Mod rewrite Tutorials

overview - **syntax** - basic - medium - advanced

Mod Rewrite syntax

The key to good mod rewritting is patterns. Patterns in your urls are how we are going to distinguish what to rewrite and what not to rewrite. We'll get to that later, first we'll need to go over the basics of the mod rewrite syntax.

RewriteRules

Rewriterules are the heart and sole of the mod rewrite, here is where you declare the file to be rewritten, where it is to be rewritten and tack on any special commands.

Rewrite rules are broken down into 4 simple blocks. I'll refer to these blocks as the **Call to action, Pattern**, **Rewrite** and **Command Flag**.

Example:

```
RewriteRule ^dir/([0-9]+)/?$ /index.php?id=$1 [L]
```

Call to action: RewriteRule
Pattern: ^dir/([0-9]+) /?$
Rewrite: /index.php?id=$1
Command Flag: [L]

Between each of these blocks of the rewrite rule there should be a space. With that being said let's go ahead and break down each of these 4 blocks and discuss what they do.

Call to action Block

The only way to screw this up is to spell RewriteRule incorrectly or leave out the space between this and the starting of the **pattern block**. If you do spell it incorrectly you'll trigger an error and the browser will out put a 500 error. Note if you ever see a 500 error on your site it mostlikely due to a bad line of code in your .htaccess file.

Pattern Block

This one little piece of the mod rewrite is where the power is. In the **pattern block** of the rewrite rule we use regular expressions to detect the requested file name or uri and from this we can extract key parts to pass to the **rewrite block**.

Pay attention because this is the hardest part of mod rewrite.

Regular expressions is just a method to detect letters, numbers and symbols using special characters. These special characters are called metacharacters.

Pattern Matching metacharacter Definitions

using special characters. These special characters are called metacharacters.

Pattern Matching metacharacter Definitions

Char.	Definition
\	Use before any of the following characters to escape or null the meaning or it. * \. \$ \+ \[\]
^	Start matching at this point
$	End point of the match
.	Any character
[]	Starts a class
|	Starts alternative match this|that would mean match this or that
()	starts a back reference point
?	match 0 or 1 time Quantifier
+	match atleast 1 or more times Quantifier
*	match 0 to infinite times Quantifier
{}	match minimum to maximum Quantifier {0,3} match up to 3 times

Class Definitions []

Char.	Definition
^	Negates the class. [^A-Z]+ means don't match any uppercases
\	Use before any of the following characters to escape or null the meaning or it. [\+]+
-	Range for matching [0-9]+ [a-zA-Z]+

I'll show a few quick samples just so you understand how to use all of the above. Then we're going to move right on to the **Rewrite Block** since we'll be going over all of this in our basic section.

In this example we just need the numbers in the ulrs below to pass through the mod rewrite to make our query. First we have to ask ourselves, "What is the common pattern in these urls"?

Example 1

In this example there are two common patterns that we can match against. The first one is they all start with category/. The second is they all end in .htm. This should be an easy match

1. category/1.htm
2. category/56.htm
3. category/092340923.htm
4. category/9334.htm

So to use regular expressions to match all of these urls below we need to set our starting point to ^category/.

Now we need to tell the rewrite rule to look for any number 1 or more times. We'll use a character class to do this [0-9]+. Since we need this number to complete our **rewrite block** we're going to tell the mod to reference this so we can use it later. We do this by surrounding the the [0-9]+ with brachets like this ([0-9]+).

To finish the match we're going to negate the . (remember this means any 1 character) even though a . is considered 1 character we're going to go ahead and negate it to read as a dot and then finish the match with htm$.

Mouse over the characters for a definition:

RewriteRule ^**category/**([0-9]+)**.htm**$ /category.php?cat_id=$1 [L]

Example 2

In this example we're going to pass a name through the rewrite. The name we want to use is the name of the first folder. So like before we need to find a

Example 2

In this example we're going to pass a name through the rewrite. The name we want to use is the name of the first folder. So like before we need to find a pattern so we can match and extract the name of the first folder.

1. kitchen-ware/spoons.htm
2. bathware2/towels/duck-patterns.htm
3. dinnerware-pieces/

The only thing we have to work with that is common among all the examples is the trailing slash /. This is kind of tricky since you can type in the 3rd url with out the trailing slash and it would still show up in your browser. We'll get to the trailing slash in a bit though lets start with the collection of the words and numbers before the /.

There are a few ways to do this. We can do a wild card match which picks up everything (.+) or (.*). We can make a class that looks for all numbers, dashes, commas and numbers. ([-a-zA-Z0-9]+) or we can use a negated class which will look for anything but a / like this ([^/]+). We'll use the latter even though all of the above would do the job.

Note: The best to use is the negated class since .+ will pick up a / since a / is defined as any given character. The [-a-zA-Z0-9]+ would just take up too much computing power over the long run. Remember the more you define the more strain there is on the system. Since a search for every thing but a / ([^/]+) requires less computing power it's not only fast it most optimal.

Our final result to pick up everything before the first trailing slash then would look like this ^([^]+)

Next we'll need to account for the possible missing trailing slash. For this we have 2 options the first option is the min max {min,max} metacharacter. If we write /{0,1} this is telling the **rewrite block** to look for a / 0 to 1 times. That would match both dinnerware-pieces/ and dinnerware-pieces every time. But the easier way to do this is to use the ? metacharacter. ? just means match the preceding character 0 or 1 times and we don't have to type as much.

So up to this point our **pattern block** should look like this. ^([^]+)/?

Then we can tack on a $ to the end so we know to stop if the trailing slash is or isn't found. An we get our final rewrite rule below.

Mouse over the characters for a definition

RewriteRule ^([^/]+)/?$ /catalog.php?product_id=$1 [L]

A word of warning if you plan to use the folder names, especially the first folder as a variable that will be passed through the mod you better know that it's going to pass all real files as well through to be rewritten. images/, includes/ css/ img/ cgi-bin/ all of these common folders are perfect matches for ^([^]+)/?$ if this is your first time doing mod rewrite you may want to put your variables in file names instead of 1st tier folders. We go over how to by pass the rewritting of all our static folders in the advanced tutorials. For now just keep this in mind.

It all looks like nonsense, I know I've been here before scratching my head trying to figure it all out. Just **memorize these 3 pattern matches** because you'll use them the most ([0-9]+) , ([^/]+) , (.*) These translate to match any number, match any folder name, or match everything. Becareful with that one though! A RewriteRule ^(.*)$ will shoot a 500 error faster than lightning. Always use .* with another pattern that can be matched like RewriteRule ^(.*).htm$.

A few more things about the pattern block

You cannot use a RewriteRule to match a query string from a dynamic url. RewriteRule is for request_uri matching. A requested uri is in bold below

You cannot use a RewriteRule to match a query string from a dynamic url. RewriteRule is for request_uri matching. A requested uri is in bold below

www.somesite.com**/some/folder/index.php**?id=23&name=foo

You can however get variables from a RewriteCond but we cover how to use RewriteCond together with RewriteRule in the medium tutorials.

Ok that's enough for now. For more information on regular expressions check the on page resources on the right for links to more tutorials.

Rewrite Block

This part is a piece of cake. Now that we've used the pattern block to reference our matches ([0-9]+) we need to rewrite to the url and add the references as needed.

Remember a reference is anything that was picked up in the () in the rewrite.

To call a reference you just add a $ follow by the reference number. This all goes in order like so. Below we'll make 3 references.

RewriteRule ^dir/(.*)/(.*)\.(.htm|.html)$ /$1/$2.$3 [R=301,L]

Rewrites using a 301 redirect
dir/some/folder/file.htm to /some/folder/file.htm

You can mix up the references if you want like so:

RewriteRule ^dir/(.*)/(.*)\.(.htm|.html)$ /**$2/$1**.$3 [R=301,L]

you can also not call a reference like so:

RewriteRule ^dir/(.*)/(.*)\.(.htm|.html)$ /$2/$1.php [R=301,L]

So lets recap a bit. The rewrite block serves 2 purposes. 1 to finalize the total mod rewrite by declare where to rewrite or to redirect. and 2. it allows us to call the backreferences we collect from the **Pattern Block.**

Note: We can use the RewriteBase to set a base directory that we want to rewrite to so you don't always have to write it in your rules.

Example:

```
RewriteBase /dir/

RewriteRule ^somefile-([0-9]+)\.htm$ index.php?id=$1 [L]
```

is the same as

```
RewriteRule ^somefile-([0-9]+)\.htm$ /dir/index.php?id=$1 [L]
```

So if you are doing all your rewites to the same directory save some time and declare you RewriteBase before all your rules. You can even declare / as your base.

Command Flag Block (Optional)

Ok I didn't tell you this is optional because half of you would skip this part. Learning the different Command Flags is a must.

The command flag definitions are as follows:

Char.	Definition
[R]	Redirect you can add an =301 or =302 to change the type.
[F]	Forces the url to be forbidden. 403 header
[G]	Forces the url to be gone 401 header
[L]	Last rule. (You should use this on all your rules that don't link together)

[R] Redirect you can add an =301 or =302 to change the type.
[F] Forces the url to be forbidden. 403 header
[G] Forces the url to be gone 401 header
[L] Last rule. (You should use this on all your rules that don't link together)
[N] Next round. Rerun the rules again from the start
[C] Chains a rewrite rule together with the next rule.
[T] use T=MIME-type to force the file to be a mime type
[NS] Use if no sub request is requested
[NC] Makes the rule case INsensitive
[QSA] Query String Append use to add to an existing query string
[NE] Turns of normal escapes that are default in the rewriterule
[PT] Pass through to the handler (together with mod alias)
[S] Skip the next rule S=3 skips the next 3 rules
[E] E=var sets an enviromental variable that can be called by other rules

See full definitions here.

Ok next is into the tutorials. If you are confused about any of the above don't be scared to move along. We will recap everything so we don't get confused. I know for myself I had to see it work and see the code before I could grasp the full mod rewite experience.