

## Article

[Home](#) » [Server-side Coding](#) » [Apache & IIS Configuration](#) » Learn Apache mod\_rewrite: 13 Real-world Examples



# Learn Apache mod\_rewrite: 13 Real-world Examples

By [DK Lynn](#)

September 26th 2007

Reader Rating: 8.5

[Apache](#) [1]'s low-cost, powerful set of features make it the server of choice for organizations around the world. One of its most valuable treasures is the [mod\\_rewrite module](#) [2], the purpose of which is to rewrite a visitor's request URI in the manner specified by a set of rules.

This article will lead you through rewrite rules, regular expressions, and rewrite conditions, and provide a great list of examples.

First off, I'm going to assume that you understand the common reasons for wanting a URI rewriting feature for your web site. If you'd like information about this field, there's a good primer in the SitePoint article, [mod\\_rewrite: A Beginner's Guide to URL Rewriting](#) [3]. There, you'll also find instructions on how to enable it on your own server.

## Testing Your Server Setup

Some hosts do not have mod\_rewrite enabled (by default it is not enabled). You can find out if your server has mod\_rewrite enabled by creating a [PHP](#) [4] script with one simple line of PHP code:

```
phpinfo();
```

If you load the script with a browser, look in the Apache Modules section. If mod\_rewrite isn't listed there, you'll have to ask your host to enable it -- or find a "good host". Most hosts will have it enabled, so you'll be good to go.

## The Magic of mod\_rewrite

Here's a simple example for you: create three text files named `test.html`, `test.php`, and `.htaccess`.

In the `test.html` file, enter the following:

```
<h1>This is the HTML file.</h1>
```

In the `test.php` file, add this:

```
<h1>This is the PHP file.</h1>
```

Create the third file, `.htaccess`, with the following:

### About the Author

**DK Lynn**



DK Lynn is a former instructor pilot and "rocket scientist" now living in New Zealand where he operates a [small business](#) [developing and hosting web sites](#).

Illustration by: [Alex Walker](#)

RewriteEngine on

```
RewriteRule ^/?test\.html$ test.php [L]
```

Upload all three files (in ASCII mode) to a directory on your server, and type:

```
http://www.example.com/path/to/test.html
```

into the location box -- using your own domain and directory path of course! If the page shows "This is the PHP file", it's working properly! If it shows "This is the HTML file," something's gone wrong.

If your test worked, you'll notice that the `test.html` URI has remained in the browser's location box, yet we've seen the contents of the `test.php` file. You've just witnessed the magic of `mod_rewrite`!

## mod-rewrite Regular Expressions

Now we can begin rewriting your URIs! Let's imagine we have a web site that displays city information. The city is selected via the URI like this:

```
http://www.example.com/display.php?country=USA&state=California&city=San_Diego
```

Our problem is that this is way too long an unfriendly to users. We'd much prefer it if visitors could use:

```
http://www.example.com/USA/California/San_Diego
```

We need to be able to tell Apache to rewrite the latter URI into the former. In order for the `display.php` script to read and parse the query string, we'll need to use regular expressions to tell `mod_rewrite` how to match the two URIs. If you're not familiar with regular expressions ([regex](#) [5]), many sites provide excellent tutorials. At the end of this article, I've listed the best pages I've found on the topic. If you're not able to follow my explanations, I recommend reviewing the first two of those links.

A very common approach is to use the expression `(.*)`. This expression combines two metacharacters: the dot character, which means ANY character, and the asterisk character, which specifies zero or more of the preceding character. Thus, `(.*)` matches everything in the `{REQUEST_URI}` string. `{REQUEST_URI}` is that part of the URI which follows the domain up to but not including the `?` character of a query string, and is the only Apache variable that a rewrite rule attempts to match.

Wrapping the expression in brackets stores it in an "atom," which is a variable that allows the matched characters to be reused within the rule. Thus, the expression above would store `USA/California/San_Diego` in the atom. To solve our problem, we'd need three of these atoms, separated by the subdirectory slashes (`/`), so the regex would become:

```
(.*)/(.*)/(.*)
```

Given the above expression, the regex engine will match (and save) three values separated by two slashes anywhere in the `{REQUEST_URI}` string. To solve our specific problem, though, we'll need to restrict this somewhat -- after all, the first and last atoms above could match anything!

To begin with, we can add the start and end anchor characters. The `^` character matches matching characters at the start of a string, and the `$` character matches characters at the end of a string.

```
^(.*)/(.*)/(.*)$
```

This expression specifies that the whole string must be matched by our regex; there cannot be anything else before or after it.

However, this approach still allows too many matches. We're storing our matches as atoms, and will be passing them to a query string, so we have to be able to trust what we match. Matching anything with `(.*)` is too much of a potential security hazard, and, when used inappropriately, could even cause `mod_rewrite` to get stuck in a loop!

To avoid unnecessary problems, let's change the atoms to specify precisely the characters that we will allow. Because the atoms represent location names, we should limit the matched characters to upper and lowercase letters from A to Z, and because we use it to represent spaces in the name, the underscore character (`_`) should also be allowed. We specify a set using square brackets, and a range using the `-` character. So the set of allowed characters is written as `[a-zA-Z_]`. And because we want to avoid matching blank names, we add the `+` metacharacter, which specifies a match only on one or more of the preceding character. Thus, our regex is now:

```
^([a-zA-Z_]+)/([a-zA-Z_]+)/([a-zA-Z_])$
```

The `{REQUEST_URI}` string starts with a `/` character. Apache changed regex engines when it changed versions, so Apache version 1 requires the leading slash while Apache 2 forbids it! We can satisfy both versions by making the leading slash optional with the expression `^/?` (`?` is the metacharacter for zero or one of the preceding character). So now we have:

```
^/?([a-zA-Z_]+)/([a-zA-Z_]+)/([a-zA-Z_])$
```

With regex in hand, we can now map the atoms to the query string:

```
display.php?country=$1&state=$2&city=$3
```

`$1` is the first (country) atom, `$2` is the second (state) atom and `$3` is the third (city) atom. Note that there can only be nine atoms created, in the order in which the opening brackets appear -- `$1 ... $9` in a regular expression.

We're almost there! Create a new `.htaccess` file with the text:

```
RewriteRule ^/?([a-zA-Z_]+)/([a-zA-Z_]+)/([a-zA-Z_+)$ display.php?country=$1&state=$2&city=$3 [L]
```

Save this to the directory in which `display.php` resides. The rewrite rule must go on one line with one space between the `RewriteRule` statement, the regex, and the redirection (and before any optional flags). We've used the `[L]`, or 'last' flag, which is the terminating flag (more on flags later).

Our rewrite rule is now complete! The atom values are being extracted from the request string and added to the query string of our rewritten URI. The `display.php` script will likely extract these values from the query string and use them in a database query or something similar.

If, however, you have only a short list of allowable countries, it might be best to avoid potential database problems by specifying the acceptable values within the regex. Here's an example:

```
^/?(USA|Canada|Mexico)/([a-zA-Z_]+)/([a-zA-Z_+)$
```

If you're concerned about capitalization because the values in your database are strictly lowercase, you can make the regex engine ignore the case by adding the No Case flag, `[NC]`, after the rewritten URI. Just don't forget to convert the values to lowercase in your script after you obtain the `$_GET` [array](#) [6].

If you want to use numbers (0, 1, ... 9) for, say, Congressional Districts, then you'll need to change an atom's specification from `([a-zA-Z_+)` to `([0-9])` to match a single digit, `([0-9]{1,2})` to match one or two digits (0 through 99), or `([0-9]+)` for one or more digits, which is useful for database IDs.

## The RewriteCond Statement

Now that you've learned how to use `mod_rewrite`'s basic `RewriteRule` statement with the `{REQUEST_URI}` string, it's time to see how we can use conditionals to access other variables with the `RewriteCond` statement. The `RewriteCond` statement is used to specify the conditions under which a `RewriteRule` statement should be applied.

`RewriteCond` is similar in format to `RewriteRule` in that you have the command name, `RewriteCond`, a variable to be matched, the regex, and flags. The logical OR flag, `[OR]`, is a useful flag to remember because all `RewriteCond` and `RewriteRule` statements are inclusive, in the sense of a logical AND relationship, until terminated by the Last, `[L]`, flag.

You can test many server variables with a `RewriteCond` statement. You can find a list in the SitePoint article I mentioned previously, but [this is the best list of server variables](#) [7] I've found.

As an example, let's assume that we want to force the `www` in your domain name. To do this, you'll need to test the Apache `{HTTP_HOST}` variable to see if the `www` is already there and, if it's not, redirect to the desired host name:

```
RewriteCond %{HTTP_HOST} !^www\.example\.com$ [NC]
RewriteRule .* http://www.example.com%{REQUEST_URI} [R=301,L]
```

Here, to denote that `{HTTP_HOST}` is an Apache variable, we must prepend a `%` character to it. The regex begins with the `!` character, which will cause the condition to be true if it doesn't match the pattern. We also have to escape the dot character so that it matches a literal dot and not any character, as is the case with the dot metacharacter. We've also added the No Case flag to make this operation case-insensitive.

The `RewriteRule` will match zero or one of any character, and will redirect to `http://www.example.com` plus the original `{REQUEST_URI}` value. The `R=301`, or redirect, flag will cause Apache to issue a HTTP 301 response, which indicates that this is a permanent redirection; the Last flag tells `mod_rewrite` that you've completed this block statement.

`RewriteCond` statements can also create atoms, but these are denoted with `%1 ... %9` in the same way that `RewriteRule` atoms are denoted with `$1 ... $9`. You'll see these atom variables in operation in the examples later on.

## mod\_rewrite Flags

`mod_rewrite` uses "flags" to give our rewrite conditions and rules additional features. We add flags at the end of a condition or rule using square brackets, and separate multiple flags with a comma. The main flags with which you'll need to be familiar are:

- `last|L` - The Last flag tells Apache to terminate a series of rewrite conditions and rewrite rules in the same way that `}` will terminate a statement block in PHP. Note that this flag does not terminate `mod_rewrite` processing!
- `nocase|NC` - The No Case flag tells Apache to ignore the case of the string in the regex and is often necessary when writing a `RewriteCond` statement that matches

the `{HTTP_HOST}` variable for a domain name, which is not case sensitive.

- `redirect|R` - The Redirect flag is used to trigger an external (visible) redirection. By default, this means that Apache will issue an HTTP 302 response to indicate that the document has been moved temporarily, but you can specify the HTTP code if you like. For example, you could use `[R=301]` to make Apache issue a HTTP 301 response (moved permanently), which is often useful if you need search engines to reindex a changed URI.
- `qsappend|QSA` - The Query String Appended flag is used to "pass through" existing query strings. You can also define a new query string to which the old string will be appended, just be careful not to replicate key names. Failure to use the `QSA` flag will cause the creation of a query string during a redirection to destroy an existing query string.
- `forbidden|F` - The Forbidden flag is used to tell Apache when not to provide a page in response to a request. As a result, Apache will issue a HTTP 403 response, which can be used to protect files from being viewed by unauthorized visitors, [bandwidth](#) [8] leeches, and so on.
- `ornext|OR` - The OR flag allows you to combine rewrite conditions with a logical OR relationship as opposed to the default AND.
- `next|N` - The Next flag tells `mod_rewrite` to restart the rewriting process from the first rule, but to use the URI that was returned from the last processed rewrite rule. This is useful for replacing characters in the `{REQUEST_URI}` when you don't know how many there will be.

You can read about the other available flags at [Apache.org's mod\\_rewrite documentation page](#) [9].

## mod\_rewrite Comments

The `RewriteEngine on` statement is needed at the start of any `mod_rewrite` code, but it is also useful in another role. As a good programmer, you know how important comments are in your code. `mod_rewrite` allows you to comment out an entire block of `mod_rewrite` code by wrapping the code in `RewriteEngine off` and `RewriteEngine on` statements:

```
RewriteEngine off
RewriteCond %{HTTP_HOST} !^www\.example\.com$ [NC]
RewriteRule .? http://www.example.com%{REQUEST_URI} [R=301,L]
RewriteEngine on
```

None of the statements above will be read by the `mod_rewrite` engine. `RewriteEngine` statements can be very helpful when developing new `mod_rewrite` code -- just use them as you would the `/* ... */` wrapper for PHP comments.

## mod\_rewrite Tips

As a webmaster, it's up to you to determine how your pages will be identified to visitors, as well as how to rewrite those URIs so that Apache can serve the appropriate content. Be sure to put some careful consideration into the creation of your new URI scheme. And don't forget that after implementing your new URI scheme, you may have to go back over old content, updating existing links to match the new scheme.

When you design your new URI scheme, make use of unique keys whenever you can. In a previous example, I used countries, states and cities as keys -- items that would be unique in a database. But as I build web sites for clients to update themselves, it's unreasonable for me to insist that they provide unique titles for all their articles. Articles in the database are typically identified by an auto-incremented ID, which would be perfect for my friendly URI scheme. It makes your rewrite rules a lot simpler because you can map a URI atom to a query string value directly.

People often attempt to use a database to redirect from a title or other such field to a specific ID value. `mod_rewrite` has a `RewriteMap` statement for this purpose, but you need to have access to your Apache main configuration file: `httpd.conf`. Typically, you'll only have access to this file if you own and operate the server. Instead, avoid the problem completely, and use the ID field to create your links.

Remember that spaces appear as `%20` in URIs, so you'll need to replace them in your PHP code. PHP's `str_replace` function is perfect for this task. Generally, we need to replace spaces with `%20` when generating links, and convert `%20` back to spaces when reading in query string values from the `$_GET` array. However, when working with unique database field values that contain spaces, I prefer to use the underscore character to replace the spaces in resulting links. To do so, you can use the following PHP code:

```
$name = str_replace ( ' ', '_', $name );
```

Be careful not to break existing relative links when you implement your new URI scheme. Developers are often surprised when suddenly their [CSS](#) [10], [JavaScript](#) [11] files, and images cease to work. Remember that relative links are relative to the current URI at the browser end -- that's the requested URI, not the rewritten one. Depending on your site's configuration, you may need to use absolute URIs, or the HTML `<base>` tag, for all your static resources.

## 13 mod\_rewrite Examples

Earlier, we looked at an example that forced the inclusion of the `www` part of a domain name for every request. Let's have a look at some more examples and see how useful `mod_rewrite` can be.

### 1. Forcing `www` for a domain while preserving subdomains

```
RewriteCond %{HTTP_HOST} ^([a-z.]+)?example\.com$ [NC]
RewriteCond %{HTTP_HOST} !^www\. [NC]
```

```
RewriteRule .? http://www.%1example.com%{REQUEST_URI} [R=301,L]
```

This rule captures the optional subdomain using the `%1` variable, and, if it doesn't start with `www.`, redirects with `www.` prepended to the subdomain. The domain and the original `{REQUEST_URI}` are appended to the result.

## 2. Eliminating `www` from a domain

```
RewriteCond %{HTTP_HOST} !^example\.com$ [NC]
RewriteRule .? http://example.com%{REQUEST_URI} [R=301,L]
```

## 3. Getting rid of the `www` but preserving a subdomain

```
RewriteCond %{HTTP_HOST} ^www\.((([a-z0-9_]+\.)?example\.com)$ [NC]
RewriteRule .? http://%1%{REQUEST_URI} [R=301,L]
```

Here, the subdomain is captured in `%2` (the inner atom) but, since it's optional and already captured in the `%1` variable, all you need is the `%1` for the subdomain.

## 4. Preventing image hotlinking

If some unscrupulous webmasters are leeching your bandwidth by linking to images from your site to post on theirs, you can use the following rule to block the requests:

```
RewriteCond %{HTTP_REFERER} !^$
RewriteCond %{HTTP_REFERER} !^http://(www\.)?example\.com/ [NC]
RewriteRule \.(gif [12]|jpg [13]|png [14])$ - [F]
```

If the `{HTTP_REFERER}` value is not blank, or from your own domain (`example.com`), this rule will block the viewing of URIs ending in `.gif`, `.jpg`, or `.png` using the forbidden flag, `F`.

If you are upset enough at these hotlinkers, you could change the image and let visitors to the site know that you know that they're hotlinking:

```
RewriteCond %{HTTP_REFERER} !^$
RewriteCond %{HTTP_REFERER} !^http://(www\.)?example\.com/.*$ [NC]
RewriteRule \.(gif|jpg|png)$ http://www.example.com/hotlinked.gif [R=301,L]
```

Instead of blocking the URI, the above rule rewrites it to a specific image in our domain. What appears in this image is completely up to your imagination!

You can block specific domains using:

```
RewriteCond %{HTTP_REFERER} !^http://(www\.)?leech_site\.com/ [NC]
RewriteRule \.(gif|jpg|png)$ - [F,L]
```

This rule blocks all requests where the `{HTTP_REFERER}` field is set to the bad domain.

Of course, the above rules rely on the `{HTTP_REFERER}` value being set correctly. It usually is, but if you'd rather rely on the IP Address, use `{REMOTE_ADDR}` instead.

## 5. Redirecting to a 404 page if the directory and file do not exist

If your host doesn't provide for a "file not found" redirection, create it yourself!

```
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule .? /404.php [L]
```

Here, `-f` matches an existing filename and `-d` matches an existing directory name. This script checks to see that the requested filename is not an existing filename or directory name before it redirects to the `404.php` script. You can extend this script: include the URI in a query string by adding `?url=$1` immediately after the URI:

```
RewriteRule ^/(?\.*)$ /404.php?url=$1 [L]
```

This way, your `404.php` script can do something with the requested URL: display it in a message, send it in an email alert, perform a search, and so on.

## 6. Renaming your directories

If you've shifted files around on your site, changing directory names, try this:

```
RewriteRule ^/old_directory/(.*)$ new_directory/$1 [R=301,L]
```

I've included the literal dot character (not the "any character" metacharacter) inside the set to allow file extensions.

## 7. Converting old `.html` links to new `.php` links

Updating your web site but need to be sure that bookmarked links will still work?

```
RewriteRule ^/(?([a-z/]+)\.html$ $1.php [L]
```

This is not a redirection, so it will be invisible to your visitors. To make it permanent (and visible), change the flag to `[R=301,L]`.

## 8. Creating extensionless links

If your site uses PHP files, and you want to make your links easier to remember -- or you just want to hide the file extension, try this:

```
RewriteRule ^/(?([a-z]+)$ $1.php [L]
```

If you have a mixture of both `.html` and `.php` files, you can use `RewriteCond` statements to check whether the filename with either extension exists as a file:

```
RewriteCond %{REQUEST_FILENAME}.php -f
RewriteRule ^/(?([a-zA-Z0-9]+)$ $1.php [L]
RewriteCond %{REQUEST_FILENAME}.html -f
RewriteRule ^/(?([a-zA-Z0-9]+)$ $1.html [L]
```

If the file name exists with the `.php` extension, that rule will be chosen.

## 9. Checking for a key in a query string

If you need to have a specific key's value in your query string, you can check for its existence with a `RewriteCond` statement:

```
RewriteCond %{QUERY_STRING} !uniquekey=
RewriteRule ^/?script_that_requires_uniquekey\.php$ other_script.php [QSA,L]
```

The above code will check the `{QUERY_STRING}` variable for a lack of the key `uniquekey` and, if the `{REQUEST_URI}` is the `script_that_requires_uniquekey`, it will redirect to an alternative URI.

## 10. Deleting the query string

Apache's `mod_rewrite` automatically passes through a query string unless you do either of the following:

- Assign a new query string (you can keep the original query string by adding a QSA flag, e.g., `[QSA,L]`).
- Add a `?` after a filename (for example, `index.php?`). The `?` will not be shown in the browser's location field.

## 11. Redirecting a working URI to a new format

Here's a curly one. Let's say, for example, that we've got a set of working URLs that look like this: `/index.php?id=nnnn`. However, we'd really like to change them to `/nnnn` and make sure search engines update their indexes to the new URI format. First, we'd have to redirect the old URIs to the new ones so that search engines update their indexes, but we'd still have to rewrite the new URI back to the old one so that the `index.php` script would run. Have I got your head spinning?

The trick here is to place into the query string a marker code that will not be seen by visitors. We redirect from the old link to the new format only if the "marker" is not present in the query string. Then we rewrite the new format link back to the old format, and add a marker to the query string, using the QSA flag to ensure we're not eliminating an existing query string. Here's how it's done:

```
RewriteCond %{QUERY_STRING} !marker
RewriteCond %{QUERY_STRING} id=([-a-zA-Z0-9_+])
RewriteRule ^/?index\.php$ %1? [R=301,L]

RewriteRule ^/(?([-a-zA-Z0-9_+])$ index.php?marker&id=$1 [L]
```

Here, the original URI, `http://www.example.com/index.php?id=nnnn`, does not contain the marker, so it's redirected by the first rule to `http://www.example.com/nnnn` with a HTTP 301 response. The second rule rewrites `http://www.example.com/nnnn` back to `http://www.example.com/index.php?marker&id=nnnn`, adding `marker` and `id=nnnn` in a new query string; then, the `mod_rewrite` process is started over.

In the second iteration, the marker is matched so the first rule is ignored and, since there's a dot character in `index.php?marker&id=nnnn`, the second rule is also ignored ... and we're finished!

Note that, while useful, this solution does require additional processing by Apache, so be careful if you're using it on shared servers with a lot of traffic.

## 12. Ensuring that a secure server is used

Apache can determine whether you're using a secure server in two ways: using the `{HTTPS}`, or `{SERVER_PORT}`, variables:

```
RewriteCond %{REQUEST_URI} ^secure_page\.php$
RewriteCond %{HTTPS} !on
RewriteRule ^/(?(secure_page\.php)$ https://www.example.com/$1 [R=301,L]
```

The above example tests that the `{REQUEST_URI}` value is equal to our secure page script, and that the `{HTTPS}` value is not equal to on. If both these conditions are met, the request is redirected to the secure server URI. Alternatively, you could do the same thing by testing the `{server_port}` value, where 443 is typically the secure server port:

```
RewriteCond %{REQUEST_URI} ^secure_page\.php$
RewriteCond %{SERVER_PORT} !^443$
RewriteRule ^/(?(secure_page\.php)$ https://www.example.com/$1 [R=301,L]
```

## 13. Enforcing secure server only on selected pages

In situations where secure and unsecured domains share the web server's `DocumentRoot` directory, you'll need a `RewriteCond` statement to check that the secure server port isn't being used, and then only redirect the request if the requested script is one in the list of those that require a secure server:

```
RewriteCond %{SERVER_PORT} !^443$
RewriteRule ^/(?(page1|page2|page3|page4|page5)$ https://www.example.com/%1 [R=301,L]
```

Here's how you'd redirect requests for pages not requiring a secure server back to port 80:

```
RewriteCond %{SERVER_PORT} ^443$
RewriteRule !^/(?(page6|page7|page8|page9)$ http://www.example.com%{REQUEST_URI} [R=301,L]
```

## Summary

Apache `mod_rewrite` is primarily used to allow [SEO](#) [15] and user friendly URIs, but it's also an extremely flexible tool for other important redirection tasks. If you want to learn more, here are some very useful resources I've found:

### Regular Expressions

- Great tutorial: [http://gnosis.cx/publish/programming/regular\\_expressions.html](http://gnosis.cx/publish/programming/regular_expressions.html) [16]
- Cheat sheet: <http://regexlib.com/CheatSheet.aspx> [17]
- A regex-capable text editor: <http://www.editpadpro.com> [18]
- Regex Coach: <http://weitz.de/regex-coach/> [19]

### mod\_rewrite

- Cheat sheet: [http://www.ilovejackdaniels.com/cheat-sheets/mod\\_rewrite-cheat-sheet/](http://www.ilovejackdaniels.com/cheat-sheets/mod_rewrite-cheat-sheet/) [20]

[Back to SitePoint.com](#)

[1] /glossary.php?q=A#term\_19

[2] [http://httpd.apache.org/docs/2.0/mod/mod\\_rewrite.html](http://httpd.apache.org/docs/2.0/mod/mod_rewrite.html)

[3] <http://www.sitepoint.com/article/guide-url-rewriting/>

[4] /glossary.php?q=P#term\_1

[5] /glossary.php?q=%23#term\_33

[6] /glossary.php?q=%23#term\_72

[7] <http://www.php.net/reserved.variables#reserved.variables.server>

[8] /glossary.php?q=B#term\_56

[9] [http://httpd.apache.org/docs-2.0/mod/mod\\_rewrite.html](http://httpd.apache.org/docs-2.0/mod/mod_rewrite.html)

[10] /glossary.php?q=C#term\_8

[11] /glossary.php?q=J#term\_9

[12] /glossary.php?q=G#term\_24

[13] /glossary.php?q=J#term\_23

[14] /glossary.php?q=P#term\_26

[15] /glossary.php?q=S#term\_64

[16] [http://gnosis.cx/publish/programming/regular\\_expressions.html](http://gnosis.cx/publish/programming/regular_expressions.html)

[17] <http://regexlib.com/CheatSheet.aspx>

[18] <http://www.editpadpro.com>

[18] <http://www.editpadpro.com>

[19] <http://weitz.de/regex-coach/>

[20] [http://www.ilovejackdaniels.com/cheat-sheets/mod\\_rewrite-cheat-sheet/](http://www.ilovejackdaniels.com/cheat-sheets/mod_rewrite-cheat-sheet/)