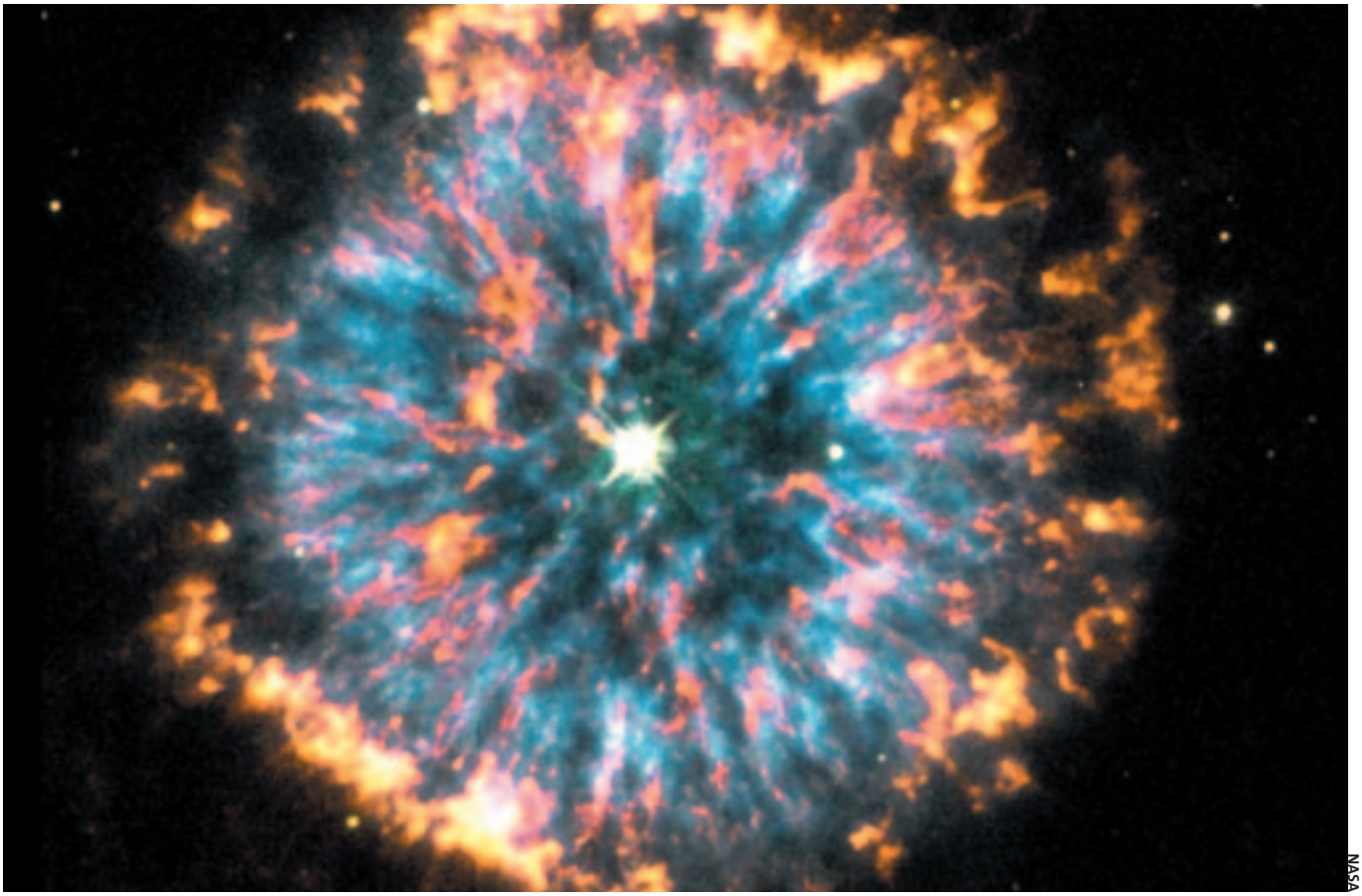
Re-Writing URLs with the Apache mod_rewrite Module

Black Magic

Static URLs pointing to static pages? No way! Apache can use the mod_rewrite module to automatically re-write requested URLs. BY MARC ANDRÉ SELIG



NASA

If a browser or any other HTTP client sends a request to a web server, the most important element of the request will always be a URL. The URL uniquely identifies the required resource. But a URL does not need to point to a file on the server's hard disk. This is obviously the case where CGI scripts or other dynamic pages are involved.

Apache will need to search for any file not directly referenced. To do so, the server needs to be told the path to the "DocumentRoot" in the configuration file "httpd.conf". On the other hand, directives such as "Alias", "ScriptAlias", and "Redirect" can be used to redirect the incoming requests. The mod_rewrite module that is pre-installed on any Apache 1.3 or 2.0 machine, is far superior to the methods mentioned so far. The module uses regular expressions to check the requested URL and, if needed, one or two additional variables. If a number of conditions apply, the module replaces the URL with a new one, although you can still access some elements of the original address. In other words mod_rewrite can either replace entire URLs or simply re-write parts of them.

This is why the author of this module, Ralf S. Engelschall, calls it the "Swiss Army Knife of URL Manipulation". But Brian Moore hits the nail on the head: "Mod_rewrite is voodoo. Damned cool voodoo, but still voodoo." Mod_rewrite is one of the most complex elements in Apache – the control logic is not particularly intuitive, and debugging is a nightmare. So, there are good reasons for not making it part of Apache's standard equipment – the maintainers simply do not want to confuse newcomers.

Not Automatically Included

This does not mean that mod_rewrite is not included with the web server, but the "./configure" will not include the module in "httpd" unless you explicitly tell it to do so. Some Linux distributions take this step for you. You might therefore like to type "httpd -l" to discover the modules already designated for Apache.

If the output does not contain a reference to "mod_rewrite.c", you may still find that the module was compiled as a dynamic shared object (DSO). In this case you will need to look for a file called "mod_rewrite.so" in your Apache module directory; you can type "locate mod_rewrite.so" to do so. If you draw a blank again, you will need to re-compile Apache supplying the "--enable-rewrite" "./configure" option.

The possibility of compiling the module as a loadable DSO is probably more interesting for Linux distributors than for the webmaster of a production site – the advantage being the fact that a DSO is completely removed from the webserver, if not used. The disadvantage being the performance impact.

Configure that Server!

You can use the following syntax to enable the DSO variant of the module:

```
LoadModule rewrite_module ➥
modules/mod_rewrite.so
```

in "httpd.conf". Other options can be placed in the server configuration file "httpd.conf", or in the directory specific ".htaccess" configuration files. (Of course, this only applies if you have not already disabled parsing of ".htaccess".)

The first important directive is "RewriteEngine on", which is used to enable the module. If this is missing, mod_rewrite will do nothing at all. For troubleshooting activities, you are recommended to log all the module's activity right from the outset. Use "RewriteLog" to supply the logfile required for this purpose.

If everything works, you will probably want to disable logging later, by commenting out the directives in Listing 1. Logging consumes valuable resources and impacts webserver performance.

Listing 1: "httpd.conf" entries for mod_rewrite

```
LoadModule rewrite_module ➥
modules/mod_rewrite.so

RewriteEngine on

RewriteLog "/export/apache/➥
logs/rewrite.log"
RewriteLogLevel 2
```

Your Apache will call "mod_rewrite" when handling requests. The module uses a set of rules (defined by "RewriteRule" in "httpd.conf" or ".htaccess"), each of which contains a search pattern, a replacement string, and possibly one or more flags. Additional flags to the ones shown in Table 1 are available for complex tasks. See [1] for a comprehensive overview.

How Does it Work?

The web server compares the URL requested by the client with the search pattern in the individual rules. For example:

```
RewriteRule ^/netscape\.html$ ➥
/mozilla.html
```

The first argument contains the search pattern and the second the replacement string; a third argument could contain flags. When the "/netscape.html" URL is requested, mod_rewrite instead returns "/mozilla.html". Since the search string is a regular expression, the period it contains needs to be protected by a backslash to prevent it being interpreted as a wildcard.

RewriteRule's can be prepended in the configuration file by one or multiple "RewriteCond" directives comprising comparitive variables, a search string and possibly a flag. They are used to provide additional conditions. The replacement string defined in the "RewriteRule" only applies if all the rewrite conditions are fulfilled:

```
RewriteCond %{TIME_HOUR}%➥
{TIME_MIN} <0600
RewriteRule ^/special/lunch\.➥
html$ /it_➥
is_too_early.html
```

This example concerns a request for a lunchtime special. The variables "TIME_HOUR" and "TIME_MIN" are concatenated. A request entered at quarter past eight in the evening would thus produce the string "2015". The expression "<0600" is then applied to the string.

If the comparison is successful, the subsequent "RewriteRule" is applied. Instead of the special, the content of the file "/it_is_too_early.html" is returned to the user. As this example shows, some special conditions apply to regular expressions in mod_rewrite – refer to Table 2 for an overview.

In addition to the variables defined in the CGI specificat mod_rewrite also offers a few extra goodies that allow you to query server information and the time.

Table 3 shows the most commonly used variables. Note that the order in which mod_rewrite parses the search keys contradicts the conventions adhered to by most programming languages. So "RewriteCond" always prepends the corresponding "RewriteRule", but the server will still parse the search string in "RewriteRule" first, before going on to check the conditions. In this case adhering to the order stipulated in the configuration file.

Tidy Appearance

One typical application of mod_rewrite is harmonizing URLs. You often find that resources can be accessed via a variety of addresses, but only one of them is the official address. Uniform URLs are particularly important for readability on search machines. If a page can be accessed via a variety of names, each one of these will achieve fewer hits.

Your ranking drops and your page is not so prominent on Google. However, you will certainly not want to remove alternative URLs to avoid impacting availability to users. But there is a solution to this issue: You can allow any possible alternatives, but redirect any

Listing 2a: Harmonizing File Requests

```
RewriteEngine on

RewriteRule  ^/download\.html   /download/    [NC,R=301]
RewriteRule  ^/downloads\.html  /download/    [NC,R=301]

RewriteRule  ^/downloads/(.*)   /download/$1  [NC,R=301]
```

users that type the unofficial URL to the official address.

Listing 2a shows an example: The first two rules redirect requests for "download.html" and "downloads.html" to the "/download/" directory. The "[NC]" flag means "no case", and refers to upper/lower case letters. The condition will thus equally apply to "DownLoad.html". "[R=301]" ensures that the user or search engine will notice they are being redirected, instead of performing this transparently on the web server. Browsers will immediately run the redirection command, whereas a robot or spider will immediately update its records.

The third and final "RewriteRule" points the content of "/downloads/" (with an s at the end) to "/download/" (without an s). The trick is, if there are links pointing to files in "/downloads", the file name (possibly including subdirectories) can be retained. The user will immediately be pointed to the requested file and will not need to climb back up the tree from "/download/".

Listing 2b is more complex: It prepends "www." to any requests. The web server is configured to react to "company.com" but, users should still type "www.company.com".

The first "RewriteCond" filters any requests already containing "www.", as they do not need to be re-written. The second blocks any requests without a "Host:" header - some older clients may still produce them, and redirection would be fatal in this case. The "RewriteRule" finally stores the path for the current request and uses it to create a new, official URL.

Virtual Hosts With a Difference

Now let's look at a more complex practical task. Suppose a company that runs the "company.com" domain, creates the "imode.company.com" subdomain for mobile phone fans. The web content for this subdomain is stored in the subdirectory "i" below the "DocumentRoot". The web server is hosted externally – which prevents the webmaster modifying "httpd.conf" to set up an independent virtual server. The only way to solve this issue is to use an ".htaccess" file, as shown in Listing 3.

Listing 2b: Harmonizing Host Names

```
RewriteCond %{HTTP_HOST} !^www\.      [NC]
RewriteCond %{HTTP_HOST} !^$
RewriteRule ^(.*)        http://www.%{HTTP_HOST}/$1   [R=301]
```

Listing 3: Virtual Hosts with mod_rewrite

```
RewriteEngine on

RewriteCond %{HTTP_HOST}    ^imode\.firma\.com$  [NC]
RewriteCond %{REQUEST_URI}  !^/i/
RewriteRule ^(.*)$          /i/$1
```

Listing 4: Static URLs instead of Query Strings with mod_rewrite

```
RewriteEngine on

RewriteRule  ^/(animals|plants)/(.+)/(.+)\.html$ ➥
/template.php?menu=$1&submenu=$2&subsubmenu=$3  [PT]
```

The first "RewriteCond" compares the "Host:" header contents with the regular expression "^imode\.company\.com$". Thus, the rewrite is restricted to requests directed to "imode.company.com". The "RewriteRule" itself simply takes the entire request and saves the expression that matches the string in parentheses in a group. Now mod_rewrite can replace the URL by "/i/" and the same group. If a user wants to access the "mail.html" page, for example, this would produce the "/i/mail.html" address.

The second "RewriteCond" prevents the system from entering an indefinite loop. None of the conditions previously checked is changed when the file path is replaced. Mod_rewrite would thus indefinitely prepend an "/i/" to the request – until the user gave up. So the second condition filters out any requests that already contain an "/i/" thus preventing the loop condition.

Using Dynamic Scripts to Produce Static Pages

Many search engines seem to be allergic to query strings. If you go to the trouble of implementing a database based template system in PHP, you may find yourself dropping down the ranks on the search engine. This is due to the fact that the spider will notice the give-away question mark in the URL and as a precautionary measure, not create an index for any subsequent pages.

Table 1: Important flags for "RewriteCond" and "RewriteRule"

[NC]	Ignore upper/lower case for comparisons.
[OR]	Links one "RewriteCond" to another using a logical OR. Normally, any existing conditions will need to be fulfilled for a replacement to take place; this flag permits several variants.
[R=301]	Performs external redirection. mod_rewrite is normally transpartent for the user. The status line in the browser will show the URL the user originally typed, although this URL has in fact been re-written. However, if you want to draw the user's attention to the redirection, you might prefer to use external redirection. The browser will then receive an error code and the URL of the new page, and will subsequently actively request this page. The error code to be returned to the browser is defined by the number that follows "R=". "301" (permanently moved) and "302" (temporary redirection) are typical codes.
[L]	Terminate mod_rewrite processing without applying any more rules. This flag prevents a correction that has been performed from being overwritten by a later rule. It also saves the administrator some confusion.
[N]	This runs the newly defined URL through any applicable mod_rewrite rules.
[C]	Only process the next "RewriteRule" if the current rule applied.

Table 2: Regular Expressions in mod_rewrite

.	Any character.
\.	A period.
.+	One or more characters.
\.\+	A period and a plus sign.
.*	No character or multiple characters.
.?	No character or any single character.
^x	"x" at the start of a URL or a file name.
x$	"x" at the end of a URL or a file name.
x|y	Either "x" or "y".
(.*)	Group: The text matched by ".*" is stored in the "$1" variable in the case of a "RewriteRule", or in the variable "%1" if the regexp is used in a "RewriteCond". If you use multiple groups in a single expression, the variables "$2", "$3", ... or "%2", "%3",... are used. These variables can be used in the replacement string.
(x|y)	A different example of a group. Searches for "x" or "y" and stores the matching text.
[-0-9a-z]*	Any number of lower case letters, figures or dashes.
[^/]*	Any number of characters but not the slash character.
!regexp	This expression is true if the regexp is not found.

Additionally, there are a number of special tests. The following operators do not search for regular expressions but compare a string with another, or check for a file name or URL

.

<4500	The comparitive expression is less than 4500. Note: This is not a numeric but an ASCII comparison.
>4500	The comparitive expression is greater than 4500.
=””	The comparitive expression is an empty string.
-d	The comparitive expression is points to a directory.
-f	The comparitive expression is points to a normal file.
-s	The comparitive expression is points to a normal file that is not empty.
-l	The comparitive expression is points to a symbolic link.
-F	The comparitive expression is points to a normal file that can be read by the current client.
-U	The comparitive expression is points to a valid URL that can be read by the current client.

Table 3: Interesting Variables for "RewriteCond" Directives

%{HTTP_ACCEPT}	Media types accepted by the client, for example "text/plain" or "audio/*".
%{HTTP_COOKIE}	Cookies set for the client.
%{HTTP_HOST}	Domain name of the virtual host queried.
%{HTTP_REFERER}	Page with a link to this page (can be omitted).
%{HTTP_USER_AGENT}	Client, such as "Mozilla/4.0".
%{QUERY_STRING}	Query string transferred by a GET form.
%{REMOTE_ADDR}	Client IP address.
%{REMOTE_HOST}	Domain name of the client, if known.
%{REMOTE_USER}	User name of the client, possibly after successfully completing authentication.
%{REQUEST_URI}	The URI requested by the client.
%{REQUEST_FILENAME}	The corresponding file on the local file system.
%{SERVER_ADDR}	The web server IP address.
%{TIME_DAY}	Current date, day.
%{TIME_MON}	Current date, month.
%{TIME_YEAR}	Current date, year.
%{TIME_HOUR}	Current time, hour.
%{TIME_MIN}	Current time, minute.
%{TIME_SEC}	Current time, second.
%{ENV:PATH}	"$PATH" environment variable for Apache.
%{HTTP:Connection}	"Connection:" header in HTTP request. This allows you to query multiple headers.

Another case for mod_rewrite: Where the template system would use a URL something like "http://somewhere.com/template.php?menu = animals&submenu = fish&subsubmenu = shark", the world outside will be shown a nice static URL, such as "http://somewhere.com/animals/fish/shark.html". Listing 4 shows the corresponding entry in "httpd.conf".

The search key comprises three groups, the first of which must contain either the string "animals" or the string "plants". This avoids impacting on any other files or directories. Mod_rewrite uses all three groups to construct an internal URL, which is then called by the PHP script.

The "[PT]" flag ensures that any "Alias", "Redirect", and "ScriptAlias" directives are applied to the result. Thus, the example will not only work for PHP, but for genuine CGI scripts written in Perl or a similar language.

Future

What you have seen so far is just a taster of what mod_rewrite can do: This modules applications are as unbounded as its complexity. Refer to [2] for a collection of useful and useless examples of practical applications. If you intend to experiment with mod_rewrite, use a lab system first. Mod_rewrite is quick enough for production systems, but unexpected configuration errors can take a system down. ■

INFO

[1] Original Documentation: *http://httpd.apache.org/docs-2.0/mod/mod_rewrite.html*

[2] A Treasure Trove of Tips, Tricks and Examples: *http://httpd.apache.org/docs-2.0/misc/rewriteguide.html*

THE AUTHOR

Marc André Selig spends half of his time working as a scientific assistant at the University of Trier and as an ongoing medical doctor in the Schramberg hospital. If he happens to find time for it, his currenty preoccupation is programing web based databases on various Unix platforms.