Apache HTTP Server Version 2.4

Apache Module mod_rewrite

Description:	Provides a rule-based rewriting engine to rewrite requested URLs on the fly
Status:	Extension
Module Identifier:	rewrite_module
Source File:	mod_rewrite.c

Summary

The `mod_rewrite` module uses a rule-based rewriting engine, based on a PCRE regular-expression parser, to rewrite requested URLs on the fly. By default, `mod_rewrite` maps a URL to a filesystem path. However, it can also be used to redirect one URL to another URL, or to invoke an internal proxy fetch.

`mod_rewrite` provides a flexible and powerful way to manipulate URLs using an unlimited number of rules. Each rule can have an unlimited number of attached rule conditions, to allow you to rewrite URL based on server variables, environment variables, HTTP headers, or time stamps.

`mod_rewrite` operates on the full URL path, including the path-info section. A rewrite rule can be invoked in `httpd.conf` or in `.htaccess`. The path generated by a rewrite rule can include a query string, or can lead to internal sub-processing, external request redirection, or internal proxy throughput.

Further details, discussion, and examples, are provided in the detailed mod_rewrite documentation (↗ ../rewrite/) .

Logging

`mod_rewrite` offers detailed logging of its actions at the `trace1` to `trace8` log levels. The log level can be set specifically for `mod_rewrite` using the `LogLevel` directive: Up to level `debug`, no actions are logged, while `trace8` means that practically all actions are logged.

Using a high trace log level for `mod_rewrite` will slow down your Apache HTTP Server dramatically! Use a log level higher than `trace2` only for debugging!

Example

```
LogLevel alert rewrite:trace3
```

RewriteLog

Those familiar with earlier versions of `mod_rewrite` will no doubt be looking for the `RewriteLog` and `RewriteLogLevel` directives. This functionality has been completely replaced by the new per-module logging configuration mentioned above.
To get just the `mod_rewrite`-specific log messages, pipe the log file through grep:

```
tail -f error_log|fgrep '[rewrite:'
```

RewriteBase Directive

Description:	Sets the base URL for per-directory rewrites
Syntax:	`RewriteBase` *`URL-path`*
Default:	`None`
Context:	directory, .htaccess
Override:	FileInfo
Status:	Extension
Module:	mod_rewrite

The `RewriteBase` directive specifies the URL prefix to be used for per-directory (htaccess) `RewriteRule` directives that substitute a relative path.

This directive is *required* when you use a relative path in a substitution in per-directory (htaccess) context unless any of the following conditions are true:

- The original request, and the substitution, are underneath the `DocumentRoot` (as opposed to reachable by other means, such as `Alias`).
- The *filesystem* path to the directory containing the `RewriteRule`, suffixed by the relative substitution is also valid as a URL path on the server (this is rare).
- In Apache HTTP Server 2.4.16 and later, this directive may be omitted when the request is mapped via `Alias` or `mod_userdir`.

In the example below, `RewriteBase` is necessary to avoid rewriting to http://example.com/opt/myapp-1.2.3/welcome.html since the resource was not relative to the document root. This misconfiguration would normally cause the server to look for an "opt" directory under the document root.

```
DocumentRoot "/var/www/example.com"
AliasMatch "^/myapp" "/opt/myapp-1.2.3"
<Directory "/opt/myapp-1.2.3">
    RewriteEngine On
    RewriteBase "/myapp/"
    RewriteRule "^index\.html$"  "welcome.html"
</Directory>
```

RewriteCond Directive

Description:	Defines a condition under which rewriting will take place
Syntax:	`RewriteCond` *`TestString CondPattern`* `[`*`flags`*`]`
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Extension
Module:	mod_rewrite

The `RewriteCond` directive defines a rule condition. One or more `RewriteCond` can precede a `RewriteRule` directive. The following rule is then only used if both the current state of the URI matches its pattern, **and** if these conditions are met.

TestString is a string which can contain the following expanded constructs in addition to plain text:

- **RewriteRule backreferences**: These are backreferences of the form **`$N`** (0 <= N <= 9). $1 to $9 provide access to the grouped parts (in parentheses) of the pattern, from the `RewriteRule` which is subject to the current set of `RewriteCond` conditions. $0 provides access to the whole string matched by that pattern.
- **RewriteCond backreferences**: These are backreferences of the form **`%N`** (0 <= N <= 9). %1 to %9 provide access to the grouped parts (again, in parentheses) of the pattern, from the last matched `RewriteCond` in the current set of conditions. %0 provides access to the whole string matched by that pattern.
- **RewriteMap expansions**: These are expansions of the form **`${mapname:key|default}`**. See the documentation for RewriteMap for more details.
- **Server-Variables**: These are variables of the form **`%{` *NAME_OF_VARIABLE* `}`** where *NAME_OF_VARIABLE* can be a string taken from the following list:

HTTP headers:	**connection & request:**	
HTTP_ACCEPT	AUTH_TYPE	
HTTP_COOKIE	CONN_REMOTE_ADDR	
HTTP_FORWARDED	CONTEXT_PREFIX	
HTTP_HOST	CONTEXT_DOCUMENT_ROOT	
HTTP_PROXY_CONNECTION	IPV6	
HTTP_REFERER	PATH_INFO	
HTTP_USER_AGENT	QUERY_STRING	
	REMOTE_ADDR	
	REMOTE_HOST	
	REMOTE_IDENT	
	REMOTE_PORT	
	REMOTE_USER	
	REQUEST_METHOD	
	SCRIPT_FILENAME	
server internals:	**date and time:**	**specials:**
DOCUMENT_ROOT	TIME_YEAR	API_VERSION
SCRIPT_GROUP	TIME_MON	CONN_REMOTE_ADDR
SCRIPT_USER	TIME_DAY	HTTPS
SERVER_ADDR	TIME_HOUR	IS_SUBREQ
SERVER_ADMIN	TIME_MIN	REMOTE_ADDR
SERVER_NAME	TIME_SEC	REQUEST_FILENAME
SERVER_PORT	TIME_WDAY	REQUEST_SCHEME
SERVER_PROTOCOL	TIME	REQUEST_URI
SERVER_SOFTWARE		THE_REQUEST

These variables all correspond to the similarly named HTTP MIME-headers, C variables of the Apache HTTP Server or `struct tm` fields of the Unix system. Most are documented here (↗ ../expr.html#vars) or elsewhere in the Manual or in the CGI specification.

SERVER_NAME and SERVER_PORT depend on the values of `UseCanonicalName` and `UseCanonicalPhysicalPort` respectively.

Those that are special to `mod_rewrite` include those below.

`API_VERSION`
: This is the version of the Apache httpd module API (the internal interface between server and module) in the current httpd build, as defined in include/ap_mmn.h. The module API version corresponds to the version of Apache httpd in use (in the release version of Apache httpd 1.3.14, for instance, it is 19990320:10), but is mainly of interest to module authors.

`CONN_REMOTE_ADDR`
: Since 2.4.8: The peer IP address of the connection (see the `mod_remoteip` module).

`HTTPS`
: Will contain the text "on" if the connection is using SSL/TLS, or "off" otherwise. (This variable can be safely used regardless of whether or not `mod_ssl` is loaded).

`IS_SUBREQ`
: Will contain the text "true" if the request currently being processed is a sub-request, "false" otherwise. Sub-requests may be generated by modules that need to resolve additional files or URIs in order to complete their tasks.

`REMOTE_ADDR`
: The IP address of the remote host (see the `mod_remoteip` module).

`REQUEST_FILENAME`
: The full local filesystem path to the file or script matching the request, if this has already been determined by the server at the time `REQUEST_FILENAME` is referenced. Otherwise, such as when used in virtual host context, the same value as `REQUEST_URI`. Depending on the value of `AcceptPathInfo`, the server may have only used some leading components of the `REQUEST_URI` to map the request to a file.

`REQUEST_SCHEME`
: Will contain the scheme of the request (usually "http" or "https"). This value can be influenced with `ServerName`.

`REQUEST_URI`
: The path component of the requested URI, such as "/index.html". This notably excludes the query string which is available as its own variable named `QUERY_STRING`.

`THE_REQUEST`
: The full HTTP request line sent by the browser to the server (e.g., "`GET /index.html HTTP/1.1`"). This does not include any additional headers sent by the browser. This value has not been unescaped (decoded), unlike most other variables below.

If the *TestString* has the special value `expr`, the *CondPattern* will be treated as an ap_expr (↗ ../expr.html) . HTTP headers referenced in the expression will be added to the Vary header if the `novary` flag is not given.

Other things you should be aware of:

1. The variables SCRIPT_FILENAME and REQUEST_FILENAME contain the same value - the value of the `filename` field of the internal `request_rec` structure of the Apache HTTP Server. The first name is the commonly known CGI variable name while the second is the appropriate counterpart of REQUEST_URI (which contains the value of the `uri` field of `request_rec`).

   If a substitution occurred and the rewriting continues, the value of both variables will be updated accordingly.

   If used in per-server context (*i.e.*, before the request is mapped to the filesystem) SCRIPT_FILENAME and REQUEST_FILENAME cannot contain the full local filesystem path since the path is unknown at this stage of processing. Both variables will initially contain the value of REQUEST_URI in that case. In order to obtain the full local filesystem path of the request in per-server context, use an URL-based look-ahead `%{LA-U:REQUEST_FILENAME}` to determine the final value of REQUEST_FILENAME.

2. `%{ENV:variable}`, where *variable* can be any environment variable, is also available. This is looked-up via internal Apache httpd structures and (if not found there) via `getenv()` from the Apache httpd server process.
3. `%{SSL:variable}`, where *variable* is the name of an SSL environment variable, can be used whether or not `mod_ssl` is loaded, but will always expand to the empty string if it is not. Example: `%{SSL:SSL_CIPHER_USEKEYSIZE}` may expand to `128`. These variables are available even without setting the `StdEnvVars` option of the `SSLOptions` directive.

4. `%{HTTP:header}`, where *header* can be any HTTP MIME-header name, can always be used to obtain the value of a header sent in the HTTP request. Example: `%{HTTP:Proxy-Connection}` is the value of the HTTP header ``Proxy-Connection:".
   If a HTTP header is used in a condition this header is added to the Vary header of the response in case the condition evaluates to true for the request. It is **not** added if the condition evaluates to false for the request. Adding the HTTP header to the Vary header of the response is needed for proper caching.

   It has to be kept in mind that conditions follow a short circuit logic in the case of the '**`ornext|OR`**' flag so that certain conditions might not be evaluated at all.

5. `%{LA-U:variable}` can be used for look-aheads which perform an internal (URL-based) sub-request to determine the final value of *variable*. This can be used to access variable for rewriting which is not available at the current stage, but will be set in a later phase.
   For instance, to rewrite according to the `REMOTE_USER` variable from within the per-server context (`httpd.conf` file) you must use `%{LA-U:REMOTE_USER}` - this variable is set by the authorization phases, which come *after* the URL translation phase (during which `mod_rewrite` operates).

   On the other hand, because `mod_rewrite` implements its per-directory context (`.htaccess` file) via the Fixup phase of the API and because the authorization phases come *before* this phase, you just can use `%{REMOTE_USER}` in that context.

6. `%{LA-F:variable}` can be used to perform an internal (filename-based) sub-request, to determine the final value of *variable*. Most of the time, this is the same as LA-U above.

CondPattern is the condition pattern, a regular expression which is applied to the current instance of the *TestString*. *TestString* is first evaluated, before being matched against *CondPattern*.

CondPattern is usually a *perl compatible regular expression*, but there is additional syntax available to perform other useful tests against the *Teststring*:

1. You can prefix the pattern string with a '`!`' character (exclamation mark) to negate the result of the condition, no matter what kind of *CondPattern* is used.
2. You can perform lexicographical string comparisons:

   <CondPattern
   : Lexicographically precedes
   Treats the *CondPattern* as a plain string and compares it lexicographically to *TestString*. True if *TestString* lexicographically precedes *CondPattern*.

   >CondPattern
   : Lexicographically follows
   Treats the *CondPattern* as a plain string and compares it lexicographically to *TestString*. True if *TestString* lexicographically follows *CondPattern*.

   =CondPattern
   : Lexicographically equal
   Treats the *CondPattern* as a plain string and compares it lexicographically to *TestString*. True if *TestString* is lexicographically equal to *CondPattern* (the two strings are exactly equal, character for character). If *CondPattern* is `""` (two quotation marks) this compares *TestString* to the empty string.

   <=CondPattern
   : Lexicographically less than or equal to
   Treats the *CondPattern* as a plain string and compares it lexicographically to *TestString*. True if *TestString* lexicographically precedes *CondPattern*, or is equal to *CondPattern* (the two strings are equal, character for character).

   >=CondPattern
   : Lexicographically greater than or equal to
   Treats the *CondPattern* as a plain string and compares it lexicographically to *TestString*. True if *TestString* lexicographically follows *CondPattern*, or is equal to *CondPattern* (the two strings are equal, character for character).

   > **Note**
   >
   > The string comparison operator is part of the *CondPattern* argument and must be included in the quotes if those are used. Eg.
   >
   > ```
   > RewriteCond %{HTTP_USER_AGENT} "=This Robot/1.0"
   > ```

3. You can perform integer comparisons:

   -eq
   : Is numerically **eq**ual to
   The *TestString* is treated as an integer, and is numerically compared to the *CondPattern*. True if the two are numerically equal.

   -ge
   : Is numerically **g**reater than or **e**qual to
   The *TestString* is treated as an integer, and is numerically compared to the *CondPattern*. True if the *TestString* is numerically greater than or equal to the *CondPattern*.

   -gt
   : Is numerically **g**reater **t**han
   The *TestString* is treated as an integer, and is numerically compared to the *CondPattern*. True if the *TestString* is numerically greater than the *CondPattern*.

   -le
   : Is numerically **l**ess than or **e**qual to
   The *TestString* is treated as an integer, and is numerically compared to the *CondPattern*. True if the *TestString* is numerically less than or equal to the *CondPattern*. Avoid confusion with the **-l** by using the **-L** or **-h** variant.

   -lt
   : Is numerically **l**ess **t**han
   The *TestString* is treated as an integer, and is numerically compared to the *CondPattern*. True if the *TestString* is numerically less than the *CondPattern*. Avoid confusion with the **-l** by using the **-L** or **-h** variant.

   -ne
   : Is numerically **n**ot **e**qual to
   The *TestString* is treated as an integer, and is numerically compared to the *CondPattern*. True if the two are numerically different. This is equivalent to `!-eq`.

4. You can perform various file attribute tests:

   -d
   : Is **d**irectory.
   Treats the *TestString* as a pathname and tests whether or not it exists, and is a directory.

   -f
   : Is regular **f**ile.
   Treats the *TestString* as a pathname and tests whether or not it exists, and is a regular file.

   -F
   : Is existing file, via subrequest.
   Checks whether or not *TestString* is a valid file, accessible via all the server's currently-configured access controls for that path. This uses an internal subrequest to do the check, so use it with care - it can impact your server's performance!

-h

Is symbolic link, bash convention.
See **-l**.

-l

Is symbolic **l**ink.
Treats the *TestString* as a pathname and tests whether or not it exists, and is a symbolic link. May also use the bash convention of **-L** or **-h** if there's a possibility of confusion such as when using the **-lt** or **-le** tests.

-L

Is symbolic link, bash convention.
See **-l**.

-s

Is regular file, with **s**ize.
Treats the *TestString* as a pathname and tests whether or not it exists, and is a regular file with size greater than zero.

-U

Is existing URL, via subrequest.
Checks whether or not *TestString* is a valid URL, accessible via all the server's currently-configured access controls for that path. This uses an internal subrequest to do the check, so use it with care - it can impact your server's performance!

This flag *only* returns information about things like access control, authentication, and authorization. This flag *does not* return information about the status code the configured handler (static file, CGI, proxy, etc.) would have returned.

-x

Has e**x**ecutable permissions.
Treats the *TestString* as a pathname and tests whether or not it exists, and has executable permissions. These permissions are determined according to the underlying OS.

For example:

```
RewriteCond /var/www/%{REQUEST_URI} !-f
RewriteRule ^(.+) /other/archive/$1 [R]
```

5. If the *TestString* has the special value `expr`, the *CondPattern* will be treated as an ap_expr (↗ ../expr.html) .

   In the below example, `-strmatch` is used to compare the `REFERER` against the site hostname, to block unwanted hotlinking.

   ```
   RewriteCond expr "! %{HTTP_REFERER} -strmatch '*://%{HTTP_HOST}/*'"
   RewriteRule "^/images" "-" [F]
   ```

You can also set special flags for *CondPattern* by appending **[*flags*]** as the third argument to the `RewriteCond` directive, where *flags* is a comma-separated list of any of the following flags:

- '**`nocase|NC`**' (**n**o **c**ase)
  This makes the test case-insensitive - differences between 'A-Z' and 'a-z' are ignored, both in the expanded *TestString* and the *CondPattern*. This flag is effective only for comparisons between *TestString* and *CondPattern*. It has no effect on filesystem and subrequest checks.
- '**`ornext|OR`**' (**or** next condition)
  Use this to combine rule conditions with a local OR instead of the implicit AND. Typical example:

  ```
  RewriteCond "%{REMOTE_HOST}"  "^host1"  [OR]
  RewriteCond "%{REMOTE_HOST}"  "^host2"  [OR]
  RewriteCond "%{REMOTE_HOST}"  "^host3"
  RewriteRule ...some special stuff for any of these hosts...
  ```

  Without this flag you would have to write the condition/rule pair three times.
- '**`novary|NV`**' (**n**o **v**ary)
  If a HTTP header is used in the condition, this flag prevents this header from being added to the Vary header of the response.
  Using this flag might break proper caching of the response if the representation of this response varies on the value of this header. So this flag should be only used if the meaning of the Vary header is well understood.

Example:

To rewrite the Homepage of a site according to the ``User-Agent:" header of the request, you can use the following:

```
RewriteCond  "%{HTTP_USER_AGENT}"  "(iPhone|Blackberry|Android)"
RewriteRule  "^/$"                 "/homepage.mobile.html"  [L]

RewriteRule  "^/$"                 "/homepage.std.html"     [L]
```

Explanation: If you use a browser which identifies itself as a mobile browser (note that the example is incomplete, as there are many other mobile platforms), the mobile version of the homepage is served. Otherwise, the standard page is served.

By default, multiple `RewriteCond`s are evaluated in sequence with an implied logical **AND**. If a condition fails, in the absence of an **`OR`** flag, the entire ruleset is abandoned, and further conditions are not evaluated.

RewriteEngine Directive

Description:	Enables or disables runtime rewriting engine
Syntax:	`RewriteEngine on|off`
Default:	`RewriteEngine off`
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Extension
Module:	mod_rewrite

The `RewriteEngine` directive enables or disables the runtime rewriting engine. If it is set to `off` this module does no runtime processing at all. It does not even update the `SCRIPT_URx` environment variables.

Use this directive to disable rules in a particular context, rather than commenting out all the `RewriteRule` directives.

Note that rewrite configurations are not inherited by virtual hosts. This means that you need to have a `RewriteEngine on` directive for each virtual host in which you wish to use rewrite rules.

`RewriteMap` directives of the type `prg` are not started during server initialization if they're defined in a context that does not have `RewriteEngine` set to `on`

RewriteMap Directive

Description:	Defines a mapping function for key-lookup
Syntax:	`RewriteMap` *`MapName MapType:MapSource`* [*`MapTypeOptions`*]
Context:	server config, virtual host
Status:	Extension
Module:	mod_rewrite
Compatibility:	The 3rd parameter, MapTypeOptions, in only available from Apache 2.4.29 and later

The `RewriteMap` directive defines a *Rewriting Map* which can be used inside rule substitution strings by the mapping-functions to insert/substitute fields through a key lookup. The source of this lookup can be of various types.

The *MapName* is the name of the map and will be used to specify a mapping-function for the substitution strings of a rewriting rule via one of the following constructs:

`${` *MapName* `:` *LookupKey* `}`
`${` *MapName* `:` *LookupKey* `|` *DefaultValue* `}`

When such a construct occurs, the map *MapName* is consulted and the key *LookupKey* is looked-up. If the key is found, the map-function construct is substituted by *SubstValue*. If the key is not found then it is substituted by *DefaultValue* or by the empty string if no *DefaultValue* was specified. Empty values behave as if the key was absent, therefore it is not possible to distinguish between empty-valued keys and absent keys.

For example, you might define a `RewriteMap` as:

```
RewriteMap examplemap "txt:/path/to/file/map.txt"
```

You would then be able to use this map in a `RewriteRule` as follows:

```
RewriteRule "^/ex/(.*)" "${examplemap:$1}"
```

The meaning of the *MapTypeOptions* argument depends on particular *MapType*. See the Using RewriteMap (↗ ../rewrite/rewritemap.html) for more information.

The following combinations for *MapType* and *MapSource* can be used:

txt
: A plain text file containing space-separated key-value pairs, one per line. (Details ...)

rnd
: Randomly selects an entry from a plain text file (Details ...)

dbm
: Looks up an entry in a dbm file containing name, value pairs. Hash is constructed from a plain text file format using the `httxt2dbm` utility. (Details ...)

int
: One of the four available internal functions provided by `RewriteMap`: toupper, tolower, escape or unescape. (Details ...)

prg
: Calls an external program or script to process the rewriting. (Details ...)

dbd or fastdbd
: A SQL SELECT statement to be performed to look up the rewrite target. (Details ...)

Further details, and numerous examples, may be found in the RewriteMap HowTo (↗ ../rewrite/rewritemap.html)

RewriteOptions Directive

Description:	Sets some special options for the rewrite engine
Syntax:	`RewriteOptions` *`Options`*
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Extension
Module:	mod_rewrite

The `RewriteOptions` directive sets some special options for the current per-server or per-directory configuration. The *Option* string can currently only be one of the following:

`Inherit`
: This forces the current configuration to inherit the configuration of the parent. In per-virtual-server context, this means that the maps, conditions and rules of the main server are inherited. In per-directory context this means that conditions and rules of the parent directory's `.htaccess` configuration or `<Directory>` sections are inherited. The inherited rules are virtually copied to the section where this directive is being used. If used in combination with local rules, the inherited rules are copied behind the local rules. The position of this directive - below or above of local rules - has no influence on this behavior. If local rules forced the rewriting to stop, the inherited rules won't be processed.

> Rules inherited from the parent scope are applied **after** rules specified in the child scope.

`InheritBefore`
: Like `Inherit` above, but the rules from the parent scope are applied **before** rules specified in the child scope.
Available in Apache HTTP Server 2.3.10 and later.

`InheritDown`
: If this option is enabled, all child configurations will inherit the configuration of the current configuration. It is equivalent to specifying `RewriteOptions Inherit` in all child configurations. See the `Inherit` option for more details on how the parent-child relationships are handled.
Available in Apache HTTP Server 2.4.8 and later.

`InheritDownBefore`
: Like `InheritDown` above, but the rules from the current scope are applied **before** rules specified in any child's scope.
Available in Apache HTTP Server 2.4.8 and later.

`IgnoreInherit`

This option forces the current and child configurations to ignore all rules that would be inherited from a parent specifying `InheritDown` or `InheritDownBefore`.
Available in Apache HTTP Server 2.4.8 and later.

`AllowNoSlash`
By default, `mod_rewrite` will ignore URLs that map to a directory on disk but lack a trailing slash, in the expectation that the `mod_dir` module will issue the client with a redirect to the canonical URL with a trailing slash.

When the `DirectorySlash` directive is set to off, the `AllowNoSlash` option can be enabled to ensure that rewrite rules are no longer ignored. This option makes it possible to apply rewrite rules within .htaccess files that match the directory without a trailing slash, if so desired.
Available in Apache HTTP Server 2.4.0 and later.

`AllowAnyURI`
When `RewriteRule` is used in `VirtualHost` or server context with version 2.2.22 or later of httpd, `mod_rewrite` will only process the rewrite rules if the request URI is a URL-path (↗ directive-dict.html#Syntax) . This avoids some security issues where particular rules could allow "surprising" pattern expansions (see CVE-2011-3368 (↗ http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-3368) and CVE-2011-4317 (↗ http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-4317)). To lift the restriction on matching a URL-path, the `AllowAnyURI` option can be enabled, and `mod_rewrite` will apply the rule set to any request URI string, regardless of whether that string matches the URL-path grammar required by the HTTP specification.
Available in Apache HTTP Server 2.4.3 and later.

> **Security Warning**
> Enabling this option will make the server vulnerable to security issues if used with rewrite rules which are not carefully authored. It is **strongly recommended** that this option is not used. In particular, beware of input strings containing the '@' character which could change the interpretation of the transformed URI, as per the above CVE names.

`MergeBase`
With this option, the value of `RewriteBase` is copied from where it's explicitly defined into any sub-directory or sub-location that doesn't define its own `RewriteBase`. This was the default behavior in 2.4.0 through 2.4.3, and the flag to restore it is available Apache HTTP Server 2.4.4 and later.

`IgnoreContextInfo`
When a relative substitution is made in directory (htaccess) context and `RewriteBase` has not been set, this module uses some extended URL and filesystem context information to change the relative substitution back into a URL. Modules such as `mod_userdir` and `mod_alias` supply this extended context info. Available in 2.4.16 and later.

`LegacyPrefixDocRoot`
Prior to 2.4.26, if a substitution was an absolute URL that matched the current virtual host, the URL might first be reduced to a URL-path and then later reduced to a local path. Since the URL can be reduced to a local path, the path should be prefixed with the document root. This prevents a file such as /tmp/myfile from being accessed when a request is made to http://host/file/myfile with the following `RewriteRule`.

```
RewriteRule /file/(.*) http://localhost/tmp/$1
```

This option allows the old behavior to be used where the document root is not prefixed to a local path that was reduced from a URL. Available in 2.4.26 and later.

RewriteRule Directive

Description:	Defines rules for the rewriting engine
Syntax:	`RewriteRule` *`Pattern Substitution`* `[`*`flags`*`]`
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Extension
Module:	mod_rewrite

The `RewriteRule` directive is the real rewriting workhorse. The directive can occur more than once, with each instance defining a single rewrite rule. The order in which these rules are defined is important - this is the order in which they will be applied at run-time.

Pattern is a perl compatible regular expression. What this pattern is compared against varies depending on where the `RewriteRule` directive is defined.

What is matched?

- In `VirtualHost` context, The *Pattern* will initially be matched against the part of the URL after the hostname and port, and before the query string (e.g. "/app1/index.html"). This is the (%-decoded) URL-path (↗ directive-dict.html#Syntax) .
- In per-directory context (`Directory` and .htaccess), the *Pattern* is matched against only a partial path, for example a request of "/app1/index.html" may result in comparison against "app1/index.html" or "index.html" depending on where the `RewriteRule` is defined.
  The directory path where the rule is defined is stripped from the currently mapped filesystem path before comparison (up to and including a trailing slash). The net result of this per-directory prefix stripping is that rules in this context only match against the portion of the currently mapped filesystem path "below" where the rule is defined.
  Directives such as `DocumentRoot` and `Alias`, or even the result of previous `RewriteRule` substitutions, determine the currently mapped filesystem path.
- If you wish to match against the hostname, port, or query string, use a `RewriteCond` with the `%{HTTP_HOST}`, `%{SERVER_PORT}`, or `%{QUERY_STRING}` variables respectively.

Per-directory Rewrites

- The rewrite engine may be used in .htaccess files and in `<Directory>` sections, with some additional complexity.
- To enable the rewrite engine in this context, you need to set "`RewriteEngine On`" **and** "`Options FollowSymLinks`" must be enabled. If your administrator has disabled override of `FollowSymLinks` for a user's directory, then you cannot use the rewrite engine. This restriction is required for security reasons.
- See the `RewriteBase` directive for more information regarding what prefix will be added back to relative substitutions.
- If you wish to match against the full URL-path in a per-directory (htaccess) RewriteRule, use the `%{REQUEST_URI}` variable in a `RewriteCond`.
- The removed prefix always ends with a slash, meaning the matching occurs against a string which *never* has a leading slash. Therefore, a *Pattern* with `^/` never matches in per-directory context.
- Although rewrite rules are syntactically permitted in `<Location>` and `<Files>` sections (including their regular expression counterparts), this should never be necessary and is unsupported. A likely feature to break in these contexts is relative substitutions.
- The `If` blocks follow the rules of the *directory* context.
- By default, mod_rewrite overrides rules when merging sections belonging to the same context. The `RewriteOptions` directive can change this behavior, for example using the *Inherit* setting.
- The `RewriteOptions` also regulates the behavior of sections that are stated at the same nesting level of the configuration. In the following example, by default only the RewriteRules stated in the second `If` block are considered, since the first ones are overridden. Using `RewriteOptions` Inherit forces mod_rewrite to merge the two sections and consider both set of statements, rather than only the last one.

```
<If "true">
  # Without RewriteOptions Inherit, this rule is overridden by the next
  # section and no redirect will happen for URIs containing 'foo'
  RewriteRule foo http://example.com/foo [R]
</If>
<If "true">
  RewriteRule bar http://example.com/bar [R]
</If>
```

For some hints on regular expressions (↗ ../glossary.html#regex) , see the mod_rewrite Introduction (↗ ../rewrite/intro.html#regex) .

In `mod_rewrite`, the NOT character ('`!`') is also available as a possible pattern prefix. This enables you to negate a pattern; to say, for instance: ``*if the current URL does **NOT** match this pattern*". This can be used for exceptional cases, where it is easier to match the negative pattern, or as a last default rule.

Note

When using the NOT character to negate a pattern, you cannot include grouped wildcard parts in that pattern. This is because, when the pattern does NOT match (ie, the negation matches), there are no contents for the groups. Thus, if negated patterns are used, you cannot use `$N` in the substitution string!

The *Substitution* of a rewrite rule is the string that replaces the original URL-path that was matched by *Pattern*. The *Substitution* may be a:

file-system path
: Designates the location on the file-system of the resource to be delivered to the client. Substitutions are only treated as a file-system path when the rule is configured in server (virtualhost) context and the first component of the path in the substitution exists in the file-system

URL-path
: A `DocumentRoot`-relative path to the resource to be served. Note that `mod_rewrite` tries to guess whether you have specified a file-system path or a URL-path by checking to see if the first segment of the path exists at the root of the file-system. For example, if you specify a *Substitution* string of `/www/file.html`, then this will be treated as a URL-path *unless* a directory named `www` exists at the root or your file-system (or, in the case of using rewrites in a `.htaccess` file, relative to your document root), in which case it will be treated as a file-system path. If you wish other URL-mapping directives (such as `Alias`) to be applied to the resulting URL-path, use the `[PT]` flag as described below.

Absolute URL
: If an absolute URL is specified, `mod_rewrite` checks to see whether the hostname matches the current host. If it does, the scheme and hostname are stripped out and the resulting path is treated as a URL-path. Otherwise, an external redirect is performed for the given URL. To force an external redirect back to the current host, see the `[R]` flag below.

  Note that a redirect (implicit or not) using an absolute URI will include the requested query-string, to prevent this see the `[QSD]` flag below.

- (dash)
: A dash indicates that no substitution should be performed (the existing path is passed through untouched). This is used when a flag (see below) needs to be applied without changing the path.

In addition to plain text, the *Substitution* string can include

1. back-references (`$N`) to the RewriteRule pattern
2. back-references (`%N`) to the last matched RewriteCond pattern
3. server-variables as in rule condition test-strings (`%{VARNAME}`)
4. mapping-function calls (`${mapname:key|default}`)

Back-references are identifiers of the form **$N** (**N**=0..9), which will be replaced by the contents of the **N**th group of the matched *Pattern*. The server-variables are the same as for the *TestString* of a `RewriteCond` directive. The mapping-functions come from the `RewriteMap` directive and are explained there. These three types of variables are expanded in the order above.

Rewrite rules are applied to the results of previous rewrite rules, in the order in which they are defined in the config file. The URL-path or file-system path (see "What is matched?" (↗ #what_is_matched) , above) is **completely replaced** by the *Substitution* and the rewriting process continues until all rules have been applied, or it is explicitly terminated by an **L** flag (↗ ../rewrite/flags.html#flag_l) , or other flag which implies immediate termination, such as **END** or **F**.

Modifying the Query String

By default, the query string is passed through unchanged. You can, however, create URLs in the substitution string containing a query string part. Simply use a question mark inside the substitution string to indicate that the following text should be re-injected into the query string. When you want to erase an existing query string, end the substitution string with just a question mark. To combine new and old query strings, use the `[QSA]` flag.

Additionally you can set special actions to be performed by appending **[*flags*]** as the third argument to the `RewriteRule` directive. *Flags* is a comma-separated list, surround by square brackets, of any of the flags in the following table. More details, and examples, for each flag, are available in the Rewrite Flags document (↗ ../rewrite/flags.html) .

Flag and syntax	Function
B	Escape non-alphanumeric characters in backreferences *before* applying the transformation. *details* ...
backrefnoplus|BNP	If backreferences are being escaped, spaces should be escaped to %20 instead of +. Useful when the backreference will be used in the path component rather than the query string.*details* ...
chain|C	Rule is chained to the following rule. If the rule fails, the rule(s) chained to it will be skipped. *details* ...
cookie|CO=*NAME*:*VAL*	Sets a cookie in the client browser. Full syntax is: CO=*NAME*:*VAL*:*domain*[:*lifetime*[:*path*[:*secure*[:*httponly*[*samesite*]]]]] *details* ...
discardpath|DPI	Causes the PATH_INFO portion of the rewritten URI to be discarded. *details* ...
END	Stop the rewriting process immediately and don't apply any more rules. Also prevents further execution of rewrite rules in per-directory and .htaccess context. (Available in 2.3.9 and later) *details* ...
env|E=[!]*VAR*[:*VAL*]	Causes an environment variable *VAR* to be set (to the value *VAL* if provided). The form !*VAR* causes the environment variable *VAR* to be unset. *details* ...
forbidden|F	Returns a 403 FORBIDDEN response to the client browser. *details* ...
gone|G	Returns a 410 GONE response to the client browser. *details* ...
Handler|H=*Content-handler*	Causes the resulting URI to be sent to the specified *Content-handler* for processing. *details* ...
last|L	Stop the rewriting process immediately and don't apply any more rules. Especially note caveats for per-directory and .htaccess context (see also the END flag). *details* ...
next|N	Re-run the rewriting process, starting again with the first rule, using the result of the ruleset so far as a starting point. *details* ...
nocase|NC	Makes the pattern comparison case-insensitive. *details* ...
noescape|NE	Prevent mod_rewrite from applying hexcode escaping of special characters in the result of rewrites that result in redirection. *details* ...

nosubreq|NS	Causes a rule to be skipped if the current request is an internal sub-request. *details* ...
proxy|P	Force the substitution URL to be internally sent as a proxy request. *details* ...
passthrough|PT	Forces the resulting URI to be passed back to the URL mapping engine for processing of other URI-to-filename translators, such as `Alias` or `Redirect`. *details* ...
qsappend|QSA	Appends any query string from the original request URL to any query string created in the rewrite target.*details* ...
qsdiscard|QSD	Discard any query string attached to the incoming URI. *details* ...
qslast|QSL	Interpret the last (right-most) question mark as the query string delimiter, instead of the first (left-most) as normally used. Available in 2.4.19 and later. *details* ...
redirect|R[=*code*]	Forces an external redirect, optionally with the specified HTTP status code. *details* ...
skip|S=*num*	Tells the rewriting engine to skip the next *num* rules if the current rule matches. *details* ...
type|T=*MIME-type*	Force the MIME-type of the target file to be the specified type. *details* ...

Home directory expansion

When the substitution string begins with a string resembling "/~user" (via explicit text or backreferences), `mod_rewrite` performs home directory expansion independent of the presence or configuration of `mod_userdir`.
This expansion does not occur when the *PT* flag is used on the `RewriteRule` directive.

Here are all possible substitution combinations and their meanings:

**Inside per-server configuration (`httpd.conf`)
for request "`GET /somepath/pathinfo`":**

Given Rule	Resulting Substitution
^/somepath(.*) otherpath$1	invalid, not supported
^/somepath(.*) otherpath$1 [R]	invalid, not supported
^/somepath(.*) otherpath$1 [P]	invalid, not supported
^/somepath(.*) /otherpath$1	/otherpath/pathinfo
^/somepath(.*) /otherpath$1 [R]	http://thishost/otherpath/pathinfo via external redirection
^/somepath(.*) /otherpath$1 [P]	doesn't make sense, not supported
^/somepath(.*) http://thishost/otherpath$1	/otherpath/pathinfo
^/somepath(.*) http://thishost/otherpath$1 [R]	http://thishost/otherpath/pathinfo via external redirection
^/somepath(.*) http://thishost/otherpath$1 [P]	doesn't make sense, not supported
^/somepath(.*) http://otherhost/otherpath$1	http://otherhost/otherpath/pathinfo via external redirection
^/somepath(.*) http://otherhost/otherpath$1 [R]	http://otherhost/otherpath/pathinfo via external redirection (the [R] flag is redundant)
^/somepath(.*) http://otherhost/otherpath$1 [P]	http://otherhost/otherpath/pathinfo via internal proxy

**Inside per-directory configuration for `/somepath`
(`/physical/path/to/somepath/.htaccess`, with `RewriteBase "/somepath"`)
for request "`GET /somepath/localpath/pathinfo`":**

Given Rule	Resulting Substitution
^localpath(.*) otherpath$1	/somepath/otherpath/pathinfo
^localpath(.*) otherpath$1 [R]	http://thishost/somepath/otherpath/pathinfo via external redirection
^localpath(.*) otherpath$1 [P]	doesn't make sense, not supported
^localpath(.*) /otherpath$1	/otherpath/pathinfo
^localpath(.*) /otherpath$1 [R]	http://thishost/otherpath/pathinfo via external redirection
^localpath(.*) /otherpath$1 [P]	doesn't make sense, not supported
^localpath(.*) http://thishost/otherpath$1	/otherpath/pathinfo
^localpath(.*) http://thishost/otherpath$1 [R]	http://thishost/otherpath/pathinfo via external redirection
^localpath(.*) http://thishost/otherpath$1 [P]	doesn't make sense, not supported
^localpath(.*) http://otherhost/otherpath$1	http://otherhost/otherpath/pathinfo via external redirection
^localpath(.*) http://otherhost/otherpath$1 [R]	http://otherhost/otherpath/pathinfo via external redirection (the [R] flag is redundant)
^localpath(.*) http://otherhost/otherpath$1 [P]	http://otherhost/otherpath/pathinfo via internal proxy

Comments

Notice:
This is not a Q&A section. Comments placed here should be pointed towards suggestions on improving the documentation or server, and may be removed by our moderators if they are either implemented or considered invalid/off-topic. Questions on how to manage the Apache HTTP Server should be directed at either our IRC channel, #httpd, on Libera.chat, or sent to our mailing lists.