Dupemgr

User Documentation

Introduction

Rationale

Some people collect different kinds of files. Some may collect viruses. Some may collect pictures of fine art, buildings or stamps. I've even heard rumors that some people collect pictures of nude women(!)

After a while such collections may contain large numbers of files, and some of the content may be duplicated in different locations or file-names.

Dupemgr is meant to be an aid for collectors to identify duplicate files, and to handle them. It can also be used to identify missing files in a collection (by comparing the data-file of one collection with the data-file(s) of other collections).

Design-goals

- Easy to use, command-line utility.
- Handling of existing file-collections; where duplicates can be identified and replaced by soft-links, or deleted.
- Handling of new files, where only unique files are copied to the destination-folder(s).
- Identification of files based on calculated hash-values rather than file-names.
- Since it can take considerable time to build hash-values for a large collection of files, the internal cache can be stored on disk in an open format (xml) and then re-used.
- To help collectors exchange unique files, the program must be able to handle several cache-files (data-files), so that you can compare your collection with someones else collection, simply by exchanging the cache-files. The same mechanism can be used to build blacklists of unwanted files, or files you have stored off-line.
- Fast. The program must be fast and efficient. To accomplish that, it's written in C++, and it stores it's working-data in memory rather than in a traditional database. The time-consuming operations are scanning large directories, and to read all the files to calculate their hash-values.

How it works

Dupemgr's core is a fast lookup-cache of MD5 (or optionally, SHA1) hash-values representing the content of files on disk. When you run the program, you give it some path(s) to work with. It will search for all the files within the path(s), and read them one by one to calculate a hash-value that is unique to each file. It is theoretically possible that two different files gives the same hash-value, but in practice it is *very* unlikely to happen. (If this is a concern for you, use SHA1 rather than MD5).

The cache also contains the path, size and last modification time for each file that was found. Once the cache is built, dupemgr can perform operations on it. It can remove duplicate files, or soft-link duplicate files to one “master” file. It can also copy only unique files from one or more source-folders to a destination-folder.

In addition to this, dupemgr can save the cache-file to an XML data-file of your choice. Later on, you can tell it to load this file:

- To skip the time-consuming scanning and hashing process.
- To list the files in your cache.
- To search for a file by it's content or hash-value.
- To manipulate the cache itself.
- To avoid copying files that exist off-line or in another collectors collection.

The modified cache can of course be saved to a disk-file as well.

Since the data-files are stored in XML-format, it's trivial to manipulate them with other programs as well.

Note that this program can delete files on your hard-disk. If you are a collector, the files probably are precious to you. You should therefore always have a backup of your files somewhere else before running dupemgr or any other program that my delete parts of your collection. In my own experience, it's also a good idea to drink beer *after* messing with programs like this...

Copyright and License

Dupemgr is written by Jarle (jgaa) Aase. www.jgaa.com Email: mailto:jgaa@jgaa.com

Copyright 2009 by Jarle (jgaa) Aase. All rights reserved.

Dupemgr's homepage is: http://dupemgr.jgaa.com

Dupemgr is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

Commands

- **copy**: Copy files or directories from source location(s) to a destination-folder.
- **remove-duplicates**: Remove duplicates in the target-location(s).
- **scan**: Scan the target location(s) and build the cache. Typically used to save the cache for one or more locations to a data-file. This command has no effect unless you save the cache to a data-file.
- **list**: List the contents of one or more data-file(s).
- **list-existing**: List files on target(s) that exist in the first cached (data-) file.
- **list-missing**: List files on targets that don't exist in the first cached (data-) file.
- **find**: Search for the specified target-file(s) in the cache (data-) file(s). This is based on content, not file-name. The specified target-file(s) are read, a hash-value is calculated for each file, and then the cache is searched for the hash-value.
- **find-hash**: The cache (data-) file(s) are searched for the specified hash-value. The hash-value must be of the same type (MD5 or SHA1) as the cache is using. (This defaults to

MD5).

- **cache-no-external**: Remove all “external” entries; that is, all entries that entered the cache trough the *–in-ext-file* command-line argument. This command has no effect unless you save the modified cache to a data-file.
- **cache-external**: Remove all non-external entries in the in-file cache. This is the opposite of the cache-no-external command.

Command-line options

Option	Effect
-h / --help	Show a brief help-message and exit.
-v / --version	Show the program,m-version and exit.
-q / --quiet	Don't print lots of information to the console.
-d # / --debug-level #	Debug-level (1 – 50). Higher values gives more output.
-f / --force	Overwrite existing files with different content during copy. By default, existing files will be preserved.
-s / --simulate	Just process the command, without actually copying/deleting any files. Can be used with –verbose to verify the effect of a command prior to executing it.
-r / --recurse	Recurse into sub-directories. The default behavior is to only process the targets, without going down to the sub-directories.
-Q / --quote	Enclose names in double quotes (for the commands that list the content of the cache, like “list” and “find”).
--list-hash	Begin the file-listing (for the commands that list the content of the cache, like “list” and “find”) with the file's hash-value.
-S / --suppress-list	Do not output the file-list that the command would else have produced.
--no-remove	Do not verify the existence of each file in a data-file when it is loaded. By default, the presence of each non-external file is verified, and missing files are removed from the cache. This can take considerable time if the data-file contains some ten thousands of files.
-o / --out-file path	XML data-file to store the file-cache.
-i / --in-file path	XML data-file(s) containing existing file-cache(s). You can repeat this option for as many files as you need to load.
--in-ext-file path	XML data-file(s) containing existing file-cache(s) from a 'static' source. Files in this cache will be used to see if a file is known, (but they will not be soft-linked to). The cache-entries from this data-file will be tagged as “external”. You can repeat this option for as many files as you need to load.

--sha1	Use SHA1 as the hash-algorithm. SHA1 produce more uniqueness than MD5, but is considerable slower.
--md5	Use MD5 as the hash-algorithm. This is the default behavior.
--never-delete-path path	Never delete files in this path and below. Use this option to protect parts of your collection where you absolutely don't want files to be deleted, even if they are duplicates. You can repeat this option for as many paths as you need to protect.
--softlink-path path	Replace duplicates in this path and below with soft-links, rather then deleting them. Use this option to protect the file-names in parts of your collection, while saving disk-space by soft-linking any duplicates to one “master”. If you also used the –never-delete-path, dupemgr will soft-link to “master”-files there (or in included cache-file(s)) if they exist there.

Examples

To scan two directories and their sub-directories, and save the cache to a data-file:

dupemgr -r -o data.xml **scan** dir1 dir2

To list the content of a cache-file

dupemgr -i data.-xml --no-remove **list**

To list the content of a cache-file, and the hash-value

dupemgr -i data.-xml --list-hash --no-remove **list**

To copy new files from dir1 and dir2 (including their sub-directories) to a collection.

dupemgr -r **copy** dir1 dir2 mycollection

To build a data-file for a collection, and then copy new (unique) files to some sub-dir in the collection

dupemgr -o cars.xml scan mycar-collection

dupemgr --no-remove -i cars.xml -r **copy** /tmp/cars/firetrucks \

mycar-collection/trucks/firetrucks

To delete duplicates from your collection, except in some folders where you want everything to remain as it is.

dupemgr -r --never-delete-path mycar-collection/precious \

remove-duplicates mycar-collection

To compare your collection (in my-collection.xml) with another collection (friends-collection.xml).

dupemgr -v --no-remove -i my-collection.xml -i friends-collection.xml **list-missing**

To compare your collection (in folder mycar-collection) with another collection (stored in friends-collection.xml).

dupemgr -v -r --no-remove -i friends-collection.xml **list-missing** mycar-collection