Find Dupes Fast Documentation

Release 0.3.6

Stephan Sokolow

Sep 27, 2017

Contents

CHAPTER 1

API Documentation

Find Dupes Fast By Stephan Sokolow (ssokolow.com)

A simple script which identifies duplicate files several orders of magnitude more quickly than fdupes by using smarter algorithms.

Todo

Figure out how to do ePyDoc-style grouping here without giving up automodule-level comfort.

`fastdupes.`**`CHUNK_SIZE`** **= 65536**
  Size for chunked reads from file handles

`fastdupes.`**`DEFAULTS`** **= {'min_size': 25, 'exclude': ['*/.svn', '*/.bzr', '*/.git', '*/.hg'], 'delete': False}**
  Default settings used by `optparse` and some functions

`fastdupes.`**`HEAD_SIZE`** **= 16384**
  Limit how many bytes will be read to compare headers

class `fastdupes.`**`OverWriter`**(*fobj*)
  Bases: `object`

  Output helper for handling overdrawing the previous line cleanly.

  `write`(*text*, *newline=False*)
    Use `\r` to overdraw the current line with the given text.

    This function transparently handles tracking how much overdrawing is necessary to erase the previous line when used consistently.

    Parameters

    - **`text`** (`str`) – The text to be outputted
    - **`newline`** (`bool`) – Whether to start a new line and reset the length count.

`fastdupes.`**`compareChunks`**(*handles*, *chunk_size=65536*)

Group a list of file handles based on equality of the next chunk of data read from them.

Parameters

- **`handles`** – A list of open handles for file-like objects with otentially-identical contents.
- **`chunk_size`** – The amount of data to read from each handle every time this function is called.

Returns

Two lists of lists:

- Lists to be fed back into this function individually
- Finished groups of duplicate paths. (including unique files as single-file lists)

Return type `(list, list)`

Attention: File handles will be closed when no longer needed

Todo

Discard chunk contents immediately once they're no longer needed

`fastdupes.`**`delete_dupes`**(*groups*, *prefer_list=None*, *interactive=True*, *dry_run=False*)

Code to handle the `--delete` command-line option.

Parameters

- **`groups`** (*`iterable`*) – A list of groups of paths.
- **`prefer_list`** – A whitelist to be compiled by *`multiglob_compile()`* and used to skip some prompts.
- **`interactive`** (`bool`) – If `False`, assume the user wants to keep all copies when a prompt would otherwise be displayed.
- **`dry_run`** (`bool`) – If `True`, only pretend to delete files.

Todo

Add a secondary check for symlinks for safety.

`fastdupes.`**`find_dupes`**(*paths*, *exact=False*, *ignores=None*, *min_size=0*)

High-level code to walk a set of paths and find duplicate groups.

Parameters

- **`exact`** (`bool`) – Whether to compare file contents by hash or by reading chunks in parallel.
- **`paths`** – See *`getPaths()`*
- **`ignores`** – See *`getPaths()`*
- **`min_size`** – See *`sizeClassifier()`*

Returns A list of groups of files with identical contents

Return type `[[path, ...], [path, ...]]`

`fastdupes.`**`getPaths`**(*roots*, *ignores=None*)

Recursively walk a set of paths and return a listing of contained files.

Parameters

- **`roots`** (`list` of `str`) – Relative or absolute paths to files or folders.
- **`ignores`** (`list` of `str`) – A list of `fnmatch` globs to avoid walking and omit from results

Returns Absolute paths to only files.

Return type `list` of `str`

Todo

Try to optimize the ignores matching. Running a regex on every filename is a fairly significant percentage of the time taken according to the profiler.

`fastdupes.`**`groupBy`**(*groups_in*, *classifier*, *fun_desc='?'*, *keep_uniques=False*, **args*, ***kwargs*)

Subdivide groups of paths according to a function.

Parameters

- **`groups_in`** (`dict` of iterables) – Grouped sets of paths.
- **`classifier`** (`function(list, *args, **kwargs) -> str`) – Function to group a list of paths by some attribute.
- **`fun_desc`** (`str`) – Human-readable term for what the classifier operates on. (Used in log messages)
- **`keep_uniques`** (`bool`) – If `False`, discard groups with only one member.

Returns A dict mapping classifier keys to groups of matches.

Return type `dict`

Attention Grouping functions generally use a `set groups` as extra protection against accidentally counting a given file twice. (Complimentary to use of `os.path.realpath()` in *`getPaths()`*)

Todo

Find some way to bring back the file-by-file status text

`fastdupes.`**`groupByContent`**(*paths*)

Byte-for-byte comparison on an arbitrary number of files in parallel.

This operates by opening all files in parallel and comparing chunk-by-chunk. This has the following implications:

- Reads the same total amount of data as hash comparison.
- Performs a *lot* of disk seeks. (Best suited for SSDs)
- Vulnerable to file handle exhaustion if used on its own.

Parameters **`paths`** (*`iterable`*) – List of potentially identical files.

Returns A dict mapping one path to a list of all paths (self included) with the same contents.

Todo

Start examining the `while handles:` block to figure out how to minimize thrashing in situations where read-ahead caching is active. Compare savings by read-ahead to savings due to eliminating false positives as quickly as possible. This is a 2-variable min/max problem.

Todo

Look into possible solutions for pathological cases of thousands of files with the same size and same pre-filter results. (File handle exhaustion)

`fastdupes.`**`groupify`**`(`*function*`)`

Decorator to convert a function which takes a single value and returns a key into one which takes a list of values and returns a dict of key-group mappings.

Parameters **`function`** (`function(value) -> key`) – A function which takes a value and returns a hash key.

Return type

```
function(iterable) ->
    {key: set ([value, ...]), ...}
```

`fastdupes.`**`hashClassifier`**`(`*paths*, **args*, ***kwargs*`)`

Sort a file into a group based on its SHA1 hash.

Parameters

- **`paths`** – See *`fastdupes.groupify()`*
- **`limit`** (`__builtins__.int`) – Only this many bytes will be counted in the hash. Values which evaluate to `False` indicate no limit.

Returns See *`fastdupes.groupify()`*

`fastdupes.`**`hashFile`**`(`*handle*, *want_hex=False*, *limit=None*, *chunk_size=65536*`)`

Generate a hash from a potentially long file. Digesting will obey *`CHUNK_SIZE`* to conserve memory.

Parameters

- **`handle`** – A file-like object or path to hash from.
- **`want_hex`** (`bool`) – If `True`, returned hash will be hex-encoded.
- **`limit`** (`int`) – Maximum number of bytes to read (rounded up to a multiple of `CHUNK_SIZE`)
- **`chunk_size`** (`int`) – Size of `read()` operations in bytes.

Return type `str`

Returns A binary or hex-encoded SHA1 hash.

Note: It is your responsibility to close any file-like objects you pass in

`fastdupes.`**`main`**`()`

The main entry point, compatible with setuptools.

`fastdupes.`**`multiglob_compile`**(*globs*, *prefix=False*)

Generate a single "A or B or C" regex from a list of shell globs.

Parameters

- **`globs`** (iterable of `str`) – Patterns to be processed by `fnmatch`.
- **`prefix`** (`bool`) – If `True`, then `match()` will perform prefix matching rather than exact string matching.

Return type `re.RegexObject`

`fastdupes.`**`print_defaults`**()

Pretty-print the contents of *`DEFAULTS`*

`fastdupes.`**`pruneUI`**(*dupeList*, *mainPos=1*, *mainLen=1*)

Display a list of files and prompt for ones to be kept.

The user may enter `all` or one or more numbers separated by spaces and/or commas.

Note: It is impossible to accidentally choose to keep none of the displayed files.

Parameters

- **`dupeList`** (`list`) – A list duplicate file paths
- **`mainPos`** (`int`) – Used to display "set X of Y"
- **`mainLen`** (`int`) – Used to display "set X of Y"

Returns A list of files to be deleted.

Return type `int`

`fastdupes.`**`sizeClassifier`**(*paths*, **args*, ***kwargs*)

Sort a file into a group based on on-disk size.

Parameters

- **`paths`** – See *`fastdupes.groupify()`*
- **`min_size`** (`__builtins__.int`) – Files smaller than this size (in bytes) will be ignored.

Returns See *`fastdupes.groupify()`*

Todo

Rework the calling of `stat()` to minimize the number of calls. It's a fairly significant percentage of the time taken according to the profiler.

CHAPTER 2

TODO Note Index

Todo

Figure out how to do ePyDoc-style grouping here without giving up automodule-level comfort.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/fastdupes/checkouts/latest/fastdupes.py:docstring of fastdupes, line 26.)

Todo

Discard chunk contents immediately once they're no longer needed

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/fastdupes/checkouts/latest/fastdupes.py:docstring of fastdupes.compareChunks, line 18.)

Todo

Add a secondary check for symlinks for safety.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/fastdupes/checkouts/latest/fastdupes.py:docstring of fastdupes.delete_dupes, line 16.)

Todo

Try to optimize the ignores matching. Running a regex on every filename is a fairly significant percentage of the time taken according to the profiler.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/fastdupes/checkouts/latest/fastdupes.py:docstring of fastdupes.getPaths, line 13.)

Todo

Find some way to bring back the file-by-file status text

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/fastdupes/checkouts/latest/fastdupes.py:docstring of fastdupes.groupBy, line 26.)

Todo

Start examining the `while handles:` block to figure out how to minimize thrashing in situations where read-ahead caching is active. Compare savings by read-ahead to savings due to eliminating false positives as quickly as possible. This is a 2-variable min/max problem.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/fastdupes/checkouts/latest/fastdupes.py:docstring of fastdupes.groupByContent, line 16.)

Todo

Look into possible solutions for pathological cases of thousands of files with the same size and same pre-filter results. (File handle exhaustion)

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/fastdupes/checkouts/latest/fastdupes.py:docstring of fastdupes.groupByContent, line 21.)

Todo

Rework the calling of `stat()` to minimize the number of calls. It's a fairly significant percentage of the time taken according to the profiler.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/fastdupes/checkouts/latest/fastdupes.py:docstring of fastdupes.sizeClassifier, line 10.) Find Dupes Fast (A.K.A. `fastdupes.py`) is a simple script which identifies duplicate files several orders of magnitude more quickly than fdupes by using smarter algorithms.

It was originally inspired by Dave Bolton's dedupe.py and Reasonable Software's NoClone and has no external dependencies beyond the Python 2.x standard library.

Full API documentation is available on ReadTheDocs and, pending proper end user documentation, the `--help` option is being constantly improved.

CHAPTER 3

Algorithm

The default mode of operation is as follows:

1. The given paths are recursively walked (subject to `--exclude`) to gather a list of files.
2. Files are grouped by size (because `stat()` is fast compared to `read()`) and single-entry groups are pruned away.
3. Groups are subdivided and pruned by hashing the first `16KiB` of each file.
4. Groups are subdivided and pruned again by hashing full contents.
5. Any groups which remain are sets of duplicates.

Because this multi-pass approach eliminates files from consideration as early as possible, it reduces the amount of disk I/O that needs to be performed by at least an order of magnitude, greatly speeding up the process.

Here are the final status messages from a cold-cache run I did on my machine to root out cases where my manual approach to backing up things that don't change left duplicates lying around:

```
$ python fastdupes.py /srv/Burned/Music /srv/Burned_todo/Music /srv/fservroot/music
Found 72052 files to be compared for duplication.
Found 7325 sets of files with identical sizes. (72042 files examined)
Found 1197 sets of files with identical header hashes. (38315 files examined)
Found 1197 sets of files with identical hashes. (2400 files examined)
```

Those `... files examined` numbers should show its merits. The total wall clock runtime was 280.155 seconds.

Memory efficiency is also kept high by building full-content hashes incrementally in `64KiB` chunks so that full files never need to be loaded into memory.

CHAPTER 4

Exact Comparison Mode

If the `-E` switch is provided on the command line, the final full-content SHA1 hashing will be omitted. Instead, all of the files in each group will be read from the disk in parallel, comparing chunk-by-chunk and subdividing the group as differences appear.

This greatly increases the amount of disk seeking and offers no benefits in the vast majority of use cases. However, if you are storing many equally-sized files on an SSD and their headers are identical but they do vary, the incremental nature of this comparison may save you time by allowing the process to stop reading a given file as soon as it becomes obvious that it's unique.

The other use for this (avoiding the risk of hash collisions in files that have identical sizes and do not differ in their first `16KiB` of data but are different elsewhere) is such a tiny risk that very few people will need it.

(Yes, it's a fact that, because files are longer than hashes, collisions are possible... but only an astronomically small number of the possible combinations of bytes are meaningful data that you'd find in a file on your hard drive.)

CHAPTER 5

The `--delete` option

Like fdupes, fastdupes.py provides a `--delete` option which produces interactive prompts for removing duplicates.

However, unlike with fdupes, these prompts make it impossible to accidentally delete every copy of a file. (Bugs excepted, of course. A full unit test suite to ensure this behaviour is still on the TODO list.)

- The `--delete` UI asks you which files you'd like to *keep* and won't accept an empty response.
- Specifying a directory more than once on the command line will not result in a file being listed as a duplicate of itself. Nor will specifying a directory and its ancestor.
- A `--symlinks` option will not be added until safety can be guaranteed.

The `--prefer` and `--noninteractive` options

Often, when deduplicating with `--delete`, you already know that files in one directory tree should be preferred over files in another.

For example, if you have a folder named `To Burn` and another named `Burned`, then you shouldn't have to tell your deduplicator that files in the former should be deleted.

By specifying `--prefer=*/Burned` on the command-line, you can skip the prompts in such a situation while still receiving prompts for other files.

Furthermore, if you'd like a fully unattended deduplication run, include the `--noninteractive` option and fastdupes will assume that you want to keep all copies (do nothing) when it would otherwise prompt.

Finally, a `--dry-run` option is provided in case you need to test the effects of a `--delete` setup without risk to your files.

CHAPTER 6

Indices and tables

- genindex
- modindex
- *TODO Note Index*
- search

Python Module Index

Index