That's it! Once you've uploaded the file, the rewrite rule should take effect immediately.

Some Content Management Systems (CMSs), like WordPress for example, overwrite .htaccess files with their own settings. In that case, you may need to figure out a way to do your rewrite from within the CMS.

http://example.com/folder1/ to http://example.com/folder2/

http://example.com/folder1/ becomes http://example.com/folder2/ or just http://example.com/.

domains/example.com/html/folder2/ must exist and have content in it for this to work.

.htaccess

This .htaccess file will redirect http://example.com/folder1/ to http://example.com/folder2/. Choose this version if you don't have the same file structure in both directories:

Filename: .htaccess

Options +FollowSymLinks RewriteEngine On RewriteRule ^folder1.*$ http://example.com/folder2/ [R=301,L]

- This .htaccess file will redirect http://example.com/folder1/ to plain http://example.com/. Choose this version if you want people redirected to your home page, not whatever individual page in the old folder they originally requested:

Filename: .htaccess.

```
Options +FollowSymLinks
RewriteEngine On
RewriteRule ^folder1.*$ http://example.com/ [R=301,L]
```

- This .htaccess file will redirect *http://example.com/folder1/file.html* to *http://example.com/folder2/file.html* .

Choose this version if your content is duplicated in both directories:

File name: .htaccess

```
Options +FollowSymLinks
RewriteEngine On
RewriteRule ^folder1/(.*)$ http://gs.mt-example.com/folder2/$1 [R=301,L]
```

Test

Upload this file to folder2 (if you followed the first or third example) or your html folder (if you followed the second example) with FTP :

Filename: index.html

```
<html>
<body>
Mod_rewrite is working!
</body>
</html>
```

Then, if you followed the first or second example, visit *http://example.com/folder1/* in your browser. You should see the URL change to *http://example.com/folder2/* or *http://example.com/* and the test page content.

If you followed the third example, visit *http://example.com/folder1/index.html*. You should be redirected to *http://example.com/folder2/index.html* and see the test page content.

Code explanation

- Options +FollowSymLinks is an Apache directive, prerequisite for mod_rewrite.

- RewriteEngine On enables mod_rewrite.
- RewriteRule defines a particular rule.
- The first string of characters after RewriteRule defines what the original URL looks like. There's a more detailed explanation of the special characters at the end of this article.

- The second string after RewriteRule defines the new URL. This is in relation to the document root (html) directory. / means the html directory itself, and subfolders can also be specified.

- $1 at the end matches the part in parentheses () from the first string. Basically, this makes sure that sub-pages get redirected to the same sub-page and not the main page. Leave it out to redirect to the main page. (It is left out in the first two examples for this reason. If you don't have the same content in the new directory that you had in the old directory, leave this out.)

- [R=301,L] - this performs a 301 redirect and also stops any later rewrite rules from affecting this URL (a good idea to add after the last rule). It's on the same line as RewriteRule
, at the end.

http://example.com/file.html to http://example.com/folder1/file.html

http://example.com/file.html becomes *http://example.com/folder1/file.html*.

Note: The directory folder1 must be unique in the URL. It won't work for *http://example.com/folder1/folder1.html* . The directory
folder1
must exist and have content in it.

.htaccess

- This .htaccess file will redirect *http://example.com/file.html* to *http://example.com/folder1/file.html* :

Filename: .htaccess

```
Options +FollowSymLinks
RewriteEngine On
RewriteCond %{HTTP_HOST} example.com$ [NC]
RewriteCond %{HTTP_HOST} !folder1
RewriteRule ^(.*)$ http://example.com/folder1/$1 [R=301,L]
```

Test

Upload this file to folder1 with FTP :

Filename: index.html

```
<html>
<body>
Mod_rewrite is working!
</body>
</html>
```

Then, visit *http://example.com/* in your browser. You should see the URL change to *http://example.com/folder1/*
and the test page content.

Code explanation

- Options +FollowSymLinks is an Apache directive, prerequisite for mod_rewrite.
- RewriteEngine On enables mod_rewrite.
- RewriteCond %{HTTP_HOST} shows which URLs we do and don't want to run through the rewrite.
- In this case, we want to match example.com.
- ! means "not." We don't want to rewrite a URL that already includes folder1, because then it would keep getting
folder1
added, and it would become an infinitely long URL.

- [NC] matches both upper- and lower-case versions of the URL.
- RewriteRule defines a particular rule.
- The first string of characters after RewriteRule defines what the original URL looks like. There's a more detailed explanation of the special characters at the end of this article.

- The second string after RewriteRule defines the new URL. This is in relation to the document root (html) directory. / means the html directory itself, and subfolders can also be specified.

- $1 at the end matches the part in parentheses () from the first string. Basically, this makes

sure that sub-pages get redirected to the same sub-page and not the main page. Leave it out to redirect to the main page of the subdirectory.

- [R=301,L] - this performs a 301 redirect and also stops any later rewrite rules from affecting this URL (a good idea to add after the last rule). It's on the same line as RewriteRule
, at the end.

Add www or https

http://example.com becomes http://www.example.com . Or, http://example.com becomes https://example.com
.

.htaccess

- This .htaccess file will redirect http://example.com/ to http://www.example.com/ . It will also work if an individual file is requested, such as
http://example.com/file.html
:

Filename:.htaccess

Options +FollowSymLinks RewriteEngine on RewriteCond %{HTTP_HOST} ^example.com [NC] RewriteRule ^(.*)$ http://www.example.com/$1 [R=301,L]

- This .htaccess file will redirect http://example.com/ to https://example.com/ . It will also work if an individual file is requested, such as
http://example.com/file.html
:

Filename: .htaccess

RewriteEngine On RewriteCond %{SERVER_PORT} 80 RewriteRule ^(.*)$ https://www.example.com/$1 [R,L]

Test

Visit *http://example.com* in your browser. You should see that the same page is displayed, but the URL has changed to *http://www.example.com* (first example) or *https://example.com* (second example).

Also, *http://example.com/file.html* will become *http://www.example.com/file.html* or *https://example.com/file.html* .

Code explanation

- Options +FollowSymLinks is an Apache directive, prerequisite for mod_rewrite.
- RewriteEngine On enables mod_rewrite.
- RewriteCond %{HTTP_HOST} shows which URLs we do and don't want to run through the rewrite.
- In this case, we want to match anything that starts with example.com.

- [NC] matches both upper- and lower-case versions of the URL.
- RewriteRule defines a particular rule.
- The first string of characters after RewriteRule defines what the original URL looks like. There's a more detailed explanation of the special characters at the end of this article.

- The second string after RewriteRule defines the new URL. This is in relation to the document root (html) directory. / means the html directory itself, and subfolders can also be specified.

- $1 at the end matches the part in parentheses () from the first string. Basically, this makes sure that sub-pages get redirected to the same sub-page and not the main page.

- [R=301,L] - this performs a 301 redirect and also stops any later rewrite rules from affecting this URL (a good idea to add after the last rule). It's on the same line as RewriteRule , at the end.

Regular expressions

Rewrite rules often contain symbols that make a regular expression (regex). This is how the server knows exactly how you want your URL changed. However, regular expressions can be tricky to decipher at first glance. Here's some common elements you will see in your rewrite rules, along with some specific examples.

- ^ begins the line to match.
- $ ends the line to match.
- So, ^folder1$ matches folder1 exactly.

- . stands for "any non-whitespace character" (example: a, B, 3).
- * means that the previous character can be matched zero or more times.
- So, ^uploads.*$ matches uploads2009, uploads2010, etc.
- ^.*$ means "match anything and everything." This is useful if you don't know what your users might type for the URL.

- () designates which portion to preserve for use again in the $1 variable in the second string. This is useful for handling requests for particular files that should be the same in the old and new versions of the URL.

See more regular expressions at perl.org .

Troubleshooting
404 Not Found

Examine the new URL in your browser closely. Does it match a file that exists on the server in the new location specified by the rewrite rule? You may have to make your rewrite rule more broad (you may be able to remove the $1 from the second string). This will direct rewrites to the main index page given in the second string. Or, you may need to copy files from your old location to the new location.

If the URL is just plain wrong (like *http://example.com/folder1//file.html* - note the two /s) you will need to re-examine your syntax. (mt) Media Temple does not support syntax troubleshooting.

Infinite URL, timeout, redirect loop

If you notice that your URL is ridiculously long, that your page never loads, or that your browser gives you an error message about redirecting, you likely have conflicting redirects in place.

You should check your entire .htaccess file for rewrite rules that might match other rewrite rules. You may also need to check .htaccess files in subdirectories. Note that FTP will not show .htaccess files unless you have enabled the option to view hidden files and folders. See our .htaccess article for details.

Also, it's possible to include redirects inside HTML and PHP pages. Check the page you were testing for its own redirects.

Adding [L] after a rewrite rule can help in some cases, because that tells the server to stop trying to rewrite a URL after it has applied that rule.