QuickHash-GUI

V3.3.3

The Free, Graphical, Cross-Platform,

Data Hashing Tool

User Manual : v3.3.3

http://www.QuickHash-GUI.org

(and previously https://sourceforge.net/projects/QuickHash)

1.0 Pre-amble

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, including any future digitally code signed versions. 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. This user manual, comprehensive as it is, uses terms you might not be familiar with and can not always account for different words that we might take to mean the same thing. e.g. Export, or Save.

The screenshots in the manual may not always fully reflect the latest version, but differences are typically few or minor.

1.1 License Agreement

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 (tedsmith@quickhash-gui.org).

Supported Platforms: Tested on Microsoft Windows 10, Linux Mint 19.3, and Apple Mac OSX Big Sur. Apple Mac users using versions of OSX older than Big Sur can use v3.2.0 and below, only.

QuickHash GUI is made available under the GPL2 license (see full details https://quickhash-gui.org/githubfeed/) 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 2 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. <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.

The hash libraries of HashLib4Pascal (https://github.com/Xor-el/HashLib4Pascal) by Ugochukwu Mmaduekwe Stanley (aka Xor-el) is used for all of the mainstream hashing algorithms used by QuickHash. Significant thanks and appreciation are expressed to Mr Stanley for the open-source library.

Further thanks to the authors of xxHash and Blake for making those open-source as well. Note that xxHash is licensed under BSD License and Haslib4Pascal is also licensed under the MIT License. Blake2b is licensed under CC0.

Thanks expressed to Joachim Metz for the C libewf library v2 (v3 not incorporated yet) which was added to v3.3.0 to add the ability to open and hash verify the content of Expert Witness Format (EWF) forensic image files, also commonly referred to as E01 images, following mainstream adoption by EnCase. Acknowledgment is given to the contributors of that as documented here.

libgcc_s_dw2-1.dll (GCC low-level runtime library) is licensed under GPL licence (https://gcc.gnu.org/) and the DLL was compiled using MSYS2

libwinpthread-1.dll is licensed under MIT, BSD (https://packages.msys2.org/package/mingw-w64-x86_64-libwinpthread) and the DLL was compiled using MSYS2 by Ted Smith

zLib is Copyright (C) 1995-2017 Jean-loup Gailly and Mark Adler (https://zlib.net/zlib_license.html) and the DLL was compiled using MSYS2

SQLite DLLs were sourced from www.sqlite.org but can be compiled from source using MSYS2. For detail relating to the license usage of SQLite, please see here : https://www.sqlite.org/copyright.html

1.3 Development Team

Lead Developer: Ted Smith

Contact: tedsmith@quickhash-gui.org – see support pages and ticket system for feature requests and suggestions

GitHub Repository   https://github.com/tedsmith/QuickHash/releases

Supporting Developer :   darealshinji

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, Zorin OS, and Ubuntu, and Apple Mac OSX.

It does not require installation and can simply be executed from a USB drive or other removable device (though on Linux and OSX, 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 there are the usual four 'mainstream' hashing algorithms available – MD5, SHA-1, SHA256 and SHA512 but in addition xxHash (when using the 32-bit version of QuickHash, only xxHash32 will be available, and visa versa for 64 bit), SHA-3 (256 bit), Blake2b (256 bit), Blake3 and CRC32 have been added over time.

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 Two Folders | Disks | Base64 Data
```

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 with the inclusion of the increasingly popular xxHash from v2.8.0 of the program. 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 a probability of 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 MD5 and SHA-1 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)*".

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" (|thus the name) 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 often pre-built into the [DEFT](), [CAINE]() and [Parrot Security]() Linux CDs so by using those systems you will have the power of QuickHash built in to your live CD instance without having to use it separately. Though, note, the release cycles of QuickHash are generally every couple of months whereas bootable distributions generally have a longer cycle refresh period. There are also DEB packages often built by darealshinji that accompany the binaries. All are typically available on the website.

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 mac OSX cannot currently utilise the 'Disk' tab, but may hash disks using the 'File' tab in the same way as is possible with Linux. In both instances, the program must be run with sudo privileges.

The Apple Mac version was first developed with version 2.5.3 in 2014. It functions in a similar vein to the Linux version. The Mac version was first compiled on the Yosemite operating system and version 3.0.0 using Sierra. Since 2014, all three operating systems have been supported as much as possible, though OSX continues to be the most challenging. As of v3.3.0, and with the release of OSX Big Sur, it was no longer possible to "reference a library" by filename. As such the dynamic linker cache is now used (as of April 2021) by QuickHash-GUI meaning v3.3.0 will not work fully on any previous OSX version other than Big Sur. As such, any OSX users who have not taken the plunge to Big Sur are limited to v3.2.0. Worse still, since the release of OSX Ventura, I cant compile it all (until further notice), so Apple users are rather stuck, now. Not my fault.

2.3 SQLite Implementation

October 2017 to January 2018 saw the biggest re-write of QuickHash since the programs inception. A large amount of the work involved migrating existing capability to interact with SQLite but the benefits are worth it. SQLite holds data very efficiently and it allows developers many more options for their programs. Many of these new and improved features are now available by right clicking the display grids, allowing some of those previous options to be removed from the interface, to free space.

In addition, there are now many many options for the display grids. They do vary a little but overall options include sorting by filename, path, size, ID Number (i.e. row Number, shortened to "No" commonly), duplicates, known to a hash list or not and many more besides (they may be re-phrased in future versions so no intricate detail here – just right click a display grid and see what comes up).

For Windows, a 32 and 64 bit pre-compiled SQLite DLL file is typically shipped with QuickHash. The correct one will be loaded automatically as QuickHash determines what architecture it is running on. It is important that these files are not renamed or moved from where they are stored in the zip file structure. For versions 3.2.0 and below, this was usually in the root folder. As of v3.3.0, they are located in a libs sub-folder of the project folder and must not be moved or renamed.

If you run the 32 bit version of Quickhash on a 64-bit version of Windows, it will utilise the 32-bit SQLite DLL but it will still work perfectly well. You can not run the 64 bit version on a 32-bit Windows system however.

On Linux, QuickHash will look in several common locations for the SQLite SO file. If it finds it, it will create an SQLite database to use for that session, and it is deleted when QuickHash is closed. This is to ensure Quickhash does not leave remnants of its activity and to ensure it starts with a clean slate every execution. The location of the database will be decided by your operating system security settings, but it is commonly and will reside there for as long as QuickHash-GUI is open:

```
/home/Username/.config/QuickHash
```

On OSX, since the release of Big Sur in 2020, dynamic libraries are no longer available on the filesystem. Instead the linker cache is used. This required some re-working in v3.3.0 of QuickHash and should now hopefully work on Big Sur, but v3.3.0 and newer will not work on older releases of OSX. And since OSX Ventura, we are totally stuck – no support there now friends.

If Quickhash does not find SQLite then the FileS and Copy tab will not work as intended. The other tabs will work as normal so depending on your needs, it may not be an issue.

The database is named based on the date and time of Quickhash execution to allow multiple instances of the program to run (a separate database will be created for each instance enabling

multiple executions of Quickhash on one computer) from the same launch location.

Since v3.3.0, it is possible to make a copy of the SQLite database easily by pressing the "Preserve DB" button. This will *attempt* to make a copy of the SQLite file for you and save it in a place of your choosing. This can be helpful if, for some reason, QuickHash itself proves insufficient for your needs and you can instead use a dedicated SQLite exploration tool and your own SQL commands to query, search, filter etc. For users who are processing enormous volumes of data, like many millions of files, this may be preferable. Note it will make a copy of the *open* database file, so it is possible some transactions may not have been committed yet, though this is not likely. Nevertheless, it comes with that health warning.
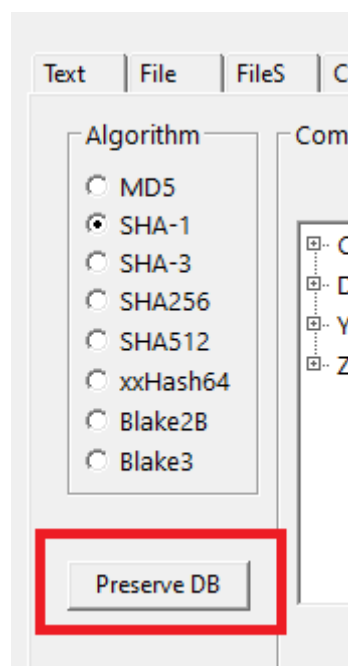
3.0 Interface

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 public key data that can be copied from somewhere else to QuickHash

File : For selecting and then hashing one individual file.

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

Copy : Essentially, this is Copy & Paste but with the added data integrity of hashing at either end of the process. 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 Two Folders : For comparing the file content of one folder to another to see if all the files inside each match based on hash and count and, since v3.3.0, filenames are also considered.

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).

Base64 Data: New to v2.8.3, it allows the user to hash an encoded Base64 file AND generate a hash of it's decoded counterpart without the user having to create the decoded version. It also allows the decoding of Base64 encoded data, just for convenience.

Several tabs contain a small drop down menu with the default value of "Set Delimiter". This is an attempt to help users choose what delimiter character is used whenever the display grids are right clicked and the user chooses to Save As CSV. Comma is used by default if "Set Delimiter" is left like that. Or the user can actually select ',' from the list, which has the same effect. The other recommended options are either the Tab character, which makes it far easier to use the output with Excel etc and helps in cases where, for example, the user may have a collection of e-mails copied to a folder. E-mails often take the name of their subject line, and so on disk they may include characters like a comma. As such, the comma can then cause delimiter confusion, so a tab works better here. Or the hyphen perhaps, though the same example can apply there too. Space is also included for anyone who fancies using it though it is not recommended for entirely obvious reasons and QuickHash will even warn of the user of that. In other words, it is providing a nudge away from crazy town.
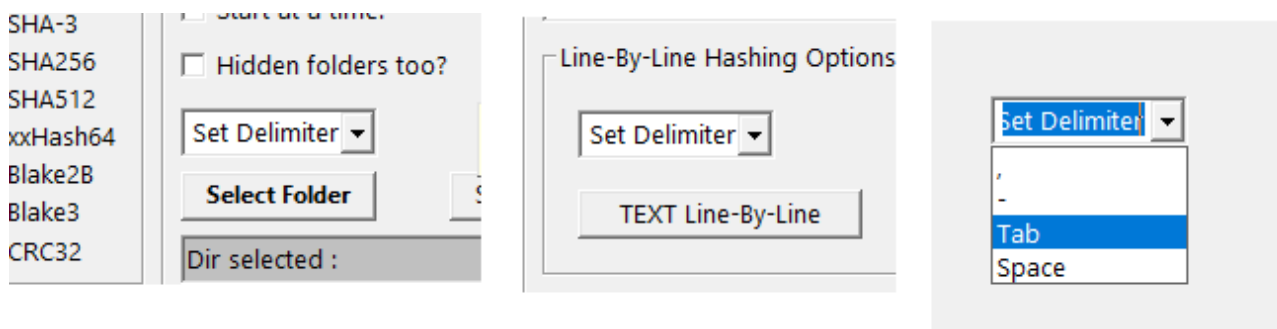
Illustration 1: Delimiter drop down new to v3.3.0

If the user decides to set it after the analysis, then as long as the display grid has not yet been closed, it **should** still take effect when the user triggers a clipboard or Save to CSV option. But it is new to v3.3.0 and may not always work. It was surprisingly more tricky to implement than the user might think.

3. 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 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. Also be aware of carriage return characters that may not visible. Use Notepad++ or similar software to sanitise your input data. You should prepare your list in advance using Microsoft Excel or Notepad++ and ensure it is correct.

In addition, a second button allows a large 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 hash, in the output. That is what this is there for.

As of v3.0.0, there are buttons to convert your inputted text to upper or lower case, for convenience. ASCII as well as Unicode text should work OK.
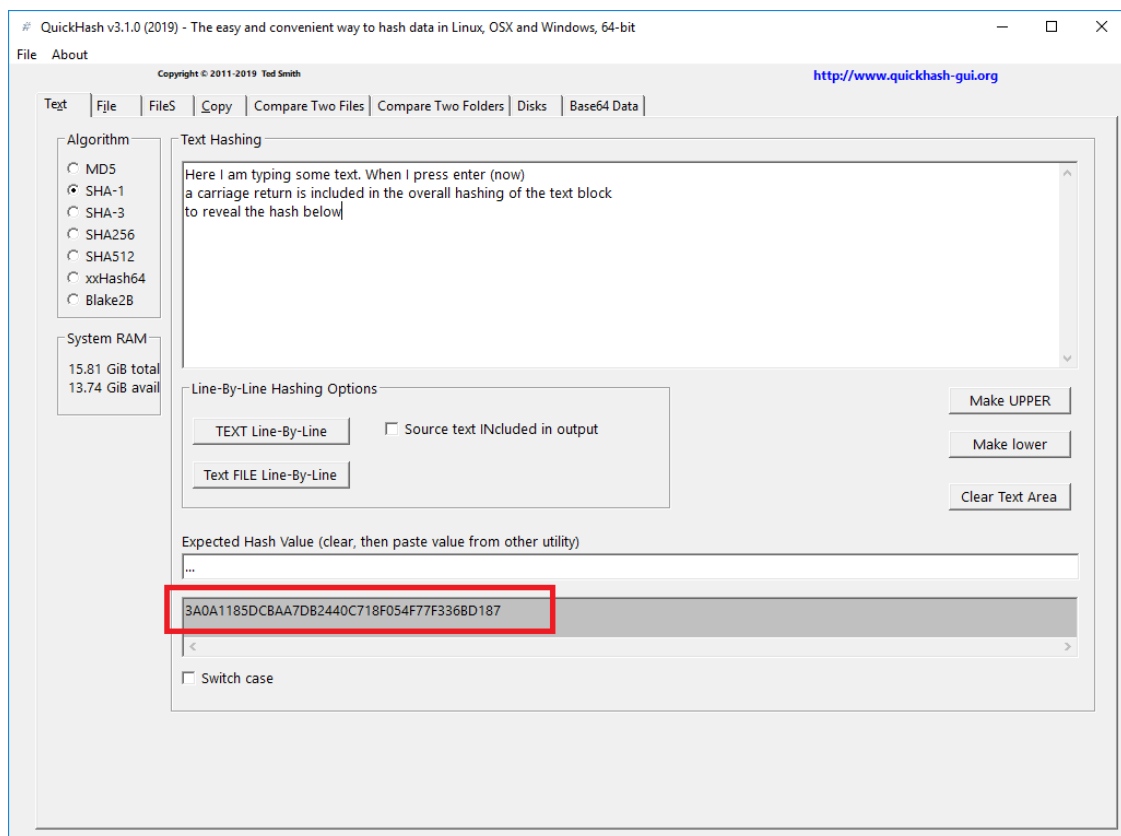
Illustration 2: Hashes values for the sum of the text are dynamically recomputed as the user types, or, using the "Hash Line-by-Line" button enables the entire list to be hashed line by line
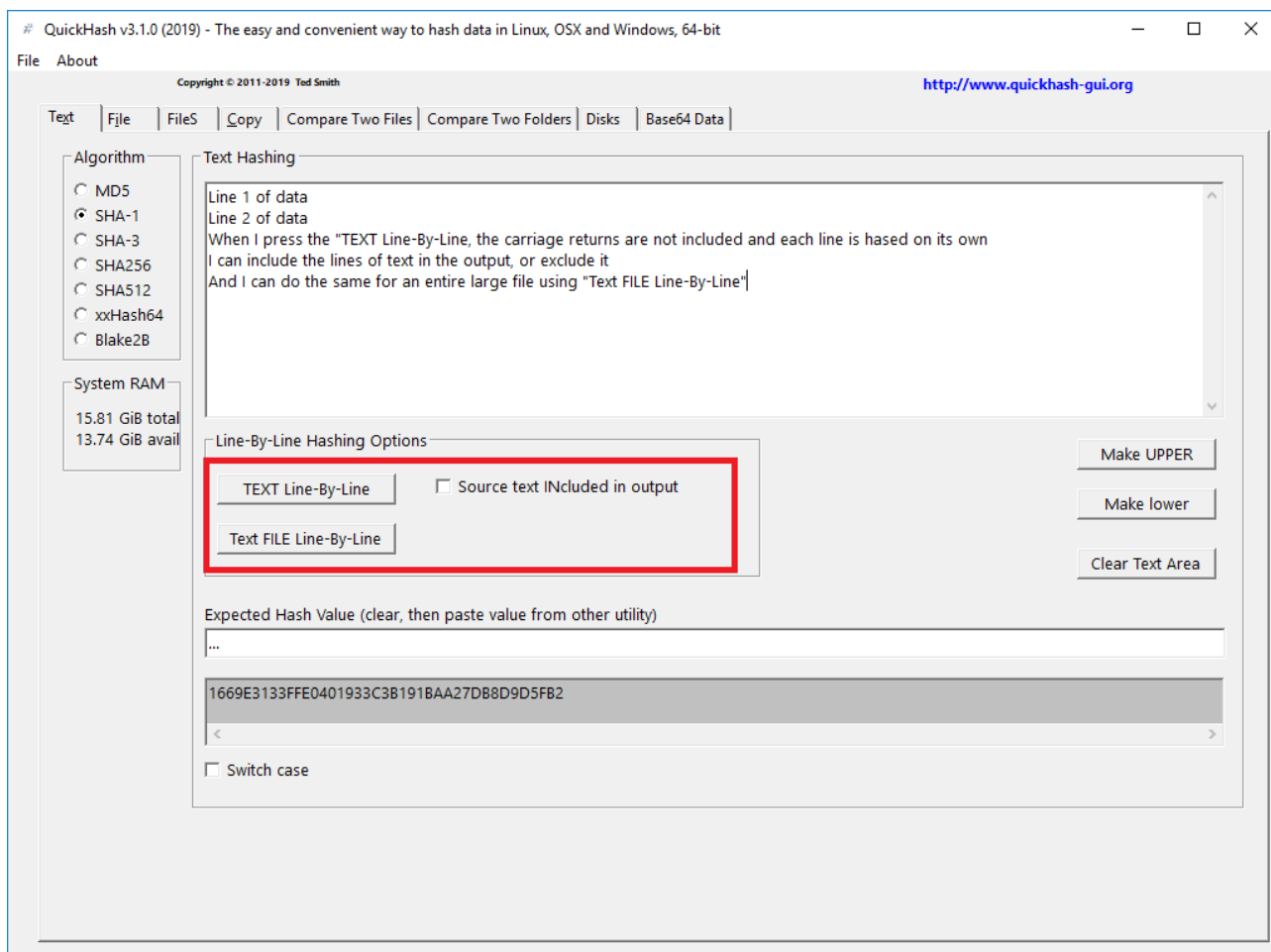
Illustration 3: The text hashing functions available since v2.6.7

3.1.2 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) 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. As of v2.8.3, you can add the value after hashing the file, and if QuickHash identifies that there is one of the 9 valid hash values in that field, it will then see if it matches the computed one. Or you can paste the value in there before hashing

the file, and it will then check after computing the hash if it matches the one the user has pasted.

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`".

The ability to hash a file 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 (or any hashing tool for that matter) to recompute the same hash on arrival and check the computed value against what you put in your e-mail. It's also very useful for users who download important data from the Internet, most commonly operating systems and patches. Using QuickHash and the hash value that the web developer places on their website, you can be sure that the file you have downloaded is the same as the file they put there.
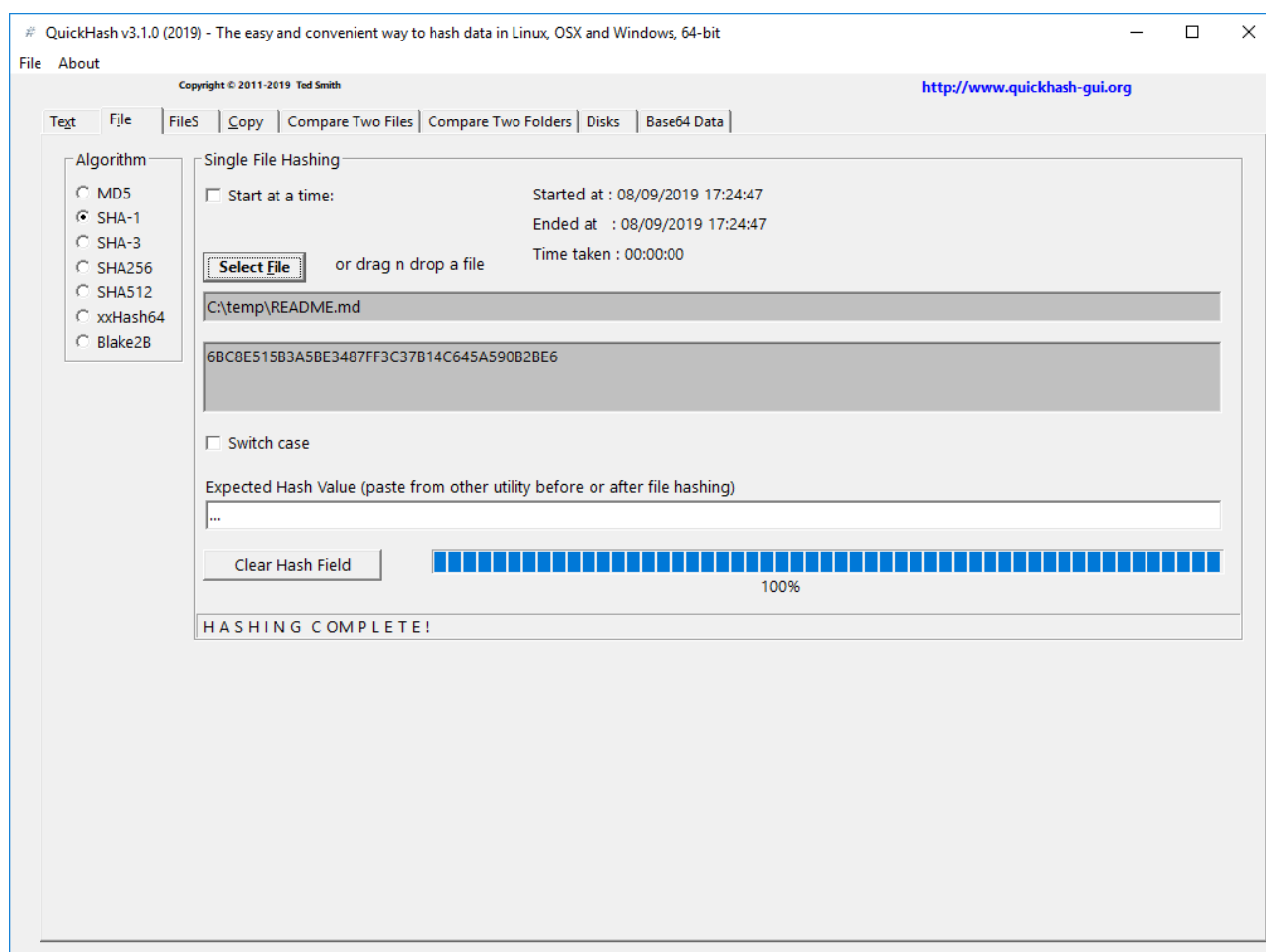
Illustration 4: The 'File' tab showing a computed hash of a file

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 and Linux as a GUI in the 'Disks' tab, but sadly not for Apple Mac OSX - see Disk Hashing, below.

With regard to forensic images created by digital forensics specialists; 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 ability to compute the internally computed hash

of the data inside the image. All versions of QuickHash **prior** to v3.3.0 could only hash the image segments, not the internal data of the E01 file. In other words, if the user navigated to the first file of a chunked image set (such as a chunked dd image or a chunked E01 image) the **resulting hash will be that of the chosen image chunk (i.e. the file) only**, not of the acquired data that sits inside the entire forensic image spanning multiple chunks.

However, with v3.3.0, for Windows and Linux only (not OSX) the libewf library by Joachim Metz has been added. This means that if a user selects an .E01 file, QuickHash will then assume the user has selected a forensic image which may include .E02, .E03...etc segments. It then calls the libewf library to check that it is in fact a proper forensic image. If it is, QuickHash will ask the user if he\ she wants to hash the internal data of the image, or the image segment that was selected.
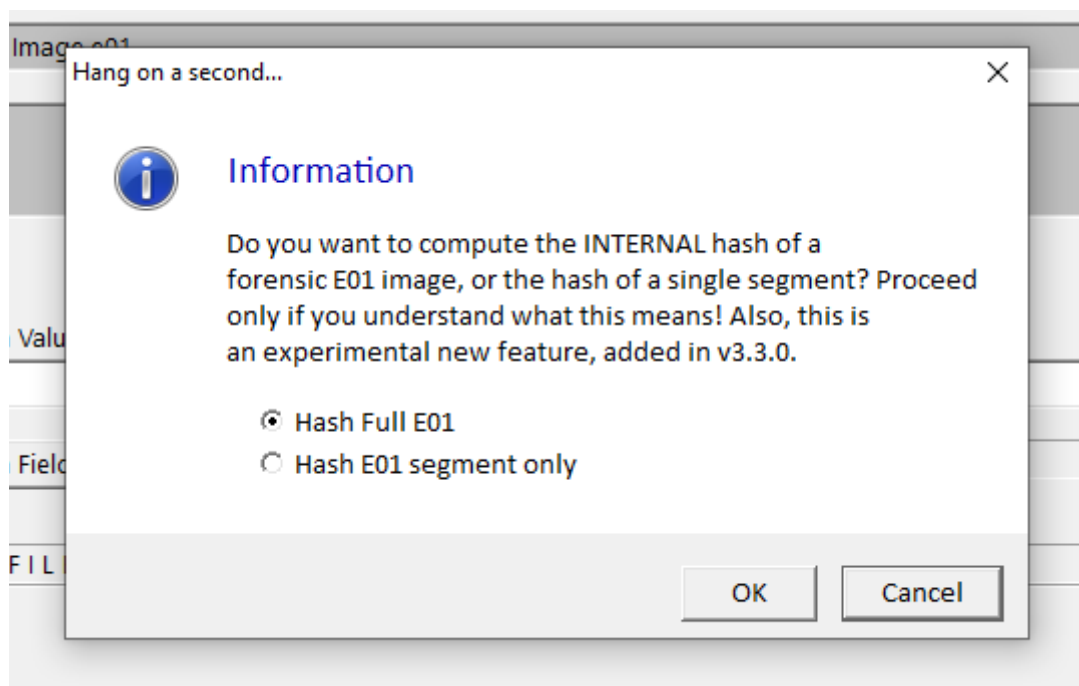
Illustration 5: Windows shown when user selects an E01 Expert Witness Format file

If the user selects to hash the full image, it will also lookup the embedded MD5 or SHA-1 hash values (which one is looked up correlates to the one chosen by the user for hashing), if it is available, that the forensic tool inserted at the time of acquisition. It will then report that in the "Expected Hash" field, and check that the computed hash matches the stored hash. This is only for MD5 and SHA-1 currently. If no embedded hash value is found, it will simply report the computed hash of the full forensic image as normal and the user can then either cross compare that value with

another tool, or simply make a note of it for audit purposes.

If the user chooses to hash just the selected E01 segment, on the other hand, then the hash will be computed of that file segment, as normal for any other file. If the user switches the algorithm without switching the chosen file, he\she will be asked again what type of hashing is to be conducted, because this is a somewhat important distinction in data hashing terms and not something to be assumed.

The results of the E01 image hash computation and verification steps have been cross checked during development using X-Ways Forensics v20.0 and the results are the same as seen in the illustration below (though obviously X-Ways Forensics computes the hash faster):
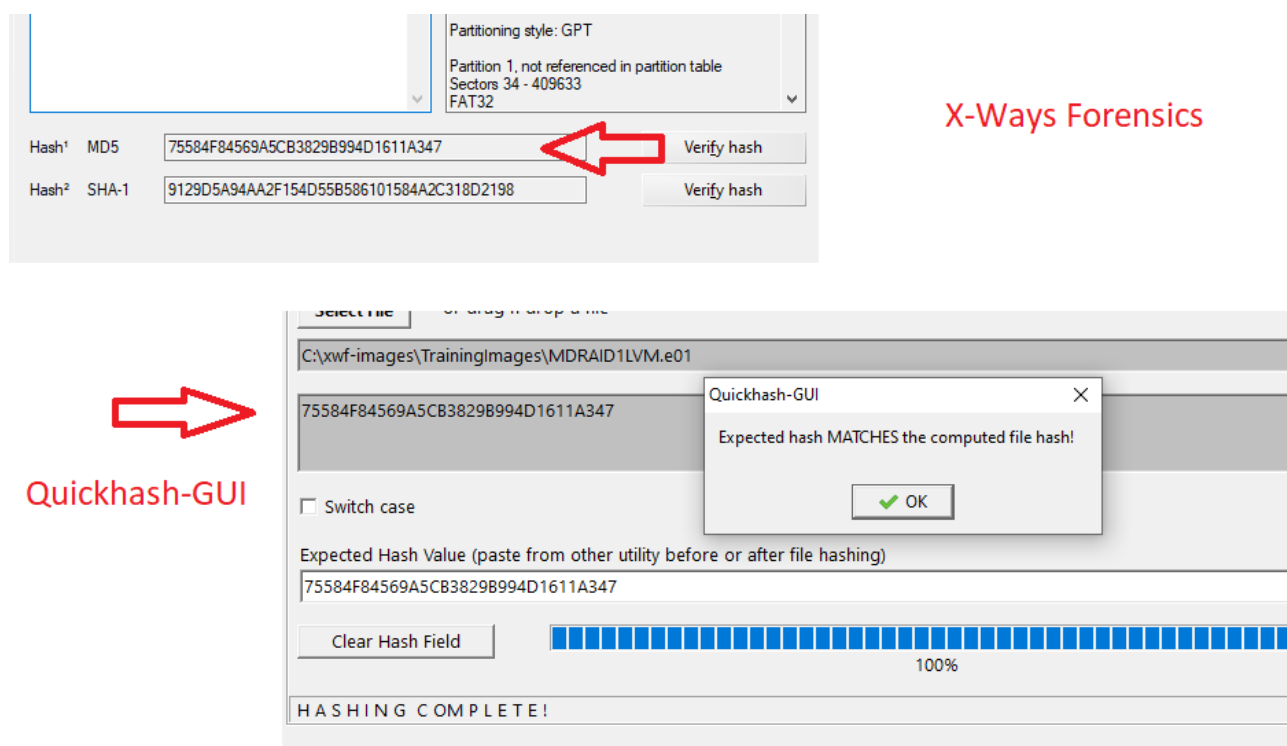
Illustration 6: E01 hashing results cross-checked against forensic tools

Note this E01 hashing functionality is only conducted using the "File" tab. E01 files found elsewhere when using QuickHash, like when using the FileS tab, the Copy tab, or Compare Two Folders, are not checked in this way as it is assumed by Quickhash that the user will only conduct such an activity on a full forensic image, and not for one that happens to have found its way into a folder with loads of other files whether by accident or by employment of some hap-hazard file management strategy.

This new functionality is designed for digital forensics experts and should not be used by folk who do not understand the actions they are conducting. As always, dual tool verification with products like X-Ways Forensics, FTK Imager, NUIX, or open source options like the Sleuthkit etc is recommended. And libewf can also be installed as a package on your system as part of the libewf-tools suite.

Note that for integrity purposes, if you have a forensic E01 image in LocationA, and you have verified it using any forensic tool in that current location, and you then copy it to LocationB, you can hash each of those image segments in LocationA and LocationB as described above using the FileS tab. If the hashes all match of each file segment, and as the originating image was verified where it was using a forensic tool, then you can be assured the copy of the image in LocationB is also essentially 'verified' by the very nature of hashing science. It is arguably not necessary to open it again in a forensic tool to re-compute the internal hash if the hashes of the segments in LocationA match LocationB, and it did verify with a forensic tool in LocationA to start with.

Those users running the 32-bit version of Quickhash-GUI may encounter an error regarding missing DLLs beyond what is included in the libs\x86 folder. For example : vcruntime140.dll. DLLs like this one, and others, are part of the .NET Framework and the Microsoft Visual C++ 2015 (or newer) Redistributable Packages. Most modern and updated versions of Windows 7 and 10 will probably already have this installed. But if you see warnings about such missing DLLs, install the Microsoft Visual C++ Redistrubtable Package at least and try again. For example the one found here https://www.microsoft.com/en-us/download/details.aspx?id=52685

Lastly, the libewf library was introduced with v3.3.0 to assess demand for it. One could call it an experimental new addition to Quickhash. With that caveat, the developer does not accept or assume any responsibility for the outcome or your understanding of the results. Digital forensics is a precise computer science, and this new feature is an attempt to implement this functionality into what has become a popular data hashing tool, for convenience and cross-platform enthusiasm. But ultimately, it is a hobby project. Digital forensic tools cost thousands for a reason, and (usually) employ dozens or hundreds of developers for a reason. Quickhash is mostly developed by one person. And it is distributed for free.

Will E01 support be added for Apple OSX? Probably one day, but not yet. The release of Big Sur hit

hard on developers. Weeks were spent trying to work out what was going on with their new library system. And compiling a new library and distributing it is still unclear. But perhaps in the future it will be more clear. Where there is a will, and at least some demand, there is some hope.

3.1.3 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. Ignoring sub-directories
2. Hidden folders too?
3. Choose file types?
4. Start at a time?
5. Load Hashlist? (since v3.0.0)
```

*Options 1 -5 all require the user to tick the box **prior** to starting the scan.*

Illustration 7: Files hashed in FileS tab

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 "`Hidden Folders too?`" option requires some detailed explanation. On Windows, hidden files will be found and hashed by default *but only if they live in <u>unhidden</u> folders*.

Files that live in <u>hidden</u> <u>folders</u>, 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 <u>folders</u> that are either hidden or unhidden, will be found. The screenshots below demonstrate.

Illustration 8: Files with various filesystem attributes
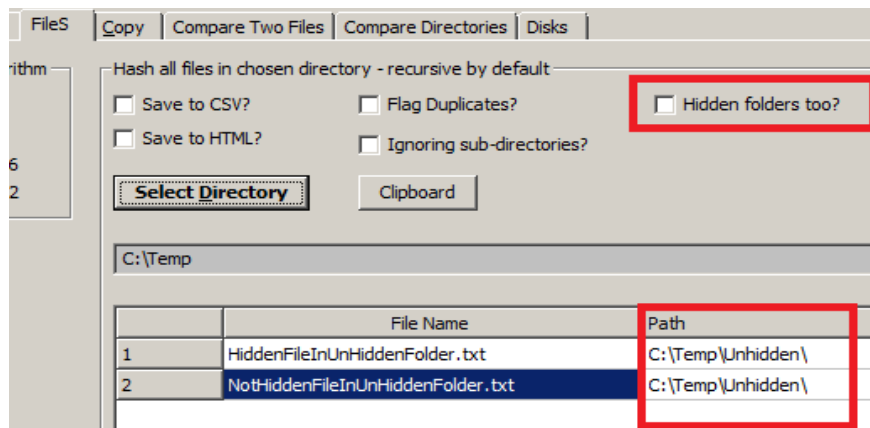
Illustration 9: QuickHash will ignore files in hidden folders if asked
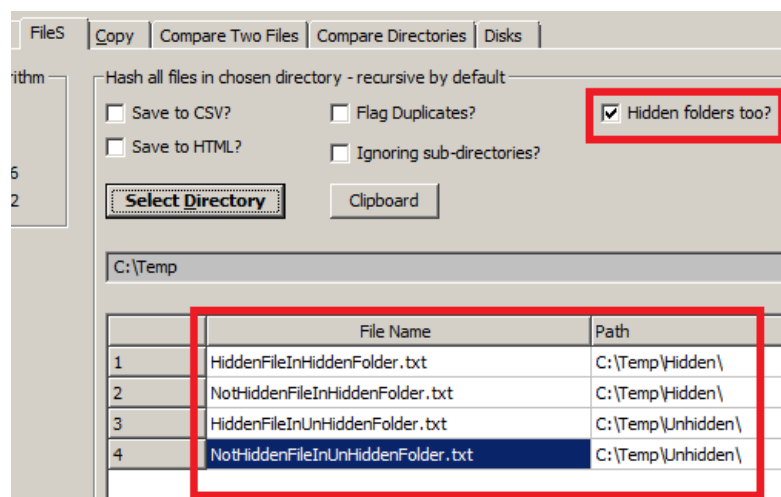
Illustration 10: QuickHash will examine files in hidden folders, if asked (v2.6.3 screenshot)

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; filename extension analysis

only.

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 file paths 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 in paths 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 however than filenames, specifically, do have a finite limit that is typically measures in character length (bearing in mind Unicode characters = 1 char, but can equal 2 bytes with UTF8, or even 4 bytes with UTF16.

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 '*Start at a time:*' option allows the user to schedule a date and time in the future to start the hashing. However be aware that in development some inconsistencies were noticed with this. It seems to work OK on some processor architectures, but not all. Your mileage may vary so try it before using it for anything important.

The "*Load Hashlist?*" option is new to v3.0.0 and was perhaps the most frequently requested feature of QuickHash by users over the years. What it enables is for the user to import a list of any number (limit yet to be discovered) of existing hash values that may have been generated at an earlier date by QuickHash or perhaps by another data hashing tool or digital forensics tool. The list must be just one column of hash values without a header row. Once selected QuickHash will rapidly ingest the values. When the user then selects a folder of files, it will compute hashes in the folder and then lookup whether the corresponding hash exists in the list imported by the user (if the tick box is ticked). If it is, it will add "Yes" in the final rightmost column of the display grid. Conversely, if it is not found, QuickHash will display "No". On completion, three options will be enabled in the right click menu allowing the user to sort or filter out the values as necessary. The column will be empty if no hashlist has been imported.

Note that a bug was identified in May 2018 and fixed in September 2018 with v3.0.3 where lowercase hashlists were not converted to uppercase, meaning values that were the same were not

identified as such. This was fixed in v3.0.3 so that all imported lists were converted to uppercase if not already uppercase.

The list will remain in memory until QuickHash is closed. The list can however be added to after one ingestion if another list needs to be added. Just click the button again to choose a second file of hash values to import and they will be added to the first one.

Once QuickHash is closed, the list will also be released from memory. After re-launching QuickHash the user will need to reimport any hashlists if they are needed, but this should not take long. Future versions may support the retention of such values long term but lets see.

Note that Hashlist importing was new to QuickHash v3.0.0 and was not an insignificant amount of work to implement so there may still be some refinement in future versions.

The computed hashes of the FileS tab cannot be dynamically re-computed, unlike in the first two tabs. Drag and drop of directories is not possible in this tab either.

As of v3.1.0, the column of hashes that have been computed for files can be exported on their own, without all the other values like filename etc. This is useful for the creation of your own hash lists and can include or exclude a header row.

UNC network paths can be selected this way as well if the dialog of your operating system supports it.

New to v3.3.0 is the ability to compute CRC32 checksums. An old algorithm for sure, but still in common use today it seems and added by popular demand for the many media moguls who use QuickHash for video and sound based files. If the user selects CRC32 in the FileS tab, the user will now be prompted for an action based on the following dialog window :
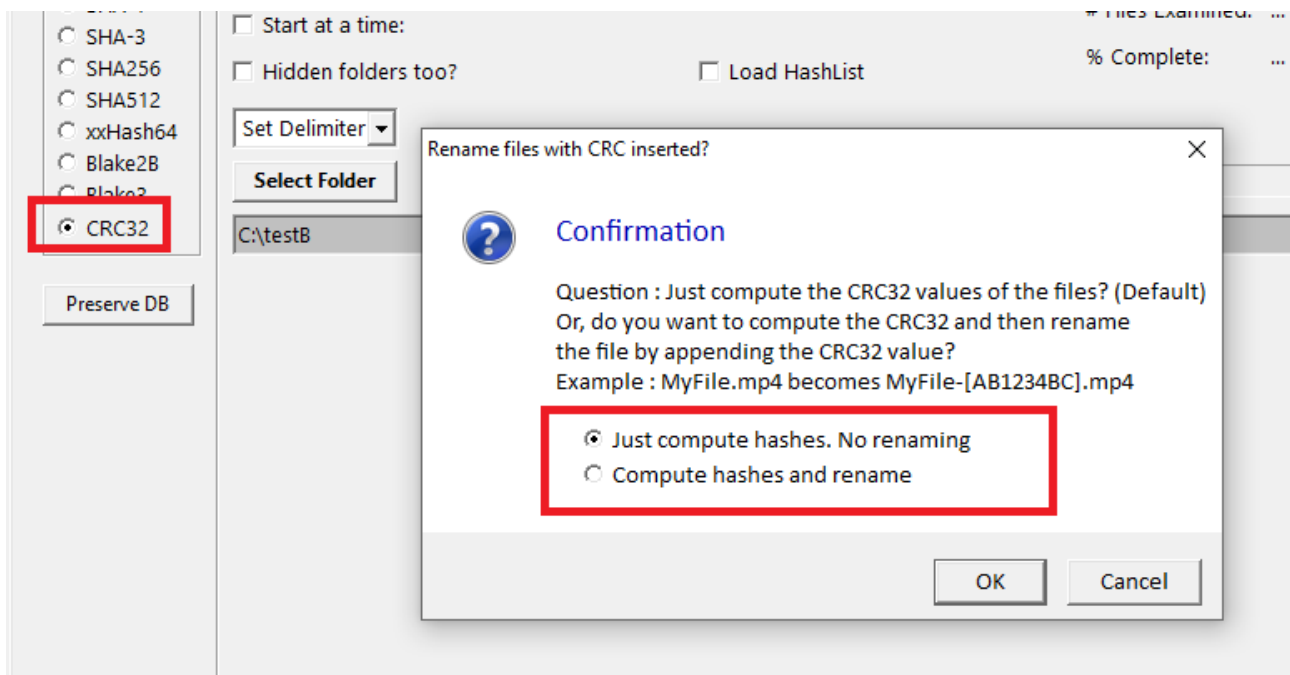
Illustration 11: Dialog window shown in FileS tab when CRC32 is selected

The default behaviour is the files will just be hashed and listed just as they are in this tab for any other algorithm. But, helpfully, the second option is to have QuickHash rename (append) the filename with newly computed CRC32 hash of the file. So for example, MyFile.mp4 becomes MyFile-[AB123456AA].mp4 and those names are then listed in the display grid and can be saved from there or copied to clipboard as required.

Caution here. The users files will be renamed if he\she chooses "`Compute hashes and rename`"! Make sure you have a backup of them first and if you do not, and the outcome is not as you expected, do not complain. Data security, conducting backups and testing restore ability is the responsibility of data holders themselves, and not of QuickHash or anyone developing it. Any such complaints will be directed to this specific paragraph of this very user manual. And the usual caveats around around use of open-source free software (no warranties etc) is reiterated.

If the user clicks Cancel here, the hashing attempt is aborted entirely and the user will have to reselect the folder of files they wish to hash as they have always done to trigger the next hashing attempt.
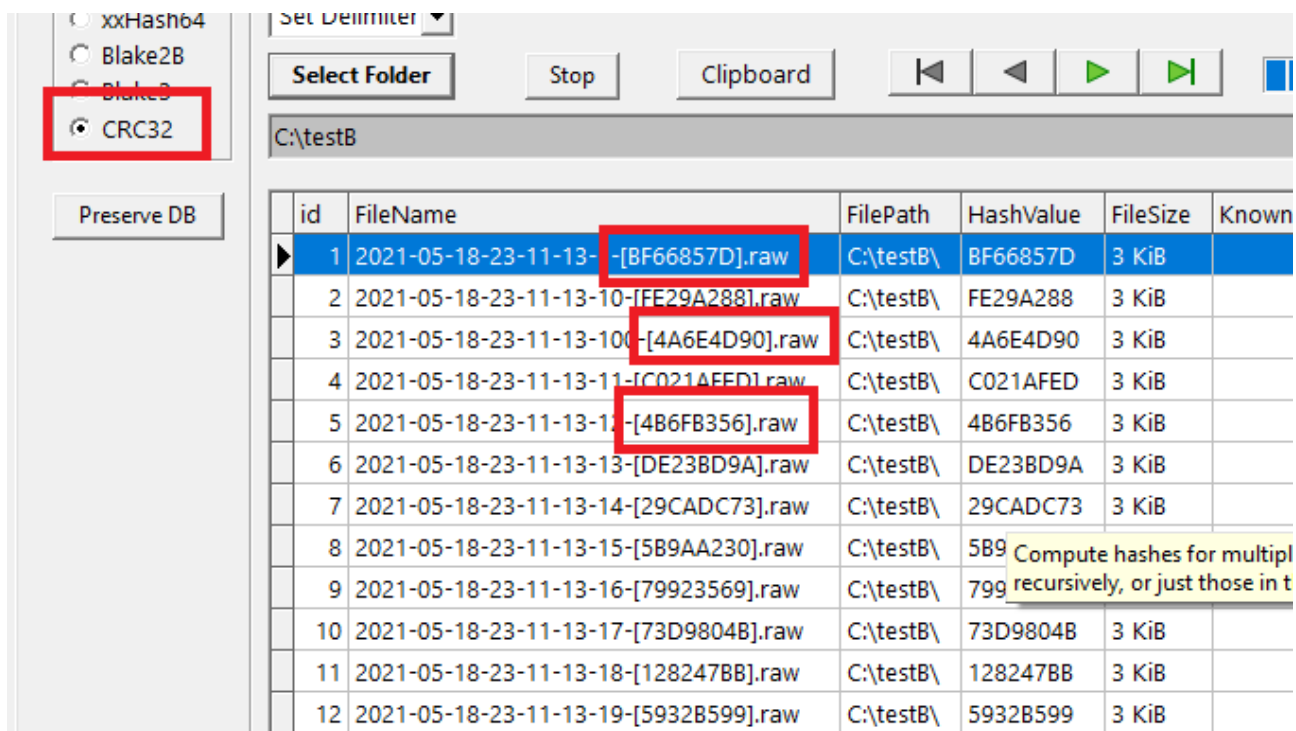
Illustration 12: Files renamed during CRC checksum computation

3.1.4 Copy : The `Copy` tab allows the user to select a folder containing files he wishes to copy from and another folder where he wishes to copy files to. Both source and destination folders can be a local folder or a mapped network drive or a UNC unmapped network address. Multiple source folders can be selected as the source (using Ctrl and left mouse click) but only one destination folder can be selected as the destination (the original folders will be rebuilt in the destination folder).

Having selected both source and destination folders, upon clicking 'Go', QuickHash will hash the files in the source folder, then copy them to the destination folder (whilst concurrently reconstructing the folder 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 8 options that are unique to this tab that require explanation.

```
1. Just LIST Directories?
2. Just LIST sub-directories and files?
3. Save Results (CSV)?
4. Start at a time?
5. Ignore sub-directories?
6. Choose file types?
7. Don't rebuild path?
8. Copy hidden files?
```

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

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 folder and its sub-folders to simply be dumped in the root of the destination folder 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 folders. 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 and is provided due to filesystem restrictions.

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 folders are both, in essence, 'files' on those systems, so a hidden folder 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 folders 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 common difficulty 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. Neither is it 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.

3.1.5 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 choosing FileB secondly using the 'File' tab.
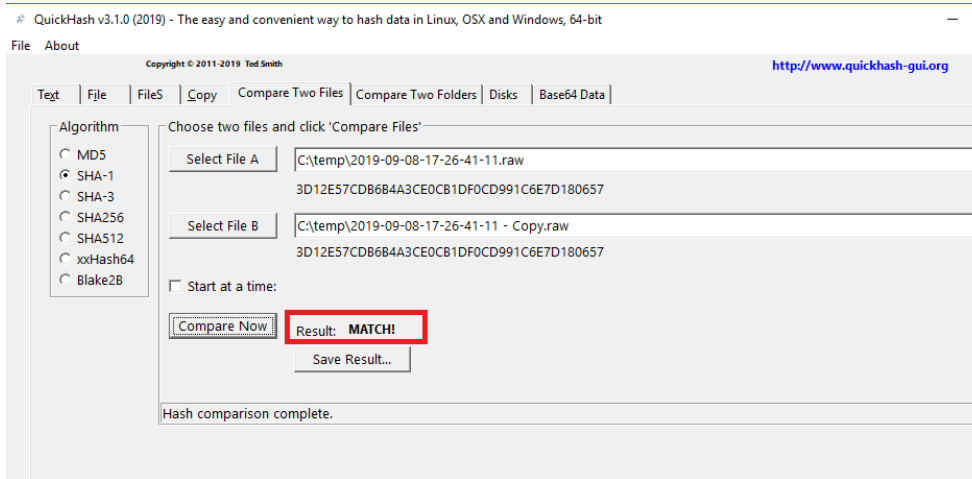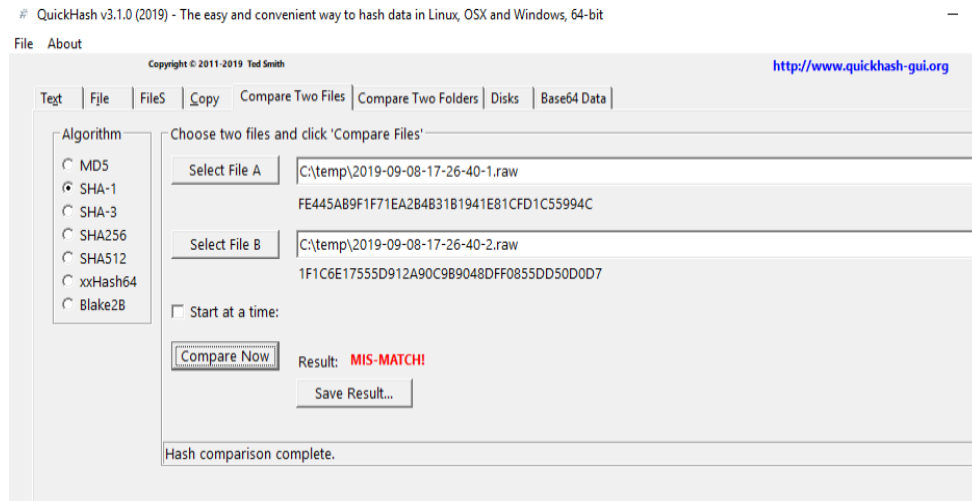
Illustration 13: Comparing the hashes of two identical and non-identical files

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

As of v2.8.1, if **the user single mouse clicks** the generated hash value, the hash value and the chosen algorithm with 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!

3.1.6 Compare Two Folders : This enables the user to compare the file content of two folders. The user must select one folder as the source (Folder A) and then a second folder to compare the source

against (Folder B). QuickHash will then count and hash all the files in Folder A and Folder B, storing the values in a list in memory. When complete, it then counts the number of files in each folder and then it hashes them all. By default, mapped drives are assumed and displayed in the treeviews. But if the user ticks "UNC Mode", it is possible to paste two UNC network paths instead. However, the user can not compare one local drive on a mapped drive, and a UNC network path. If that requirement exists, please temporarily mount the network path as a drive letter, and then use the non-UNC mode instead, and as is the default route. Then simply unmount the network path after analysis. *Note : if the program is run using Administrator rights, the mapped network drives may not show up in the treeview. If that happens, exit the program and launch it as a regular non-administrative user.*

The list of hashes is also held in memory where they are then compared very quickly.

However, the comparison system does require some understanding and is not to be taken always at face value. Generally speaking, if both hash lists match and the file count matches, it will report a match even if the file names are different. The functionality is designed to compare tow folders that are assumed to be the same, not assumed to be different. But a wealth of new features have been added to v3.3.0 to help identify variances where variances do exist (see below).

Note that filenames are not considered in this analysis for a hash match; it is the **content** of each file that is analysed. But, the display grid may confuse users because if a file is found in Folder A called "Bob.doc" and the exact same file (by hash) is in Folder B called "Alice.doc", a match will be reported (because the binary content is the same) but the display grid will show a blank field for each folder, for each of the two files, which may make the user think there is a mis-match. There isnt. It is simply stating that a file called Bob.doc was found in one location with one hash, and the same hash was also found in the other folder called Alice.doc, but a file called Bob.doc with the same hash was not found. This is designed to be helpful and to highlight the disparity whilst ensuring the hash values of each file are still listed. Ultimately the output is often better analysed using data analysis tools like Excel.

If you have 3 files called A.doc, B.doc and C.doc in Folder A and 3 files called D.doc, E.doc and F.doc in Folder B all with **exactly the same <u>content</u>**, this will be reported as a match, because the "hashable content" and file count is the same even if the filenames are not. **So this feature is not a comparison of <u>file names</u>. It is a comparison of <u>file count and content</u> between two folders**.

Users who are restricted to v3.2.0 can see the results shown in a display grid that has various right-click options. Note however that v3.2.0 had a bug that mis-aligned the rows if the count of one folder was greater than the other. A mis-match was still correctly reported, but the results, if saved, would be out of line. That was corrected in v3.3.0. Since v3.3.0, there is a myriad of new right click options in the display grid, to allow the user to check for mismatches based on name, hash, duplicate hashes, missing hashes, missing filenames, and more. See illustration below.
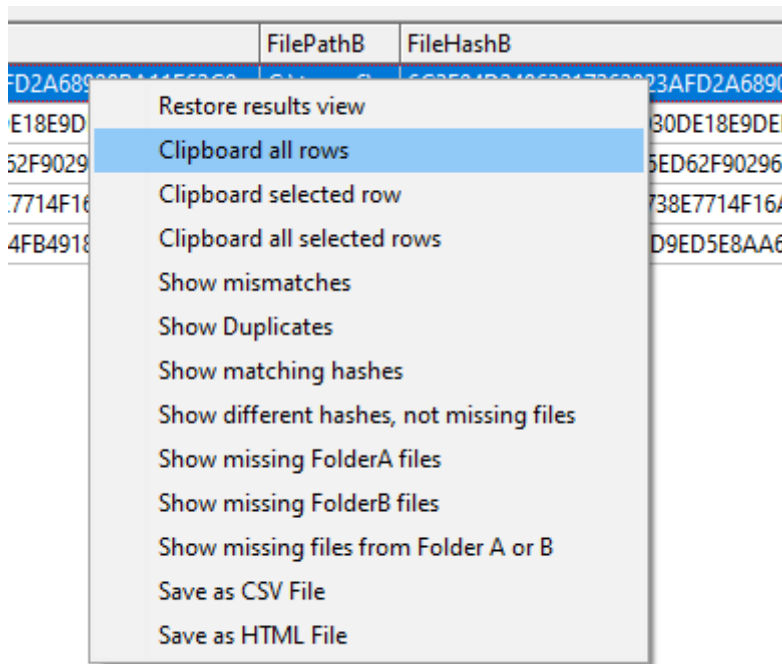
Illustration 14: The new right click menu of v3.3.0 for Compare Two Folders

If, however, there is a mismatch of hash even if the count is the same QuickHash then does go on to compare the hash of each file to work out which ones differ in Folder A to Folder B. The results are saved by default (unless the user unticks 'Log results?') to a text file and a more sensible results file can be saved by the user by right clicking and choosing "Save as CSV" or "Save as HTML File", as seen above.

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

Below are some screenshots that show the v3.3.0 Compare Two Folders functionality. Folder TestA had 100 files created in it, and TestB then contained a copy. Note that the filename is listed once, because its corresponding match was found in TestB (as is described above a little earlier in this section). Below is how it looks if both folders match :
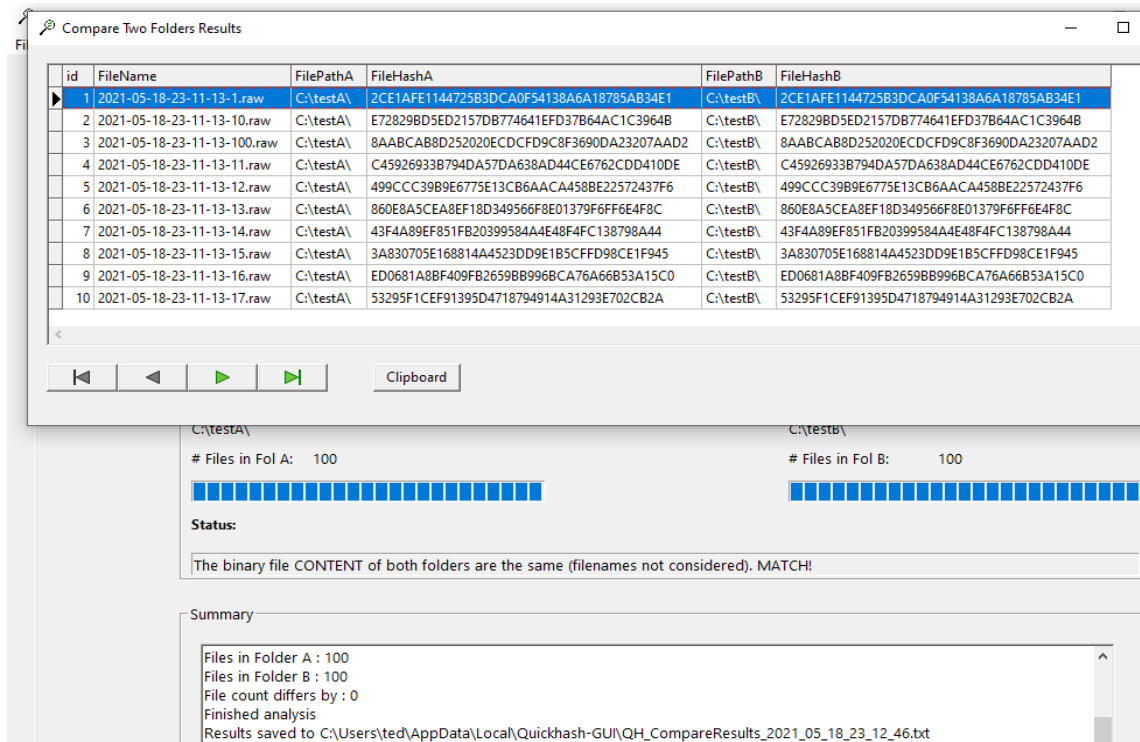
Illustration 15: A standard matching comparison of two folders.

Now lets change one file in TestB but the count of 100 files and the names remain the same:
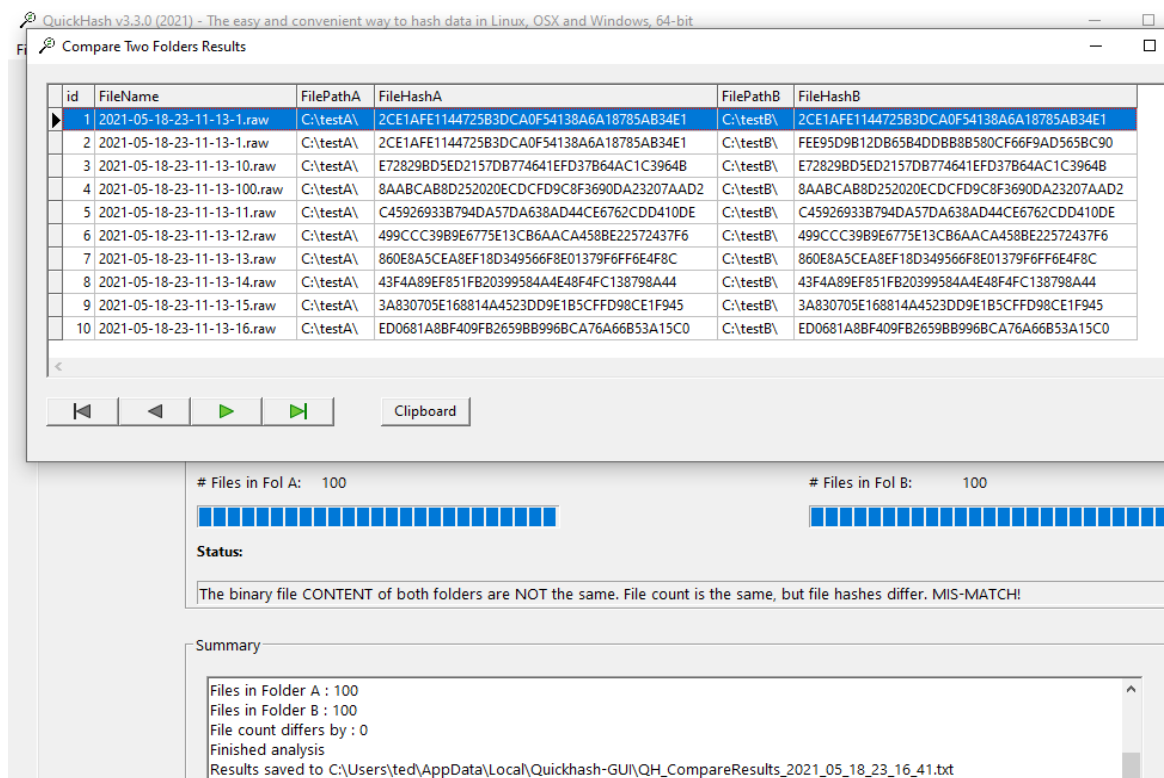
Now lets try and find the different file using the right click options:
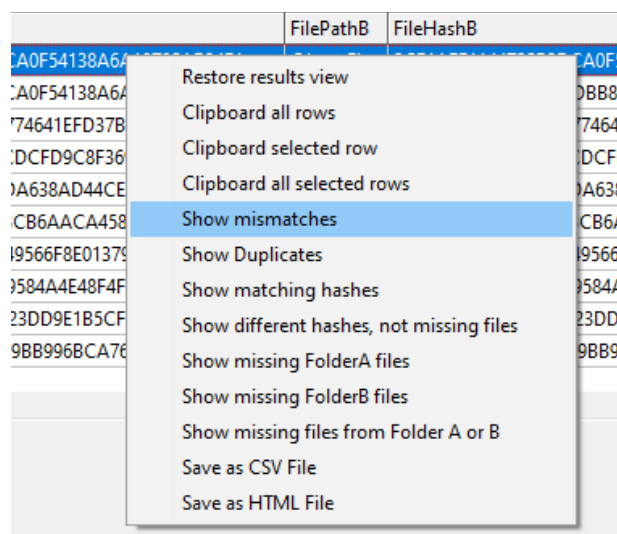
Now lets remove one file from TestB, and leave the other file that we changed as well.

Now lets see what is missing using a right click filter:

id	FileName	FilePathA	FileHashA	FilePathB	FileHashB
13	2021-05-18-23-11-13-2.raw	C:\testA\	B648C6DDBBE8D660353724DE4588C26D0CA9D96D		

...and round and round we go, so on and so forth.

It might not be perfect, but it is a fairly good system. Consider that the intention behind this tab is to help users confirm, or provide a best indication, that two folders match each other, perhaps after a data migration exercise or similar endeavour. Or that two folders are 99% similar with 1% variance. It is not designed to help users look at two folders that are 99% different, just to help identify the 1% that is the same. If that is your need, you need a file manager. Not a data hasher. Again, if this is your need, other tools may assist you better.

Upon completion, the log file is automatically saved to a location deemed safe on the users particular operating system. This path can vary, but the user is told in the Summary window at the bottom, where s/he is then able to navigate to it and open it with any text editor.

Please be aware that the "Compare Two Folders" functionality is extremely tricky to develop in a way that suits everyone. V3.3.0 has seen the biggest revamp of this section and has taken many weeks to develop to the stage that it is at in this version. If, despite all the many options that are now sported, it still falls short of your needs, then please use something that does suit your needs. There are many "folder comparison" tools; some commercial, some free. Directory Opus is one that is highly recommended, along with Beyond Compare and of course the Linux terminal. There will be many more choices. I really can't do much more than I have, and much of that was achieved with thanks to the open-source community and the various SQL ninjas out there.

3.1.7 Disks : This tab was only available in the Windows version prior to v2.7.0 but since then it has been available for both Windows and Linux users. Apple Mac OSX is 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 or logical volume of the computer (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:
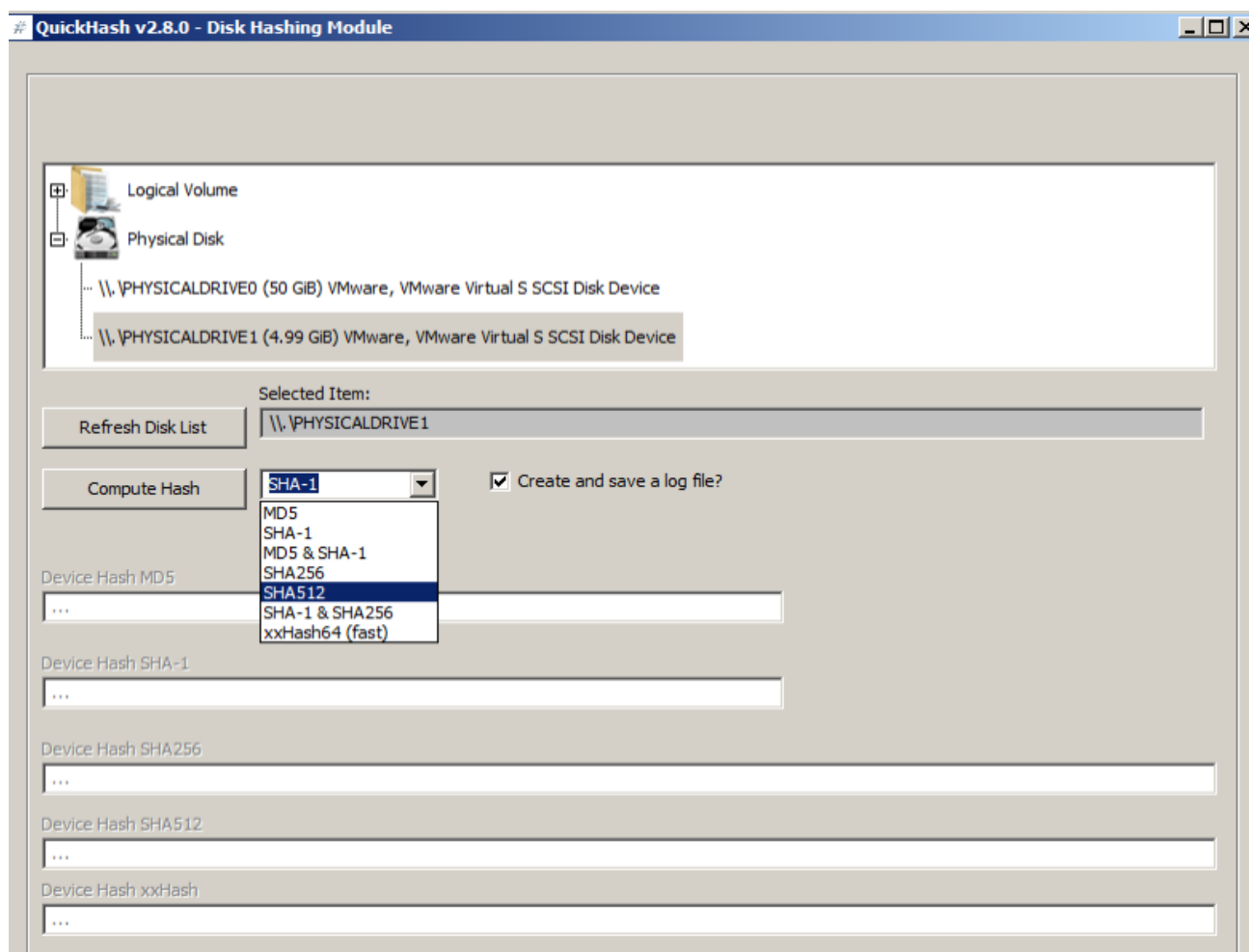

Illustration 16: The Disk Hashing Module of QuickHash v2.8.0

Since 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, but not generally maintained anymore (subject to change).

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 v3.2.0, it is possible to compute SHA-1, or MD5, MD5 & SHA-1 together, SHA256, SHA-1 and SHA256 together, SHA512, xxHash (32 bit on x86), xxHash64 (64-bit on x64) and Blake3.
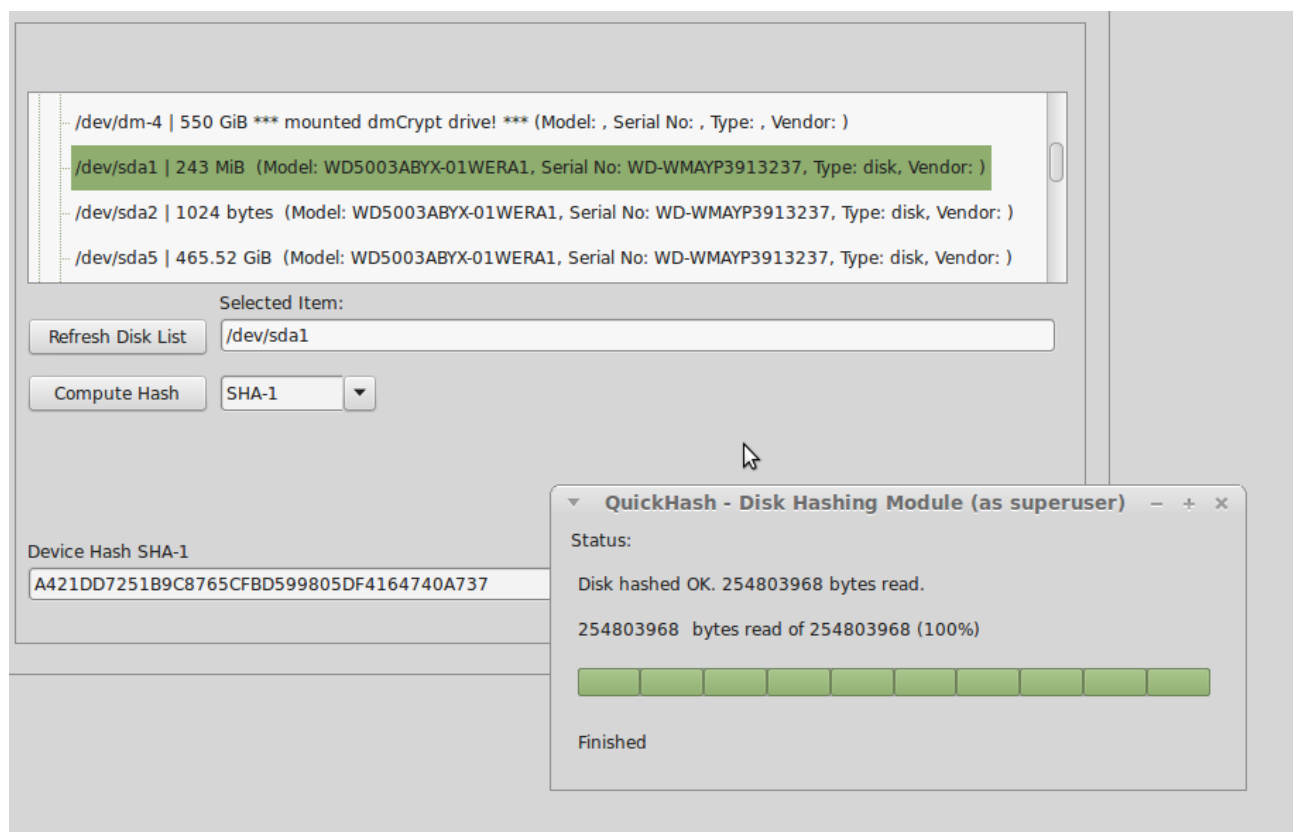

Illustration 17: Hashing a logical volume in Linux using QuickHash

If the user receives an error like "`could not convert variant of type (Null) into type Int64`", this is most likely due to the existence of an installed removable drive bay device (those that enable SD card reading, extra USB ports etc) which typically present, to MS Windows, a list of logical drives like "Removable Drive X:" in Windows Explorer, even if they are empty. The error appears therefore because these entries appear "as a valid drive", but one without

any disk capacity, and that causes the error. Attempts have been made, especially in v3.0.1 and especially in v3.3.0 when this module code was largely re-written, to reduce this, but it may still happen. It is very hard to debug because it is hardware specific.
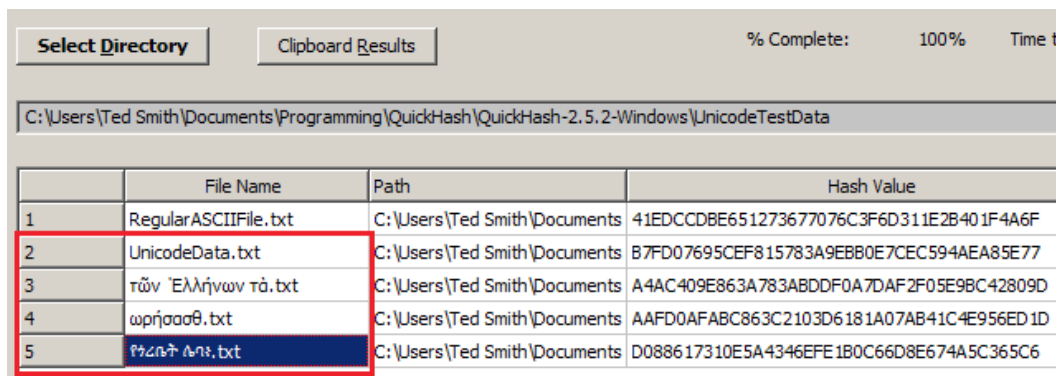
3.1.8 Base64 Data

New to v2.8.3, the Base64 tab allows the user to hash an encoded Base64 file AND generate a hash of it's <u>decoded</u> counterpart without the user having to create the decoded version first. This can be done one file at a time or the user can select a folder full of Base64 encoded files and QuickHash will generate encoded and decoded hash values of them all. The results are output to a display grid, from which the user can right click to copy single rows, all the rows, or save the whole grid.

There is a third button 'Decode and Save files...' which, if clicked, will ask the user for a folder of Base64 encoded files, and then QuickHash will ask for a second folder to put the decoded versions in. It will then decode all Base64 encoded files and save new decoded versions in the output folder. No hashing is done here. It's just a quick and easy way for users to decode their Base64 files without having to use online systems. If the users wishes to have the files hashed as well, either in their encoded or decoded form, then use the second option of the tab to compute those values as just described (the second button ('Decode and hash files')) or use the File tab to do one at a time (choose either the encoded file or decoded version if you've decoded it), or the FileS tab.

3.2 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 <u>Windows</u> versions prior to 2.3 were not Unicode aware*.

Illustration 18: QuickHash showing Unicode awareness on Windows

3.3 Open Files

Ever since v1.0 of QuickHash back in 2011, files that have been opened and not shared by the operating system have caused access denied errors with QuickHash. As of v3.0.1 however, that problem will hopefully have been reduced due to some improved exception handling. With v3.0.1 upwards, if the program is given a file that is open by another program, the program will simply return a hash value that reads "could not access file" and continue to the next file, rather than showing an error message and forcing the user to abort the program.

Hashing open files is not wise anyway as the hash will likely change when the file is closed, but in all previous versions the program simply could not bypass the open status and would crash. This solution is considered more favourable.

3.4 Libraries

v3.3.0 now ships with a subfolder, called 'libs'. For Windows and Linux, this must be kept in the same place where QuickHash is executed from. On Windows the folder contains the following DLL files :

The SHA-1 hashes for these for v3.3.0, as of May 2021, are :

libewf-x64.dll	5D33227712DA76316613DCD88B2749916DBA5ACA
libewf-x86.dll	915E3F26E170A062312A8CD73462AE6ECA6EF7BA
libgcc_s_dw2-1.dll	201924954A5A593C4CA24EE0FE799A764B41598D
libwinpthread-1.dll	34E84ED8F69F05FCAD212B02C2B064A5C7377904
sqlite3-win32.dll	0B25A2BA06DD5B8FE2A5F33C5C8442D4C12A2B70
sqlite3-win64.dll	2092405B3755C71E12E2F6EE4D193B321999CB62
zlib1.dll (32-bit copy)	B1D1FECBB568EDCF712232738BA3805B47BC6036
zlib1.dll (64-bit copy)	A10687C37DEB2CE5422140B541A64AC15534250F

Linux versions include libewf-Linux-x64.so file which is the 64-bit compiled SO file for the libewf library.

libewf-Linux-x64.so	2376C9092754ABF401CFA1D17C00801DAAB4D143

Note that all these are either new or updated for v3.3.0. Older versions of QuickHash will contain some library files with hashes that differ to those stated above.

3.5 The About Menu

For some years, there has been a standard About menu to show license statements, use of shared libraries, and acknowledge contributions from the open-source community. As of v3.3.0, however, there are two new menu options in the About menu. SQLite version will report the version of SQLite being used by QuickHash. And the Environment Checker attempts to conduct some due diligence around the libraries it expects to find on your chosen OS and it also displays some data about SQLite such as the name of the database file that is sitting behind the scenes, to make it easier for you to find it, if you need to. The results display will differ between operating systems.
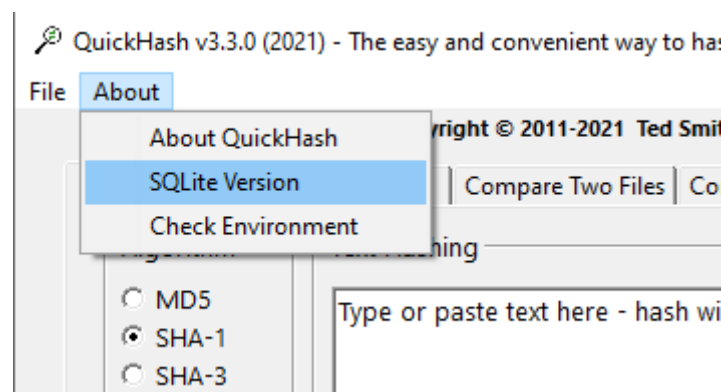

Illustration 19: New About menu options that come with v3.3.0

3.6 Other tools of the same or similar 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 some of 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 with very similar names. These include online and standalone executable tools as well as libraries.

It is important to point out, however, that QuickHash-GUI was the first graphical, standalone, free, open-source, Unicode aware and cross-platform data hashing tool that was named "QuickHash" and it was published on Sourceforge in 2011 at http://sourceforge.net/projects/QuickHash .

The pages for this program are www.QuickHash-gui.org (since late 2016) and Sourceforge at http://sourceforge.net/projects/QuickHash since 2011 up to 2016. You should not download it from anywhere else.

3.7 Donations

QuickHash is created in my spare time, which is very tight with a full time job and a family. It is generally written late at night, which can sometimes explain oversights and can be the cause of some bugs! You will also notice the website is ad free (I did try out ads for a few months but I think it just ruins the image).

If you value the program, or if your organisation, company or agency does, then please do consider making a donation using https://paypal.me/quickhashgui. Doing so helps fund the AWS server costs and also helps inspire future development. Case studies are also of interest...if QuickHash has helped you with a major task, or achieve anything of importance then do please consider submitting a testimonial for the website. Both methods are a good opportunity to get your company listed on the website of one of the worlds most widely used data hashing tools; http://www.QuickHash-gui.org.

To donate : www.paypal.me/quickhashgui