CHAPTER 3

Logging

Apache can, and usually does, record information about every request it processes. Controlling how this is done and extracting useful information out of these logs after the fact is at least as important as gathering the information in the first place.

The logfiles may record two types of data: information about the request itself, and possibly one or more messages about abnormal conditions encountered during processing (such as file permissions). You, as the Webmaster, have a limited amount of control over the logging of error conditions, but a great deal of control over the format and amount of information logged about request processing (*activity logging*). The server may log activity information about a request in multiple formats in multiple log files, but it will only record a single copy of an error message.

One aspect of activity logging you should be aware of is that the log entry is formatted and written *after* the request has been completely processed. This means that the interval between the time a request begins and when it finishes may be long enough to make a difference.

For example, if your logfiles are rotated while a particularly large file is being downloaded, the log entry for the request will appear in the new logfile when the request completes, rather than in the old logfile when the request was started. In contrast, an error message is written to the error log as soon as it is encountered.

The Web server will continue to record information in its logfiles as long as it's running. This can result in extremely large logfiles for a busy site and uncomfortably large ones even for a modest site. To keep the file sizes from growing ever larger, most sites rotate or *roll over* their logfiles on a semi-regular basis. Rolling over a logfile simply means persuading the server to stop writing to the current file and start recording to a new one. Due to Apache's determination to see that no records are lost, cajoling it to do this according to a specific timetable may require a bit of effort; some of the recipes in this chapter cover how to accomplish the task successfully and reliably (see Recipe 3.8 and Recipe 3.9).

The log declaration directives, *CustomLog* and *ErrorLog*, can appear inside <*Virtual-Host*> containers, outside them (in what's called the *main* or *global server*, or sometimes the *global scope*), or both. Entries will only be logged in one set or the other; if a *<VirtualHost>* container applies to the request or error and has an applicable log directive, the message will be written only there and won't appear in any globally declared files. On the other hand, if no *<VirtualHost>* log directive applies, the server will fall back on logging the entry according to the global directives.

However, whichever scope is used for determining what logging directives to use, all *CustomLog* directives in that scope are processed and treated independently. That is, if you have a *CustomLog* directive in the global scope and two inside a *<VirtualHost>* container, *both* of these will be used. Similarly, if a *CustomLog* directive uses the `env=` option, it has no effect on what requests will be logged by other *CustomLog* directives in the same scope.

Activity logging has been around since the Web first appeared, and it didn't take long for the original users to decide what items of information they wanted logged. The result is called the *common log format* (CLF). In Apache terms, this format is:

```
"%h %l %u %t \"%r\" %>s %b"
```

That is, it logs the client's hostname or IP address, the name of the user on the client (as defined by RFC 1413 and if Apache has been told to snoop for it with an *IdentityCheck On* directive), the username with which the client authenticated (if weak access controls are being imposed by the server), the time at which the request was received, the actual HTTP request line, the final status of the server's processing of the request, and the number of bytes of content that were sent in the server's response.

Before long, as the HTTP protocol advanced, the common log format was found to be wanting, so an enhanced format, called the *combined log format*, was created:

```
"%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\""
```

The two additions were the `Referer` (it's spelled incorrectly in the specifications) and the `User-agent`. These are the URL of the page that linked to the document being requested, and the name and version of the browser or other client software making the request.

Both of these formats are widely used, and many logfile analysis tools assume log entries are made in one or the other.

The Apache Web server's standard activity logging module allows you to create your own formats; it is highly configurable and is called (surprise!) *mod_log_config*. Apache 2.0 has an additional module, *mod_logio*, which enhances *mod_log_config* with the ability to log the number of bytes actually transmitted or received over the network. If these doesn't meet your requirements, though, there are a significant number of third-party modules available from the module registry at *http://modules.apache.org/*.

The status code entry in the `common` and `combined` log formats deserve some mention, because its meaning is not immediately clear. The status codes are defined by the HTTP protocol specification documents (currently RFC 2616 at *ftp://ftp.isi.edu/in-notes/*

rfc2616.txt). Table 3-1 gives a brief description of the codes defined in the HTTP specification at the time of this writing; other specifications (such as that for WebDAV) define additional staus conditions, but we're not going to include them here because they're more advanced and there are lots of them.

Table 3-1. HTTP status codes

Code	Abstract
Informational 1xx	
100	Continue
101	Switching protocols
Successful 2xx	
200	OK
201	Created
202	Accepted
203	Nonauthoritative information
204	No content
205	Reset content
206	Partial content
Redirection 3xx	
300	Multiple choices
301	Moved permanently
302	Found
303	See other
304	Not modified
305	Use proxy
306	(Unused)
307	Temporary redirect
Client error 4xx	
400	Bad request
401	Unauthorized
402	Payment required
403	Forbidden
404	Not found
405	Method not allowed
406	Not acceptable
407	Proxy authentication required

Code	Abstract
408	Request timeout
409	Conflict
410	Gone
411	Length required
412	Precondition failed
413	Request entity too large
414	Request-URI too long
415	Unsupported media type
416	Requested range not satisfiable
417	Expectation failed
Server error 5xx	
500	Internal server error
501	Not implemented
502	Bad gateway
503	Service unavailable
504	Gateway timeout
505	HTTP version not supported

The one-line abstracts shown in Table 3-1 are sometimes terse to the point of being confusing, but they should at least give you an inkling of what the server thinks happened. The first digit is used to separate the codes into classes or categories; for example, all codes starting with 5 indicate there is a problem handling the request, and the server thinks the problem is on its end rather than on the client's end.

For a complete description of the various status codes, you'll need to read a document about the HTTP protocol or the RFC itself.

3.1 Getting More Details in Your Log Entries

Problem

You want to add a little more detail to your access log entries.

Solution

Use the *combined* log format, rather than the *common* log format:

```
CustomLog logs/access_log combined
```

Discussion

The default Apache logfile enables logging with the *common* log format, but it also provides the *combined* log format as a predefined *LogFormat* directive.

The *combined* log format offers two additional pieces of information not included in the *common* log format: the `Referer` (where the client linked from) and the `User-agent` (what browser they are using).

Every major logfile parsing software package is able to handle the *combined* format as well as the *common* format, and many of them give additional statistics based on these added fields. So you lose nothing by using this format and potentially gain some additional information.

See Also

- *http://httpd.apache.org/docs/2.1/mod/mod_log_config.html*
- *http://httpd.apache.org/docs/2.2/mod/mod_log_config.html*

3.2 Getting More Detailed Errors

Problem

You want more information in the error log in order to debug a problem.

Solution

Change (or add) the *LogLevel* line in your *httpd.conf* file. There are several possible arguments, which are enumerated below:

For example:

```
LogLevel Debug
```

Discussion

There are several hierarchical levels of error logging available, each identified by its own keyword. The default value of *LogLevel* is *warn*. Listed in descending order of importance, the possible values are:

`emerg`
    Emergencies; Web server is unusable

`alert`
    Action must be taken immediately

`crit`
    Critical conditions

`error`
: Error conditions

`warn`
: Warning conditions

`notice`
: Normal but significant condition

`info`
: Informational

`debug`
: Debug-level messages

emerg results in the least information being recorded and *debug* in the most. However, at `debug` level a lot of information will probably be recorded that is unrelated to the issue you're investigating, so it's a good idea to revert to the previous setting when the problem is solved.

Even though the various logging levels are hierarchical in nature, one oddity is that *notice* level messages are *always* logged regardless of the setting of the *LogLevel* directive.

The severity levels are rather loosely defined and even more loosely applied. In other words, the severity at which a particular error condition gets logged is decided at the discretion of the developer who wrote the code—your opinion may differ.

Here are some sample messages of various severities:

```
[Thu Apr 18 01:37:40 2002] [alert] [client 64.152.75.26] /home/smith/public_html/
     test/.htaccess: Invalid command 'Test', perhaps mis-spelled or defined by a
     module not included in the server configuration
[Thu Apr 25 22:21:58 2002] [error] PHP Fatal error:  Call to undefined function:
     decode_url(  ) in /usr/apache/htdocs/foo.php on line 8
[Mon Apr 15 09:31:37 2002] [warn] pid file /usr/apache/logs/httpd.pid overwritten --
     Unclean shutdown of previous Apache run?
[Mon Apr 15 09:31:38 2002] [info] Server built: Apr 12 2002 09:14:06
[Mon Apr 15 09:31:38 2002] [notice] Accept mutex: sysvsem (Default: sysvsem)
```

These are fairly normal messages that you might encounter on a production Web server. If you set the logging level to `Debug`, however, you might see many more messages of cryptic import, such as:

```
[Thu Mar 28 10:29:50 2002] [debug] proxy_cache.c(992): No CacheRoot, so no caching.
     Declining.
[Thu Mar 28 10:29:50 2002] [debug] proxy_http.c(540): Content-Type: text/html
```

These are exactly what they seem to be: debugging messages intended to help an Apache developer figure out what the proxy module is doing.

See Also

At the time of this writing, there is an effort underway to provide a dictionary of Apache error messages, what they mean, and what to do about the conditions they report, but it doesn't have anything concrete to show at this point. When it does, it should be announced at the Apache server developer site:

> *http://httpd.apache.org/dev/*

It will be mentioned on this book's companion Web site, as well:

> *http://Apache-Cookbook.Com/*

In addition, see the detailed documentation of the *LogLevel* directive at the Apache site:

> *http://httpd.apache.org/docs/2.2/mod/core.html#loglevel*

3.3 Logging POST Contents

Problem

You want to record data submitted with the POST method, such as from a Web form.

Solution

Ensure that *mod_dumpio* is installed and enabled, and put the following in your configuration file:

```
# DumpIOLogLevel notice - 2.3.x and later
LogLevel debug
DumpIOInput On
```

Or, with *mod_security*:

```
SecAuditLogType Concurrent
SecAuditLogStorageDir /var/www/audit_log/data/
SecAuditLog /var/www/audit_log/index
SecAuditLogParts ABCFHZ
```

Discussion

mod_dumpio is a new module in Apache 2.0 (that is to say, it's not available for Apache 1.3) which allows the complete input and output of each HTTP transaction to be logged. In the example above, we're enabling input logging only, using the *DumpIOInput* directive.

On Apache 2.0 and 2.2, *LogLevel* needs to be set to *debug* in order for these records to be logged. In 2.3 and later, there's a new directive *DumpIOLogLevel* that allows you to set the *LogLevel* at which the entries will be logged. For example, if you set *Dump-*

IOLogLevel to *notice*, then these entries will be logged when *LogLevel* is set to *notice* or higher.

Log entries for POST data will look like:

```
[Sun Feb 11 16:49:27 2007] [debug] mod_dumpio.c(51): mod_dumpio: dumpio_in (data-HEAP): 11 bytes
[Sun Feb 11 16:49:27 2007] [debug] mod_dumpio.c(67): mod_dumpio: dumpio_in (data-HEAP): foo=example
```

In the log entry shown here, the form value `foo` was set to `example`.

The output from *mod_dumpio* is very noisy. A typical request may generate somewhere between 30 and 50 lines of log entries. The entry shown here is just a tiny part of what was logged with the *POST*.

mod_security, also, permits the logging of request data. In the *mod_security* configuration shown in the recipe above, a log file is created containing all available request headers, and the request body itself.

See Also

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

3.4 Logging a Proxied Client's IP Address

Problem

You want to log the IP address of the actual client requesting your pages, even if they're being requested through a proxy.

Solution

None.

Discussion

Unfortunately, the HTTP protocol itself prevents this from being possible. From the client side, proxies are intended to be completely transparent; from the side of the origin server, where the content actually resides, they are meant to be almost utterly opaque, concealing the identity of a request.

Your best option is to log the IP address from which the request came. If it came directly from a browser, it will be the client's address; if it came through one or more proxy servers, it will be the address of the one that actually contacts your server.

Both the `combined` and `common` log formats include the `%h` format effector, which represents the (remote) client's identity. However, this may be a hostname rather than an address, depending on the setting of your *HostNameLookups* directive, among other things. If you always want the client's IP address to be included in your logfile, use the %a effector instead.

See Also

- The HTTP protocol specification at *ftp://ftp.isi.edu/in-notes/rfc2616.txt*

3.5 Logging Client MAC Addresses

Problem

You want to record the MAC (hardware) address of clients that access your server.

Solution

This cannot be logged reliably in most network situations and not by Apache at all.

Discussion

The MAC address is not meaningful except on local area networks (LANs) and is not available in wide area network transactions. When a network packet goes through a router, such as when leaving a LAN, the router will typically rewrite the MAC address field with the router's hardware address.

See Also

- The TCP/IP protocol specifications (see *http://www.rfc-editor.org/cgi-bin/rfcsearch.pl* and search for "TCP" in the title field)

3.6 Logging Cookies

Problem

You want to record all the cookies sent to your server by clients and all the cookies your server asks clients to set in their databases; this can be useful when debugging Web applications that use cookies.

Solution

To log cookies received from the client:

```
CustomLog logs/cookies_in.log "%{UNIQUE_ID}e %{Cookie}i"
CustomLog logs/cookies2_in.log "%{UNIQUE_ID}e %{Cookie2}i"
```

To log cookie values set and sent by the server to the client:

```
CustomLog logs/cookies_out.log "%{UNIQUE_ID}e %{Set-Cookie}o"
CustomLog logs/cookies2_out.log "%{UNIQUE_ID}e %{Set-Cookie2}o"
```

In versions before to 2.0.56, using the %{Set-Cookie}o format effector for debugging is not recommended if multiple cookies are (or may be) involved. Only the first one will be recorded in the logfile. See the Discussion text for an example.

Discussion

Cookie fields tend to be very long and complex, so the previous statements will create separate files for logging them. The cookie log entries can be correlated against the client request access log using the server-set UNIQUE_ID environment variable (assuming that *mod_unique_id* is active in the server and that the activity log format includes the environment variable with a %{UNIQUE_ID}e format effector).

At the time of this writing, the Cookie and Set-Cookie header fields are most commonly used. The Cookie2 and corresponding Set-Cookie2 fields are newer and have been designed to correct some of the shortcomings in the original specifications, but they haven't yet achieved much penetration.

Because of the manner in which the syntax of the cookie header fields has changed over time, these logging instructions may or may not capture the complete details of the cookies.

Bear in mind that these logging directives will record all cookies, and not just the ones in which you may be particularly interested. For example, here is the log entry for a client request that included two cookies, one named RFC2109-1 and one named RFC2109-2:

```
PNCSUsCoF2UAACI3CZs RFC2109-1="This is an old-style cookie, with space characters
     embedded"; RFC2109-2=This_is_a_normal_old-style_cookie
```

Even though there's only one log entry, it contains information about two cookies.

On the cookie-setting side, here are the Set-Cookie header fields sent by the server in its response header:

```
Set-Cookie: RFC2109-1="This is an old-style cookie, with space characters embedded";
     Version=1; Path=/; Max-Age=60; Comment="RFC2109 demonstration cookie"
Set-Cookie: RFC2109-2=This_is_a_normal_old-style_cookie; Version=1; Path=/; Max-
     Age=60; Comment="RFC2109 demonstration cookie"
```

And here's the corresponding log entry for the response (this was all one line in the log file, so line wrapping was added to make it all fit on the page):

```
eCF1vsCoF2UAAHB1DMIAAAAA RFC2109-1=\"This is an old-style cookie, with space characters embedded\";
    Version=1; Path=/; Max-Age=60; Comment=\"RFC2109 demonstration cookie\",
    RFC2109-2=This_is_a_normal_old-style_cookie; Version=1; Path=/; Max-Age=60; Comment=\"RFC2109 demons
```

Before version 2.0.56, Apache *httpd* didn't log multiple cookies correctly; it would only log one.

See Also

- RFC 2109, "*HTTP State Management Mechanism*" (IETF definition of `Cookie` and `Set-Cookie` header fields) at *ftp://ftp.isi.edu/in-notes/rfc2109.txt*
- RFC 2965, "*HTTP State Management Mechanism*" (IETF definition of `Cookie2` and `Set-Cookie2` header fields) at *ftp://ftp.isi.edu/in-notes/rfc2965.txt*
- The original Netscape cookie proposal at *http://home.netscape.com/newsref/std/cookie_spec.html*

3.7 Not Logging Image Requests from Local Pages

Problem

You want to log requests for images on your site, except when they're requests from one of your own pages. You might want to do this to keep your logfile size down, or possibly to track down sites that are hijacking your artwork and using it to adorn their pages.

Solution

Use *SetEnvIfNoCase* to restrict logging to only those requests from outside of your site:

```
<FilesMatch \.(jpg|gif|png)$>
    SetEnvIfNoCase Referer "^http://www.example.com/" local_referrer=1
</FilesMatch>
CustomLog logs/access_log combined env=!local_referrer
```

Discussion

In many cases, documents on a Web server include references to images also kept on the server, but the only item of real interest for log analysis is the referencing page itself. How can you keep the server from logging all the requests for the images that happen when such a local page is accessed?

The *SetEnvIfNoCase* will set an environment variable if the page that linked to the image is from the *www.example.com* site (obviously you should replace that site name with your own) and the request is for a GIF, PNG, or JPEG image.

SetEnvIfNoCase is the same as *SetEnvIf* except that variable comparisons are done in a case-insensitive manner.

The *CustomLog* directive will log all requests that do not have that environment variable set, *i.e.*, everything except requests for images that come from links on your own pages.

This recipe only works for clients that actually report the referring page. Some people regard the URL of the referring page to be no business of anyone but themselves, and some clients permit the user to select whether to include this information or not. There are also “anonymizing” sites on the Internet that act as proxies and conceal this information.

See Also

- Recipe 6.5

3.8 Rotating logfiles at a particular time

Problem

You want to automatically roll over the Apache logs at specific times without having to shut down and restart the server.

Solution

Use *CustomLog* and the *rotatelogs* program:

```
CustomLog "| /path/to/rotatelogs /path/to/logs/access_log.%Y-%m-%d 86400" combined
```

Discussion

The *rotatelogs* script is designed to use an Apache feature called *piped logging*, which is just a fancy name for sending log output to another program rather than to a file. By inserting the *rotatelogs* script between the Web server and the actual logfiles on disk, you can avoid having to restart the server to create new files; the script automatically opens a new file at the designated time and starts writing to it.

The first argument to the *rotatelogs* script is the base name of the file to which records should be logged. If it contains one or more `%` characters, it will be treated as a `strftime(3)` format string; otherwise, the rollover time (in seconds since 1 January 1970), in the form of a 10-digit number, will be appended to the base name. For example, a base name of `foo` would result in logfile names like *foo.1020297600*, whereas a base name of `foo.%Y-%m-%d` would cause the logfiles to be named something like *foo.2002-04-29*.

The second argument is the interval (in seconds) between rollovers. Rollovers will occur whenever the system time is a multiple of this value. For instance, a 24-hour day contains 86,400 seconds; if you specify a rollover interval of 86400, a new logfile will be created every night at midnight—when the system time, which is based at representing midnight on 1 January 1970, is a multiple of 24 hours.

Note that the rollover interval is in actual clock seconds elapsed, so when time changes because of daylight savings, this does not in any way affect the interval between rollovers.

See Also

- The *rotatelogs* manpage; try

  ```
  % man -M /path/to/ServerRoot/man rotatelogs.8
  ```

  replacing the `/path/to/ServerRoot` with the actual value of your installation's *ServerRoot* directive in *httpd.conf* or view the documentation online at `http://httpd.apache.org/docs/2.2/programs/rotatelogs.html`

3.9 Rotating Logs on the First of the Month

Problem

You want to close the previous month's logs and open new ones on the first of each month.

Solution

The Apache distribution doesn't come with a script that does this, but there is a free program that provides this, and many other useful features. It is called Cronolog, and may be obtained from `http://cronolog.org/`.

Obtain and install Cronolog, and then place the following in your configuration file:

```
CustomLog "|/usr/bin/cronolog /www/logs/access%Y%m.log" combined
```

Discussion

Cronolog has been around for a long time, and provides many of the features that people wished were available in the standard *rotatelogs* utility. Over the years, *rotatelogs* has improved, but Cronolog has a number of other useful features that are of interest to sites with rapidly growing logfiles.

One of these is the ability to automatically rotate logfiles by day, week, month, or year, based on the format of the filename specified in the *CustomLog* directive.

In the example given, the logfile is rotated at the start of a new month, because the logfile name given contains only the year and month variables (`%Y` and `%m`, respectively).

See Also

- *http://httpd.apache.org/docs/logs.html#piped*

- *http://httpd.apache.org/docs/2.2/programs/rotatelogs.html*
- *http://cronolog.org/*

3.10 Logging Hostnames Instead of IP Addresses

Problem

You want to see hostnames in your activity log instead of IP addresses.

Solution

You can let the Web server resolve the hostname when it processes the request by enabling runtime lookups with the Apache directive:

```
HostnameLookups On
```

Or you can let Apache use the IP address during normal processing and let a piped logging process resolve them as part of recording the entry:

```
HostnameLookups Off
CustomLog "| /path/to/logresolve -c >> /path/to/logs/access_log.resolved" combined
```

Or you can let Apache use and log the IP addresses, and resolve them later when analyzing the logfile. Add this to *http.conf*:

```
CustomLog /path/to/logs/access_log.raw combined
```

And analyze the log with:

```
% /path/to/logresolve -c < access_log.raw > access_log.resolved
```

Discussion

The Apache activity logging mechanism can record either the client's IP address or its hostname (or both). Logging the hostname directly requires that the server spend some time to perform a DNS lookup to turn the IP address (which it already has) into a hostname. This can have some serious impact on the server's performance, however, because it needs to consult the name service in order to turn the address into a name; and while a server child or thread is busy waiting for that, it isn't handling client requests. One alternative is to have the server record only the client's IP address and resolve the address to a name during logfile postprocessing and analysis. At the very least, defer it to a separate process that won't directly tie up the Web server with the resolution overhead.

In theory this is an excellent choice; in practice, however, there are some pitfalls. For one thing, the *logresolve* application included with Apache (usually installed in the *bin/* subdirectory under the *ServerRoot*) will only resolve IP addresses that appear at the very beginning of the log entry, and so it's not very flexible if you want to use a nonstandard format for your logfile. For another, if too much time passes between the collection and resolution of the IP addresses, the DNS may have changed sufficiently so that misleading or incorrect results may be obtained. This is especially a problem with dynamically allocated IP addresses such as those issued by ISPs. Although, for these dynamically allocated IP addresses, the hostnames tend not to be particularly informative anyway.

An additional shortcoming becomes apparent if you feed your log records directly to *logresolve* through a pipe: as of Apache 1.3.24 at least, *logresolve* doesn't flush its output buffers immediately, so there's the possibility of lost data if the logging process or the system should crash.

In practice, however, all log analysis software provides hostname resolution functionality, and it generally makes most sense to use that functionality than trying to resolve the IP addresses in the logfile before that stage.

See Also

- The *logresolve* manpage:

  ```
  % man -M /path/to/ServerRoot/man/logresolve.8
  ```
- *http://httpd.apache.org/docs/2.2/programs/logresolve.html*

3.11 Maintaining Separate Logs for Each Virtual Host

Problem

You want to have separate activity logs for each of your virtual hosts, but you don't want to have all the open files that multiple *CustomLog* directives would use.

Solution

Use the *split-logfile* program that comes with Apache. To split logfiles after they've been rolled over (replace `/path/to/ServerRoot` with the correct path):

```
# cd /path/to/ServerRoot
# mv logs/access_log logs/access_log.old
# bin/apachectl graceful
[wait for old logfile to be completely closed]
# cd logs
# ../bin/split-logfile < access_log.old
```

To split records to the appropriate files as they're written, add this line to your *httpd.conf* file:

```
CustomLog "| /path/to/split-logfile /usr/local/Apache/logs" combined
```

Discussion

In order for *split-logfile* to work, the logging format you're using must begin with "% v" (note the blank after the v). This inserts the name of the virtual host at the beginning of each log entry; *split-logfile* will use this to figure out to which file the entry should be written. The hostname will be removed from the record before it gets written.

There are two ways to split your access logfile: after it's been written, closed, and rolled over, or as the entries are actually being recorded. To split a closed logfile, just feed it into the *split-logfile* script. To split the entries into separate files as they're actually being written, modify your configuration to pipe the log messages directly to the script.

Each method has advantages and disadvantages. The rollover method requires twice as much disk space (for the unsplit log plus the split ones) and that you verify that the logfile is completely closed. (Unfortunately there is no guaranteed, simple way of doing this without actually shutting down the server or doing a graceless restart; it's entirely possible that a slow connection may keep the old logfile open for a considerable amount of time after a graceful restart.) Splitting as the entries are recorded is sensitive to the logging process dying—although Apache will automatically restart it, log messages waiting for it can pile up and constipate the server.

See Also

- Recipe 3.10

3.12 Logging Proxy Requests

Problem

You want to log requests that go through your proxy to a different file than the requests coming directly to your server.

Solution

Use the *SetEnv* directive to earmark those requests that came through the proxy server, in order to trigger conditional logging:

```
<Directory proxy:*>
    SetEnv is_proxied 1
</Directory>
CustomLog logs/proxy_log combined env=is_proxied
```

Or, for 2.x, use a <Proxy> block:

```
<Proxy *>
    SetEnv is_proxied 1
    </Proxy>
CustomLog logs/proxy_log combined env=is_proxied
```

Discussion

Apache 1.3 has a special syntax for the *<Directory>* directive, which applies specifically to requests passing through the proxy module. Although the * makes it appear that wildcards can be used to match documents, it's misleading; it isn't really a wildcard. You may either match explicit paths, such as *proxy:http://example.com/foo.html*, or use * to match *everything*. You can not do something like *proxy:http://example.com/*.html*.

If you want to apply different directives to different proxied paths, you need to take advantage of another module. Because you're dealing with requests that are passing through your server rather than being handled by it directly (*i.e.*, your server is a *proxy* rather than an *origin server*), you can't use *<Files>* or *<FilesMatch>* containers to apply directives to particular proxied documents. Nor can you use *<Location>* or *<LocationMatch>* stanzas, because they can't appear inside a *<Directory>* container. You can, however, use *mod_rewrite*'s capabilities to make decisions based on the path of the requested document. For instance, you can log proxied requests for images in a separate file with something like this:

```
<Directory proxy:*>
    RewriteEngine On
    RewriteRule "\.(gif|png|jpg)$" "-" [ENV=proxied_image:1]
    RewriteCond "%{ENV:proxied_image}" "!1"
    RewriteRule "^" "-" [ENV=proxied_other:1]
</Directory>
CustomLog logs/proxy_image_log combined env=proxied_image
CustomLog logs/proxy_other_log combined env=proxied_other
```

Directives in the *<Directory proxy:*>* container will only apply to requests going through your server. The first *RewriteRule* directive sets an environment variable if the requested document ends in *.gif*, *.png*, or *.jpg*. The *RewriteCond* directive tests to see if that envariable isn't set, and the following *RewriteRule* will set a different envariable if so. The two *CustomLog* directives send the different types of requests to different logfiles according to the environment variables.

See Also

- The *mod_rewrite* and *mod_log_config* documentation

3.13 Logging Errors for Virtual Hosts to Multiple Files

Problem

Unlike access logs, Apache only logs errors to a single location. You want Apache to log errors that refer to a particular virtual host to the host's error log, as well as to the global error log.

Solution

There are at least two possible ways of doing this:

1. Use piped logging to send entries to a custom script that will copy and direct error messages to the appropriate files.
2. Use piped logging to duplicate log entries:

```
    ErrorLog "| tee logfile1 | tee logfile2 > logfile3"
```

Discussion

Unlike activity logs, Apache will log error messages only to a single location. If the error is related to a particular virtual host and this host's <*VirtualHost*> container includes an *ErrorLog* entry, the error will be logged only in this file, and it won't appear in any global error log. If the <*VirtualHost*> does not specify an *ErrorLog* directive, the error will be logged only to the global error log. (The global error log is the last *ErrorLog* directive encountered that isn't in a <*VirtualHost*> container.)

Currently, the only workaround to this is to have the necessary duplication performed by a separate process (i.e., by using piped logging to send the error messages to the process as they occur). Of the two solutions given above, the first, which involves a custom script you develop yourself, has the most flexibility. If all you want is simply duplication of entries, the second solution is simpler but requires that your platform have a *tee* program (Windows does not). It also may be subject to lagging messages if your *tee* program doesn't flush its buffers after each record it receives. This could also lead to lost messages if the pipe breaks or the system crashes.

An alternate approach may be to send the error log to syslog, and then have your syslog server log entries to multiple places.

See Also

- *http://httpd.apache.org/docs/logs.html#piped*

3.14 Logging Server IP Addresses

Problem

You want to log the IP address of the server that responds to a request, possibly because you have virtual hosts with multiple addresses each.

Solution

Use the `%A` format effector in a *LogFormat* or *CustomLog* directive:

```
CustomLog logs/served-by.log "%A"
```

Discussion

The %A effector signals the activity logging system to insert the local IP address—that is, the address of the server—into the log record at the specified point. This can be useful when your server handles multiple IP addresses. For example, you might have a configuration that includes elements such as the following:

```
Listen 10.0.0.42
Listen 192.168.19.243
Listen 263.41.0.80
<VirtualHost 192.168.19.243>
    ServerName Private.Example.Com
</VirtualHost>
<VirtualHost 10.0.0.42 263.41.0.80>
    ServerName Foo.Example.Com
    ServerAlias Bar.Example.Com
</VirtualHost>
```

This might be meaningful if you want internal users to access *Foo.Example.Com* using the 10.0.0.42 address rather than the one published to the rest of the network (such as to segregate internal from external traffic over the network cards). The second virtual host is going to receive requests aimed at both addresses even though it has only one *ServerName*; using the %A effector in your log format can help you determine how many hits on the site are coming in over each network interface.

See Also

- The *mod_log_config* documentation

3.15 Logging the Referring Page

Problem

You want to record the URL of pages that refer clients to yours, perhaps to find out how people are reaching your site.

Solution

Add the following effector to your activity log format:

```
%{Referer}i
```

Discussion

One of the fields that a request header may include is called the Referer. Referer is the URL of the page that linked to the current request. For example, if file *a.html* contains a link such as:

```
<a href="b.html">another page</a>
```

When the link is followed, the request header for *b.html* will contain a `Referer` field that has the URL of *a.html* as its value.

The *Referer* field is not required nor reliable; some users prefer software or anonymizing tools that ensure that you can't tell where they've been. However, this is usually a fairly small number and may be disregarded for most Web sites.

See Also

- Recipe 3.17
- Recipe 6.5

3.16 Logging the Name of the Browser Software

Problem

You want to know the software visitors use to access your site, for example, so you can optimize its appearance for the browser that most of your audience uses.

Solution

Add the following effector to your activity log format:

```
%{User-Agent}i
```

Discussion

Request headers often include a field called the `User-agent`. This is defined as the name and version of the client software being used to make the request. For instance, a `User-agent` field value might look like this:

```
User-Agent: Mozilla/4.77 [en] (X11; U; Linux 2.4.4-4GB i686)
```

This tells you that the client is claiming to be Netscape Navigator 4.77, run on a Linux system and using X-windows as its GUI.

The `User-agent` field is neither required nor reliable; many users prefer software or anonymizing tools that ensure that you can't tell what they're using. Some software even lies about itself so it can work around sites that cater specifically to one browser or another; users have this peculiar habit of thinking it's none of the Webmaster's business which browser they prefer. It's a good idea to design your site to be as browser-agnostic as possible for this reason, among others. If you're going to make decisions based on the value of the field, you might as well believe it hasn't been faked—because there's no way to tell if it has.

See Also

- Recipe 3.17

3.17 Logging Arbitrary Request Header Fields

Problem

You want to record the values of arbitrary fields clients send to their request header, perhaps to tune the types of content you have available to the needs of your visitors.

Solution

Use the `%{...}i` log format variable in your access log format declaration. For example, to log the Host header, you might use:

```
%{Host}i
```

Discussion

The HTTP request sent by a Web browser can be very complex, and if the client is a specialized application rather than a browser, it may insert additional metadata that's meaningful to the server. For instance, one useful request header field is the `Accept` field, which tells the server what kinds of content the client is capable of and willing to receive. Given a *CustomLog* line such as this:

```
CustomLog logs/accept_log "\"%{Accept}i\""
```

a resulting log entry might look like this:

```
PNb6VsCoF2UAAH1dAUo "text/html, image/png, image/jpeg, image/gif,
     image/x-xbitmap, */*"
```

This tells you that the client that made that request is explicitly ready to handle HTML pages and certain types of images, but, in a pinch, will take whatever the server gives it (indicated by the wildcard */* entry).

See Also

- Recipe 3.15
- Recipe 3.17

3.18 Logging Arbitrary Response Header Fields

Problem

You want to record the values of arbitrary fields the server has included in a response header, probably to debug a script or application.

Solution

Use the `%{...}o` log format variable in your access log format declaration. For example, to log the Last-Modified header, you would do the following:

```
%{Last-Modified}o
```

Discussion

The HTTP response sent by Apache when answering a request can be very complex, according to the server's configuration. Advanced scripts or application servers may add custom fields to the server's response, and knowing what values were set may be of great help when trying to track down an application problem.

Other than the fact that you're recording fields the server is *sending* rather than receiving, this recipe is analogous to Recipe 3.17 in this chapter; refer to that recipe for more details. The only difference in the syntax of the logging format effectors is that response fields are logged using an o effector, and request fields are logged using `i`.

See Also

- Recipe 3.17

3.19 Logging activity to a MySQL database

Problem

Rather than logging accesses to your server in flat text files, you want to log the information directly to a database for easier analysis.

Solution

Install the latest release of *mod_log_sql* from *http://www.outoforder.cc/projects/apache/mod_log_sql/* according to the modules directions (see Recipe 2.1), and then issue the following commands:

```
# mysqladmin create apache_log
# mysql apache_log < access_log.sql
# mysql apache_log
mysql> grant insert,create on apache_log.* to webserver@localhost identified by 'wwwpw';
```

Add the following lines to your *httpd.conf* file:

```
<IfModule mod_log_sql.c>
LogSQLLoginInfo mysql://webserver:wwwpw@dbmachine.example.com/apache_log
LogSQLCreateTables on
</IfModule>
```

Then, in your *VirtualHost* container, add the following log directive:

```
LogSQLTransferLogTable access_log
```

Discussion

Replace the values of `webserver` and `wwwpw` with a less guessable username and password when you run these commands.

Consult the documentation on the referenced website to ensure that the example here reflects the version of the module that you have installed, as the configuration syntax changed with the 2.0 release of the module.

See Also

- *http://www.outoforder.cc/projects/apache/mod_log_sql/*

3.20 Logging to syslog

Problem

You want to send your log entries to syslog.

Solution

To log your error log to syslog, simply tell Apache to log to *syslog*:

```
ErrorLog syslog:user
```

Some other *syslog* reporting class than `user`, such as `local1` might be more appropriate in your environment.

Logging your access log to syslog takes a little more work. Add the following to your configuration file:

```
CustomLog |/usr/local/apache/bin/apache_syslog combined
```

Where *apache_syslog* is a program that looks like the following:

```
#!/usr/bin/perl
use Sys::Syslog qw( :DEFAULT setlogsock );

setlogsock('unix');
openlog('apache', 'cons', 'pid', 'user');

while ($log = <STDIN>) {
        syslog('notice', $log);
}
```

```
    closelog;
```

Discussion

There are several compelling reasons for logging to syslog. The first of these is to have many servers log to a central logging facility. The second is that there are many existing tools for monitoring syslog and sending appropriate notifications on certain events. Allow Apache to take advantage of these tools, and your particular installation may benefit. Also, in the event that your server is either compromised, or has some kind of catastrophic failure, having log files on a dfferent physical machine can be of enormous benefit in finding out what happened.

Apache supports logging your error log to syslog by default. This is by far the more useful log to handle this way, since syslog is typically used to track error conditions, rather than merely informational messages.

The syntax of the *ErrorLog* directive allows you to specify `syslog` as an argument, or to specify a particular syslog facility. In the example above, the `user` syslog facility was specified. In your */etc/syslog.conf* file, you can specify where a particular log facility should be sent—whether to a file, or to a remote syslog server.

Because Apache does not support logging your access log to syslog by default, you need to accomplish this with a piped log file directive. The program that we use to accomplish this is a simple Perl program using the *Sys::Syslog* module, which is a standard module with your Perl installation. Because the piped log file handler is launched at server startup, and merely accepts input on STDIN for the life of the server, there is no performance penalty for using Perl.

If you have several Web servers, and want to have all of them log to one central log file, this can be accomplished by having all of your servers log to syslog, and pointing that syslog facility to a central syslog server. Note that this may cause your log entries to be in non-sequential order, which should not really matter, but may appear strange at first. This effect can be reduced by ensuring that your clocks are synchronized *via* NTP.

Consult your *syslogd* manual for further detail on setting up a networked syslog server.

Finally, depending on what particular operating system you are using, you may be able to use the *logger* utility to accomplish the same thing:

```
AccessLog "|/usr/bin/logger" combined
```

See Also

- The *man* pages for *syslogd* and *syslog.conf*

3.21 Logging user directories

Problem

You want each user directory Web site (i.e., those accessed *via* http://server/~username (*http://servername/~username*)) to have its own log file.

Solution

In *httpd.conf*, add the directive:

```
CustomLog "|/usr/local/apache/bin/userdir_log" combined
```

Then, in the file */usr/local/apache/bin/userdir_log*, place the following code:

```
#!/usr/bin/perl

my $L = '/usr/local/apache/logs'; # Log directory

my %is_open = (); # File handle cache
$|=1;
open(F, ">>$L/access_log"); # Default error log

while (my $log = <STDIN>) {
    if ($log =~ m!\s/~(.*?)/!) {
        my $u = $1;
        unless ($is_open{$u}) {
            my $fh;
            open $fh, '>>' . $L . '/'. $u;
            $is_open{$u} = $fh;
        }
        select ($is_open{$u});
        $|=1;
        print $log;
    }
    else {
        select F;
        $|=1;
        print F $log;
    }
}

close F;
foreach my $h (keys %is_open) {
    close $h;
}
```

Discussion

Usually, requests to user directory Web sites are logged in the main server log, with no differentiation between one user's site and another. This can make it very hard for a user to locate log messages for their personal Web site.

The recipe above allows you to break out those requests into one log file *per* user, with requests not going to a userdir Web site going to the main log file. The log handler can, of course, be modified to put all log messages in the main log file as well as in the individual log files.

In order to lessen the amount of disk activity necessary, file handles are cached, rather than opened and closed with each access. This results in a larger number of file handles which are open at any given time. For sites with a very large number of user Web sites, this may cause you to run out of system resources.

Because Perl buffers output by default, we need to explicitly tell our script not to buffer the output, so that log entries make it into the log file immediately. This is accomplished by setting the autoflush variable, `$|`, to a true value. This tells Perl not to buffer output to the most-recently selected file handle. Without this precaution, output will be buffered, and it will appear that nothing is being written to your log files.

An alternate approach might involve setting an environment variable using *mod_rewrite* and then adding that variable to your *LogFormat* directive:

```
RewriteRule ^/~([^/]+)/ - [E=userdir:$1]
LogFormat "%{userdir}e %h %l %u %t \"%r\" %>s %b" common
```

Having done this, you could then use the *split-logfile* script to split the logfile up into one file per individual user.

See Also

- *http://httpd.apache.org/docs/2.0/mod/mod_log_config.html*
- *http://httpd.apache.org/docs/2.2/mod/mod_log_config.html*
- *http://httpd.apache.org/docs/2.2/programs/other.html*