Download Counter

Release 0.9.0

Steve Daulton

Mar 28, 2022

CONTENTS

Version 0.9.0

CHAPTER

ONE

CONTENTS

1.1 Getting Started

1.1.1 Installation

Before you start

To avoid problems, read this page in full before running the app on a live server.

Dependencies

Download Counter requires Python3. Tested with Python 3.8.10 - slightly earlier versions may work, but are untested.

Permissions

dlcounter.py may either be run by passing the command to python:

```
$ python3 dlcounter.py <args>
```

or by making dlcounter.py executable, then run by entering the command *path/name* and (optional) argments:

```
$ sudo chmod +x dlcounter.py
$ dlcounter.py <args>
```

Reading server access logs requires root / admin access. Run dlcounter.py as root / admin. For example, to initialise the database on Linux:

```
$ sudo ./dlcounter.py -i /var/log/nginx/access.log -v
```

Installing

To use Download Counter, place **dlcounter.py**, **dlcounter_html.py** and **dlcounter.cfg** into a suitable directory *outside* of your website. For example, they could be placed in a folder in your home directory:

```
$ mkdir ~/download_counter
$ mv dlcounter.py ~/download_counter
$ mv dlcounter_html.py ~/download_counter
$ mv dlcounter.cfg ~/download_counter
$ sudo chmod +x ~/download_counter/dlcounter.py
```

Alternatively, if Download Counter is obtained as an archive file, simply extract the entire package to a convenient location outside of you website. Remember to set file execute permission for dlcounter.py if required.

Files

- **dlcounter.py** is the main app.
- **dlcounter.cfg** contains the configuration settings. (This file must be *customised* before use.)
- **dlcounter_html.py** provides an HTML template for html output.
- **downloads.db** (created when Download Counter is *initialised*) is the database that stores the data.

1.1.2 Configuration

Configuration file

The app has a configuration file *"dlcounter.cfg"* that may be edited with any plain text editor. For example, to edit with nano :

```
$ cd ~/download_counter
$ nano dlcounter.cfg
```

Sections

- **[ACCESSLOGS]** One or more server logs to analyse.
- **[FILEPATH]** The first part of the name of file(s) to be counted from the access log.
- **[FILENAMES]** The final part of the name of file(s) to be counted from the access log. Typically this will be one or more file extensions.
- **[WEBPAGE]** The HTML file to display download totals. Typically this will be within the website html directory.
- **[DATETIME]** Datetime formats for reading access logs and writing HTML.
  - **datetime_read** Format for reading access logs.
  - **datetime_write** Format for writing html webpage.

Further details can be found in the *Customisation* section.

Command Line

The following command line switches are provided:

-d, --docs
Show built-in documentation and exit.

-h, --help
Show short help and exit.

-i, --init'path/to/access.logs'*
Initialise database. (See: *"Initialising"*).

-v, --verbose
Verbose output.
Prints arguments, options, and the database contents to stdout.
Along with -D (--debug) this can be useful to check the
configuration and for debugging.

-D, --debug
Prints debug information to stdout.
Along with -v (--verbose) this can be useful to check the
configuration and for debugging.

-V, --version
Show version and exit.

Initialising

Download Counter is initialised by running dlcounter.py as root/admin with the -i (--init) switch, and the path to to the access logs. It is a good idea to run this manually from the command line with the -i (--init), -D (--debug), and -v (--verbose) options. Note that the path must include the base filename.

All access logs in *'path/to/access.logs'*, (including .gz files), are read. Matching download files are counted regardless of when they were downloaded. This option overrides [ACCESSLOGS] in dlcounter.cfg and should only be used on first run. If the database already exists, the downloads table will be deleted and recreated.

Example

```
python dlcounter.py -i '/var/log/nginx/access.log'
```

This example will read all logs:

- /var/log/nginx/access.log
- /var/log/nginx/access.log.1
- /var/log/nginx/access.log.2.gz
- /var/log/nginx/access.log.3.gz
- …

It does not attempt to read other logs such as `error.log`.

Running the App

After *initialising* the database, and setting up the *configuration* options, dlcounter.py may be run at any time to update the database and html output. To ensure that all downloads are caught, dlcounter.py should be run immediately prior to log rotation. (Alternatively, logs from the current and previous day could be analysed.)

Tip:

Run the program with the -v (--verbose) and -D (--debug) switches, and redirect output to a text file to check that it is running as expected.

Log Rotation

By default, Ubuntu uses logrotate to rotate logs once per day. Ideally, download counter should be run immediately before the logs are rotated so that all records for the previous 24 hours are analyzed.

How it works

This example describes the default setup for Ubuntu / Nginx.

The the script `/etc/cron.daily/logrotate` runs daily and executes

```
/usr/sbin/logrotate /etc/logrotate.conf
```

The **logrotate.conf** file contains the global default settings for logrotate and *includes* additional configuration files in `/etc/logrotate.d`.

Among the files in `/etc/logrotate.d` is the logrotate configuration for nginx:

```
/var/log/nginx/*.log {
   daily
   missingok
   rotate 14
   compress
   delaycompress
   notifempty
   create 0640 www-data adm
   sharedscripts
   prerotate
      if [ -d /etc/logrotate.d/httpd-prerotate ]; then \
         run-parts /etc/logrotate.d/httpd-prerotate; \
      fi \
   endscript
   postrotate
      invoke-rc.d nginx rotate >/dev/null 2>&1
   endscript
}
```

The line `compress` specifies that the log files are compressed, but `delaycompress` delays the compression of the most recent log until the next rotation cycle.

The section between `prerotate` and `endscript` checks for the existence of the directory `/etc/logrotate.d/httpd-prerotate`. If it exists, then *executable* scripts within that directory are run before the logs are rotated. Thus a script can be scheduled to run immediately before rotation by placing it in the folder `/etc/logrotate.d/httpd-prerotate`.

Note: By default, `run-parts` requires that script names must consist entirely of ASCII upper- and lower-case letters, ASCII digits, ASCII underscores, and ASCII minus-hyphens. In particular, **dots are not allowed**, so file extensions must not be used.

Example

If `/etc/logrotate.d/httpd-prerotate` does not exist, create it:

```
$ sudo mkdir /etc/logrotate.d/httpd-prerotate
```

Create a bash script to run **dlcounter.py**. Initially we will run with verbose (-v) and debug (-D) options, and redirect the output to a file to check that it is running as expected

```
$ sudo nano /etc/logrotate.d/httpd-prerotate/dlcounter
```

Example script:

```
#!/bin/bash

output="/home/<username>/dlcount.txt"

date +"%Y-%m-%d %H:%M:%S.%N %z" > $output
echo -e "-----------------\n" >> $output
/home/<username>/dlcounter/dlcounter.py -v -D >> $output
```

And make the script executable:

```
sudo chmod +x /etc/logrotate.d/httpd-prerotate/dlcounter
```

1.2 Customisation

Before use, the **dlcounter.cfg** file must be customised to suit your server setup. This file must be in the same directory as **dlcounter.py**.

Below are descriptions of the sections of the config file, and suggested settings for common setups based on a WordPress installation running on Nginx and Ubuntu Linux.

1.2.1 [ACCESSLOGS]

Specify the log file(s) to analyse.

Typically, for a WordPress site on Nginx, this will be: `/var/log/nginx/access.log`, which contains data since the last log rotation. Provided that downloadcounter runs immediately before log rotation it is sufficient to analyze just this one log file once per day. See the *"Log Rotation"* section for more information about how to use downloadcounter with logrotate.

Example

```
[ACCESSLOGS]
log1 = /var/log/nginx/access.log
```

Downloads are only counted if their time stamp is more recent than any downloads already counted. If it is necessary to analyze aditional (older) log files, then the files must be listed oldest first. All log files listed here must be plain text files.

Example

```
[ACCESSLOGS]
log1 = /var/log/nginx/access.log.1
log2 = /var/log/nginx/access.log
```

1.2.2 [FILEPATH]

The first part of the search string to identify the downloads in the log files.

This version of Download Counter supports only one FILEPATH parameter. Typically the downloads to be counted will have a common file path, which for WordPress sites is in the form:

```
.../wp-content/uploads/*<year>*/*<month>*/*<filename>*
```

As "/wp-content/uploads/" is common to all download files, this is used as the first part of the search string when finding downloaded files.

Example

```
[FILEPATH]
path = /wp-content/uploads/
```

1.2.3 [FILENAMES]

The last part of the file(s) to search for in the log files. Typically this will be a list of file extensions.

Example

```
[FILENAMES]
file1 = .zip
file2 = .exe
```

1.2.4 [WEBPAGE]

The location for html output.

Download Counter generates a web page for viewing the download totals. Typically this will be located within the public html directory of your website. **This must be a fully qualified path**. If omitted, no html will be generated.

Example

```
[WEBPAGE]
path = /var/www/html/downloads.html
```

1.2.5 [DATETIME]

Datetime formats for reading access logs and writing the html webpage. This section has two settings, both of which are *required*:

datetime_read

Format for reading access logs.

Default: %d/%b/%Y:%H:%M:%S %z

The default matches: “01/Jan/2022:23:35:05 +0000”

datetime_write

Format for writing html webpage.

Default: %a %d %b %H:%M

The default matches: “Mon 01 Jan 18:35”

1.3 API

1.3.1 Download counter

Searches access.log files for successful downloads that match a specified search string. Matching downloads are tallied in an SQLite database, and results output to an html file.

1.3.2 Usage

The main functional parameters are set in dlcounter.cfg. Additional options may be set through command line arguments.

Command line switches

Usage: dlcounter.py [-d] [-h] [-i] [-v] [-D] [-V]

Arguments:

-d, --docs	Show this documentation and exit.
-h, --help	Show short help and exit.
-i, --init string	This option is required if you wish to count downloads in old archived '.gz' files. All access logs in path, (including .gz files), are read. Matching download files are counted regardless of when they were downloaded, so this option should only be used on first run, (before the database contains data). This option overrides ACCESSLOGS in dlcounter.cfg.

Example

The path string should be entered in the form:

```
$ python3 dlcounter.py -n '/var/log/nginx/access.log'
```

To read all logs:

```
* /var/log/nginx/access.log
* /var/log/nginx/access.log.1
* /var/log/nginx/access.log.2.gz
* /var/log/nginx/access.log.3.gz
* ...
```

-v, --verbose	Print commands and database contents to stdout.
-D, --debug	Print additional debug strings to stdout.
-V, --version	Show program version and exit.

Configuration file

The configuration file ('dlcounter.cfg') must be in the same directory as 'download_counter.py'.

[ACCESSLOGS] One or more access logs.

Log files must be plain text (not .gz archives). When more than one access.log files specified, files must be in reverse chronological order (process oldest first).

Default: log1 = /var/log/nginx/access.log

[FILEPATH] The first part of the download file's string.

This refers to the string as it appears in the access log. If not supplied, all file names matching the [FILENAMES] option(s) will be counted.

Default: path = /wp-content/uploads/

The default option will catch files in any of:

- …/website/downloads/2001/
- …/website/downloads/2002/
- …/website/downloads/…/

[FILENAMES] Download files end of string.

The default options will catch .zip and .exe files that begin with 'FILEPATH'.

Default:

- file1 = .zip
- file2 = .exe

[WEBPAGE] Fully qualified path for html output.

HTML output is disabled if this path is not specified.

Default: path = /var/www/html/downloads.html

[DATETIME] Datetime formats for reading access logs and writing HTML.

- **datetime_read**

  Format for reading access logs. The default matches: 01/Jan/2022:23:35:05 +0000

  Default: %d/%b/%Y:%H:%M:%S %z

- **datetime_write**

  Format for writing html webpage. The default matches: Mon 01 Jan 18:35

  Default: %a %d %b %H:%M

Note:

> [FILEPATH] and [FILENAMES] options are just strings to search for in the accesslog file(s). Regex is used to search the log file(s) for: "<path-string> any-characters <file-string>"

`dlcounter.`**`check_path`**`(`*name*, *file*`)`

Check if file exists.

Parameters

- **`name`** (*`string`*) – File identifier / file name.
- **`file`** (*`string`*) – File path.

Returns True or exit.

Return type bool

`dlcounter.`**`db_path`**`()`

Path to database file.

Parameters **`None`** –

Returns Fully qualified path to SQLite database file.

Return type string

`dlcounter.`**`first_item_in_section`**`(`*cfg*, *section*`)`

Return the first item from cfg section.

Parameters

- **`cfg`** (*`configparser.ConfigParser`*) – The ConfigParser object.
- **`section`** (*`string`*) – Section key.

Returns First value from section, or empty string.

Return type string

`dlcounter.`**`format_datetime_output`**`(`*dt_string*`)`

Format dt_string as required for html output.

Parameters **`dt_string`** (*`datetime`*) – datetime object.

Returns Reformatted datetime string.

Return type string

`dlcounter.`**`get_config`**`()`

Return dict of arguments from dlcounter.cfg.

List values are retrieved by *`list_section()`*. Single values retrieved by *`first_item_in_section()`*. Also print parameters from command line and config file when –verbose command line argument is passed.

Parameters **`None`** –

Returns Values from configuration file.

Return type dict

`dlcounter.`**`get_db_time`**`(`*con*`)`

Return most recent timestamp from database.

Parameters **`con`** (*`connection`*) – Connection to the database.

Returns datetime.min if timestamp not found.

Return type datetime

`dlcounter.`**`get_record`**(*record*, *pattern*)

Return (filename, time) from line when start and end of name are found, or None if not found.

Parameters

- **`record`** (*`string`*) – One line from access log.
- **`pattern`** (*`string`*) – Regex pattern: f'GET {re.escape(start)}.*{re.escape(end)}'

Returns (Short filename string, datetime object) if successful,or None.

Return type tuple

`dlcounter.`**`get_time`**(*record*)

Return timestamp from record.

Parameters **`record`** (*`string`*) –

Returns Exit if timestamp not found.

Return type datetime

`dlcounter.`**`init_db`**(*logpath*, *opt*)

Initialise database.

Similar to main() but reads all logs that start with 'logpath' and does NOT check timestamp before counting. If the database already exists, the old table will be deleted and a new table created.

Parameters

- **`logpath`** (*`string`*) – Path to access logs.
- **`opt`** (*`dict`*) – Parameters from dlcounter.cfg.

Return type None

`dlcounter.`**`list_section`**(*cfg*, *section*)

Return list of values from cfg section.

Parameters

- **`cfg`** (*`configparser.ConfigParser`*) – The ConfigParser object.
- **`section`** (*`string`*) – Section key.

Returns List of zero or more values.

Return type list

`dlcounter.`**`log_to_sql`**(*con*, *file*, *searchstring*, *timecheck=None*)

Copy download data from log file to database.

Read one log file and update database. Updating is handled by `update_db()`.

Parameters

- **`con`** (*`connection`*) – Connection to the database.
- **`file`** (*`_io.TextIOWrapper`*) – Pointer to access log.
- **`searchstring`** (*`string`*) – Regex pattern for filename in log.
- **`timecheck`** (*`datetime`*) – Initialise if timecheck=None.

Returns True when database has been modified.

Return type bool

`dlcounter.main(opt)`

Count downloads from access logs.

Search for file names in the access logs that match the search criteria, and update the database as necessary. The database is created automatically if it does not exist. Downloads with timestamps older than the last update are ignored.

Parameters **opt** (*dict*) – Contains string values: acclogs, searchstring, and html_out.

Return type None

`dlcounter.print_table()`

Print contents of database to stdout.

This function is used only with –verbose option.

Parameters **None** –

Return type None

`dlcounter.sql_table(con)`

Create 'downloads' table if it doesn't exist.

Parameters **con** (*connection*) – Connection to the database.

Return type None

`dlcounter.time_format(readf='', writef='')`

Return the date-time format

Values from config for reading access logs and writing html. Call either time_format.read or time_format.write.

Parameters

- **readf** (*string, default ''*) – Time format for reading access logs.
- **writef** (*string, default ''*) – Time format for writing html.

`dlcounter.read`

Type string

`dlcounter.write`

Type string

Return type None

`dlcounter.update_db(con, fname, timestamp)`

Update database.

If fname exists in database, update its download total and timestamp, else insert it into the database with a count of 1.

Parameters

- **con** (*connection*) – Connection to the database.
- **fname** (*string*) – Name of the downloaded file.
- **timestamp** (*datetime*) – Timestamp of download.

Return type None

`dlcounter.`**`write_html`**`(`*con*, *htmlfile*`)`

Write sql data to web page.

Parameters

- **`con`** (*`connection`*) – Connection to the database.
- **`htmlfile`** (*`string`*) – Path to html output file.

Return type None

1.4 Web Page Template

HTML code for generating web page output.

dlcounter.py creates table data between `html_top()` and `html_bottom()`. This module provides the rest of the HTML for the download counter web page.

This file may be modified according to need.

`dlcounter_html.`**`html_bottom`**`()`

Return end of html page.

Returns HTML output.

Return type string

`dlcounter_html.`**`html_top`**`(`*timestamp*`)`

Return beginning of html page.

Returns HTML output.

Return type string

1.5 Download Counter

A standalone Python app to keep a tally of downloads from a website.

This app was developed for use with WordPress / NGINX / Ubuntu, but will probably work on other platforms with minimal alterations. Patches to support other common platforms are welcome.

CHAPTER

TWO

RATIONALE

While there are several WordPress modules for managing downloads, in 2022 it seems that they are all large, complex modules aimed at e-commerce. From a security perspective, it is advantageous to avoid presenting a larger attack target than absolutely necessary, hence this small download counter utility.

CHAPTER

THREE

WHAT IT DOES:

This app scans the server access logs and searches for downloaded files that match the given search criteria. When found, they are logged in an SQLite database, along with the download timestamp and a count of how many times the file has been downloaded.

Finally, the app generates an HTML page to display the database contents.

CHAPTER

FOUR

HOW IT WORKS

Nginx logs all access traffic in its ‘access.log’. On a daily schedule the logs are rotated by default (Ubuntu / Nginx):

- access.log -> access.log.1
- access.log.1 -> access.log.2.gz
- …
- access.log.13.gz -> access.log.14.gz
- access.log.14.gz -> deleted.

Each file downloaded from a website on the server has a log entry in the form:

```
Remote-IP - - [local-time-date] "GET /path/to/file.ext protocol"/
status-code bytes_sent "http-referer" "http-user-agent"
```

As (by default) the ‘access.log’ file only contains logs for the current day, this app should be run once per day (by a root cron job), immediately before log rotation. See the section *“Log Rotation”* for how to do this. (Alternatively, both ‘access.log’ and ‘access.log.1’ could be analyzed at any time of day, though this is much less efficient.)

From the log we need to find:

1. lines containing the file search string
2. with status code 200
3. after the last time it was counted

The results are stored in an SQLite database, and written to an HTML file.

PYTHON MODULE INDEX

INDEX