Hello! Thanks for visiting! This site looks best in any browser that *isn't* Internet Explorer. The information should be readable, but if not please switch to a different theme.

Warning – JavaScript Required
Unfortunately your browser has disabled scripting. To view the page, enable JavaScript or click here to use a different theme.

Stupid htaccess Tricks

Posted on January 10, 2006 in Websites by Jeff Starr

Welcome to Perishable Press! This article, Stupid htaccess Tricks, covers just about every htaccess "trick" in the book, and is easily the site's most popular offering. In addition to this htaccess article, you may also want to explore the rapidly expanding htaccess tag archive. Along with all things htaccess, Perishable Press also focuses on (X)HTML, CSS, PHP, JavaScript, security, and just about every other aspect of web design, blogging, and online success. If these topics are of interest to you, I encourage you to subscribe to Perishable Press for a periodic dose of online enlightenment ;)

Table of Contents

General Information [^]

.htaccess Definition [1] ^

Apache server software provides distributed (i.e., directory-level) configuration via *Hypertext Access* files. These `.htaccess` files enable the localized fine-tuning of Apache's universal system-configuration directives, which are defined in Apache's main configuration file. The localized `.htaccess` directives must operate from within a file named `.htaccess`. The user must have appropriate file permissions to access and/or edit the `.htaccess` file. Further, `.htaccess` file permissions should never allow world write access — a secure permissions setting is "644", which allows universal read access and user-only write access. Finally, `.htaccess` rules apply to the parent directory and all subdirectories. Thus to apply configuration rules to an entire website, place the `.htaccess` file in the root directory of the site.

Commenting .htaccess Code ^

Comments are essential to maintaining control over any involved portion of code. Comments in `.htaccess` code are fashioned on a per-line basis, with each line of comments beginning with a pound sign #. Thus, comments spanning multiple lines in the `.htaccess` file require multiple pound signs. Further, due to the extremely volatile nature of htaccess voodoo, it is wise to include only alphanumeric characters (and perhaps a few dashes and underscores) in any `.htaccess` comments.

Important Notes for .htaccess Noobs ^

As a configuration file, `.htaccess` is very powerful. Even the slightest syntax error (like a missing space) can result in severe server malfunction. Thus it is crucial to make backup copies of *everything* related to your site (including any original `.htaccess` files) *before* working with your Hypertext Access file(s). It is also important to check your entire website thoroughly after making any changes to your `.htaccess` file. If any errors or other problems are encountered, employ your backups immediately to restore original functionality.

Performance Issues ^

`.htaccess` directives provide directory-level configuration without requiring access to Apache's main server cofiguration file (httpd.conf). However, due to performance and security concerns, the main configuration file should always be used for server directives whenever possible. For example, when a server is configured to process `.htaccess` directives, Apache must search every directory within the domain and load any and all `.htaccess` files upon every document request. This results in increased page processing time and thus decreases performance. Such a performance hit may be unnoticeable for sites with light traffic, but becomes a more serious issue for more popular websites. Therefore, `.htaccess` files should only be used when the main server configuration file is inaccessible. See the "Performance Tricks" section of this article for more information.

Regex Character Definitions for htaccess [2] ^

`#`
: the # instructs the server to ignore the line. used for including comments. each line of comments requires it's own #. when including comments, it is good practice to use only letters, numbers, dashes, and underscores. this practice will help eliminate/avoid potential server parsing errors.

`[F]`
: Forbidden: instructs the server to return a `403 Forbidden` to the client.

`[L]`
: Last rule: instructs the server to stop rewriting after the preceding directive is processed.

`[N]`
: Next: instructs Apache to rerun the rewrite rule until all rewriting directives have been achieved.

`[G]`
: Gone: instructs the server to deliver `Gone (no longer exists)` status message.

`[P]`
: Proxy: instructs server to handle requests by `mod_proxy`

`[C]`
: Chain: instructs server to chain the current rule with the previous rule.

`[R]`
: Redirect: instructs Apache to issue a redirect, causing the browser to request the rewritten/modified URL.

`[NC]`
: No Case: defines any associated argument as case-*in*sensitive. i.e., "NC" = "No Case".

`[PT]`
: Pass Through: instructs `mod_rewrite` to pass the rewritten URL back to Apache for further processing.

`[OR]`
: Or: specifies a logical "or" that ties two expressions together such that either one proving true will cause the associated rule to be applied.

`[NE]`
: No Escape: instructs the server to parse output without escaping characters.

`[NS]`
: No Subrequest: instructs the server to skip the directive if internal sub-request.

`[QSA]`
: Append Query String: directs server to add the query string to the end of the expression (URL).

`[S=x]`
: Skip: instructs the server to skip the next "x" number of rules if a match is detected.

`[E=variable:value]`

Environmental Variable: instructs the server to set the environmental variable "variable" to "value".

`[T=MIME-type]`

Mime Type: declares the mime type of the target resource.

`[]`

specifies a character class, in which any character within the brackets will be a match. e.g., [xyz] will match either an x, y, or z.

`[]+`

character class in which any combination of items within the brackets will be a match. e.g., [xyz]+ will match any number of x's, y's, z's, or any combination of these characters.

`[^]`

specifies *not* within a character class. e.g., [^xyz] will match any character that is neither x, y, nor z.

`[a-z]`

a dash (-) between two characters within a character class ([]) denotes the range of characters between them. e.g., [a-zA-Z] matches all lowercase and uppercase letters from a to z.

`a{n}`

specifies an exact number, n, of the preceding character. e.g., x{3} matches exactly three x's.

`a{n,}`

specifies n or more of the preceding character. e.g., x{3,} matches three or more x's.

`a{n,m}`

specifies a range of numbers, between n and m, of the preceding character. e.g., x{3,7} matches three, four, five, six, or seven x's.

`()`

used to group characters together, thereby considering them as a single unit. e.g., (perishable)?press will match press, with or without the perishable prefix.

`^`

denotes the beginning of a regex (regex = regular expression) test string. i.e., begin argument with the proceeding character.

`$`

denotes the end of a regex (regex = regular expression) test string. i.e., end argument with the previous character.

`?`

declares as optional the preceding character. e.g., monzas? will match monza or monzas, while mon(za)? will match either mon or monza. i.e., x? matches zero or one of x.

`!`

declares negation. e.g., "`!string`" matches everything except "`string`".

`.`

a dot (or period) indicates any single arbitrary character.

`-`

instructs "not to" rewrite the URL, as in "`...domain.com.* - [F]`".

`+`

matches one or more of the preceding character. e.g., G+ matches one or more G's, while "+" will match one or more characters of any kind.

`*`

matches zero or more of the preceding character. e.g., use "`.*`" as a wildcard.

`|`

declares a logical "or" operator. for example, `(x|y)` matches x or y.

`\`

escapes special characters (`^ $ ! . * |`). e.g., use "`\.`" to indicate/escape a literal dot.

`\.`

indicates a literal dot (escaped).

`/*`

zero or more slashes.

`.*`

zero or more arbitrary characters.

`^$`

defines an empty string.

`^.*$`

the standard pattern for matching everything.

`[^/.]`

defines one character that is neither a slash nor a dot.

`[^/.]+`

defines any number of characters which contains neither slash nor dot.

`http://`

this is a literal statement — in this case, the literal character string, “http://”.

`^domain.*`

defines a string that begins with the term “domain”, which then may be proceeded by any number of any characters.

`^domain\.com$`

defines the exact string “domain.com”.

`-d`

tests if string is an existing directory

`-f`

tests if string is an existing file

`-s`

tests if file in test string has a non-zero value

Redirection Header Codes ^

- 301 - Moved Permanently
- 302 - Moved Temporarily
- 403 - Forbidden
- 404 - Not Found
- 410 - Gone

Essentials [^]

Commenting your htaccess Files ^

It is an excellent idea to consistenly and logically comment your htaccess files. Any line in an htaccess file that begins with the pound sign (#) tells the server to ignore it. Multiple lines require multiple pounds and use letters/numbers/dash/underscore only:

```
# this is a comment
# each line must have its own pound sign
# use only alphanumeric characters along with dashes - and underscores _
```

Enable Basic Rewriting ^

Certain servers may not have “mod_rewrite” enabled by default. To ensure mod_rewrite (basic rewriting) is enabled throughout your site, add the following line once to your site’s root htaccess file:

```
# enable basic rewriting
RewriteEngine on
```

Enable Symbolic Links ^

Enable symbolic links (symlinks) by adding the following directive to the target directory’s htaccess file. Note: for the FollowSymLinks directive to function, AllowOverride Options privileges must be enabled from within the server configuration file (see proceeding paragraph for more information):

```
# enable symbolic links
Options +FollowSymLinks
```

Enable AllowOverride ^

For directives that require AllowOverride in order to function, such as FollowSymLinks (see above paragraph), the following directive must be added to the server configuration file. For performance considerations, it is important to only enable AllowOverride in the specific directory or directories in which it is required. In the following code chunk, we are enabling the AllowOverride privs only in the specified directory (/www/replace/this/with/actual/directory). Refer to this section for more information about AllowOverride and performance enhancement:

```
# enable allowoverride privileges
<Directory /www/replace/this/with/actual/directory>
 AllowOverride Options
</Directory>
```

Rename the htaccess File ^

Not every system enjoys the extension-only format of htaccess files. Fortunately, you can rename them to whatever you wish, granted the name is valid on your system. Note: This directive must be placed in the server-wide configuration file or it will not work:

```
# rename htaccess files
AccessFileName ht.access
```

Note: If you rename your htaccess files, remember to update any associated configuration settings. For example, if you are protecting your htaccess file via `FilesMatch`, remember to inform it of the renamed files:

```
# protect renamed htaccess files
<FilesMatch "^ht\.">
 Order deny,allow
 Deny from all
</FilesMatch>
```

Retain Rules Defined in httpd.conf ^

Save yourself time and effort by defining replicate rules for multiple virtual hosts once and only once via your httpd.conf file. Then, simply instruct your target htaccess file(s) to inherit the httpd.conf rules by including this directive:

```
RewriteOptions Inherit
```

Performance [^]

Improving Performance via AllowOverride ^

Limit the extent to which htaccess files decrease performance by enabling `AllowOverride` only in required directories. For example, if `AllowOverride` is enabled throughout the entire site, the server must dig through every directory, searching for htaccess files that may not even exist. To prevent this, we disable the `AllowOverride` in the site's root htaccess file and then enable `AllowOverride` only in required directories via the server config file (refer to this section for more information). Note: if you do not have access to your site's server config file and also need `AllowOverride` privileges, do not use this directive:

```
# increase performance by disabling allowoverride
AllowOverride None
```

Improving Performance by Passing the Character Set ^

Prevent certain 500 error displays by passing the default character set parameter before you get there. Note: replace the "utf-8" below with the charset that your site is using:

```
# pass the default character set
AddDefaultCharset utf-8
```

Improving Performance by Preserving Bandwidth ^

To increase performance on PHP enabled servers, add the following directive:

```
# preserve bandwidth for PHP enabled servers
<ifmodule mod_php4.c>
 php_value zlib.output_compression 16386
</ifmodule>
```

Disable the Server Signature ^

Here we are disabling the digital signature that would otherwise identify the server:

```
# disable the server signature
ServerSignature Off
```

Set the Server Timezone ^

Here we are instructing the server to synchronize chronologically according to the time zone of some specified state:

```
# set the server timezone
```

```
SetEnv TZ America/Washington
```

Set the Email Address for the Server Administrator ^

Here we are specifying the default email address for the server administrator:

```
# set the server administrator email
SetEnv SERVER_ADMIN default@domain.com
```

Improve Site Transfer Speed by Enabling File Caching ^

The htaccess genius over at askapache.com explains how to dramatically improve your site's transfer speed by enabling file caching [3]. Using time in seconds* to indicate the duration for which cached content should endure, we may generalize the htaccess rules as such (edit file types and time value to suit your needs):

```
# cache images and flash content for one month
<FilesMatch ".(flv|gif|jpg|jpeg|png|ico|swf)$">
Header set Cache-Control "max-age=2592000"
</FilesMatch>

# cache text, css, and javascript files for one week
<FilesMatch ".(js|css|pdf|txt)$">
Header set Cache-Control "max-age=604800"
</FilesMatch>

# cache html and htm files for one day
<FilesMatch ".(html|htm)$">
Header set Cache-Control "max-age=43200"
</FilesMatch>

# implement minimal caching during site development
<FilesMatch "\.(flv|gif|jpg|jpeg|png|ico|js|css|pdf|swf|html|htm|txt)$">
Header set Cache-Control "max-age=5"
</FilesMatch>

# explicitly disable caching for scripts and other dynamic files
<FilesMatch "\.(pl|php|cgi|spl|scgi|fcgi)$">
Header unset Cache-Control
</FilesMatch>

# alternate method for file caching
ExpiresActive On
ExpiresDefault A604800 # 1 week
ExpiresByType image/x-icon A2419200 # 1 month
ExpiresByType application/x-javascript A2419200 # 1 month
ExpiresByType text/css A2419200 # 1 month
ExpiresByType text/html A300 # 5 minutes
# disable caching for scripts and other dynamic files
<FilesMatch "\.(pl|php|cgi|spl|scgi|fcgi)$">
ExpiresActive Off
</FilesMatch>
```

- * **Convert common time intervals into seconds:**
- 300 = 5 minutes
- 2700 = 45 minutes
- 3600 = 1 hour
- 54000 = 15 hours
- 86400 = 1 day
- 518400 = 6 days
- 604800 = 1 week
- 1814400 = 3 weeks
- 2419200 = 1 month
- 26611200 = 11 months
- 29030400 = 1 year = never expires

Set the default language and character set ^

Here is an easy way to set the default language for pages served by your server (edit the language to suit your needs):

```
# set the default language
DefaultLanguage en-US
```

Likewise, here we are setting the default character set (edit to taste):

```
# set the default character set
AddDefaultCharset UTF-8
```

Declare specific/additional MIME types ^

```
# add various mime types
AddType application/x-shockwave-flash .swf
AddType video/x-flv .flv
AddType image/x-icon .ico
```

Send character set and other headers without meta tags ^

```
# send the language tag and default character set
# AddType 'text/html; charset=UTF-8' html
AddDefaultCharset UTF-8
DefaultLanguage en-US
```

Limit server request methods to GET and PUT ^

```
# limit server request methods to GET and PUT
Options -ExecCGI -Indexes -All
RewriteEngine on
RewriteCond %{REQUEST_METHOD} ^(TRACE|TRACK|OPTIONS|HEAD) RewriteRule .* - [F]
```

Selectively process files according to server request method ^

```
# process files according to server request method
Script PUT /cgi-bin/upload.cgi
Script GET /cgi-bin/download.cgi
```

Execute various file types through a cgi script ^

For those special occasions where certain file types need to be processed with some specific cgi script, let em know who sent ya:

```
# execute all png files via png-script.cgi
Action image/png /cgi-bin/png-script.cgi
```

Security [^]

Prevent Access to .htaccess ^

Add the following code block to your htaccess file to add an extra layer of security. Any attempts to access the htaccess file will result in a 403 error message. Of course, your first layer of defense to protect htaccess files involves setting htaccess file permissions via CHMOD to 644:

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

Prevent Access to a Specific File ^

To restrict access to a specific file, add the following code block and edit the file name, “secretfile.jpg”, with the name of the file that you wish to protect:

```
# prevent viewing of a specific file
<files secretfile.jpg>
 order allow,deny
 deny from all
</files>
```

Prevent access to multiple file types ^

To restrict access to a variety of file types, add the following code block and edit the file types within parentheses to match the extensions of any files that you wish to protect:

```
<FilesMatch "\.(htaccess|htpasswd|ini|phps|fla|psd|log|sh)$">
 Order Allow,Deny
 Deny from all
</FilesMatch>
```

Prevent Unauthorized Directory Browsing ^

Prevent unauthorized directory browsing by instructing the server to serve a "xxx Forbidden - Authorization Required" message for any request to view a directory. For example, if your site is missing it's default index page, everything within the root of your site will be accessible to all visitors. To prevent this, include the following htaccess rule:

```
# disable directory browsing
Options All -Indexes
```

Conversely, to enable directory browsing, use the following directive:

```
# enable directory browsing
Options All +Indexes
```

Likewise, this rule will prevent the server from listing directory contents:

```
# prevent folder listing
IndexIgnore *
```

And, finally, the `IndexIgnore` directive may be used to prevent the display of select file types:

```
# prevent display of select file types
IndexIgnore *.wmv *.mp4 *.avi *.etc
```

Change Default Index Page ^

This rule tells the server to search for and serve "business.html" as the default directory index. This rule must exist in the htaccess files of the root directory for which you wish to replace the default index file (e.g., "index.html"):

```
# serve alternate default index page
DirectoryIndex business.html
```

This rule is similar, only in this case, the server will scan the root directory for the listed files and serve the first match it encounters. The list is read from left to right:

```
# serve first available alternate default index page from series
DirectoryIndex filename.html index.cgi index.pl default.htm
```

Disguise Script Extensions ^

To enhance security, disguise scripting languages by replacing actual script extensions with dummy extensions of your choosing. For example, to change the "`.foo`" extension to "`.php`", add the following line to your htaccess file and rename all affected files accordingly:

```
# serve foo files as php files
AddType application/x-httpd-php .foo

# serve foo files as cgi files
AddType application/x-httpd-cgi .foo
```

Limit Access to the Local Area Network (LAN) ^

```
# limit access to local area network
<Limit GET POST PUT>
order deny,allow
deny from all
allow from 192.168.0.0/33
</Limit>
```

Secure Directories by IP Address and/or Domain ^

In the following example, all IP addresses are *allowed access* except for 12.345.67.890 and domain.com:

```
# allow all except those indicated here
<Limit GET POST PUT>
order allow,deny
allow from all
deny from 12.345.67.890
deny from .*domain\.com.*
</Limit>
```

In the following example, all IP addresses are *denied access* except for 12.345.67.890 and domain.com:

```
# deny all except those indicated here
<Limit GET POST PUT>
order deny,allow
deny from all
allow from 12.345.67.890
allow from .*domain\.com.*
</Limit>
```

This is how to block unwanted visitors based on the referring domain. You can also save bandwidth by blocking specific file types — such as `.jpg, .zip, .mp3, .mpg` — from specific referring domains. Simply replace “scumbag” and “wormhole” with the offending domains of your choice:

```
# block visitors referred from indicated domains
<IfModule mod_rewrite.c>
RewriteEngine on
RewriteCond %{HTTP_REFERER} scumbag\.com [NC,OR]
RewriteCond %{HTTP_REFERER} wormhole\.com [NC,OR]
RewriteRule .* - [F]
</ifModule>
```

Prevent or allow domain access for a specified range of IP addresses ^

There are several effective ways to block a range of IP addresses via htaccess. This first method blocks an IP range specified by their CIDR (Classless Inter-Domain Routing) number. This method is useful for blocking mega-spammers such as RIPE, Optinet, and others. If, for example, you find yourself adding line after line of Apache deny directives for addresses beginning with the same first few numbers, choose one of them and try a whois lookup. Listed within the whois results will be the CIDR value representing every IP address associated with that particular network. Thus, blocking via CIDR is an effective way to eloquently prevent all IP instances of the offender from accessing your site. Here is a generalized example for blocking by CIDR (edit values to suit your needs):

```
# block IP range by CIDR number
<Limit GET POST PUT>
order allow,deny
allow from all
deny from 10.1.0.0/16
deny from 80.0.0/8
</Limit>
```

Likewise, to allow an IP range by CIDR number:

```
# allow IP range by CIDR number
```

```
<Limit GET POST PUT>
order deny,allow
deny from all
allow from 10.1.0.0/16
allow from 80.0.0/8
</Limit>
```

Another effective way to block an entire range of IP addresses involves truncating digits until the desired range is represented. As an IP address is read from left to right, its value represents an increasingly specific address. For example, a fictitious IP address of 99.88.77.66 would designate some uniquely specific IP address. Now, if we remove the last two digits (66) from the address, it would represent *any* address beginning with the remaining digits. That is, 99.88.77 represents 99.88.77.1, 99.88.77.2, ... 99.88.77.99, ...etc. Likewise, if we then remove another pair of digits from the address, its range suddenly widens to represent every IP address 99.88.x.y, where x and y represent any valid set of IP address values (i.e., you would block 256*256 = 65,536 unique IP addresses). Following this logic, it is possible to block an entire range of IP addresses to varying degrees of specificity. Here are few generalized lines exemplifying proper htaccess syntax (edit values to suit your needs):

```
# block IP range by address truncation
<Limit GET POST PUT>
order allow,deny
allow from all
deny from 99.88.77.66
deny from 99.88.77.*
deny from 99.88.*.*
deny from 99.*.*.*
</Limit>
```

Likewise, to allow an IP range by address truncation:

```
# allow IP range by address truncation
<Limit GET POST PUT>
order deny,allow
deny from all
allow from 99.88.77.66
allow from 99.88.77.*
allow from 99.88.*.*
allow from 99.*.*.*
</Limit>
```

Block or allow multiple IP addresses on one line ^

Save a little space by blocking multiple IP addresses or ranges on one line. Here are few examples (edit values to suit your needs):

```
# block two unique IP addresses
deny from 99.88.77.66 11.22.33.44
# block three ranges of IP addresses
deny from 99.88 99.88.77 11.22.33
```

Likewise, to allow multiple IP addresses or ranges on one line:

```
# allow two unique IP addresses
allow from 99.88.77.66 11.22.33.44
# allow three ranges of IP addresses
allow from 99.88 99.88.77 11.22.33
```

Miscellaneous rules for blocking and allowing IP addresses ^

Here are few miscellaneous rules for blocking various types of IP addresses. These rules may be adapted to *allow* the specified IP values by simply changing the deny directive to `allow`. Check 'em out (edit values to suit your needs):

```
# block a partial domain via network/netmask values
deny from 99.1.0.0/255.255.0.0

# block a single domain
```

```
deny from 99.88.77.66

# block domain.com but allow sub.domain.com
order deny,allow
deny from domain.com
allow from sub.domain.com
```

Stop Hotlinking, Serve Alternate Content ^

To serve 'em some unexpected alternate content when hotlinking is detected, employ the following code, which will protect all files of the types included in the last line (add more types as needed). Remember to replace the dummy path names with real ones. Also, the name of the nasty image being served in this case is "eatme.jpe", as indicated in the line containing the `RewriteRule`. Please advise that this method will also block services such as FeedBurner from accessing your images.

```
# stop hotlinking and serve alternate content
<IfModule mod_rewrite.c>
 RewriteEngine on
 RewriteCond %{HTTP_REFERER} !^$
 RewriteCond %{HTTP_REFERER} !^http://(www\.)?domain\.com/.*$ [NC]
 RewriteRule .*\.(gif|jpg)$ http://www.domain.com/eatme.jpe [R,NC,L]
</ifModule>
```

Note: To deliver a standard (or custom, if configured) error page instead of some nasty image of the Fonz, replace the line containing the `RewriteRule` in the above htaccess directive with the following line:

```
# serve a standard 403 forbidden error page
RewriteRule .*\.(gif|jpg)$ - [F,L]
```

Note: To grant linking permission to a site other than yours, insert this code block after the line containing the "domain.com" string. Remember to replace "goodsite.com" with the actual site domain:

```
# allow linking from the following site
RewriteCond %{HTTP_REFERER} !^http://(www\.)?goodsite\.com/.*$ [NC]
```

Block Evil Robots, Site Rippers, and Offline Browsers ^

Eliminate some of the unwanted scum from your userspace by injecting this handy block of code. After such, any listed agents will be denied access and receive an error message instead. Please advise that there are much more comprehensive lists available this example has been truncated for business purposes. Note: DO NOT include the "[OR]" on the very last `RewriteCond` or your server will crash, delivering "500 Errors" to all page requests.

```
# deny access to evil robots site rippers offline browsers and other nasty scum
RewriteBase /
RewriteCond %{HTTP_USER_AGENT} ^Anarchie [OR]
RewriteCond %{HTTP_USER_AGENT} ^ASPSeek [OR]
RewriteCond %{HTTP_USER_AGENT} ^attach [OR]
RewriteCond %{HTTP_USER_AGENT} ^autoemailspider [OR]
RewriteCond %{HTTP_USER_AGENT} ^Xaldon\ WebSpider [OR]
RewriteCond %{HTTP_USER_AGENT} ^Xenu [OR]
RewriteCond %{HTTP_USER_AGENT} ^Zeus.*Webster [OR]
RewriteCond %{HTTP_USER_AGENT} ^Zeus
RewriteRule ^.* - [F,L]
```

Or, instead of delivering a friendly error message (i.e., the last line), send these bad boys to the hellish website of your choice by replacing the `RewriteRule` in the last line with one of the following two examples:

```
# send em to a hellish website of your choice
RewriteRule ^.*$ http://www.hellish-website.com [R,L]
```

Or, to send em to a virtual blackhole of fake email addresses:

```
# send em to a virtual blackhole of fake email addresses
RewriteRule ^.*$ http://english-61925045732.spampoison.com [R,L]
```

You may also include specific referrers to your blacklist by using HTTP_REFERER. Here, we use the infamously scummy domain, “iaea.org” as our blocked example, and we use “yourdomain” as your domain (the domain to which you are blocking iaea.org):

```
RewriteCond %{HTTP_REFERER} ^http://www.iaea.org$
RewriteRule !^http://[^/.]\.yourdomain\.com.* - [F,L]
```

More Stupid Blocking Tricks ^

Note: Although these redirect techniques are aimed at blocking and redirecting nasty scumsites, the directives may also be employed for friendly redirection purposes:

```
# redirect any request for anything from spamsite to differentspamsite
RewriteCond %{HTTP_REFERER} ^http://.*spamsite.*$ [NC]
RewriteRule .* http://www.differentspamsite.com [R]

# redirect all requests from spamsite to an image of something at differentspamsite
RewriteCond %{HTTP_REFERER} ^http://.*spamsite.*$ [NC]
RewriteRule .* http://www.differentspamsite/something.jpg [R]

# redirect traffic from a certain address or range of addresses to another site
RewriteCond %{REMOTE_ADDR} 192.168.10.*
RewriteRule .* http://www.differentspamsite.com/index.html [R]
```

Even More Scum-Blocking Tricks ^

Here is a step-by-step series of code blocks that should equip you with enough knowledge to block any/all necessary entities. Read through the set of code blocks, observe the patterns, and then copy, combine and customize to suit your specific scum-blocking needs:

```
# set variables for user agents and referers and ip addresses
SetEnvIfNoCase User-Agent ".*(user-agent-you-want-to-block|php/perl).*" BlockedAgent
SetEnvIfNoCase Referer ".*(block-this-referrer|and-this-referrer|and-this-referrer).*"
BlockedReferer
SetEnvIfNoCase REMOTE_ADDR ".*(666.666.66.0|22.22.22.222|999.999.99.999).*" BlockedAddress

# set variable for any class B network coming from a given netblock
SetEnvIfNoCase REMOTE_ADDR "66.154.*" BlockedAddress

# set variable for two class B networks 198.25.0.0 and 198.26.0.0
SetEnvIfNoCase REMOTE_ADDR "198.2(5|6)\..*" BlockedAddress

# deny any matches from above and send a 403 denied
<Limit GET POST PUT>
 order deny,allow
 deny from env=BlockedAgent
 deny from env=BlockedReferer
 deny from env=BlockedAddress
 allow from all
</Limit>
```

Password-Protect Directories ^

Here is an excellent online tool for generating the necessary elements for a password-protected directory:

```
# password protect directories
```

htaccess Password Generator

Password-protect Files, Directories, and More.. ^

Secure site contents by requiring user authentication for specified files and/or directories. The first example shows how to password-protect any single file type that is present beneath the directory which houses the htaccess rule. The second rule employs the FilesMatch directive to protect any/all files which match any of the specified character strings. The third rule demonstrates how to protect an entire directory. The fourth set of rules provides password-protection for all IP’s except those specified. Remember to edit these rules according to your specific needs.

```
# password-protect single file
<Files secure.php>
AuthType Basic
AuthName "Prompt"
AuthUserFile /home/path/.htpasswd
Require valid-user
</Files>

# password-protect multiple files
<FilesMatch "^(execute|index|secure|insanity|biscuit)*$">
AuthType basic
AuthName "Development"
AuthUserFile /home/path/.htpasswd
Require valid-user
</FilesMatch>

# password-protect the directory in which this htaccess rule resides
AuthType basic
AuthName "This directory is protected"
AuthUserFile /home/path/.htpasswd
AuthGroupFile /dev/null
Require valid-user

# password-protect directory for every IP except the one specified
# place in htaccess file of a directory to protect that entire directory
AuthType Basic
AuthName "Personal"
AuthUserFile /home/path/.htpasswd
Require valid-user
Allow from 99.88.77.66
Satisfy Any
```

Require SSL (Secure Sockets Layer) ^

Here is an excellent method for requiring SSL (via askapache.com [3]):

```
# require SSL
SSLOptions +StrictRequire
SSLRequireSSL
SSLRequire %{HTTP_HOST} eq "domain.tld"
ErrorDocument 403 https://domain.tld

# require SSL without mod_ssl
RewriteCond %{HTTPS} !=on [NC]
RewriteRule ^.*$ https://%{SERVER_NAME}%{REQUEST_URI} [R,L]
```

Automatically CHMOD Various File Types ^

This method is great for ensuring the CHMOD settings for various file types. Employ the following rules in the root htaccess file to affect *all* specified file types, or place in a specific directory to affect only those files (edit file types according to your needs):

```
# ensure CHMOD settings for specified file types
# remember to never set CHMOD 777 unless you know what you are doing
# files requiring write access should use CHMOD 766 rather than 777
# keep specific file types private by setting their CHMOD to 400
chmod .htpasswd files 640
chmod .htaccess files 644
chmod php files 600
```

Disguise all file extensions ^

This method will disguise all file types (i.e., any file extension) and present them as .php files (or whichever extension you choose):

```
# diguise all file extensions as php
ForceType application/x-httpd-php
```

Protect against denial-of-service (DOS) attacks by limiting file upload size ^

One method to help protect your server against DOS attacks involves limiting the maximum allowable size for file uploads. Here, we are limiting file upload size to 10240000 bytes, which is equivalent to around 10 megabytes. For this rule, file sizes are expressed in bytes. Check here for help with various file size conversions. Note: this code is only useful if you actually allow users to upload files to your site.

```
# protect against DOS attacks by limiting file upload size
LimitRequestBody 10240000
```

Secure directories by disabling execution of scripts ^

Prevent malicious brainiacs from actively scripting secure directories by adding the following rules to the representative htaccess file (edit file types to suit your needs):

```
# secure directory by disabling script execution
AddHandler cgi-script .php .pl .py .jsp .asp .htm .shtml .sh .cgi
Options -ExecCGI
```

Usability Tricks [^]

Minimize CSS Image Flicker in IE6 ^

Add the following htaccess rules to minimize or even eliminate CSS background-image “flickering” in MSIE6:

```
# minimize image flicker in IE6
ExpiresActive On
ExpiresByType image/gif A2592000
ExpiresByType image/jpg A2592000
ExpiresByType image/png A2592000
```

Deploy Custom Error Pages ^

Replicate the following patterns to serve your own set of custom error pages. Simply replace the “`/errors/###.html`” with the correct path and file name. Also change the “`###`” preceding the path to summon pages for other errors. Note: your custom error pages must be larger than 512 bytes in size or they will be completely ignored by Internet Explorer:

```
# serve custom error pages
ErrorDocument 400 /errors/400.html
ErrorDocument 401 /errors/401.html
ErrorDocument 403 /errors/403.html
ErrorDocument 404 /errors/404.html
ErrorDocument 500 /errors/500.html
```

Provide a Universal Error Document ^

```
# provide a universal error document
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^.*$ /dir/error.php [L]
```

Employ Basic URL Spelling Check ^

This bit of voodoo will auto-correct simple spelling errors in the URL:

```
# automatically corect simple speling erors
<IfModule mod_speling.c>
 CheckSpelling On
</IfModule>
```

Instruct browser to download multimedia files rather than display them ^

Here is a useful method for delivering multimedia file downloads to your users. Typically, browsers will attempt to play or stream such files when direct links are clicked. With this method, provide a link to a multimedia file and a dialogue box will provide users the choice of saving the file or opening it. Here are a few htaccess rules demonstrating the technique (edit file types according to your specific needs):

```
# instruct browser to download multimedia files
AddType application/octet-stream .avi
AddType application/octet-stream .mpg
AddType application/octet-stream .wmv
AddType application/octet-stream .mp3
```

Instruct server to display source code for dynamic file types ^

There are many situations where site owners may wish to display the contents of a dynamic file rather than executing it as a script. To exercise this useful technique, create a directory in which to place dynamic files that should be displayed rather than executed, and add the following line of code to the htaccess file belonging to that directory. This method is known to work for `.pl`, `.py`, and `.cgi` file-types. Here it is:

```
RemoveHandler cgi-script .pl .py .cgi
```

Redirect visitors to a temporary site during site development ^

During web development, maintenance, or repair, send your visitors to an alternate site while retaining full access for yourself. This is a very useful technique for preventing visitor confusion or dismay during those awkward, web-development moments. Here are the generalized htaccess rules to do it (edit values to suit your needs):

```
# redirect all visitors to alternate site but retain full access for you
ErrorDocument 403 http://www.alternate-site.com
Order deny,allow
Deny from all
Allow from 99.88.77.66
```

Provide a password prompt for visitors during site development ^

Here is another possible solution for "hiding" your site during those private, site-under-construction moments. Here we are instructing Apache to provide visitors with a password prompt while providing open access to any specifically indicated IP addresses or URL's. Edit the following code according to your IP address and other development requirements (thanks to Caleb at askapache.com for sharing this trick [8]):

```
# password prompt for visitors
AuthType basic
AuthName "This site is currently under construction"
AuthUserFile /home/path/.htpasswd
AuthGroupFile /dev/null
Require valid-user
# allow webmaster and any others open access
Order Deny,Allow
Deny from all
Allow from 111.222.33.4
Allow from favorite.validation/services/
Allow from googlebot.com
Satisfy Any
```

Prevent file or directory access according to specified time periods ^

Prevent viewing of all pictures of Fonzi during the midnight hour — or *any* files during *any* time period — by using this handy htaccess ruleset:

```
# prevent access during the midnight hour
RewriteCond %{TIME_HOUR} ^12$
RewriteRule ^.*$ - [F,L]

# prevent access throughout the afternoon
RewriteCond %{TIME_HOUR} ^(12|13|14|15)$
```

```
RewriteRule ^.*$ - [F,L]
```

Redirect Tricks [^]

Important Note About Redirecting via mod_rewrite ^

For all redirects using the `mod_rewrite` directive, it is necessary to have the `RewriteEngine` enabled. It is common practice to enable the `mod_rewrite` directive in either the server configuration file or at the top of the site's root htaccess file. If the `mod_rewrite` directive is not included in either of these two places, it should be included as the first line in any code block that utilizes a rewrite function (i.e., `mod_rewrite`), but only needs to be included once for each htaccess file. The proper `mod_rewrite` directive is included here for your convenience, but may or may not also be included within some of the code blocks provided in this article:

```
# initialize and enable rewrite engine
RewriteEngine on
```

Redirect from http://www.domain.com to http://domain.com ^

This method uses a "301 redirect" to establish a permanent redirect from the "www-version" of a domain to its respectively corresponding "non-www version". Be sure to test *immediately* after preparing 301 redirects and remove it immediately if any errors occur. Use a "server header checker" to confirm a positive 301 response. Further, always include a trailing slash "/" when linking directories. Finally, be consistent with the "www" in all links (either use it always or never).

```
# permanently redirect from www domain to non-www domain
RewriteEngine on
Options +FollowSymLinks
RewriteCond %{HTTP_HOST} ^www\.domain\.tld$ [NC]
RewriteRule ^(.*)$ http://domain.tld/$1 [R=301,L]
```

Redirect from http://old-domain.com to http://new-domain.com ^

For a basic domain change from "old-domain.com" to "new-domain.com" (and folder/file names have *not* been changed), use the `Rewrite` rule to remap the old domain to the new domain. When checking the redirect live, the old domain may appear in the browser's address bar. Simply check an image path (right-click an image and select "properties") to verify proper redirection. Remember to check your site thoroughly after implementing this redirect.

```
# redirect from old domain to new domain
RewriteEngine On
RewriteRule ^(.*)$ http://www.new-domain.com/$1 [R=301,L]
```

Redirect String Variations to a Specific Address ^

For example, if we wanted to redirect any requests containing the character string, "perish", to our main page at http://perishablepress.com/, we would replace "some-string" with "perish" in the following code block:

```
# redirect any variations of a specific character string to a specific address
RewriteRule ^some-string http://www.domain.com/index.php/blog/target [R]
```

Here are two other methods for accomplishing string-related mapping tasks:

```
# map URL variations to the same directory on the same server
AliasMatch ^/director(y|ies) /www/docs/target

# map URL variations to the same directory on a different server
RedirectMatch ^/[dD]irector(y|ies) http://domain.com
```

Other Fantastic Redirect Tricks ^

Redirect an entire site via 301:

```
# redirect an entire site via 301
redirect 301 / http://www.domain.com/
```

Redirect a specific file via 301:

```
# redirect a specific file via 301
redirect 301 /current/currentfile.html http://www.newdomain.com/new/newfile.html
```

Redirect an entire site via permanent redirect:

```
# redirect an entire site via permanent redirect
Redirect permanent / http://www.domain.com/
```

Redirect a page or directory via permanent redirect:

```
# redirect a page or directory
Redirect permanent old_file.html http://www.new-domain.com/new_file.html
Redirect permanent /old_directory/ http://www.new-domain.com/new_directory/
```

Redirect a file using RedirectMatch:

```
# redirect a file using RedirectMatch
RedirectMatch 301 ^.*$ http://www.domain.com/index.html
```

Note: When redirecting specific files, use Apache's `Redirect` rule for files within the same domain. Use Apache's `RewriteRule` for *any* domains, especially if they are different. The `RewriteRule` is more powerful than the `Redirect` rule, and thus should serve you more effectively.

Thus, use the following for a stronger, harder page redirection (first line redirects a file, second line a directory, and third a domain):

```
# redirect files directories and domains via RewriteRule
RewriteRule http://old-domain.com/old-file.html http://new-domain.com/new-file.html
RewriteRule http://old-domain.com/old-dir/ http://new-domain.com/new-dir/
RewriteRule http://old-domain.com/ http://new-domain.com/
```

Send visitors to a subdomain ^

This rule will ensure that all visitors are viewing pages via the subdomain of your choice. Edit the "subdomain", "domain", and "tld" to match your subdomain, domain, and top-level domain respectively:

```
# send visitors to a subdomain
RewriteCond %{HTTP_HOST} !^$
RewriteCond %{HTTP_HOST} !^subdomain\.domain\.com$ [NC]
RewriteRule ^/(.*)$ http://subdomain.domain.tld/$1 [L,R=301]
```

More fun with RewriteCond and RewriteRule ^

```
# rewrite only if the file is not found
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^(.+)special\.html?$ cgi-bin/special/special-html/$1

# rewrite only if an image is not found
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule images/special/(.*).gif cgi-bin/special/mkgif?$1

# seo-friendly rewrite rules for various directories
RewriteRule ^(.*)/aud/(.*)$ $1/audio-files/$2 [L,R=301]
RewriteRule ^(.*)/img/(.*)$ $1/image-files/$2 [L,R=301]
RewriteRule ^(.*)/fla/(.*)$ $1/flash-files/$2 [L,R=301]
RewriteRule ^(.*)/vid/(.*)$ $1/video-files/$2 [L,R=301]

# broswer sniffing via htaccess environmental variables
RewriteCond %{HTTP_USER_AGENT} ^Mozilla.*
RewriteRule ^/$ /index-for-mozilla.html [L]
RewriteCond %{HTTP_USER_AGENT} ^Lynx.*
RewriteRule ^/$ /index-for-lynx.html [L]
RewriteRule ^/$ /index-for-all-others.html [L]

# redirect query to Google search
```

```
Options +FollowSymlinks
RewriteEngine On
RewriteCond %{REQUEST_URI} .google\.php*
RewriteRule ^(.*)$ ^http://www.google.com/search?q=$1 [R,NC,L]

# deny request according to the request method
RewriteCond %{REQUEST_METHOD} ^(TRACE|TRACK|OPTIONS|HEAD)$ [NC]
RewriteRule ^.*$ - [F]

# redirect uploads to a better place
RewriteCond %{REQUEST_METHOD} ^(PUT|POST)$ [NC]
RewriteRule ^(.*)$ /cgi-bin/upload-processor.cgi?p=$1 [L,QSA]
```

More fun with Redirect 301 and RedirectMatch 301 ^

```
# seo friendly redirect for a single file
Redirect 301 /old-dir/old-file.html http://domain.com/new-dir/new-file.html

# seo friendly redirect for multiple files
# redirects all files in dir directory with first letters xyz
RedirectMatch 301 /dir/xyz(.*) http://domain.com/$1

# seo friendly redirect entire site to a different domain
Redirect 301 / http://different-domain.com
```

WordPress Tricks [^]

Secure WordPress Contact Forms ^

Protect your insecure WordPress contact forms against online unrighteousness by verifying the domain from whence the form is called. Remember to replace the “domain.com” and “contact.php” with *your* domain and contact-form file names, respectively.

```
# secure wordpress contact forms via referrer check
RewriteCond %{HTTP_REFERER} !^http://www.domain.com/.*$ [NC]
RewriteCond %{REQUEST_POST} .*contact.php$
RewriteRule .* - [F]
```

WordPress Permalinks ^

In our article, The htaccess rules for all WordPress Permalinks, we revealed the precise htaccess directives used by the WordPress blogging platform for permalink functionality. Here, for the sake of completeness, we repeat the directives only. For more details please refer to the original article:

If WordPress is installed in the site’s root directory, WordPress creates and uses the following htaccess directives:

```
# BEGIN WordPress
<IfModule mod_rewrite.c>
 RewriteEngine On
 RewriteBase /
 RewriteCond %{REQUEST_FILENAME} !-f
 RewriteCond %{REQUEST_FILENAME} !-d
 RewriteRule . /index.php [L]
</IfModule>
# END WordPress
```

If WordPress is installed in some subdirectory “foo”, WordPress creates and uses the following htaccess directives:

```
# BEGIN WordPress
<IfModule mod_rewrite.c>
 RewriteEngine On
 RewriteBase /foo/
 RewriteCond %{REQUEST_FILENAME} !-f
 RewriteCond %{REQUEST_FILENAME} !-d
 RewriteRule . /foo/index.php [L]
```

```
</IfModule>
# END WordPress
```

Random Tricks [^]

Activate SSI for HTML/SHTML file types: ^

```
# activate SSI for HTML and or SHTML file types
AddType text/html .html
AddType text/html .shtml
AddHandler server-parsed .html
AddHandler server-parsed .shtml
AddHandler server-parsed .htm
```

Grant CGI access in a specific directory: ^

```
# grant CGI access in a specific directory
Options +ExecCGI
AddHandler cgi-script cgi pl
# to enable all scripts in a directory use the following
SetHandler cgi-script
```

Disable magic_quotes_gpc for PHP enabled servers: ^

```
# turn off magic_quotes_gpc for PHP enabled servers
<ifmodule mod_php4.c>
 php_flag magic_quotes_gpc off
</ifmodule>
```

Enable MD5 digests: ^

Note: enabling this option may result in a relative decrease in server performance.

```
# enable MD5 digests via ContentDigest
ContentDigest On
```

Expression Engine Tricks: ^

```
# send Atom and RSS requests to the site docroot to be rewritten for ExpressionEngine
RewriteRule .*atom.xml$ http://www.yoursite.com/index.php/weblog/rss_atom/ [R]
RewriteRule .*rss.xml$ http://www.yoursite.com/index.php/weblog/rss_2.0/ [R]

# cause all requests for index.html to be rewritten for ExpressionEngine
RewriteRule /.*index.html$ http://www.domain.com/index.php [R]
```

References

- [1] Wikipedia htaccess Resource
- [2] Apache Cookbook
- [3] Ultimate htaccess Article
- More on regular expressions
- Apache htaccess Reference
- Apache htaccess Tutorial
- Apache mod_rewrite
- htaccess Forum
- Behind the Scenes with htaccess
- Automatic htaccess file generator

Related articles

- Stupid htaccess Tricks Redux
- HTAccess Password-Protection Tricks
- Stupid htaccess Trick: Enable File or Directory Access to Your Password-Protected Site
- Permanently Redirect a Specific IP Request for a Single Page via htaccess
- htaccess Combo Pack: WordPress Permalinks and non-www Redirect

- htaccess Redirect to Maintenance Page
- Stupid Twitter Tricks

Tags: apache, htaccess, mod_rewrite, notes, plus, tricks, upgrade, WordPress

← Previous Post

Hide Comments

84 Responses

Next Post →

- Full Feed, Delivered Fresh

Subscribe to Perishable Press

Join thousands of subscribers

- **Awesome Design Resources**
- **Learn how to WordPress**

Lead

June 9th, 2006 at 12:16 pm

Very nice... this is just the overview that pretty much says it all.... Great work !

Perishable

June 11th, 2006 at 1:55 pm

My pleasure! :)

Deep Singh

August 1st, 2006 at 8:56 am

Absolutely good contents really helpful

Perishable

August 1st, 2006 at 12:00 pm

At your service! In fact, I appreciate the kind remarks so much that I am planning to update this post with even more stupid htaccess tricks, including new redirect and blocking tips as well as some very useful character information.

m0n

August 21st, 2006 at 11:11 am

UPDATE: We have completely rewritten this entire article, which now includes almost twice as many stupid htaccess tricks, a nice library of regex character definitions, and even a handy table of contents for easy navigation. - Enjoy!

mattems

August 24th, 2006 at 3:49 pm

excellent!! just what i needed to get through those stupid htaccess issues !!

great work!

Perishable

August 27th, 2006 at 4:22 pm

You are too kind, mattems!

scandiman

September 4th, 2006 at 10:44 pm

Fantastic resource! Helped resolve many questions, but I remain totally stumped on one rewrite I am trying to accomplish.

Currently, I have a CMS that does Search Engine Friendly URLs using the following htaccess code:

```
RewriteEngine on
RewriteBase /
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d

RewriteRule ^(.*)$ /index.php?q=$1 [L, QSA]
```

A click to the home page will give me the URL "mydomain.com/index.html"

I am trying to rewrite that URL to just give me "mydomain.com/" Is this possible in the htaccess file considering the CMS is using it to rewrite the URL's as well? Or should I look into hacking the CMS?

Any help from an htaccess guru is appreciated!

Perishable

SEPTEMBER 6TH, 2006 AT 7:42 AM

Hei scandiman,

Have you tried:

```
# redirect from index.html to /
Options +FollowSymLinks
RewriteEngine on
RewriteCond %{THE_REQUEST} ^[A-Z]{3,9}\ /index\.html\
HTTP/
RewriteRule ^index\.html$ http://mydomain.com/ [R=301,L]
```

Employing the "THE_REQUEST" server variable, the pattern will match against the client request header only for the original URI request, thereby restricting redirection to the default index page.

H.Walther

NOVEMBER 23RD, 2006 AT 1:11 AM

This is a great article! Thank you very much!!!

Instead of disguiseing script extensions, what is the best way to completely remove it? Say `www.foo.com/foo.php` should be `www.foo.com/foo`?

Randy Gille

NOVEMBER 23RD, 2006 AT 12:35 PM

Email me here if you need any help with this:

`slayer1001@gmail.com`

Thanks!

Douglas Kemp

DECEMBER 5TH, 2006 AT 3:00 AM

Absolutely brilliat work mate, this is a fanstasic .htacess resource.

Gabe Trann

December 5th, 2006 at 6:13 am

Yes this is by far probably one of the best htaccess tuts.. nice work!

Oh heres another good one - http://www.askapache.com/2006/htaccess/htaccesselite-ultimate-htaccess-article.html.

Perishable

December 5th, 2006 at 8:09 am

Thank you both for your encouraging comments — they inspire us to expand further the scope of this htaccess reference. In fact, we have already written up several additional rules and examples, and are simply waiting for a break in the schedule to incorporate them here.

Happy Holidays!

Maria Langer

December 6th, 2006 at 12:54 am

Thanks so much for putting up this very easy to understand guide!

MrLeN

December 26th, 2006 at 3:25 am

Is it possible to redirect people from a particular country? ie: I might want people who live in china to be recognised and automatically redirected to a chinese page, or people who libe in germany to be automatically redirected to a german page.

Perishable

December 27th, 2006 at 9:47 am

MrLeN,
Yes, of course.. it is possible to execute country-specific redirects. If you have GeoIP Apache API configured on your server, simply extrapolate the following:

```
RewriteEngine on
RewriteCond %{ENV:GEOIP_COUNTRY_CODE} MX [NC]
RewriteRule ^$ index.mx.php [L]
```

This method employs country codes (we used Mexico in the example).

Of course, you could also redirect based on country-specific TLD or even IP addresses.

Bad Monkey

December 28th, 2006 at 8:08 pm

I am trying to deny from a particular host, and for a time I even want to deny an entire top-level domain.
E.g. deny from .nz
(new zealand)
But it does not work. Access is still allowed.
Similarly deny from myhost.ext does not work.

It **does** work with my IP.
Any ideas?

Perishable

December 30th, 2006 at 9:24 am

Bad Monkey,
I need to see the htaccess code you are using.. htaccess rules are super-sensitive — many problems are solved by simply correcting subtle errors in syntax, order, etc.

Bad Monkey

December 30th, 2006 at 5:13 pm

Simply one line...
`deny from 124.197.12.164`
works, that is my IP and it bans me, but
`deny from .nz`
or
`deny from .callplus.net.nz`
(my host) does not.
Reverse DNS on my ip gives
124-197-12-164.callplus.net.nz
so what am I missing?

Perishable

January 1st, 2007 at 11:20 am

You may want to try it within the following context:

```
<Limit GET POST>
order allow,deny
allow from all
deny from .nz
deny from .callplus.net.nz
</Limit>
```

If that still does not work, you might try a RewriteRule instead:

Top