*Released: September 2012*

40 MINS

- JavaScript
- Microsoft
  - BOOK SECTION

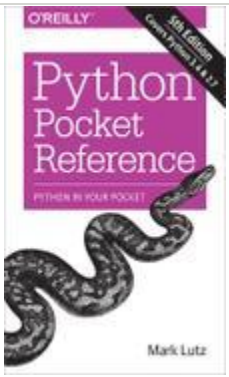


Troubleshooting bad connectionsfrom: Linux Utilities Cookbook by James Kent Lewis*Released: October 2013*

5 MINS

- Information Technology/Operations

- BOOK SECTION



Python Pocket Referencefrom: Python Pocket Reference, 5th Edition by Mark Lutz*Released: January 2014*

311 MINS

- Python

- Settings
- Blog
- Support
- Feedback
- Sign Out