Logging and Monitoring Web Server Activity

Lesson 13b

This lesson describes how the logging system in Apache works and how you can customize it which information to store and where to do it. Additionally, you will learn a quick way to use PHP and MySQL to log specific items of interest to you, outside the realm of the Apache log files.

In this lesson, you will learn how to

- Understand Apache log formats and logging levels
- Rotate and analyze Apache logs
- Interpret common errors that might appear in your logs
- Create scripts that log specific items to database tables
- Create custom reports based on these logging tables

Standard Apache Access Logging

Using Apache's basic logging features, you can keep track of who visits your Web sites by logging accesses to the servers hosting them. You can log every aspect of the browser requests and server responses, including the IP address of the client, user, and resource accessed. You need to take three steps to create a request log:

1. Define what you want to log your log format.
2. Define where you want to log it your log files, a database, an external program.
3. Define whether or not to log conditional logging rules.

The next few sections will take a closer look at these steps.

Deciding What to Log

As well as logging nearly every aspect associated with the request, you can define how your log entries appear by creating a log format. A log format is a string that contains text mixed with log formatting directives. Log formatting directives start with a % and are followed by a directive name or identifier, usually a letter indicating the piece of information to be logged.

When Apache logs a request, it scans the string and substitutes the value for each directive. For example, if the log format is `This is the client address %a`, the log entry is something like `This is the client address 10.0.0.2`. That is, the logging directive %a is replaced by the IP address of the client making the request. Table 1 provides a comprehensive list of all formatting directives.

Table 1. Log Formatting Directives

Formatting Options	Explanation
Data from the Client	
`%a`	Remote IP address, from the client.
`%h`	Hostname or IP address of the client making the request. Whether or not the hostname is logged depends on two factors: The IP address of the client must resolve to a hostname via a reverse DNS lookup, and Apache must be configured to do that lookup using the HostNameLookups directive, explained later in this lesson. If these conditions are not met, the IP address of the client will be logged instead of the hostname.
`%l`	Remote user, obtained via the `identd` protocol. This option is

	not very useful because this protocol is not supported on the majority of the client machines.
`%u`	Remote user, from the HTTP basic authentication protocol.
Data from the Server	
`%A`	Local IP address, from the server.
`%D`	Time it took to serve the request, in microseconds.
`%{env_variable}e`	Value for an environment variable named `env_variable` (`there are many`).
`%{time_format}t`	Current time. If `{time_format}` is present, it will be interpreted as an argument to the Unix `strftime` function. See the `logresolve` Apache manual page for details.
`%T`	Time it took to serve the request, in seconds.
`%v`	Canonical name of the server that answered the request.
`%V`	Server name according to the `UseCanonicalName` directive.
`%X`	Status of the connection to the server. A value of `x` means the connection was aborted before the server could send the data. A + means the connection will be kept alive for further requests from the same client. A - means the connection will be closed.
Data from the Request	
`%{cookie_name}C`	Value for a cookie named `cookie_name`.
`%H`	Request protocol, such as HTTP or HTTPS.
`%m`	Request method such as `GET`, `POST`, `PUT`, and so on.
`%{header_name}i`	Value for a header named `header_name` in the request from the client. This information can be useful, for example, to log the names and versions of your visitors' browsers.
`%r`	Text of the original HTTP request.
`%q`	Query parameters, if any, prefixed by a ?.
`%U`	Requested URL, without query parameters.
`%y`	Username for the HTTP authentication (basic or digest).
Data from the Response	
`%b, %B`	Size, in bytes, of the body of the response sent back to the client (excluding headers). The only difference between the options is that if no data was sent, `%b` will log a - and `%B` will log `0`.

`%f`	Path of the file served, if any.
`%t`	Time when the request was served.
`%{header_name}o`	Value for a header named `header_name` in the response to the client.
`%>s`	Final status code. Apache can process several times the same request (internal redirects). This is the status code of the final response.

The Common Log Format (CLF) is a standard log format. Most Web sites can log requests using this format, and the format is understood by many log processing and reporting tools. Its format is the following:

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

That is, it includes the hostname or IP address of the client, remote user via `identd`, remote user via HTTP authentication, time when the request was served, text of the request, status code, and size in bytes of the content served.

NOTE:

You can read the Common Log Format documentation of the original W3C server at http://www.w3.org/Daemon/User/Config/Logging.html.

The following is a sample CLF entry:

```
10.0.0.1 - - [26/Aug/2004:11:27:56 -0800] "GET / HTTP/1.1" 200 1456
```

You are now ready to learn how to define log formats using the `LogFormat` directive. This directive takes two arguments: The first argument is a logging string, and the second is a nickname that will be associated with that logging string.

For example, the following directive from the default Apache configuration file defines the CLF and assigns it the nickname `common`:

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

You can also use the `LogFormat` directive with only one argument, either a log format string or a nickname. This will have the effect of setting the default value for the logging format used by the `TRansferLog` directive, explained in "Logging Accesses to Files" later in this lesson.

The `HostNameLookups` Directive

When a client makes a request, Apache knows only the IP address of the client. Apache must perform what is called a reverse DNS lookup to find out the hostname associated with the IP address. This operation can be time-consuming and can introduce a noticeable lag in the request processing. The `HostNameLookups` directive allows you to control whether to perform the reverse DNS lookup.

The `HostNameLookups` directive can take one of the following arguments: `on`, `off`, or `double`. The default is `off`. The `double` lookup argument means that Apache will find out the hostname from the IP and then will try to find the IP from the hostname. This process is necessary if you are really concerned with security, as described in http://httpd.apache.org/docs/current/mod/core.html#hostnamelookups. If you are using hostnames as part of your `Allow` and Deny rules, a double DNS lookup is performed regardless of the `HostNameLookups` settings.

If `HostNameLookups` is enabled (`on` or `double`), Apache will log the hostname. This does cause extra load on your server, which you should be aware of when making the decision to turn `HostNameLookups` on or off. If you choose to keep `HostNameLookups` off, which would be recommended for medium-to-high traffic sites, Apache will log only the associated IP address. There are plenty of tools to resolve the IP addresses in the logs later. Refer to the "Managing Apache Logs" section later in this lesson. Additionally, the result will be passed to CGI scripts via the environment variable `REMOTE_HOST`.

The `IdentityCheck` Directive

At the beginning of the lesson, we explained how to log the remote username via the `identd` protocol using the `%l` log formatting directive. The `IdentityCheck` directive takes a value of on or `off` to enable or disable checking for that value and making it available for inclusion in the logs. Because the information is not reliable and takes a long time to check, it is switched off by default and should probably never be enabled. We mentioned %l only because it is part of the CLF. For more information on the `identd` protocol, see RFC 1413 at https://www.ietf.org/rfc/rfc1413.txt.

Status Code

You can specify whether to log specific elements in a log entry. At the beginning of the lesson, you learned that log directives start with a %, followed by a directive identifier. In between, you can insert a list of status codes, separated by commas. If the request status is one of the listed codes, the parameter will be logged; otherwise, a - will be logged.

For example, the following directive identifier logs the browser name and version for malformed requests (status code 400), and requests with methods not implemented (status code 501). This information can be useful for tracking which clients are causing problems.

```
%400,501{User-agent}i
```

You can precede the method list with an ! to log the parameter if the methods are implemented:

```
%!400,501{User-agent}i
```

Logging Accesses to Files

Logging to files is the default way of logging requests in Apache. You can define the name of the file using the `TRansferLog` and `CustomLog` directives.

The `transferLog` directive takes a file argument and uses the latest log format defined by a `LogFormat` directive with a single argument (the nickname or the format string). If no log format is present, it defaults to the CLF.

The following example shows how to use the `LogFormat` and `transferLog` directives to define a log format that is based on the CLF but that also includes the browser name:

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{User-agent}i\""
TransferLog logs/access_log
```

The `CustomLog` directive enables you to specify the logging format explicitly. It takes at least two arguments: a logging format and a destination file. The logging format can be specified as a nickname or as a logging string directly.

For example, the directives

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{User-agent}i\"" myformat
CustomLog logs/access_log myformat
```

and

```
CustomLog logs/access_log "%h %l %u %t \"%r\" %>s %b \"%{User-
agent}i\""
```

are equivalent.

Logging Environment Variables with `CustomLog`

The `CustomLog` directive accepts an environment variable as a third argument. If the environment variable is present, the entry will be logged; otherwise, it will not. If the environment variable is negated by prefixing an ! to it, the entry will be logged if the variable is not present.

The following example shows how to avoid logging images in GIF and JPEG format in your logs:

```
SetEnvIf Request_URI "(\.gif|\.jpg)$" image
CustomLog logs/access_log common env=!image
```

NOTE:

The regular expression used for pattern matching in this and other areas of the `httpd.conf` file follow the same format for regular expressions in PHP and other programming languages.

Logging Accesses to a Program

Both `transferLog` and `CustomLog` directives can accept an executable program, prefixed by a pipe sign |, as an argument. Apache will write the log entries to the standard input of this program. The program will, in turn, process the input by logging the entries to a database, transmitting them to another system, and so on.

If the program dies for some reason, the server makes sure that it is restarted. If the server stops, the program is stopped as well. The `rotatelogs` utility, bundled with Apache and explained later in this lesson, is an example of a logging program.

As a general rule, unless you have a specific requirement for using a particular program, it is easier and more reliable to log to a file on disk and do the processing, merging, analysis of logs, and so on, at a later time, possibly on a different machine.

NOTE:

Make sure that the program you use for logging requests is secure because it runs as the user Apache was started with. On Unix, this usually means root because the external program will be started before the server changes its user ID to the value of the User directive, typically nobody or `www`.

Standard Apache Error Logging

Apache can be configured to log error messages and debugging information, in addition to client requests. In addition to errors generated by Apache itself, CGI errors can be logged.

Each error log entry is prefixed by the time the error occurred and the client IP address or hostname, if available. As with HTTP request logging, you can log error information to a file or program. On Unix systems, you can also log to the `syslog` daemon. On Windows, errors can be logged in the Windows event log and would then be viewable via the Windows Event Viewer. Use the `ErrorLog` directive to define where you want your logs to go.

Logging Errors to a File

A file argument indicates the path to the error log file. If the path is relative, it is assumed to be relative to the server root. By default, the error log file will be located in the `logs` directory and will be named `error_log` on Unix and `error.log` on Windows. The following is an example:

```
ErrorLog logs/my_error_log
```

Logging Errors to a Program

You can specify the path to a program, prefixed by a pipe |. Apache will log errors to the standard input of the program, and the program will further process them. The following is an example:

```
ErrorLog "|/usr/local/bin/someprogram"
```

The `syslog` Daemon Argument

On a Unix system, if you specify `syslog` as an argument, you can log error messages to the Unix system log daemon `syslogd`. By default, log errors are logged to the `syslog` facility `local7`. The facility is the part of the system generating the error. You can specify a facility by providing `syslog: facility` as an argument. Examples of syslog facilities are `mail, uucp, local0, local1`, and so on. For a complete list, look at the documentation for `syslog` included with your system (try `man syslogd` or `man syslogd.conf` at the command line). The following is an example of logging to `syslog`:

```
ErrorLog syslog:local6
```

The `LogLevel` Directive

The error information provided by Apache has several degrees of importance. You can choose to log only important messages and disregard informational or trivial warning messages. The `LogLevel` directive takes an error-level argument. Only errors of that level of importance or higher will be logged.

Table 2 specifies the valid values for the `LogLevel` directive, as specified by the Apache documentation. By default, the `LogLevel` value is `warn`. That should be enough for most Apache installations. If you are trying to troubleshoot a specific configuration, you can alter the level to `debug`.

Table 2. `LogLevel` Options as Described in the Apache Documentation

Setting	Description	Example
`emerg`	Emergenciessystem is unusable	`Child cannot open lock file. Exiting.`
`alert`	Action must be taken immediately	`getpwuid: couldn't determine user name from uid.`
`crit`	Critical conditions	`socket: Failed to get a socket, exiting child.`
`error`	Error conditions	`Premature end of script headers.`
`warn`	Warning conditions	`Child process 1234 did not exit, sending another SIGHUP.`
`notice`	Normal but significant conditions	`httpd: caught SIGBUS, attempting to dump core in...`
`info`	Informational	`Server seems busy, (You may need to increase StartServers, or Min/MaxSpareServers)...`
`debug`	Debug-level messages	`Opening config file...`

Managing Apache Logs

Apache provides several tools for managing your logs. Other Apache-specific third-party tools are available and are mentioned here. Because Apache can log requests in the CLF, most generic log processing tools can be used with Apache as well.

Resolving Hostnames

Earlier in the lesson, you learned how to use the `HostNameLookups` directive to enable or disable hostname resolution at the time the request is made. If `HostNameLookups is set to off` (the default), the log file will contain only IP `addresses`. Later, you can use the command-line `logresolve` utility on Unix or `logresolve.exe` on Windows to process the log file and convert the IP addresses to hostnames.

The `logresolve` utility reads log entries from standard input and outputs the result to its standard output. To read to and from a file, you can use redirection, on both Unix and Windows:

```
logresolve < access.log > resolved.log
```

Log-resolving tools are efficient because they can cache results and they do not cause any delay when serving requests to clients.

Log Rotation

In Web sites with high traffic, access log files can quickly grow in size. You should have a mechanism to rotate logs periodically, archiving and compressing older logs at defined intervals.

Log files should not be removed while Apache is running, because the server is writing directly to them. A solution would be to use an intermediate program to log the requests. The program will, in turn, take care of rotating the logs.

Apache provides the `rotatelogs` program on Unix and `rotatelogs.exe` on Windows for this purpose. It accepts three arguments: a filename, a rotate interval in seconds, and an optional offset in minutes against UTC (Coordinated Universal Time).

For example,

```
TransferLog "|bin/rotatelogs /var/logs/apachelog 86400"
```

will create a new log file and move the current log to the `/var/logs` directory daily. (At the end of the command, 86400 is the number of seconds in one day.)

NOTE:

If the path to the program includes spaces, you might need to escape them by prefixing them with a \ (backslash)for example, `My\ Documents`. This is especially common in the Windows platform.

If the name of the file includes `%` prefixed options, the name will be treated as input to the `strftime` function that converts the % options to time values. The manual page for the `rotatelogs` utility contains a complete listing of options, but here's an example:

```
TransferLog "|bin/rotatelogs /var/logs/apachelog%m_%d_%y 86400"
```

This command will add the current month, day, and year to the log filename.

If the name does not include any %-formatted options, the current time in seconds is added to the name of the archived file.

Merging and Splitting Logs

When you have a cluster of Web servers serving similar content, perhaps behind a load balancer, you often need to merge the logs from all the servers in a unique log stream before passing it to analysis tools.

Similarly, if a single Apache server instance handles several virtual hosts, sometimes it is useful to split a single log file into different files, one per each virtual host.

Log tools is a collection of log-manipulation tools that can be found at http://www.coker.com.au/logtools/. Additionally, Apache includes the `split-file` Perl script for splitting logs. You can find it in the `support` subdirectory of the Apache distribution.

Log Analysis

After you collect the logs, you can analyse them and gain information about traffic and visitor behaviourr.

Many commercial and freely available applications are available for log analysis and reporting. Two popular open source applications are Webalizer (http://www.mrunix.net/webalizer/) and awstats (http://awstats.sourceforge.net/).

W usage is a nice, inexpensive commercial alternative and can be found at http://www.boutell.com/wusage/.

Monitoring Error Logs

If you run Apache on a Unix system, you can use the `tail` command-line utility to monitor, in real-time, log entries both to your access and error logs. The syntax is

```
tail -f logname
```

where `logname` is the path to the Apache log file. It will print onscreen the last few lines of the log file and will continue to print entries as they are added to the file.

You can find additional programs that enable you to quickly identify problems by scanning your error log files for specific errors, malformed requests, and so on, and reporting on them:

- Logscan can be found at http://sourceforge.net/projects/logscan/.
- ScanErrLog can be found at http://www.librelogiciel.com/software/.

Logging Custom Information to a Database

Creating your own logging tables in MySQL, matched up with snippets of PHP code, can help you to capture access-related information for specific pages of your site. Using this information, you can create customized reports. This method can be much less cumbersome than wading through Apache log files, especially when you are just searching for a subset of access information. The following sections outline a simple version of this process.

Creating the Database Table

The first step in your custom logging method is to create the database table. The following table creation command will create a table called `access_tracker` in your MySQL database, with fields for an ID, page title, user agent, and date of access:

```
mysql> create table access_tracker (
    -> id int not null primary key auto_increment,
    -> page_title varchar(50),
    -> user_agent text,
    -> date_accessed date
    -> );
```

Next, you'll create the code snippet that will write to this table.

Creating the PHP Code Snippet

As you may have gathered already, code snippet essentially means a little bit of code. In other words, something that doesn't qualify as a long script, but just serves a simple purpose. In this case, the code snippet in Listing 1 will write some basic information to the `access_tracker` table.

Listing 1. Code Snippet for Access Tracking

```
 1: <?
 2: //set up static variables
 3: $page_title = "sample page A";
 4: $user_agent = getenv("HTTP_USER_AGENT");
 5:
 6: //connect to server and select database
 7: $conn = mysql_connect("localhost", "joeuser", "somepass") or
die(mysql_error());
 8: $db = mysql_select_db("testDB", $conn) or die(mysql_error());
 9:
10: //create and issue query
11: $sql = "insert into access_tracker values
12:    ('', '$page_title', '$user_agent', now())";
13: mysql_query($sql,$conn);
14: ?>
```

What you'll do with this snippet is simple: Place it at the beginning of every page you want to track. For each page, change the value of `$page_title` in the snippet to represent the actual title of the page.

Now create a sample script called `sample1.php`, containing the contents of Listing 1 and then the content in Listing 2.

Listing 2. Sample HTML Page

```
1: <HTML>
2: <HEAD>
3: <TITLE>Sample Page A</TITLE> 4: </HEAD>
5: <BODY>
6: <h1>Sample Page A</h1>
7: <P>Blah blah blah.</p>
8: </BODY>
9: </HTML>
```

Create a few copies of this file, with different filenames and values for `$page_title`. Then access these different pages with your Web browser to fill up your logging table.

Creating Sample Reports

When you have the data in your `access_tracker` table, you can create a simple report screen to disseminate this information. The code in Listing 3 creates a report that issues queries to count total results as well as the breakdown of browsers in use. Each of these blocks will be explained after the code listing.

Listing 3. Creating an Access Report

```
 1: <?php
 2: //connect to server and select database
 3: $conn = mysql_connect("localhost", "joeuser", "somepass")
 4:     or die(mysql_error());
 5: $db = mysql_select_db("testDB", $conn) or die(mysql_error());
 6:
 7: //issue query and select results for counts
 8: $count_sql = "select count(page_title) from access_tracker ";
 9: $count_res = mysql_query($count_sql, $conn) or
die(mysql_error());
10: $all_count = mysql_result($count_res, 0, "count(page_title)");
11:
12: //issue query and select results for user agents
13: $user_agent_sql = "select distinct user_agent, count(user_agent)
as count
14:     from access_tracker group by user_agent order by count desc";
15: $user_agent_res = mysql_query($user_agent_sql, $conn)
16:     or die(mysql_error());
17: //start user agent display block
18: $user_agent_block = "<ul>";
19:
20: //loop through user agent results
21: while ($row_ua = mysql_fetch_array($user_agent_res)) {
22:    $user_agent = $row_ua['user_agent'];
23:    $user_agent_count = $row_ua['count'];
24:    $user_agent_block .= "
25:    <li>$user_agent
26:         <ul>
27:         <li><em>accesses per browser: $user_agent_count</em>
28:         </ul>";
```

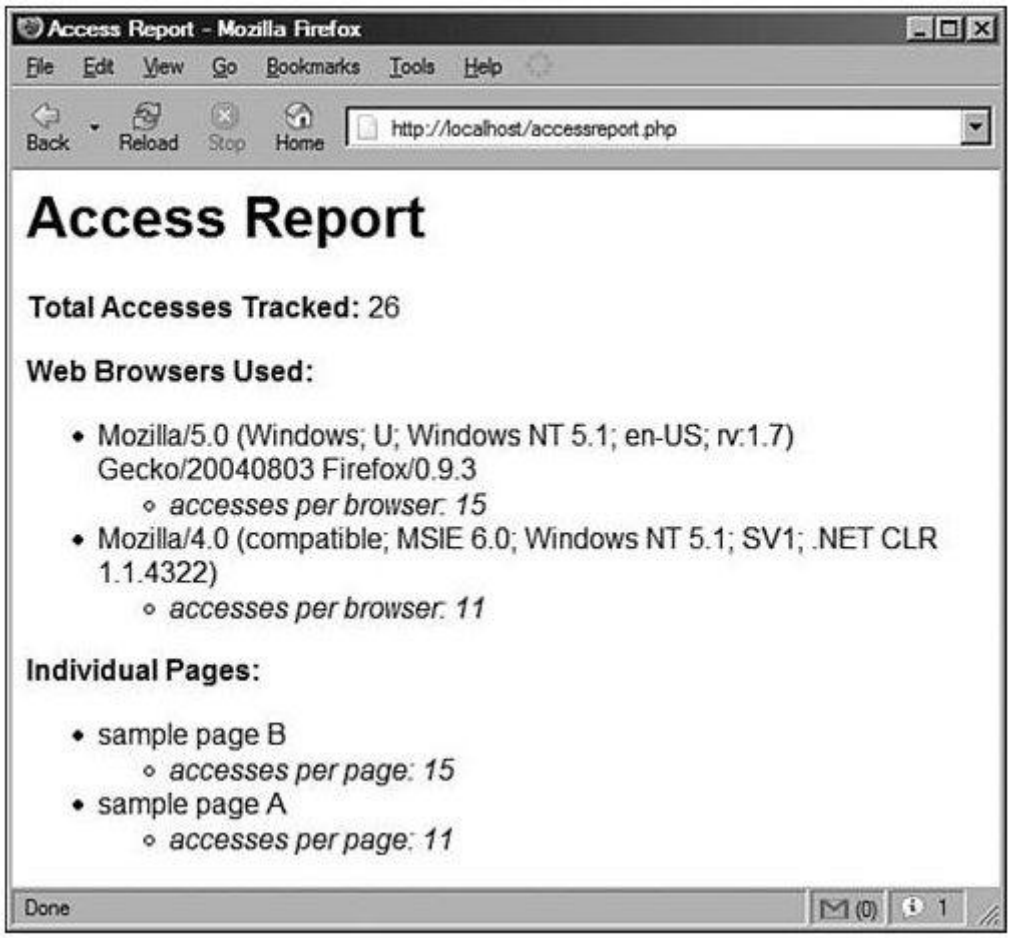
```
29: }
30:
31: //finish up the user agent block
32: $user_agent_block .= "</ul>";
33:
34: //issue query and select results for pages
35: $page_title_sql = "select distinct page_title, count(page_title)
as count
36:     from access_tracker group by page_title order by count desc";
37: $page_title_res = mysql_query($page_title_sql, $conn)
38:     or die(mysql_error());
39: //start page title display block
40: $page_title_block = "<ul>";
41:
42: //loop through results
43: while ($row_pt = mysql_fetch_array($page_title_res)) {
44:    $page_title = $row_pt['page_title'];
46:    $page_count = $row_pt['count'];
47:    $page_title_block .= "
48:    <li>$page_title
49:        <ul>
50:        <li><em>accesses per page: $page_count</em>
51:        </ul>";
52: }
53:
54: //finish up the page title block
55: $page_title_block .= "</ul>";
56:
57:?>
58: <HTML>
59: <HEAD>
60: <TITLE>Access Report</TITLE>
61: </HEAD>
62: <BODY>
63: <h1>Access Report</h1>
64: <P><strong>Total Accesses Tracked:</strong> <? echo "$all_count";
?></p>
65: <P><strong>Web Browsers Used:</strong>
66: <?php print "$user_agent_block"; ?>
67: <P><strong>Individual Pages:</strong>
68: <?php print "$page_title_block"; ?>
69: </BODY>
70: </HTML>
```

Lines 35 connect to the database so that you can issue the queries against the `access_tracker` table. Lines 810 issue the query to select the count of all pages, and lines 13 - 15 count the user agent accesses. Line 18 starts an unordered list block for the results of the user agent query, while lines 21- 29 loop through the results and create the list, which is closed in line 32.

Lines 35 - 37 create and issue the query to count the individual pages. Line 40 starts an unordered list block for the results of this query, and lines 43 - 52 loop through the results and create the list of accessed pages, which is closed in line 55.

Put these lines into a text file called `accessreport.php`, and place this file in your Web server document root. When you access this report, you will see something like Figure 1your page names, counts, and browsers will be different, but you get the idea.

Figure 1. Custom access report for tracked pages.



This sort of tracking is a lot easier than wading through Apache access logs, but I wouldn't recommend completely replacing your access logs with a database-driven system. That's a bit too much database-connection overhead, even if MySQL is particularly nice on your system. Instead, target your page tracking to something particularly important.

Workshop

The workshop is designed to help you anticipate possible questions, review what you've learned, and begin learning how to put your knowledge into practice.

Quiz

1. How would you avoid logging hits from a client accessing your Web site from a particular network?

2. How can you log images to a different file?

Answers

1. In some situations, you may want to ignore requests coming from a particular network, such as your own, so that they do not skew the results. You can do this either by post-processing the logs and removing them or by using the `SetEnvIf` directive:

```
SetEnvIf Remote_Addr 10\.0\.0\. intranet
CustomLog logs/access_log "%h %l %u %t \"%r\" %>s %b"
!intranet
```

2. Earlier in the lesson, you learned how to avoid logging images. Instead of ignoring images altogether, you can easily log them to a separate file, using the same environment variable mechanism:

```
SetEnvIf Request_URI "(\.gif|\.jpeg)$" image
CustomLog logs/access_log common env=!image
CustomLog logs/images_log common env=image
```