

QuickHash GUI

V2.8.1

*The Free, Graphical, Cross-Platform,
Data Hashing Tool*

User Manual : v1.0.10 (May 2017)

Copyright (c) 2011-2017 Ted Smith

All rights reserved

<http://www.quickhash-gui.org>

(and previously <https://sourceforge.net/projects/quickhash>)

1.0 Pre-ambble

This manual is designed as a user-aid only. It is not an authority on the subject matter of hashing algorithms, filesystems or anything else, neither in part or in full. The software comes without any warranty. Use it at your own risk and if you are unsure of any results, please consider cross referencing your findings with other software. There are many free and commercial data hashing tools available to cross-reference your findings.

Constructive feedback is encouraged and welcomed but complaints will not be tolerated. If the user is unhappy with the software, s/he is encouraged to use something else.

1.1 License Agreement

There are no restrictions. Users can run it on as many computers as they wish, as many times as they wish, for as long as they wish. There are no dongles, no DLL's, no installation wizards or license files – just click and go. All that is asked of the users is that they share their thoughts and help contribute ideas back to the developer (tedsmith28@yahoo.co.uk).

Supported Platforms: Tested on Microsoft Windows (XP, Vista, 7, 10), Linux Mint 16 and 17, Apple Mac Yosemite.

QuickHash GUI is made available under the GPL license as follows.

QuickHash GUI 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. See a copy of the GNU General Public License at [<http://www.gnu.org/licenses/>](http://www.gnu.org/licenses/).

1.2 Acknowledgements

QuickHash is written and compiled using the [Lazarus Project IDE](#) and the [Freepascal language](#) using the Freepascal Compiler. Thanks are therefore expressed to the developers and contributors of both Lazarus and FPC, without which QuickHash would not have existed.

A customised version of the Freepascal MD5 and SHA-1 libraries were used for MD5 and SHA-1 and the DCPCrypt library was used for SHA256 and SHA512 hashing algorithms up until v2.8.0 of QuickHash. With v2.8.0 however, the libraries were both discarded in favour of HashLib4Pascal (<https://github.com/Xor-el/HashLib4Pascal>) by Ugochukwu Mmaduekwe Stanley (aka Xor-el). It is more actively maintained, did not require 64-bit tweaking to allow hashing of large data, is Freepascal v3.0 compliant, avoided the use of a separate customised MD5 and SHA-1 libraries, and includes a full range of hash algorithms which will allow easier implementation of said algorithms by Quickhash in the future. So as of v2.8.0, the library is now used for all of the mainstream hashing algorithms used by QuickHash (i.e. MD5, SHA-1, SHA256, SHA512) plus the addition of the faster 'xxHash' algorithm which is newer and pretty fast! Significant thanks and appreciation are expressed to Mr Stanley for the open-source library. Further thanks to the author of xxHash for making that open-source as well. Note that xxHash is licensed under BSD License and Haslib4Pascal is licensed under the MIT License.

1.3 Development Team

Lead Developer:	Ted Smith
Contact:	https://sourceforge.net/projects/quickhash – see support pages and ticket system for feature requests and suggestions
GitHub Repository	https://github.com/tedsmith/quickhash/releases
Supporting Developer :	Dareal Shinji
GitHub Page	https://github.com/darealshinji

2.0 Introduction

QuickHash is quite simply a free, open source, cross platform, fast and easy-to-use data hashing tool for Microsoft Windows, Desktop GNU/Linux distributions like Mint and Ubuntu, and Apple Mac Yosemite and above.

It does not require installation and can simply be executed from a USB drive or other removable device (though on Linux, the software has to have executable permissions assigned and the external device would require a filesystem that can store executable permissions – so EXT4 or something and not FAT32).

As of v2.8.0 four ‘mainstream’ hashing algorithms are available – MD5, SHA-1, SHA256 and SHA512 and in addition there is now a fifth algorithm to choose; xxHash (when using the 32-bit version of Quickhash, only xxHash32 will be available, and visa versa for 64 bit), which is gaining much credibility and implementation worldwide for its speeds. It’s much newer than the other algorithms though and, in science, these things take a while to become widely adopted.

The interface is deliberately simple and comes as a tabbed system – each tab for a different type of data.

Text | File | FileS | Copy | Compare Two Files | Compare Directories | Disks

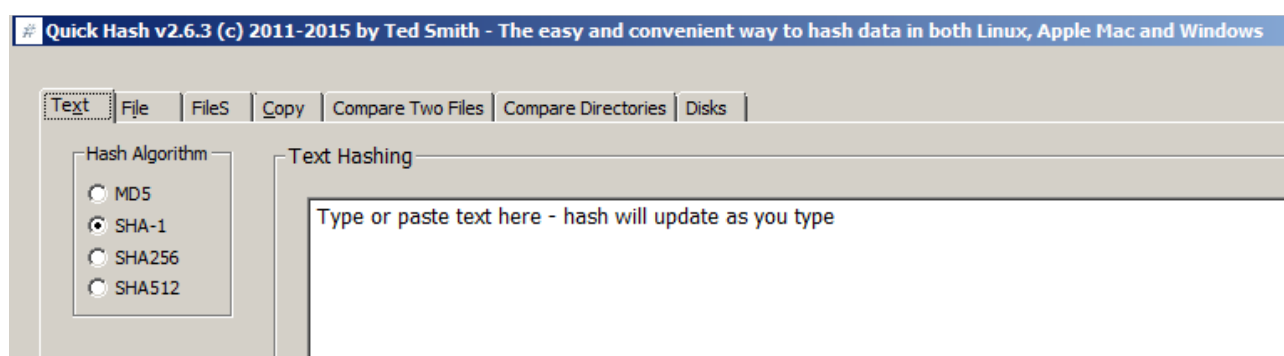


Illustration 1: The tabbed interface of Quickhash

2.1 What is a hash?

The simplest explanation is that it is like a unique fingerprint of digital data. There are many common hash algorithms, but QuickHash is coded to utilise four of the common mainstream ones :

[MD5](#), [SHA-1](#), [SHA256](#) and [SHA512](#). Very basically, if you compute the MD5 value of your typed name, the result is theoretically unique to one in 3.4028...E38, which, put more simply is one in 340 billion, billion, billion, billion (1 undecillion) meaning that the chances of any other digital data other than that string of characters generating the same hash are infinitely unlikely (engineered collisions aside).

The research surrounding [hash collisions](#) is duly noted and the reader can find more information about that in publications and articles on the Internet or academic papers, if it is a concern to his\her work area.

[XxHash](#), by Yann Collet (<https://github.com/Cyan4973>) is “*an Extremely fast Hash algorithm, running at RAM speed limits. It successfully completes the [SMHasher](#) test suite which evaluates collision, dispersion and randomness qualities of hash functions. Code is highly portable, and hashes are identical on all platforms (little / big endian)*”. Its implementation into QuickHash typically shows results of about 20%-50% speed increases over SHA-1, depending on the source data. So if a file takes 30 seconds on a given PC using SHA-1, it will take about 20 seconds using xxHash, give or take.

Published benchmark tests report figures around the following :

Name	Speed	Quality	Author
xxHash	5.4 GB/s	10	Y.C.
MurmurHash 3a	2.7 GB/s	10	Austin Appleby
SBox	1.4 GB/s	9	Bret Mulvey
Lookup3	1.2 GB/s	9	Bob Jenkins
CityHash64	1.05 GB/s	10	Pike & Alakuijala
FNV	0.55 GB/s	5	Fowler, Noll, Vo
CRC32	0.43 GB/s	9	
MD5-32	0.33 GB/s	10	Ronald L.Rivest
SHA1-32	0.28 GB/s	10	

At the time of writing (Feb 2017) there is no official documentation other than the source code to explain xxHash and its ways of working. However, some work has been done by Stephan Brumme to illustrate it, which is useful and may well form the basis of a Wikipedia entry one day.
<http://create.stephan-brumme.com/xxhash/>

It would very beneficial to the world to see xxHash put through its paces in a test environment so

that it can be held with same regard as SHA-1 and SHA256. With such significant apparent speeds, it would be tragic for it not be more widely used. In science circles, the old timers like SHA-1 will take some beating with regard to reliability, but 0.28 Gb/s vs 5.4Gb/s....that alone has to be worth some serious investment of research. That said, QuickHash itself does not exhibit those kind of speeds with the algorithm but that is perhaps more likely due to my developing skills more than the fault of the algorithm.

QuickHash adopts [Merkle–Damgård constructions](#) which theoretically enables a method of building collision-resistant cryptographic hash functions. More can be read about that online.

2.2 Cross Platform

QuickHash was originally designed for Linux to enable less advanced Linux users to easily and quickly generate a list of hash values for files using a simple graphical interface without having to resort to command line tools like SHA1SUM. It was specifically designed to run with live boot CD's like DEFT, CAINE, PALLADIN, HELIX, PARROT SECURITY and others. However, over time, the hashing functionality has improved and indeed now become faster at generating hash values than many other tools – both free and commercial ones. So it is now not only a tool that enables the quick selection of files to hash, but it is also a tool that does compute the hashes quickly. So the name “QuickHash” is really rather fitting.

It is pre-built into the [DEFT](#) and [CAINE](#) Linux CD's and hopefully soon it will be included with [Parrot Security](#) and possibly soon even [Debian](#) (not confirmed yet), so by downloading them you will have the power of QuickHash built in to your live CD instance without having to use it separately. Though, note, the releases of QuickHash are generally every couple of months whereas bootable distributions generally have a longer cycle refresh period.

In addition to the Linux version, due to demand from Windows users, a Microsoft Windows compatible version was made and sports some features that are necessary in a Windows environment but not necessary in a Linux one. For example, it is possible to compute the hash of a physical disk using QuickHash in Linux by running QuickHash as root, clicking the “File” tab and navigating to `/dev/sdX` or `/dev/sdXX`. Or, as of v2.7.0, you can use the ‘Disks’ tab the same as Windows users. Apple Max OSX cannot currently utilise the ‘Disk’ tab, but may hash disks using the ‘File’ tab in the same way as is possible with Linux.

An Apple Mac version was then developed and first appeared with version 2.5.3 in 2014. It functions in a similar vein to the Linux version. The Mac version was compiled on the Yosemite operating system, so older Mac's with Power PC systems may not run QuickHash successfully – not tested.

3.0 Interface

Tabs Explained – Quick Summary

Text : For hashing chunks of text like paragraphs from a file, a name, a character string, a list of values (to be hashed line by line) or key data that can be copied from somewhere else to QuickHash

File : For selecting and then hashing one individual file (and disks in Linux\Apple Mac).

FileS : For hashing multiple files in a directory (aka 'folder') recursively.

Copy : Copy & Paste essentially, but with some added data integrity. Designed to enable a user to copy files from one place to another but have the copy process verified and supported by hash values and a retained log of the original date and time attributes.

Compare Two Files : Simply choose two files in two different locations and have the hashes of both compared automatically.

Compare Directories : For comparing the file content of one directory to another to see if all the files inside each match, or not.

Disks : Easily hash entire physical disks and logical volumes (as of v2.4.0 upwards and for both Windows and Linux since v2.7.0).

3.1 Tabs Explained – Detailed Explanation

3.1.1 Text : For hashing chunks of text like paragraphs or key data that can be copied from

somewhere to QuickHash. You can also type in the text box and QuickHash will dynamically recompute the hash value as you type. The user can also switch the generated hash to one computed by another algorithm simply by clicking one of the other three radio buttons.

As of v2.6.2, an “Expected Hash Value” field allows the user to paste an existing hash value (perhaps computed by another tool) and QuickHash will compare the generated hash of the inputted text segment against the one supplied by the user. An alert will be displayed if the hashes do not match. To cancel the comparison, replace the hash value with three dots ('...').

As of v2.6.5, the functionality was added to break the text area down line-by-line. This was at user request because it turned out that Google Adwords and perhaps similar services require customers to provide their e-mail address lists as lowercase SHA256 hash values. So now, the user can paste the list of thousands of addresses (up to 2Gb of text) into QuickHash and if the user then clicks the “Hash Line-By-Line” button, they will get a comma separated output file containing all the hash values for each e-mail address in seconds. **Be aware that lists of e-mail addresses in capital letters will generate different output to an e-mail list of addresses in lower case! Me@Me.com is different to me@me.com. You should prepare your list in advance. QuickHash will not conduct case adjustments of your text data.**

In addition, a second button allows a text file to be opened and then each line of that file to be hashed, line by line. And as of v2.6.7, there is a toggle tickbox that allows the user to include or exclude the originating text data in the output file. Useful for Google Adwords where it expects just a file containing hashes of e-mail addresses, and not the e-mail addresses themselves that were used to generate the values in the first place. But there will be other occasions where the user might want to see the text that was hashed as well as the has in the output. That is what this is there for.

ASCII as well as Unicode text should work OK.

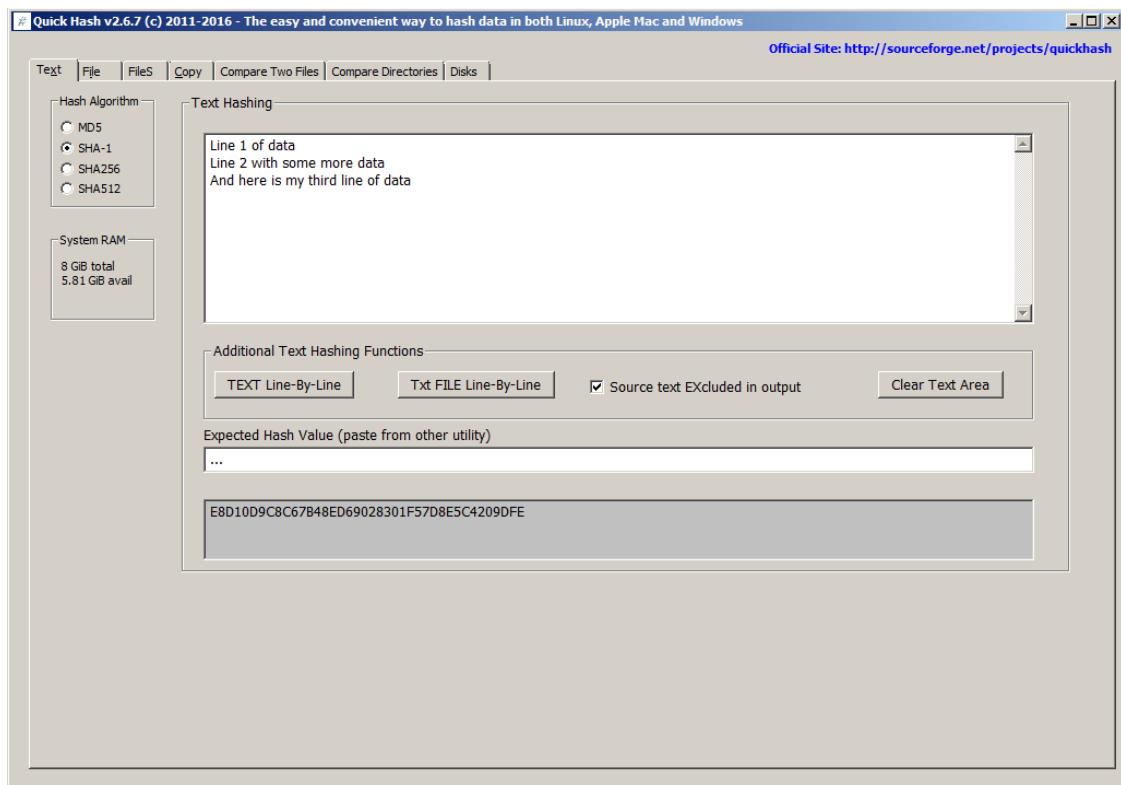


Illustration 2: Hashes values for the sum of the text are dynamically recomputed as the user types, or, using the “Hash Line-by-Line” button enables the entire list to be hashed line by line

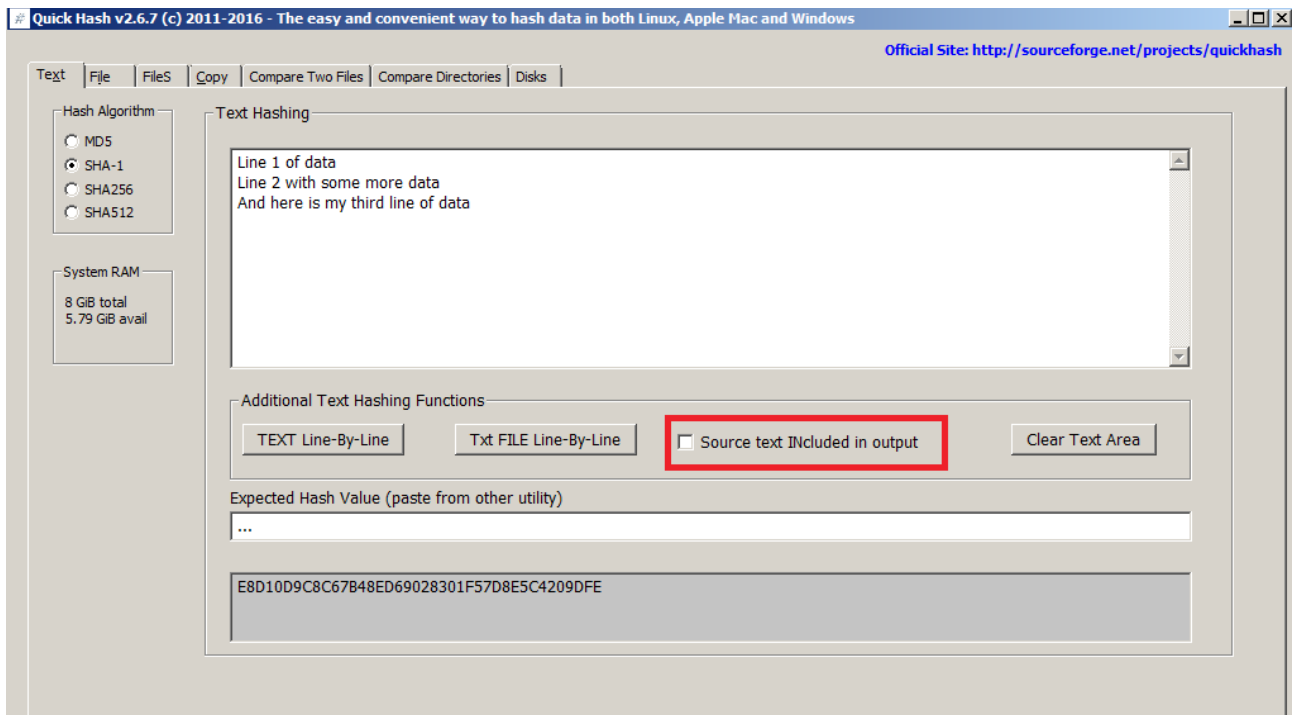


Illustration 3: The new text hashing functions available since v2.6.7

3.2.2 File : For selecting and then hashing one individual file. Simply click the 'Select File' button, navigate to the file (or drag and drop a file onto the program), and the hash will be computed. There are no size limits (since v2.1 at least – earlier versions struggled only if the file was a) larger than 4Gb and b) being hashed with SHA256 or SHA512 due to a 32-bit limitation of the original DCPCrypt library) other than those that are imposed by the filesystem storing the file which QuickHash has no control over, of course. So there is no need to worry about 4Gb boundaries and so on.

As of v2.6.2, there is also the same “Expected Hash Value” field that appears in the 'Text' tab, that again allows the user to paste an existing hash value (perhaps computed by another tool) and QuickHash will compare the generated hash of the file against the one supplied by the user. Case sensitivity is dealt with by QuickHash, so don't worry about converting your expected hash from either lower to upper or upper to lower case – QuickHash will deal with that for you. An alert will be displayed if the hashes do not match though. To cancel the comparison, replace the hash value with three dots ('...'). Also be sure to remove all three dots when pasting your 'expected hash value'!

Unicode characters in the filename or file content is also dealt with automatically.

As with text, the resulting hash can be re-computed simply by choosing a different algorithm in the radio box selection. Larger files will display a message saying “Recomputing hash”.

This functionality is useful, for example, when you have written a document of some kind and finished it and you want to send it to someone and be certain the file they receive from you is the same as when you finished and sent it. In such a case, hash the file before you send it, attach it to your e-mail along with a copy of the computed hash, and then tell the recipient to use QuickHash to recompute the same hash on arrival and check the computed value against what you put in your e-mail.



Illustration 4: The 'File' tab showing a computed hash of a file

With v2.8.1, a progress indicator was added to this tab because users kept reporting that the program “had crashed” whilst hashing large files, when it merely could not be interrupted. With v2.8.1, the interface will update every few seconds in this tab and give an idea of how many Mb have been read so far.

On Linux systems, everything is a file so this can include physical disks (e.g. /dev/sda) or logical drives (/dev/sda1) if QuickHash is run with root access. Disk hashing is also available for Windows – see Disk Hashing, below.

With regard to forensic images; it is sometimes useful to hash the individual chunks of a forensic image if forensic software states a problem with an image, to try and diagnose if one particular chunk has not been moved or copied correctly from a master copy. However, do not confuse this functionality with the total **inability** of QuickHash **to compute the internally computed hash of the data inside the image**. At the moment, QuickHash can not be used to verify E01 image sets, or AFF, or even DD sequences. In other words, if the user navigates to the first file of a chunked image set (such as a chunked dd image or an E01 image) the **resulting hash will be that of the chosen image chunk only**, not of the acquired data that sits inside the entire forensic image spanning multiple chunks. To verify an actual complete forensic image, please use forensic software like X-Ways Forensics, EnCase, FTK, ewfacquire, Sleuthkit and so on. In the future, the LibEWF library, by Joachim Metz may be incorporated to allow hashing of the data inside E01 images. But as there are so many other tools for opening and reading data inside E01 sets, this remains merely an open suggestion.

3.2.3 FileS : For hashing multiple files in a directory (aka 'folder') recursively. Put simply, choose a directory and QuickHash will find all the files below that directory and inside its child directories and compute the hashes for all of the files, outputting the results to screen. If you have many files, choosing 'xxHash' as your chosen hash algorithm will be considerably quicker than any of the others.

There are several options in this tab :

1. Saved to a CSV log file?
2. Saved to an HTML log file?
3. Ignoring sub-directories
4. Flag Duplicates?
5. Hidden folders too?
6. Choose file types?

Options 1 -6 require the user to tick the box prior to starting the scan.

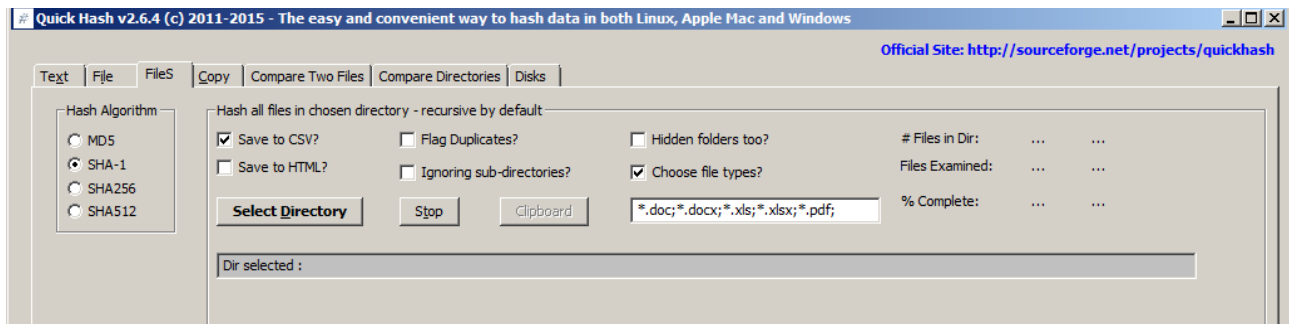


Illustration 5: The various options of the 'FileS' tab

The “Save to CSV?” and “Save to HTML?” options enable the content of the completed display grid to be saved permanently with a view to the user then being able to use the power of spreadsheet software to sort, search and filter the data (if saved as CSV), or to present the data more neatly by e-mail or using a web browser (if saved as HTML) like Mozilla Firefox, Google Chrome and so on.

The “Ignoring sub-directories?” enables the user to compute hashes of files in the root of the chosen directory but not any of the files located in the child sub-directories that may be below that chosen root directory.

The “Flag Duplicates?” option simply populates an extra column with information about any file that is found to be a duplicate of another in the chosen directory structure. After hashing the files, if this tick box is enabled AND duplicate files were found, QuickHash will ask the user if he wishes to delete them by way of “Yes, No, Cancel”. Choosing Yes will delete the duplicate files. If in doubt, choose “No”! The user may prefer to run the hashing once to see what files are duplicated first and whether or not they can safely be deleted having examined the grid. If so, run the scan again and choose “Yes” when asked if the duplicates are to be deleted.

The “Hidden Folders too?” option requires some detailed explanation. On Windows, hidden files will be found and hashed by default in the 'Copy' tab of Quickhash *but only if they live in unhidden folders*.

Files that live in hidden folders, regardless of whether the files are themselves hidden or unhidden, will not be found unless this option is ticked. By ticking the box however, all files, hidden or unhidden, that live in folders that are either hidden or unhidden, will be found. The screenshots

below demonstrate.

Name	Folder path ^
Hidden	C:\Temp
Unhidden	C:\Temp
HiddenFileInHiddenFolder.txt	C:\Temp\Hidden
NothiddenFileInHiddenFolder.txt	C:\Temp\Hidden
HiddenFileInUnHiddenFolder.txt	C:\Temp\Unhidden
NothiddenFileInUnHiddenFolder.txt	C:\Temp\Unhidden

Illustration 6: Files with various filesystem attributes

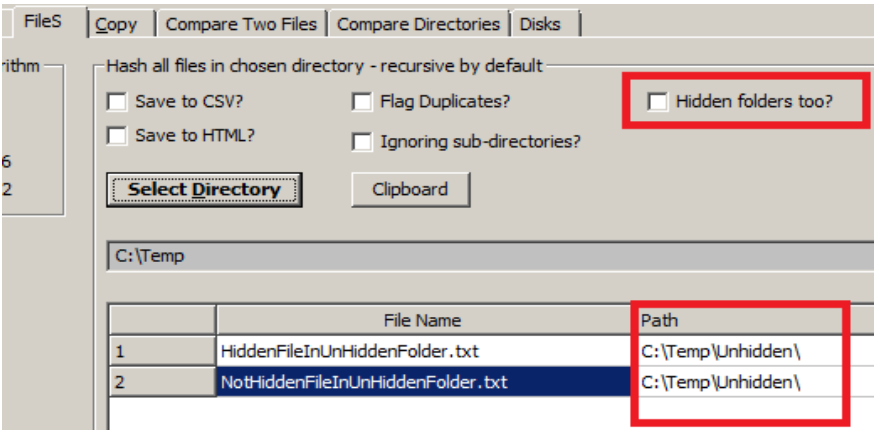


Illustration 7: Quickhash will ignore files in hidden folders if asked

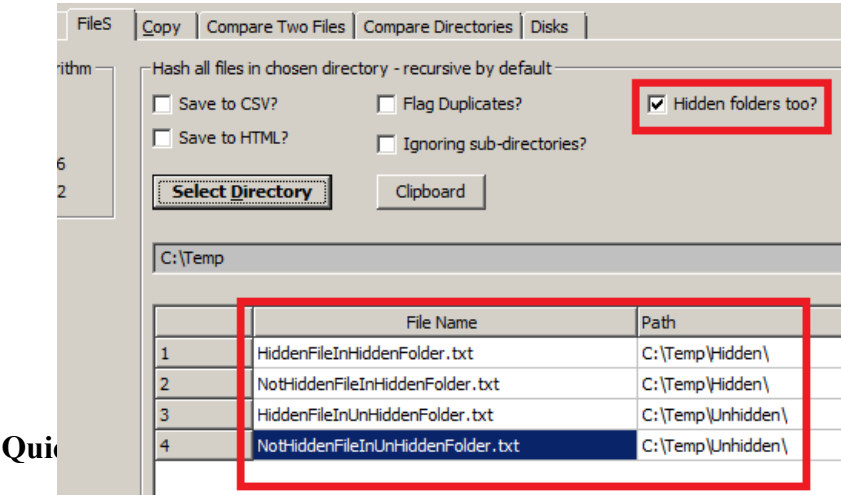


Illustration 8: Quickhash will examine files in hidden folders, if asked (v2.6.3 screenshot)

The “Choose file types?” feature was added to the “Files” tab in v2.6.4 by user request, meaning the user can recursively hash the content of an entire folder and its sub-folders but only files that have the inputted extension(s) will be analysed. Each value is to be separated by a semi-colon (;). Note that file header signature analysis is not conducted.

Since v2.6.4, QuickHash will also find files and folders that exceed the MAX_PATH value of 260 characters imposed by MS Windows. The underlying filesystems of most operating systems, including NTFS, support filenames of much longer than 260 characters, but Windows itself does not currently, even though the underlying filesystem does. It is possible, though, for some software to deliberately exceed this limit, meaning files may exist that the user cannot generally access. With v2.6.4, those files will be found and hashed up to a length of 32K characters for Windows and 4K for Linux. Note that between v2.6.4 and 2.6.8, this functionality was a bit buggy. Hopefully, as of v2.6.9, these bugs have mostly been ironed out for both source and destination locations with long paths, regardless of whether they are drive letter paths or UNC network paths.

The results can also be copied to the clipboard from the display grid by clicking the “Clipboard results” button, which will be 'click-able' once a scan has finished; disabled until then.

The hashes cannot be dynamically re-computed in this tab, unlike in the first two tabs.

Drag and drop of directories is not possible in this tab either.

3.2.4 Copy : The Copy tab allows the user to select a directory containing files he wishes to copy and a destination directory for where he wishes to copy the files to. Both source and destination directories can be a local directory or a mapped network drive or a UNC unmapped network address.

Having selected both directories, upon clicking 'Go', QuickHash will hash the files in the source directory, then copy them to the destination directory (whilst concurrently reconstructing the directory path of the original files, by default) where it then re-hashes the files to check they match the hash values of the original computations. It is, in essence, a “forensic copy and paste”. As with the “FileS” tab, there are options to save the results to CSV or HTML and sub-directories of the source directory can be ignored. Also note that there is an option for the user to not have the source

directory structure rebuilt in the destination by ticking the “Don't rebuild path?” option.

There are five other options that are unique to this tab that require explanation.

1. Just LIST Directories?
2. Just LIST sub-directories and files?
3. Choose file types?
4. Don't rebuild path?
5. Copy hidden files?

Options 1 and 2 are simply for listing (thus 'LIST' is capitalised) either the directory structure of the chosen source directory (excluding files) or for listing the names of directories AND files of the chosen source directory but without actually hashing any of the files. This is useful when a user needs a textual representation of a directory structure to paste into some other software or a report.

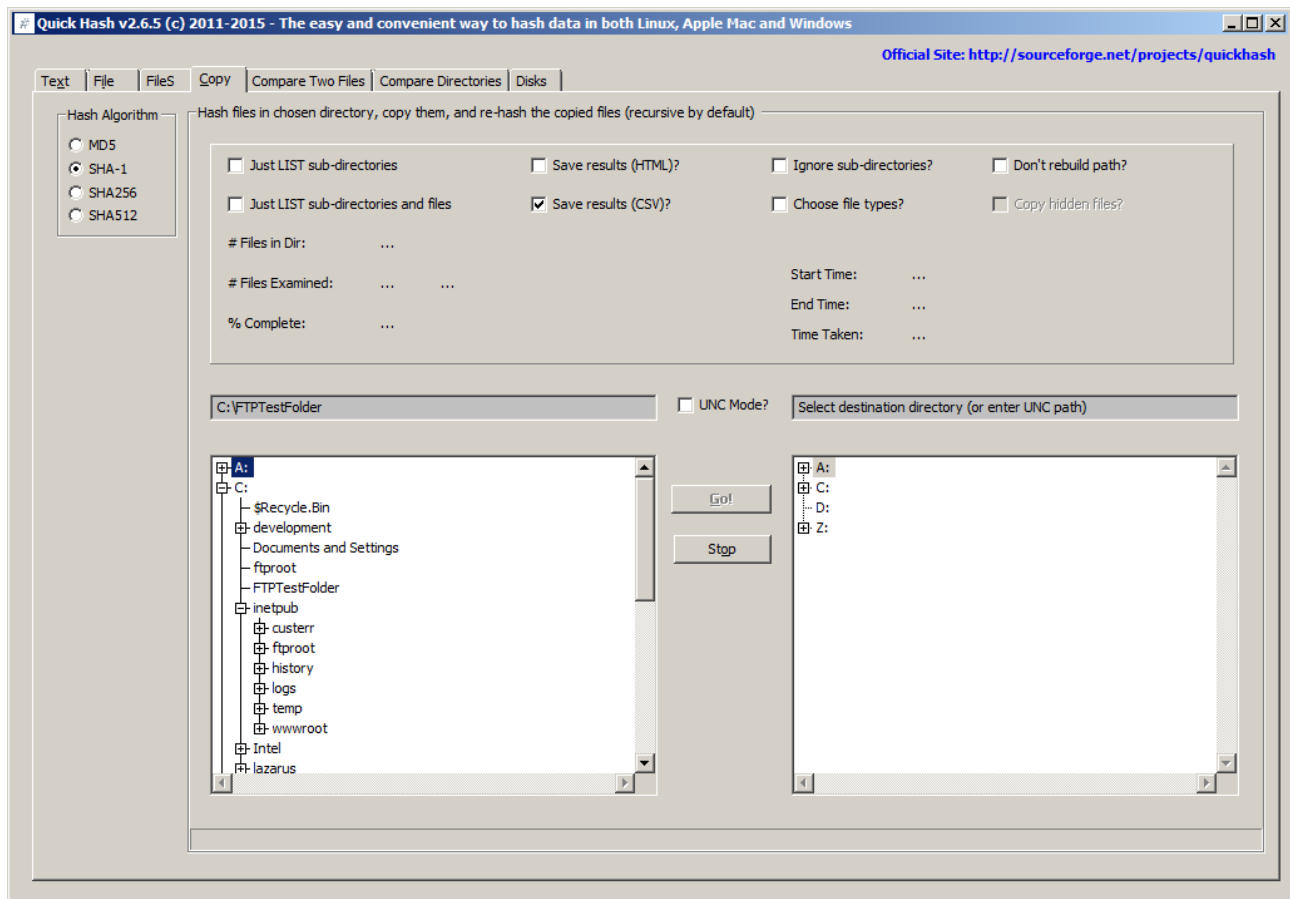


Illustration 9: The default 'Copy' tab showing tree views

The “Choose file types” box allows the user to specify what file types to find, hash, and

copy. So for example, if the user is only interested in .doc files, by entering just “.doc;”, only those file types will be found and copied. Multiple extensions can be used if separated with a ';' ONLY (no spaces). Note that this type identification is conducted by filename only – not the more accurate file header signature analysis (which is currently not available in QuickHash).

The “Don't rebuild path?” option allows all files found in the source directory and its sub-directories to simply be dumped in the root of the destination directory without rebuilding the original path in the destination path. Obviously though two files of the same name cannot exist in the same directory on the same filesystem whereas two files of the same name may exist in one folder and any of its other subdirectories. To account for this, when this option is enabled (it is off by default) QuickHash will check for the existence of a file with the same name in the destination directory for each file it copies. Where found, it will rename the second, third, fourth (and so on) instance of the file by renaming it to `FileName.ext_DuplicatedNameX` where X is the counter of duplicated filenames detected. Note this is not a check of file hash based on content – merely filename.

The “Copy hidden files?” is disabled in the Windows version, because hidden files in both hidden and unhidden folders are found by default with this particular tab in QuickHash (different to the 'FileS' tab and its 'Hidden folders too?' tick box). However, in Linux and Apple Mac, the box is enabled, due to the way that files and directories are both, in essence, 'files' on those systems, so a hidden directory needs to be dealt with differently to a hidden file. If you want such files when using either of those systems, tick this box. But Windows users need not be concerned about it.

The hashes cannot be dynamically re-computed in this tab, unlike in the first two tabs.

Drag and drop of directories is not possible in this tab either.

On Windows, as of v2.6.4, files that live in a folder whose length exceeds 260 characters should now be found by QuickHash and copied. Previous versions of the program could only copy files TO a folder longer than 260, but it could not read/detect them. As of v2.6.4, it should be able to do both.

This is due to a restriction called 'MAX_PATH' (see earlier description) and is a MS Windows limitation. It is not an NTFS filesystem limitation; that can allow up to 32K characters. Nor a Linux limitation, which allows up to 4K characters.

Note, however, that if the files that are found are already near to the 260 character limit, but not beyond it, it is likely that when copied, the path length will exceed 260 characters. QuickHash will deal with this by implementing a filesystem bypass.

3.2.5 Compare Two Files : It is commonly the case that a file exists in two different places, for example, a backup of a file. This tab allows a user to specifically choose one file, and then hash it against another file automatically, perhaps leaving them overnight if they are large. This avoids the need for the user to have to hash all the files in the folder of these two respective files (using the 'FileS' tab), or without the need to hash FileA first, and then manually choose FileB second using the 'File' tab.

Results can be saved to a text file, if needed, after hashing has completed and by clicking the 'Save As' button.

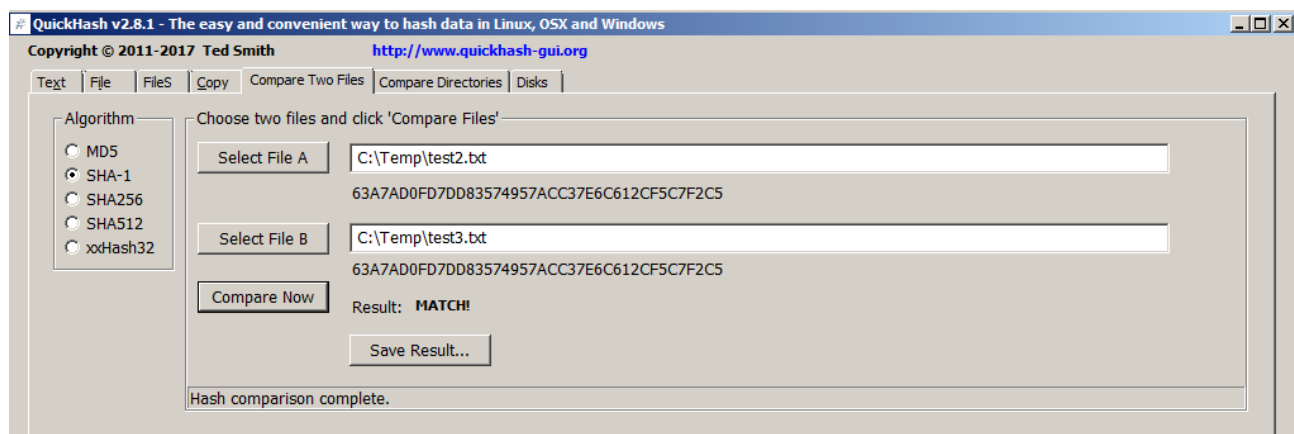
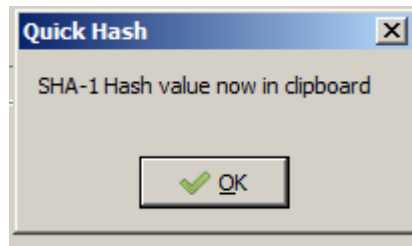


Illustration 10: Comparing the hashes of two identical files

As of v2.8.1, if **the user single mouse clicks** the generated hash value, the hash value and the chosen algorithm will be copied to the clipboard. This only applies to the “Compare two Files” tab, for now.



Of of course, the user can just take a screenshot of the results!

3.2.6 Compare Directories : This enables the user to compare the file content of two directories. The user must select one directory as the source (Dir A) and then a second to compare the source against (Dir B). QuickHash will then count and hash all the files in Dir A and Dir B, storing the values in a list in memory. When complete, it computes a hash of both the memory resident lists (which it can do in a millisecond) and compares them. If the “hash of the hash lists” are the same, the two directories must match, which avoids the need to then compare individually each hash value, value by value. This makes QuickHash very fast at this comparison.

If, however, there is a mis-match at this stage, QuickHash then does go on to compare the hash of each file and then alerts the user as to which files are in one directory but not the other, or which files are found in both directories by name but are different by hash.

Note therefore it compares the binary content of two directories based on the files inside each. It does not compare the directory structure of each. So you could have Dir A with 1000 files in it spread across 5 different sub-directories but if those same 1000 files are all in the root of Dir B with no sub-directories at all, QuickHash will report a match, because the **file content** (file count and file hashes) of both directories is the same.

Upon completion, the user can copy the content of either grid or both grids to the clipboard using the two buttons in the summary panel (as of v2.6.1). Prior to completion, the buttons will be greyed out.

With v2.7.0, a new option was introduced which is enabled by default which is to only tabulate errors, instead of the complete file and hash listing. This makes Quickhash much faster, especially

when dealing with large data volumes. So if there are 10K files in Directory A and 10K files in Directory B, but only two have a hash mis-match and, say, five files are missing from Dir B that were found in Dir A, it will only tabulate seven files, instead of 20K file entries. If the user wants a list of files in a directory along with their hashes, he should use the “FileS” tab, as has always been the case. The comparison tab was never intended as a means of generating a log of file names and hashes; it has always been to compare to directories for a match or mis-match.

If the option to tabulate only errors is disabled, then the same behaviour as before is seen. Two tables are shown with content of each. Note though that if errors are encountered they will be included (listed) with the matching data, but prefixed with text such as “Hash does not match” or “File found in Dir A but not in Dir B”. So by sorting the columns by single clicking their top row, the faulty files will be obvious.

Lastly, if errors are encountered the user is now prompted (as of v2.7.0) to save the details as a text files. This will list the filename and hash of the files that are seen to be different or missing.

3.2.7 Disks : This tab was only available in the Windows version prior to v2.7.0 but is now available for both Windows and Linux users. Apple Max OSX are not currently supported.

Windows and Linux users must run Quickhash as Administrator or (on Linux) root or sudo.

The functionality enables the computation of a hash for the physical disk of the computer or a logical volume (like Drive E:). Useful for comparing the value computed by one forensic tool to “another tool”. On modern disks, speeds of between 7Gb – 14Gb per minute have been observed. Though speeds of 4Gb per minute are quite common and still fast compared to many other tools. With the newly added xxHash algorithm, speeds of 15-20Gb a minute should be observed.

To use the functionality, the user must click the “Launch Disk Hashing Module” button in the tab “Disks” and he will then be presented with the following screen:

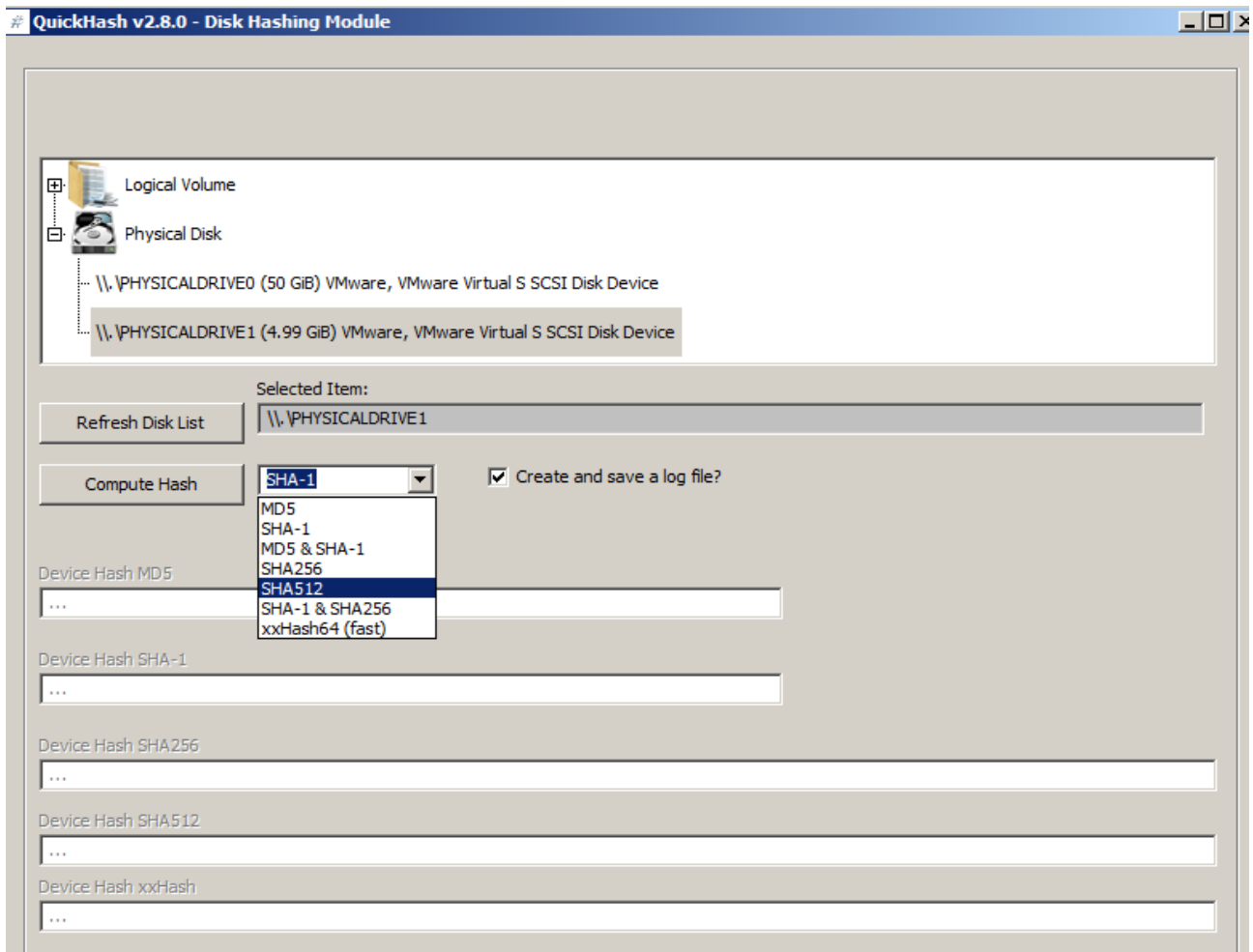


Illustration 11: The Disk Hashing Module of QuickHash v2.8.0

With v2.8.0, a full logging feature is available that records the version of QuickHash, date and time of starting and finishing with time taken, the hashes computed and so on that is written to a text file on completion in a place of the users choosing.

The disk hashing module is largely based on the sister project of QuickHash called YAFFI ('Yet Another Free Forensic Imager' also by Ted Smith and also open-source), available from <https://github.com/tedsmith/yaffi>

Windows and Linux users only : To choose a disk, the user must simply single click the physical disk or logical volume, select their preferred hash algorithm (SHA-1 default) and then click "Compute Hash". As of v2.8.0, it is possible to compute SHA-1, or MD5, MD5 & SHA-1 together, SHA256, SHA-1 and SHA256 together, SHA512 or xxHash. XxHash64 especially is seemingly the fastest hash algorithm available and is about as reliable cryptographically as MD-5 (though xxHash

is not strictly speaking a cryptographical algorithm from what I can gather).

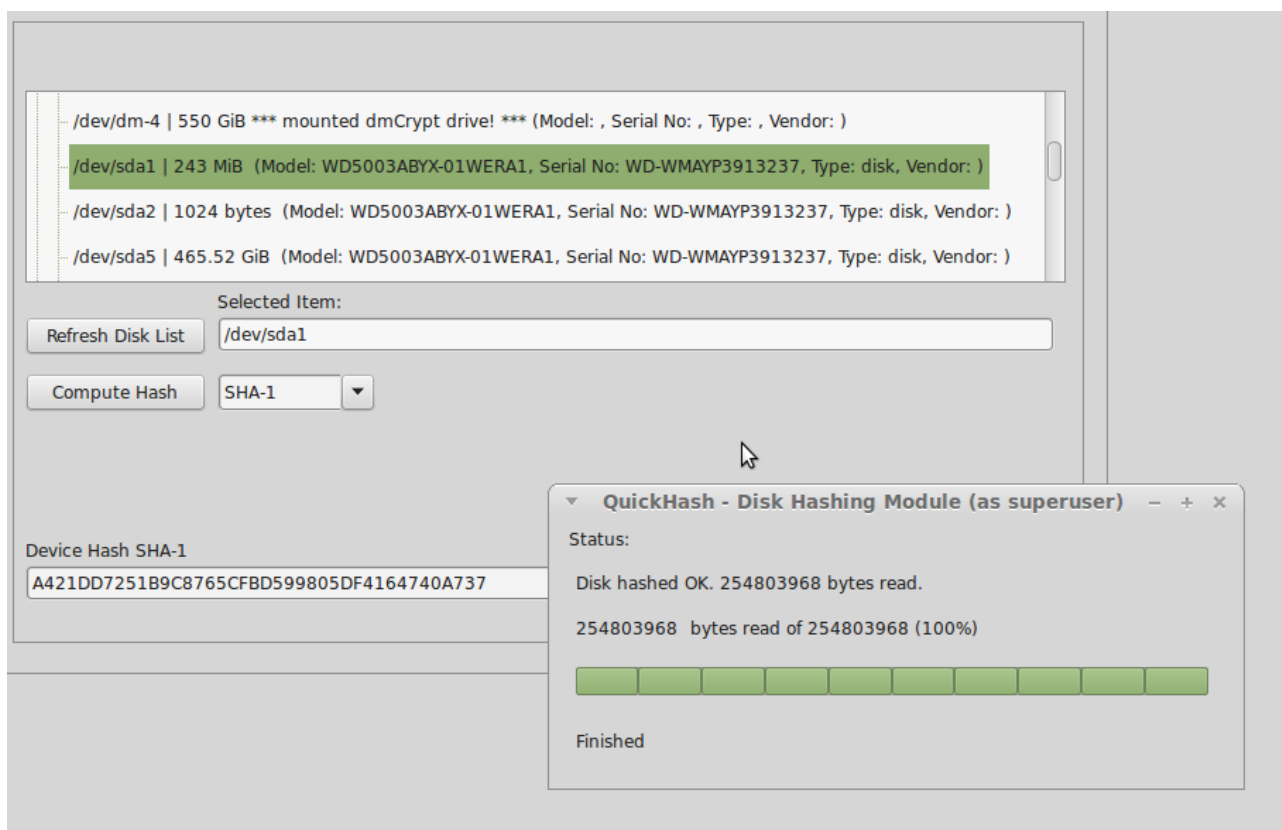


Illustration 12: Hashing a logical volume in Linux using QuickHash

3.3 Unicode

Be aware that QuickHash is Unicode aware on Linux, Apple Mac and Windows systems. It will process files with Unicode characters in their filenames or in their content without difficulty. *Note that Windows versions prior to 2.3 were not Unicode aware.*

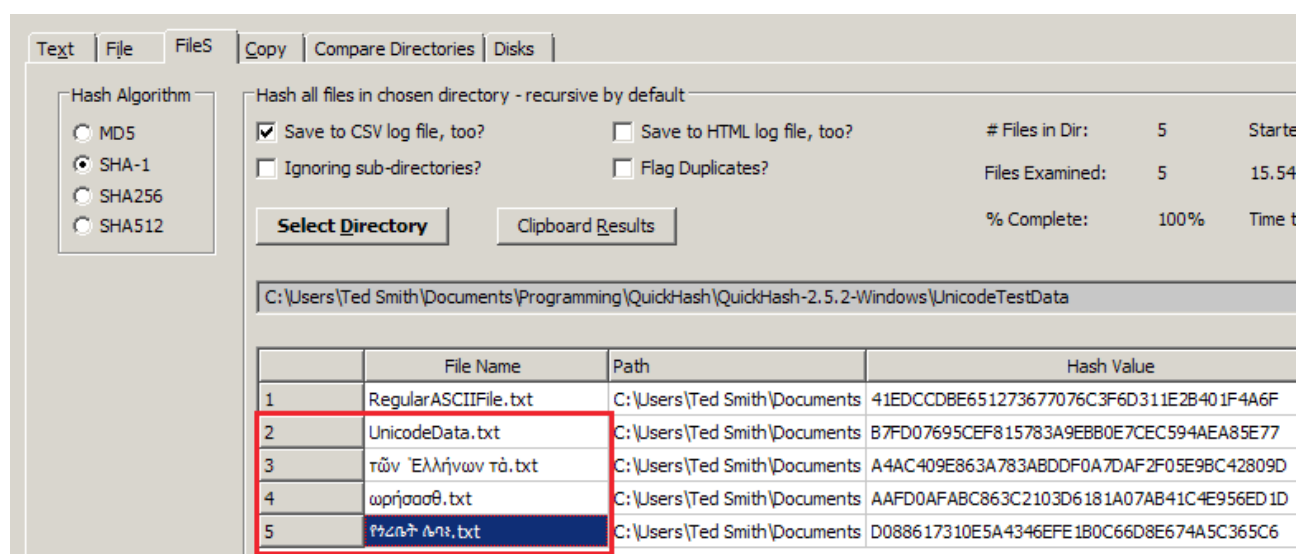


Illustration 13: Quickhash showing Unicode awareness on Windows

3.4 Other tools of the same name


There are many hashing tools available – too many to mention - and they all have various strengths and weaknesses, just as QuickHash does. This section is written to try and help with the enquiries received that are worded along the lines of *“I downloaded QuickHash following our chat the other day but it doesn't do half the things you said it could”* which, since about 2012, caused some enquiries of my own, at which point it became clear that new projects have been developed since QuickHash that are also called “QuickHash”!

It is important to point out, however, that QuickHash was the first tool of its kind (a graphical, standalone, free, open-source, Unicode aware and cross-platform data hashing tool) named “QuickHash” and it was published on Sourceforge in 2011 at <http://sourceforge.net/projects/quickhash> . Unfortunately, the nature of open-source means there will always be others who might choose the same name for a tool that does the same thing. Without trademarking the name, there is no way to prevent that happening. So please be advised that any other tools appearing to be called or actually being called “QuickHash” are not the same as QuickHash GUI as seen at www.quickhash-gui.org (since late 2016) and Sourceforge at <http://sourceforge.net/projects/quickhash> since 2011 up to 2016.

3.5 Donations

QuickHash is created in my spare time, which is very tight with a full time job and family. It is generally written late at night, which can sometimes explain oversights and can be the cause of some bugs!

If you value the program, or if your organisation, company or agency has found that it has helped them, then please consider making a donation using <https://paypal.me/quickhash>. Doing so helps fund the server costs and also helps inspire future development. With your permission, your donation may be published on the website under the donors section, so it's a good opportunity to get your company listed on the website of one of the worlds most widely used data hashing tools, <http://quickhash-gui.org>.



EDWARD SMITH
Derby, GB

SEND MONEY TO EDWARD

£0.00

GBP ▾

Next