

# Apache HTTP Server Version 2.0

## URL Rewriting Guide

---

Originally written by  
*Ralf S. Engelschall* <[rse@apache.org](mailto:rse@apache.org)>  
December 1997

This document supplements the `mod_rewrite` reference documentation ([↗ ../mod/mod\\_rewrite.html](http://httpd.apache.org/docs-2.0/mod/mod_rewrite.html)) . It describes how one can use Apache's `mod_rewrite` to solve typical URL-based problems with which webmasters are commonly confronted. We give detailed descriptions on how to solve each problem by configuring URL rewriting rulesets.

- Introduction to `mod_rewrite`
- Practical Solutions
- URL Layout
- Content Handling
- Access Restriction
- Other

### Introduction to `mod_rewrite`

---

The Apache module `mod_rewrite` is a killer one, i.e. it is a really sophisticated module which provides a powerful way to do URL manipulations. With it you can do nearly all types of URL manipulations you ever dreamed about. The price you have to pay is to accept complexity, because `mod_rewrite`'s major drawback is that it is not easy to understand and use for the beginner. And even Apache experts sometimes discover new aspects where `mod_rewrite` can help.

In other words: With `mod_rewrite` you either shoot yourself in the foot the first time and never use it again or love it for the rest of your life because of its power. This paper tries to give you a few initial success events to avoid the first case by presenting already invented solutions to you.

### Practical Solutions

---

Here come a lot of practical solutions I've either invented myself or collected from other people's solutions in the past. Feel free to learn the black magic of URL rewriting from these examples.

ATTENTION: Depending on your server-configuration it can be necessary to slightly change the examples for your situation, e.g. adding the `[PT]` flag when additionally using `mod_alias` and `mod_userdir`, etc. Or rewriting a ruleset to fit in `.htaccess` context instead of per-server context. Always try to understand what a particular ruleset really does before you use it. It avoid problems.

### URL Layout

---

#### Canonical URLs

##### Description:

On some webservers there are more than one URL for a resource. Usually there are canonical URLs (which should be actually used and distributed) and those which are just shortcuts, internal

ones, etc. Independent of which URL the user supplied with the request he should finally see the canonical one only.

**Solution:**

We do an external HTTP redirect for all non-canonical URLs to fix them in the location view of the Browser and for all subsequent requests. In the example ruleset below we replace `/~user` by the canonical `/u/user` and fix a missing trailing slash for `/u/user`.

```
RewriteRule  ^/~([^/]+)/?(.*)    /u/$1/$2  [R]
RewriteRule  ^/([uge])/([^/]+)$  /$1/$2/   [R]
```

**Canonical Hostnames****Description:**

The goal of this rule is to force the use of a particular hostname, in preference to other hostnames which may be used to reach the same site. For example, if you wish to force the use of **www.example.com** instead of **example.com**, you might use a variant of the following recipe.

**Solution:**

```
# For sites running on a port other than 80
RewriteCond %{HTTP_HOST}    !^fully\.qualified\.domain\.name [NC]
RewriteCond %{HTTP_HOST}    !^$
RewriteCond %{SERVER_PORT}  !^80$
RewriteRule  ^/(.*)         http://fully.qualified.domain.name:%{SERVER_PORT}

# And for a site running on port 80
RewriteCond %{HTTP_HOST}    !^fully\.qualified\.domain\.name [NC]
RewriteCond %{HTTP_HOST}    !^$
RewriteRule  ^/(.*)         http://fully.qualified.domain.name/$1 [L,R]
```

**Moved DocumentRoot****Description:**

Usually the **DocumentRoot** of the webserver directly relates to the URL `/`. But often this data is not really of top-level priority, it is perhaps just one entity of a lot of data pools. For instance at our Intranet sites there are `/e/www/` (the homepage for WWW), `/e/sww/` (the homepage for the Intranet) etc. Now because the data of the **DocumentRoot** stays at `/e/www/` we had to make sure that all inlined images and other stuff inside this data pool work for subsequent requests.

**Solution:**

We redirect the URL `/` to `/e/www/`:

```
RewriteEngine on
RewriteRule  ^/$  /e/www/  [R]
```

Note that this can also be handled using the **RedirectMatch** directive:

```
RedirectMatch ^/$ http://example.com/e/www/
```

**Trailing Slash Problem****Description:**

Every webmaster can sing a song about the problem of the trailing slash on URLs referencing

directories. If they are missing, the server dumps an error, because if you say `/~quux/foo` instead of `/~quux/foo/` then the server searches for a *file* named `foo`. And because this file is a directory it complains. Actually it tries to fix it itself in most of the cases, but sometimes this mechanism need to be emulated by you. For instance after you have done a lot of complicated URL rewritings to CGI scripts etc.

**Solution:**

The solution to this subtle problem is to let the server add the trailing slash automatically. To do this correctly we have to use an external redirect, so the browser correctly requests subsequent images etc. If we only did a internal rewrite, this would only work for the directory page, but would go wrong when any images are included into this page with relative URLs, because the browser would request an in-lined object. For instance, a request for `image.gif` in `/~quux/foo/index.html` would become `/~quux/image.gif` without the external redirect!

So, to do this trick we write:

```

RewriteEngine on
RewriteBase /~quux/
RewriteRule ^foo$ foo/ [R]

```

The crazy and lazy can even do the following in the top-level `.htaccess` file of their homedir. But notice that this creates some processing overhead.

```

RewriteEngine on
RewriteBase /~quux/
RewriteCond %{REQUEST_FILENAME} -d
RewriteRule ^(.+[^/])$ $1/ [R]

```

**Webcluster through Homogeneous URL Layout****Description:**

We want to create a homogeneous and consistent URL layout over all WWW servers on a Intranet webcluster, i.e. all URLs (per definition server local and thus server dependent!) become actually server *independent*! What we want is to give the WWW namespace a consistent server-independent layout: no URL should have to include any physically correct target server. The cluster itself should drive us automatically to the physical target host.

**Solution:**

First, the knowledge of the target servers come from (distributed) external maps which contain information where our users, groups and entities stay. The have the form

```

user1 server_of_user1
user2 server_of_user2
:      :

```

We put them into files `map.xxx-to-host`. Second we need to instruct all servers to redirect URLs of the forms

```

/u/user/anypath
/g/group/anypath
/e/entity/anypath

```

to

```

http://physical-host/u/user/anypath
http://physical-host/g/group/anypath
http://physical-host/e/entity/anypath

```

when the URL is not locally valid to a server. The following ruleset does this for us by the help of the map files (assuming that server0 is a default server which will be used if a user has no entry in the map):

```

RewriteEngine on

RewriteMap      user-to-host      txt:/path/to/map.user-to-host
RewriteMap      group-to-host     txt:/path/to/map.group-to-host
RewriteMap      entity-to-host    txt:/path/to/map.entity-to-host

RewriteRule     ^/u/([^/]+)/?(.*) http://${user-to-host:$1|server0}/u
RewriteRule     ^/g/([^/]+)/?(.*) http://${group-to-host:$1|server0}/g
RewriteRule     ^/e/([^/]+)/?(.*) http://${entity-to-host:$1|server0}/e

RewriteRule     ^/([uge])/([^/]+)/?$      /$1/$2/.www/
RewriteRule     ^/([uge])/([^/]+)/([^.]+)  /$1/$2/.www/$3\

```

## Move Homedirs to Different Webserver

### Description:

Many webmasters have asked for a solution to the following situation: They wanted to redirect just all homedirs on a webserver to another webserver. They usually need such things when establishing a newer webserver which will replace the old one over time.

### Solution:

The solution is trivial with `mod_rewrite`. On the old webserver we just redirect all `~/user/anypath` URLs to `http://newserver/~user/anypath`.

```

RewriteEngine on
RewriteRule    ^/~(.+) http://newserver/~$1 [R,L]

```

## Structured Homedirs

### Description:

Some sites with thousands of users usually use a structured homedir layout, i.e. each homedir is in a subdirectory which begins for instance with the first character of the username. So, `~/foo/anypath` is `/home/f/foo/.www/anypath` while `~/bar/anypath` is `/home/b/bar/.www/anypath`.

### Solution:

We use the following ruleset to expand the tilde URLs into exactly the above layout.

```

RewriteEngine on
RewriteRule     ^/~(([a-z])[a-z0-9]+)(.*) /home/$2/$1/.www$3

```

## Filesystem Reorganization

### Description:

This really is a hardcore example: a killer application which heavily uses per-directory `RewriteRules` to get a smooth look and feel on the Web while its data structure is never

touched or adjusted. Background: *net.sw* is my archive of freely available Unix software packages, which I started to collect in 1992. It is both my hobby and job to do this, because while I'm studying computer science I have also worked for many years as a system and network administrator in my spare time. Every week I need some sort of software so I created a deep hierarchy of directories where I stored the packages:

```
drwxrwxr-x  2 netsw  users    512 Aug  3 18:39 Audio/
drwxrwxr-x  2 netsw  users    512 Jul  9 14:37 Benchmark/
drwxrwxr-x 12 netsw  users    512 Jul  9 00:34 Crypto/
drwxrwxr-x  5 netsw  users    512 Jul  9 00:41 Database/
drwxrwxr-x  4 netsw  users    512 Jul 30 19:25 Dicts/
drwxrwxr-x 10 netsw  users    512 Jul  9 01:54 Graphic/
drwxrwxr-x  5 netsw  users    512 Jul  9 01:58 Hackers/
drwxrwxr-x  8 netsw  users    512 Jul  9 03:19 InfoSys/
drwxrwxr-x  3 netsw  users    512 Jul  9 03:21 Math/
drwxrwxr-x  3 netsw  users    512 Jul  9 03:24 Misc/
drwxrwxr-x  9 netsw  users    512 Aug  1 16:33 Network/
drwxrwxr-x  2 netsw  users    512 Jul  9 05:53 Office/
drwxrwxr-x  7 netsw  users    512 Jul  9 09:24 SoftEng/
drwxrwxr-x  7 netsw  users    512 Jul  9 12:17 System/
drwxrwxr-x 12 netsw  users    512 Aug  3 20:15 Typesetting/
drwxrwxr-x 10 netsw  users    512 Jul  9 14:08 X11/
```

In July 1996 I decided to make this archive public to the world via a nice Web interface. "Nice" means that I wanted to offer an interface where you can browse directly through the archive hierarchy. And "nice" means that I didn't want to change anything inside this hierarchy - not even by putting some CGI scripts at the top of it. Why? Because the above structure should be later accessible via FTP as well, and I didn't want any Web or CGI stuff to be there.

#### Solution:

The solution has two parts: The first is a set of CGI scripts which create all the pages at all directory levels on-the-fly. I put them under `/e/netsw/.www/` as follows:

```
-rw-r--r--  1 netsw  users    1318 Aug  1 18:10 .wwwacl
drwxr-xr-x 18 netsw  users     512 Aug  5 15:51 DATA/
-rw-rw-rw-  1 netsw  users   372982 Aug  5 16:35 LOGFILE
-rw-r--r--  1 netsw  users     659 Aug  4 09:27 TODO
-rw-r--r--  1 netsw  users    5697 Aug  1 18:01 netsw-about.html
-rwxr-xr-x  1 netsw  users     579 Aug  2 10:33 netsw-access.pl
-rwxr-xr-x  1 netsw  users    1532 Aug  1 17:35 netsw-changes.cgi
-rwxr-xr-x  1 netsw  users    2866 Aug  5 14:49 netsw-home.cgi
drwxr-xr-x  2 netsw  users     512 Jul  8 23:47 netsw-img/
-rwxr-xr-x  1 netsw  users   24050 Aug  5 15:49 netsw-lsdir.cgi
-rwxr-xr-x  1 netsw  users    1589 Aug  3 18:43 netsw-search.cgi
-rwxr-xr-x  1 netsw  users    1885 Aug  1 17:41 netsw-tree.cgi
-rw-r--r--  1 netsw  users     234 Jul 30 16:35 netsw-unlimit.lst
```

The `DATA/` subdirectory holds the above directory structure, i.e. the real *net.sw* stuff and gets automatically updated via `rdist` from time to time. The second part of the problem remains: how to link these two structures together into one smooth-looking URL tree? We want to hide the `DATA/` directory from the user while running the appropriate CGI scripts for the various URLs. Here is the solution: first I put the following into the per-directory configuration file in the `DocumentRoot` of the server to rewrite the announced URL `/net.sw/` to the internal path `/e/netsw/`:

```

RewriteRule ^net.sw$      net.sw/          [R]
RewriteRule ^net.sw/(.*)$ e/netsw/$1

```

The first rule is for requests which miss the trailing slash! The second rule does the real thing. And then comes the killer configuration which stays in the per-directory config file `/e/netsw/.www/.wwwacl`:

```

Options          ExecCGI FollowSymLinks Includes MultiViews

RewriteEngine on

# we are reached via /net.sw/ prefix
RewriteBase     /net.sw/

# first we rewrite the root dir to
# the handling cgi script
RewriteRule    ^$                netsw-home.cgi      [L]
RewriteRule    ^index\.html$     netsw-home.cgi      [L]

# strip out the subdirs when
# the browser requests us from perdir pages
RewriteRule    ^./(/netsw-[^/]+/.)$ $1                    [L]

# and now break the rewriting for local files
RewriteRule    ^netsw-home\.cgi.* -                    [L]
RewriteRule    ^netsw-changes\.cgi.* -                  [L]
RewriteRule    ^netsw-search\.cgi.* -                   [L]
RewriteRule    ^netsw-tree\.cgi$ -                      [L]
RewriteRule    ^netsw-about\.html$ -                    [L]
RewriteRule    ^netsw-img/.*$ -                          [L]

# anything else is a subdir which gets handled
# by another cgi script
RewriteRule    !^netsw-lsdir\.cgi.* -                    [C]
RewriteRule    (.*)              netsw-lsdir.cgi/$1

```

Some hints for interpretation:

1. Notice the L (last) flag and no substitution field ('-') in the forth part
2. Notice the ! (not) character and the C (chain) flag at the first rule in the last part
3. Notice the catch-all pattern in the last rule

## NCSA imagemap to Apache `mod_imap`

### Description:

When switching from the NCSA webserver to the more modern Apache webserver a lot of people want a smooth transition. So they want pages which use their old NCSA `imagemap` program to work under Apache with the modern `mod_imap`. The problem is that there are a lot of hyperlinks around which reference the `imagemap` program via `/cgi-bin/imagemap/path/to/page.map`. Under Apache this has to read just `/path/to/page.map`.

### Solution:

We use a global rule to remove the prefix on-the-fly for all requests:

```

RewriteEngine on
RewriteRule ^/cgi-bin/imagemap(.*) $1 [PT]

```

## Search pages in more than one directory

### Description:

Sometimes it is necessary to let the webserver search for pages in more than one directory. Here MultiViews or other techniques cannot help.

### Solution:

We program an explicit ruleset which searches for the files in the directories.

```

RewriteEngine on

# first try to find it in custom/...
# ...and if found stop and be happy:
RewriteCond /your/docroot/dir1/{REQUEST_FILENAME} -f
RewriteRule ^(.+) /your/docroot/dir1/$1 [L]

# second try to find it in pub/...
# ...and if found stop and be happy:
RewriteCond /your/docroot/dir2/{REQUEST_FILENAME} -f
RewriteRule ^(.+) /your/docroot/dir2/$1 [L]

# else go on for other Alias or ScriptAlias directives,
# etc.
RewriteRule ^(.+) - [PT]

```

## Set Environment Variables According To URL Parts

### Description:

Perhaps you want to keep status information between requests and use the URL to encode it. But you don't want to use a CGI wrapper for all pages just to strip out this information.

### Solution:

We use a rewrite rule to strip out the status information and remember it via an environment variable which can be later dereferenced from within XSSI or CGI. This way a URL `/foo/S=java/bar/` gets translated to `/foo/bar/` and the environment variable named `STATUS` is set to the value "java".

```

RewriteEngine on
RewriteRule ^(.*)/S=( [^/]+ )/(.*) $1/$3 [E=STATUS:$2]

```

## Virtual User Hosts

### Description:

Assume that you want to provide `www.username.host.domain.com` for the homepage of username via just DNS A records to the same machine and without any virtualhosts on this machine.

### Solution:

For HTTP/1.0 requests there is no solution, but for HTTP/1.1 requests which contain a Host: HTTP header we can use the following ruleset to rewrite `http://www.username.host.com/anypath` internally to `/home/username/anypath`:

```

RewriteEngine on
RewriteCond    %{HTTP_HOST}          ^www\.[^.]+\\.host\.com$
RewriteRule    ^(.+)                  %{HTTP_HOST}$1          [C]
RewriteRule    ^www\.( [^.]+)\.host\.com(.*) /home/$1$2

```

## Redirect Homedirs For Foreigners

### Description:

We want to redirect homedir URLs to another webserver `www.somewhere.com` when the requesting user does not stay in the local domain `ourdomain.com`. This is sometimes used in virtual host contexts.

### Solution:

Just a rewrite condition:

```

RewriteEngine on
RewriteCond    %{REMOTE_HOST}      !^.+\.ourdomain\.com$
RewriteRule    ^(/~.+              http://www.somewhere.com/$1 [R,L]

```

## Redirect Failing URLs To Other Webserver

### Description:

A typical FAQ about URL rewriting is how to redirect failing requests on webserver A to webserver B. Usually this is done via `ErrorDocument` CGI-scripts in Perl, but there is also a `mod_rewrite` solution. But notice that this performs more poorly than using an `ErrorDocument` CGI-script!

### Solution:

The first solution has the best performance but less flexibility, and is less error safe:

```

RewriteEngine on
RewriteCond    /your/docroot/%{REQUEST_FILENAME}  !-f
RewriteRule    ^(.+)                               http://webserverB.dom/$1

```

The problem here is that this will only work for pages inside the `DocumentRoot`. While you can add more Conditions (for instance to also handle homedirs, etc.) there is better variant:

```

RewriteEngine on
RewriteCond    %{REQUEST_URI}      !-U
RewriteRule    ^(.+)               http://webserverB.dom/$1

```

This uses the URL look-ahead feature of `mod_rewrite`. The result is that this will work for all types of URLs and is a safe way. But it does a performance impact on the webserver, because for every request there is one more internal subrequest. So, if your webserver runs on a powerful CPU, use this one. If it is a slow machine, use the first approach or better a `ErrorDocument` CGI-script.

## Extended Redirection

### Description:

Sometimes we need more control (concerning the character escaping mechanism) of URLs on redirects. Usually the Apache kernels URL escape function also escapes anchors, i.e. URLs like `"url#anchor"`. You cannot use this directly on redirects with `mod_rewrite` because the

`uri_escape()` function of Apache would also escape the hash character. How can we redirect to such a URL?

### Solution:

We have to use a kludge by the use of a NPH-CGI script which does the redirect itself. Because here no escaping is done (NPH=non-parseable headers). First we introduce a new URL scheme `xredirect:` by the following per-server config-line (should be one of the last rewrite rules):

```
RewriteRule ^xredirect:(.+) /path/to/nph-xredirect.cgi/$1 \
    [T=application/x-httpd-cgi,L]
```

This forces all URLs prefixed with `xredirect:` to be piped through the `nph-xredirect.cgi` program. And this program just looks like:

```
#!/path/to/perl
##
## nph-xredirect.cgi -- NPH/CGI script for extended redirects
## Copyright (c) 1997 Ralf S. Engelschall, All Rights Reserved.
##

$| = 1;
$url = $ENV{'PATH_INFO'};

print "HTTP/1.0 302 Moved Temporarily\n";
print "Server: $ENV{'SERVER_SOFTWARE'}\n";
print "Location: $url\n";
print "Content-type: text/html\n";
print "\n";
print "<html>\n";
print "<head>\n";
print "<title>302 Moved Temporarily (EXTENDED)</title>\n";
print "</head>\n";
print "<body>\n";
print "<h1>Moved Temporarily (EXTENDED)</h1>\n";
print "The document has moved <a HREF=\"$url\">here</a>.<p>\n";
print "</body>\n";
print "</html>\n";

##EOF##
```

This provides you with the functionality to do redirects to all URL schemes, i.e. including the one which are not directly accepted by `mod_rewrite`. For instance you can now also redirect to `news:newsgroup` via

```
RewriteRule ^anyurl xredirect:news:newsgroup
```

Notice: You have not to put `[R]` or `[R,L]` to the above rule because the `xredirect:` need to be expanded later by our special "pipe through" rule above.

## Archive Access Multiplexer

### Description:

Do you know the great CPAN (Comprehensive Perl Archive Network) under <http://www.perl.com/CPAN> ([↗ http://www.perl.com/CPAN](http://www.perl.com/CPAN)) ? This does a redirect to one of

several FTP servers around the world which carry a CPAN mirror and is approximately near the location of the requesting client. Actually this can be called an FTP access multiplexing service. While CPAN runs via CGI scripts, how can a similar approach implemented via `mod_rewrite`?

**Solution:**

First we notice that from version 3.0.0 `mod_rewrite` can also use the "ftp:" scheme on redirects. And second, the location approximation can be done by a `RewriteMap` over the top-level domain of the client. With a tricky chained ruleset we can use this top-level domain as a key to our multiplexing map.

```

RewriteEngine on
RewriteMap    multiplex          txt:/path/to/map.cxan
RewriteRule   ^/CxAN/(.*)       %{REMOTE_HOST}::$1
RewriteRule   ^.+\.([a-zA-Z]+)::(.*)$  ${multiplex:$1|ftp.default.dom}$2

##
##  map.cxan -- Multiplexing Map for CxAN
##

de          ftp://ftp.cxan.de/CxAN/
uk          ftp://ftp.cxan.uk/CxAN/
com         ftp://ftp.cxan.com/CxAN/
:
##EOF##

```

**Time-Dependent Rewriting****Description:**

When tricks like time-dependent content should happen a lot of webmasters still use CGI scripts which do for instance redirects to specialized pages. How can it be done via `mod_rewrite`?

**Solution:**

There are a lot of variables named `TIME_XXX` for rewrite conditions. In conjunction with the special lexicographic comparison patterns `<STRING`, `>STRING` and `=STRING` we can do time-dependent redirects:

```

RewriteEngine on
RewriteCond   %{TIME_HOUR}%{TIME_MIN} >0700
RewriteCond   %{TIME_HOUR}%{TIME_MIN} <1900
RewriteRule   ^foo\.html$             foo.day.html
RewriteRule   ^foo\.html$             foo.night.html

```

This provides the content of `foo.day.html` under the URL `foo.html` from `07:00-19:00` and at the remaining time the contents of `foo.night.html`. Just a nice feature for a homepage...

**Backward Compatibility for YYYY to XXXX migration****Description:**

How can we make URLs backward compatible (still existing virtually) after migrating `document.YYYY` to `document.XXXX`, e.g. after translating a bunch of `.html` files to `.phtml`?

**Solution:**

We just rewrite the name to its basename and test for existence of the new extension. If it exists, we take that name, else we rewrite the URL to its original state.

```
# backward compatibility ruleset for
# rewriting document.html to document.phtml
# when and only when document.phtml exists
# but no longer document.html
RewriteEngine on
RewriteBase /~quux/
# parse out basename, but remember the fact
RewriteRule ^(.*)\.html$ $1 [C,E=WasHTML:yes]
# rewrite to document.phtml if exists
RewriteCond %{REQUEST_FILENAME}.phtml -f
RewriteRule ^(.*)$ $1.phtml [S=1]
# else reverse the previous basename cutout
RewriteCond %{ENV:WasHTML} ^yes$
RewriteRule ^(.*)$ $1.html
```

## Content Handling

---

### From Old to New (intern)

#### Description:

Assume we have recently renamed the page `foo.html` to `bar.html` and now want to provide the old URL for backward compatibility. Actually we want that users of the old URL even not recognize that the pages was renamed.

#### Solution:

We rewrite the old URL to the new one internally via the following rule:

```
RewriteEngine on
RewriteBase /~quux/
RewriteRule ^foo\.html$ bar.html
```

### From Old to New (extern)

#### Description:

Assume again that we have recently renamed the page `foo.html` to `bar.html` and now want to provide the old URL for backward compatibility. But this time we want that the users of the old URL get hinted to the new one, i.e. their browsers Location field should change, too.

#### Solution:

We force a HTTP redirect to the new URL which leads to a change of the browsers and thus the users view:

```
RewriteEngine on
RewriteBase /~quux/
RewriteRule ^foo\.html$ bar.html [R]
```

## Browser Dependent Content

#### Description:

At least for important top-level pages it is sometimes necessary to provide the optimum of browser dependent content, i.e. one has to provide a maximum version for the latest Netscape variants, a

minimum version for the Lynx browsers and a average feature version for all others.

### Solution:

We cannot use content negotiation because the browsers do not provide their type in that form. Instead we have to act on the HTTP header "User-Agent". The following config does the following: If the HTTP header "User-Agent" begins with "Mozilla/3", the page `foo.html` is rewritten to `foo.NS.html` and the rewriting stops. If the browser is "Lynx" or "Mozilla" of version 1 or 2 the URL becomes `foo.20.html`. All other browsers receive page `foo.32.html`. This is done by the following ruleset:

```

RewriteCond %{HTTP_USER_AGENT} ^Mozilla/3.*
RewriteRule ^foo\.html$ foo.NS.html [L]

RewriteCond %{HTTP_USER_AGENT} ^Lynx/.* [OR]
RewriteCond %{HTTP_USER_AGENT} ^Mozilla/[12].*
RewriteRule ^foo\.html$ foo.20.html [L]

RewriteRule ^foo\.html$ foo.32.html [L]

```

## Dynamic Mirror

### Description:

Assume there are nice webpages on remote hosts we want to bring into our namespace. For FTP servers we would use the `mirror` program which actually maintains an explicit up-to-date copy of the remote data on the local machine. For a webserver we could use the program `webcopy` which acts similar via HTTP. But both techniques have one major drawback: The local copy is always just as up-to-date as often we run the program. It would be much better if the mirror is not a static one we have to establish explicitly. Instead we want a dynamic mirror with data which gets updated automatically when there is need (updated data on the remote host).

### Solution:

To provide this feature we map the remote webpage or even the complete remote webarea to our namespace by the use of the *Proxy Throughput* feature (flag [P]):

```

RewriteEngine on
RewriteBase /~quux/
RewriteRule ^hotsheet/(.*)$ http://www.tstimpreso.com/hotsheet/

RewriteEngine on
RewriteBase /~quux/
RewriteRule ^usa-news\.html$ http://www.quux-corp.com/news/in

```

## Reverse Dynamic Mirror

### Description:

...

### Solution:

```

RewriteEngine on
RewriteCond /mirror/of/remotesite/$1 -U
RewriteRule ^http://www\.remotesite\.com/(.*)$ /mirror/of/remotesite/$1

```

## Retrieve Missing Data from Intranet

### Description:

This is a tricky way of virtually running a corporate (external) Internet webserver (`www.quux-corp.dom`), while actually keeping and maintaining its data on a (internal) Intranet webserver (`www2.quux-corp.dom`) which is protected by a firewall. The trick is that on the external webserver we retrieve the requested data on-the-fly from the internal one.

### Solution:

First, we have to make sure that our firewall still protects the internal webserver and that only the external webserver is allowed to retrieve data from it. For a packet-filtering firewall we could for instance configure a firewall ruleset like the following:

```

ALLOW Host www.quux-corp.dom Port >1024 --> Host www2.quux-corp.dom Port
DENY  Host *                      Port *      --> Host www2.quux-corp.dom Port

```

Just adjust it to your actual configuration syntax. Now we can establish the `mod_rewrite` rules which request the missing data in the background through the proxy throughput feature:

```

RewriteRule ^/~([^/]+)/?(.*) /home/$1/.www/$2
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^/home/([^/]+)/.www/?(.*) http://www2.quux-corp.dom/~$1/pub/$:

```

## Load Balancing

### Description:

Suppose we want to load balance the traffic to `www.foo.com` over `www[0-5].foo.com` (a total of 6 servers). How can this be done?

### Solution:

There are a lot of possible solutions for this problem. We will discuss first a commonly known DNS-based variant and then the special one with `mod_rewrite`:

#### 1. DNS Round-Robin

The simplest method for load-balancing is to use the DNS round-robin feature of BIND. Here you just configure `www[0-9].foo.com` as usual in your DNS with A(address) records, e.g.

```

www0    IN  A      1.2.3.1
www1    IN  A      1.2.3.2
www2    IN  A      1.2.3.3
www3    IN  A      1.2.3.4
www4    IN  A      1.2.3.5
www5    IN  A      1.2.3.6

```

Then you additionally add the following entry:

```

www     IN  CNAME  www0.foo.com.
        IN  CNAME  www1.foo.com.
        IN  CNAME  www2.foo.com.
        IN  CNAME  www3.foo.com.
        IN  CNAME  www4.foo.com.
        IN  CNAME  www5.foo.com.

```

```
IN CNAME www6.foo.com.
```

Notice that this seems wrong, but is actually an intended feature of BIND and can be used in this way. However, now when `www.foo.com` gets resolved, BIND gives out `www0-www6` - but in a slightly permuted/rotated order every time. This way the clients are spread over the various servers. But notice that this not a perfect load balancing scheme, because DNS resolve information gets cached by the other nameservers on the net, so once a client has resolved `www.foo.com` to a particular `wwwN.foo.com`, all subsequent requests also go to this particular name `wwwN.foo.com`. But the final result is ok, because the total sum of the requests are really spread over the various webservers.

## 2. DNS Load-Balancing

A sophisticated DNS-based method for load-balancing is to use the program `lbnamed` which can be found at <http://www.stanford.edu/~schemers/docs/lbnamed/lbnamed.html> ([↗ http://www.stanford.edu/~schemers/docs/lbnamed/lbnamed.html](http://www.stanford.edu/~schemers/docs/lbnamed/lbnamed.html)). It is a Perl 5 program in conjunction with auxilliary tools which provides a real load-balancing for DNS.

## 3. Proxy Throughput Round-Robin

In this variant we use `mod_rewrite` and its proxy throughput feature. First we dedicate `www0.foo.com` to be actually `www.foo.com` by using a single

```
www IN CNAME www0.foo.com.
```

entry in the DNS. Then we convert `www0.foo.com` to a proxy-only server, i.e. we configure this machine so all arriving URLs are just pushed through the internal proxy to one of the 5 other servers (`www1-www5`). To accomplish this we first establish a ruleset which contacts a load balancing script `lb.pl` for all URLs.

```
RewriteEngine on
RewriteMap lb prg:/path/to/lb.pl
RewriteRule ^/(.+)$ ${lb:$1} [P,L]
```

Then we write `lb.pl`:

```
#!/path/to/perl
##
## lb.pl -- load balancing script
##

$| = 1;

$name = "www"; # the hostname base
$first = 1; # the first server (not 0 here, because 0 is mys
$last = 5; # the last server in the round-robin
$domain = "foo.dom"; # the domainname

$cnt = 0;
while (<STDIN>) {
    $cnt = (($cnt+1) % ($last+1-$first));
    $server = sprintf("%s%d.%s", $name, $cnt+$first, $domain);
    print "http://$server/$_";
}

##EOF##
```

A last notice: Why is this useful? Seems like `www0.foo.com` still is overloaded? The answer is yes, it is overloaded, but with plain proxy throughput requests, only! All SSI, CGI, ePerl, etc. processing is completely done on the other machines. This is the essential point.

#### 4. Hardware/TCP Round-Robin

There is a hardware solution available, too. Cisco has a beast called LocalDirector which does a load balancing at the TCP/IP level. Actually this is some sort of a circuit level gateway in front of a webcluster. If you have enough money and really need a solution with high performance, use this one.

## New MIME-type, New Service

### Description:

On the net there are a lot of nifty CGI programs. But their usage is usually boring, so a lot of webmaster don't use them. Even Apache's Action handler feature for MIME-types is only appropriate when the CGI programs don't need special URLs (actually `PATH_INFO` and `QUERY_STRINGS`) as their input. First, let us configure a new file type with extension `.scgi` (for secure CGI) which will be processed by the popular `cgiwrap` program. The problem here is that for instance we use a Homogeneous URL Layout (see above) a file inside the user homedirs has the URL `/u/user/foo/bar.scgi`. But `cgiwrap` needs the URL in the form `/~user/foo/bar.scgi/`. The following rule solves the problem:

```
RewriteRule ^/[uge]/([^/]+)/\.www/(.+)\.scgi(.*) ...
... /internal/cgi/user/cgiwrap/~$1/$2.scgi$3 [NS,T=application/x-httpd-scgi]
```

Or assume we have some more nifty programs: `wwwlog` (which displays the `access.log` for a URL subtree and `wwidx` (which runs Glimpse on a URL subtree). We have to provide the URL area to these programs so they know on which area they have to act on. But usually this ugly, because they are all the times still requested from that areas, i.e. typically we would run the `swwidx` program from within `/u/user/foo/` via hyperlink to

```
/internal/cgi/user/swwidx?i=/u/user/foo/
```

which is ugly. Because we have to hard-code **both** the location of the area **and** the location of the CGI inside the hyperlink. When we have to reorganize the area, we spend a lot of time changing the various hyperlinks.

### Solution:

The solution here is to provide a special new URL format which automatically leads to the proper CGI invocation. We configure the following:

```
RewriteRule ^/([uge])/([^/]+)/(?.*)/* /internal/cgi/user/wwidx?i=/$1/
RewriteRule ^/([uge])/([^/]+)/(?.*):log /internal/cgi/user/wwwlog?f=/$1/
```

Now the hyperlink to search at `/u/user/foo/` reads only

```
HREF="*" "
```

which internally gets automatically transformed to

```
/internal/cgi/user/wwidx?i=/u/user/foo/
```

The same approach leads to an invocation for the access log CGI program when the hyperlink `:log` gets used.

## From Static to Dynamic

### Description:

How can we transform a static page `foo.html` into a dynamic variant `foo.cgi` in a seamless way, i.e. without notice by the browser/user.

### Solution:

We just rewrite the URL to the CGI-script and force the correct MIME-type so it gets really run as a CGI-script. This way a request to `/~quux/foo.html` internally leads to the invocation of `/~quux/foo.cgi`.

```

RewriteEngine on
RewriteBase /~quux/
RewriteRule ^foo\.html$ foo.cgi [T=application/x-httpd-cgi]

```

## On-the-fly Content-Regeneration

### Description:

Here comes a really esoteric feature: Dynamically generated but statically served pages, i.e. pages should be delivered as pure static pages (read from the filesystem and just passed through), but they have to be generated dynamically by the webserver if missing. This way you can have CGI-generated pages which are statically served unless one (or a cronjob) removes the static contents. Then the contents gets refreshed.

### Solution:

This is done via the following ruleset:

```

RewriteCond %{REQUEST_FILENAME} !-s
RewriteRule ^page\.html$ page.cgi [T=application/x-httpd-cgi,L]

```

Here a request to `page.html` leads to a internal run of a corresponding `page.cgi` if `page.html` is still missing or has filesize null. The trick here is that `page.cgi` is a usual CGI script which (additionally to its `STDOUT`) writes its output to the file `page.html`. Once it was run, the server sends out the data of `page.html`. When the webmaster wants to force a refresh the contents, he just removes `page.html` (usually done by a cronjob).

## Document With Autorefresh

### Description:

Wouldn't it be nice while creating a complex webpage if the webbrower would automatically refresh the page every time we write a new version from within our editor? Impossible?

### Solution:

No! We just combine the MIME multipart feature, the webserver NPH feature and the URL manipulation power of `mod_rewrite`. First, we establish a new URL feature: Adding just `:refresh` to any URL causes this to be refreshed every time it gets updated on the filesystem.

```

RewriteRule ^(/[uge]/[^\s]+/?.*):refresh /internal/cgi/apache/nph-refresh

```

Now when we reference the URL

```
/u/foo/bar/page.html:refresh
```

this leads to the internal invocation of the URL

```
/internal/cgi/apache/nph-refresh?f=/u/foo/bar/page.html
```

The only missing part is the NPH-CGI script. Although one would usually say "left as an exercise to the reader" ;-) I will provide this, too.

```
#!/sw/bin/perl
##
## nph-refresh -- NPH/CGI script for auto refreshing pages
## Copyright (c) 1997 Ralf S. Engelschall, All Rights Reserved.
##
$| = 1;

# split the QUERY_STRING variable
@pairs = split(/&/, $ENV{'QUERY_STRING'});
foreach $pair (@pairs) {
    ($name, $value) = split(/=/, $pair);
    $name =~ tr/A-Z/a-z/;
    $name = 'QS_' . $name;
    $value =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;
    eval "\$$name = \"$value\"";
}
$QS_s = 1 if ($QS_s eq '');
$QS_n = 3600 if ($QS_n eq '');
if ($QS_f eq '') {
    print "HTTP/1.0 200 OK\n";
    print "Content-type: text/html\n\n";
    print "&lt;b&gt;ERROR&lt;/b&gt;: No file given\n";
    exit(0);
}
if (! -f $QS_f) {
    print "HTTP/1.0 200 OK\n";
    print "Content-type: text/html\n\n";
    print "&lt;b&gt;ERROR&lt;/b&gt;: File $QS_f not found\n";
    exit(0);
}

sub print_http_headers_multipart_begin {
    print "HTTP/1.0 200 OK\n";
    $bound = "ThisRandomString12345";
    print "Content-type: multipart/x-mixed-replace;boundary=$bound\n";
    &print_http_headers_multipart_next;
}

sub print_http_headers_multipart_next {
    print "\n--$bound\n";
}

sub print_http_headers_multipart_end {
    print "\n--$bound--\n";
}

sub displayhtml {
```

```

    local($buffer) = @_;
    $len = length($buffer);
    print "Content-type: text/html\n";
    print "Content-length: $len\n\n";
    print $buffer;
}

sub readfile {
    local($file) = @_;
    local(*FP, $size, $buffer, $bytes);
    ($x, $x, $x, $x, $x, $x, $x, $size) = stat($file);
    $size = sprintf("%d", $size);
    open(FP, "&lt;$file");
    $bytes = sysread(FP, $buffer, $size);
    close(FP);
    return $buffer;
}

$buffer = &readfile($QS_f);
&print_http_headers_multipart_begin;
&displayhtml($buffer);

sub mystat {
    local($file) = $_[0];
    local($time);

    ($x, $x, $x, $x, $x, $x, $x, $x, $x, $mtime) = stat($file);
    return $mtime;
}

$mtimeL = &mystat($QS_f);
$mtime = $mtime;
for ($n = 0; $n &lt; $QS_n; $n++) {
    while (1) {
        $mtime = &mystat($QS_f);
        if ($mtime ne $mtimeL) {
            $mtimeL = $mtime;
            sleep(2);
            $buffer = &readfile($QS_f);
            &print_http_headers_multipart_next;
            &displayhtml($buffer);
            sleep(5);
            $mtimeL = &mystat($QS_f);
            last;
        }
        sleep($QS_s);
    }
}

&print_http_headers_multipart_end;

exit(0);

##EOF##

```

## Mass Virtual Hosting

**Description:**

The `<VirtualHost>` feature of Apache is nice and works great when you just have a few dozens virtual hosts. But when you are an ISP and have hundreds of virtual hosts to provide this feature is not the best choice.

**Solution:**

To provide this feature we map the remote webpage or even the complete remote webarea to our namespace by the use of the *Proxy Throughput* feature (flag [ P ]):

```
##
## vhost.map
##
www.vhost1.dom:80 /path/to/docroot/vhost1
www.vhost2.dom:80 /path/to/docroot/vhost2
:
www.vhostN.dom:80 /path/to/docroot/vhostN
```

```
##
## httpd.conf
##
:
# use the canonical hostname on redirects, etc.
UseCanonicalName on

:
# add the virtual host in front of the CLF-format
CustomLog /path/to/access_log "%{VHOST}e %h %l %u %t \"%r\" %>s %b"
:

# enable the rewriting engine in the main server
RewriteEngine on

# define two maps: one for fixing the URL and one which defines
# the available virtual hosts with their corresponding
# DocumentRoot.
RewriteMap lowercase int:tolower
RewriteMap vhost txt:/path/to/vhost.map

# Now do the actual virtual host mapping
# via a huge and complicated single rule:
#
# 1. make sure we don't map for common locations
RewriteCond %{REQUEST_URI} !^/commonurl1/. *
RewriteCond %{REQUEST_URI} !^/commonurl2/. *
:
RewriteCond %{REQUEST_URI} !^/commonurlN/. *
#
# 2. make sure we have a Host header, because
# currently our approach only supports
# virtual hosting through this header
RewriteCond %{HTTP_HOST} !^$
#
# 3. lowercase the hostname
RewriteCond ${lowercase:%{HTTP_HOST}|NONE} ^(.+)$
#
# 4. lookup this hostname in vhost.map and
```

```

#       remember it only when it is a path
#       (and not "NONE" from above)
RewriteCond  ${vhost:%1}  ^(/.*)$
#
#   5. finally we can map the URL to its docroot location
#       and remember the virtual host for logging puposes
RewriteRule  ^(/.*)$    %1/%1  [E=VHOST:${lowercase:%{HTTP_HOST}}]
:

```

## Access Restriction

---

### Blocking of Robots

#### Description:

How can we block a really annoying robot from retrieving pages of a specific webarea? A `/robots.txt` file containing entries of the "Robot Exclusion Protocol" is typically not enough to get rid of such a robot.

#### Solution:

We use a ruleset which forbids the URLs of the webarea `/~quux/foo/arc/` (perhaps a very deep directory indexed area where the robot traversal would create big server load). We have to make sure that we forbid access only to the particular robot, i.e. just forbidding the host where the robot runs is not enough. This would block users from this host, too. We accomplish this by also matching the User-Agent HTTP header information.

```

RewriteCond  %{HTTP_USER_AGENT}    ^NameOfBadRobot.*
RewriteCond  %{REMOTE_ADDR}        ^123\.45\.67\.[8-9]$
RewriteRule  ^/~quux/foo/arc/.+    -    [F]

```

### Blocked Inline-Images

#### Description:

Assume we have under `http://www.quux-corp.de/~quux/` some pages with inlined GIF graphics. These graphics are nice, so others directly incorporate them via hyperlinks to their pages. We don't like this practice because it adds useless traffic to our server.

#### Solution:

While we cannot 100% protect the images from inclusion, we can at least restrict the cases where the browser sends a HTTP Referer header.

```

RewriteCond  %{HTTP_REFERER}    !^$
RewriteCond  %{HTTP_REFERER}    !^http://www.quux-corp.de/~quux/.*$ [NC]
RewriteRule  .*\.gif$           -    [F]

```

```

RewriteCond  %{HTTP_REFERER}    !^$
RewriteCond  %{HTTP_REFERER}    !.* /foo-with-gif\.html$
RewriteRule  ^inlined-in-foo\.gif$ -    [F]

```

### Host Deny

#### Description:

How can we forbid a list of externally configured hosts from using our server?

**Solution:**

For Apache >= 1.3b6:

```

RewriteEngine on
RewriteMap    hosts-deny    txt:/path/to/hosts.deny
RewriteCond   ${hosts-deny:%{REMOTE_HOST}|NOT-FOUND} !=NOT-FOUND [OR]
RewriteCond   ${hosts-deny:%{REMOTE_ADDR}|NOT-FOUND} !=NOT-FOUND
RewriteRule   ^/.* - [F]

```

For Apache <= 1.3b6:

```

RewriteEngine on
RewriteMap    hosts-deny    txt:/path/to/hosts.deny
RewriteRule   ^/(.*)$ ${hosts-deny:%{REMOTE_HOST}|NOT-FOUND}/$1
RewriteRule   !^NOT-FOUND/.* - [F]
RewriteRule   ^NOT-FOUND/(.*)$ ${hosts-deny:%{REMOTE_ADDR}|NOT-FOUND}/$1
RewriteRule   !^NOT-FOUND/.* - [F]
RewriteRule   ^NOT-FOUND/(.*)$ /$1

```

```

##
## hosts.deny
##
## ATTENTION! This is a map, not a list, even when we treat it as such.
## mod_rewrite parses it for key/value pairs, so at least a
## dummy value "-" must be present for each entry.
##
193.102.180.41 -
bsdt11.sdm.de -
192.76.162.40 -

```

## Proxy Deny

**Description:**

How can we forbid a certain host or even a user of a special host from using the Apache proxy?

**Solution:**

We first have to make sure `mod_rewrite` is below(!) `mod_proxy` in the Configuration file when compiling the Apache webserver. This way it gets called *before* `mod_proxy`. Then we configure the following for a host-dependent deny...

```

RewriteCond   %{REMOTE_HOST}    ^badhost\.mydomain\.com$
RewriteRule   !^http://[^\./]\.mydomain.com.* - [F]

```

...and this one for a user@host-dependent deny:

```

RewriteCond   %{REMOTE_IDENT}@%{REMOTE_HOST}    ^badguy@badhost\.mydomain
RewriteRule   !^http://[^\./]\.mydomain.com.* - [F]

```

## Special Authentication Variant

**Description:**

Sometimes a very special authentication is needed, for instance a authentication which checks for a

set of explicitly configured users. Only these should receive access and without explicit prompting (which would occur when using the Basic Auth via `mod_auth`).

**Solution:**

We use a list of rewrite conditions to exclude all except our friends:

```
RewriteCond %{REMOTE_IDENT}@%{REMOTE_HOST} !^friend1@client1.quux-corp
RewriteCond %{REMOTE_IDENT}@%{REMOTE_HOST} !^friend2@client2.quux-corp
RewriteCond %{REMOTE_IDENT}@%{REMOTE_HOST} !^friend3@client3.quux-corp
RewriteRule ^/~quux/only-for-friends/ -
```

## Referer-based Deflector

**Description:**

How can we program a flexible URL Deflector which acts on the "Referer" HTTP header and can be configured with as many referring pages as we like?

**Solution:**

Use the following really tricky ruleset...

```
RewriteMap deflector txt:/path/to/deflector.map

RewriteCond %{HTTP_REFERER} !=" "
RewriteCond ${deflector:%{HTTP_REFERER}} ^-$
RewriteRule ^.* %{HTTP_REFERER} [R,L]

RewriteCond %{HTTP_REFERER} !=" "
RewriteCond ${deflector:%{HTTP_REFERER}|NOT-FOUND} !=NOT-FOUND
RewriteRule ^.* ${deflector:%{HTTP_REFERER}} [R,L]
```

... in conjunction with a corresponding rewrite map:

```
##
## deflector.map
##

http://www.badguys.com/bad/index.html -
http://www.badguys.com/bad/index2.html -
http://www.badguys.com/bad/index3.html http://somewhere.com/
```

This automatically redirects the request back to the referring page (when "-" is used as the value in the map) or to a specific URL (when an URL is specified in the map as the second argument).

## Other

---

### External Rewriting Engine

**Description:**

A FAQ: How can we solve the FOO/BAR/QUUX/etc. problem? There seems no solution by the use of `mod_rewrite`...

**Solution:**

Use an external `RewriteMap`, i.e. a program which acts like a `RewriteMap`. It is run once on startup of Apache receives the requested URLs on `STDIN` and has to put the resulting (usually

rewritten) URL on STDOUT (same order!).

```
RewriteEngine on
RewriteMap    quux-map      prg:/path/to/map.quux.pl
RewriteRule   ^/~quux/(.*)$  /~quux/${quux-map:$1}
```

```
#!/path/to/perl

#  disable buffered I/O which would lead
#  to deadlocks for the Apache server
$| = 1;

#  read URLs one per line from stdin and
#  generate substitution URL on stdout
while (<>) {
    s|^foo/|bar/|;
    print $_;
}
```

This is a demonstration-only example and just rewrites all URLs `/~quux/foo/...` to `/~quux/bar/...`. Actually you can program whatever you like. But notice that while such maps can be **used** also by an average user, only the system administrator can **define** it.

---

**Copyright 1995-2005 The Apache Software Foundation or its licensors, as applicable.  
Licensed under the Apache License, Version 2.0.**