FREE eBook

LEARNING

Hypertext Access file

Free unaffiliated eBook created from
Stack Overflow contributors.

#.htaccess

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: hypertext-access-file

It is an unofficial and free Hypertext Access file ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Hypertext Access file.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with Hypertext Access file

Remarks

An `.htaccess` file controls how Apache interacts with your site. When an `.htaccess` file is placed in your domain's directory (usually root directory), the file is detected and executed by Apache.

An `.htaccess` file is commonly used for the following:

- Denying specific IPs to your site
- Password protecting your site
- Rewriting URLs
- Custom error pages
- Compressing and Caching Files
- General Security and Hack Prevention

Versions

Various Apache releases

Version	Current version	Release
1.3	1.3.42	1998-06-06
2.0	2.0.65	2002-04-06
2.2	2.2.31	2005-12-01
2.4	2.4.23	2012-02-21

Examples

Setting up .htaccess

> `.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.

An .htaccess file controls how Apache interacts with your site. It is used to alter the requests and modify default behavior without needing to alter the core server configuration files.

Setting up `.htaccess` is as simple as opening a notepad and saving it as `.htaccess`. Generally, this

file will placed on the `root` directory of your website files, but you can use it under multiple different directories. This is especially useful if you're looking to password protect specific directories.

Enabling .htaccess

> Sometimes even a single error in your `httpd.conf` or `.htaccess` file will result in a temporary meltdown of the server, and users will see *500 - Internal Server Error* page. So, make sure to always make a backup of your `httpd.conf` and `.htaccess` files before you make a change.

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

`.htaccess` files are normally enabled by default. This is controlled by `AllowOverride` directive in the `httpd.conf` file. This directive can only be placed inside of a `<Directory>` section.

Beside `All` there are numerous other values that limit configuration of only certain contexts. Some of them are:

- **None** - Completely disable `.htaccess`.
- **AuthConfig** - Authorization directives such as those dealing with Basic Authentication.
- **FileInfo** - Directives that deal with setting Headers, Error Documents, Cookies, URL Rewriting, and more.
- **Indexes** - Default directory listing customizations.
- **Limit** - Control access to pages in a number of different ways.
- **Options** - Similar access to Indexes but includes even more values such as ExecCGI, FollowSymLinks, Includes and more.

```
# Only allow .htaccess files to override Authorization and Indexes
AllowOverride AuthConfig Indexes
```

Custom Error Pages

`.htaccess` can be used to set a custom error pages that matches the theme of your website instead of seeing a white error page with black techno-babble when users end up on at a page with an error server response code. The error page can be any browser parseable file, including (But not limited to) .html, .php, .asp, .txt, .xml.

Examples for almost all common error response codes:

```
#Client Errors

ErrorDocument 400 /mycool400page.html    # Bad Request
ErrorDocument 401 /mycool401page.html    # Unauthorized
ErrorDocument 402 /mycool402page.html    # Payment Required
ErrorDocument 403 /mycool403page.html    # Forbidden
ErrorDocument 404 /mycool404page.html    # Page Not Found
```

```
#Server Errors

ErrorDocument 500 /mycool500page.html    # Internal Server Error
ErrorDocument 501 /mycool501page.html    # Not Implemented
ErrorDocument 502 /mycool502page.html    # Bad Gateway
ErrorDocument 503 /mycool503page.html    # Service Unavailable
ErrorDocument 504 /mycool504page.html    # Gateway Timeout
ErrorDocument 505 /mycool505page.html    # Internal Server Error
```

It is always good practice to include Error Documents for the most common error responses, 400, 403, 404, and 500, as these errors are able to occur on all browsers.

the 500 error is one of the most notorious errors as it occurs if anything fails while loading the page to send, most commonly server html preprocessing failures from things like PHP, ASP, and other html preprocessors. It is good practice while testing to set the 500 page to display the error that occurred, rather then an unspecific 500 error page.

To enable the 500 error page to write a specific error see one of the following based on what html preprocessor you are using: php asp

Setting Server Timezone

There are many time zones around the world, it is important to make sure your server is set to the right one. This is done in `.htaccess` by using:

```
SetEnv TZ America/Indianapolis
```

A few example of possible other time zones:

```
America/Los_Angeles
America/Los_Angeles - Pacific Time
Pacific/Honolulu - Hawaii
```

Just make sure you use `SetEnv` in front of your selected time zone.

Read Getting started with Hypertext Access file online: https://riptutorial.com/dot-htaccess/topic/1023/getting-started-with-hypertext-access-file

Chapter 2: Denying Access

Examples

Denying IPs

```
order allow,deny
deny from 255.0.0.0
allow from all
```

This denies access to the IP `255.0.0.0`.

```
order allow,deny
deny from 123.45.6.
allow from all
```

This denies access to all IPs in the range `123.45.6.0` to `123.45.6.255`.

Hot Link Prevention

```
RewriteEngine on
RewriteCond %{HTTP_REFERER} !^$
RewriteCond %{HTTP_REFERER} !^http://(www\.)?yourdomain.com/.*$ [NC]
RewriteRule \.(gif|jpg|css)$ - [F]
```

This blocks all the links to '.gif', '.jpg' and '.css' files which are not from the domain name `http://www.yourdomain.com`.

Display alternate content:

```
RewriteEngine on
RewriteCond %{HTTP_REFERER} !^$
RewriteCond %{HTTP_REFERER} !^http://(www\.)?yourdomain.com/.*$ [NC]
RewriteRule \.(gif|jpg)$ http://www.yourdomain.com/angryman.jpg [R,L]
```

This blocks all links to '.gif' and '.jpg' files which are not from the domain name 'http://www.yourdomain.com/' and displays the file 'http://www.yourdomain.com/angryman.jpg' instead.

Denying access from IPs to files/directories

```
# Deny access to a directory from the IP 255.0.0.0
<Directory /path/to/directory>
    order allow,deny
    deny from 255.0.0.0
    allow from all
</Directory>
```

```
# Deny access to a file from the IP 255.0.0.0
<FilesMatch "^\.ht">
    order allow,deny
    deny from 255.0.0.0
    allow from all
</FilesMatch>
```

Read Denying Access online: https://riptutorial.com/dot-htaccess/topic/4741/denying-access

Chapter 3: General Security and Hack Prevention

Remarks

.htaccess redirection is a common vector for malicious hackers to exploit and infect websites. We have seen what .htaccess files are, how they are used by malicious hackers, and how to protect your website.

Examples

Hack Prevention

Prevent access to your `.htaccess` file

```
<Files .htaccess>
order allow,deny
deny from all
</Files>

# Rename the file
AccessFileName thehtfile.ess
```

Prevent URL attacks

```
# Enable rewrites
RewriteEngine On

# Block <script> tags from executing in the URL
RewriteCond %{QUERY_STRING} (<|%3C).*script.*(>|%3E) [NC,OR]

# Block scripts from setting a PHP Globals variable
RewriteCond %{QUERY_STRING} GLOBALS(=|[|\%[0-9A-Z]{0,2}) [OR]

# Block scripts from using base64_encode
RewriteCond %{QUERY_STRING} base64_encode.*(.*) [OR]

# Block scripts from using the a_REQUEST variable
RewriteCond %{QUERY_STRING} _REQUEST(=|[|\%[0-9A-Z]{0,2})
```

Disable use of scripts on your directories..

```
AddHandler cgi-script .php .pl .py .jsp .asp .htm .shtml .sh .cgi
Options -ExecCGI
```

Disable directory index

Enabled directory index means that if someone access to any folder which don't contains index.php , index.html, index.htm or any other default file defined in DirectoryIndex in apache configuration then all files in that folder will be listed in browser if you try to visit that page.

Often directory index is enabled by default on your apache server, in these cases good security practice is to disable directory index with following line:

```
Options -Indexes
```

Read General Security and Hack Prevention online: https://riptutorial.com/dot-htaccess/topic/2531/general-security-and-hack-prevention

Chapter 4: Handling File Types

Examples

Enable PHP to be parsed in HTML

If you want to include PHP code in your HTML file and you don't want to rename the file type from `.html` or `.htm` to `.php`, the below allows your HTML file to parse your PHP code correctly.

```
AddHandler application/x-httpd-php .html .htm
```

Read Handling File Types online: https://riptutorial.com/dot-htaccess/topic/1690/handling-file-types

Chapter 5: Rewriting and Redirecting

Remarks

Before URLs can be rewritten, a module called `mod_rewrite.c` needs to be enabled. Usually, it is disabled in the configuration by default.

`mod_rewrite` can be enabled by executing the command

```
$ sudo a2enmod mod_rewrite
$ sudo service apache2 restart
```

or by commenting out the lines

```
#LoadModule rewrite_module modules/mod_rewrite.so
#AddModule mod_rewrite.c
```

in `httpd.conf` file.

Examples

Popular Rewrite Flags

F|forbidden

Similar to `Deny`, this flag forces the server to immediately return a *403 Forbidden* status code to the requesting browser or client for the request.

Example: Deny access to requests that end with `exe`:

```
RewriteRule .exe$ - [F]
```

G|gone

If a requested resource was available in the past, but is no longer available, you can use this flag to force the server to immediately return a *410 Gone* status code to the requesting browser or client for the request.

Example: Tell a visitor that an old product no longer exists:

```
RewriteRule ^old-product.html$ - [G]
```

L|last

In most contexts, other than `.htaccess`, this flag instructs `mod_rewrite` to stop processing the current condition/rule set, much the same way `last` and `break` (Perl and C, respectively) do.

However, in the `.htaccess` or `<Directory>` context, a request that has been rewritten using a `RewriteRule` with this flag will be passed back to the URL parsing engine for further processing. As such, it is possible, for the rewritten URI to be handled by the same context, and perhaps altered further.

A general recommendation is to use the `END` flag to not only stop processing the current condition/rule set, but also to prevent any further rewriting in these contexts.

Note: The `F` and `G` flags, discussed above, both use `L` implicitly, so you do not need to specify them separately.

N|next

This flag will re-run the rewriting process from the beginning, starting again with the first condition/rule set. This time, the URL to match is no longer the original URI, but rather the rewritten URI returned by the last rule set. Use this flag to restart the rewriting process.

A word of warning: Use this flag with caution, as it may result in an infinite-loop!

NC|nocase

This instructs `mod_rewrite` to match the `Pattern` of a `RewriteRule` without being case-sensitive. To clarify, `MyIndex.html` and `myindex.html` would be regarded by the module as the same thing. Further, this flag allows you to use `a-z` instead of `A-Za-z` in a regular expression.

R|redirect

This flag is used to send an HTTP redirect response to the requesting browser/client.

By default, if no code is given, a redirect response with the *302 Found* (similar to a temporary redirect) status code will be returned. If you wish to use a more permanent redirect, then you should use the `302` (*301 Moved Permanently*) status code.

Generally, only status codes in the range 300-399 should be used with this flag. If status codes outside of this range are used (which is perfectly acceptable), then the substitution string is discarded and rewriting is stopped as if the `L` flag were used. In some cases, this is a handy way to force *404 Not Found* responses, even if the request points to an existing resource.

Example: Issue a *302 Found* redirect response:

```
RewriteRule ^bus$ /train [R,L]
```

Example: Issue a *301 Moved Permanently* redirect response:

```
RewriteRule ^speed-train$ /hyperloop [R=301,L]
```

Example: Force a *404 Not Found*:

```
RewriteRule ^blip$ - [R=404,L]
```

www and non-www redirects

Redirect any naked domain to `www.[your_domain].tld`:

```
# Start Apache Rewriting engine
RewriteEngine On
# Make sure you're not already using www subdomain
# and that the host string is not empty
RewriteCond %{HTTP_HOST} !^$
RewriteCond %{HTTP_HOST} !^www\.
# We check for http/https connection protocol
RewriteCond %{HTTPS}s ^on(s)|
# In case the previous conditions matches, redirect to www
RewriteRule ^(.*)$ http%1://www.%{HTTP_HOST}/$1 [R=301,L]
```

Redirect `www.[your_domain].tld` to `[your_domain].tld`

```
# Start Apache Rewriting engine
RewriteEngine On
# We check if we're on the www subdomain
RewriteCond %{HTTP_HOST} ^www\.([^\.]+\.[^\.]+)$
# In case the previous condition matches, redirect to non-www
RewriteRule ^(.*)$ http://%1/$1 [R=301,L]
```

Redirect any level of nested subdomains to your main domain:

```
# Start Apache Rewriting engine
RewriteEngine On
# We check if there's a subdomain
RewriteCond %{HTTP_HOST} \.([^.]+\.[^.]+)$
# redirect to the main domain name
RewriteRule ^ http://%1%{REQUEST_URI} [R=301,L]
```

SEO Friendly URLs

Search engines won't index your products if you have a URL like the following:

```
http://www.yourdomain.com/product.php?id=123
```

SEO friendly URL would look like `http://www.yourdomain.com/123/product-name/`. The following code helps achieve this without having to change `product.php` code.

```
RewriteEngine On
RewriteRule ^product/([0-9]+)/product-name-slug/?$ product.php?id=$1
```

Adding a trailing slash at the end

```
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_URI} !(.*)/$
RewriteRule ^(.*)$ /$1/ [L,R=301]
```

The first `RewriteCond` helps exclude the files. The second `RewriteCond` checks if there is already a trailing slash. If the case is so `RewriteRule` is not applied.

If you have any URL that shouldn't be rewritten, you can add one more `RewriteCond`.

```
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_URI} !(.*)/$
RewriteCond %{REQUEST_URI} !url/to/not/rewrite
RewriteRule ^(.*)$ /$1/ [L,R=301]
```

http and https redirects and HSTS configuration

Generic redirect *to https*:

```
# Enable Rewrite engine
RewriteEngine on

# Check if URL does not contain https
RewriteCond %{HTTPS} off [NC]
# If condition is true, redirect to https
RewriteRule (.*) https://%{SERVER_NAME}/$1 [R=301,L]
```

Generic redirect *to http*:

```
# Enable Rewrite engine
RewriteEngine on

# Check if URL does contain https
RewriteCond %{HTTPS} on [NC]
# If condition is true, redirect to http
RewriteRule (.*) http://%{SERVER_NAME}/$1 [R=301,L]
```

Forcing HTTPS connection (HSTS):

```
<IfModule mod_headers.c>
    Header always set Strict-Transport-Security "max-age=31536000; includeSubDomains"
```

```
</IfModule>
```

where, the `includeSubDomains` option can be removed, if HSTS should be applied only to the base domain, or the domain with the above configuration.

Redirect with/without query params

Redirect without query params:

```
RewriteRule ^route$ /new_route_without_query [L,R=301,QSD]
```

Redirect with query params:

```
RewriteCond %{QUERY_STRING} ^$
RewriteRule ^/?route$ %{REQUEST_URI}?query=param1&query2=param2 [NC,L,R=301]
```

Read Rewriting and Redirecting online: https://riptutorial.com/dot-htaccess/topic/1550/rewriting-and-redirecting

Chapter 6: Speed Optimization

Examples

Enable Compression (Apache 2.0+)

> Enabling gzip compression can reduce the size of the transferred response by up to 90%, which can significantly reduce the amount of time to download the resource, reduce data usage for the client, and improve the time to first render of your pages. — *PageSpeed Insights*

Compression can be enabled with this:

```
AddOutputFilterByType DEFLATE "text/html"/
                              "text/plain"/
                              "text/xml"/
                              "text/css"/
                              "text/javascript"/
                              "application/javascript"
```

Apache Docs

Leverage Browser Caching (Apache 2.0+)

> Fetching resources over the network is both slow and expensive: the download may require multiple roundtrips between the client and server, which delays processing and may block rendering of page content, and also incurs data costs for the visitor. All server responses should specify a caching policy to help the client determine if and when it can reuse a previously fetched response. — *PageSpeed Insights*

You can leverage browser caching like this:

```
# Enable browser caching
ExpiresActive On

# Set the default caching duration
ExpiresDefault "access plus 1 week"

# Change the caching duration by file type
ExpiresByType text/html "access plus 2 weeks"
```

Apache Docs

Enable KeepAlive (Apache 2.0+)

> The Keep-Alive extension to HTTP/1.0 and the persistent connection feature of HTTP/1.1 provide long-lived HTTP sessions which allow multiple requests to be sent over the same TCP connection. In some cases this has been shown to result in an

> almost 50% speedup in latency times for HTML documents with many images. To enable Keep-Alive connections, set KeepAlive On. — *Apache Docs*

```
# Enable KeepAlive
KeepAlive On

# OPTIONAL — limit the amount of requests per connection with 'MaxKeepAliveRequests'
# Example: MaxKeepAliveRequests 500

# OPTIONAL — limit the amount of time the server will wait before it closes
# the connection with 'KeepAliveTimeout'
# Example: KeepAliveTimeout 500
```

Apache Docs

Read Speed Optimization online: https://riptutorial.com/dot-htaccess/topic/3893/speed-optimization

Credits

S. No	Chapters	Contributors
1	Getting started with Hypertext Access file	Community, Dilip Raj Baral, hjpotter92, James Oswald, Lag, Mike Rockétt, tbodt
2	Denying Access	Dilip Raj Baral, John R Perry, tbodt
3	General Security and Hack Prevention	ban17, Dilip Raj Baral, John R Perry, Lag, Meysam, Mike Rockétt, OpenWebWar
4	Handling File Types	Dilip Raj Baral, John R Perry, Jon Lin, Marvin, mauris, Mike Rockétt, Nicholas Qiao
5	Rewriting and Redirecting	Bogdan Alexandru Militaru, Dilip Raj Baral, Florian Lemaitre, hjpotter92, James, John R Perry, Mike Rockétt, shaN, Sven Reuter
6	Speed Optimization	John R Perry