CHAPTER – 7

SETTING UP A WEB SERVER

Introduction to Web Servers:

The World Wide Web, as it is known today, began as a project of Tim Berners-Lee at the European Center for Particle Physics (CERN). The original goal was to provide one consistent interface for geographically dispersed researchers and scientists who needed access to information in a variety of formats. From this idea came the concept of using one client (the Web browser) to access data (text, images, sounds, video, and binary files) from several types of servers (HTTP, FTP, SMTP, Gopher, NNTP, WAIS, Finger, and streaming-media servers).

The Web server usually has a simpler job: to accept HTTP (HyperText Transfer Protocol) requests and send a response to the client. However, this job can get much more complex (as the server can also), executing functions such as:

- Performing access control based on file permissions, user name/password pairs, and host name/IP address restrictions.
- Parsing a document (substituting appropriate values for any conditional fields within the document) before sending it to the client.
- Spawning a CGI (Common Gateway Interface) script or custom API (Application Program Interface) program to evaluate the contents of a submitted form, presenting a dynamically created document, or accessing a database.
- Sending a Java applet to the client.
- Logging any successful accesses, failures, and errors.

The Apache Web server:

The Apache Web server was originally based on HTTPd, a free server from NCSA (the National Center for Supercomputing Applications). At the time, HTTPd was the most common server on the Internet. Unfortunately, the development of the server wasn't keeping up with the needs of Webmasters, and several security problems had been discovered. Many Webmasters had been independently applying their own features and fixes to the NCSA source code. In early 1995, a group of these developers pooled their efforts and created "a patchy server," initially just a collection of patches to the HTTPd code. Since then, the Apache Group has largely rewritten the code and created a stable, multiplatform Web server daemon.

Apache is also the base for several other Web servers, most of which use Apache's freely available source code and add improved security features such as SSL (Secure Sockets Layer) for encrypted data transfer or advanced authentication modules.

The main features of the Apache Web server include:

- The stability and rapid development cycle associated with a large group of cooperative volunteer programmers.
- Full source code, downloadable at no charge.
- Ease of configuration using plain-text files.

- Access-control based on client host name/IP address or user name/password combinations.
- Support for server-side scripting as well as CGI scripts.
- A custom API that enables external modules (for example, for extended logging capabilities, improved authentication, caching, connection tracking, and so on) to be utilized by the server daemon.

Apache is not the only Web server available for Red Hat Linux, but it is the one most commonly used with Red Hat Linux, and is the most popular server used on the Internet according to recent Netcraft surveys (www.netcraft.com/survey). In addition to Apache, Red Hat Linux comes with the TUX Web server.

Starting the Apache Web Server:

If Apache wasn't installed during Red Hat installation, we can install it later from the CD-ROMs. We will need the httpd package and optionally the httpd-manual package (named apache and apache-manual in earlier versions).

Note: *It is possible for a new version of Apache to be released before an equivalent Red Hat package is available. Or perhaps we'd prefer to customize the server's compile-time options and build Apache directly from the source code. Download the full source code distribution from www.apache.org/dist/ or we can select our closest mirror from www.apache.org/dyn/closer.cgi.*

Here's a quick way to get our Apache Web server going. From here, we'll want to customize it to match our needs and our environment (as described in the section that follows).

1. Make sure that Apache is installed by typing the following from a Terminal window:

```
$ rpm -qa | grep httpd
redhat-config-httpd-1.1.0-1.1
httpd-devel-2.0.45-14
httpd-2.0.45-14
httpd-manual-2.0.45-14
```

The version number we see may be different. We need only the httpd package to get started. It is recommend httpd-manual because it has excellent information on the whole Apache setup. The httpd-devel package includes the apxstool for building and installing extension modules. The redhat-config-httpd package contains a GUI-based Apache Configuration tool.

2. A valid host name is recommended for our Apache server (for example, abc.handsonhistory.com). If we don't have a real fully-qualified domain name, we can edit the /etc/httpd/conf/httpd.conf file and define the ServerName as our computer's IP address. Open the httpd.conf file (as the root user) in any text editor, search for the line containing ServerName new.host.name:80, and uncomment it. It should appear as follows:

```
ServerName new.host.name:80
```

To make the Web server available to our LAN, we can use our IP address instead of new.host.name (for example, ServerName 10.0.0.1). The :80 represents the port number (which is the default). For a public Web server, get a real DNS host name.

3. Add an administrative e-mail address where someone can contact us in case an error is encountered with our server. In the /etc/httpd/conf/httpd.conf file, the default administrative address appears as follows:

```
ServerAdmin root@localhost
```

Change root@localhost to the e-mail address of our Apache administrator.

4. Start the httpd server. As root user, type the following:

```
# /etc/init.d/httpd start
```

If all goes well, this message should appear: Starting httpd: [OK]. Now we're ready to go.

5. To have httpd start every time we boot our system, run the command as root user.

```
# chkconfig httpd on
```

6. To make sure that the Web server is working, open Mozilla (or another Web browser) and type the following into the location box and press Enter:

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

7. We will see the Test Page for the Apache Web server, as shown in Figure below. To access this page from another computer, we will need to enter our Apache server's host name or IP address.

Figure: Appearance of the Test Page indicates that the Apache installation succeeded.

Tip: *It is not necessary to be connected to a network (or even to have a network connection) just to test the server or to view the files on our machine. Rather than specify the server's real name in the URL, just use "localhost" (that is, http://localhost/) from a browser on the same computer. In fact, it's best to fully test the configuration before making the server accessible on an unprotected network.*

8. The Test Page is actually an error condition, indicating that we haven't added any content to our Web site yet. To get started, we should add an index.html file that contains our own home page content to the /var/www/html directory. Then we can continue to add our own content to that directory structure.

Now that we have gotten our Web server to work (or at least, let's hope we have), we should step through the next section. It helps us understand how to set up more complex Web server arrangements and protect our server from misuse. Stepping through that section will also help us troubleshoot our Web server, in case it isn't working.

Tip: *If our Web server is accessible from our local host, but not available to others from our LAN or the Internet, we may need to change our firewall rules to allow greater access. In particular, we need to open port 80.*

Configuring the Apache Server:

The primary file for configuring our Apache Web server is httpd.conf (located in the /etc/httpd/conf directory). For older installations, we may also have srm.conf and access.conf files configured. In recent releases, however, Apache developers recommend us to put all directives into httpd.conf and not create the other two files.

All Apache configuration files are plain-text files and can be edited with our favorite text editor. The /etc/httpd/conf/httpd.conf file is reproduced in its entirety in the following section, with explanations inserted after related blocks of options (intended to supplement the comments provided with each file).

Some individual modules within Apache, such as perl, php, and mysql, have individual configuration files that may interest us. Those files are contained in the /etc/httpd/conf.d directory. The "Configuring modules (/etc/httpd/conf.d/*.conf)" section later in this chapter addresses modules.

☑ Configuring The Web Server (Httpd.Conf):

The httpd.conf file is the primary configuration file for the Apache Web server. It contains options that pertain to the general operation of the server. The default filename (/etc/httpd/conf/httpd.conf) can be overridden by the -f filename command-line argument to the httpd daemon or the ServerConfigFile directive. The following sections list the contents of the httpd.conf file and describe how to use the file.

The first section contains comments about the httpd.conf file:

```
#
# Based on the NCSA server configuration files originally by Rob McCool
#
# This is the main Apache server configuration file. It contains the
# configuration directives that give the server its instructions.
# See <URL:http://httpd.apache.org/docs-2.0/> for detailed information
# about the directives.
#
```

```
# Do NOT simply read the instructions in here without understanding
# what they do.  They're here only as hints or reminders.  If you are
# unsure consult the online docs. You have been warned.
#
# The configuration directives are grouped into three basic sections:
#  1. Directives that control the operation of the Apache server
#     process as a whole (the 'global environment').
#  2. Directives that define parameters of the main or default server,
#     which responds to requests that aren't handled by a virtual host.
#     These directives also provide default values for the settings
#     of all virtual hosts.
#  3. Settings for virtual hosts, which allow Web requests to be sent
#     to different IP addresses or hostnames and have them handled by
#     the same Apache server process.
#
# Configuration and logfile names: If the filenames you specify for
# many of the server's control files begin with "/" (or "drive:/" for
# Win32), the server will use that explicit path.  If the filenames do
# *not* begin with "/", the value of ServerRoot is prepended -- so
# "logs/foo.log"with ServerRoot set to "/usr/local/apache" will be
# interpreted by the server as "/usr/local/apache/logs/foo.log".
#
```

This section consists entirely of comments. It basically tells us how information is grouped together in this file and how the httpd daemon accesses this file. By default, log files are in the /var/log/httpd directory.

☑ Setting The Global Environment:

In "Section 1: Global Environment" of the httpd.conf file, we set directives that affect the general workings of the Apache server. Here is what the different directives are for:

```
### Section 1: Global Environment
#
# The directives in this section affect overall operation of Apache,
# such as the number of concurrent requests it can handle or where it
# can find its configuration files.
#
```

Revealing Subcomponents:

The ServerTokens directive lets us prevent remote computers from finding out what subcomponents we are running on our Apache server. Comment out this directive if we don't mind exposing this information. To prevent exposure, ServerTokens is set as follows:

```
#
# Don't give away too much information about all the subcomponents
# we are running.  Comment out this line if you don't mind remote sites
# finding out what major optional modules you are running
ServerTokens OS
```

Setting the Server Root Directory:

The ServerRoot directive specifies the directory that contains the configuration files, a link to the log file directory, and a link to the module directory. An alternative ServerRoot path name can be specified using the -d command-line argument to httpd.

```
# ServerRoot: The top of the directory tree under which the server's
# configuration, error, and log files are kept.
#
# NOTE! If you intend to place this on an NFS (or other network) mounted
# filesystem then please read the LockFile documentation (available
# at <URL:http://httpd.apache.org/docs-2.0/mod/core.html#lockfile>);
# you will save yourself a lot of trouble.
#
# Do NOT add a slash at the end of the directory path.
#
ServerRoot "/etc/httpd"
```

Storing The Server's Scoreboard File:

On some systems, a ScoreBoard file is created to store internal server process information. Red Hat Linux does not need to use the ScoreBoardFile, as the status information usually saved in that file is stored in memory instead.

Storing the Server's PID File:

The Apache Web server keeps track of the PID for the running server process. We can change the locations of this file using the entry described next:

```
#
# PidFile: The file in which the server should record its process
# identification number when it starts.
#
PidFile run/httpd.pid
```

Apache uses the PidFile to store the process ID of the first (root-owned) master daemon process. This information is used by the /etc/init.d/httpd script when shutting down the server and also by the server-status handler (as described later).

Configuring Timeout Values:

We can set several values that relate to timeout. Some of these values are described in the text following the code:

```
#
# Timeout: The number of seconds before receives and sends time out.
#
Timeout 300


#
# KeepAlive: Whether or not to allow persistent connections (more than
# one request per connection). Set to "Off" to deactivate.
#
KeepAlive Off


#
# MaxKeepAliveRequests: The maximum number of requests to allow
# during a persistent connection. Set to 0 to allow unlimited amount.
# We recommend you leave this number high, for maximum performance.
#
MaxKeepAliveRequests 100


#
# KeepAliveTimeout: Number of seconds to wait for the next request from
# the same client on the same connection.
#
KeepAliveTimeout 15
```

The Timeout directive determines the number of seconds that Apache will hold a connection open between the receipt of packets for a PUT or POST HTTP request method, between the receipt of acknowledgments on sent responses, or while receiving an incoming request. The default of five minutes (300 seconds) can certainly be lowered if we find an excessive number of open, idle connections on our machine.

The KeepAlive directive instructs Apache to hold a connection open for a period of time after a request has been handled. This enables subsequent requests from the same client to be processed faster, as a new connection (to a different server process, most likely) doesn't need to be created for each request.

The MaxKeepAliveRequests directive sets a limit on the number of requests that can be handled with one open connection. The default value is certainly reasonable because most connections will hit the KeepAliveTimeout before MaxKeepAliveRequests.

The KeepAliveTimeout directive specifies the number of seconds to hold the connection while awaiting another request. We may wish to increase the default (15 seconds), depending on how long it may take a client to peruse our average page and select a link from it.

☑ Setting The Number Of Server Processes:

To allow Apache to dynamically change the number of server processes running to meet the demands of the server, several parameters are set to indicate:

- How many server processes should be started (StartServers)
- The minimum number of server processes kept spare (MinSpareServers)
- The maximum number of server processes kept spare (MaxSpareServers)
- The maximum number of server processes allowed to start (MaxClients)
- The maximum number of requests a process can serve (MaxRequestsPerChild)

With the Multi-Processing Module (MPM) feature in Apache 2.0, minimum and maximum server options can be set to take advantage of more processors and threads. As load increases, the number of processes available to handle it increases, based on these settings. Because starting each process takes time, spare servers are there to handle occasional spikes in load.

```
##
## Server-Pool Size Regulation (MPM specific)
##
<IfModule prefork.c>
StartServers       8
MinSpareServers    5
MaxSpareServers   20
MaxClients       150
MaxRequestsPerChild  1000
</IfModule>

<IfModule worker.c>
StartServers         2
MaxClients         150
MinSpareThreads     25
MaxSpareThreads     75
ThreadsPerChild     25
MaxRequestsPerChild  0
</IfModule>

<IfModule perchild.c>
```

```
NumServers           5
StartThreads         5
MinSpareThreads      5
MaxSpareThreads     10
MaxThreadsPerChild  20
MaxRequestsPerChild  0
</IfModule>
```

Apache starts a master daemon process owned by root that binds to the appropriate Port, then switches to a nonprivileged user. More servers (equivalent to the value of the StartServers directive) will then be started as the same nonprivileged user (the apache user in this case).

Apache attempts to intelligently start and kill servers based on the current load. If the amount of traffic decreases and there are too many idle servers, some will be killed (down to the number of servers noted in MinSpareServers). Similarly, if many requests arrive in close proximity and there are too few servers waiting for new connections, more servers will be started (up to the number of servers noted in MaxSpareServers).

Using the values specified above, when the daemon is started, eight server processes will run, waiting for connections (as defined by StartServers). As more requests arrive, Apache will ensure that at least eight servers are ready to answer requests. When requests have been fulfilled and no new connections arrive, Apache will begin killing processes until the number of idle Web server processes is below 20. The value of StartServers should always be somewhere between MinSpareServers and MaxSpareServers.

Apache limits the total number of simultaneous server processes with the MaxClients directive. The default value is 150, which should be sufficiently high. However, if we find that we frequently have nearly that many servers running, remember that any connection beyond the 150th will be rejected. In such cases, if our hardware is sufficiently powerful (and if our network connection can handle the load), we should increase the value of MaxClients.

To minimize the effect of possible memory leaks (and to keep the server pool "fresh"), each server process is limited in the number of requests that it can handle (equal to the value of MaxRequestsPerChild). After servicing 1000 requests (the value specified above), the process will be killed. It is even more accurate to say that each process can service 1000 *connections* because all KeepAlive requests (occurring prior to encountering a KeepAliveTimeout) are calculated as just one request.

In a multiprocessor environment, setting thread values described previously can both limit the number of threads that servers can consume and supply as many threads as we will allow to handle server processing. MinSpareThreads and MaxSpareThreads control the number of threads available that are not being used. More are added if available threads fall below MinSpareThreads. If spare threads go above MaxSpareThreads, some are dropped.

☑ Binding To Specific Addresses:

We can bind to specific IP addresses using the Listen directive. Listen directives can be used to add to the default bindings we already have:

```
# Listen: Allows you to bind Apache to specific IP addresses and/or
# ports, in addition to the default. See also the <VirtualHost>
# directive.
#
# Change this to Listen on specific IP addresses as shown below to
# prevent Apache from glomming onto all bound IP addresses (0.0.0.0)
#Listen 12.34.56.78:80
Listen 80
```

The Listen directive is more flexible than the BindAddress and Port directives. Multiple Listen commands can be specified, enabling us to specify several IP address/port number combinations. It can also be used to specify just IP addresses (in which case the Port directive is still necessary) or just port numbers. By default, Apache listens to port 80 on all interfaces on the local computer (which is where Web browsers expect to find Web content).

☑ Including Module-Specific Configuration Files:

The following lines cause Apache to load configuration files from the /etc/httpd/conf.d directory. This directory contains configuration files associated with specific modules.

```
#
# Load config files from the config directory "/etc/httpd/conf.d".
#
Include conf.d/*.conf
```

☑ Selecting Modules In Httpd.Conf:

During the compilation process, individual Apache modules can be selected for dynamic linking. Dynamically-linked modules are not loaded into memory with the httpd server process unless LoadModule directives explicitly identify those modules to be loaded. The blocks of code that follow select several modules to be loaded into memory by using the LoadModule directive with the module name and the path to the module (relative to ServerRoot). The following text shows a partial listing of these modules:

```
#
# Dynamic Shared Object (DSO) Support
# Example:
# LoadModule foo_module modules/mod_foo.so
#
LoadModule access_module modules/mod_access.so
LoadModule auth_module modules/mod_auth.so
LoadModule auth_anon_module modules/mod_auth_anon.so
LoadModule auth_dbm_module modules/mod_auth_dbm.so
```

```
LoadModule auth_digest_module modules/mod_auth_digest.so
LoadModule include_module modules/mod_include.so
LoadModule log_config_module modules/mod_log_config.so
LoadModule env_module modules/mod_env.so
LoadModule mime_magic_module modules/mod_mime_magic.so
      .
      .
      .
LoadModule actions_module modules/mod_actions.so
LoadModule speling_module modules/mod_speling.so
LoadModule userdir_module modules/mod_userdir.so
LoadModule alias_module modules/mod_alias.so
LoadModule rewrite_module modules/mod_rewrite.so
```

Apache modules are included in the list of active modules via the LoadModule directive. The ClearModuleList directive removes all entries from the current list of active modules. Each of the standard modules that come with Apache is described in Table below.

Module	Description
mod_access	Provides access control based on originating host name/IP address.
mod_actions	Conditionally executes CGI scripts based on the file's MIME type or the request method.
mod_alias	Allows for redirection and mapping parts of the physical file system into logical entities accessible through the Web server.
mod_asis	Enables files to be transferred without adding any HTTP headers (for example, the Status, Location, and Content-type header fields).
mod_auth	Provides access-control based on user name/password pairs. The authentication information is stored in a plain-text file, although the password is encrypted using the crypt()system call.
mod_auth_anon	Similar to anonymous FTP, this module enables predefined user names access to authenticated areas by using a valid e-mail address as a password.
mod_auth_dbm	Provides access-control based on user name/password pairs. The authentication information is stored in a DBM binary database file, with encrypted passwords.
mod_auth_digest	Provides MD5 Digest user authentication.
mod_auth_ldap	Let us use an LDAP directory to store the HTTP Basic authentication database.
mod_autoindex	Implements automatically generated directory indexes.
mod_cache	Allows local or proxied Web content to be cached. Used with the mod_disk_cache or mod_file_cache modules.
mod_cern_-meta	Offers a method of emulating CERN HTTPD meta file semantics.
mod_cgi	Controls the execution of files that are parsed by the "cgi-script" handler or that have a MIME type of x-httpd-cgi.
mod_dav	Provides Web-based Distributed Authoring and Versioning (WebDAV) to copy, create, move, and delete resources.

mod_dav_fs	Used to provide filesystem features to the mod_dav module.
mod_deflate	Includes the DEFLATE output filter, to compress data before it is sent to the client.
Mod_dir	Sets the list of filenames that may be used if no explicit filename is selected in a URL that references a directory.
mod_disk_cache	Enables a disk-based storage manager to use with mod_proxy.
mod_env	Controls environment variables passed to CGI scripts.
mod_expires	Implements time limits on cached documents by using the Expires HTTP header.
mod_file_cache	Allows caching of frequently requested static files.
mod_headers	Enables the creation and generation of custom HTTP headers.
mod_imap	Controls inline image map files, which have a MIME type of x-httpd-imap or that are parsed by the imap handler.
mod_include	Implements Server-Side Includes (SSI), which are HTML documents that include conditional statements parsed by the server prior to being sent to a client.
mod_info	Provides a detailed summary of the server's configuration, including a list of actively loaded modules and the current settings of every directive defined within each module.
mod_ldap	Used to speed performance of Web sites using LDAP servers.
mod_log_config	Enables a customized format for information contained within the log files.
mod_mem_cache	Used with mod_cache to provide memory-based storage.
mod_mime	Alters the handling of documents based on predefined values or the MIME type of the file.
mod_mime_magic	Similar to the UNIX file command, this module attempts to determine the MIME type of a file based on a few bytes of the file's contents.
mod_negotiation	Provides for the conditional display of documents based upon the Content-Encoding, Content-Language, Content-Length, and Content-Type HTTP header fields.
mod_proxy	Implements an HTTP 1.1 proxy/gateway server.
mod_proxy_connect	Entension to mod_proxy to handle CONNECT requests.
mod_proxy_ftp	Extension to mod_proxy to handle FTP requests.
mod_proxy_http	Extension to mod_proxy to handle HTTP requests.
mod_rewrite	Provides a flexible and extensible method for redirecting client requests and mapping incoming URLs to other locations in the file system.
mod_setenvif	Conditionally sets environment variables based on the contents of various HTTP header fields.
mod_so	The only module other than http_core that must be statically compiled in the server, this module contains the directives necessary to implement loading dynamic shared objects.
mod_speling	Attempts to automatically correct misspellings in requested URLs.
mod_ssl	Implements cryptography using SSL and TLS protocols.
mod_status	Provides a summary of the activities of each individual httpd server process, including CPU and bandwidth usage levels.
mod_suexec	Lets CGI scripts run with permission of particular user/group.
mod_unique_id	Attempts to assign a token to each client request that is unique across all server processes on all machines within a cluster.
mod_userdir	Specifies locations that can contain individual users' HTML documents.
mod_usertrack	Uses cookies to track the progress of users through a Web site.

mod_vhost_alias	Contains support for dynamically configured mass virtual hosting.

If a particular module contains features that are not necessary, it can easily be commented out of the above list. Similarly, we may want to add the features or functionality of a third-party module (for example, mod_perl, which integrates the Perl run-time library for faster Perl script execution, or mod_php, which provides a scripting language embedded within HTML documents) by including those modules in the lists above.

More information about each module (and the directives that can be defined within it) can be found on our server at http://localhost/manual/mod/.

☑ Setting The Main Server's Configuration:

The second section of the http.conf file relates to directives handled by our main server. In other words, these values are used in all cases except when they are changed in virtual host definitions. To change the same directives for particular virtual hosts, add them within virtual host containers.

☑ Choosing The Server's User And Group:

The httpd daemon doesn't have to run as the root user; in fact, our system is more secure if it doesn't. By setting User and Group entries, we can have the httpd daemon run using the permissions associated with a different user and group:

```
# If you wish httpd to run as a different user or group, you must run
# httpd as root initially and it will switch.
#
# User/Group: The name (or #number) of the user/group to run httpd as.
#  . On SCO (ODT 3) use "User nouser" and "Group nogroup".
#  . On HPUX you may not be able to use shared memory as nobody, and
#  the suggested workaround is to create a user www and use that user.
#  NOTE that some kernels refuse to setgid(Group) or semctl(IPC_SET)
#  when the value of (unsigned)Group is above 60000;
#  don't use Group #-1 on these systems!
#
User apache
Group apache
```

By default, apache is defined as both the user and group for the server. If we change the User and Group directives, we should specify a nonprivileged entity. This minimizes the risk of damage if our site is compromised. The first daemon process that is started runs as root. This is necessary to bind the server to a low-numbered port and to switch to the user and group specified by the User and Group directives. All other servers run under the user ID (UID) and group ID (GID) defined by those directives.

Choosing the HTTP Port Number:

Apache listens on particular ports for HTTP requests. The port that is used is set by the Port entry as follows:

```
#
# Port: The port to which the standalone server listens. For
# ports < 1023, you will need httpd to be run as root initially.
#
Port 80
```

Port 80 is the default for HTTP traffic, which is why http://www.apache.org:80/ is the same as http://www.apache.org/. If the server is bound to a different port, the port number must be specified in the URL. Most Web servers run on port 80, although they can accept connections on any port (below 65536) that is not already bound to a particular service (see the /etc/servicesfile for a list of common services/protocols and the ports they use). Only root can run programs that listen for connections on privileged ports (those below 1024).

Setting an E-Mail Address:

We can identify an address where users can send e-mail if there is a problem with our server. This is done with the ServerAdmin directive:

```
#
# ServerAdmin: Your address, where problems with the server should be
# e-mailed.  This address appears on some server-generated pages, such
# as error documents. e.g. admin@your-domain.com
#
ServerAdmin you@your.address
```

The ServerAdmin directive can be set to any valid e-mail address. It can direct mail to a user account (on the same or another machine) or redirect mail to an alias (such as Webmaster) that could distribute mail to several individuals. The default is root@localhost.

Setting the Server Name:

If our server name is anything but our exact registered host/domain name, we should identify our server name here. As the comments point out, the ServerName directive can be set to a value other than the actual host name of our machine. However, this other name should still refer to our machine in DNS if the server is to be a public Internet server. Frequently, www is just an alias for the real name of the machine (for example, a machine may respond to www.linuxtoys.net, but its real name may be al.linuxtoys.net).

```
ServerName jukebox.linuxtoys.net
```

Apache tries to use our host name as the ServerName if we don't enter a valid server name. It is recommended that we explicitly enter a ServerName here.

Setting Canonical Names:

Use the UseCanonicalName directive to create a self-referencing URL, as follows:

```
## UseCanonicalName: Determines how Apache constructs self-referencing
# URLs and the SERVER_NAME and SERVER_PORT variables.
# When set "Off", Apache will use the Hostname and Port supplied
# by the client.  When set "On", Apache will use the value of the
# ServerName directive.
#
UseCanonicalName Off
```

The UseCanonicalName directive provides a form of naming consistency. When it is set to on, Apache uses the ServerNameand Port directives to create a URL that references a file on the same machine (for example, http://www.linuxtoys.net/docs/). When UseCanonicalName is Off, the URL consists of whatever the client specified (for example, the URL could be http://al.linuxtoys.net/docs/ or http://al/docs/ if the client is within the same domain).

This can be problematic, particularly when access-control rules require user name/password authentication: if the client is authenticated for the host al.linuxtoys.net but a link sends him or her to www.linuxtoys.net (physically the same machine), the client will be prompted to enter a user name and password again. It is recommended that UseCanonicalName be set to on. In the preceding situation, the authentication would not need to be repeated, as any reference to the same server would always be interpreted as www.linuxtoys.net.

Identifying HTTP Content Directories:

There are several directives for determining the location of our server's Web content. The main location for our Web content is set to /var/www/html by the DocumentRoot directive. (Note that this location has changed from versions of Red Hat Linux prior to Red Hat Linux 7. The location was formerly in /home/http.)

```
# DocumentRoot: The directory out of which you will serve your
# documents. By default, all requests are taken from the directory, but
# symbolic links and aliases may be used to point to other locations.
DocumentRoot "/var/www/html"
```

Setting Access Options And Overrides:

We can set individual access permissions for each directory in the Web server's directory structure. The default is fairly restrictive. Here is the default:

```
<Directory />
    Options FollowSymLinks
    AllowOverride None
</Directory>
```

This segment sets up a default block of permissions for the Options and AllowOverride directives. The <Directory /> . . . </Directory> tags enclose the directives that are to be applied to the / directory (which is /var/www/html by default).

The Options FollowSymLinks directive instructs the server that symbolic links within the directory can be followed to allow content that resides in other locations on the computer. None of the other special server features will be active in the /directory, or in any directory below that, without being explicitly specified later. Next, the following access options are specifically set for the root of our Web server (/var/www/html).

```
<Directory "/var/www/html">
   Options Indexes FollowSymLinks
   AllowOverride None
   Order allow,deny
   Allow from all
</Directory>
```

If we have changed the value of DocumentRoot earlier in this file, we need to change the /var/www/html to match that value. The Options set for the directory are Indexes and FollowSysLinks. Those and other special server features are described in Table 21-2. The AllowOverride None directive instructs the server that the .htaccess file (or the value of AccessFileName) cannot override any of the special access features. We can replace None with any of the special access features described in Table below.

Note: *Remember that unless we specifically enable a feature described in Tables below, that feature is not enabled for our server (with the exceptions of Indexes and FollowSymLinks).*

Feature	Description
ExecCGI	The execution of CGI scripts is permitted.
FollowSymLinks	The server will traverse symbolic links.
Includes	Server-Side Includes are permitted.
IncludesNOEXEC	Server-Side Includes are permitted, except the #exec and #include elements.
Indexes	If none of the files specified in the DirectoryIndex directive exists, a directory index will be generated.
MultiViews	The server allows a content-based filename pattern-matching search.
SymLinksIfOwnerMatch	The server will traverse symbolic links only if the owner of the target is the same as the owner of the link.
None	None of the features above are enabled.
All	All the features above are enabled, with the exception of MultiViews. This must be explicitly enabled.

Feature	Description
AuthConfig	Enables authentication-related directives (AuthName, AuthType, AuthUserFile, AuthGroupFile, Require, and so on).
FileInfo	Enables MIME-related directives (AddType, AddEncoding, AddLanguage, LanguagePriority, and so on).

Indexes	Enables directives related to directory indexing (FancyIndexing, DirectoryIndex, IndexOptions, IndexIgnore, HeaderName, ReadmeName, AddIcon, AddDescription, and so on).
Limit	Enables directives controlling host access (Allow, Deny, and Order).
Options	Enables the Options directive (as described in Table 21-5).
None	None of the access features above can be overridden.
All	All the access features above can be overridden.

☑ Disabling Indexes In Root Directory:

The following directives disable indexing for the root directory (probably /var/www/html), so that our index.html, or the default one put there by Apache, is displayed when that directory is requested. (The default way of listing the contents of directories on our Apache Web server is defined in the "Defining Indexing" section later in this chapter).

```
#
# Disable autoindex for the root directory, and present a
# default Welcome page if no other index page is present.
#
<LocationMatch "^/$>
    Options -Indexes
    ErrorDocument 403 /error/noindex.html
</LocationMatch>
```

☑ Identifying User Directories:

We can identify the name that is appended to a user's home directory when a request for a user's directory (~*user*) is received. This feature used to be set to public_html by default; however, it is now turned off by default.

To allow access to our users' personal Web pages, add a comment character (#) to the UserDir disable line. Then remove the # from the UserDir public_html line to make users' personal public_html directories accessible through the Web server. After removing extra comment lines, the following text shows what the enabled section looks like:

```
# UserDir: The name of the directory which is appended onto a user's
# home directory if a ~user request is received.
<ifModule mod_userdir.c>
#  UserDir disable
UserDir public_html
</IfModule>
```

Besides uncommenting the UserDir public_html line shown above, we must make both the user's home directory and public_html directory executable by everyone in order for the UserDir directive to allow access to a particular user's public_html directory. For example, the user cjb could type the following to make those directories accessible.

```
$ chmod 711 /home/cjb
$ mkdir /home/cjb/public_html
$ chmod 755 /home/cjb/public_html
```

For UserDir to work, the mod_userdir module must also be loaded (which it is by default).

There are two ways in which the UserDir directive can handle an incoming request that includes a user name (for example, ~cjb). One possible format identifies the physical path name of the individual users' publicly accessible directories. The other can specify a URL to which the request is redirected. A few examples are presented in Table below, using the URL http://www.mybox.com/~cjb/proj/c004.html as a sample request.

Table: UserDir Path Name and URL Examples

UserDir Directive	Referenced Path or URL
UserDir public_html	~cjb/public_html/proj/c004.html
UserDir /public/*/WWW	/public/cjb/WWW/proj/c004.html
UserDir /usr/local/web	/usr/local/web/cjb/proj/c004.html
UserDir http://www.mybox.com/users	http://www.mybox.com/users/cjb/proj/c004.html
UserDir http://www.mybox.com/~*	http://www.mybox.com/~cjb/proj/c004.html
UserDir http://www.mybox.com/*/html	http://www.mybox.com/cjb/html/proj/c004.html

The UserDir directive can also be used to explicitly allow or deny URL-to-path name translation for particular users. For example, it is a good idea to include the following line to avoid publishing data that shouldn't be made public:

```
UserDir disable root
```

Alternatively, use the following lines to disable the translations for all but a few users:

```
UserDir disable
UserDir enable wilhelm cjb jsmith
```

The DirectoryIndex directive establishes a list of files that is used when an incoming request specifies a directory rather than a file. For example, a client requests the URL http://www.mybox.com/~jsmith. Because it's a directory, it is automatically translated to http://www.mybox.com/~jsmith/. Now that directory is searched for any of the files listed in the DirectoryIndexdirective. The first match (from the default list of index.html and index.html.var) is used as the default document in that directory. If none of the files exist and the Indexes option (as in the httpd.conf file) is selected, the server will automatically generate an index of the files in the directory.

```
# DirectoryIndex: sets the file that Apache will serve if a directory
# is requested.
#
# The index.html.var file (a type-map) is used to deliver content-
# negotiated documents.  The MultiViews Option can be used for the
```

```
# same purpose, but it is much slower.
#
```

DirectoryIndex index.html index.html.var

Setting Directory-Access Control:

We can add an access file to each directory to control access to that directory. By default, the AccessFileName directive sets .htaccess as the file containing this information. The following lines set this filename and prevent the contents of that file from being viewed by visitors to the Web site. If we change the file to a name other than .htaccess, be sure to change the line below ("\.ht") that denies access to that file.

```
#
# AccessFileName: The name of the file to look for in each directory
# for access control information.
#
AccessFileName .htaccess
<Files ~ "^\.ht">
    Order allow,deny
    Deny from all
</Files>
```

We can add the same access directives to a .htaccess file as as do to the httpd.conf file. In general, it is more efficient to use a <Directory> directive in the httpd.conf file than it is to create a .htaccess file. Because directives you put in .htaccess apply to all directives below the current directory, any time we add a .htaccess file to a directory, Apache must search all directories above that point (for example, /, /var, /var/www. and so on) to include settings from possible .htaccess files in those directories as well.

Setting MIME-Type Defaults:

The location of the MIME type definitions file is defined by the TypesConfig directive. The DefaultType directive sets the MIME type:

```
# TypesConfig describes where the mime.types file (or equivalent) is
# to be found.
#
TypesConfig /etc/mime.types
#
# DefaultType is the default MIME type the server will use for a
# document if it cannot otherwise determine one, such as from filename.
# extensions If your server contains mostly text or HTML documents,
# "text/plain" is a good value.  If most of the content is binary, such
# as applications or images, you may want to use "application/octet-
```

```
# stream" instead to keep browsers from trying to display binary files
# as though they are text.
#
DefaultType text/plain
```

Using the mod_mime_magic module, a server can look for hints to help figure out what type of file is being requested. We must make sure this module is loaded to Apache for it to be used (it is loaded by default). The module can use hints from the /usr/share/magic.mime (off by default) and /etc/httpd/conf/magic (on by default) files to determine the contents of a requested file. Here are the directives that cause that module to be used:

```
<IfModule mod_mime_magic.c>
#   MIMEMagicFile /usr/share/magic.mime
    MIMEMagicFile conf/magic
</IfModule>
```

Setting Host Name Lookups:

With the Apache Web server, we can have the server look up addresses for incoming client requests. Turning on the HostnameLookups entry can do this:

```
# HostnameLookups: Log the names of clients or just their IP addresses
HostnameLookups Off
```

If the HostnameLookups directive is turned on, every incoming connection will generate a DNS lookup to translate the client's IP address into a host name. If our site receives many requests, the server's response time could be adversely affected. The HostnameLookups should be turned off unless you use a log file analysis program or statistics package that requires fully qualified domain names and cannot perform the lookups on its own. The logresolve program that is installed with the Apache distribution can be scheduled to edit log files by performing host name lookups during off-peak hours.

Configuring HTTP logging:

We can set several values related to logging of Apache information. When a relative path name is shown, the /etc/httpddirectory is appended (for example, /e)

Starting and Stopping the Server:

The procedure for starting and stopping the Apache Web server is no different from that of many other server processes. We can use the chkconfig command to set the httpd service to start at boot time.

The /etc/init.d/httpd shell script accepts any of five command-line arguments. If it is called with the argument start, the httpd script will run one master daemon process (owned by "root"), which will spawn other daemon processes (equal to the number specified by the StartServers directive) owned by the user apache (from the User and Group directives).

These processes are responsible for responding to incoming HTTP requests. If called with stop, the server will be shut down as all httpd processes are terminated.

If given a command-line argument of restart, the script will simply execute stop and start procedures in sequence. Using reload as the argument will send the hangup signal (-HUP) to the master httpd daemon, which causes it to reread its configuration files and restart all the other httpd daemon processes. The shell script also supports an argument of status, which will report if the daemon is running and, if it is, the PIDs of the running processes. All other command-line arguments result in an error and cause a usage message to be printed.

The actual server daemon for Apache, /usr/sbin/httpd, supports several command-line arguments, although the default values are typically used. The possible command-line arguments are listed in Table below.

Argument	Description
-c directive	Read the configuration files and then process the directive. This may supersede a definition for the directive within the configuration files.
-C directive	Process the directive and then read the configuration files. The directive may alter the evaluation of the configuration file, but it may also be superseded by another definition within the configuration file.
-d directory	Use directory as the ServerRoot directive, specifying where the module, configuration, and log file directories are located.
-D parameter	Define parameter to be used for conditional evaluation within the IfDefine directive.
-f file	Use file as the ServerConfigFile directive, rather than the default of /etc/httpd/conf/httpd.conf.
-h	Display a list of possible command-line arguments.
-l	List the modules linked into the executable at compile time:
-L	Print a verbose list of directives that can be used in the configuration files, along with a short description and the module that contains each directive.
-S	List the configured settings for virtual hosts.
-t	Perform a syntax check on the configuration files. The results will either be: Syntax OK or an error notification, for example: Syntax error on line 118 of /etc/httpd/conf/httpd.conf:
-T	Same as -t, except that there is no check of the DocumentRoot value.
-v	Print the version information: Server version: Apache/2.0.40 Server built: Feb 11 2003 12:02:50
-V	List the version information and any values defined during compilation: Server version: Apache/2.0.40 Server built: Feb 11 2003 12:02:50 Server's Module Magic Number: 20020628:0 Architecture: 32-bit Server compiled with.... -D APACHE_MPM_DIR="server/mpm/prefork" -D APR_HAS_MMAP

	-D APR_HAVE_IPV6 ??????????. ??????????. ??????????. -D SERVER_CONFIG_FILE="conf/httpd.conf"
-X	Only the single master daemon process is started, and no other httpd processes will be spawned. This should be used only for testing purposes directly from the command line.

Monitoring Server Activities:

Apache provides two unique built-in methods to check the performance of your Web server. The server-status handler can be configured to show information about server processes. The server-info handler can be configured to display a detailed summary of the Web server's configuration. We can activate these services by adding the following lines to the /etc/httpd/conf/httpd.conf file, respectively:

```
<Location /server-status>
    SetHandler server-status
    Order deny,allow
    Deny from all
    Allow from handsonhistory.com
</Location>
<Location /server-info>
    SetHandler server-info
    Order deny,allow
    Deny from all
    Allow from handsonhistory.com
</Location>
```

In this example, all computers in the handsonhistory.com domain can display the server-info and server-status pages. We can change handsonhistory.com to the name of any domain or host that our Apache server is hosting.

☑ Displaying Server Information:

The Server Information (server-info) page contains the server version information and various general configuration parameters and breaks up the rest of the data by module. Each loaded module is listed, with information about all directives supported by that module, and the current value of any defined directives from that module.

The Server Information is usually quite verbose and contains much more information than can be displayed in Figure below, which shows only the links to each module's section and the general Server Settings section.

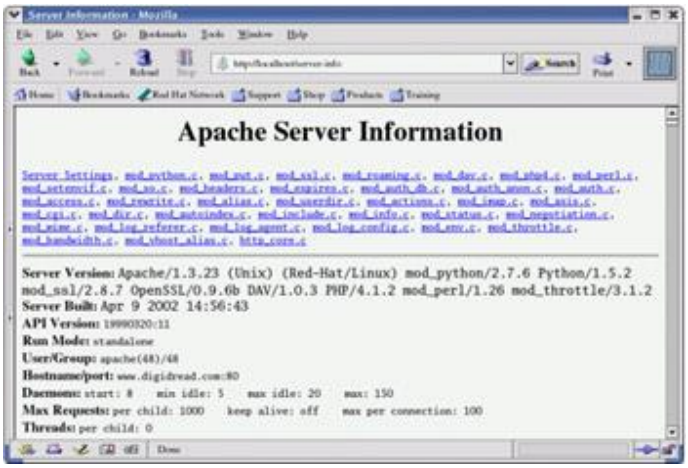

Figure: The server-info page displays server and module information.

☑ Displaying Server Status:

The contents of the server-status page include version information for the server, the current time, a timestamp of when the server was last started, and the server's uptime. The page also details the status of each server process, choosing from several possible states (waiting for a connection, just starting up, reading a request, sending a reply, waiting to read a request before reaching the number of seconds defined in the KeepAliveTimeout, performing a DNS lookup, logging a transaction, or gracefully exiting).

The bottom of the server-status page lists each server by process ID (PID) and indicates its state, using the same possible values. Figure below is an example of this page.

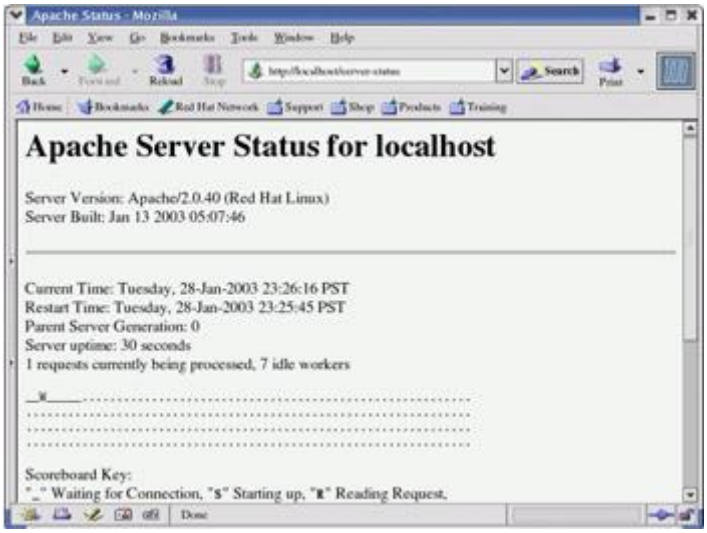

Figure: The Apache server-status page displays General Apache information and reports on individual server process activities.

The server-status page can also perform automatic updates to provide even closer monitoring of the server. If the URL http://localhost/server-status?refresh=40 is specified, the server-status page displayed in our browser will be updated every 40 seconds. This enables a browser window to be devoted entirely to continually monitoring the activities of the Web server.

By default, only basic status information is generated. If we would like to generate full status information, we need to turn on the ExtendedStatus directive by uncommenting the last line in the following code:

```
#
# ExtendedStatus: controls whether Apache will generate "full" status
# information (ExtendedStatus On) or basic information (ExtendedStatus
# Off) when the "server-status" handler is called. The default is Off.
#
#ExtendedStatus On
```

☑ Further Security Of Server-Info And Server-Status:

Because both the server-info and server-status pages contain private information that should not be accessible to just anyone on the network, there are a few extra ways you can secure that information. We can restrict that information only to the local host; however, in some environments that may not be practical.

If we must allow other machines or networks access to such detailed configuration information, allow only as many machines as is necessary, and preferably only those machines on our local network. Also, be aware that, in the wrong hands, the information displayed by the server-info and server-status pages can make it much easier for the security of our entire machine to be compromised.

It may also be beneficial to change the URL used to reference both of the aforementioned pages. This is an example of "security through obscurity," which should not be relied on but which can make it just a little more difficult for unauthorized individuals to obtain information about our Web server's configuration (particularly if we cannot restrict such connections to the local network). To accomplish this, simply change the filename in the Location directive, as in the lines below:

```
<Location /server.information.page>
and:
<Location /server.status.page>
```

☑ Logging Errors:

The error log contains messages generated by the server that describe various error conditions. The ErrorLog and LogLeveldirectives in the httpd.conf file (as described in the section on configuring the server) can modify the filename and the amount of information that is logged. The default file is /etc/httpd/logs/error_log (which is a link to /var/log/httpd/error_log). A few sample lines from the error log are shown below:

```
[Thu Feb 15 10:29:13 2003] [notice] Apache/2.0.40 (Red Hat/Linux)
       configured -- resuming normal operations
[Thu Feb 15 10:43:07 2003] [error] [client 127.0.0.1] client denied by
       server configuration: /var/www/html/server-status
[Thu Feb 15 10:06:42 2003] [error] [client 127.0.0.1] File
       does not exist: /var/www/html/newfile.html
[Thu Feb 15 01:12:28 2003 [notice] caught SIGTERM, shutting down
```

The first line indicates that the server has just been started and will be logged regardless of the LogLevel directive. The second line indicates an error that was logged to demonstrate a denied request. The third line shows an error, which represents a request for a file that does not exist. The fourth line, also logged regardless of the LogLevel directive, indicates that the server is shutting down. The error log should also be monitored periodically because it will contain the error messages from CGI scripts that may need repair.

☑ Logging Transfers:

Every incoming HTTP request generates an entry in the transfer log (by default, /etc/httpd/logs/access_log, which is a link to /var/log/httpd/access_log). Statistics packages and log file analysis programs typically use this file because manually reading through it can be a rather tedious exercise.

The format of the transfer log can be altered by the LogFormat and CustomLog directives in the httpd.conf file (as described in the "Configuring the Server" section). If you attempted to access http://localhost/ following the installation procedure (from Figure 21-1), the following lines (in the "common" format) would be written to the access_log:

```
127.0.0.1 - - [15/Feb/2003:23:32:28 -0400] "GET / HTTP/1.1" 200 1945
127.0.0.1 - - [15/Feb/2003:23:32:36 -0400] "GET /powered_by.gif
       HTTP/1.1" 200 1817
127.0.0.1 - - [15/Feb/2003:23:32:36 -0400] "GET /icons/apache_pb.gif
       HTTP/1.1" 200 2326
```

Viewing the server-info and server-status pages (as shown in Figures above, respectively) generated the following entries:

```
127.0.0.1 - - [15/Feb/2003:23:40:41 -0400] "GET /server-info HTTP/1.1"
     200 42632
127.0.0.1 - - [15/Feb/2003:23:41:49 -0400] "GET /server-status
     HTTP/1.1" 200 1504
```

The denied attempt to access the server-status page logged the following line (note the 403 server response code):

Analyzing Web-server traffic

```
127.0.0.1 - - [15/Feb/2003:23:43:07 -0400] "GET /server-statu
s
     HTTP/1.1" 403 211
```

The webalizer command can take Apache log files and produce usage reports for our server. Those reports are created in HTML format so we can display the information graphically. Information is produced in both table and graph form.

To use the webalizer command, the webalizer package must be installed. We can run webalizer with no options and have it take the values in the /etc/webalizer.conf files to get the information it needs. As an alternative, we can use command-line options to override settings in the webalizer.conf file. To use the defaults, simply run the following:

```
# webalizer
```

If all goes well, the command should run for a few moments and exit silently. Based on the information in the /etc/webalizer.conf file, the /var/log/httpd/access_log log file is read and an index.html file is copied to the /var/www/html/usage/ directory. We can view the output by opening the file in any browser window. For example, we could type the following:

```
# mozilla /var/www/html/usage/index.html
```

The output report shows a 12-month summary of Web server activity. On the bar chart, for each month a green bar represents the number of hits on the Web site, the dark blue bar shows the number of different files hit, and the light blue bar shows the number of pages opened. It also shows data for the number of visits and the number of sites that visited in the right column. The amount of data transferred, in kilobytes, is displayed as well.

Figure below shows an example of a webalizer output file for a Web server that has been running for several months.

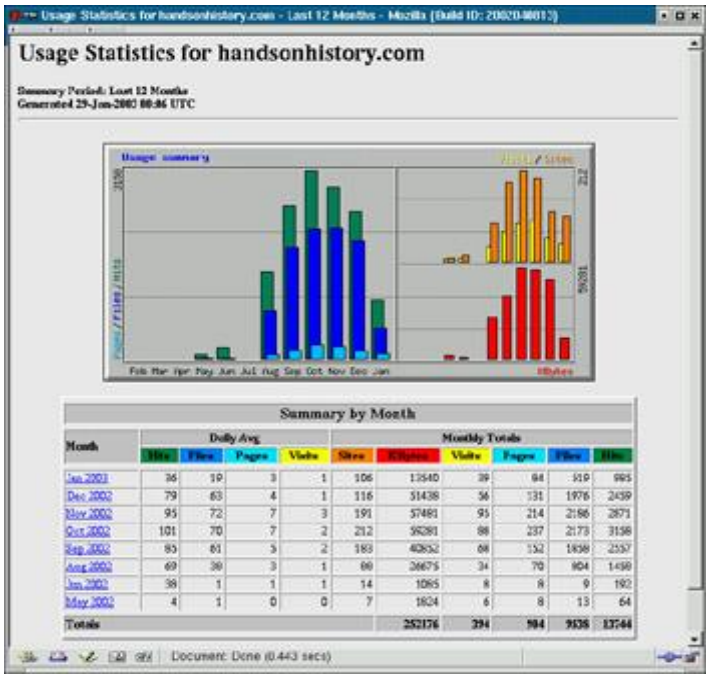

Figure: Webalizer displays Web data in chart and column formats.

Below the chart, a table shows daily and monthly summaries for activity during each month. Users can click the name of a month to see detailed activity.

Tip: *Because webalizer supports both common log format (CLF) and combined log format, it can be used to display information for log files other than those produced for Apache. For example, we could display statistics for our FTP server or Squid server.*

Several other software tools are available for analyzing transfer statistics. The accompanying sidebar on statistics packages available for Red Hat Linux describes some of these tools.

Statistics Packages Available For Red Hat Linux:

Analyzing the transfer log by hand isn't much fun. Several packages have been written to automate this task, two of which are described below:

Analog:

This free log-file analyzer is very fast and easily configurable, and it produces very detailed output (including bar graphs and hypertext links). More information can be found at www.analog.cx.

AWStats:

The Advanced Web Statistics tool (awstats.sourceforge.net) produces graphical statistics representing Web-server access. AWStats can work with log files in the Apache common log format. It can report statistics, such as the number of people who have visited, visits made per person, the domain and country of each visitor, and the number of visits made by robots.