

[Advertise Here](#)

# An In Depth Guide to mod\_rewrite for Apache

[Joseph Pecoraro](#) on Apr 26th 2011 with [82 comments](#)

## Tutorial Details

- 
- **Subject:** Apache Mod Rewrite
- **Difficulty** - Intermediate to Advanced

Twice a month, we revisit some of our readers' favorite posts from throughout the history of Nettuts+. This tutorial was first published last September.

When people think of `.htaccess` configuration, the first thing that might pop into their minds is URL manipulation with `mod_rewrite`. But they're often frustrated by `mod_rewrite`'s complexity. This tutorial will walk you through everything you need to know for the most common `mod_rewrite` tasks.

## Mod\_rewrite Rants

Thoughts on `mod_rewrite` vary quite a bit. To gain a quick feel for what the world thinks, I just ran a Twitter search on "mod\_rewrite". Here's a sample of what was returned.

[mldk](#): Aargh! `.htaccess` and `mod_rewrite` can be such a pain in the —!

[bsterzenbach](#): Man do I love `mod_rewrite`. I could work with it the rest of my life and still not master it — so powerful

[mikemackay](#): Still loving the total flexibility of `mod_rewrite` — coming to the rescue again. Often so

overlooked...and easier than you might think too!

[hostpc](#): I hate mod\_rewrite. Can't get this dang application to work properly :(

[awanderingmind](#): Oh WordPress and Apache, how thou dost vex me. Mod\_rewrite be damned!

[danielshiding](#): Why won't mod\_rewrite work! Damn it!

A few things I noticed are that people clearly recognize the power of `mod_rewrite`, but are often frustrated by the syntax. That's not surprising, considering the front page of [Apache's mod\\_rewrite documentation](#) says essentially the same thing:

Despite the tons of examples and docs, `mod_rewrite` is voodoo. Damned cool voodoo, but still voodoo.”  
— [Brian Moore](#)

What a turn off! So, in this article, I'm really going to bring things down a notch. We'll address not only `mod_rewrite`'s syntax, but I'll also provide a workflow that you can use to debug and solve your `mod_rewrite` problems. Along the way, we'll review a few useful real-world examples.

Before we begin, a note of caution: as with many subjects – this one in particular – you won't learn unless you try on your own! That is one of the reasons why I'm going to focus on teaching a debug workflow. As usual I'll demonstrate how to get your system setup if you don't already have the module loaded. I urge you to work through the examples on your own server – preferably, in a test environment.

---

## What is mod\_rewrite?

`mod_rewrite` is an Apache module that allows for server-side manipulation of requested URLs.

`mod_rewrite` is an Apache module that allows for server-side manipulation of requested URLs. Incoming URLs are checked against a series of rules. The rules contain a regular expression to detect a particular pattern. If the pattern is found in the URL, and the proper conditions are met, the pattern is replaced with a provided substitution string or action. This process continues until there are no more rules left or the process is explicitly told to stop.

This is summarized in these three points:

- There are a list of rules that are processed in order.
- If a rule matches, it checks the conditions for that rule.
- If everything is a go, it makes a substitution or action.

---

## Advantages of mod\_rewrite

There are some obvious advantages to using a URL rewriting tool like this, but there are others that might not be as obvious.

`mod_rewrite` is most commonly used to transform ugly, cryptic URLs into what are known as “friendly URLs” or “[clean URLs](#).”

As an added bonus, these URLs are also more search engine friendly. Consider the following example:

[view plain](#)[copy to clipboard](#)[print?](#)

1. Not so friendly: `http://example.com/user.php?id=4512`
2. Much friendlier: `http://example.com/user/4512/`
3. Even better: `http://example.com/user/Joe/`

Not only is the final link easier on the eyes, it's also possible for search engines to extract semantic meaning from it. This basic kind of URL rewriting is one way that `mod_rewrite` is used. However, as you will see, it can do a whole lot more than just these simple transformations.

Expanding on the same example, some people claim there are security benefits to having `mod_rewrite` transform your URLs. Given the same example, imagine the following attack on the user id:

[view plain](#)[copy to clipboard](#)[print?](#)

1. `http://example.com/user.php?id=AHHHHHH`
- 2.
3. `http://example.com/user/AHHHHHHH/`

In the first example, the PHP script is explicitly being invoked and must handle the invalid `id` number. A poorly written script would likely fail, and, in a more extreme case (in a poorly written web application), bad input could cause data corruption. However, if the user is only ever shown the friendlier URLs, they would never know that the `user.php` page existed.

Trying the same attack in that case would likely fail before it even reaches the PHP script. This is because, at the core of `mod_rewrite` is regular expression pattern matching. In the example case above, you would have been expecting a number, for example `(\d+)`, not characters like `a-z`. This extra layer of abstraction is nice from a security perspective.

---

## Enabling mod\_rewrite on the Server

Enabling `mod_rewrite` or any apache module must be done from the global configuration file (`httpd.conf`).

Just like enabling `.htaccess` support, enabling `mod_rewrite` or any Apache module must be done from the global configuration file (`httpd.conf`). Just as before, since `mod_rewrite` usage is so widespread, hosting companies almost always have it enabled. However, if you suspect that your hosting company does not – and we will test for that below – contact them and they will likely enable it.

If you rolled your own Apache installation, it's worth noting that `mod_rewrite` needs to be included when compiled, as it is not done so by default. However, it's so common that nearly all installation guides, including [Apache's](#) show how in their example.

If you're the administrator for your web server, and you want to make sure that you load the module, you should look in the `httpd.conf` file. In the configuration file, there will be a large section which loads a bunch of modules. The following line will likely appear somewhere within the file. If it is, great! If it's commented out, meaning there is a `#` symbol at the start of the line, then uncomment it by removing the `#`:

1. `LoadModule rewrite_module modules/mod_rewrite.so`

Older versions of Apache 1.3 may require you to add the following directive after the `LoadModule` directive.

1. `# Only in Apache 1.3 AddModule mod_rewrite.c`

However, this seems to have [disappeared in Apache 2](#) and later. Only the `LoadModule` directive is required.

If you had to modify the configuration file at all (not likely), then you will need to restart the web server. As always, you should remember to make a backup of the original file in case you need to revert back to it later.

## Testing for mod\_rewrite

You can test if `mod_rewrite` is enabled/working in a number of ways. One of the simplest methods is to view the output from PHP's `phpinfo` function. Create this very simple PHP page, open it in your browser, and search for `"mod_rewrite"` in the output.

1. `<?php phpinfo(); ?>`

`mod_rewrite` should show up in the "Loaded Modules" section of the page like so:

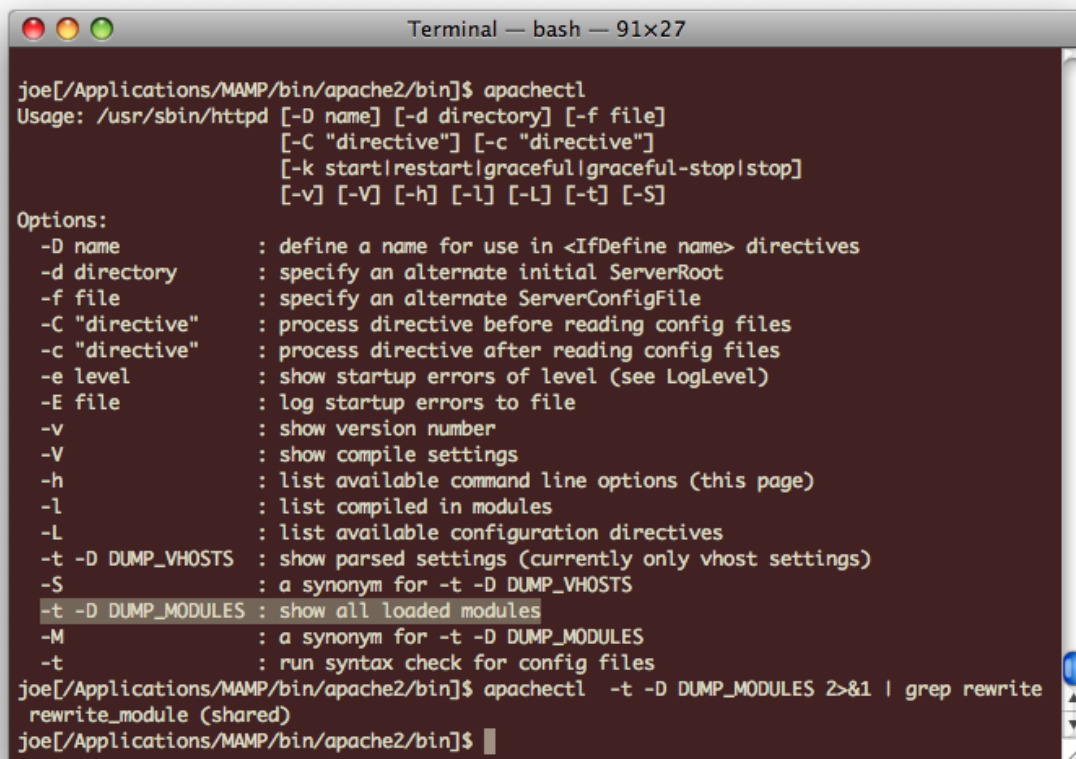
apache2handler	
Apache Version	Apache/2.0.59 (Unix) PHP/5.2.3 DAV/2
Apache API Version	20020903
Server Administrator	you@example.com
Hostname:Port	localhost:8888
User/Group	joe(501)/-1
Max Requests	Per Child: 0 - Keep Alive: on - Max Per Connection: 100
Timeouts	Connection: 300 - Keep-Alive: 15
Virtual Server	No
Server Root	/Applications/MAMP/Library
Loaded Modules	core prefork http_core mod_so mod_access mod_auth mod_auth_anon mod_auth_dbm mod_auth_digest mod_file_cache mod_echo mod_charset_lite mod_cache mod_disk_cache mod_mem_cache mod_example mod_case_filter mod_case_filter_in mod_ext_filter mod_include mod_deflate mod_log_config mod_env mod_mime_magic mod_cern_meta mod_expires mod_headers mod_usertrack mod_setenvif mod_proxy proxy_connect proxy_ftp proxy_http mod_bucketeer mod_mime mod_dav mod_status mod_autoindex mod_asis mod_info mod_cgi mod_cgid mod_dav_fs mod_vhost_alias mod_negotiation mod_dir mod_imap mod_actions mod_speling mod_userdir mod_alias mod_rewrite mod_php5

If you're not using PHP (although I will for the rest of the tutorial), there are some other ways to check. Apache comes with a number of command line tools that you can refer to. You can also use other tools, like `apachectl` or `httpd` to directly test for the module. There are command line switches that allow you to check all of the loaded modules in the existing installation. You can execute the following to get a listing of all of the loaded modules.

[view plaincopy to clipboardprint?](#)

1. `shell> apachectl -t -D DUMP_MODULES`

This command will display the “help” page for the command. I then run the command and search for “rewrite” in the results and it shows there was a line of output that matched!



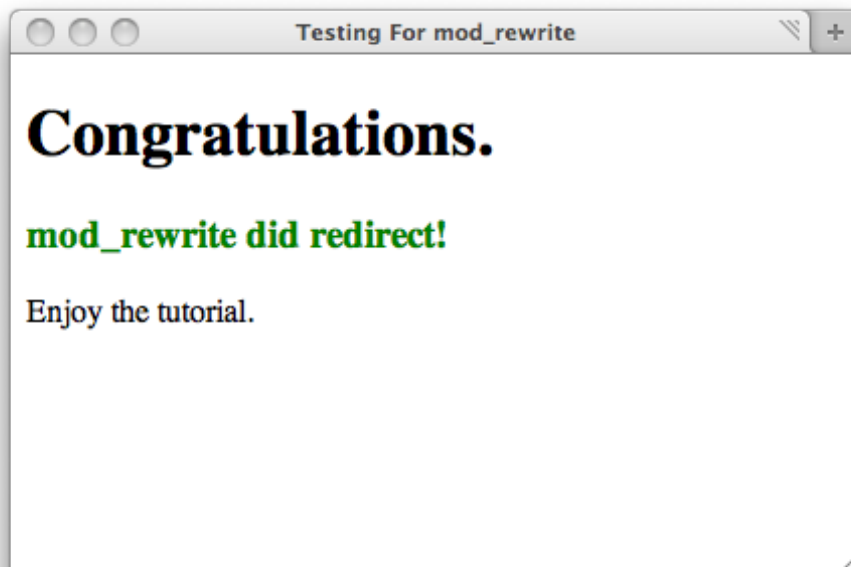
```
Terminal — bash — 91x27

joe[/Applications/MAMP/bin/apache2/bin]$ apachectl
Usage: /usr/sbin/httpd [-D name] [-d directory] [-f file]
                        [-C "directive"] [-c "directive"]
                        [-k start|restart|graceful-stop|stop]
                        [-v] [-V] [-h] [-l] [-L] [-t] [-S]

Options:
  -D name           : define a name for use in <IfDefine name> directives
  -d directory      : specify an alternate initial ServerRoot
  -f file           : specify an alternate ServerConfigFile
  -C "directive"    : process directive before reading config files
  -c "directive"    : process directive after reading config files
  -e level          : show startup errors of level (see LogLevel)
  -E file           : log startup errors to file
  -v               : show version number
  -V               : show compile settings
  -h               : list available command line options (this page)
  -l               : list compiled in modules
  -L               : list available configuration directives
  -t -D DUMP_VHOSTS : show parsed settings (currently only vhost settings)
  -S               : a synonym for -t -D DUMP_VHOSTS
  -t -D DUMP_MODULES : show all loaded modules
  -M               : a synonym for -t -D DUMP_MODULES
  -t               : run syntax check for config files
joe[/Applications/MAMP/bin/apache2/bin]$ apachectl -t -D DUMP_MODULES 2>&1 | grep rewrite
rewrite_module (shared)
joe[/Applications/MAMP/bin/apache2/bin]$
```

Finally, if you are still unsure if it’s enabled, just give it a shot! The following `.htaccess` file will redirect any request in the given folder to the `good.html` file. That means, if `mod_rewrite` is working, you should see `good.html`.

1. # Redirect everything in this directory to "good.html"
2. RewriteEngine on RewriteRule .\* good.html





---

## Inside .htaccess

As always, anything that you can put in a `.htaccess` file can also be placed inside the global configuration file. With `mod_rewrite`, there is a small differences if you put a rule in one or the other. Most notably:

If you're putting [...] rules in an `.htaccess` file [...] the directory prefix (/) is removed from the `REQUEST_URI` variable, as all requests are automatically assumed to be relative to the current directory.  
– [Apache Documentation](#)

This is something to keep in mind if you see examples online or if you're trying an example yourself: beware of the leading slash. I will attempt to clarify this below when we work through some examples together.

---

## Regular Expressions

This tutorial does not intend to teach you regular expressions. For those of you who are familiar with them, the regular expressions used in `mod_rewrite` seem to vary between versions of Apache. In Apache 2.0 they're [Perl Compatible Regular Expressions \(PCRE\)](#). This means that many of the shortcuts you are used to, such as `\w` referring to `[A-Za-z0-9_]`, `\d` referring to `[0-9]`, and much more *do* exist. However, my particular hosting company uses Apache 1.3 and the regular expressions are more limited.

## Helpful RegEx Resources

If you don't know regular expressions here are some useful tutorials that will bring you up to speed quickly.

- [Jeffrey's Crash Course](#)

- [The Absolute Bare Minimum Every Programmer Should Know About Regular Expressions](#)
- [Quick And Practical Tutorial](#)
- [Smashing Magazine Links on Regular Expressions](#)

And a few references that everyone should know about:

- [Popular Added Bytes Cheatsheet For Regular Expressions](#)
- [Added Bytes Cheatsheet for mod\\_rewrite](#)
- [Explain Regular Expressions](#)

If you haven't yet taken the time to learn regular expressions, I highly suggest doing so. It's an incredibly helpful tool to have. As is usually the case, they are not quite as complex as some might think. I selected the links above from my years of experience working with regular expressions. I feel that these guides do a very good job of getting the basics across.

Regular expression knowledge is a necessity if you want to effectively use mod\_rewrite.

---

## Getting a Feel for it.

Okay, you've waited patiently enough; let's run through a quick example. This is included in the linked source files. Here is the code from the `.htaccess` file:

[view plain](#)[copy to clipboard](#)[print?](#)

```
1. # Enable Rewriting
2. RewriteEngine on
3.
4. # Rewrite user URLs
5. # Input: user/NAME/
6. # Output: user.php?id=NAME
7. RewriteRule ^user/(\w+)/?$ user.php?id=$1
```

Before I can explain any of the code above, we should quickly review the other files in the directory.

The directory contains an `index.php` and a `user.php` file. The index only has some links, of various formats, to the user page. The PHP code is used purely for debugging purposes to confirm that the page was accessed and what the given "id" parameter contained. Here is the contents of `user.php`:

[view plain](#)[copy to clipboard](#)[print?](#)

```
1. <?php
2.
3. // Get the username from the url
4. $id = $_GET['id'];
5.
6. ?><!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
7. "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
8. <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
9. <head>
```



```

10. <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
11. <title>Simple mod_rewrite example</title>
12. <style type="text/css"> .green { color: green; } </style>
13. </head>
14. <body>
15. <h1>You Are on user.php!</h1>
16. <p>Welcome: <span class="green"><?php echo $id; ?></span></p>
17. </body>
18. </html>

```

This example has a few different section. First, notice that URL Rewriting must be enabled via the `RewriteEngine` directive! If your `.htaccess` file is going to use rewrite rules, you should always include this line. Otherwise, you can't be sure if its enabled or not. As a rule of thumb, always include it. The string "on" is case insensitive.

The first `RewriteRule` is for handling the `user.php` page. As the comments indicate, we are rewriting the friendly URL into the format of the normal URL. To do so, when the friendly URL comes in as input, we are actually transforming it into the standard query string URL. Breaking it down we get:

[view plaincopy to clipboardprint?](#)

1. The Rule:
2. `RewriteRule ^user/(\w+)/?$ user.php?id=$1`
- 3.
4. Pattern to Match:
5. `^` Beginning of Input
6. `user/` The `REQUEST_URI` starts with the literal string "user/"
7. `(\w+)` Capture any word characters, put in \$1
8. `/?` Optional trailing slash "/"
9. `$` End of Input
- 10.
11. Substitute with:
12. `user.php?id=` Literal string to use.
13. `$1` The first (capture) noted above.

Here are some examples and an explanation for each:

User.php				
Incoming	Match	Capture	Outgoing	Result
user.php?id=joe	No		user.php?id=joe	Normal
user/joe	Yes	joe	user.php?id=joe	Good
user/joe/	Yes	joe	user.php?id=joe	Good
user/joe/x	No		user/joe/x	Fail

The first example goes through unaffected by the `RewriteRule` and works just fine. The second and third examples match the `RewriteRule`, are rewritten accordingly and end up working fine, as well. The last example does not match the rule and proceeds untouched. The server doesn't have a `user` directory and fails trying to find it. This is as expected, because `user/joe/x` is a bad URL in the first place!

This example was rather easy to understand. However, that said, there were a lot of minute details that I glossed over.



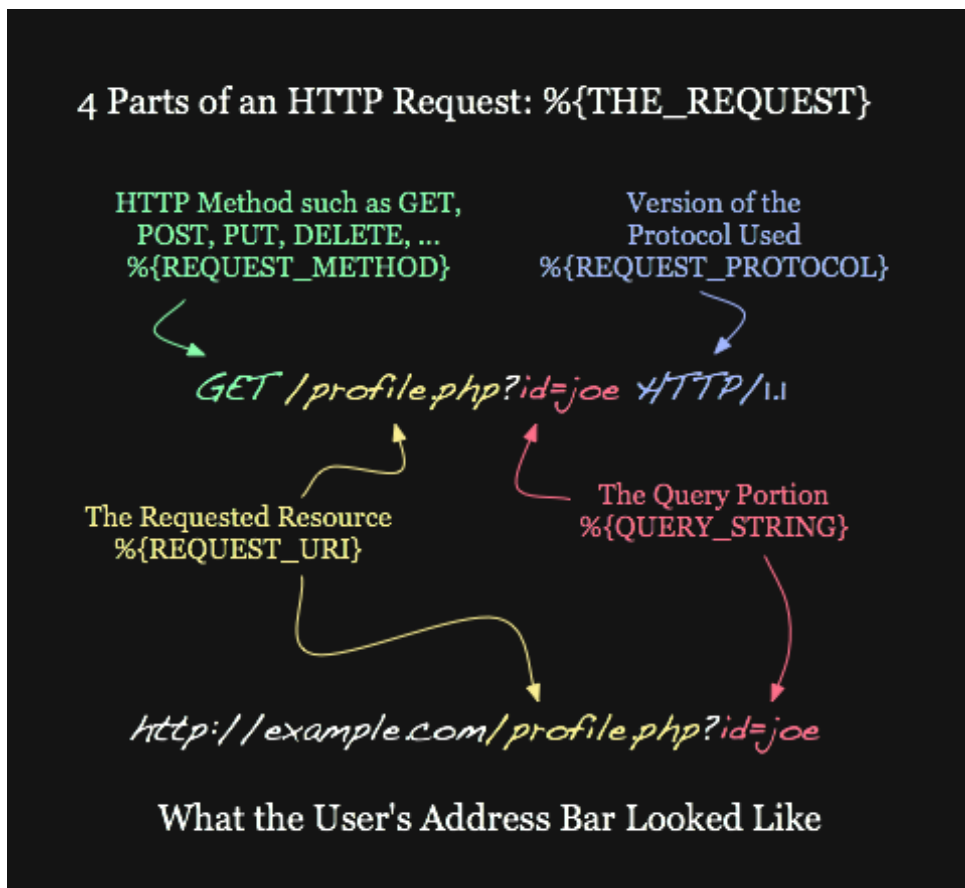
To execute more complex scripts, we should clarify exactly what is happening above. In the following section, I'm going to walk through every step in the cycle.

**NOTE:** If this example above didn't work for you, it's possible that your Apache or mod\_rewrite versions are not PCRE compatible. Try changing `^user/(\w+)/?$` into `^user/([a-z]+)/?$`. Notice that I did not use the `\w` shorthand. If this version works for you, then you will have to avoid the regex shortcuts and instead use their longer equivalents (see the Regular Expressions section above).

## Flow of Execution in Detail

The flow of execution through the rewrite rules is simple, though not exactly straight forward. So, I'm going to break it down into painful detail.

It all begins with the user making a request to your server. They type a URL into their browser's address bar, their browser translates that into an HTTP request to send to the server, Apache receives that request, and then parses it into pieces. Here is an example:



Note that whenever I mention one of Apache's variables, I use an odd looking syntax: `%{APACHE_VAR}`. I only do so because its similar to the syntax that mod\_rewrite uses to access its variables. However, it is the name inside the braces that is important.

So what part does mod\_rewrite deal with? If you're working inside a `.htaccess` file, then you're working with the `REMOTE_URI` portion but *without the leading slash*. I made of note of this before; it tends to be something that is very confusing for most people when they start out. If you're working from inside the global configuration file, however, then

you would leave the leading slash in.

To be as specific as possible, buried in the [Apache Documentation](#) is this description of the “URL Part” that `mod_rewrite` acts on:

The Pattern is always a regular expression matched against the URL-Path of the incoming request (the part after the hostname but before any question mark indicating the beginning of a query string). [Apache Documentation](#)

To remove any ambiguity, highlighted in gold in these two URLs below is the “URL Part” that `mod_rewrite` acts on inside a `.htaccess` file:

#### Incoming:

```
http://example.com/profile/joe/  
http://example.com/profile.php?id=joe
```

#### HTTP Request Equivalents:

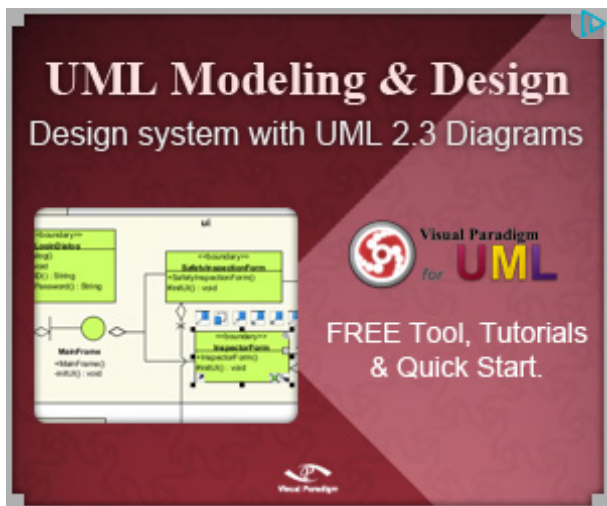
```
GET /profile/joe/ HTTP/1.1  
GET /profile.php?id=joe HTTP/1.1
```

#### URL used by mod\_rewrite in gold:

```
GET /profile/joe/ HTTP/1.1  
GET /profile.php?id=joe HTTP/1.1
```

NOTE: This is different then `%{REQUEST_URI}` because this does not include the leading slash!

For the rest of this section, I’ll be using these two URLs to describe the flow of execution. I’ll also refer to the first url as the “green” URL and the second as the “blue” URL. I will be using “URL Part” throughout this analysis, meaning the `REMOTE_URI` without the leading slash.



## URL vs. URI

For those pedantic readers, these two things that I am calling URLs are actually URIs. The definition of a Uniform Resource Identifier (URI) differs from a Uniform Resource Locator (URL).

- **URI:** An indicator of where a resource is. This means that multiple URIs can point to the same resource but are themselves different addresses. Following a URI might take any number of hops or redirections until it actually arrives at the resource.
- **URL:** a stricter term that identifies the exact location of a resource. This subtle difference has blurred over time such that nobody cares about the difference. I will continue to use the term URL, because people are more comfortable with it.

Now, we know what the rewrite rules are going to be acting on. Once Apache has parsed the request, it translates that to the file it thinks is needed and proceeds to fetch that file. At this point, it will traverse directories and encounter the `.htaccess` files. Assuming this file enables the `RewriteEngine`, any `RewriteRule` could change the URL. A drastic enough change (such as one that points Apache to another directory instead of the original directory it was heading toward) will cause Apache to issue a sub-request and proceed to fetch the new file.

In most cases, sub-requests are invisible to you.

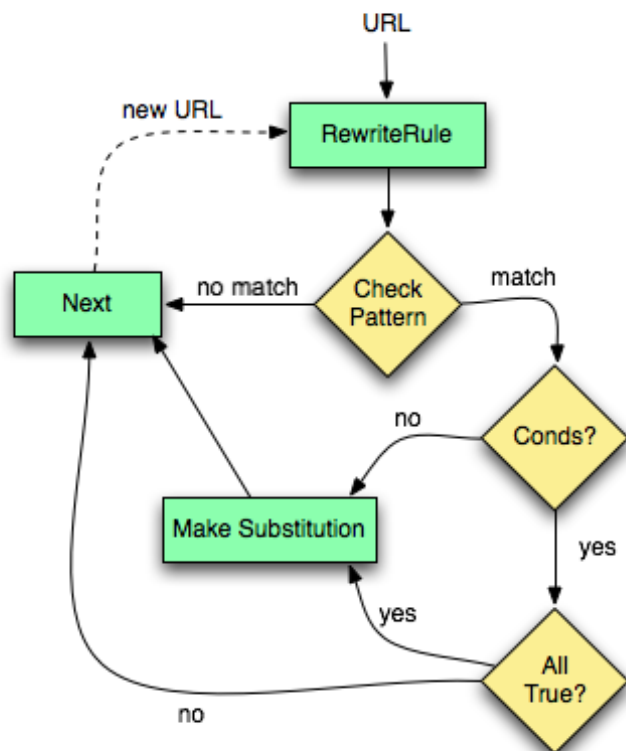
In most cases, sub-requests are invisible to you. This implementation detail is not important to know for the majority of the simple rewrites that you will ever write or use. What is more important to know is how Apache processes the rewrite rules inside a `.htaccess` file.

The rules in a `.htaccess` file are processed in the order that they appear. Note that each `RewriteRule` is acting on the “URL Part” that is similar to the `REMOTE_URI`. When a rule makes a substitution, the modified “URL Part” will be handed to the next rule. This means that the URL that a rule is processing may have been edited by a previous rule! The URL is continually being updated by each rule that it matches. This is important to remember!

## Flow Chart

Here is a flow chart that tries to provide a visualization of the generic flow of execution across multiple rules in a

.htaccess file:



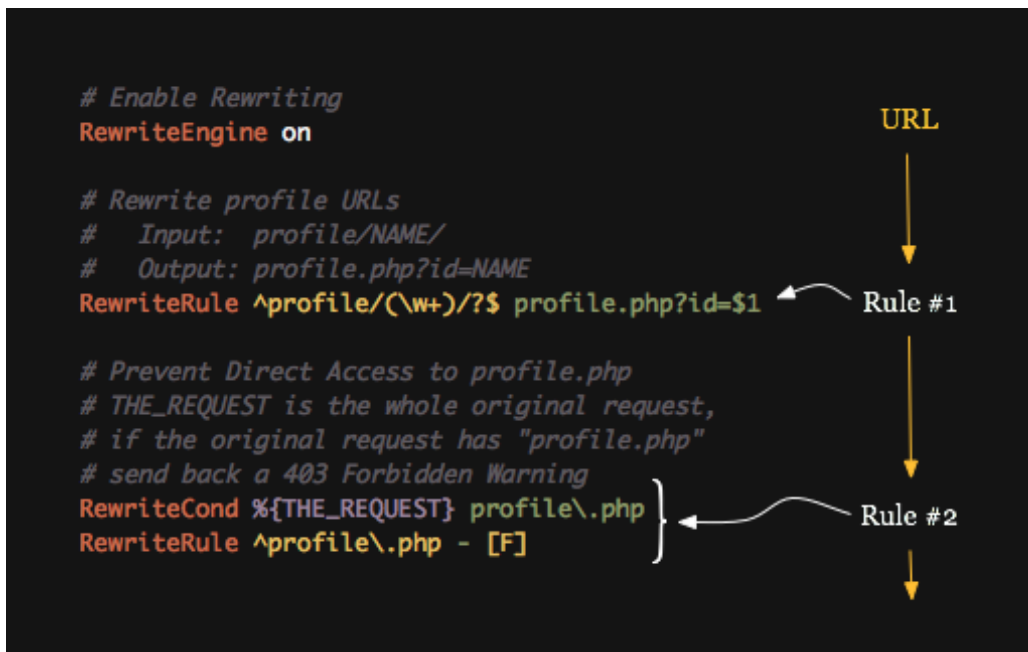
Note that, at the top of the flow chart, the value going into the rewrite rules is that “URL Part” and if the substitution is successful, the modified part proceeds into the next rule.

Each `RewriteCond` is associated with a single `RewriteRule`.

I referred to rewriting conditions earlier, but didn’t go into detail. Each `RewriteCond` is associated with a single `RewriteRule`. The conditions appear before the rule they are associated with one another, but only get evaluated if the rule’s pattern matched. As the flow chart illustrates, if a rewrite rule’s pattern matches, then Apache will check to see if there are any conditions for that rule. If there aren’t, then it *will* make the substitution and continue. If there are conditions, on the other hand, then it will only make the substitution if *all* of the conditions are true. Let’s visualize this in a concrete example.

The URLs that I’m working with are part of the “Profile Example” that I’ve included in the source code download in the “profile\_example” directory. This is similar to the previous example with the `user.php` but it now has a `profile.php` page, an added rewrite rule, and a condition!

Let’s take a look at the code and Apache’s flow of execution through it:



Here, there are two rules. Rule #1 is the same as the user example we reviewed previously. Rule #2 is new; notice that it has a condition. The “URL Part” we have been discussing goes through the rules in order, top to bottom.

The key to understanding this example is to first understand the goal. I am going to allow friendly profile URLs, but I’m actually going to explicitly forbid access to the PHP page directly. Note, some people might say argue that this is a bad idea. They might say that, as a developer this will make things harder for you to debug. That’s certainly true; I don’t actually recommend doing a trick like this, but it makes for an excellent example! More practical uses for `mod_rewrite` will show up later in this tutorial.

With that in mind, let’s see what happens with our green URL. We want this one to be successful.

THE\_REQUEST: *GET /profile/joe/ HTTP/1.1*

*profile/joe/*

Rule #1:

Match?	<i>^profile/(\w+)/?\$</i>	✓
Conds?	<i>none</i>	✓
Make Substitution		✓
Flags?	<i>none</i>	✓

The substitution has been made and the new URL gets passed to the next rule.

*profile.php?id=joe*

The "?id=joe" portion is hidden in the background. The next rule deals only with the green part.

Rule #2:

Match?	<i>^profile\.php</i>	✓
Conds?	<i>yes</i>	✓
Check THE_REQUEST		✗
Make Substitution		✗
Flags?	<i>skip</i>	✗

*profile.php?id=joe*

At the top, you'll see Apache's `THE_REQUEST` variable. I put this at the top because, unlike many of the Apache variables we will deal with, during the duration of the request, this variable's value will *never change*! That is one of the reasons why Rule #2 uses `%{THE_REQUEST}`. Underneath `THE_REQUEST`, we see the green "URL Part" going into the first rule:

- The URL matches the pattern.
- There are no conditions, so it continues.
- The substitution is made.
- There are no flags, so it continues.

After making it through the first rule, the URL has changed. The total URL has been rewritten to `profile.php?id=joe`, which Apache then breaks down and updates many of its variables. The `?id=joe` portion gets hidden from us and `profile.php`, the new “URL Part”, continues into the second rule. This is our first encounter with conditions:

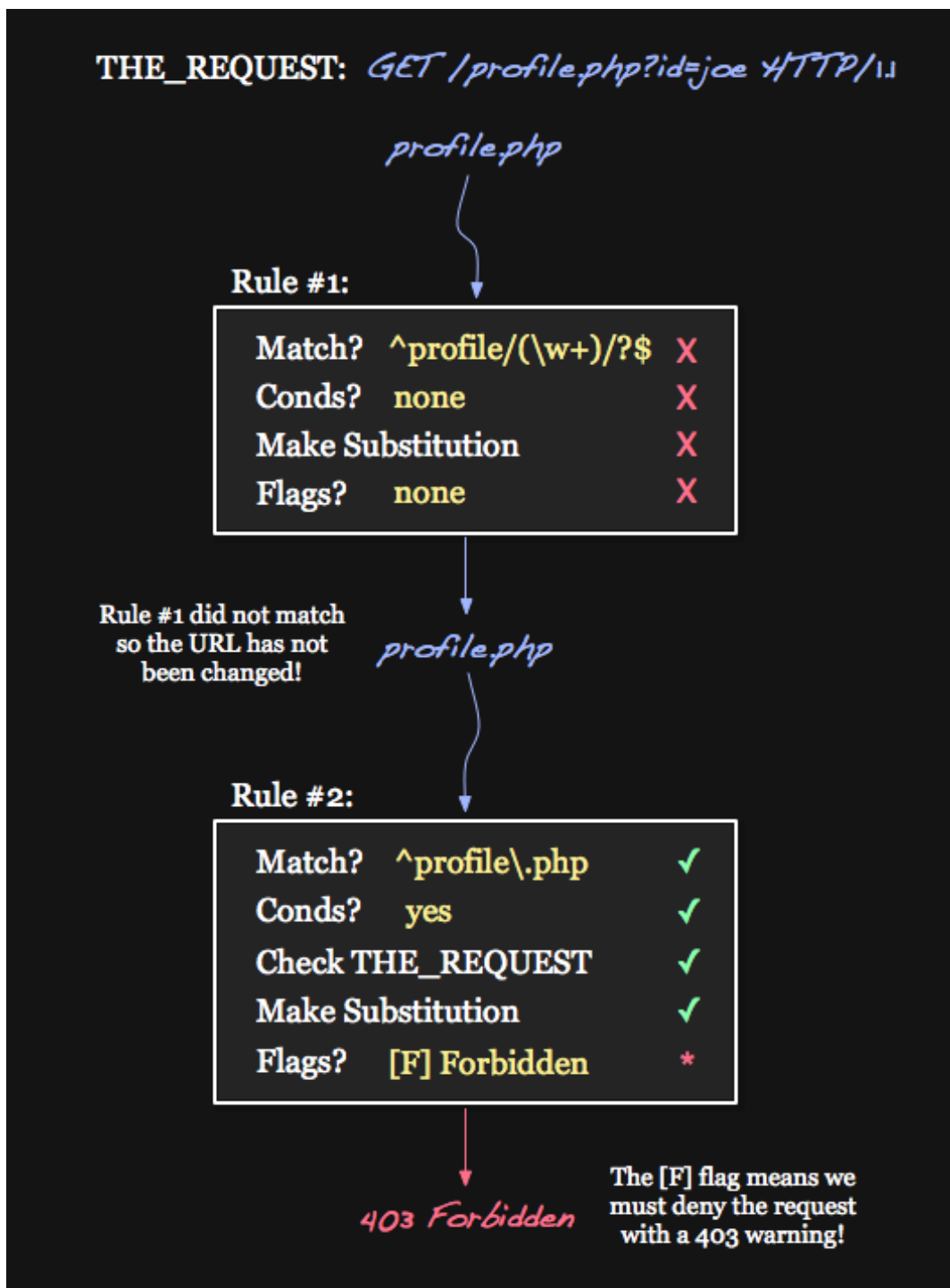
- The URL matches the pattern.
- There are conditions, so we will try the conditions.
- `THE_REQUEST` does not contains `profile.php`, so the condition fails.
- Because a condition failed, we ignore the substitution and flags.
- The URL is unchanged by this rule.

At this point, we made it through all the rewrites and the `profile.php?id=joe` page will be fetched properly.

---

Here is how the execution looks for the blue URL – the one we want to fail:





Again I place the `THE_REQUEST` value at the top. The blue “URL Part” enters Rule #1:

- The URL does not match the pattern.
- Everything else is ignored and the URL proceeds unchanged.

The first rule was easy. As is often the case, a URL that you have won’t match a rule’s pattern and will proceed untouched. Next, it enters Rule #2:

- The URL matches the pattern.
- There are conditions, so we will try the conditions.
- `THE_REQUEST` contains `profile.php`, so the condition passes.
- We can make the substitution.

- `''` is a special substitution that means: don't change anything.
- There are flags on the rule, so we process the flags.
- There is a `F` flag, which means return a forbidden response.
- A `403 Forbidden` response is sent to the client.

The `F` flag refers to a “forbidden response.”

A few things are worth re-iterating. In order for the substitution to work, all of the conditions have to pass. In this case, there is only one; it passes, so the substitution occurs. Note that `-` is a special substitution that doesn't change anything. This is useful when you want to use flags to do something for you, which is exactly what we want to do in this case.

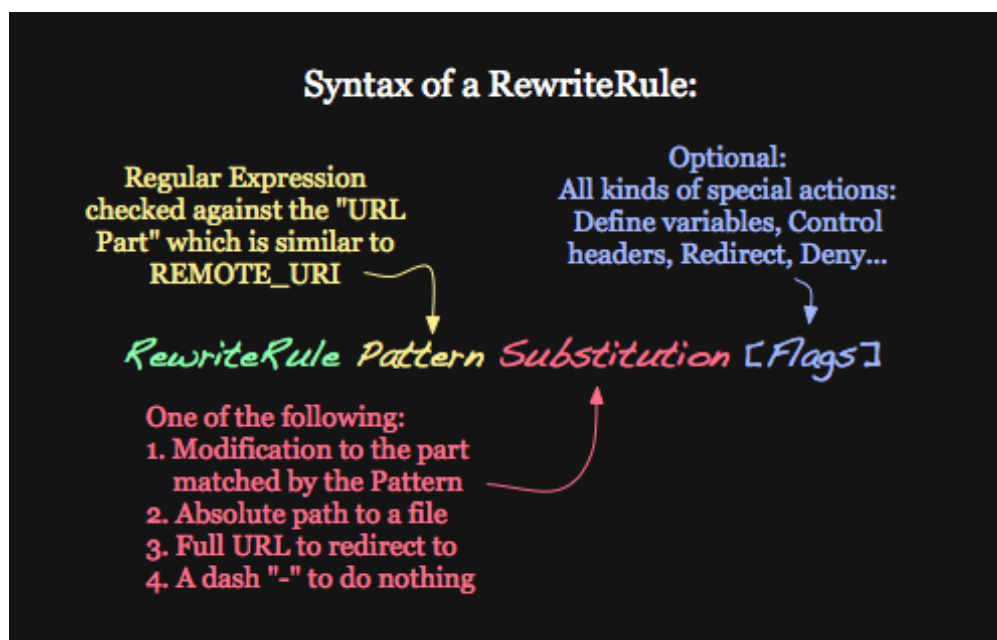
Here is the familiar table breakdown of example URLs and their responses:

Profile.php				
Incoming	Match	Capture	Outgoing	Result
profile.php?id=joe	Yes (#2)		profile.php?id=joe	Forbidden
profile/joe	Yes (#1)	joe	profile.php?id=joe	Good
profile/joe/	Yes (#1)	joe	profile.php?id=joe	Good
profile/joe/x	No		profile/joe/x	Fail

## Syntax

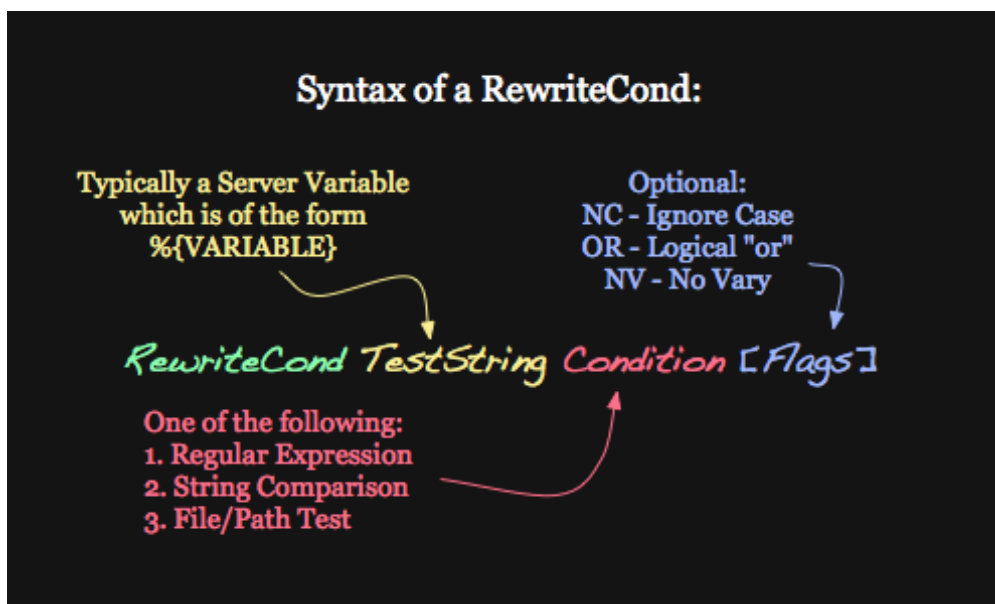
Before going over the syntax of `RewriteRule` and `RewriteCond`, I suggest that you first download the [AddedBytes Cheatsheet](#). This cheatsheet lists the most useful server variables and flags, has regular expression tips, and even a few examples.

Let's start with `RewriteRule`. You can always visit [Apache's Documentation on RewriteRule](#) if you require more information or instruction.



The cheatsheet, linked to above, displays the various flags that are available to you. While many tutorials cover these in

detail, we'll keep things simple and review the ones that I see most commonly used in real world projects.



## Debug Workflow

When working with `mod_rewrite` and creating new rules, always begin with a simple, dumbed down version of the rule, and work your way up to the final version. Resist the urge to do everything at once. The same applies for conditions. Add rules and conditions one at a time. Test often!

The key concept that I am trying to get across with this approach is that this will let you know quickly if a change you made doesn't function properly, or causes something to work incorrectly. Otherwise, you'll inevitably run into some form of error, and will have to revert all of the changes you made to track down what the problem was. This is a very roller coaster approach and will likely lead to frustration. However, if you're always steadily advancing, and each step along the way moving to workable checkpoints, you'll be in much, much better shape.

People often ignore this advice, create a complex rule, and it ends up not working. Hours later they find out the problem was not in the complex portion, but instead was a simple mistake in the regular expression that could have been caught much earlier had they carefully constructed the rule like I've explained above. The same goes for deconstructing a rule to reverse engineer a problem. This approach will seriously reduce frustration!

## In the Examples

In the examples below, we will always assume that the website's domain is `example.com`. This domain name is important because it affects the `HTTP_HOST` variable, as well as specifies a redirect URL to another file on your website. Keep this in mind if you intend to modify any of the following examples for your own website. If so, simply replace `"example.com"` with your domain. For example, Nettuts+ would replace `"example.com"` with `"nettuts.com"`.

## Removing www

This is the most classic rewrite rule. The following script will *listen* for anyone who comes to your website via `http://www.example.com`. Those who do will receive a hard redirect, and, thus, the location bar in their browser will update accordingly.

[view plaincopy to clipboardprint?](#)

1. RewriteEngine on
2. RewriteCond %{HTTP\_HOST} ^www\.example\.com\$ [NC]
3. RewriteRule ^(.\*)\$ http://example.com/\$1 [R=301,L]

The `RewriteRule` above matches anything, and saves it as `$1` – as specified by the wrapping parens. The important part in this example, though, is the `RewriteCond`. This condition checks the `HTTP_HOST` variable to determine if it started with “`www.`” If this condition is `true`, the rewrite occurs:

- The substitution is a full URL (it starts with `http://`)
- The substitution contains `$1`, which was captured earlier
- The `[R=301]` flag redirects the browser to the rewritten URL. This is a hard redirect in the sense that it forces the browser to load the new page and update its location bar with the new URL.
- The `[L]` flag indicates that this is the last rule to parse. Beyond this line, the rewrite engine should stop.

If the incoming URL had been “`http://www.example.com/user/index.html`”, then `HTTP_HOST` would have been set to `www.example.com` and the rewrite would trigger, creating `http://example.com/user/index.html`.

On the other hand, If the incoming URL had been “`http://example.com/user/index.html`”, then `HTTP_HOST` would have been `example.com`, the condition would fail, and the rewrite engine would proceed with the URL unchanged.

---

## Forbid Hotlinking

Hotlinking, referred to as [Inline Linking on Wikipedia](#), is the term used to describe one site leeching off of another site.

Hotlinking, referred to as [Inline Linking on Wikipedia](#), is the term used to describe one site leeching off of another site. Usually one site – the Leecher – will include a link to some media file (let[s say an image or video) that is hosted on another site, the Content Host. In this scenario, the Content Host’s servers are wasting bandwidth serving content to some other website.

The most common and basic approach to preventing hotlinking is to whitelist a specified number of websites, and block everything else. To determine who is requesting the content from your site, you can check the referrer.

The `HTTP_REFERER` header is set by the browser or client that is requesting the resource.

Ultimately, is not 100% reliable, however, it's generally more than effective at ceasing the majority of hotlinking. So, in our script, we need to verify if the referrer is include in a whilelist of acceptable referrers. If not, then we should them a forbidden warning:

[view plaincopy to clipboardprint?](#)

1. # Give Hotlinkers a 403 Forbidden warning.

2. RewriteEngine on
3. RewriteCond %{HTTP\_REFERER} !^http://example\.net/?.\*\$ [NC]
4. RewriteCond %{HTTP\_REFERER} !^http://example\.com/?.\*\$ [NC]
5. RewriteRule \.(gif|jpe?g|png|bmp)\$ - [F,NC]

Above, the `RewriteRule` is checking for the request of a file with any popular image extension, such as `.gif`, `.png`, or `.jpg`. Feel free to add other extensions to this list if you want to protect `.flv`, `.swf`, or other files.

The domains which are allowed to access this content are “example.net” and “example.com”. In either of these two instances, a Rewrite Conditions will fail and the substitution won’t occur. If any other domain makes an attempt, however - let’s say “sample.com” - then all the Rewrite Conditions will pass, the substitution will happen, and the `[F]` forbidden action will trigger.

---

## Give Hotlinkers a Warning Image

The previous example returns a 404 Forbidden warning when someone attempts to hotlink content from your server. You can actually go one step further, and send the hotlinker any resource of your choice! For instance, you can return a warning image with text stating, “hotlinking is not allowed”. This way, the abuser will realize their mistake and host a copy on their own server. The only required change is to follow through with the rewrite substitution, and provide your chosen image instead of the one being requested:

[view plaincopy to clipboardprint?](#)

1. # Redirect Hotlinkers to "warning.png"
2. RewriteEngine on
3. RewriteCond %{HTTP\_REFERER} !^http://example\.net/?.\*\$
4. RewriteCond %{HTTP\_REFERER} !^http://example\.com/?.\*\$ [NC]
5. RewriteRule \.(gif|jpe?g|png|bmp)\$ http://example.com/warning.png [R,NC]

Note that this is an example of what I call a “hard” or “external” redirect. The `RewriteRule` has a URL in the substitution portion and it also has the `[R]` flag.

---

## Custom 404

One neat trick that you can do with `htaccess` is to determine if the current “URL Part” leads to an actual file or directory on the web server. This is an excellent way to create a custom 404 “File not Found” page. For example, if a user tries to fetch a page in a particular directory that doesn’t exist, you can redirect him to any page you wish, such as the `index` page or a custom 404 page.

[view plaincopy to clipboardprint?](#)

1. # Generic 404 to show the "custom\_404.html" page
2. # If the requested page is not a file or directory
3. # Silent Redirect: the user's URL bar is unchanged.
4. RewriteEngine on
5. RewriteCond %{REQUEST\_FILENAME} !-f

6. RewriteCond %{REQUEST\_FILENAME} !-d
7. RewriteRule .\* custom\_404.html [L]

This is a great example of mod\_rewrite's file test operators. They are identical to file tests in bash shell scripts and even Perl scripts. Above, the condition checks if the `REQUEST_FILENAME` is not a file and not a directory. In the case where it is neither, there is no such file for the request.

If the incoming request filename can't be found, then this script loads a "custom404.html" page. *Note that there is no [R] flag - this is a silent redirect, not a hard redirect. The user's Location Bar will not change, but the contents of the page will be "custom404.html".*

---

## Safety First

If you have various mod\_rewrite snippets that you want to easily distribute to other servers or environments, you might want to be careful. Any invalid directive in a `.htaccess` file will likely trigger an internal server error. So, if an environment you move the snippet to doesn't support mod\_rewrite, you could temporarily break it.

One solution to this problem is the "check" for the mod\_rewrite module. This is possible with any module; simply wrap your mod\_rewrite code in an `<IfModule>` block and you'll be all set:

[view plain](#)[copy to clipboard](#)[print](#)?

1. <IfModule mod\_rewrite.c>
  - 2.
  3. # Turn on
  4. RewriteEngine on
  - 5.
  6. # Always remove www (with a hard redirect)
  7. RewriteCond %{HTTP\_HOST} ^www\.example\.com\$ [NC]
  8. RewriteRule ^(.\*)\$ http://example.com/\$1 [R=301,L]
  - 9.
  10. # Generic 404 for anyplace on the site
  11. # ...
  - 12.
  13. </IfModule>
- 

## Conclusion

I hope that this tutorial has proven that mod\_rewrite isn't *too* scary. In fact, its quirks and speed bumps can be avoided with careful development practices. Let me know if you have any questions!

Like

134 likes. [Sign Up](#) to see what your friends like.



Tags: [htaccess](#)[mod\\_rewrite](#)[mod\\_rewrite](#)

## By **Joseph Pecoraro**

My name is Joseph Pecoraro. I'm a web developer and designer from western New York. I am presently attending the great Rochester Institute of Technology to earn my MS in Computer Science by the end of 2009.