CHAPTER 13

Miscellaneous Topics

With its hundreds of configuration directives, and dozens upon dozens of modules providing additional functionality, the Apache Web server can be terrifically complex. So, too, can the questions about how to use it. We have collected many of the most common questions we have seen and categorized them, putting related topics into their own chapters when there were enough of them.

However, some of the things that come up don't fall readily into one of the categories we have chosen, or perhaps are more fundamental and we've collected them into this catch-all chapter of "things that don't belong anywhere else."

13.1 Placing Directives Properly

Problem

You know what directive you need but aren't sure where to put it.

Solution

If you wish the scope of the directive to be global (i.e., you want it to affect all requests to the Web server), then it should be put in the main body of the configuration file or it should be put in the section starting with the line <*Directory* /> and ending with </*Directory*>.

If you wish the directive to affect only a particular directory, it should be put in a <*Directory*> section that specifies that directory. Be aware that directives specified in this manner also affect subdirectories of the stated directory.

Likewise, if you wish the directive to affect a particular virtual host or a particular set of URLs, then the directive should be put in a <*VirtualHost*> section, <*Location*> section, or perhaps a <*Files*> section, referring to the particular scope in which you want the directive to apply.

In short, the answer to "Where should I put it?" is "Where do you want it to be in effect?"

Discussion

This question is perhaps the most frequently asked question in every Apache help venue. It is usually answered in a way that is relevant to the specific situation but not in a general all-purpose kind of way.

The situation is further complicated by the fact that the configuration file is frequently split over several files, which are loaded *via Include* directives, and the (usually) mistaken impression that it will make a difference whether a directive is put in one file or another.

Knowing exactly where to put a particular directive comes from understanding how Apache deals with sections (such as <*Directory*> and <*Location*>). There is seldom one magic place that a directive must be placed to make it work. However, there are usually a number of places where you can put a directive and have it produce an undesired effect.

There are two main situations in which a directive, when added to your configuration file, will not have the desired effect. These are when a directive is overridden by a directive appearing in the same scope but later in the configuration, and when there is a directive in a more specific scope.

For the first of these two situations, it is important to understand that the Apache configuration file is parsed from top to bottom. Files that are *Include*'ed are considered to appear in their entirety in the location where the *Include* directive appears. Thus, if you have the same directive appearing twice but with different values, the last one appearing will be the one that is actually in effect.

In the other situation, it's important to understand that, while directives in one directory apply to subdirectories, a <*Directory*> section referring to a more specific or "deeper" directory will have precedence over sections referring to "shallower" directories. For example, consider the following configuration:

```
<Directory /www/docs>
    Options ExecCGI
</Directory>

<Directory /www/docs/mod>
    Options Includes
</Directory>
```

Files accessed from the directory */www/docs/mod/misc/* will have *Options Includes* in effect but will not have *Options ExecCGI* in effect, because the more specific directory section is the configuration that applies.

Finally, you must consider *.htaccess* files as well, which can override settings in the main server configuration file, and cause situations that are confusing and difficult to be tracked down.

See Also

For .htaccess files:

> *http://httpd.apache.org/docs/howto/htaccess.html*
> *http://httpd.apache.org/docs/2.2/howto/htaccess.html*

For directories:

> *http://httpd.apache.org/docs/mod/core.html#directory*
> *http://httpd.apache.org/docs/mod/core.html#directorymatch*
> *http://httpd.apache.org/docs/2.2/mod/core.html#directory*
> *http://httpd.apache.org/docs/2.2/mod/core.html#directorymatch*

For location:

> *http://httpd.apache.org/docs/mod/core.html#location*
> *http://httpd.apache.org/docs/mod/core.html#locationmatch*
> *http://httpd.apache.org/docs/2.2/mod/core.html#location*
> *http://httpd.apache.org/docs/2.2/mod/core.html#locationmatch*

For Apache:

> *http://httpd.apache.org/docs/mod/core.html#files*
> *http://httpd.apache.org/docs/mod/core.html#filesmatch*
> *http://httpd.apache.org/docs/2.2/mod/core.html#files*
> *http://httpd.apache.org/docs/2.2/mod/core.html#filesmatch*

13.2 Renaming .htaccess Files

Problem

You want to change the default name of *per*-directory configuration files to something else, such as on a Windows system, because filenames beginning with a dot can cause problems.

Solution

Use the *AccessFileName* directive to specify the new name:

```
AccessFileName ht.access
```

Discussion

In addition to the server-wide configuration files, you can add directives to special files in individual directories. These are called *.htaccess* (*aitch tee access*) files because that's the default name for them.

However, the Unixish convention of filenames that begin with a dot doesn't play well on all platforms; on Windows in particular it can be difficult to edit files with such names.

Apache allows you to change the name it will use when looking for these *per*-directory files with the *AccessFileName* directive (which can only appear in the server-wide configuration files). You can use any name that's valid on your platform.

If you use the *AccessFileName* directive, be sure to make any additional appropriate changes to your configuration such as the *<FilesMatch "^\.ht">* container that keeps the files from being fetchable over the Web:

```
<FilesMatch "^ht\.">
    Order deny,allow
    Deny from all
</FilesMatch>
```

See Also

- Recipe 11.7
- *http://httpd.apache.org/docs/howto/htaccess.html*
- *http://httpd.apache.org/docs/2.2/howto/htaccess.html*

13.3 Generating Directory/Folder Listings

Problem

You want to see a directory listing when a directory is requested.

Solution

Turn on *Options Indexes* for the directory in question:

```
<Directory /www/htdocs/images>
    Options +Indexes
</Directory>
```

Discussion

When a URL maps to a directory or folder in the filesystem, Apache will respond to the request in one of three ways:

- If *mod_dir* is part of the server configuration, and the mapped directory is within the scope of a *DirectoryIndex* directive, and the server can find one of the files identified in that directive, then the file will be used to generate the response.
- If *mod_autoindex* is part of the server configuration and the mapped directory is within the scope of an *Options* directive that has enabled the *Indexes* keyword, then the server will construct a directory listing at runtime and supply it as the response.

monospace

- The server will return a 404 (Resource Not Found) status.

Enabling directory listings

The real keys to enabling the server's ability to automatically generate a listing of files in a directory are the inclusion of *mod_autoindex* in the configuration and the **Indexes** keyword to the *Options* directive. This can be done either as an absolute form, as in:

```
Options FollowSymLinks Indexes
```

or in a selective or relative form such as:

```
Options -ExecCGI +Indexes
```

Enabling directory listings should be done with caution. Because of the scope inheritance mechanism, directories farther down the tree also will be affected; and because the server will apply the sequence of rules listed at the beginning of this section in an effort to provide some sort of response, a single missing file can result in the inadvertent exposure of your filesystem's contents.

Disabling directory indexing below an enabled directory

There are essentially two ways to work around this issue and ensure that the indexing applies only to the single directory:

- Add an *Options* **–Indexes** to *.htaccess* files in each subdirectory.
- Add an *Options* **–Indexes** to a *<Directory>* container that matches all the subdirectories.

For example, to permit directory indexes for directory */usr/local/htdocs/archives* but not any subdirectories, use:

```
<Directory /usr/local/htdocs/archives>
    Options +Indexes
</Directory>

<Directory /usr/local/htdocs/archives/*>
    Options -Indexes
</Directory>
```

See Also

- *http://httpd.apache.org/docs/mod/core.html#options*
- *http://httpd.apache.org/docs/mod/mod_dir.html*
- *http://httpd.apache.org/docs/mod/mod_autoindex.html*

13.4 Solving the "Trailing Slash" Problem

Problem

Loading a particular URL works with a trailing slash but does not work without it.

Solution

Make sure that *ServerName* is set correctly and that none of the *Alias* directives have a trailing slash.

Discussion

The "trailing slash" problem can be caused by one of two configuration problems: an incorrect or missing value of *ServerName*, or an *Alias* with a trailing slash that doesn't work without it.

Incorrect ServerName

An incorrect or missing *ServerName* seems to be the most prevalent cause of the problem, and it works something like this: when you request a URL such as *http://example.com/something*, where *something* is the name of a directory, Apache actually sends a redirect to the client telling it to add the trailing slash.

The way that it does this is to construct the URL using the value of *ServerName* and the requested URL. If *ServerName* is not set correctly, then the resultant URL, which is sent to the client, will generate an error on the client end when it can't find the resulting URL.

If, by contrast, *ServerName* is not set at all, Apache will attempt to guess a reasonable value when you start it up. This will often lead it to guess incorrectly, using values such as 127.0.0.1 or localhost, which will not work for remote clients. Either way, the client will end up getting a URL that it cannot retrieve.

Invalid Alias directive

In the second incarnation of this problem, a slightly malformed *Alias* directive may cause a URL with a missing trailing slash to be an invalid URL entirely.

Consider, for example, the following directive:

```
Alias /example/ /home/www/example/
```

The *Alias* directive is very literal, and aliases URLs starting with */example/*, but it does not alias URLs starting with */example*. Thus, the URL *http://example.com/example/* will display the default document from the directory */home/www/example/*, while the URL *http://example.com/example* will generate a "file not found" error message, with an error log entry that will look something like:

```
File does not exist: /usr/local/apache/htdocs/example
```

The solution to this is to create *Alias* directives without the trailing slash, so that they will work whether or not the trailing slash is used:

```
Alias /example /home/www/example
```

See Also

- *http://httpd.apache.org/docs/misc/FAQ-E.html#set-servername*

13.5 Setting the Content-Type According to Browser Capability

Problem

You want to set `Content-Type` headers differently for different browsers, which may render the content incorrectly otherwise.

Solution

Check the `Accept` headers with *RewriteCond* and then set the `Content-Type` header with a `T` flag:

```
RewriteCond "%{HTTP_ACCEPT}" "application/xhtml\+xml"
RewriteCond "%{HTTP_ACCEPT}" "!application/xhtml\+xml\s*;\s*q=0+(?:\.0*[^0-9])"
RewriteRule . - [T=application/xhtml+xml;charset=iso-8859-1]
```

Discussion

Different browsers tend to deal with content differently and sometimes need a nudge in the right direction. In this example, for browsers that specify (using the `HTTP_ACCEPT` header) that they prefer XHTML content, we want to send a `Content-Type` header specifying that the content we are sending fulfills that requirement.

The `T` (Type) flag sets the `Content-Type` for the response.

See Also

- *http://httpd.apache.org/docs/mod/mod_rewrite.html*

13.6 Handling Missing Host: Header Fields

Problem

You want to treat differently all requests that are made without a `Host:` request header field.

Solution

```
SetEnvIf Host "^$" no_host=1
Order Allow,Deny
Allow from all
Deny from env=no_host
RewriteCond "%{HTTP_HOST}" "^$"
RewriteRule ".*"           -     [F,L]
```

Discussion

The Host: request header field is essential to correct handling of name-based virtual hosts (see Recipe 4.1). If the client doesn't include it, the chances are very good that the request will be directed to the wrong virtual host. All modern browsers automatically include this field, so only custom-written or very old clients are likely to encounter this issue.

monospace

The solutions given will cause such requests to be rejected with a 403 Forbidden status; the exact text of the error page can be tailored with an *ErrorDocument 403* directive.

The first solution is slightly more efficient.

See Also

- Recipe 4.1

13.7 Alternate Default Document

Problem

You want to have some file other than *index.html* appear by default.

Solution

Use *DirectoryIndex* to specify the new name:

```
DirectoryIndex default.htm
```

Discussion

When a directory is requested—that is, a URL ending in a / rather than in a file name —*mod_dir* will select the index document from that directory and serve that file in response. By default, the index file is assumed to be *index.html*, but this can be configured to something else with the *DirectoryIndex* directive.

Note also that *DirectoryIndex* can be set to several files, which are listed in order of precedence:

```
DirectoryIndex index.html index.htm index.php default.htm
```

Finally, note that you also can provide a relative URL if you want to load content from some other directory, such as a CGI program:

```
DirectoryIndex /cgi-bin/index.pl
```

See Also

- *http://httpd.apache.org/docs/mod/mod_dir.html*

13.8 Setting Up a Default "Favicon"

Problem

You want to define a default favorite icon, or "favicon," for your site, but allow individual sites or users to override it.

Solution

Put your default *favicon.ico* file into the */icons/* subdirectory under your *ServerRoot*, and add the following lines to your server configuration file in the scope where you want it to take effect (such as inside a particular <*VirtualHost*> container or outside all of them):

```
AddType image/x-icon .ico
<Files favicon.ico>
    ErrorDocument 404 /icons/favicon.ico
</Files>
```

Discussion

favicon.ico files allow Web sites to provide a small (16 x 16 pixels) image to clients for use in labeling pages; for instance, the Mozilla browser will show the favicon in the location bar and in any page tabs. These files are typically located in the site's *DocumentRoot* or in the same directory as the pages that reference them.

What the lines in the solution do is trap any references to *favicon.ico* files that don't exist and supply a default instead. An *ErrorDocument* is used instead of a *RewriteRule*, because we want the default to be supplied *only* if the file isn't found where expected. A rewrite, unless carefully crafted, would force the specified file to be used regardless of whether a more appropriate one existed.

See Also

- Chapter 5

Directory Listings in *ScriptAlias*ed Directories

13.9 ~~Setting Up a Default "Favicon"~~

Problem

You want to allow directory indexing in a directory named in a *ScriptAlias* directive.

This is considered a bad idea, because it can reveal to strangers the names of specific scripts that may be subvertible.

Solution

Add the following lines to the <*Directory*> container that defines the characteristics of your *ScriptAlias*ed directory:

```
<Files ".">
    Options Indexes FollowSymLinks
    SetHandler httpd/unix-directory
</Files>
```

Discussion

The *ScriptAlias* directive imposes a lot of restrictions on directories to which it is applied, primarily for reasons of security. After all, such directories contain scripts of arbitrary code that will be executed on your system; if you should happen to be using a well-known and popular script in which a vulnerability is subsequently detected, anyone on the Web may be able to take advantage of it.

One of the restrictions imposed explicitly by design is disallowing directory listings in *ScriptAlias*ed parts of the filesystem. This amounts to what's called "security through obscurity"—namely, hiding an issue and hoping that no one discovers it even though it's easily accessible—but it's better than advertising what scripts your server can execute.

However, under some circumstances you may want to allow directory listings in such directories—or at least the use of pseudolistings provided by files named in a *DirectoryIndex* directive. To do this you need to override the special protections. In particular, you need to indicate that they *don't* apply to the directory itself (indicated by the <*Files* "."> container), and that it should be treated as a directory (the *AddHandler* directive) and not a script.

This is actually playing rather fast and loose with Apache's internal mechanisms, and taking advantage of an unintentional feature. As a consequence, it may not continue to work in future versions of the software.

See Also

- Chapter 12

13.10 Enabling .htaccess Files

Problem

You want to enable the use of *.htaccess* files in directories on your server.

Solution

Add the following line to your *httpd.conf* file in a scope that applies to the directory (or directories) for which you want to enable *.htaccess* files:

```
AllowOverride keyword ...
```

Discussion

As long as the `keyword` you specify isn't *None*, this will instruct the server to process *.htaccess* files in the relevant scope. Which keyword or keywords you use depends on what sort of things you want the *.htaccess* file to be able to affect.

Some platforms, such as Microsoft Windows, object to filenames that begin with a dot. To placate this particular idiosyncrasy, you can use the *AccessFileName* directive to tell the server to use a different name for these files, such as:

```
AccessFileName ht.access
```

If Apache seems to be ignoring an *.htaccess* file, you can verify this by putting some normal text into the file and browsing to a document in the same directory:

```
This is not an Apache directive
```

If the server is reading the *.htaccess* file, it will complain about the text (because it isn't an Apache directive); you'll get both a message in the server's error log and an "Internal Server Error" page in your browser. If neither of these happens, then the file *is* being ignored, and you should check your scoping and *AllowOverride* directives.

See Also

- *http://httpd.apache.org/docs/mod/core.html#allowoverride*

13.11 Converting IBM/Lotus Server-Side Includes to Apache

Problem

You are migrating documents from a Web server running IBM's Web Traffic Express or Lotus Domino Go to a server running Apache.

Solution

Most of the WTE/LDGW server-side include directives are directly portable to Apache's format, but there are a few that either have no parallel or need to be modified to work properly. Here is a list of the exceptions:

- *config cmntmsg*—There is no Apache equivalent for this setting.
- *echo* directive variables `SSI_DIR`, `SSI_FILE`, `SSI_INCLUDE`, `SSI_PARENT`, and `SSI_ROOT` —There are no built-in Apache equivalents for these automatic variables.
- *global*—There is no direct equivalent for this SSI directive. Variables set in the current file may be referenced later in the file, but they are not available to included documents.
- *set* directive—Apache's version of this directive is roughly the same as WTE's/LDGW's, except that it *does not* understand the `&varname;` type of SSI variable substitution.

Discussion

Apache's *mod_include* module implements the canonical list of SSI directives, but WTE and LDGW have added extensions to the "standard" list. Because SSI is now largely considered a legacy technology for providing dynamic content, the usual recommendation is to use servlets, template engines, or scripting languages to provide equivalent functionality. It is extremely unlikely that Apache's implementation will be extended, so if you're taking advantage of some of the WTE/LDGW extensions, your time would probably be better spent migrating to a newer technology instead of trying to duplicate the original effect with Apache's SSI implementation.

See Also

- *http://httpd.apache.org/docs/mod/mod_include.html*