

***weblog* 1.1: A web access logging tool**

John W. Shipman

\$Date: 2003/02/16 00:44:32 \$

Table of Contents

1. Requirements	1
2. Input: the access logs	2
3. Filtering of the logs	4
4. Operation of the program	5
5. Outputs	5
6. Internals of the <i>weblog</i> script	5
6.1. The <i>webstats.py</i> script.....	6
6.2. Objects for the <i>weblog</i> application: <i>weblog.py</i>	6
6.3. Objects for the <i>weblog</i> application: <i>pageget.py</i>	7

1. Requirements

Web page authors may be curious about whether anyone is seeing their pages. The purpose of the *weblog* program is to provide a web page where anyone can see which pages are being visited.

The output of this program addresses two kinds of questions:

- Is anyone visiting this page? For this query, we need a list of all the pages sorted by URL, with hit counts for each one.
- What are the most popular pages on this server? To answer this question, we need a “hit parade” that lists page hits in descending order.

Both reports show, for each URL, the percentage of off-site accesses. Accessors not in the **nmt.edu** domain (129.138.*) are considered off-site.

At the request of the New Mexico Tech Public Information Office, the program should maintain data covering at least the last month's accesses.

2. Input: the access logs

At the New Mexico Tech Computer Center, logs of all Web page accesses are maintained in directory `/u/www/logs`.

The server is constantly writing to file `access_log`. The format of this file is described in the Apache Web server documentation (http://www.apache.org/docs/mod/mod_log_config.html), under "Common Log Format."

As of late December 2002, here are the fields of this log:

- The primary accessing host name, in symbolic (`array500.nmt.edu`) or numeric (`129.138.79.1`) form.
- Following one space, the next field is either missing (for logs through 2003-1-1), or equal to `'-'`, or it contains a list of secondary host names with a comma and space between them. If it is `'-'`, the primary host is the actual accessor. Otherwise, the primary host is only a proxy, and the list of secondary hosts represents one access from each host.

A host is considered to be *local* if its name is a single word containing no periods, or if it ends in `nmt.edu`, or if it starts with `129.138`. The exception is the single name *unknown*, which often appears in secondary host lists, and is considered not local.

- The literal string `' - - ['`. For access to a password-controlled page, the second hyphen is replaced by the user name.
- The date, time, and zone correction. For example: `'08/Jan/2003:00:45:59 -0700'`.
- Literal `'] "'` (close bracket, space, double-quote).
- The command, typically `'GET'` or `'POST'`, followed by one space. Very occasionally other values will appear, but those appear to be garbage values sent by defective clients elsewhere.
- The URL that the accessor wants to load. This may or may not be available (it might not exist, or not be world readable); refer to the status code to find out whether it loaded successfully.

The URL in the log file uses “URL encoding” (e.g., %7E for the tilde (~) character), so we’ll decode all those characters. Also, if the URL contains a CGI epilogue starting with ‘?’, we’ll ignore that part so we treat all references to a CGI script as the same page.

- One space, then the protocol used, typically ‘HTTP/1.0’ or ‘HTTP/1.1’.
- The closing double-quote and one space.
- The result code. This is 200 for a normal access; codes 300 and up represents redirects or failures.
- One space, then the length of the page fetched, or ‘-’.
- One space, then the URL of the page on which this link was clicked, enclosed in double-quotes. If the URL is unknown, the field contains ‘”-”’. As with the previous URL field, we’ll strip any CGI epilogue and decode any URL-encoded characters.

The *effective host list* is the list of hosts actually accessing the page:

- If there are no secondary hosts, the effective host list is just the primary host.
- If one or more secondary hosts are given, the primary host is just a proxy, and the effective host list is the list of secondary hosts.

To clarify, here are some example primary and secondary host fields, and the corresponding effective host lists:

Primary/Secondary	Effective Host List
infohost - -	infohost
armstrong-21.nmt.edu - - -	armstrong-21.nmt.edu
array500.nmt.edu 207.66.112.35 - -	207.66.112.35
193.231.29.177 unknown, unknown - -	unknown, unknown
metis02.abnamro.nl 10.50.102.150, 10.120.34.113, 10.120.34.14 - -	10.50.102.150, 10.120.34.113, 10.120.34.14
proxy.google.com 158.110.40.90, unknown - -	158.110.40.90, unknown

Every night around midnight, a *cron* job runs that rotates the log files. In directory

`/u/www/logs`, there are subdirectories for about two months' worth of old files. Each subdirectory has the name `yyyymmdd` and contains the `access_log` and other files. There is a handy soft link named `yesterday` that points to yesterday's subdirectory, and it is the `access_log` in that directory that is the input to *weblog*.

Since *weblog* must maintain at least a month's statistics, and because the log files are pretty sizeable, part of *weblog*'s job is to summarize the data from the log files and write its own database. Summarizing the data reduces disk space requirements, and the database is kept elsewhere so that the timespan can be the required length of a month.

Accordingly, the *weblog* program will have this intended function:

```
[ database      := database - expired-records + new log file
  web-pages     := reports summarizing (database -
                                       expired-records + new log file) ]
```

so that it can be run as a cron job, taking its input from the (unzipped) `access_log.1.gz` file, discarding any records older than a month, adding records from the latest daily log, and updating the web pages that show its results.

3. Filtering of the logs

Not every page access is counted by this program. In particular:

- Page fetches with a status over 300 are considered failed accesses. That includes redirects; if a page is redirected, there will be a separate record for the successful access that will be counted.
- Image files are ignored. An URL is considered an image file if it ends with `.jpg`, `.jpeg`, `.gif`, or `.png`, ignoring case.
- Stylesheets (files ending in `.css`) are also ignored.
- Accesses from known search engine spiders are ignored. At this writing, only the local machine `infohost` is considered a spider.
- Accesses to pages requiring password authentication are ignored. The intent of this program is to track only publicly accessible pages. Those parties interested in accesses to closed sections can look at the log files, which are world-readable.

We have had requests for statistics showing the referring page, where known. Although this information is in the access logs, the storage requirements for this program without that feature already strain the local computing resources, the

data files requiring hundreds of megabytes. Keeping track of several referring URLs for every page fetched would be prohibitively expensive in storage and time.

4. Operation of the program

The script runs as a *cron* job, at some reasonable time after the log files are rotated (say, an hour).

The script should be run on **infohost** for performance. The line to start it should look like this:

```
nice +5 zcat ~/www/logs/yesterday/access_log.gz | \  
/u/www/docs/tcc/projects/weblog/webstats.py
```

The program reads the new log file from its standard input, merges it with its own summary database at a known location, rewrites the summary database, and builds the output files (q.v.).

5. Outputs

The program generates three pages in directory `/u/www/docs/tcc/webstats/`:

- `homepage.html` describes the range of dates contained in this report, summarizes the totals by off- and on-campus accessors, and has links to the other two pages.
- `byurl.html` allows users to look at all the usage stats for their particular page structure. It shows the hit count for the Institute homepage, followed by a list of links for every "segment," that is, for every unique part of the URL just following the `www.nmt.edu` part. For example, there will be a link for the `LDP` subdirectory and the `mainpage` subdirectory and all the other directories under `/u/www/docs`. Following those links will be the segments for usernames of the form `~username`. To reduce page sizes, each segment lives in a separate page in subdirectory `seg`; files are numbered `s00001.html`, `s00002.html`, and so on.
- `byhits.html` presents the same items as `byurl.html`, but they are sorted in descending order by number of hits. Entries with the same number of hits are alphabetized by URL.

6. Internals of the *weblog* script

These files contain the code for this script:

- File `run` is a wrapper allowing the Python script to run with `setuid`. This is a short C program whose source is `run.c`.
- File `webstats.py` is the main Python script.
- File `weblog.py` contains three classes and a function that are used by the `run` script.
- File `pageget.py` encapsulates the logic for scanning Apache's `access_log` files.

6.1. The `webstats.py` script

Overall, the script has these steps:

1. Read our database file, `weblog_db`, containing the usage data from previous days. This database is represented as a **WebAccessDB** object (see `weblog.py` for the definition of this class). The **WebAccessDB** object is stored in variable **db**.

If this is the first run of the script, the database file doesn't exist yet. If it doesn't (we determine this by using `os.path.exists()`), we call the **WebAccessDB** constructor with a value of **None** for the file name, which causes it to return an empty **WebAccessDB** object.

The logic for this step is contained in function `openDatabase()`.

2. Read the new log entries from the standard input stream, adding them to the database object **db**.

This step corresponds to function `addLog()`.

3. Write the database back out to the file. This is subroutine `writeDatabase`.
4. Write three web pages into directory `/u/www/docs/tcc/webstats/`. The first page is `homepage.html` and shows the time range and the total on- and off-campus accesses. This page is pretty short. The other two pages are `byurl.html`, which lists all accesses alphabetized by URL, and `byhits.html`, showing the same data but sorted in descending order of number of hits.

The logic for this step is in function `writeWeb()`.

6.2. Objects for the *weblog* application: `weblog.py`

The `weblog.py` module contains these items used by the *weblog* application.

- Class **WebAccessDB** is a container class for all the data we're keeping about web accesses. It segregates the data by date, so we can discard records that are older than we want. It can also write itself to a file, and its constructor can read back that file.

This object also keeps track of the total number of near (on-campus) and far (off-campus) accesses that it contains, as well as timestamps for its oldest and newest access.

- Class **PageUsage** contains a summary of one or more page accesses. Its contents include the URL of a page and counters for the number of near and far accesses.
- Function `Freq_Cmp()` is passed as a compare function to sort `PageUsage` objects into descending order by total hits.

6.3. Objects for the *weblog* application: `pageget.py`

The `pageget.py` module contains one external function and one class:

- Function `scanAccessLog()` is a Python function that reads an Apache `access_log` file and generates a stream of `PageGet` objects representing the accesses described in that file.

This function uses a relatively new Python feature called *generators*. For more on generators, see *What's new in Python 2.2* (<http://www.python.org/doc/2.2.1/whatsnew/>) by Andrew Kuchling.

Generators are particularly elegant in this case because there isn't a one-to-one correspondence between log lines and page accesses. Although most log lines describe a single access, some lines aren't valid, and some describe multiple accesses through a proxy. I've seen examples of up to three accesses in a single log line. See file `oddballs` in the development directory for some interesting examples.

- Class **PageGet** represents one line from the access log file (located in `/u/www/logs/access_log`). It encapsulates the logic for scanning log file lines and determining whether they are near or far accesses.

