

mod_rewrite: A Beginner's Guide to URL Rewriting

Problem:

I want to "rewrite" my website's URL using mod_rewrite. How can I start it ?
Applications Must Be Safe

A user must not be able to harm your site in any way by modifying a URL that points to your applications. In order to ensure your site's safe, check all the GET variables coming from your visitors (I think it's trivial to mention that the POST variables are a must to examine).

For example, imagine we have a simple script that shows all the products in a category. Generally, it's called like this:

```
myapp.php?target=showproducts&categoryid=123
```

But what will this application do if ScriptKiddie(tm) comes and types this in his browser:

```
myapp.php?target=showproducts&categoryid=youarebeinghacked
```

Well, many of the sites I've seen will drop some error message complaining about use of the wrong SQL query, invalid MySQL resource ID, and so on... These sites are not secure. And can anyone guarantee that a site-to-be-finished-yesterday will have all the parameter verifications --even in a programmer group having only 2 or 3 people?

Applications Must Be Search-Engine Friendly

It's not generally known, but many of the search engines will not index your site in depth if it contains links to dynamic pages like the one mentioned above. They simply take the "name" part of the URL (that's everything before the question mark, which contains the parameters that are needed for most of the scripts to run correctly), and then try to fetch the contents of the page. To make it clear, here are some links from our fictitious page:

```
myapp.php?target=showproducts&categoryid=123
myapp.php?target=showproducts&categoryid=124
myapp.php?target=showproducts&categoryid=125
```

Unfortunately, there's a big chance that some of the search engines will try to download the following page:

```
myapp.php
```

In most cases calling a script like this causes an error - but if not, I'm sure it will not show the proper contents the link was pointing to. Just try this search at google.com: "you have an error in your sql syntax" .php -forum

There are both huge bugs and security in the scripts listed -- again, these scripts are not search-engine friendly.

Applications must be user-friendly

If your application uses links like:

```
http://www.downloadsite.com?category=34769845698752354
```

then most of your visitors will find it difficult to get back to their favourite category (eg. Nettools/Messengers) every time they start from the main page of your site. Instead, they'd like to see URLs like this:

```
http://www.downloadsite.com/Nettools/Messengers
```

It's even easier for the user to find (pick) the URL from the browsers' drop-down list as they type into the Location field (though of course this only works if the user has visited that previously).

And what about you?

Now you have everything you need to answer the following questions:

- Is your site really safe enough?
- Can you protect your site from hackers?
- Are your Websites search-engine compatible?
- Are the URLs on your site 'user friendly' - are they easy to remember? ...and would you like it to be? (everyone who answered 'yes' to all 4 questions: have a beer!)

An elegant solution

Okay, okay, I think you want to know the solution. Well, let's get started. You'll need:

- everyone's favourite Apache Webserver installed (v1.2 or later)
- optionally, your favourite CGI scripts configured for Apache. Yes, I've said optionally, since what we're going to do will happen right inside Apache and not PHP, or Perl, etc.
- since (nearly) everything in Apache is controlled through its configuration files (httpd.conf, .htaccess, etc.), being familiar with these files might help you. You'll also need to have write access to this file, and access to restart the Apache. I'd strongly recommend you do everything on a private testserver first, rather than on your own, or your company's, production server!

Most of you will have read and/or heard about mod_rewrite -- yes, it's an Apache module, and it's even installed by default! Go and check your modules directory (note that under *nix operating systems there's a chance that your Apache was compiled with missing mod_rewrite, in which case, consult your sysadmin).

We're going use this tiny module to achieve everything mentioned above. To use this module, first we have to enable it, since it's initially disabled in the configuration file. Open the httpd.conf file and uncomment the following lines (remove the trailing #s):

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

The first line tells Apache to load the mod_rewrite module, while the second one enables the use of it. After you restart Apache, mod_rewrite should be enabled, but not yet running. What is the mod_rewrite Solution, Exactly?

But what does it exactly do? Hey! Here comes the whole point of this article!

mod_rewrite catches URLs that meet specific conditions, and rewrites them as it was told to.

For example, you can have a non-existing

`http://www.mysite.com/anything`

URL that is rewritten to:

`http://www.mysite.com/deep/stuff/very_complicated_url?text=`

`having_lots_of_extra_characters`

Did you expect something more? Be patient...

```
<IfModule mod_rewrite.c>
RewriteEngine on
RewriteRule ^/shortcut$ /complicated/and/way/too/long/url/here
</IfModule>
```

Of course this, too, should go into the httpd.conf file again, (you can even put it into a virtualhost context).

After you restart Apache (you'll get used to it soon!) you can type this into your browser:

`http://localhost/shortcut`

If there's a directory structure `/complicated/and/way/too/long/url/here` existing in your document root, you're going to be "redirected" there, where you'll see the contents of this directory (eg, the directory listing, index.html, whatever there is).

To understand mod_rewrite better, it's important to know that this is not true redirection. "Classic" redirection is done with the Location: header of the HTTP protocol, and tells the browser itself to go to another URL. There are numerous ways to do this, for example, in PHP you could write:

```
<?
// this PHP file is located at http://localhost/shortcut/index.php
header
```

```
("Location: /complicated/and/way/too/long/url/here");
?>
```

This code shows the same page by sending a HTTP header back to the browser. That header tells the browser to move to another URL location instantly. But, what `mod_rewrite` does is totally different: it 'tricks' the browser, and serves the page as if it were really there - that's why this is an URL rewriter and not a simple redirector (you can even verify the HTTP headers sent and received to understand the difference).

But it's not just shortening paths that makes `mod_rewrite` the "Swiss Army Knife of URL manipulation"... Rules

You've just seen how to specify a really simple RewriteRule. Now let's take a closer look...

RewriteRule Pattern Substitution [Flag(s)]

RewriteRule is a simple instruction that tells `mod_rewrite` what to do. The magic is that you can use regular expressions in the Pattern and references in the Substitution strings. What do you think of the following rule?

```
RewriteRule /products/([0-9]+) /siteengine/products.php?id=$1
```

Now you can use the following syntax in your URLs:

```
http://localhost/products/123
```

After restarting Apache, you'll find this is translated as:

```
http://localhost/siteengine/products.php?id=123
```

If you use only 'fancy' URLs in your scripts, there will be no way for your visitor to find out where your script resides (/siteengine in the example), what its name is (products.php), or what the name of the parameter to pass (productid) is! Do you like it? We've just completed two of our tasks, look!

- Search-engine compatibility: there are no fancy characters in the URL, so the engines will explore your whole site
- Security: ScriptKiddie(tm)-modified URLs will cause no error, as they're verified with the regular expression first to be a number - URLs with no proper syntax can't even reach the script itself.

Of course, you can create more complex RewriteRules. For example, here's a set of rules I'm using on a site:

```
RewriteRule ^/products$ /content.php
RewriteRule ^/products/([0-9]+)$ /content.php?id=$1
RewriteRule
^/products/([0-9]+),([adj]*),([0-9]{0,3}),([0-9]*),([0-9]*$)
/marso/content.php?id=$1&sort=$2&order=$3&start=$4
```

Thanks to these rules I can use the followings links in the application:

- Show an opening page that contains product categories: <http://somesite.hu/products>
- Product listing, categoryid is 123, page 1 (as default), default order: <http://somesite.hu/products/123>
- Product listing, categoryid is 123, page 2, descending order by third field (d for descending, 3 for third field): <http://somesite.hu/products/123,d,3,2>

This is also an example of the use of multiple RewriteRules. When there's a RegExp match, the proper substitution occurs, `mod_rewrite` stops running and Apache serves the page with the substituted URL. Should there be no match (after processing all the rules), a usual 404 page comes up. And of course you can also define one or more rules (eg. `^.*$` as last pattern) to specify which script(s) to run depending on the mistaken URL.

The third, optional part of RewriteRule is:

RewriteRule Pattern Substitution Flag(s)

With flags, you can send specific headers to the browser when the URL matches the pattern, such as:

- 'forbidden' or 'f' for 403 forbidden,
- 'gone' or 'g' for 410 gone,

- you may also force redirection, or force a MIME-type.

You can even use the:

- 'nocase' or 'NC' flag to make the pattern case-insensitive
- 'next/N' to loop back to the first rule ('next round' -- though this may result in an endless loop, be careful with it!)
- 'skip=N'/'S=N' to skip the following N rules

...and so on.

I hope you feel like I felt while playing around with this module for the first time!

Conditions

But that's not all! Though RewriteRule gives you an opportunity to have professional URL rewriting, you can make it more customized using conditions.

The format of the conditions is simple:

RewriteCond Something_to_test Condition

Any RewriteCond condition affects the behaviour of the following RewriteRule, which is a little confusing, as RewriteCond won't be evaluated until the following RewriteRule pattern matches the current URL.

It works like this: mod_rewrite takes all the RewriteRules and starts matching the current URL against each RewriteRule pattern. If there's a RewriteRule pattern that matches the URL, mod_rewrite checks if there are existing conditions for this RewriteRule, and if the first one returns true. If it does, the proper substitution will occur, but if not, mod_rewrite looks for remaining conditions. When there are no more conditions, the subsequent RewriteRule is checked.

This way you can customize URL rewriting using conditions based on practically everything that's known during a HTTP transfer in Apache -- and a lot more! Basically you can use all of these variables in the Something_to_test string:

- HTTP header variables: HTTP_USER_AGENT, HTTP_REFERER, HTTP_COOKIE, HTTP_FORWARDED, HTTP_HOST, HTTP_PROXY_CONNECTION, HTTP_ACCEPT
- Connection & request variables: REMOTE_ADDR, REMOTE_HOST, REMOTE_USER, REMOTE_IDENT, REQUEST_METHOD, SCRIPT_FILENAME, PATH_INFO, QUERY_STRING, AUTH_TYPE
- Server internal variables: DOCUMENT_ROOT, SERVER_ADMIN, SERVER_NAME, SERVER_ADDR, SERVER_PORT, SERVER_PROTOCOL, SERVER_SOFTWARE
- System variables: TIME_YEAR, TIME_MON, TIME_DAY, TIME_HOUR, TIME_MIN, TIME_SEC, TIME_WDAY, TIME
- mod_rewrite special values: API_VERSION, THE_REQUEST, REQUEST_URI, REQUEST_FILENAME, IS_SUBREQ

The condition can be a simple string or a standard regular expression, with additions like:

- <, >, = simple comparison operators
- -f if Something_to_test is a file
- -d if Something_to_test is a directory

As you can see, these are more than enough to specify a condition like this one (taken from the mod_rewrite manual):

```
RewriteCond %{HTTP_USER_AGENT} ^Mozilla.*
RewriteRule ^/$ /homepage.max.html [L]
```

```
RewriteCond %{HTTP_USER_AGENT} ^Lynx.*
RewriteRule ^/$ /homepage.min.html [L]
```

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

When a browser requests the index page, 3 things can happen:

- browser with a Mozilla engine the browser will be served homepage.max.html
- using Lynx (character-based browser) the homepage.min.html will open

- if the browser's name doesn't contain 'Mozilla' nor 'Lynx', the standard homepage.std.html file will be sent

You can even disable users from accessing images from outside your server:

```
RewriteCond %{HTTP_REFERER} !^$
RewriteCond %{HTTP_REFERER} !^http://localhost/*$ [OR,NC]
RewriteCond %{HTTP_REFERER} !^http://mysite.com/*$ [OR,NC]
RewriteCond %{HTTP_REFERER} !^http://www.mysite.com/*$ [OR,NC]
RewriteRule *\.(gif|GIF|jpg|JPG)$ http://mysite/images/bad.gif [L,R]
```

But of course, there are endless possibilities, including IP- or time-dependant conditions, etc.

For Advanced Users

I mentioned user-friendliness in the introduction, and haven't dealt with it. First, let's imagine we're having a huge download site that has the downloadable software separated into categories, each with a unique id (which is used in the SQL SELECTs). We use links like `open.php?categoryid=23487678` to display the contents of a category.

To ensure that our URLs were easily memorized (eg. `http://www.downloadsites.com/Nettools/Messengers`) we could use:

```
RewriteRule ^/NetTools$ /test.php?target=3
RewriteRule ^/NetTools/Messengers$ /test.php?target=34
```

assuming the ID is 3 for the NetTools category and 34 for Messengers subcategory.

But our site is huge, as I've mentioned - who wants to hunt down all the IDs from the database, and then edit the config file by hand? No-one! Instead, we can use the mapping feature of `mod_rewrite`. `Map` allows us to provide a replacement-table - stored in a single text file -- within a hash file (for fast lookups), or even served through an external program!

For better performance I'd generate a single text file using PHP, which contains the following:

```
NetTools      3
NetTools/Messengers 34
```

```
.
```

and so on.

The `httpd.conf` file would contain:

```
RewriteMap categories txt:/path/to/file/categoryids.txt
RewriteRule ^(.*)$ open.php?categoryid=${categories:$1|0}
```

These lines tell `mod_rewrite` to read the `categoryids.txt` file upon Apache startup, and provide the ID for the URL for `open.php`. The `|0` means that `categoryid` will be 0 if there's no matching key in the textfile.

You can also choose to serve the IDs on-the-fly via a script or other executable code. The program is started by Apache on server startup, and runs until shutdown. The program must have buffered I/O disabled, read from the `stdin`, and write results to `stdout` -- it's that simple!

With `RewriteMap` you can do a lot more, including:

- load balancing through servers (using `rnd`),
- creation of a Webcluster that has an homogenous URL layout,
- redirection to mirror sites without modifying your Web application,
- denial of user access based on a hostlist,

and so on.

Tips, Tricks and Advice

- Before using `mod_rewrite` in a production server, I'd recommend setting up a testserver (or playground, whatever you prefer to call it).

- During development, you must avoid using 'old-fashioned' URLs in your application.

- There might still be need to verify data passed through the URL (passing non-existing -- too large or small - IDs, for example, might be risky).

- Writing 'intelligent' RewriteRules saved me coding time and helped me write simpler code. I'm using `error_reporting(E_ALL)`; everywhere (and I recommend it!), but I find it boring to do the following for the ten thousandths time:

```
if (isset($_GET['id']) && (validNumber($_GET['id']))
if (isset($_GET['todo']) && ($_GET['todo']=='deleteitem'))
```

The following trick helped me to get rid of the extra `isset()` expression by providing all the needed parameters each time in the RewriteRules:

```
RewriteRule ^/products/[0-9]+$ products.php?id=$1&todo=
```

I know, I know it's not the answer to the meaning of life -- but it's hard to show how nice and clear a solution this might provide in such a short example.

Finally...

That's all for our 'brief' overview of `mod_rewrite`. After you've mastered the basics, you'll find you can easily create your own rules. If you like the idea of URL rewriting, may want to play with `mod_rewrite` - some ideas follow (note that the underlying PHP code is not important in this case):

```
http://www.mysite.com/1/2/3/content.html
=> 1_2_3_content.html
http://www.mysite.com/1/2/3/content.html
=> content.php ? category=1
```

```
http://www.mysite.com/1/2/3/
=> content.php ? category=1 & subcat1 = 2 & subcat2 = 3
```

```
http://www.mysite.com/1/2/3/details
=> content.php ? category=1 & subcat1 = 2 & subcat2 = 3
http://www.mysite.com/bookshop/browse/bytitle
=> library.php ? target=listbooks & order = title
http://www.mysite.com/bookshop/browse/byauthor
=> library.php ? target=listbooks & order = author
```

```
http://www.mysite.com/bookshop/product/123
=> library.php ? target=showproduct & itemid=123
```

```
http://www.mysite.com/bookshop/helpdesk/2
=> library.php ? target=showhelp & page=2
http://www.mysite.com/bookshop/registration
=> library.php ? target=reg
```