***ddeedduupp* v0.1a**

**User Manual**

April 2, 2016

David T. Ashley ([dashley@gmail.com](mailto:dashley@gmail.com))

Table of Contents

[1 License 4](#_Toc28644290)

[2 Introduction and Overview 4](#_Toc28644291)

[2.1 Brief Description of *ddeedduupp* Functionality 4](#_Toc28644292)

[3 Background Information 5](#_Toc28644293)

[3.1 Tree Nomenclature 5](#_Toc28644294)

[3.2 The SHA-512 Cryptographic Hash 6](#_Toc28644295)

[4 Detailed Description of *ddeedduupp* Functionality 6](#_Toc28644296)

[4.1 General Notes and Cautions 6](#_Toc28644297)

[4.2 Rules for Choosing Which Duplicate to Retain 7](#_Toc28644298)

[4.3 Invocation 7](#_Toc28644299)

[4.4 Command-Line Options 8](#_Toc28644300)

[4.5 Subcommand 9](#_Toc28644301)

[4.6 Detailed Description of the Base Directory 10](#_Toc28644302)

[4.7 Detailed Description of the Reference Directory, Reference Set, and Base Set 10](#_Toc28644303)

[4.8 Output, Error Termination, and Exit Code 10](#_Toc28644304)

[5 Invocation Example 10](#_Toc28644305)

[6 Technical Description and Internal Operation of *ddeedduupp* 11](#_Toc28644306)

[6.1 Heap Allocation 11](#_Toc28644307)

[6.2 RAM Data Structures 11](#_Toc28644308)

[6.3 Approximate Limits on File Set 11](#_Toc28644309)

[6.4 Approximate Initialization Time 11](#_Toc28644310)

[6.5 Method of Moving Files from *tgt* to *dup* 11](#_Toc28644311)

[6.6 Use on a Changing File Set 11](#_Toc28644312)

[6.7 Use on a Networked Drive 11](#_Toc28644313)

[6.8 Redistributing *ddeedduupp* 11](#_Toc28644314)

[6.9 Rebuilding *ddeedduupp* from Source Code 11](#_Toc28644315)

[6.10 Modifying *ddeedduupp* 11](#_Toc28644316)

[*6.11* Redistributing modified *ddeedduupp* 11](#_Toc28644317)

[7 References 12](#_Toc28644318)

[8 Index 13](#_Toc28644319)

# License

*ddeedduupp* (source files, binaries, documentation, and all other files) is made available under the *MIT License*, reproduced below.

Copyright 2020 David T. Ashley  
  
Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:  
  
The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.  
  
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

# Introduction and Overview

## Brief Description of *ddeedduupp* Functionality

*ddeedduupp* is a simple *Windows* and *\*nix* tool for identifying and selectively deleting duplicate files. It was originally devised by the author to identify and eliminate duplicate photos and documents.

The general characteristics of *ddeedduupp* are:

* On invocation, it calculates the SHA-512 cryptographic hash of every file in the set of files in which duplicates are to be identified.[[1]](#footnote-1)
* The SHA-512 cryptographic hash is used to quickly identify duplicates.[[2]](#footnote-2)
* *ddeedduupp* is written in ANSI C and should compile on virtually any platform. Full source code and a *Windows* binary are provided. For non-Windows systems, the end user will have to rebuild the program from source code.

The limitations and shortcomings of *ddeedduupp* are:

* The program performs a narrow range of de-duplication functions (one size likely does not fit all).
* On each invocation of the program, the SHA-512 hash of every file is calculated. This is potentially a time-consuming operation.
* The program supports only ASCII characters in file names, and does not support Unicode. It is not known how the program will behave on operating systems where directory and file names may contain non-ASCII characters.
* *ddeedduupp* requires that the pool of files in which duplicates are to be identified and removed be staged in a single directory on a single disk volume.
* *ddeedduupp* is single-threaded. (A substantial performance increase during the initial SHA-512 calculation might be possible on some platforms if a multi-threaded model were adopted, but this has not been evaluated.)
* Unlike other more sophisticated de-duplication programs, *ddeedduupp* has no notion of files which are *nearly* identical. *ddeedduupp* will identify and remove only files which are *exactly* identical.
* The program forms a tree data structure in RAM to represent the directory and file hierarchy of the set of directories and files being examined for duplicates, as well as to contain the SHA-512 cryptographic hashes of files. Approximately 1 gigabyte of RAM is necessary for every 5 million files to be examined. There may be uncommon cases (a RAM-constrained machine combined with a large number of small files to be examined) that will cause a computer to run out of RAM.

# Background Information

## Tree Nomenclature

On invocation, the first step performed by *ddeedduupp* is to identify the files and directories in the base directory. Internally, *ddeedduupp* forms a tree data structure in RAM.

The notion of a tree to represent a computer file system is probably familiar to most computer users. Only directories (not files) may have children, and the children of directories may be either directories or files.



Figure : Example Tree (from Wikipedia article https://en.wikipedia.org/wiki/Tree\_structure)

Figure 1 (copied from [this](https://en.wikipedia.org/wiki/Tree_structure) Wikipedia article) is a prototypical tree to illustrate the nomenclature for the relationships between file system objects as used in this document and in the output from *ddeedduupp*.

* *Node*: Each filesystem object (directory, file) is a node. *Encyclopaedia*, *Science*, *Culture*, *Art*, and *Craft* are nodes.
* *Child*, *Children*: Only a directory may have children. *Science* is a child of *Encyclopaedia*, and *Craft* is a child of *Culture*.
* *Parent*: The directory containing the directory or file. *Culture* is a parent of *Art*, and *Encyclopaedia* is a parent of *Science*.
* *Root*: A node with no parent. Only *Encyclopaedia* is the root.
* *Leaf*, *Leaf Node*: A node with no children. Every file is a leaf node (because files may not have children). Only empty directories are leaf nodes.
* *Sibling*: A node with the same parent. *Art* and *Craft* are siblings.
* *Uncle*: A sibling of a node’s parent. *Science* is an uncle of *Art*.
* *Ancestor*: A node that can be reached by traveling up the tree. *Craft* has the ancestors *Culture* and *Encyclopaedia*.
* *Descendant*: A node that can be reached by traveling down the tree. *Encyclopaedia* has all other nodes as its descendants. *Culture* has *Art* and *Craft* as its descendants.

## The SHA-512 Cryptographic Hash

The SHA-512 cryptographic hash is part of the SHA-2 [] suite, and is an NIST standard. Numerous web pages exist describing the algorithm and providing reference code.

*ddeedduupp* uses the SHA-512 cryptographic hash to identify likely duplicate files. Although the SHA‑512 cryptographic hash is quite reliable to determine whether two files are identical, *ddeedduupp* compares files before deleting to be certain that the files are identical, and will not perform a deletion if two files have the same SHA-512 cryptographic hash but are in fact non-identical files.

How likely is it that two files that are not identical have the same SHA-512 cryptographic hash?

* No SHA‑512 hash collisions have been found to date.
* 2512 is approximately 10154. (For comparison, the number of atoms in the observable universe is estimated to be 1082, a much smaller number. It is *extremely* unlikely that two non-identical files would have the same SHA-512 hash; but *ddeedduupp* does protect for this possibility.)

# Detailed Description of *ddeedduupp* Functionality

## General Notes and Cautions

1. On \*nix systems, *ddeedduupp* follows both hard and soft links.
2. On Windows systems, *ddeedduupp* follows soft links.
3. *ddeedduupp* makes the assumption that it is the only program modifying the filesystem of the target directory. Any modification of the filesystem of the target directory while *ddeedduupp* is running may have unpredictable results, including loss of data.
4. *ddeedduupp* commands that remove duplicates are not undoable. The duplicate files are deleted permanently.
5. *ddeedduupp* will *not* remove the last file of a set of duplicates (such an operation must be done manually, using some method other than *ddeedduupp*).

## Rules for Choosing Which Duplicate to Retain

In contexts where *ddeedduupp* is to retain only one of a set of duplicates and where the choice of which duplicate to retain is not specified by other rules, the program chooses the duplicate to retain using these rules:

1. The duplicate highest in the directory tree is chosen.
2. If the previous rule results in a tie, the file in the directory whose components are first in sort order is chosen.
3. If the previous two rules result in a tie, the file name first in sort order is chosen.

Note that these rules tend to:

* Dispose of duplicates lower in the directory hierarchy.
* Dispose of duplicates whose ancestors are later in the sort order.

This typical effect of the rules is intentional. Duplication tends to involve entire directories of duplicate files, and retaining the duplicates that are higher in the directory tree and first in path sort order tends to create the result desired by most human users.

## Invocation

Ddeedduupp is invoked from the command prompt (Windows) or from a shell (Unix) as follows:

ddeedduupp [options] subcommand base\_directory [reference\_directory]

The square brackets (“[options]”, “[reference\_directory]”) are used to indicate that an entity is optional. The square brackets should not be typed on the command line.

The options (§) are always optional.

The reference\_directory may or may not be required, depending on the subcommand (§).

The components of the command-line/shell invocation form above are explained further in subsequent sections.

1. ddeedduupp  
   The program name.
2. [options]  
   Options that may modify how the program behaves (§).
3. subcommand  
   Describes the program’s major mode of operation (§).
4. base\_directory  
   The directory in which the program scans for duplicates (§).
5. [reference\_directory]  
   The reference directory, which must be an improper subdirectory of the base directory, potentially along with some command-line options (§4.4), serves to partition the file system objects into the base set and the reference set. This is described more fully in §.

## Command-Line Options

Table 1 lists the options that can be used when invoking *ddeedduupp*. Not all options are applicable to all subcommands.

Table 1: ddeedduupp Command-Line Options

|  |  |  |
| --- | --- | --- |
| **Command Option Number** | **Command** | **Action** |
| 1 | *-dryrun* | Causes the command to be executed with no effect on the file system. This provides a preview of what the program would do if invoked without the *‑dryrun* option. |
| 2 | *-notouch\_rd* | Prohibits the program from deleting files within the reference directory itself. (However, this does not prohibit deletion of files in ancestors or descendants of the reference directory.) |
| 3 | *-notouch\_rd\_descendants* | Prohibits the program from deleting files within the descendants of the reference directory. (However, this does not prohibit deletion of files in the reference directory or its ancestors.) |
| 4 | *-refset\_include\_rd* | Causes the reference directory to be included in the current reference set.  This option is at this time never necessary, because the reference directory is always included in the reference set by default. |
| 5 | *-refset\_exclude\_rd* | Causes the reference directory to be excluded from the reference set. |
| 6 | *‑refset\_include\_rd\_descendants* | Causes the descendants of the reference directory to be included in the reference set.  This option is at this time never necessary, because the descendants of the reference directory are always included in the reference set by default. |
| 7 | *‑refset\_exclude\_rd\_descendants* | Causes the descendants of the reference directory to be excluded from the reference set. |
| 8 | *‑refset\_include\_rd\_ancestors* | Causes the ancestors of the reference directory to be included in the reference set. |
| 9 | *-refset\_exclude\_rd\_ancestors* | Causes the ancestors of the current reference directory to be excluded from the current reference set.  This option is at this time never necessary, because the ancestors of the reference directory are always excluded from the reference set by default. |
| 10 | *-refset\_include\_rd\_siblings* | Causes the siblings of the reference directory to be included in the reference set. |
| 11 | *-refset\_emptydir\_remove* | After duplicate files are removed, any empty directories (directories that contain no directories or files) are removed.  Directories are analyzed and removed from the bottom of the tree up, so empty directories that contain only empty directories will also be removed. |

## Subcommand

describes the *ddeedduup* subcommands.

Table : Subcommands

|  |  |  |  |
| --- | --- | --- | --- |
| **Subcommand Number** | **Subcommand** | ***reference\_directory* Argument Supported** | **Description** |
|  | *noop* | Yes | Performs a full analysis, but performs no deduplication operations. This command can be used for two purposes:   * To get information about the base directory and reference directory (and descendants), including information about number and identity of duplicates. * With the *‑refset\_emptydir\_remove* option, to delete empty directories but perform no other operations. |
|  | *dd\_bs* | No | All duplicate files within the base set are removed. |
|  | *dd\_rs\_pri* | Yes | All files in the base set that are duplicates of file(s) in the reference set are deleted.  No file within the reference set is deleted. |
|  | *dd\_rs\_sec* | Yes | All files in the reference set that are duplicates of files(s) in the base set are deleted.  No file within the base set is deleted. |

## Detailed Description of the Base Directory

The base directory (see item 4, §4.3) is the directory in which *ddeedduupp* operates to identify and remove duplicates. *ddeedduupp* will not open, read, or delete a file or directory not in the base directory and its descendants.

When *ddeedduupp* is initially invoked, the program calculates the SHA-512 cryptographic hash of every file in the base directory or its descendants.

## Detailed Description of the Reference Directory, Reference Set, and Base Set

The reference directory (see item , §4.3) must be specified as an improper descendant of the base directory.

In specifying the base directory and reference directory on the command line:

* No referential nomenclature for paths (“.”, “..”) is allowed by *ddeedduupp*.
* *ddeedduupp* must be able to verify that the reference directory is an improper descendant of the base directory. This means that the string representing the base directory must appear at the start of the specified reference directory.

Once the base directory and reference directory are known, ddeedduupp conceptually calculates the reference set and base set as follows:

* The *reference set* is the reference directory, combined with all its descendants, but adjusted by the command-line options that modify the reference set.
* The *base set* is the base directory and all its descendants, with the elements of the reference set removed.

Note that:

* The reference set and base set have no intersection.
* The union of the reference set and the base set is the base directory and all its descendants.
* It is possible for the reference set or base set to be empty.

## Output, Error Termination, and Exit Code

The program writes informative and error output to *stdout* and error output to *stderr*.

It is safe to redirect *stdout* to a file but leave *stderr* directed to the console: any error messages will be written to both *stdout* and *stderr*, so that the lack of console output would mean that the program completed successfully.

Nearly all errors and suspicious situations are treated as an error and will result in program termination. If the program completes without errors, the exit code will be 0; an error termination gives an exit code of 4.

# Invocation Example

# Technical Description and Internal Operation of *ddeedduupp*

## Heap Allocation

## RAM Data Structures

## Approximate Limits on File Set

## Approximate Initialization Time

## Method of Moving Files from *tgt* to *dup*

## Use on a Changing File Set

## Use on a Networked Drive

## Redistributing *ddeedduupp*

## Rebuilding *ddeedduupp* from Source Code

## Modifying *ddeedduupp*

## Redistributing modified *ddeedduupp*

# References

[1] SHA-2, <https://en.wikipedia.org/wiki/SHA-2>

# Index

ancestor

defined, 6

base directory, 5, 10

defined, 10

base set

defined, 10

child

defined, 6

descendant

defined, 6

leaf

defined, 6

license, 4

MIT license, 4

node

defined, 6

parent

defined, 6

reference directory, 10

reference set

defined, 10

root

defined, 6

SHA-512 cryptographic hash function, 6

sibling

defined, 6

tree, 5

uncle

defined, 6

Unicode, 5

1. Calculating the SHA-512 cryptographic hash of every file allows *ddeedduupp* to identify duplicates very quickly once the initial calculation is done. [↑](#footnote-ref-1)
2. Please see §. [↑](#footnote-ref-2)