

# **Your First Time: Apache Setup and Configuration**

Prepared and presented by: Ken Coar  
VP of Conference Planning,  
Apache Software Foundation and  
Senior Software Engineer,  
IBM Corporation

## Some Definitions

### Config File:

A text file containing *directives* that control how the Web server should operate. Config files live in the `conf` subdirectory under the *ServerRoot*.

### Container:

A paired set of *directives* that define a *scope* within which the enclosed directives have effect. Sometimes a container and the directives enclosed within it are called a *stanza*.

### Directive:

A one-line text command in a *config file* which instructs the server in some aspect of its operation. Directive names are not case-sensitive, but the arguments that follow them on the line usually are. Examples: `User`, `DirectoryIndex`.

### DocumentRoot:

The top of the directory tree where your Web content actually lives.

### Internet Media Type:

An attribute of a document that describes the format of the content. The IMT is split into a major and minor portion, separated by a slash. Some examples are `text/plain` for normal unformatted text or prose; `text/html` for text that contains HTML tags; `image/gif` denotes a binary image in GIF format; and `application/octet-stream` is frequently used to identify unknown or arbitrary binary content. IMTs are not case-sensitive; `text/plain` and `Text/Plain` are equivalent. This same major/minor syntax is used in other cases, such as when a browser indicates what types of content it can accept (e.g., `text/*`).

### MIME Type:

*MIME* is an acronym for Multipurpose Internet Mail Extensions, a set of standards defining how mail of various types and contents can be exchanged. Since the *MIME* system defined a very flexible way of assigning attributes to message bodies, it was adopted by the Web, so you'll frequently hear about the 'MIME type' of a Web document. The more general term is *Internet Media Type* (*q.v.*), though.

### Overrides:

The types of settings that can be changed by directives in *.htaccess* files. If the `FileInfo` keyword is in the list of allowed overrides, for example, that means that directives that affect file information in *.htaccess* files can *override* any settings declared at a broader *scope*.

### Scope:

A bounded portion of your server's URI space or filesystem. Scopes are nestable, and *directive* effects typically are inherited from higher-level scopes. The narrowest applicable scope ultimately defines the attributes of Web resources within it. See the section on **Scoping** for more information.

### ServerRoot:

The top of the directory tree where the server application itself, and all the configuration files, lives.

### Stanza:

See *container*.

### Virtual Hosting:

A means of making a single system appear to be multiple ones.

## Hardware and Operating System decisions

The Web does not live by software alone... it needs hardware! The Apache software will run on Windows NT V4.0 and almost variety of Unix, so your system hardware needs to be something that supports one or the other – or both. Intel systems running Linux or Windows NT V4 work just fine, but Apache will also run on IBM's AIX operating system, Hewlett-Packard's HP/UX, Compaq's Tru64 UNIX on Alpha, Solaris, SunOS, and OS/2, among others. So if you have any hardware available *at all*, you can probably get Apache to run on it.

Whether the server will run as well as you want is another question. If you're just experimenting with Apache in the privacy of your home office, your desktop system – whatever it is – will almost certainly suffice. If you're planning to set up a serious high-volume corporate machine, you should use iron that's sized appropriately. I've had excellent results with an AMD K6-2 350MHZ system under moderate load (hundreds to thousands of requests *per* minute), so use that as a starting point if you like.

Once you have your hardware lined up, you need to know (or choose) the operating system you'll be using. For the dedicated hobbyist, the free Linux and FreeBSD operating systems work very well; for the occasional dabbler who feels more comfortable with the Windows environment, Windows NT is a viable alternative.

**Note:** You should *NOT* try to run Apache on Windows 95 or Windows 98. While it may work, those platforms are designed to be 'end-user' environments rather than servers, and the recommendation is that you use the Windows system intended for that purpose: NT.

If you're installing Apache on an existing Unix or Windows NT system, your choice has already been made.

### A Note about Windows, Apache, and Slashes

I'm sure you've noticed that Web addresses – URLs – contain forward slashes ('/', called 'slashes') rather than backslashes ('\ ', sometimes called 'sloshes'). This has its roots in the origin of the first Web servers, which ran on Unix, where the slash in a filename meant much the same as a slosh in a Windows filename today. This also carries forward into the Apache configuration directives, partly because Apache was first produced for Unix and partly for the sake of consistency across all platforms. Apache on Windows will *usually* do the right thing if you accidentally use a slosh instead of a slash, but to be on the safe side you should **always** use forward slashes when giving Apache instructions. Save the sloshes for when you're dealing with Windows itself.

### Obtaining the Package

There are a couple of different ways in which you can obtain the Apache software:

1. Get it bundled with your system, such as with Red Hat Linux, or
2. Download it from the Internet.

#### Developer Site

If you really want to get into the details of rebuilding the Apache software from scratch, and modifying it to suit your needs, check out the Apache developer Web site:

`<http://dev.apache.org/>`

Bear in mind that the content of this site counts as documentation, and the information may lag behind current events (because the developers are all spending their time on the fun stuff – developing!)

If you want to go get it yourself rather than depending on a package distributor, the most important Web locations you need to know are the Apache download sites:

```
<http://www.apache.org/dyn/closer.cgi>  
and  
<http://www.apache.org/dist/>
```

The former will help you find the Apache mirror<sup>1</sup> site that's closest to you, and the latter is the master distribution site (currently located in California, USA).

## Binary or source

There are two parts to the Apache software: a ready-to-run binary application, and the source code from which it was built. An Apache package, such as the download distribution at the Apache Web site or the RPM package from Red Hat, may contain one or the other or both. The Windows package, for example, includes both but lets you choose which pieces to install.

While the Apache software runs on well over a hundred different platforms, pre-built packages are only available for a limited subset – specifically, whatever platforms are available to the Apache developers at the time the software was released. It's possible that there may not *be* a ready-to-run binary for your chosen platform, in which case you're going to need to build one from the source.

Building Apache is not very difficult nor very complicated.

**Note:** If you end up going with the Windows NT platform for your Apache Web server, the pre-built or roll-your-own decision has essentially been made for you. Unlike most Unix and Unix-like systems, Windows does *not* include bundled software development tools. The Apache developers chose Microsoft Visual C++ version 5 as the Windows development environment, so unless you have a copy of that available you're going to need to stick with the canned Apache package for Windows.

The Apache source is always available on the Internet, so even if you end up with a package that didn't include it you can still download it later.

## ***Building Apache***

Since the Windows development environment is not generally available and probably less than 1% of the people who run Apache on Windows want to build the software from scratch, I'm not going to cover it here.

Building the software on a Unix system is pretty simple. There are actually two ways to compile it: one that highly resembles (but is not identical to) the method used by GNU tools and other free software, such as `gcc` and `Perl`. This method is called "APACI" because that was the name given it by Ralf Engelschall, the fellow that put it together. The other method, sometimes called "Apache Classic," is the method that was used to build Apache before APACI was developed, and it still works. Apache Classic only builds the software, though, and doesn't give you any help in choosing your functionality nor actually installing Apache in the right directories when it's built, both of which *are* done by APACI. Since APACI is the more complete approach, I'm going to spend more time describing it. The Apache Classic method is more for the die-hard do-it-yourself hacker.

---

<sup>1</sup> 'Mirrors' are sites that contain identical copies of some master set of files. Mirrors are frequently used on the Web to make the information on popular sites available 'closer to home' for many visitors.

The assumption made for the commands described here is that you're installing Apache 1.3.6, have the package file (called a 'gzipped tarball') in the directory /usr/local/kits, and will build it in a subdirectory.

```
% cd /usr/local/kits
% gunzip -c < apache_1.3.6.tar.gz | tar xf -
% cd apache_1.3.6
% CC=gcc CFLAGS="-g -O2" ./configure \
    --enable-shared-max --enable-module=most \
    --with-layout=Apache
% make
```

## The Apache Directory Tree

There are a total of three directory trees associated with the Apache Web server package:

1. The kit (source) tree;
2. The tree used by the running server (the ServerRoot); and
3. The tree containing your Web documents (the DocumentRoot).

If you use the commands above, your source tree will be /usr/local/kits/ apache\_1.3.6, your ServerRoot will be /usr/local/apache, and your DocumentRoot will be /usr/local/apache/htdocs.

```
/usr/local/apache
bin/
libexec/
man/
conf/
icons/
htdocs/
cgi-bin/
include/
logs/
proxy/
```

**Figure 1 The ServerRoot tree**

```
apache_1.3.6/
  cgi-bin/
  cgi-src/
  conf/
  htdocs/
    manual/
      images/, info/, misc/, mod/, search/,
      vhosts/
  icons/
    small/
  logs/
  src/
    ap/
    helpers/
    include/
    lib/
    main/
    modules/
      example/, experimental/, extra/,
proxy/,
  standard/, test/
  os/
    bs2000/, emx/, os2/, tpf/, unix/, win32
  regex/
  support/
  test/
```

**Figure 2 The Source tree**

The source tree contains the files as they come from the distribution package, so if anything happens to them they're easily re-created – just fetch and unpack a new copy of the distribution.

The DocumentRoot tree is where your own custom content will go, and so is the most difficult to recover in case of an accident. Immediately following the software installation, the DocumentRoot contains the Apache manual, which is easily replaced – but as soon as you start adding your own documents and files, watch out, because you have to provide your own backups.

The ServerRoot is sort of a mix of these two extremes. It gets populated with files from the source tree, but they're modified to make the server behave as you desire. It also contains files that are constantly being updated: namely, the error and access logs. For safety's sake, you should treat the ServerRoot in the same way as the DocumentRoot: as containing files which you need to back up yourself.

Figure 1 displays the layout of the default ServerRoot tree that will be created if you use the commands above. Figure 1 is a bit abbreviated; the `icons` and `htdocs` directories, for instance, are actually the tops of trees containing the same files found in the corresponding locations in the source tree.

Figure 2 shows the directories in the source tree. This is the directory tree in which the software actually gets built, and which supplies the template files for the ServerRoot when you install the software after building it. You may notice that some of the directory names match names in the ServerRoot tree; this is because those directories typically provide the template files for the ones used by the running server.

If you use the default layout, your DocumentRoot tree will be the `htdocs` directory under your ServerRoot. When you start adding content of your own, you will probably want to move the supplied contents of the DocumentRoot (the Apache manual) to some other location.

## Installing the Apache Web Server

If you're using the Windows platform for your Apache Web server, installation is very simple. Just figure out where you want the server to go, double-click on the installer application (probably called `apache_1_3_6_win32.exe` or something similar), and away you go. When it's finished, ta-daaaah! You're done.

### Apache Windows Installation and Rebooting

When you install Apache on a Windows system, it **may or may not** claim that a reboot is necessary. If it does, it's only because the installation had a newer version of one of the Windows system files and needs to replace the older one. This should only happen on your first Apache installation on that system, if then.

**Note:** If you're upgrading an existing Apache installation on a Windows system, the safest approach is to **uninstall** the existing server before installing the new version. (See the next section.) This avoids any possible problems with the Windows registry.

Installation of Apache on a Unix system is rather more complicated. If you used the APACI method to build it, you selected the ServerRoot when you ran the `./configure` command. One more command will install all the software and configuration files in their proper locations under it:

```
% cd /usr/local/kits/apache_1.3.6
% make install
```

It's usually a good idea to be logged on as `root` when you do this, to make sure that all the files are given the right ownership and permissions.

When the command completes, the Apache software should be ready to run. It has been installed in the correct directories and all the necessary Apache files have been edited to reflect this.

**Note:** This process does *nothing* to actually start the server, nor to ensure that it gets started at system reboot or stopped at system shutdown. That is a very system-specific thing, so you're going to have to make those changes to your system yourself.

One of the files that is provided by the `make install` step is the `apachectl` script. This is the tool you use to start, stop, and otherwise manage your Apache Web server. The installation procedure updated it so that it correctly refers to the location in which your Apache software was installed. In the case of the default installation layout, the script will be found in the `/usr/local/apache/bin` directory;

you might want to add this directory to the default PATH for whatever account you'll be using to manage Apache.

When you modify your system's startup and shutdown procedures, you'll want to use the `apachectl` script to handle the running and stopping of the Apache server. Be careful to only invoke it at a point in the sequence at which the appropriate filesystem (e.g., `/usr/local/apache`) is actually mounted and available, or else the server won't be up when your system is.

## UNinstalling Apache

As with installation, removal of the Apache Web server software is very simple on Windows, and somewhat more involved on other platforms.

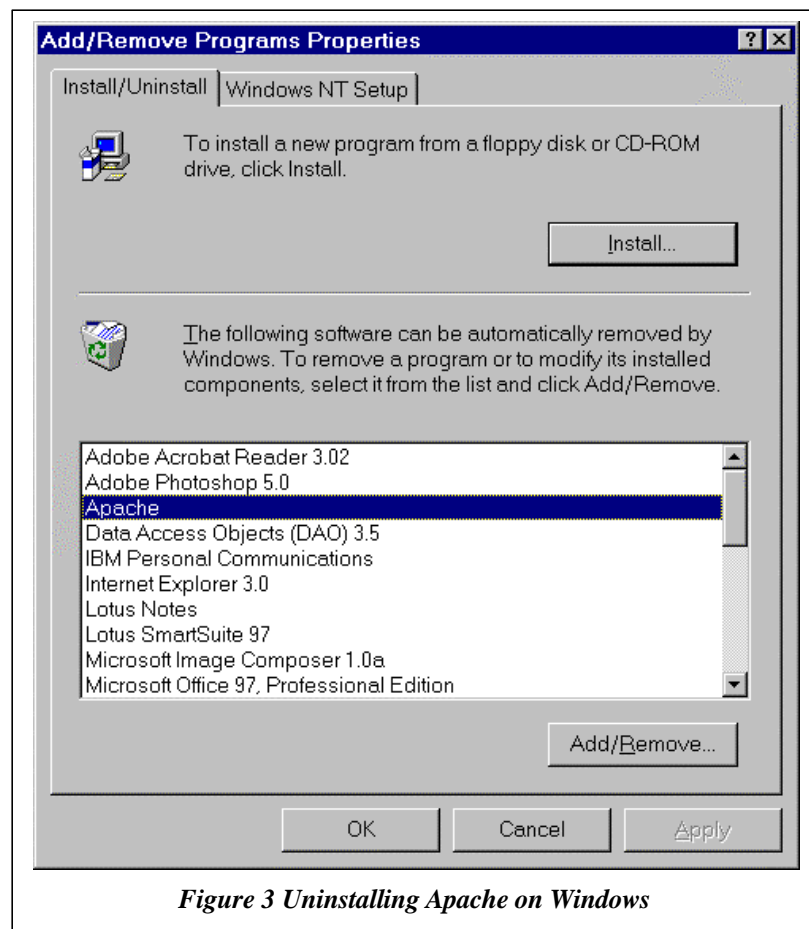
### Windows

On the Windows platform, the Apache software is registered like many other applications, and you can remove it using standard Windows tools.

To remove Apache from your Windows system, choose **Start => Settings => Control Panel** from your task bar, select the **Add/Remove Programs** control panel applet, and you should get a screen similar to that shown in Figure 2.

Click on the **Apache** line, then the **Add/Remove...** button in the lower right corner of the applet, and Windows should remove the Apache Web server software from your system.

You should *not* have to reboot your system for the change to take effect, and the removal should *not* delete any of your custom Web documents – just the files that were installed as part of the Apache package.



*Figure 3 Uninstalling Apache on Windows*

**Note:** As with any Windows installation, if you try to move the software after installing it, there is an excellent chance the software may stop working correctly and the uninstall procedure fail to delete it properly. It is much better to choose the correct installation location in the beginning, or, failing that, to un- and re-install the software to move it to the desired place.

## Unix and Unix-like Systems

Since Unix systems vary so widely, and each seems to have a unique ‘standard’ method of installing and uninstalling software, the removal of the Apache package is much more complex and usually requires a lot of manual manipulation. The things that you should do include:

1. Stop the Apache Web server software if it’s currently running;
2. Make a backup of your Web content (your DocumentRoot);
3. Remove Apache from your system’s startup/shutdown procedures;
4. Delete the ServerRoot tree; and
5. Delete the directory tree into which you unpacked the Apache distribution kit (in our example, `/usr/local/kits/apache_1.3.6`).

When you’ve successfully completed those 5 steps, you should have uninstalled the Apache software.

**Note:** By default, the APACI installation process puts the DocumentRoot *under* the ServerRoot, so deleting the latter will automatically delete the former as well – which is why you want to make a backup of it first.

### **The First Test**

Now that you have your server installed, it’s time to give it a first run-through to make sure it’s doing what it should. Start it up according to your platform:

**Unix:** Be sure you’re logged on as `root`, and then type  
`# /usr/local/apache/bin/apachectl start`

**Windows:** Log on as Administrator, and choose **Start => Programs => Apache Web Server => Apache Server** from your Taskbar. A DOS window should appear and will probably display a couple of messages (which you can ignore), and then should stay up and running with the window title bar set to “Apache Server”.

Your server should now be running, at least for the moment. Try using a browser on your system to access the default DocumentRoot page, `<http://127.0.0.1/>`. Figure 4 shows what you *should* see.

**Note:** It is *really, really* important that you change this default file as quickly as you can after installing the Apache server. This is especially important if you are migrating an existing Web site to use the Apache software. If you don’t, some of the people who visit your site are going to think that Apache has stolen your Web site, or possibly that Apache has been installed on *their* desktop system. Even though the page clearly describes what is going on, the number of reports that get sent to the Apache developers about “why have you stolen my favourite Web site?” is astonishing. Some people just won’t bother to read the text past the headline.

Now, it’s possible that things didn’t work as expected. In this case, one or more of the following probably occurred:

- Your browser complained about “there was no response”;
- The DOS window appeared, but disappeared again before you could read any of the text on the screen;
- Your `apachectl` command complained about the server not being started; or

- Your `apachectl` command said the server was started, but your browser couldn't access it.

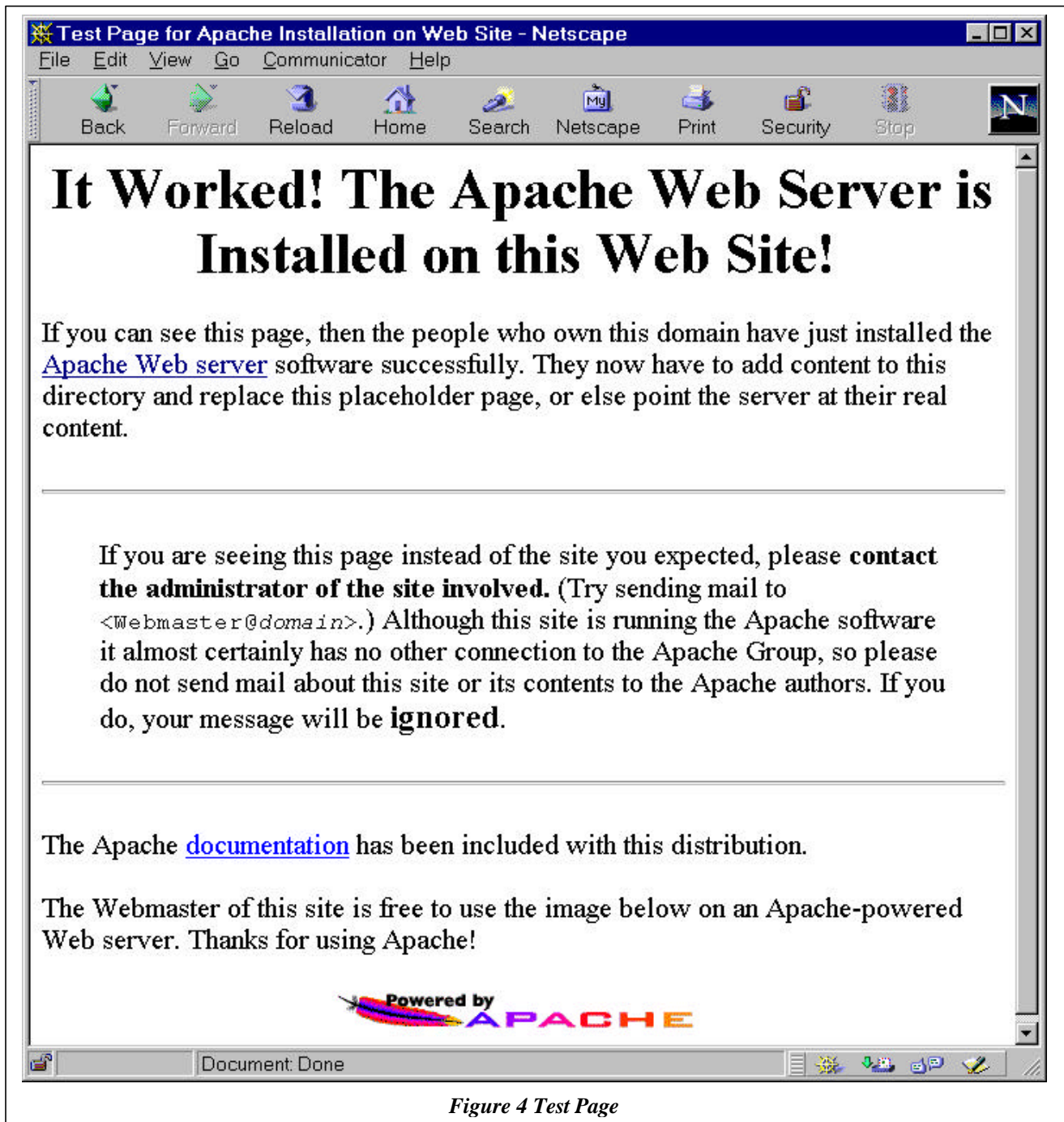


Figure 4 Test Page

In each case, the problem was probably caused by some sort of syntax error in the config files. This *shouldn't* happen for a fresh installation, though. See the section on **Common setup problems and solutions** for instructions on how to determine the cause of the problem.

If you saw the test page correctly, shut down your Apache server again as follows:

**Unix:** Be sure you're logged on as root, and then type  
# /usr/local/apache/bin/apachectl stop

**Windows:** Type CTRL/C in the DOS window to stop the server and close the window.

Your server should now be shut down, and all attempts to access it with a Web browser should give the "there was no response" message.

## **Initial Configuration**

Your server should be able to run as soon as you install it, and by default it's configured with a standard set of options that suffice for most single-use Web sites. If you're going to use to run an Internet Service Provider (ISP) with multiple virtual hosts, or a high-traffic site like Slashdot, though, these defaults aren't going to be sufficient.

You should take a little time to browse through the `httpd.conf` config file found in the `conf` subdirectory under the `ServerRoot` (`/usr/local/apache/conf` in the default Unix installation, and "`C:\Program Files\Apache Group\Apache\conf`" in the default Windows installation). It's just a text file, so any editor should suffice. **Be sure not to save any changes you might accidentally make!** Figure 5 shows a sample excerpt:

```
# First, we configure the "default" to be a very restrictive set of
# permissions.
#
<Directory />
    Options FollowSymLinks
    AllowOverride None
</Directory>
```

**Figure 5 Sample Directive**

Lines beginning with "#" are comments, and are ignored.

There are four directives shown in Figure 5:

1. **<Directory** – Declare a scope to which the following directives apply
2. **Options** – Specify what sorts of operations are allowed within this scope
3. **AllowOverride** – Define the categories of directives that will be processed if found in `.htaccess` files within the scope
4. **</Directory>** -- Mark the end of the scope declaration

The "**<Directory**" and "**</Directory**" directives form what's called a *container*, and the directives between them only apply to the directory listed in the "**<Directory**" directive, and any subdirectories of it. (Yes, the `/>` isn't part of the directive; it's the *argument* to the "**<Directory**" directive, telling it upon which directory to operate.) This selective applicability is called *scoping*, and is described in the next section.

## Scoping

One concept that's very important to Apache configuration is the idea of directive *scope*. The scope of a directive is the portion of the server's URI space or filesystem to which it applies.

Scopes are nestable, with each subsequent level refining any settings made at upper (or 'outer') levels. The effects within a particular scope are typically a result of merging all of the settings for all of the outer scopes that apply, with the more specific ones taking precedence. (This merging is called 'inheritance.')

There are two types of scopes: those that apply to directories on the server's filesystems, and those that apply to the URIs your server processes. They don't always overlap (it's possible to have a URI that doesn't refer to a file, for instance), so be sure to keep the distinction in mind. The container directives help; the ones that refer to files and directories are called "<Files>" and "<Directory>", and the one that refers to Web locations is called "<Location>".

There are two ways in which you can apply a scope: either in the server config files, or in *per*-directory directive files (called ".htaccess" [pronounced "dot aitch tee access"] files). .htaccess files by their very nature can't include any "<Directory>" or "<Location>" containers. In fact, whether a particular directive *can* be used in a .htaccess file is determined by the definition of the directive itself, and the setting of the allowed overrides. Some directives are never allowed in .htaccess files, but the rest are always allowed – they may just be ignored if their category isn't in the allowed override list defined by the inherited AllowOverride directive settings.

The Apache server processes scopes from the general to the specific. That means it 'walks' down the directory tree toward the document's directory, processing any <Directory> stanzas that apply to each subdirectory in turn. At each directory, the server looks for and processes any .htaccess file found in the directory.

After looking through the directories, the server examines any <Files> containers it found and processes the directives in any that apply.

As a next-to-the-last step, Apache looks for any <Location> stanzas that apply to the document, and processes any contained directives in those that do.

Finally, the server knows exactly what directives and attributes apply to the document, and it can deal with it as appropriate.

Probably the best way to explain scoping is to demonstrate it. Consider the environment shown in Figure 6, and interpreting the following requests (assuming the server's name is "localhost") according to those settings:

1. `http://localhost/`

Since there's nothing following the slash after the host name, the server will treat it as a request for something in the DocumentRoot, C:\Temp\Apache. Ordinarily, if there was an `index.htm` file there the server would display it because of the `DirectoryIndex` directive. If there wasn't, and the `Indexes` option was enabled, the server would display a list of files in the C:\Temp\Apache directory. Unfortunately, the server won't do any of this because it finds the "Deny from All" statement at an outer scope, and nothing any closer to the C:\Temp\Apache directory has changed it. So this request will get a "**Forbidden**" error.

2. `http://localhost/secret.rpm`

This request would usually result in either a "**Resource Not Found**" or "**Forbidden**" error message, depending upon whether the file C:\Temp\Apache\secret.rpm actually existed or not. But the same

“Deny from All” directive that made the previous request fail with “**Forbidden**” before even getting that far will have the same effect on this one.

3. `http://localhost/foo.rpm`  
If the file `C:\Temp\Apache\foo.rpm` actually existed, and access weren't denied to `C:\Temp\Apache` by the misconfiguration that caused the previous two requests to fail, the server would send the file to the client and tell it the document was a RealAudio file.
4. `http://localhost/Linux/foo.rpm`  
Now we get some scoping inheritance going. The “Allow from All” directive finally allows browsers to access things in this directory, and the “AddType” directive overrides the one at the outer scope and indicates that the file `C:\Temp\Apache\Linux\foo.rpm` – if it exists – should be labeled as being a binary file. The “Deny from .linux-haters.org” line will modify the inherited access and prevent only clients from that domain from accessing any files in the directory.
5. `http://localhost/Linux/secret.rpm`  
The “<Files>” stanza in container number 6 overrides the one defined by container number 1 in the outermost scope, so this request will succeed (if the file exists) and will be marked as being a binary file.

```
DocumentRoot "C:/Temp/Apache"
DirectoryIndex index.htm
1. <Files "secret.rpm">
    Order Allow,Deny
    Deny from All
</Files>
2. <Directory "C:/">
    AllowOverride None
    Options None
    Order Allow,Deny
    Deny from All
</Directory>
3. <Directory "C:/Temp/Apache">
    AddType audio/x-pn-realaudio-plugin .rpm
</Directory>
4. <Directory "C:/Temp/Apache/Linux">
    Options Indexes
    AddType application/octet-stream .rpm
    Order Deny,Allow
    Allow from All
</Directory>
5. <Location "/Linux">
    Order Deny,Allow
    Deny from .linux-haters.org
6. <Files "secret.rpm">
    Order Deny,Allow
    Allow from All
</Files>
</Location>
```

*Figure 6 Example Scoping*

## Logging

*Logging* refers to the records that the Apache server keeps about what it does and what happens to it. There are two main types of logs:

- **Access logs**, in which the server tracks information about the requests it receives from clients, and
- **Error logs**, which it uses to let you know about things it had problems doing.

**Note:** When in doubt, look at the error log!

Depending upon your configuration, you can several of each of these types of logs, but the default configuration only contains one of each. Both reside in the `logs` subdirectory under the `ServerRoot`.

Since the error log contains reports from the server about how something may have gone wrong or otherwise interfered with it processing something, you have no control over the format. The server uses whatever words it has been programmed to use to get the message across. Figure 7 shows a sample excerpt from an Apache server error log.

```
[Wed Jun  9 00:47:56 1999] [info] [client 24.94.179.175] (32)Broken pipe: client
stopped connection before send mmap completed
[Wed Jun  9 05:03:31 1999] [info] [client 205.230.159.47] (32)Broken pipe: client
stopped connection before rwrite completed
[Wed Jun  9 05:28:25 1999] [error] [client 205.230.159.22] File does not exist:
/home/coar/public_html/cgi/[13]
```

*Figure 7 Excerpt from an error log*

Not all of the error log messages will look like this, of course; each tries to include sufficient information to understand what happened, if not to correct it.

While you can't control the format of the messages in the error log, you *can* control the severity of messages that get recorded. The `LogLevel` directive sets the lowest severity of message that will be saved in error logs in the current scope. It takes a single keyword as an argument, which may be one of the following (listed in order of increasing severity):

- **Debug** – Causes the server to report in the most detail, including stuff which is probably only useful to people working on the source code. This can result in a rapidly-growing error log.
- **Info** – This will result in the server recording non-problem messages about its operation. Experiment with it to see if there are any you really care about; you'll probably find that there aren't.
- **Notice** – This includes messages about the server's starting up, restarting, and shutting down, among others. This level is useful if you want to keep track of when your server is restarted.
- **Warning** – This is the level at which legitimate problems start getting recorded. Config file protection problems and the like fall into this category.
- **Error** – The messages that appear at this level usually deal with problems responding to requests, such as missing or protected document files, or CGI scripts that fail to function properly.
- **Critical** – The server regards basic problems that affect (or can affect) multiple requests as being in this category. Problems with permissions on `.htaccess` files, an inability to access the network, and so on are considered critical.

- **Alert** – This severity is usually reserved for server startup problems, such as with the server user identity, or things which cause the running server to die.
- **Emergency** – Emergency conditions are those that prevent the server from starting at all, or dying horribly once it was operational.

If you set your LogLevel to 'Critical,' you won't get any error messages of Error, Warning, or lower severity in your log file. This means the log file probably won't get huge – but it also means that you're likely to be surprised by behaviour for which you won't have an explanation. I recommend a setting of Info to start, so you can become accustomed to the sorts of things the server reports, and then moving to Warning when you're comfortable with your setup.

Alert and Emergency level messages will definitely require your intervention before the server can function again.

## ***Controlling the Running Server***

A brief introduction to server management was given in the **The First Test** section, but it was a little bald and not suited for use in a production environment. There are actually four things you can do to manage your Apache Web server:

- **Start** – this should be fairly obvious. If the server is already running, an attempt to start it has no effect.
- **Shutdown** – likewise self-explanatory. Trying to stop the server when it isn't running won't have any effect. If the server is in the middle of processing any requests, they will be aborted immediately.
- **Graceful Restart** – this instructs the server to reload its configuration files. Any existing requests in process are allowed to complete first.
- **Graceless Restart** – forces an immediate reload of the server config files, aborting any current requests immediately. New requests will be handled again as soon as the reload is complete.

**Note:** Graceless restart is currently **not available** on the Windows platform. To perform a graceless restart, stop the server and start it again.

The ways in which you issue these instructions to the server depends upon the platform you're using.

## **Windows**

If you install the Apache Web server as an NT service, you can stop and start it from the Services control panel. You can also stop and start it from a DOS window using the **apache** command:

```
C:\>"C:\Program Files\Apache Group\Apache \apache" command
```

Where *command* is one of the following:

- **-k shutdown** – This will shut down the running Apache server, whether it was started as a service or from a DOS window.
- **-k restart** – This forces the running Apache server to perform a graceful restart.

- **-S -d “C:/Program Files/Apache Group/Apache”** – Tests the server config files for syntax errors and display any virtualhost information. **Warning: You may need to press CTRL/C after issuing this command.**

## Unix

On a Unix platform, you use the `apachectl` script (described earlier) to manage your server. The format of the command is:

```
# /usr/local/apache/bin/apachectl command
```

where *command* is one of the following:

- **start** – If the server isn’t running, this should start it. This process automatically does a **configtest** (see below) first. If the server is running, the script will say so and do nothing else.
- **stop** – As might be expected, this will shut down the running server. The shutdown is graceless, meaning that any requests currently being processed will be aborted rather than allowed to finish cleanly.
- **graceful** – This instructs the running server to perform a graceful restart, finishing any current requests and then reloading the config files. It’s a good idea to run a **configtest** (see below) first, because if there’s a problem in the config files the server will not successfully reload them and will die instead.
- **restart** – This causes a graceless restart. As with the **graceful** option, you should precede this with a **configtest** run to make sure the server *is* restartable.
- **configtest** – This will check the server config files for syntax errors, and display any that are found. This is a good step to take any time you change the server config files.

**Note:** You need to be logged on as root in order to manage the Apache Web server.

### **Setup essentials: MIME types and more**

In order for a Web client (such as a browser) to be able to deal intelligently – or even semi-intelligently – with documents it receives from Web servers, it needs to know the format of the document’s contents. For example, it needs to be able to recognise an image in order to display it graphically, or to be able to tell that a binary application is, in fact, binary, so it won’t try to display it on the screen.

This information is one of the document attributes that the server provides as part of its response to the client’s request. It’s called the document’s “content type,” or occasionally the “MIME type<sup>2</sup>” or “media type” of the document.

How does the server know what content-type to send? By consulting its configuration files, which means that ultimately *you*, as Web master, make the determination.

---

<sup>2</sup> This is actually a misnomer; it *should* be called the document’s *media type*. See the brief glossary at the beginning of this document for details.

The most common way of associating a particular Internet media type (IMT) with documents is by relating it to a particular filename pattern – typically an extension. That is, you might say “All files that have ‘.html’ as part of their names have media type ‘text/html’.”

There are two basic ways these associations are declared to the Apache software. One is through a file containing a standard list of IMT-to-extension mappings, and the other is through the `AddType` Apache directive. Associations are processed in that order, as well, which means that this dual mechanism allows you to override the conventional settings with `AddType` directives in your config or `.htaccess` files.

Both the `AddType` directive and the default IMT associations file (“`mime.types`” in the “`conf`” subdirectory under the `ServerRoot`) allow a one-to-many syntax. That is, you can associate a single IMT with multiple extensions with one line. For example,

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

marks all files with one of those extensions as being an HTML document.

The `AddType` directive is controlled by the `FileInfo` override, which means that you can use it on a *per-directory* basis in `.htaccess` files as long as they are within the scope of an “`AllowOverride FileInfo`” directive.

## Handlers

In order to get sent to the requesting client, every document must be processed by a *content handler*. The content handler is responsible for doing whatever is necessary to turn the resource on the server into the results to be sent to the client.

By default, the Apache server deals with files on disk. If a client requests a document that doesn’t translate to a file, the server will report an error. If the request *does* refer to a file on disk, the document will be processed by what’s called the *default handler*. The default handler simply copies the file from the disk to the client over the network, with no special additional processing.

Things get a bit more complicated when you want to be able to handle CGI scripts, server-side includes (SSIs), or other more-than-just-a-file resources. To deal with documents like this, you need to let the Apache server know which handlers to use.

Enter the `AddHandler` directive. It looks a lot like the `AddType` directive, but it associates a handler with a set of extensions, rather than a content-type.

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

The portion of the Apache server software (if any) that knows how to do this sort of handling will be used to process the document before it is sent to the client.

If an `AddHandler` directive names a handler that the server doesn’t recognise, an error message will be written to the error log and the default handler will be used to process the file instead.

Examples of both `AddHandler` and `AddType` directives can be found in the `httpd.conf` file in the `conf` subdirectory under the `ServerRoot`.

## ***Common setup problems and solutions***

The Apache Web server is a very complex piece of software, and there are almost innumerable ways in which things can go wrong. At least 95% of the problems reported have to do with configuration problems rather than actual bugs in the software itself, so you can see there's a definite likelihood that any problem you have is caused by some subtle configuration nuance rather than a real bug.

Since setup and configuration problems are so common, here are the top questions (and answers!) about them from the Apache FAQ document. They're all written first-person from the post of view of the Apache Group.

Q: Why do I get "setgid: Invalid argument" at startup?

A: Your `Group` directive (probably in `conf/httpd.conf`) needs to name a group that actually exists in the `/etc/group` file (or your system's equivalent). This problem is also frequently seen when a negative number is used in the `Group` directive (e.g., "Group #-1"). Using a group name – not group number – found in your system's group database should solve this problem in all cases.

Q: Why am I getting "httpd: could not set socket option TCP\_NODELAY" in my error log?

A: This message almost always indicates that the client disconnected before Apache reached the point of calling `setsockopt()` for the connection. It shouldn't occur for more than about 1% of the requests your server handles, and it's advisory-only in any case.

Q: Why am I getting "connection reset by peer" in my error log?

A: This is a normal message and nothing about which to be alarmed. It simply means that the client canceled the connection before it had been completely set up - such as by the end-user pressing the "Stop" button. People's patience being what it is, sites with response-time problems or slow network links may experience this more than high-capacity ones or those with large pipes to the network.

Q: The errorlog says Apache dumped core, but where's the dump file?

A: In Apache version 1.2, the error log message about dumped core includes the directory where the dump file should be located. However, many Unixes do not allow a process that has called `setuid()` to dump core for security reasons; the typical Apache setup has the server started as `root` to bind to port 80, after which it changes UIDs to a non-privileged user to serve requests.

Dealing with this is extremely operating system-specific, and may require rebuilding your system kernel. Consult your operating system documentation or vendor for more information about whether your system does this and how to bypass it. If there is a documented way of bypassing it, it is recommended that you bypass it only for the `httpd` server process if possible.

The canonical location for Apache's core-dump files is the `ServerRoot` directory. As of Apache version 1.3, the location can be set *via* the `CoreDumpDirectory` directive to a different directory. Make sure that this directory is writable by the user the server runs as (as opposed to the user the server is started as).

Q: When I run it under Linux I get "shmget: function not found", what should I do?

A: Your kernel has been built without SysV IPC support. You will have to rebuild the kernel with that support enabled (it's under the "General Setup" submenu). Documentation for kernel building is

beyond the scope of this FAQ; you should consult the Kernel HOWTO, or the documentation provided with your distribution, or a Linux newsgroup/ mailing list. As a last-resort workaround, you can comment out the `#define USE_SHMGET_SCOREBOARD` definition in the LINUX section of `src/conf.h` and rebuild the server (prior to 1.3b4, simply removing `#define HAVE_SHMGET` would have sufficed). This will produce a server which is slower and less reliable.

Q: Server hangs, or fails to start, and/or error log fills with "fcntl: F\_SETLKW: No record locks available" or similar messages.

A: These are symptoms of a file locking problem, which usually means that the server is trying to use a synchronization file on an NFS filesystem. Because of its parallel-operation model, the Apache Web server needs to provide some form of synchronization when accessing certain resources. One of these synchronization methods involves taking out locks on a file, which means that the filesystem whereon the lockfile resides must support locking. In many cases this means it can't be kept on an NFS-mounted filesystem.

To cause the Web server to work around the NFS locking limitations, include a line such as the following in your server configuration files:

```
LockFile /var/run/apache-lock
```

The directory should not be generally writable (*e.g.*, don't use `/var/tmp`). See the `LockFile` documentation for more information.

Q: Why am I getting "Expected </Directory> but saw </Directory>" when I try to start Apache?

A: This is a known problem with certain versions of the AIX C compiler. IBM are working on a solution, and the issue is being tracked by problem report #2312.

Q: I'm using RedHat Linux and I have problems with `httpd` dying randomly or not restarting properly.

A: RedHat Linux versions 4.x (and possibly earlier) RPMs contain various nasty scripts which do not stop or restart Apache properly. These can affect you even if you're not running the RedHat supplied RPMs.

If you're using the default install then you're probably running Apache 1.1.3, which is outdated. From RedHat's ftp site you can pick up a more recent RPM for Apache 1.2.x. This will solve one of the problems.

If you're using a custom built Apache rather than the RedHat RPMs then you should `rpm -e apache`. In particular you want the mildly broken `/etc/logrotate.d/apache` script to be removed, and you want the broken `/etc/rc.d/init.d/httpd` (or `httpd.init`) script to be removed. The latter is actually fixed by the `apache-1.2.5` RPMs but if you're building your own Apache then you probably don't want the RedHat files.

We can't stress enough how important it is for folks, especially vendors, to follow the stopping Apache directions given in our documentation. In RedHat's defense, the broken scripts were necessary with Apache 1.1.x because the Linux support in 1.1.x was very poor, and there were various race conditions on all platforms. None of this should be necessary with Apache 1.2 and later.

## Check the error log

This is *absolutely* the **first** thing you should do. If you have a problem with your Apache server, and try to get assistance from an expert without looking in your error log first, the probability is extremely high that you will be flamed to a cinder where you stand. Most Apache problems result in an error message, and most of the error messages make it obvious what's wrong (file permissions, for instance). After consulting your error log and determining which message(s) relate to your problem, and finding them confusing, ambiguous, or inscrutable, you can go on to the next stage – which is to:

## Check the FAQ

The FAQ (Frequently Asked Questions list) for the Apache server is located on the Web at

`<http://www.apache.org/docs/misc/FAQ.html>`

It's probably not as up-to-date as it should be, but that's a factor of it being documentation and software engineers preferring to write code rather than documentation. Even so, a lot of conventional wisdom has grudgingly been added to the FAQ, so that's the second place you should look in your search for problem resolution.

If the FAQ doesn't hold the answer, the next step is to:

## Check the Apache bugdb

Problems with the Apache Web server software are stored in an online bug database (called the 'bugdb'), which you can find on the Web at:

`<http://bugs.apache.org/>`

That URL will take you to a page that will let you search the database for keywords; try typing in one or two from your error log message and seeing what comes up. If nothing does, then it's time to:

## Check/ask USENET

The main USENET newsgroups for issues with the Apache software are

`comp.infosystems.www.servers.*`

The most traffic is usually found in the `.unix` sub-group, but Windows-specific discussions happen nearby in the related groups. If you can't find any current discussions that seem to relate to your problem, you can search the DejaNews archives at `<http://deja.com/>`. If you still come up empty, then it's time to:

## Submit a bug report

This definitely should be your last resort. If you don't exercise due diligence in searching the other mechanisms already described, and your problem turns out to have already been reported, it's Crispy-Critter time for you. If you're *sure* you've found a bug in the software, go to the same URL listed above for the bug database and click on the '**Submit a new problem report**' button. Be as complete as you can be in your descriptions, and be patient – it may be months before your problem is solved by the volunteers.