Apache HTTP Server Version 2.4

RewriteRule Flags

This document discusses the flags which are available to the `RewriteRule` directive, providing detailed explanations and examples.

Introduction

A `RewriteRule` can have its behavior modified by one or more flags. Flags are included in square brackets at the end of the rule, and multiple flags are separated by commas.

```
RewriteRule pattern target [Flag1,Flag2,Flag3]
```

Each flag (with a few exceptions) has a short form, such as `CO`, as well as a longer form, such as `cookie`. While it is most common to use the short form, it is recommended that you familiarize yourself with the long form, so that you remember what each flag is supposed to do. Some flags take one or more arguments. Flags are not case sensitive.

Flags that alter metadata associated with the request (T=, H=, E=) have no affect in per-directory and htaccess context, when a substitution (other than '-') is performed during the same round of rewrite processing.

Presented here are each of the available flags, along with an example of how you might use them.

B (escape backreferences)

The [B] flag instructs `RewriteRule` to escape non-alphanumeric characters before applying the transformation.

`mod_rewrite` has to unescape URLs before mapping them, so backreferences are unescaped at the time they are applied. Using the B flag, non-alphanumeric characters in backreferences will be escaped. For example, consider the rule:

```
RewriteRule "^search/(.*)$" "/search.php?term=$1"
```

Given a search term of 'x & y/z', a browser will encode it as 'x%20%26%20y%2Fz', making the request 'search/x%20%26%20y%2Fz'. Without the B flag, this rewrite rule will map to 'search.php?term=x & y/z', which isn't a valid URL, and so would be encoded as `search.php?term=x%20&y%2Fz=`, which is not what was intended.

With the B flag set on this same rule, the parameters are re-encoded before being passed on to the output URL, resulting in a correct mapping to `/search.php?term=x%20%26%20y%2Fz`.

```
RewriteRule "^search/(.*)$" "/search.php?term=$1" [B,PT]
```

Note that you may also need to set `AllowEncodedSlashes` to `On` to get this particular example to work, as httpd does not allow encoded slashes in URLs, and returns a 404 if it sees one.

This escaping is particularly necessary in a proxy situation, when the backend may break if presented with an unescaped URL.

An alternative to this flag is using a `RewriteCond` to capture against %{THE_REQUEST} which will capture strings in the encoded form.

In 2.4.26 and later, you can limit the escaping to specific characters in backreferences by listing them: `[B=#?;]`. Note: The space character can be used in the list of characters to escape, but you must quote the entire third argument of `RewriteRule` and the space must not be the last character in the list.

```
# Escape spaces and question marks.
RewriteRule "^search/(.*)$" "/search.php?term=$1" "[B= ?]"
```

BNP|backrefnoplus (don't escape space to +)

The [BNP] flag instructs `RewriteRule` to escape the space character in a backreference to %20 rather than '+'. Useful when the backreference will be used in the path component rather than the query string.

This flag is available in version 2.4.26 and later.

C|chain

The [C] or [chain] flag indicates that the `RewriteRule` is chained to the next rule. That is, if the rule matches, then it is processed as usual and control moves on to the next rule. However, if it does not match, then the next rule, and any other rules that are chained together, are skipped.

CO|cookie

The [CO], or [cookie] flag, allows you to set a cookie when a particular `RewriteRule` matches. The argument consists of three required fields and five optional fields.

The full syntax for the flag, including all attributes, is as follows:

```
[CO=NAME:VALUE:DOMAIN:lifetime:path:secure:httponly:samesite]
```

If a literal ':' character is needed in any of the cookie fields, an alternate syntax is available. To opt-in to the alternate syntax, the cookie "Name" should be preceded with a ';' character, and field separators should be specified as ';'.

```
[CO=;NAME;VALUE:MOREVALUE;DOMAIN;lifetime;path;secure;httponly;samesite]
```

You must declare a name, a value, and a domain for the cookie to be set.

Domain
: The domain for which you want the cookie to be valid. This may be a hostname, such as `www.example.com`, or it may be a domain, such as `.example.com`. It must be at least two parts separated by a dot. That is, it may not be merely `.com` or `.net`. Cookies of that kind are forbidden by the cookie security model.

You may optionally also set the following values:

Lifetime
The time for which the cookie will persist, in minutes.
A value of 0 indicates that the cookie will persist only for the current browser session. This is the default value if none is specified.

Path
The path, on the current website, for which the cookie is valid, such as `/customers/` or `/files/download/`.
By default, this is set to `/` - that is, the entire website.

Secure
If set to `secure`, `true`, or `1`, the cookie will only be permitted to be translated via secure (https) connections.

httponly
If set to `HttpOnly`, `true`, or `1`, the cookie will have the `HttpOnly` flag set, which means that the cookie is inaccessible to JavaScript code on browsers that support this feature.

samesite
If set to anything other than `false` or `0`, the `SameSite` attribute is set to the specified value. Typical values are `None`, `Lax`, and `Strict`. Available in 2.4.47 and later.

Consider this example:

```
RewriteEngine On
RewriteRule "^/index\.html" "-" [CO=frontdoor:yes:.example.com:1440:/]
```

In the example give, the rule doesn't rewrite the request. The "-" rewrite target tells mod_rewrite to pass the request through unchanged. Instead, it sets a cookie called 'frontdoor' to a value of 'yes'. The cookie is valid for any host in the `.example.com` domain. It is set to expire in 1440 minutes (24 hours) and is returned for all URIs.

DPI|discardpath

The DPI flag causes the PATH_INFO portion of the rewritten URI to be discarded.

This flag is available in version 2.2.12 and later.

In per-directory context, the URI each `RewriteRule` compares against is the concatenation of the current values of the URI and PATH_INFO.

The current URI can be the initial URI as requested by the client, the result of a previous round of mod_rewrite processing, or the result of a prior rule in the current round of mod_rewrite processing.

In contrast, the PATH_INFO that is appended to the URI before each rule reflects only the value of PATH_INFO before this round of mod_rewrite processing. As a consequence, if large portions of the URI are matched and copied into a substitution in multiple `RewriteRule` directives, without regard for which parts of the URI came from the current PATH_INFO, the final URI may have multiple copies of PATH_INFO appended to it.

Use this flag on any substitution where the PATH_INFO that resulted from the previous mapping of this request to the filesystem is not of interest. This flag permanently forgets the PATH_INFO established before this round of mod_rewrite processing began. PATH_INFO will not be recalculated until the current round of mod_rewrite processing completes. Subsequent rules during this round of processing will see only the direct result of substitutions, without any PATH_INFO appended.

E|env

With the [E], or [env] flag, you can set the value of an environment variable. Note that some environment variables may be set after the rule is run, thus unsetting what you have set. See the Environment Variables document (↗ ../env.html) for more details on how Environment variables work.

The full syntax for this flag is:

```
[E=VAR:VAL]
[E=!VAR]
```

`VAL` may contain backreferences (`$N` or `%N`) which are expanded.

Using the short form

```
[E=VAR]
```

you can set the environment variable named `VAR` to an empty value.

The form

```
[E=!VAR]
```

allows to unset a previously set environment variable named `VAR`.

Environment variables can then be used in a variety of contexts, including CGI programs, other RewriteRule directives, or CustomLog directives.

The following example sets an environment variable called 'image' to a value of '1' if the requested URI is an image file. Then, that environment variable is used to exclude those requests from the access log.

```
RewriteRule "\.(png|gif|jpg)$" "-" [E=image:1]
CustomLog "logs/access_log" combined env=!image
```

Note that this same effect can be obtained using `SetEnvIf`. This technique is offered as an example, not as a recommendation.

END

Using the [END] flag terminates not only the current round of rewrite processing (like [L]) but also prevents any subsequent rewrite processing from occurring in per-directory (htaccess) context.

This does not apply to new requests resulting from external redirects.

F|forbidden

Using the [F] flag causes the server to return a 403 Forbidden status code to the client. While the same behavior can be accomplished using the `Deny` directive, this allows more flexibility in assigning a Forbidden status.

The following rule will forbid `.exe` files from being downloaded from your server.

```
RewriteRule "\.exe" "-" [F]
```

This example uses the "-" syntax for the rewrite target, which means that the requested URI is not modified. There's no reason to rewrite to another URI, if you're going to forbid the request.

When using [F], an [L] is implied - that is, the response is returned immediately, and no further rules are evaluated.

G|gone

The [G] flag forces the server to return a 410 Gone status with the response. This indicates that a resource used to be available, but is no longer available.

As with the [F] flag, you will typically use the "-" syntax for the rewrite target when using the [G] flag:

```
RewriteRule "oldproduct" "-" [G,NC]
```

When using [G], an [L] is implied - that is, the response is returned immediately, and no further rules are evaluated.

H|handler

Forces the resulting request to be handled with the specified handler. For example, one might use this to force all files without a file extension to be parsed by the php handler:

```
RewriteRule "!\." "-" [H=application/x-httpd-php]
```

The regular expression above - `!\.` - will match any request that does not contain the literal `.` character.

This can be also used to force the handler based on some conditions. For example, the following snippet used in per-server context allows `.php` files to be *displayed* by `mod_php` if they are requested with the `.phps` extension:

```
RewriteRule "^(/source/.+\.php)s$" "$1" [H=application/x-httpd-php-source]
```

The regular expression above - `^(/source/.+\.php)s$` - will match any request that starts with `/source/` followed by 1 or n characters followed by `.phps` literally. The backreference $1 referrers to the captured match within parenthesis of the regular expression.

L|last

The [L] flag causes `mod_rewrite` to stop processing the rule set. In most contexts, this means that if the rule matches, no further rules will be processed. This corresponds to the `last` command in Perl, or the `break` command in C. Use this flag to indicate that the current rule should be applied immediately without considering further rules.

If you are using `RewriteRule` in either `.htaccess` files or in `<Directory>` sections, it is important to have some understanding of how the rules are processed. The simplified form of this is that once the rules have been processed, the rewritten request is handed back to the URL parsing engine to do what it may with it. It is possible that as the rewritten request is handled, the `.htaccess` file or `<Directory>` section may be encountered again, and thus the ruleset may be run again from the start. Most commonly this will happen if one of the rules causes a redirect - either internal or external - causing the request process to start over.

It is therefore important, if you are using `RewriteRule` directives in one of these contexts, that you take explicit steps to avoid rules looping, and not count solely on the [L] flag to terminate execution of a series of rules, as shown below.

An alternative flag, [END], can be used to terminate not only the current round of rewrite processing but prevent any subsequent rewrite processing from occurring in per-directory (htaccess) context. This does not apply to new requests resulting from external redirects.

The example given here will rewrite any request to `index.php`, giving the original request as a query string argument to `index.php`, however, the `RewriteCond` ensures that if the request is already for `index.php`, the `RewriteRule` will be skipped.

```
RewriteBase "/"
RewriteCond "%{REQUEST_URI}" "!=/index.php"
RewriteRule "^(.*)" "/index.php?req=$1" [L,PT]
```

N|next

The [N] flag causes the ruleset to start over again from the top, using the result of the ruleset so far as a starting point. Use with extreme caution, as it may result in loop.

The [Next] flag could be used, for example, if you wished to replace a certain string or letter repeatedly in a request. The example shown here will replace A with B everywhere in a request, and will continue doing so until there are no more As to be replaced.

```
RewriteRule "(.*)A(.*)" "$1B$2" [N]
```

You can think of this as a `while` loop: While this pattern still matches (i.e., while the URI still contains an `A`), perform this substitution (i.e., replace the `A` with a `B`).

In 2.4.8 and later, this module returns an error after 32,000 iterations to protect against unintended looping. An alternative maximum number of iterations can be specified by adding to the N flag.

```
# Be willing to replace 1 character in each pass of the loop
RewriteRule "(.+)[><;]$" "$1" [N=64000]
# ... or, give up if after 10 loops
RewriteRule "(.+)[><;]$" "$1" [N=10]
```

NC|nocase

Use of the [NC] flag causes the `RewriteRule` to be matched in a case-insensitive manner. That is, it doesn't care whether letters appear as upper-case or lower-case in the matched URI.

In the example below, any request for an image file will be proxied to your dedicated image server. The match is case-insensitive, so that `.jpg` and `.JPG` files are both acceptable, for example.

```
RewriteRule "(.*\.(jpg|gif|png))$" "http://images.example.com$1" [P,NC]
```

NE|noescape

By default, special characters, such as & and ?, for example, will be converted to their hexcode equivalent for rules that result in external redirects. Using the [NE] flag prevents that from happening.

```
RewriteRule "^/anchor/(.+)" "/bigpage.html#$1" [NE,R]
```

The above example will redirect `/anchor/xyz` to `/bigpage.html#xyz`. Omitting the [NE] will result in the # being converted to its hexcode equivalent, `%23`, which will then result in a 404 Not Found error condition.

NS|nosubreq

Use of the [NS] flag prevents the rule from being used on subrequests. For example, a page which is included using an SSI (Server Side Include) is a subrequest, and you may want to avoid rewrites happening on those subrequests. Also, when `mod_dir` tries to find out information about possible directory default files (such as `index.html` files), this is an internal subrequest, and you often want to avoid rewrites on such subrequests. On subrequests, it is not always useful, and can even cause errors, if the complete set of rules are applied. Use this flag to exclude problematic rules.

To decide whether or not to use this rule: if you prefix URLs with CGI-scripts, to force them to be processed by the CGI-script, it's likely that you will run into problems (or significant overhead) on sub-requests. In these cases, use this flag.

Images, javascript files, or css files, loaded as part of an HTML page, are not subrequests - the browser requests them as separate HTTP requests.

P|proxy

Use of the [P] flag causes the request to be handled by `mod_proxy`, and handled via a proxy request. For example, if you wanted all image requests to be handled by a back-end image server, you might do something like the following:

```
RewriteRule "/(.*)\.(jpg|gif|png)$" "http://images.example.com/$1.$2" [P]
```

Use of the [P] flag implies [L] - that is, the request is immediately pushed through the proxy, and any following rules will not be considered.

You must make sure that the substitution string is a valid URI (typically starting with `http://`*hostname*) which can be handled by the `mod_proxy`. If not, you will get an error from the proxy module. Use this flag to achieve a more powerful implementation of the `ProxyPass` directive, to map remote content into the namespace of the local server.

Security Warning

Take care when constructing the target URL of the rule, considering the security impact from allowing the client influence over the set of URLs to which your server will act as a proxy. Ensure that the scheme and hostname part of the URL is either fixed, or does not allow the client undue influence.

Performance warning

Using this flag triggers the use of `mod_proxy`, without handling of persistent connections. This means the performance of your proxy will be better if you set it up with `ProxyPass` or `ProxyPassMatch`
This is because this flag triggers the use of the default worker, which does not handle connection pooling/reuse.
Avoid using this flag and prefer those directives, whenever you can.

Note: `mod_proxy` must be enabled in order to use this flag.

PT|passthrough

The target (or substitution string) in a RewriteRule is assumed to be a file path, by default. The use of the [PT] flag causes it to be treated as a URI instead. That is to say, the use of the [PT] flag causes the result of the `RewriteRule` to be passed back through URL mapping, so that location-based mappings, such as `Alias`, `Redirect`, or `ScriptAlias`, for example, might have a chance to take effect.

If, for example, you have an `Alias` for /icons, and have a `RewriteRule` pointing there, you should use the [PT] flag to ensure that the `Alias` is evaluated.

```
Alias "/icons" "/usr/local/apache/icons"
RewriteRule "/pics/(.+)\.jpg$" "/icons/$1.gif" [PT]
```

Omission of the [PT] flag in this case will cause the Alias to be ignored, resulting in a 'File not found' error being returned.

The `PT` flag implies the `L` flag: rewriting will be stopped in order to pass the request to the next phase of processing.

Note that the `PT` flag is implied in per-directory contexts such as `<Directory>` sections or in `.htaccess` files. The only way to circumvent that is to rewrite to `-`.

QSA|qsappend

When the replacement URI contains a query string, the default behavior of `RewriteRule` is to discard the existing query string, and replace it with the newly generated one. Using the [QSA] flag causes the query strings to be combined.

Consider the following rule:

```
RewriteRule "/pages/(.+)" "/page.php?page=$1" [QSA]
```

With the [QSA] flag, a request for `/pages/123?one=two` will be mapped to `/page.php?page=123&one=two`. Without the [QSA] flag, that same request will be mapped to `/page.php?page=123` - that is, the existing query string will be discarded.

QSD|qsdiscard

When the requested URI contains a query string, and the target URI does not, the default behavior of `RewriteRule` is to copy that query string to the target URI. Using the [QSD] flag causes the query string to be discarded.

This flag is available in version 2.4.0 and later.

Using [QSD] and [QSA] together will result in [QSD] taking precedence.

If the target URI has a query string, the default behavior will be observed - that is, the original query string will be discarded and replaced with the query string in the `RewriteRule` target URI.

QSL|qslast

By default, the first (left-most) question mark in the substitution delimits the path from the query string. Using the [QSL] flag instructs `RewriteRule` to instead split the two components using the last (right-most) question mark.

This is useful when mapping to files that have literal question marks in their filename. If no query string is used in the substitution, a question mark can be appended to it in combination with this flag.

This flag is available in version 2.4.19 and later.

R|redirect

Use of the [R] flag causes a HTTP redirect to be issued to the browser. If a fully-qualified URL is specified (that is, including `http://servername/`) then a redirect will be issued to that location. Otherwise, the current protocol, servername, and port number will be used to generate the URL sent with the redirect.

Any valid HTTP response status code may be specified, using the syntax [R=305], with a 302 status code being used by default if none is specified. The status code specified need not necessarily be a redirect (3xx) status code. However, if a status code is outside the redirect range (300-399) then the substitution string is dropped entirely, and rewriting is stopped as if the `L` were used.

In addition to response status codes, you may also specify redirect status using their symbolic names: `temp` (default), `permanent`, or `seeother`.

You will almost always want to use [R] in conjunction with [L] (that is, use [R,L]) because on its own, the [R] flag prepends `http://thishost[:thisport]` to the URI, but then passes this on to the next rule in the ruleset, which can often result in 'Invalid URI in request' warnings.

S|skip

The [S] flag is used to skip rules that you don't want to run. The syntax of the skip flag is [S=*N*], where *N* signifies the number of rules to skip (provided the `RewriteRule` matches). This can be thought of as a `goto` statement in your rewrite ruleset. In the following example, we only want to run the `RewriteRule` if the requested URI doesn't correspond with an actual file.

```
# Is the request for a non-existent file?
RewriteCond "%{REQUEST_FILENAME}" "!-f"
RewriteCond "%{REQUEST_FILENAME}" "!-d"
# If so, skip these two RewriteRules
RewriteRule ".?" "-" [S=2]

RewriteRule "(.*\.gif)" "images.php?$1"
RewriteRule "(.*\.html)" "docs.php?$1"
```

This technique is useful because a `RewriteCond` only applies to the `RewriteRule` immediately following it. Thus, if you want to make a `RewriteCond` apply to several `RewriteRule`s, one possible technique is to negate those conditions and add a `RewriteRule` with a [Skip] flag. You can use this to make pseudo if-then-else constructs: The last rule of the then-clause becomes `skip=N`, where N is the number of rules in the else-clause:

```
# Does the file exist?
RewriteCond "%{REQUEST_FILENAME}" "!-f"
RewriteCond "%{REQUEST_FILENAME}" "!-d"
# Create an if-then-else construct by skipping 3 lines if we meant to go to the "else" stanza.
RewriteRule ".?" "-" [S=3]

# IF the file exists, then:
    RewriteRule "(.*\.gif)" "images.php?$1"
    RewriteRule "(.*\.html)" "docs.php?$1"
    # Skip past the "else" stanza.
    RewriteRule ".?" "-" [S=1]
# ELSE...
    RewriteRule "(.*)" "404.php?file=$1"
# END
```

It is probably easier to accomplish this kind of configuration using the `<If>`, `<ElseIf>`, and `<Else>` directives instead.

T|type

Sets the MIME type with which the resulting response will be sent. This has the same effect as the `AddType` directive.

For example, you might use the following technique to serve Perl source code as plain text, if requested in a particular way:

```
# Serve .pl files as plain text
RewriteRule "\.pl$" "-" [T=text/plain]
```

Or, perhaps, if you have a camera that produces jpeg images without file extensions, you could force those images to be served with the correct MIME type by virtue of their file names:

```
# Files with 'IMG' in the name are jpg images.
RewriteRule "IMG" "-" [T=image/jpg]
```

Please note that this is a trivial example, and could be better done using `<FilesMatch>` instead. Always consider the alternate solutions to a problem before resorting to rewrite, which will invariably be a less efficient solution than the alternatives.

If used in per-directory context, use only `-` (dash) as the substitution *for the entire round of mod_rewrite processing*, otherwise the MIME-type set with this flag is lost due to an internal re-processing (including subsequent rounds of mod_rewrite processing). The `L` flag can be useful in this context to end the *current* round of mod_rewrite processing.

Comments