

more .htaccess tips and tricks..

url: <http://corz.org/serv/tricks/htaccess2.php>

```
<ifModule>
more clever stuff here
</ifModule>
```

redirecting and rewriting

"The great thing about mod_rewrite is it gives you all the configurability and flexibility of Sendmail. The downside to mod_rewrite is that it gives you all the configurability and flexibility of Sendmail."

- Brian Behlendorf, Apache Group

One of the more powerful tricks of the .htaccess hacker is the ability to rewrite URLs. This enables us to do some mighty manipulations on our links; useful stuff like transforming very long URL's into short, cute URLs, transforming dynamic ?generated=page&URL's into /friendly/flat/links, redirect missing pages, preventing hot-linking, performing automatic language translation, and much, much more.

Make no mistake, mod_rewrite is complex. This isn't the subject for a quick bite-size tech-snack, probably not even a week-end crash-course, I've seen guys pull off some real cute stuff with mod_rewrite, but with kudos-hat tipped firmly towards that bastard operator from hell, Ralf S. Engelschall, author of the magic module itself, I have to admit that a great deal of it still seems so much voodoo to me.

The way that rules can work one minute and then seem not to the next, how browser and other in-between network caches interact with rules and testing rules is often baffling, maddening. When I feel the need to bend my mind completely out of shape, I mess around with mod_rewrite!

After all this, it does work, and while I'm not planning on taking that week-end crash-course any time soon, I have picked up a few wee tricks myself, messing around with web servers and web sites, this place..

The plan here is to just drop some neat stuff, examples, things that have proven useful, and work on a variety of server setups; there are Apache's all over my LAN, I keep coming across old .htaccess files stuffed with past rewriting experiments that either worked; and I add them to my list, or failed dismally; and I'm surprised that more often these days, I can see exactly why!

Very little here is my own invention. Even the bits I figured out myself were already well documented, I just hadn't understood the documents, or couldn't find them. Sometimes, just looking at the same thing from a different angle can make all the difference, so perhaps this humble stab at URL Rewriting might be of some use. I'm writing it for me, of course. but I do get some credit for this

```
# time to get dynamic, see..
RewriteRule (.*)\.htm $1.php
```

beginning rewriting.. Whenever you use mod_rewrite (the part of Apache that does all this magic), you need to do:

```
you only need to do this once per .htaccess file:
Options +FollowSymlinks
RewriteEngine on
```

..before any ReWrite rules. note: +FollowSymLinks must be enabled for any rules to work, this is a security requirement of the rewrite engine. Normally it's enabled in the root and you shouldn't have to add it, but it doesn't hurt to do so, and I'll insert it into all the examples on this page, just in case*.

The next line simply switches on the rewrite engine for that folder. if this directive is in you main .htaccess file, then the ReWrite engine is theoretically enabled for your entire site, but it's wise to always add that line before you write any redirections, anywhere.

- Although highly unlikely, your host may have +FollowSymLinks enabled at the root level, yet disallow its addition in .htaccess; in which case, adding +FollowSymLinks will break your setup (probably a 500 error), so just remove it, and your rules should work fine.

Important: While some of the directives on this page may appear split onto two lines in your browser, in your .htaccess file they must exist completely on one line. If you drag-select and copy the directives on this page, they should paste just fine into any text editor.

simple rewriting Simply put, Apache scans all incoming URL requests, checks for matches in our .htaccess file and rewrites those matching URLs to whatever we specify. something like this..:

```
all requests to whatever.htm will be sent to whatever.php:
Options +FollowSymlinks
RewriteEngine on
RewriteRule ^(.*)\.htm$ $1.php [NC]
```

Handy for anyone updating a site from static htm (you could use .html, or .htm(.*), .htm?, etc) to dynamic php pages; requests to the old pages are automatically rewritten to our new urls. no one notices a thing, visitors and search engines can access your content either way. leave the rule in; as an added bonus, this enables us to easily split php code and its included html structures into two separate files, a nice idea; makes editing and updating a breeze. The [NC] part at the end means "No Case", or "case-insensitive"; more on the switches, later.

Folks can link to whatever.htm or whatever.php, but they always get whatever.php in their browser, and this works even if whatever.htm doesn't exist! But I'm straying..

As it stands, it's a bit tricky; folks will still have whatever.htm in their browser address bar, and will still keep bookmarking your old .htm URL's. Search engines, too, will keep on indexing your links as .htm, some have even argued that serving up the same content from two different places could have you penalized by the search engines. This may or not bother you, but if it does, mod_rewrite can do some more magic..:

```
this will do a "real" external redirection:
Options +FollowSymlinks
RewriteEngine on
RewriteRule ^(.+)\.htm$ http://corz.org/$1.php [R,NC]
```

This time we instruct mod_rewrite to do a proper external rewrite, aka, "redirection". Now, instead of just background rewriting on-the-fly, the user's browser is physically redirected to a new URI, and whatever.php appears in their browser's address bar - search engines and other spidering entities will automatically update their links to the .php versions; everyone wins. You can take your time with the updating, too.

Note: if you use [R] alone, it defaults to sending an HTTP "MOVED TEMPORARILY" redirection, aka, "302". But you can send other codes, like so..:

```
this performs the exact same as the previous example RewriteRule.
RewriteRule ^(.+)\.htm$ http://corz.org/$1.php [R=302,NC]
```

Okay, I sent the exact same code, but I didn't have to. For details of the many 30* response codes you can send, see here. Most people seem to want to send 301, aka, "MOVED PERMENENTLY".

Note: if you add an "L" flag to the mix; meaning "Last Rule", e.g. [R=302,NC,L]; Apache will stop processing rules for this request at that point, which may or may not be what you want. Either way, it's useful to know.

not-so-simple rewriting ... flat links and more You may have noticed, the above examples use regular expression to match variables. What that simply means is.. match the part inside (.+) and use it to construct "\$1" in the new URL. In other words, (.+) = \$1 you could have multiple (.+) parts and for each, mod_rewrite automatically creates a matching \$1, \$2, \$3, etc, in your target (aka. 'substitution') URL. This facility enables us to do all sorts of tricks, and the most common of those, is the creation of "flat links"..

Even a cute short link like <http://mysite/grab?file=my.zip> is too ugly for some people, and nothing less than a true old-school solid domain/path/flat/link will do. Fortunately, mod_rewrite makes it easy to convert URLs with query strings and multiple variables into exactly this, something like..:

```
a more complex rewrite rule:
Options +FollowSymlinks
RewriteEngine on
RewriteRule ^files/([^/]+)/([^/]+).zip /download.php?section=$1&file=$2 [NC]
```

would allow you to present this link as..

<http://mysite/files/games/hoopy.zip>

and in the background have that transparently translated, server-side, to..

<http://mysite/download.php?section=games&file=hoopy>

which some script could process. You see, many search engines simply don't follow our ?generated=links, so if you create generating pages, this is useful. However, it's only the dumb search engines that can't handle these kinds of links; we have to ask ourselves.. do we really want to be listed by the dumb search engines? Google will handle a good few parameters in your URL without any problems, and the (hungry hungry) msn-bot stops at nothing to get that page, sometimes again and again and again...

I personally feel it's the search engines that should strive to keep up with modern web technologies, in other words; we shouldn't have to dumb-down for them. But that's just my opinion. Many users will prefer /files/games/hoopy.zip to /download.php?section=games&file=hoopy but I don't mind either way. As someone pointed out to me recently, presenting links as standard/flat/paths means you're less likely to get folks doing typos in typed URL's, so something like..:

```
an even more complex rewrite rule:
Options +FollowSymlinks
RewriteEngine on
RewriteRule ^blog/([0-9]+)-([a-z]+) http://corz.org/blog/index.php?archive=$1-$2 [NC]
```

would be a neat trick, enabling anyone to access my blog archives by doing..

<http://corz.org/blog/2003-nov>

in their browser, and have it automagically transformed server-side into..

<http://corz.org/blog/index.php?archive=2003-nov>

which corzblog would understand. It's easy to see that with a little imagination, and a basic understanding of posix regular expression, you can perform some highly cool URL manipulations.

Here's the very basics of regexp (expanded from the Apache mod_rewrite documentation)..:

```
Escaping:

\char escape that particular char

    For instance to specify special characters.. [ ] . ( ) \ etc.

Text:
```

```
.          Any single character (on its own = the entire URI)
[chars]    Character class: One of following chars
[^chars]   Character class: None of following chars
text1|text2 Alternative: text1 or text2 (i.e. "or")
```

```
e.g. [^/] matches any character except /
      (foo|bar)\.html matches foo.html and bar.html
```

Quantifiers:

```
? 0 or 1 of the preceding text
* 0 or N of the preceding text (hungry)
+ 1 or N of the preceding text
```

```
e.g. (.+)\.html? matches foo.htm and foo.html
      (foo)?bar\.html matches bar.html and foobar.html
```

Grouping:

```
(text) Grouping of text
```

Either to set the borders of an alternative or for making backreferences where the nth group can be used on the target of a RewriteRule with \$n

```
e.g. ^(.*)\.html foo.php?bar=$1
```

Anchors:

```
^ Start of line anchor
$ End of line anchor
```

An anchor explicitly states that the character right next to it MUST be either the very first character ("^"), or the very last character ("\$") of the URI string to match against the pattern, e.g..

```
^foo(.*) matches foo and foobar but not eggfoo
(.*)l$ matches fool and cool, but not foo
```

shortening URLs One common use of mod_rewrite is to shorten URL's. Shorter URL's are easier to remember and, of course, easier to type. An example..:

```
beware the regular expression:
Options +FollowSymlinks
RewriteEngine On
RewriteRule ^grab /public/files/download/download.php
```

this rule would transform this user's URL..

<http://mysite/grab?file=my.zip>

server-side, into..

<http://mysite/public/files/download/download.php?file=my.zip>

which is a wee trick I use for my distro machine, among other things. everyone likes short URL's, and so will you; using this technique, you can move /public/files/download/ to anywhere else in your site, and all the old links still work fine; simply alter your .htaccess file to reflect the new location. edit one line, done - nice - means even when stuff is way deep in your site you can have cool links like this..

/trueview/sample.php and this; links which are not only short, but flat..

capturing variables Slapping (.*) onto the end of the request part of a ReWriteRule is just fine when using a simple \$_GET variable, but sometimes you want to do trickier things, like capturing particular variables and converting them into other variables in the target URL. Or something else..

When capturing variables, the first thing you need to know about, is the [QSA] flag, which simply tags all the original variables back onto the end of the target url. This may be all you need, and will happen automatically for simple rewrites. The second thing, is %{QUERY_STRING}, an Apache server string we can capture variables from, using simple RewriteCond (aka. conditional) statements.

RewriteCond is very like doing if...then...do in many programming languages. If a certain condition is true, then do the rewrite that follows..

In the following example, the RewriteCond statement checks that the query string has the foo variable set, and captures its value while it's there. In other words, only requests for /grab that have the variable foo set, will be rewritten, and while we're at it, we'll also switch foo, for bar, just because we can..:

```
capturing a $_GET variable:
Options +FollowSymlinks
RewriteEngine On
RewriteCond %{QUERY_STRING} foo=(.*)
RewriteRule ^grab(.*) /page.php?bar=%1
```

would translate a link/user's request for..

<http://domain.com/grab?foo=bar>

server-side, into..

<http://domain.com/page.php?bar=bar>

Which is to say, the user's browser would be fed page.php (without an [R] flag in the RewriteRule, their address bar would still read /grab?foo=bar). The variable bar would be available to your script, with its value set to bar. This variable has been magically created, by simply using a regular ? in the target of the RewriteRule, and tagging on the first captured backreference, %1.. ?bar=%1

Note how we use the % character, to specify variables captured in RewriteCond statements, aka "Backreferences". This is exactly like using \$1 to specify numbered backreferences captured in RewriteRule patterns, except for strings captured inside a RewriteCond statement, we use % instead of \$. Simple.

You can use the [QSA] flag in addition to these query string manipulations, merge them. In the next example, the value of foo becomes the directory in the target URL, and the variable file is magically created. The original query string is then tagged back onto the end of the whole thing..:

```
QSA Overkill!
Options +FollowSymlinks
RewriteEngine On
RewriteCond %{QUERY_STRING} foo=(.+)
RewriteRule ^grab/(.*) /%1/index.php?file=$1 [QSA]
```

So a request for..

<http://domain.com/grab/foobar.zip?level=5&foo=bar>

is translated, server-side, into..

<http://domain.com/bar/index.php?file=foobar.zip&level=5&foo=bar>

Depending on your needs, you could even use flat links and dynamic variables together, something like this could be useful..:

```
mixing flat and dynamic links in a single ruleset..
Options +FollowSymlinks
RewriteEngine On
RewriteCond %{QUERY_STRING} version=(.+)
RewriteRule ^grab/([^/]+)/(.*) /%1/index.php?section=$1&file=$2 [QSA]
```

By the way, you can easily do the opposite, strip a query string from a URL, by simply putting a ? right at the end of the target part. This example does exactly that, whilst leaving the actual URI intact..:

```
just a demo!
Options +FollowSymlinks
RewriteEngine On
RewriteCond %{QUERY_STRING} .
RewriteRule foo.php(.*) /foo.php? [L]
```

The RewriteCond statement only allows requests that have something in their query string, to be processed by the RewriteRule, or else we'd end up in that hellish place, dread to all mod_rewriters.. the endless loop. RewriteCond is often used like this; as a safety-net.

cooler access denied In part one I demonstrated a drop-dead simple mechanism for denying access to particular files and folders. The trouble with this is the way our user gets a 403 "Access Denied" error, which is a bit like having a door slammed in your face. Fortunately, mod_rewrite comes to the rescue again and enables us to do less painful things. One method I often employ is to redirect the user to the parent folder..:

```
they go "huh?.. ahhh!"
# send them up!
Options +FollowSymlinks
RewriteEngine on
RewriteRule ^(.*)$ ../ [NC]
```

It works great, though it can be a wee bit tricky with the URLs, and you may prefer to use a harder location, which avoids potential issues in indexed directories, where folks can get in a loop..:

```
they go damn! Oh!
# send them exactly there!
Options +FollowSymlinks
RewriteEngine on
RewriteRule ^(.*)$ /comms/hardware/router/ [NC]
```

Sometimes you'll only want to deny access to most of the files in the directory, but allow access to maybe one or two files, or file types, easy..:

```
deny with style!
# users can load only "special.zip", and the css and js files.
Options +FollowSymlinks
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !^(.+)\.css$
RewriteCond %{REQUEST_FILENAME} !^(.+)\.js$
RewriteCond %{REQUEST_FILENAME} !special.zip$
RewriteRule ^(.+)$ /chat/ [NC]
```

Here we take the whole thing a stage further. Users can access .css (stylesheet) and Javascript files without problem, and also the file called "special.zip", but requests for any other file types are immediately redirected back up to the main "/chat/" directory. You can add as many types as you need. You could also

bundle the filetypes into one line using | (or) syntax, though individual lines are perhaps clearer.

Here's what's currently cooking inside my /inc/ directory..:

```
all-in-one control..
RewriteEngine on
Options +FollowSymlinks
# allow access with no restrictions to local machine at 192.168.1.3
RewriteCond %{REMOTE_ADDR} !192.168.1.3
# allow access to all .css and .js in sub-directories..
RewriteCond %{REQUEST_URI} !\.css$
RewriteCond %{REQUEST_URI} !\.js$
# allow access to the files inside img/, but not a directory listing..
RewriteCond %{REQUEST_URI} !img/(.*)\..
# allow access to these particular files...
RewriteCond %{REQUEST_URI} !comments.php$
RewriteCond %{REQUEST_URI} !corzmail.php$
RewriteCond %{REQUEST_URI} !digitrack.php$
RewriteCond %{REQUEST_URI} !gd-verify.php$
RewriteCond %{REQUEST_URI} !post-dumper.php$
RewriteCond %{REQUEST_URI} !print.php$
RewriteCond %{REQUEST_URI} !source-dump.php$
RewriteCond %{REQUEST_URI} !textview.php$
RewriteRule ^(.*)$ / [R,NC,L]
```

Ban User Agents, referrers, script-kiddies and more.. There are many valid reasons to ban a particular request from sucking up your site's resources; resources that could be better served to valid, interested users. It might be some cross-site attack script, or inward link from a place you don't want to be associated with, or perhaps a web sucker or download manager, whatever; .htaccess + mod_rewrite provides ways to protect your content from unwanted "guests".

The basic formula is standard if-then logic: if the request meets a particular CONDITION, then REWRITE the request. The "conditions" can be many things; perhaps the referrer header sent by their browser (the site they came from), or the page they asked for, or a particular query parameter, or the type of client (browser, etc.) they are using, or any other piece of information Apache has attached to the request. Here's an example which will deny access to "Teleport Pro", a download manager which is known to suck, hard..:

```
Who need's a local copy, when I'm right here?..
RewriteCond %{HTTP_USER_AGENT} ^Teleport\ Pro [NC]
RewriteRule . abuse.txt [L]
```

It's your site, and just like your home, you have every right to exert some control over who gets in. You may have a huge list of user agents you'd rather not have eating your bandwidth; so use the [OR] flag, and line 'em up..:

```
A little garlic for the net vampires..
RewriteCond %{HTTP_USER_AGENT} ^BackWeb [NC,OR]
RewriteCond %{HTTP_USER_AGENT} ^Bandit [NC,OR]
RewriteCond %{HTTP_USER_AGENT} ^BatchFTP [NC,OR]
RewriteCond %{HTTP_USER_AGENT} ^BecomeBot [NC,OR]
RewriteCond %{HTTP_USER_AGENT} ^BlackWidow [NC,OR]
# etc..
RewriteCond %{HTTP_USER_AGENT} ^Net\ Vampire [NC]
RewriteRule . abuse.txt [L]
```

This forms the basis of what often becomes a HUGE list of ban-lines. Remember, we aren't limited to user agent strings..:

```
Suckers, h4x0rz, kiddies, cross-site scripters and more.. Bye now!
# why not come visit me directly?
RewriteCond %{HTTP_REFERER} \.opendirviewer\. [NC,OR]
# this prevents stoopid cross-site discovery attacks..
RewriteCond %{THE_REQUEST} \?\ HTTP/ [NC,OR]
# please stop pretending to be the Googlebot..
RewriteCond %{HTTP_REFERER} users\.skynet\.be.* [NC,OR]
# really, we need a special page for these twats..
RewriteCond %{QUERY_STRING} \=\\w\| [NC,OR]
RewriteCond %{THE_REQUEST} etc/passwd [NC,OR]
RewriteCond %{REQUEST_URI} owssvr\.dll [NC,OR]
# you can probably work these out..
RewriteCond %{QUERY_STRING} \=\\w\| [NC,OR]
RewriteCond %{THE_REQUEST} \\*\ HTTP/ [NC,OR]
# etc..
RewriteCond %{HTTP_USER_AGENT} Sucker [NC]
RewriteRule . abuse.txt [L]
```

Fortunately, mod_rewrite can parse enormous lists of ban-lines in milliseconds, so feel free to be as specific and comprehensive as required.

As ever, thorough testing is strongly recommended. Simply send requests matching your conditions and see what happens. And importantly; normal requests, too. Firefox, Opera, Konqueror, and most other decent browsers, allow you to alter the user agent string; though you would quickly find the process tedious in a testing situation. Far better to use some tool better designed to send fake HTTP requests..

It's not too difficult to mock up a web request on the command-line with any-old user agent using a scripting language like php or Perl, if you have these things available (read: most Linux/UNIX/BSD/etc. as well as many other OS). Many examples exist online. In fact, you could quickly create a suite of tests, designed to interrogate all your rewrite rules, with results logging and much more, if required. cURL is always useful for jobs like this, so long as you don't add a cURL ban-line!

On a Windows desktop, Sam Spade can send a single spoofed request with a couple of clicks, along with a stack of similarly handy tricks, and regularly proves itself invaluable.

Don't let just anyone hammer your site! While I'm on the subject of abusive web clients, you will probably have noticed that many clients (bots, spiders, automated suckers and such) like to disguise their user agent information, in fact any information, in an attempt to bring your site to its knees, hammering your pages so-many times per second in the process. Oh dear.

If you are interested in a way to defeat hammering web clients, regardless of who they pretend to be, or whether or not they accept cookies or any such malarkey, check out Anti-Hammer. It's free.

prevent hot-linking

Believe it or not, there are some webmasters who, rather than coming up with their own content will steal yours. Really! Even worse, they won't even bother to copy to their own server to serve it up, they'll just link to your content! no, it's true, in fact, it used to be incredibly common. These days most people like to prevent this sort of thing, and .htaccess is one of the best ways to do it.

This is one of those directives where the mileage variables are at their limits, but something like this works fine for me..:

```
how DARE they!
Options +FollowSymlinks
```

```
# no hot-linking
RewriteEngine On
RewriteCond %{HTTP_REFERER} !^$
RewriteCond %{HTTP_REFERER} !^http://(www\.)?corz\.org/ [NC]
RewriteCond %{REQUEST_URI} !hotlink\.(gif|png) [NC]
RewriteRule .*\. (gif|jpg|png)$ http://corz.org/img/hotlink.png [NC]
```

You may see the last line broken into two, but it's all one line (all the directives on this page are). Let's have a wee look at what it does..

We begin by enabling the rewrite engine, as always.

The first RewriteCond line allows direct requests (not from other pages - an "empty referrer") to pass unmolested. The next line means; if the browser did send a referrer header, and the word "corz.org" is not in the domain part of it, then DO rewrite this request.

The all-important final RewriteRule line instructs mod_rewrite to rewrite all matched requests (anything without "corz.org" in its referrer) asking for gifs, jpegs, or pngs, to an alternative image.

There are loads of ways you can write this rule; Google for "hot-link protection" and get a whole heap. Simple is best. You could send a wee message instead, or direct them to some evil script, or something. Mine is a simple corz.org logo, which I think is rather clever. Actually, these days, I do something even cleverer-er..

lose the "www"

I'm often asked how I prevent the "www" part showing up at my site, so I guess I should add something about that. Briefly, if someone types <http://www.corz.org/> into their browser (or uses the www part for any link at corz.org) it is redirected to the plain, rather neat, <http://corz.org/> version. This is very easy to achieve, like this..:

```
beware the regular expression:
Options +FollowSymlinks
RewriteEngine on
RewriteCond %{http_host} ^www\.corz\.org [NC]
RewriteRule ^(.*)$ http://corz.org/$1 [R=301,NC]
```

You don't need to be touched by genius to see what's going on here. There are other ways you could write this rule, but again, simple is best. Like most of the examples here, the above is pasted directly from my own main .htaccess file, so you can be sure it works perfectly. In fact, I recently updated it so that I could share rules between my dev mirror and live site without any .htaccess editing..:

```
here's what I'm currently using:
Options +FollowSymlinks
RewriteEngine on
RewriteCond %{HTTP_HOST} ^www\.(.*) [NC]
RewriteRule ^(.*)$ http://%1/$1 [R=301,NC,L]
```

multiple domains in one root

If you are in the unfortunate position of having your sites living on a host that doesn't support multiple domains, you may be forced to roll your own with .htaccess and mod_rewrite. So long as your physical directory structure is well thought-out, this is fairly simple to achieve.

For example, let's say we have two domains, pointing at a single hosted root; domain-one.com and domain-two.com. In our web server root, we simply create a folder for each domain, perhaps one/, and two/ then in our main (root) .htaccess, rewrite all incoming requests, like this..:

```

All requests NOT already rewritten into these folders, transparently rewrite..
#two domains served from one root..
RewriteCond %{HTTP_HOST} domain-one.com
RewriteCond %{REQUEST_URI} !^/one
RewriteRule ^(.*)$ one/$1 [L]

RewriteCond %{HTTP_HOST} domain-two.com
RewriteCond %{REQUEST_URI} !^two
RewriteRule ^(.*)$ two/$1 [L]

```

All requests for the host domain-one.com are rewritten (not R=redirected) to the one/ directory, so long as they haven't already been rewritten there (the second RewriteCond). Same story for domain-two.com. Note the inconsistency in the RewriteCond statement; !^/dir-name and !^dir-name should both work fine.

Also note, with such a simple domain & folder naming scheme, you could easily merge these two rule sets together. This would be unlikely in the real world though, which is why I left them separate; but still, worth noting.

Other general settings and php directives can also go in this root .htaccess file, though if you have any further rewrite you'd like to perform; short URL's, htm to php conversion and what-not; it's probably easier and clearer to do those inside the sub-directory's .htaccess files.

automatic translation

If you don't read English, or some of your guests don't, here's a neat way to have the wonderful Google translator provide automatic on-the-fly translation for your site's pages. Something like this..:

```

they simply add their country code to the end of the link, or you do..
Options +FollowSymlinks
RewriteEngine on
RewriteRule ^(.*)-fr$ http://www.google.com/translate_c?hl=fr&sl=en&u=http://corz.org/$1 [R,NC]
RewriteRule ^(.*)-de$ http://www.google.com/translate_c?hl=de&sl=en&u=http://corz.org/$1 [R,NC]
RewriteRule ^(.*)-es$ http://www.google.com/translate_c?hl=es&sl=en&u=http://corz.org/$1 [R,NC]
RewriteRule ^(.*)-it$ http://www.google.com/translate_c?hl=it&sl=en&u=http://corz.org/$1 [R,NC]
RewriteRule ^(.*)-pt$ http://www.google.com/translate_c?hl=pt&sl=en&u=http://corz.org/$1 [R,NC]

```

You can create your menu with its flags or whatever you like, and add the country code to end of the links.. <a href="page.html-fr" id=""... Want to see this page in French?

Although it is very handy, and I've been using it here for a couple of years here at the org, for my international blog readers, all two of them, heh. Almost no one knows about it, mainly because I don't have any links . One day I'll probably do a wee toolbar with flags and what-not. Perhaps not. Trouble is, the Google translator stops translating after a certain amount of characters (which seems to be increasing, good), though these same rules could easily be applied to other translators, and if you find a good one, one that will translate a really huge document on-the-fly, do let me know!

If you wanted to be really clever, you could even perform some some kind of IP block check and present the correct version automatically, but that is outside the scope of this document. note: this may be undesirable for pages where technical commands are given (like this page) because the commands will also be translated. "RewriteEngine dessus" will almost certainly get you a 500 error page!

Another thing you might like to try; rather than individual country flags; fr, de, etc., use the "u" flag, for "Universal". In theory, Google will check the client's location, and automatically translate to that language. One line in your .htaccess would cover all languages, and automatically cover new ones as Google adds them.

While I'm here, slightly related; if you are non-Englishman speaking, note, you can do a similar thing browser-side, create a "bookmarklet" (a regular bookmark, except that it "does something"), using this code for the location..

the same sort of thing, except browser-side..
`javascript:void(location.href='http://translate.google.com/translate?u='+location.href)`

httpd.conf Remember, if you put these rules in the main server conf file (usually httpd.conf) rather than an .htaccess file, you'll need to use `^/... ..` instead of `^... ..` at the beginning of the RewriteRule line, in other words, add a slash.

inheritance..

If you are creating rules in sub-folders of your site, you need to read this.

You'll remember how rules in top folders apply to all the folders inside those folders too. we call this "inheritance". normally this just works. but if you start creating other rules inside subfolders you will, in effect, obliterate the rules already applying to that folder due to inheritance, or "decandancy", if you prefer. not all the rules, just the ones applying to that subfolder. a wee demonstration..

Let's say I have a rule in my main /.htaccess which redirected requests for files ending .htm to their .php equivalent, just like the example at the top of this very page. now, if for any reason I need to add some rewrite rules to my /osx/.htaccess file, the .htm >> .php redirection will no longer work for the /osx/ subfolder, I'll need to reinsert it, but with a crucial difference..:

```
this works fine, site-wide, in my main .htaccess file
# main (top-level) .htaccess file..
# requests to file.htm goto file.php
Options +FollowSymlinks
RewriteEngine on
RewriteRule ^(.*)\.htm$ http://corz.org/$1.php [R=301,NC]
```

Here's my updated /osx/.htaccess file, with the .htm >> .php redirection rule reinserted..:

```
but I'll need to reinsert the rules for it to work in this sub-folder
# /osx/.htaccess file..
Options +FollowSymlinks
RewriteEngine on
RewriteRule some rule that I need here
RewriteRule some other rule I need here
RewriteRule ^(.*)\.htm$ http://corz.org/osx/$1.php [R=301,NC]
```

Spot the difference in the subfolder rule, highlighted in red. you must add the current path to the new rule. now it works again, and all the osx/ subfolders will be covered by the new rule. if you remember this, you can go replicating rewrite rules all over the place.

If it's possible to put your entire site's rewrite rules into the main .htaccess file, and it probably is; do that, instead, like this..:

```
it's a good idea to put all your rules in your main .htaccess file..
# root /.htaccess file..
Options +FollowSymlinks
RewriteEngine on
# .htm >> .php is now be covered by our main rule, there's no need to repeat it.
# But if we do need some /osx/-specific rule, we can do something like this..
RewriteRule ^osx/(.*)\.foo$ /osx/$1.bar [R=301,NC]
```

Note, no full URL (with domain) in the second example. Don't let this throw you; with or without is functionally identical, on most servers. Essentially, try it without the full URL first, and if that doesn't work, sigh, and add it - maybe on your next host!

The latter, simpler form is preferable, if only for its tremendous portability it offers - my live site, and my development mirror share the exact same .htaccess files - a highly desirable thing.

By the way, it perhaps doesn't go without saying that if you want to disable rewriting inside a particular subfolder, where it is enabled further up the tree, simply do:

handy for avatar folders, to allow hot-linking, etc.. RewriteEngine off

cookies Lastly, a quick word about cookies. While it's easy enough to set cookies in .htaccess without any mod_rewrite..:

```
create a cookie called "example-cookie", and set its value to "true"..  
Header set Set-Cookie "example-cookie=true"
```

..you will need it to read the cookie information back again, and "do stuff" with it. It's easy. For example, to check if the above cookie exists and has the correct value set, we could simply do..:

```
check for that same cookie + value..  
Options +FollowSymlinks  
RewriteEngine on  
RewriteCond %{HTTP_COOKIE} !example-cookie=true  
RewriteRule .* /err/401.php
```

..which could easily form the basis of a simple authentication system. As with any RewriteCond, you can get pretty complex, checking multiple cookies, utilizing regexp and more, but that's enough to get you started.

conclusion

In short, mod_rewrite enables you to send browsers from anywhere to anywhere. You can create rules based not simply on the requested URL, but also on such things as IP address, browser agent (send old browsers to different pages, for instance), and even the time of day; the possibilities are practically limitless.

The ins-and outs of mod_rewrite syntax are topic for a much longer document than this, and if you fancy experimenting with more advanced rewriting rules, I urge you to check out the Apache documentation.

If you have Apache installed on your system, there will likely be a copy of the Apache manual, right here, and the excellent mod_rewriting guide, lives right here. do check out the URL Rewriting Engine notes for the juicy syntax bits. That's where I got the cute quote for the top of the page, too.

;o) Cor

troubleshooting tips..

Fatal Redirection If you start messing around with 301 redirects [R=301], aka. "Permanently Redirected", and your rule isn't working, you could give yourself some serious headaches..

Once the browser has been redirected permanently to the wrong address, if you then go on to alter the wonky rule, your browser will still be redirected to the old address (because it's a browser thing), and you may even go on to fix, and then break the rule all over again without ever knowing it. Changes to 301 redirects can take a long time to show up in your browser.

Solution: restart your browser, or use a different one.

Better Solution: Use [R] instead of [R=301] while you are testing . When you are 100% certain the rule does exactly as it's expected to, then switch it to [R=301] for your live site.

rewrite logging.. When things aren't working, you may want to enable rewrite logging. I'll assume you are testing these mod_rewrite directives on your development mirror, or similar setup, and can access the

main httpd.conf file. If not, why not? Testing mod_rewrite rules on your live domain isn't exactly ideal, is it? Anyway, put this somewhere at the foot of your http.conf..:

```
Expect large log files..
#
# ONLY FOR TESTING REWRITE RULES!!!!
#
RewriteLog "/tmp/rewrite.log"
#RewriteLogLevel 9
RewriteLogLevel 5
```

Set the file location and logging level to suit your own requirements. If your rule is causing your Apache to loop, load the page, immediately hit your browser's "STOP" button, and then restart Apache. All within a couple of seconds. Your rewrite log will be full of all your diagnostic information, and your server will carry on as before.

Setting a value of 1 gets you almost no information, setting the log level to 9 gets you GIGABYTES! So you must remember to comment out these rules and restart Apache when you are finished because, not only will rewrite logging create space-eating files, it will seriously impact your web server's performance.

RewriteLogLevel 5 is very useful, but 2 is probably enough information for most issues.

debug-report.php A php script to make your mod_rewrite life easier! When things aren't working as you would expect, rewrite logging is a good option, but on a hosted server, you probably won't have that option, without access to httpd.conf. Fortunately, what's usually required is no more than a quick readout of all the current server variables, \$_GET array, and so on; so you can see exactly what happened to the request.

For another purpose, I long ago created debug.php, and later, finding all this information useful in chasing down wonky rewrites, created a "report" version, which rather than output to a file, spits the information straight back into your browser, as well as \$_POST, \$_SESSION, and \$_SERVER arrays, special variables, like __FILE__, and much more.

Usage is simple; you make it your target page, so in a rule like this..

```
RewriteRule ^(.*)\.html$ /catch-all.php?var=$1
```

You would have a copy of debug-report.php temporarily renamed to catch-all.php in the root of your server, and type <http://testdomain.org/foobar.html> into your address bar and, with yer mojo working, debug-report.php leaps into your browser with a shit-load of exactly the sort of information you need to figure out all this stuff. When I'm messing with mod_rewrite, debug-report.php saves me time, a lot. It's way faster than enabling rewrite logging, too. Also, it's free..

icon-sized image of a zip archive debug-report.php

Useful Links.. Apache mod_rewrite docs THE reference document for all things mod_rewrite Apache 2 mod_rewrite docs As above, but for Apache 2 Apache URL rewriting guide In more easily understood language, very useful. Apache 2 URL rewriting guide As above but for Apache 2 Hot-Link Test Tool Messing with HotLink code - test it right here. webmasterworld forum More help from those in the know David Mertz's regex tutorial Get cubed up on regular expressionism! Mod ReWrite Reference Charts I Love "I Love Jack Daniel's" ! .htaccess generator Rather neat php online .htaccess generator tool - aka. 'Dot Htaccesser', by Chris Todd .htaccess generator source The php source for the above tool. The original site has disappeared. French version, part 1 & part 2 The official French human-translation of "more .htaccess tricks and tips". Dynamic vs. Static URLs Dynamic vs. Static URLs - The official word, from Google