CHAPTER 5

Aliases, Redirecting, and Rewriting

When Apache receives a request, it is assumed that the client will be served a file out of the *DocumentRoot* directory. However, there will be times when you want these resources to be served from some other location. For example, if you wanted to place a set of documents on your Web site, it may be more convenient to leave them where they are, rather than to move them to a new location.

In this chapter, we deal with three general categories of these sorts of cases. *Aliasing* refers to mapping a URL to a particular directory. *Redirecting* refers to mapping a URL to another URL. And *Rewriting* refers to using *mod_rewrite* to alter the URL in some way.

Other recipes in this chapter are related because they map URLs to resources that are at unexpected places in the filesystem.

These topics are particularly interesting to Webmasters who want to avoid link-rot or have sites that are periodically subject to upheaval (files or directories are moved around, or even moved from server to server). The redirection and rewriting capabilities of the Apache Web server allow you to conceal such ugly behind-the-scenes disturbances from the eyes of your Internet visitors.

5.1 Mapping a URL to a Directory

Problem

You want to serve content out of a directory other than the *DocumentRoot* directory. For example, you may have an existing directory of documents, which you want to have on your Web site but that you do not want to move into the Apache document root.

Solution

```
Alias "/desired-URL-prefix" "/path/to/other/directory"
```

Discussion

The example given maps URLs starting with */desired-URL-prefix* to files in the */path/to/other/directory* directory. For example, a request for the URL:

http://example.com/desired-URL-prefix/something.html

results in the file */path/to/other/directory/something.html* being sent to the client.

This same effect could be achieved on Unixish systems by simply creating a symbolic link from the main document directory to the target directory and turning on the *Options +FollowSymLinks* directive.* However, using *Alias* explicitly allows you to keep track of these directories more easily. Creating symlinks to directories makes it hard to keep track of the location of all of your content. Additionally, a stray symlink may cause you to expose a portion of your filesystem that you did not intend to.

You may also need to add a few configuration directives to permit access to the directory that you are mapping to. An error message (in your *error_log* file) saying that the request was "denied by server configuration" usually indicates this condition. It is fairly common—and recommended in the documentation (*http://httpd.apache.org/docs/2.2/misc/security_tips.html#protectserverfiles*)—to configure Apache to deny all access, by default, outside of the *DocumentRoot* directory. Thus, you must override this for the directory in question, with a configuration block as shown below:

```
<Directory "/path/to/other/directory">
    Order allow,deny
    Allow from all
</Directory>
```

This permits access to the specified directory.

Note that the *Alias* is very strict with respect to slashes. For example, consider an *Alias* directive as follows:

```
Alias "/puppies/" "/www/docs/puppies/"
```

This directive aliases URLs starting with */puppies/* but does *not* alias the URL */puppies* (i.e., without the trailing slash). This may result in a "trailing slash problem." That is, if a user attempts to go to the URL *http://example.com/puppies* he gets a 404 error, whereas if he goes to the URL *http://example.com/puppies/* with the trailing slash, he receives content from the desired directory. To avoid this problem, create *Alias*es without the trailing slash on each argument.

Finally, make sure that if you have a trailing slash on the first argument to *Alias*, you also have one on the second argument. Consider the following example:

```
Alias "/icons/" "/usr/local/apache/icons"
```

* See the documentation for the *Option* directive at *http://httpd.apache.org/docs/2.2/mod/core.html#options*

A request for *http://example.com/icons/test.gif* results in Apache attempting to serve the file */usr/local/apache/iconstest.gif* rather than the expected */usr/local/apache/icons/test.gif*.

This is called "maintaining slash parity," which is a fancy way of saying, "If you end the alias with a slash, end the directory with one too; if the alias doesn't end with a slash, the directory shouldn't either."

See Also

- *http://httpd.apache.org/docs/2.2/mod/mod_alias.html*
- *http://httpd.apache.org/docs/2.2/mod/core.html#options*

5.2 Creating a New URL for Existing Content

Problem

You have an existing directory that you want to access using a different name.

Solution

Use an *Alias* directive in *httpd.conf*:

```
Alias "/newurl" "/www/htdocs/oldurl"
```

Discussion

Although *Alias* is usually used to map URLs to a directory outside of the *DocumentRoot* directory tree, this is not necessarily required. There are many times when it is desirable to have the same content accessible *via* a number of different names. This is typically the case when a directory has its name changed, and you wish to have the old URLs continue to work, or when different people refer to the same content by different names.

Remember that *Alias* only affects the mapping of a local URI (the */foo/bar.txt* part of *http://example.com/foo/bar.txt*); it doesn't affect or change the hostname part of the URL (the *http://example.com/* part). To alter that portion of the URL, use the *Redirect* or *RewriteRule* directives.

See Also

- Recipe 5.1
- *http://httpd.apache.org/docs/2.2/mod/mod_alias.html*
- *http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html*

5.3 Giving Users Their Own URLs

Problem

You want to give each user on your system his own Web space.

Solution

If you want users' Web locations to be under their home directories, add this to your *httpd.conf* file:

```
UserDir public_html
```

To put all users' Web directories under a central location:

```
UserDir "/www/users/*/htdocs"
```

If you want to let users access their home directory without having to use a tilde (~) in the URL, you can use *mod_rewrite* to perform this mapping:

```
RewriteEngine On
RewriteCond "/home/$1/public_html" -d [NC]
RewriteRule "^/([^/]+)/(.*)" "/home/$1/public_html/$2"
```

Finally, if you have *mod_perl* installed, you can do something more advanced like this (again, added to your *httpd.conf* file):

```
<Perl>
# Folks you don't want to have this privilege
my %forbid = map { $_ => 1 } qw(root postgres bob);
opendir H, '/home/';
my @dir = readdir(H);
closedir H;
foreach my $u (@dir) {
    next if $u =~ m/^\./;
    next if $forbid{$u};
    if (-e "/home/$u/public_html") {
        push @Alias, "/$u/", "/home/$u/public_html/";
    }
}
</Perl>
```

Discussion

The first solution is the simplest and most widely used of the possible recipes we present here. With this directive in place, all users on your system are able to create a directory called *public_html* in their home directories and put Web content there. Their Web space is accessible *via* a URL starting with a tilde (~), followed by their usernames. So, a user named *bacchus* accesses his personal Web space *via* the URL:

```
http://www.example.com/~bacchus/
```

If you installed Apache from the standard source distribution, your default configuration file includes an example of this setup. It also contains a *<Directory>* section referring to the directory */home/*/public_html*, with various options and permissions turned on. You need to uncomment that section in order for anyone to have access to these user web sites. This section should look something like the following:

```
<Directory "/home/*/public_html">
    AllowOverride FileInfo AuthConfig Limit
    Options MultiViews Indexes SymLinksIfOwnerMatch IncludesNoExec
    <Limit GET POST OPTIONS PROPFIND>
        Order allow,deny
        Allow from all
    </Limit>
    <LimitExcept GET POST OPTIONS PROPFIND>
        Order deny,allow
        Deny from all
    </LimitExcept>
</Directory>
```

Make sure you understand what each of these directives is enabling before you uncomment this section in your configuration.

The second solution differs in that the argument to *UserDir* is given as a full pathname and so is not interpreted as relative to the user's home directory, but as an actual filesystem path. The * in the file path is replaced by the username. For example, *http://example.com/~smith/* is translated to */www/users/smith/htdocs*. This directory structure needs to be configured in a manner similar to the previous example.

The third solution is slightly more sophisticated, and provides *UserDir* functionality without having to use a tilde (~) in the URL.

Using the *RewriteCond* directive, we first check for the existence of the user's home directory, and, if it exists, we rewrite requests into that directory. Performing this check first ensures that other URLs continue to work correctly, and only those URLs starting with a valid username are rewritten to a user's home directory.

This rewrite ruleset takes advantage of a little-known fact about *mod_rewrite*—in particular, that a *RewriteRule* is always considered first, and, if it matches, the *RewriteCond* is evaluated after that. Consequently, we can use $1 in the *RewriteCond*, even though the value of that variable is set in the *RewriteRule* appearing on the following line.

The fourth and final solution requires *mod_perl* and provides alias mappings for all top directories under the */home* hierarchy (typically user directories). It differs from the first two by *not* including the tilde prefix; user `smith`'s Web location would be specified as *http://example.com/smith/* instead of *http://example.com/~smith/*, but is still the filesystem location */home/smith/public_html*.

In each case, the directory in question, and directories in the path leading up to it, need to be readable for the Apache user (usually *nobody* or *www* or *httpd*), and also have the execute bit set for that user, so the Apache server can read content out of that directory.

The execute bit is needed in order to get a directory listing. Thus, for user *bob*, the directories /, */home*, */home/bob*, and */home/bob/public_html* (or the corresponding directory paths for the other solutions) all need to execute access, and the last one also requires read access.

On Unixish systems, you would set these permissions by issuing the following commands:

```
% chmod o+x / /home /home/bob
% chmod o+rx /home/bob/public_html
```

The files within the directory need only be readable:

```
% find /home/bob/public_html -type f
        | xargs chmod 644
```

either merge these two lines, or add a b
backslash at the end of the first and an
greater-than at the beginning of the sec

This will recurse through subdirectories, and change the file permission on all files, but not on directories.

If you use the first solution, many users may be concerned about these file permissions, and rightly so, as it usually allows all other users read access to these directories. Make sure that your users are aware of this, and that they keep personal files in directories that are not world readable.

The advantage of the second solution over the previous one is that these files are stored in a location that is not inside the user's home directory, and so the user may keep sensible file permissions on her home directory. This lets her store personal files there without concern that other users may have free access to them.

The last Solution is completely different and requires that you have *mod_perl* installed. The list of directives previously mentioned goes in your configuration file, using the <*Perl*> configuration directive supplied by *mod_perl*, which allows you to put Perl code in your configuration file to dynamically add things to the configuration file at server startup.

At server startup, the code shown looks in the */home/* directory for any user that has a *public_html* directory and creates an *Alias* for them. This has the advantage over the previous two solutions because the URLs no longer contain that annoying tilde character, which people tend to think unprofessional. So user *bacchus* is now able to access his personal Web space *via* the URL *http://www.example.com/bacchus/*.

The **%forbid** list at the top of the code provides a list of users who should not be given this special alias for one reason or another. This allows you to eliminate users for which this feature may cause a security risk, such as *root*, or users who have shown that they can't be trusted with such privileges.

As with the previous examples, this should be accompanied by a <*Directory*> section that enables read access for the directory */home/*/public_html*.

And, of course, you can have this code point these aliases at any location, if you want to serve content out of some other location rather than the home directories of the users.

See Also

- *http://httpd.apache.org/docs/2.2/mod/mod_userdir.html*

5.4 Aliasing Several URLs with a Single Directive

Problem

You want to have more than one URL map to the same directory but don't want multiple *Alias* directives.

Solution

Use *AliasMatch* in *http.conf* to match against a regular expression:

```
AliasMatch "^/pupp(y|ies)" "/www/docs/small_dogs"
AliasMatch "^/P-([[:alnum:]])([^/]*)" "/usr/local/projects/$1/$1$2"
```

Discussion

The *AliasMatch* directive allows you to use regular expressions to match arbitrary patterns in URLs and map anything matching the pattern to the desired URL. Think of it as *Alias* with a little more flexibility.

The first *AliasMatch* causes URLs starting with */puppy*, as well as URLs starting with */puppies*, to be mapped to the directory */www/docs/small_dogs*. The second *AliasMatch* is designed to map to the appropriate project directory if they're organised under the first character of their names. For instance, project `Example`'s URI would be `/P-Example/` and would be mapped to */usr/local/projects/E/Example/*.

Apache's regular expression syntax is discussed in much greater detail in Appendix A.

See Also

- Appendix A
- *Mastering Regular Expressions* by Jeffrey Friedl (O'Reilly)

5.5 Mapping Several URLs to the Same CGI Directory

Problem

You want to have a number of URLs map to the same CGI directory but don't want to have multiple *ScriptAlias* directives.

Solution

Use *ScriptAliasMatch* in *httpd.conf* to match against a regular expression:

```
ScriptAliasMatch "^/[sS]cripts?|cgi(-bin)?/" "/www/cgi-bin/"
```

Discussion

This is a more complicated recipe than the previous one and may require that you read Appendix A. This directive maps requests starting with */script/*, */scripts/*, */Script/*, */Scripts/*, */cgi/*, and */cgi-bin/* to the directory */www/cgi-bin/*, and it causes all files in that directory to be treated as CGI programs.

This kind of directive is generally used to clean up a mess that you have made. If you design your Web site well from the start, this sort of thing is never necessary, but the first time you redesign, or otherwise rearrange your Web site, you'll find the necessity for these sorts of contortions.

See Also

- Recipe 5.4
- Appendix A

5.6 Creating a CGI Directory for Each User

Problem

You want each user to have their own *cgi-bin* directory rather than giving them all access to the main server CGI directory.

Solution

Put this in your *httpd.conf*:

```
<Directory "/home/*/public_html/cgi-bin/">
    Options ExecCGI
    SetHandler cgi-script
</Directory>
ScriptAliasMatch "/~([^/]+)/cgi-bin/(.*)" "/home/$1/public_html/cgi-bin/$2"
```

Discussion

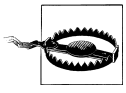
You can't use *ScriptAlias* in this case, because for each user, the first argument to *ScriptAlias* would be different. The *<Directory>* container and the *ScriptAliasMatch* directive are functionally equivalent.

This recipe lets each user put CGI scripts in her own personal Web space. Files accessed *via* URLs starting with:

```
http://www.example.com/~username/cgi-bin/
```

are treated as CGI scripts.

If you have suexec enabled, CGI programs run from this target directory will be run with the userid of the user specified in the URL. For example, a CGI program accessed *via* the URL *http://www.example.com/~rbowen/cgi-bin/example.cgi* would be run as the user rbowen.



Allowing users to set up their own scripts to be automatically executed without some sort of review is asking for trouble, either from malicious users (perish the thought!) or exploitable insecure scripts.

See Also

- Recipe 8.1

5.7 Redirecting to Another Location

Problem

You want requests to a particular URL to be redirected to another server.

Solution

Use a *Redirect* directive in *httpd.conf*, and give an absolute URL on the second argument:

```
Redirect "/example" "http://www2.example.com/new/location"
```

Discussion

Whereas *Alias* maps a URL to something in the local filesystem, *Redirect* maps a URL to another URL, usually on another server. The second argument is a full URL and is sent back to the client (browser), which makes a second request for the new URL.

It is also important to know that the *Redirect* directive preserves path information, if there is any. Therefore, this recipe redirects a request for *http://original.example.com/example/something.html* to *http://other.example.com/new/location/something.html*.

Redirections come in several different flavors, too; you can specify which particular type of redirect you want to use by inserting the appropriate keyword between the *Redirect* directive and the first URL argument. All redirects instruct the client where the requested document is *now*; the different types of redirection inform where the client should look for the document in the future. If no keyword is specified, the temp meaning is used by default.

temp
: A temporary redirection is used when the document is not in the originally requested location at the moment, but is expected to be there again some time in the

future. So the client remembers the URL it used on the original request and will use it on future requests for the same document.

permanent
: A **permanent** redirection indicates that not only is the requested document not in the location specified by the client, but that the client should never look for it there again. In other words, the client should remember the *new* location indicated in the redirect response and look there in all subsequent requests for the resource.

gone
: This keyword means that the document doesn't exist in this location, and it shouldn't bother asking any more. This differs from the 404 Not Found error response in that the **gone** status admits that the document *was* once here, even though it isn't any more.

seeother
: A **seeother** redirection tells the client that the original document isn't located here any more and has been superseded by another one in a different location. That is, the original request might have been for:

  http://example.com/chapter2.html

  but the server answers with a **seeother** redirection to:

  http://bookname.com/edition-2/chapter2.html

  indicating that the desired content can be gotten at a different URL, and that a second request should be made to get it.

The semantics of the 303 **see other** status code suggest that you only use it if you're really familiar with it. Use the **temporary** keyword instead if you're not sure whether **see other** is appropriate.

By default, if no keyword is present, a **temporary** redirection is issued.

Here's an example of the various directive formats, including the HTTP status code number in case you want to use an *ErrorDocument* to customize the server's response text:

```
#
# These are equivalent, and return a response with a 302 status.
#
Redirect      /foo.html http://example.com/under-construction/foo.html
Redirect temp /foo.html http://example.com/under-construction/foo.html
RedirectTemp  /foo.html http://example.com/under-construction/foo.html
#
# These are equivalent to each other as well, returning a 301 status
#
Redirect permanent /foo.html http://example.com/relocated/foo.html
RedirectPermanent  /foo.html http://example.com/relocated/foo.html
```

```
#
# This tells the client that the old URL is dead, but the document
# content has been replaced by the specified new document.  It
# returns a 303 status.
#
Redirect seeother /foo.html http://example.com/relocated/bar.html
#
# Returns a 410 status, telling the client that the document has been
# intentionally removed and won't be coming back.  Note that there
# is no absoluteURL argument.
#
Redirect gone     /foo.html
```

See Also

- *http://httpd.apache.org/docs/2.2/mod/mod_alias.html*

5.8 Redirecting Several URLs to the Same Destination

Problem

You want to redirect a number of URLs to the same place. For example, you want to redirect requests for */fish* and */Fishing* to *http://fish.example.com/*.

Solution

Use *RedirectMatch* in *httpd.conf* to match against a regular expression:

```
RedirectMatch "^/[fF]ish(ing)?(/.*)?" "http://fish.example.com/$2"
```

Discussion

This recipe redirects requests on one server for URLs starting with fish or fishing, with either an upper-case or lower-case f, to a URL on another server, *fish.example.com*. As with *Redirect*, the path information, if any, is preserved. That is, a request for *http://original.server/Fishing/tackle.html* is redirected to *http://fish.example.com/tackle.html* so that existing relative links continue to work.

As with several of the earlier examples, *RedirectMatch* uses regular expressions to provide arbitrary text pattern matching.

See Also

- Appendix A

5.9 Permitting Case-Insensitive URLs

Problem

You want requested URLs to be valid whether uppercase or lowercase letters are used.

Solution

Use *mod_speling* to make URLs case-insensitive:

```
CheckSpelling On
```

Discussion

The *mod_speling* module is part of the standard Apache distribution but is not enabled by default, so you need to explicitly enable it.

In addition to making URLs case-insensitive, *mod_speling*, as the name implies, provides simple spellchecking capability. In particular, in the case of a "not found" error, *mod_speling* attempts to find files that may have been intended, based on similar spelling, transposed letters, or perhaps letters swapped with similar-looking numbers, like O for 0 and l for 1.

When *mod_speling* is installed, it may be turned on for a particular scope (such as a directory, virtual host, or the entire server) by setting the *CheckSpelling* directive to *On*.

And, yes, that is the correct spelling of the module name.

See Also

- *http://httpd.apache.org/docs/2.2/mod/mod_speling.html*

5.10 Showing Highlighted PHP Source without Symlinking

Problem

You want to be able to see the syntax-highlighted source to your PHP scripts without having to set up symbolic links for all of them.

Solution

Add a line such as the following to your *httpd.conf* or .htaccess file:

```
RewriteRule "^(.+\.php)s$" "$1" [T=application/x-httpd-php-source]
```

Or, for versions 2.2 and later:

```
RewriteRule "^(.+\.php)s$" "$1" [H=application/x-httpd-php-source]
```

Alternate Solution

Add a line such as the following to your *httpd.conf* file:

```
RewriteRule "^(.*\.php)s$" "/cgi-bin/show.php?file=$1" [PT,L]
```

Create a file named *show.php* as shown below, and put it in your server's */cgi-bin/* directory:

```
<?php
/*
 * Show the highlighted source of a PHP script without a symlink or copy.
 */
if ((! isset($_GET))
    || (! isset($_GET['file']))
    || (! ($file = $_GET['file']))) {
    /*
     * Missing required arguments, so bail.
     */
    return status('400 Bad Request',
                  "Data insufficient or invalid.\r\n");
}

$file = preg_replace('/\.phps$/', '.php', $file);
if (! preg_match('/\.php$/', $file)) {
    return status('403 Forbidden',
                  "Invalid document.\r\n");
}
$docroot = $_SERVER['DOCUMENT_ROOT'];
if ((! preg_match(";^$docroot;", $file))
    || (! preg_match(";^/home/[^/]+/public_html;", $file))) {
    return status('403 Forbidden',
                  "Invalid document requested.\r\n");
}
Header('Content-type: text/html; charset=iso-8859-1');
print highlight_file($file);
return;

function status($msg, $text) {
    Header("Status: $msg");
    Header('Content-type: text/plain; charset=iso-8859-1');
    Header('Content-length: ' . strlen($text));
    print $text;
}
?>
```

Discussion

The PHP interpreter has a built-in function to display PHP source code syntax color-coded. Ordinarily, this function is invoked for *.phps* files when your configuration file contains the following line:

```
AddHandler application/x-httpd-php-source .phps
```

However, in order to take advantage of this functionality, you need to make a copy, or symbolic link, of each PHP file you wish to treat this way, replacing the *.php* file extension with a *.phps* file extension. This is impractical and inconvenient.

The recipe given removes the need to do this, by rewriting any request for a *.phps* file to that same filename, but with a *.php* file extension instead, and associating the *php-source* handler with that request.

The *[H]* flag to the *RewriteRule* directive is new in version 2.2, and so for earlier versions, you will need to use the *[T]* flag instead, which provides a similar functionality.

The script in the alternate solution uses a built-in PHP function to display the script's source in highlighted form. It's a more complex solution than the one-liner change in the first solution, but it allows you to change the behaviour without having to restart the server, just by altering the script. The *preg_match* against **$docroot** verifies the requested file is under the server's *DocumentRoot*. The next *preg_match* also permits files in users' *public_html* directories.

See Also

- Recipe 2.5

5.11 Replacing Text in Requested URLs

Problem

You want to change all occurrences of **string**1 to **string**2 in a request's URI.

Solution

```
RewriteRule "(.*)string1(.*)" "$1string2$2" [N,PT]
```

Discussion

The **[N]** flag tells Apache to rerun the rewrite rule. This rule will get run repeatedly until the *RewriteCond* fails. Thus, it will get rerun as long as the URL contains the string that you want to replace. As soon as all occurrences of this string have been replaced, the *RewriteCond* will fail, and the rule will stop. The **[PT]** tells *mod_rewrite* to pass the rewritten URL on to the rest of Apache for any additional processing once the rewriting is done.

Care should be taken to avoid infinite loops, such as might happen if **string**1 is part of **string**2.

See Also

- Appendix A

5.12 Rewriting Path Information to CGI Arguments

Problem

You want to pass arguments as part of the URL but have these components of the URL rewritten as CGI QUERY_STRING arguments.

monospace

Solution

This is just an example, of course; make appropriate changes to the RewriteRule line to fit your own environment and needs:

```
RewriteEngine on
RewriteRule "^/book/([^/]*)/([^/]*)" "/cgi-bin/book.cgi?author=$1&subject=$2" [PT]
```

Discussion

One reason you might want or need to do this is if you're gluing together two legacy systems that do things in different ways, such as a client application and a vendor script.

This is *not* how an ampers
appears on people's screen
suggest changing these two
italicised lines to monospac

For example, the *RewriteRule* in the Solution will cause:

http://www.example.com/book/apache/bowen

to be rewritten as:

http://www.example.com/cgi-bin/book.cgi?subject=apache&author=bowen

The [PT] flag on the *RewriteRule* directive instructs Apache to keep processing the URL even after it has been modified; without the flag, the server would directly try to treat the rewritten URL as a filename, instead of continuing to the step at which it determines it's a CGI script. It also allows multiple *RewriteRule* directives to make additional refinements to the URL.

If the URL being rewritten already has a query string, or might, change the [PT] to [QSA,PT]. The QSA means "query string add" and will cause the query string generated by the rewrite to be added to the query string in the original URL. Without QSA, the original query string will be replaced.

See Also

- *http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html*

5.13 Denying Access to Unreferred Requests

Problem

You want to prevent other Web sites from using your images (or other types of documents) in their pages and allow your images to be accessed only if they were referred from your own site.

Solution

Put this in your *httpd.conf*:

```
RewriteEngine On
RewriteCond "%{HTTP_REFERER}" !=""
RewriteCond "%{HTTP_REFERER}" "!^http://mysite.com/.*$" [NC]
RewriteRule "\.(jpg|gif|png)$ - [F]
```

Discussion

This recipe is a series of *RewriteCond* directives, designed to determine whether an image file is requested from within a document on your site or if it is embedded in a page from another server. If the the latter, then the other site is stealing your images and needs to be stopped.

The first rule checks to see if the referer is even set. Some clients don't send a referer, and some browsers can be configured not to send referers. If we deny requests from all clients that don't send a referer, we'll deny a lot of valid requests; so we let these ones in.

Next, we check to see if the referer appears to be from some site other than our own. If so, we keep going through the rules. Otherwise, we'll stop processing the rewrite.

Finally, we check to see if this is a request for an image file. If the file is a nonimage file, such as an HTML file, then we want to allow people to link to these files from somewhere offsite.

If we've reached this point in the ruleset, we know that we have a request for an image file from within a page on another Web site. The *RewriteRule* matches a request and returns `Forbidden` to the client.

See Also

- *http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html*
- Recipe 5.14

5.14 Redirecting Unreferred Requests to an Explanation Page

Problem

When a request comes to your server without a referring URL being mentioned, you want to redirect it to a page that explains why you're not satisfying it.

Solution

Add lines such as the following to your configuration files:

```
RewriteEngine On
RewriteCond "%{HTTP_REFERER}" "^$"
RewriteRule "(.*)" "/cgi-bin/need-referer" [PT,E=ORIG:$1]
```

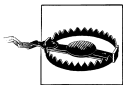
Be sure to put these directives in an appropriate scope, or people won't be able to type in your site's URLs at all!

Discussion

The specific problem being addressed in the Solution is requests coming to your server with no information about where they got the URL (which is passed in the `Referer` request header field). This frequently indicates someone surfing directly to the page rather than following a link, and in the case of images informaltable may be the result of a 'Save As' request on the user's part. If you want to only serve requests that came from links on your own site, you can change the *RewriteCond* directive to

```
RewriteCond "%{HTTP_REFERER}" "!http://%{SERVER_NAME}/" [NC]
```

The *RewriteRule* will pass the request to a CGI script named *need-referer*, and will put the original URI into the environment variable `ORIG` for the script to reference. Note that only the local portion of the URI is included; the hostname will not be, for instance.



The lack of a `Referer` request header field does not always signify skulduggery. Some people consider it none of a site's business how they came to it, and don't care to have their surfing patterns recorded. However, there's no way Apache can tell *why* the header field was omitted, so your *need-referer* script should explain the situation.

See Also

- Recipe 5.13

5.15 Rewriting Based on the Query String

Problem

You want to translate one URI into another based on the value of the query string.

Solution

Put this in your *httpd.conf*:

```
RewriteCond "%{QUERY_STRING}"    "^user=([^=]*)"
RewriteRule "/people"            "http://%1.users.example.com/" [R]
```

Discussion

mod_rewrite does not consider the query string as part of the URI for matching and rewriting purposes, so you need to treat it separately. The given example translates requests of the form:

http://example.com/people?user=jones
http://jones.users.example.com/

The `[R]` tells *mod_rewrite* to direct the browser to the URL constructed by the *RewriteRule* directive.

See Also

- *http://httpd.apache.org/docs/2.2/mod/mod_alias.html*

5.16 Redirecting All—or Part—of Your Server to SSL

Problem

You want certain parts of your non-SSL Web space to be redirected to a secured area.

Solution

You can redirect everything that is attached to port 80 with the following *RewriteRule*:

```
RewriteCond "%{SERVER_PORT}"        "^80$"
RewriteRule "^(.*)$"                "https://%{SERVER_NAME}$1" [R,L]
```

You can redirect particular URLs to a secure version:

```
RewriteRule "^/normal/secure(/.*)" "https://%{HTTP_HOST}$1" [R,L]
```

You can check to see whether the HTTPS environment variable is set:

```
RewriteCond "%{HTTPS}" "!=on"
RewriteRule "^(/secure/.*)" "https://%{HTTP_HOST}$1" [R,L]
```

Or, you can simply use the *Redirect* directive in the http section of *httpd.conf* file to to cause a URL to be served as HTTPS:

```
Redirect "/" "https://secure.example.com/"
```

Make sure that this appears only in in the **http** scope and not in the **https** scope, or all **https** requests will loop.

Discussion

The first solution causes all requests that come in on port 80 (normally the unencrypted HTTP port) to be redirected to the same locations on the current server but accessed through SSL. Note the use of SERVER_NAME; because this is a complete site redirection, it's simplest to use the server's official name for itself.

The directive shown in the second solution causes all portions of the server's Web space under *http://myhost/normal/secure* to be redirected to the SSL location rooted at *https://myhost/*. The use of HTTP_HOST rather than SERVER_NAME means that only the location and the scheme in the visitor's browser, not the server name.

Note that using HTTP_HOST might break here, because it may contain the port number, too. If you expect HTTP_HOST to contain the port number, therefore, you'll need to compensate for this in your ruleset.

Note that the paths to the SSL and non-SSL locations differ; if you want the paths to be the same except for the security, you can use something like the directives given in the third solution.

See Also

- *http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html*

5.17 Turning Directories into Hostnames

Problem

You want to migrate pathnames under a single hostname to distinct hostnames.

Solution

Use *RewriteRule* in *httpd.conf*:

```
RewriteRule "^/(patha|pathb|pathc)(/.*)" "http://$1.example.com$2" [R]
RewriteRule "^/([^./]*)(/.*)" "http://$1.example.com$2" [R]
RewriteRule "^/~([^./]*)(/.*)" "http://$1.example.com$2" [R]
```

Discussion

The first recipe redirects requests of the form *http://example.com/pathseg/some/file.html* to a different host, such as *http://pathseg.example.com/some/file.html*, but only for those requests in which `pathseg` is `patha`, `pathb`, or `pathc`.

The second recipe does the same thing, except that *any* top-level path segment is redirected in this manner.

The third recipe splits the difference, redirecting all "user" requests to distinct hosts with the same name as the user.

See Also

- *http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html*

5.18 Redirecting All Requests to a Single Host

Problem

You want all requests made of your system to be redirected to a specific host.

Solution

Put this in your *httpd.conf*:

```
RewriteCond "%{HTTP_HOST}"    "!^www.example.com"       [NC,OR]
RewriteCond "%{SERVER_NAME}" "!^www.example.com"       [NC]
RewriteRule "(.*)"            "http://www.example.com$1" [R]
```

Discussion

Any request handled by your server within the scope of the directives in the Solution (which aren't directed to the `www.example.com` host) is redirected there.

The two different *RewriteCond* directives are used to catch all requests made by some host other than `www.example.com`, regardless of the redirection method.

The `NC` (No Case) flag makes the regular expression case-insensitive. That is, it makes it match regardless of whether letters are upper- or lowercase.

The `OR` flag is a logical "or," allowing the two conditions to be strung together so that either one being true is a sufficient condition for the rule to be applied.

Finally, the `R` flag causes an actual Redirect to be issued, so that the browser will make another request for the generated URL.

See Also

- *http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html*

5.19 Turning Document Names into Arguments

Problem

You want to redirect requests for documents to a CGI script, or other handler, that gets the document names as an argument.

Solution

Use *RewriteRule* in *httpd.conf*:

```
RewriteRule "^/dir/([^./]*)\.html" "/dir/script.cgi?doc=$1" [PT]
```

Discussion

This solution causes all requests for HTML documents in the specified location to be turned into requests for a handler script that receives the document name as an argument in the QUERY_STRING environment variable.

The PT flag should be included to allow any appropriate subsequent URL rewriting or manipulation to be performed.

See Also

- *http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html*

5.20 Rewriting Elements between Path and Query String

Problem

You want to turn portions of a URL into query-string parameters, or *vice versa*.

Solution

To rewrite *http://example.com/path/to/5* to *http://example.com/path/to?id=5*:

```
RewriteRule "^(/path/to)/(\d+)" "$1?id=$2" [PT]
```

To go the other way,

```
RewriteCond "%{QUERY_STRING}" "\bid=(\d+)\b"
RewriteRule "(/path/to)" "$1/%2" [PT,QSA]
```

Note that the arguments to the *RewriteRule* directives need to be structured differently if they occur in a *.htaccess* file. The above solution illustrates the syntax if they're put into the server-wide configuration files.

Discussion

It is quite common for sites to have to change the way their URLs are structured, either moving portions into or out of the query-string portion. This often becomed necessary when the underlying software is changed—say from one blogging package to another. The new software requires URLs to be in a different format, but you don't want to invalidate all the old-style URLs that might be bookmarked or otherwise spread around the Web.

This is a perfect application for *mod_rewrite*, and quite simple to accomplish, as well, as demonstrated in the solution.

See Also

- *http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html*

5.21 Rewriting a Hostname to a Directory

Problem

You want requests for *http://bogus.example.com/* to be turned into requests for *http://example.com/bogus/*.

Solution

TBS

```
RewriteCond "%{HTTP_HOST}" "^([^.]+)\.example\.com" [NC]
RewriteRule "(.*)" "http://example.com/%1$1" [R]"
```

To do this transparently, without a redirect:

```
RewriteCond "%{HTTP_HOST}" "^([^.]+)\.example\.com$" [NC]
RewriteRule "(.*)" "/%1$1" [PT]
```

Discussion

This technique is occasionally needed when wildcard hostnames are being supported. It gives the illusion that URLs are pointing to separate hosts rather than subdirectories on only one system. Of course, using the redirection (`[R]`) flag will void the illusion because the replacement URL will be visible to the end user. If you want it to be completely transparent to the user, you can use the second option to get the equivalent result with a rewrite internal to the server that the client never sees.

See Also

- *http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html*

5.22 Turning URL Segments into Query Arguments

Problem

You want to turn requests of the form

```
http://example.com/foo/bar.html
```

into something like

```
http://example.com/cgi-bin/remap?page=foo/bar.html
```

Solution

Add a line such as the following to your configuration file:

```
RewriteRule "/(.*)" "/cgi-bin/remap?page=$1" [QSA,PT]
```

Discussion

The solution performs the rewrite in a manner completely transparent to the user, since all the changes to how to find things are done within the server rather than giving the users's client a new URL to load.

The `QSA` option allows any query-string information that was on the original request to be retained and merged with the one being added by the *RewriteRule*. The `PT` option tells the server to continue processing the request rather than treating it as completely finished. This is necessary in order to have the new URL, which involves a CGI script, handled correctly.

See Also

- Recipe 5.21

5.23 Using *AliasMatch*, *ScriptAliasMatch*, and *RedirectMatch*

Problem

You want to use the basic functionality of the *Alias* or *Redirect* directives, but need to handle a bunch of related cases without needing one directive for each.

Solution

Use the grouping syntax to manipulate the URI into filesystem paths:

```
AliasMatch "/users/(.*)/" "/home/webonly/$1/"
```

Force browsers to learn the new URLs for directories relocated to their own hosts:

```
RedirectMatch permanent "/projects/([^/]+)(/.*)" "http://$1.projectdomain.com$2"
```

Discussion

The *AliasMatch*, *ScriptAliasMatch*, and *RedirectMatch* directives use the same regular expression syntax as the rest of Apache's configuration language. They're not as powerful as the *mod_rewrite* directives, though, because they can't be made conditional and can't reference environment variables. However, they tend to have less of a performance impact on the server precisely because of their simpler syntax.

See Also

- Recipe 5.4
- Recipe 5.5