Apache HTTP Server Tutorial: .htaccess files

Available Languages: en | es | fr | ja | ko | pt-br

`.htaccess` files provide a way to make configuration changes on a per-directory basis.

.htaccess files

Related Modules	Related Directives
core	AccessFileName
mod_authn_file	AllowOverride
mod_authz_groupfile	Options
mod_cgi	AddHandler
mod_include	SetHandler
mod_mime	AuthType
	AuthName
	AuthUserFile
	AuthGroupFile
	Require

You should avoid using `.htaccess` files completely if you have access to httpd main server config file. Using `.htaccess` files slows down your Apache http server. Any directive that you can include in a `.htaccess` file is better set in a `Directory` block, as it will have the same effect with better performance.

What they are/How to use them

`.htaccess` files (or "distributed configuration files") provide a way to make configuration changes on a per-directory basis. A file, containing one or more configuration directives, is placed in a particular document directory, and the directives apply to that directory, and all subdirectories thereof.

Note:

If you want to call your `.htaccess` file something else, you can change the name of the file using the `AccessFileName` directive. For example, if you would rather call the file `.config` then you can put the following in your server configuration file:

```
AccessFileName ".config"
```

In general, `.htaccess` files use the same syntax as the main configuration files. What you can put in these files is determined by the `AllowOverride` directive. This directive specifies, in categories, what directives will be honored if they are found in a `.htaccess` file. If a directive is permitted in a `.htaccess` file, the documentation for that directive will contain an Override section, specifying what value must be in `AllowOverride` in order for that directive to be permitted.

For example, if you look at the documentation for the `AddDefaultCharset` directive, you will find that it is permitted in `.htaccess` files. (See the Context line in the directive summary.) The Override line reads `FileInfo`. Thus, you must have at least `AllowOverride FileInfo` in order for this directive to be honored in `.htaccess` files.

Example:

Context:	server config, virtual host, directory, .htaccess

```
Override: FileInfo
```

If you are unsure whether a particular directive is permitted in a `.htaccess` file, look at the documentation for that directive, and check the Context line for ".htaccess".

When (not) to use .htaccess files

In general, you should only use `.htaccess` files when you don't have access to the main server configuration file. There is, for example, a common misconception that user authentication should always be done in `.htaccess` files, and, in more recent years, another misconception that `mod_rewrite` directives must go in `.htaccess` files. This is simply not the case. You can put user authentication configurations in the main server configuration, and this is, in fact, the preferred way to do things. Likewise, `mod_rewrite` directives work better, in many respects, in the main server configuration.

`.htaccess` files should be used in a case where the content providers need to make configuration changes to the server on a per-directory basis, but do not have root access on the server system. In the event that the server administrator is not willing to make frequent configuration changes, it might be desirable to permit individual users to make these changes in `.htaccess` files for themselves. This is particularly true, for example, in cases where ISPs are hosting multiple user sites on a single machine, and want their users to be able to alter their configuration.

However, in general, use of `.htaccess` files should be avoided when possible. Any configuration that you would consider putting in a `.htaccess` file, can just as effectively be made in a `<Directory>` section in your main server configuration file.

There are two main reasons to avoid the use of `.htaccess` files.

The first of these is performance. When `AllowOverride` is set to allow the use of `.htaccess` files, httpd will look in every directory for `.htaccess` files. Thus, permitting `.htaccess` files causes a performance hit, whether or not you actually even use them! Also, the `.htaccess` file is loaded every time a document is requested.

Further note that httpd must look for `.htaccess` files in all higher-level directories, in order to have a full complement of directives that it must apply. (See section on how directives are applied.) Thus, if a file is requested out of a directory `/www/htdocs/example`, httpd must look for the following files:

```
/.htaccess
/www/.htaccess
/www/htdocs/.htaccess
/www/htdocs/example/.htaccess
```

And so, for each file access out of that directory, there are 4 additional file-system accesses, even if none of those files are present. (Note that this would only be the case if `.htaccess` files were enabled for `/`, which is not usually the case.)

In the case of `RewriteRule` directives, in `.htaccess` context these regular expressions must be re-compiled with every request to the directory, whereas in main server configuration context they are compiled once and cached. Additionally, the rules themselves are more complicated, as one must work around the restrictions that come with per-directory context and `mod_rewrite`. Consult the Rewrite Guide for more detail on this subject.

The second consideration is one of security. You are permitting users to modify server configuration, which may result in changes over which you have no control. Carefully consider whether you want to give your users this privilege. Note also that giving users less privileges than they need will lead to additional technical support requests. Make sure you clearly tell your users what level of privileges you have given them. Specifying exactly what you have set `AllowOverride` to, and pointing them to the relevant documentation, will save yourself a lot of confusion later.

Note that it is completely equivalent to put a `.htaccess` file in a directory `/www/htdocs/example` containing a directive, and to put that same directive in a Directory section `<Directory "/www/htdocs/example">` in your main server configuration:

`.htaccess` file in `/www/htdocs/example`:

Contents of .htaccess file in `/www/htdocs/example`

```
AddType text/example ".exm"
```

Section from your `httpd.conf` file

```
<Directory "/www/htdocs/example">
    AddType text/example ".exm"
</Directory>
```

However, putting this configuration in your server configuration file will result in less of a performance hit, as the configuration is loaded once when httpd starts, rather than every time a file is requested.

The use of `.htaccess` files can be disabled completely by setting the `AllowOverride` directive to `none`:

```
AllowOverride None
```

How directives are applied

The configuration directives found in a `.htaccess` file are applied to the directory in which the `.htaccess` file is found, and to all subdirectories thereof. However, it is important to also remember that there may have been `.htaccess` files in directories higher up. Directives are applied in the order that they are found. Therefore, a `.htaccess` file in a particular directory may override directives found in `.htaccess` files found higher up in the directory tree. And those, in turn, may have overridden directives found yet higher up, or in the main server configuration file itself.

Example:

In the directory `/www/htdocs/example1` we have a `.htaccess` file containing the following:

```
Options +ExecCGI
```

(Note: you must have "`AllowOverride Options`" in effect to permit the use of the "`Options`" directive in `.htaccess` files.)

In the directory `/www/htdocs/example1/example2` we have a `.htaccess` file containing:

```
Options Includes
```

Because of this second `.htaccess` file, in the directory `/www/htdocs/example1/example2`, CGI execution is not permitted, as only

Options Includes is in effect, which completely overrides any earlier setting that may have been in place.

Merging of .htaccess with the main configuration files

As discussed in the documentation on Configuration Sections, .htaccess files can override the <Directory> sections for the corresponding directory, but will be overridden by other types of configuration sections from the main configuration files. This fact can be used to enforce certain configurations, even in the presence of a liberal AllowOverride setting. For example, to prevent script execution while allowing anything else to be set in .htaccess you can use:

```
<Directory "/www/htdocs">
    AllowOverride All
</Directory>

<Location "/">
    Options +IncludesNoExec -ExecCGI
</Location>
```

This example assumes that your DocumentRoot is /www/htdocs.

Authentication example

If you jumped directly to this part of the document to find out how to do authentication, it is important to note one thing. There is a common misconception that you are required to use .htaccess files in order to implement password authentication. This is not the case. Putting authentication directives in a <Directory> section, in your main server configuration file, is the preferred way to implement this, and .htaccess files should be used only if you don't have access to the main server configuration file. See above for a discussion of when you should and should not use .htaccess files.

Having said that, if you still think you need to use a .htaccess file, you may find that a configuration such as what follows may work for you.

.htaccess file contents:

```
AuthType Basic
AuthName "Password Required"
AuthUserFile "/www/passwords/password.file"
AuthGroupFile "/www/passwords/group.file"
Require group admins
```

Note that AllowOverride AuthConfig must be in effect for these directives to have any effect.

Please see the authentication tutorial for a more complete discussion of authentication and authorization.

Server Side Includes example

Another common use of .htaccess files is to enable Server Side Includes for a particular directory. This may be done with the following configuration directives, placed in a .htaccess file in the desired directory:

```
Options +Includes
AddType text/html "shtml"
AddHandler server-parsed shtml
```

Note that AllowOverride Options and AllowOverride FileInfo must both be in effect for these directives to have any effect.

Please see the SSI tutorial for a more complete discussion of server-side includes.

Rewrite Rules in .htaccess files

When using RewriteRule in .htaccess files, be aware that the per-directory context changes things a bit. In particular, rules are taken to be relative to the current directory, rather than being the original requested URI. Consider the following examples:

```
# In httpd.conf
RewriteRule "^/images/(.+)\.jpg" "/images/$1.png"

# In .htaccess in root dir
RewriteRule "^images/(.+)\.jpg" "images/$1.png"

# In .htaccess in images/
RewriteRule "^(.+)\.jpg" "$1.png"
```

In a .htaccess in your document directory, the leading slash is removed from the value supplied to RewriteRule, and in the images subdirectory, /images/ is removed from it. Thus, your regular expression needs to omit that portion as well.

Consult the mod_rewrite documentation for further details on using mod_rewrite.

CGI example

Finally, you may wish to use a .htaccess file to permit the execution of CGI programs in a particular directory. This may be implemented with the following configuration:

```
Options +ExecCGI
AddHandler cgi-script "cgi" "pl"
```

Alternately, if you wish to have all files in the given directory be considered to be CGI programs, this may be done with the following configuration:

```
Options +ExecCGI
SetHandler cgi-script
```

Note that AllowOverride Options and AllowOverride FileInfo must both be in effect for these directives to have any effect.

Please see the CGI tutorial for a more complete discussion of CGI programming and configuration.

Troubleshooting

When you put configuration directives in a .htaccess file, and you don't get the desired effect, there are a number of things that may be going wrong.

Most commonly, the problem is that AllowOverride is not set such that your configuration directives are being honored. Make sure that you don't have a AllowOverride None in effect for the file scope in question. A good test for this is to put garbage in your .htaccess file and reload the page. If a server error is not generated, then you almost certainly have AllowOverride None in effect.

If, on the other hand, you are getting server errors when trying to access documents, check your httpd error log. It will likely tell you that the directive used in your .htaccess file is not permitted.

```
[Fri Sep 17 18:43:16 2010] [alert] [client
192.168.200.51] /var/www/html/.htaccess: DirectoryIndex
not allowed here
```

This will indicate either that you've used a directive that is never permitted in `.htaccess` files, or that you simply don't have `AllowOverride` set to a level sufficient for the directive you've used. Consult the documentation for that particular directive to determine which is the case.

Alternately, it may tell you that you had a syntax error in your usage of the directive itself.

```
[Sat Aug 09 16:22:34 2008] [alert] [client
192.168.200.51] /var/www/html/.htaccess: RewriteCond: bad
flag delimiters
```

In this case, the error message should be specific to the particular syntax error that you have committed.