

# TZWorks® Disk Utility & Packer (*dup*) Tool Users Guide



## Abstract

***dup*** is a command-line tool that can pull out disk statistics, analyze Master Boot Records, and copy disk sectors, volumes, folders and files. Primarily for Windows, but has functionality for Linux and macOS as well.

Copyright © TZWorks LLC

[www.tzworks.com](http://www.tzworks.com)

Contact Info: [info@tzworks.com](mailto:info@tzworks.com)

Document applies to v0.48 of ***dup***

Updated: May 5, 2024

## Table of Contents

1	Introduction .....	3
2	How to Use <i>dup</i> .....	3
2.1	Drive Analysis .....	5
2.1.1	Master Boot Record (MBR) Scan.....	5
2.2	Imaging Disks .....	6
2.2.1	Using Multiple Threads to Speed up Imaging .....	7
2.2.2	Imaging options.....	7
2.2.3	Concatenating Image Segments .....	8
2.2.4	Working with Encrypted Disks .....	8
2.3	Copying Volume Snapshots.....	8
2.4	Copying Files .....	9
2.4.1	copyfile .....	9
2.4.2	copydir .....	12
2.4.3	copygroup .....	13
2.4.4	Filtering on specific files.....	16
2.4.5	copyscript .....	16
2.5	Working with Linux and macOS – <i>pulling artifacts</i> .....	18
2.6	Copying Metadata and Logging .....	19
2.6.1	Extracting INDX data from the Folders .....	20
2.6.2	Extracting Spotlight Metadata .....	21
2.6.3	Metadata table.....	21
2.7	Copying files from a ‘dd’ image.....	22
2.8	File Utilities.....	23
2.8.1	untar.....	23
2.8.2	expand (or uncompress) .....	23
2.8.3	compress .....	23
2.8.4	merge .....	23
2.8.5	combine_files .....	24

2.8.6	Hashing Options .....	24
2.8.7	Logging Options .....	24
3	<i>dup</i> Library Dependencies .....	25
4	Available Options .....	25
5	Authentication and the License File.....	29
6	References .....	30

# TZWorks® Disk Utility & Packer (*dup*) Users Guide

---

Copyright © TZWorks LLC

Webpage: [http://www.tzworks.com/prototype\\_page.php?proto\\_id=37](http://www.tzworks.com/prototype_page.php?proto_id=37)

Contact Information: [info@tzworks.com](mailto:info@tzworks.com)

## 1 Introduction

**dup** is a command-line tool that was designed to be used for incident response in going after master boot records, raw clusters or image a portion of the drive. Added to the functionality of the tool, is the ability to copy files and/or folders. If targeting NTFS type volumes, the tool can use its NTFS engine to copy files that are locked down by the operating system by accessing their underlying data clusters. Originally architected for Windows, the tool also has compiled versions for Linux and macOS. The tool makes use of the *zlib v.1.2.11* library from *Jean-loup Gailly* and *Mark Adler* for general compression.

This tool should be considered prototype, until more community testing has been done to exercise different hard drive configurations.

## 2 How to Use *dup*

The first category of options is to gather disk statistics. One can use the **-scandrives** option to enumerate all the disks attached to a computer or to target a specific disk via the **-diskstats <drive number>** option. The report generated shows the drive and the associated partitions, listed by disk offset and size.

The second category is for extracting and analyzing the MBR. With the advent of malware using the boot record as an injection vector as well as a persistence mechanism to compromise your computer, analysis of the MBR should be part of a normal investigation. The **dup** tool has some basic capabilities to assist in this area. It allows one to output the MBR for quick analysis (**-mbr**) or compare the content of the MBR with a known good version (**-mbr\_compare**). The quick analysis functionality will look for abnormal branches from the MBR code to other locations. This simple behavior is indicative of a number of malware techniques.

The third category is for disk and volume imaging, or specific sections of a drive/volume. While there are many other tools available to do this, adding this functionality to **dup** was something that could be added with minimal code.

The fourth category, and in some cases, duplicative to *ntfscopy* (from TZWorks) is the ability to copy files. Different than *ntfscopy*, **dup** offers one the ability to: (a) target files in volumes that are not NTFS,

(b) collect groups of files, (c) use a script to specify custom sets of files, and (d) the ability to compress the results during the copy operation. *ntfscopy*, on the other hand, focuses only on NTFS volumes; and within the NTFS volume, on the internals of the file that is copied, and therefore, offers the option to extract the metadata for each file copy. So, while there is overlap between these two tools, there are areas of *file copy* functionality specific to each tool. The target audience for the **dup** tool is geared for the incident responder, allowing the responder to quickly collect and gather specific groups of files in a seamless way. Below is screenshot showing the various options with syntax.

```
Usage:

Disk stats
-scan drives
-scan mountpts
-diskstats <#>
-scan drives
-imagestats <dd image path/file>
-vmrk "file1 | file2 | .."

MBR stats/data
-mbr [-disk <#> | -file <MBR file>]
-mbr_compare -disk1 <disk#1> -disk2 <disk#2>
-mbr_compare -disk1 <disk#> [-file <MBR expected>] -image <dd img>]

Image drive or volume
-copydrive <drv #> -out <dst> [image options]
-copyvolume <vol letter> -out <dst> [image options]

[image options]
-offset <byte offset>      = rounded up to a sector boundary
-size <# bytes>            = rounded up to a sector boundary
-read_timeout <# secs>     = 2 secs or above [only for bad drives]
-retries <num>             = # attempts after cluster read failure
-gzip                     = compress the output
[-md5 | -sha1]             = compute MD5 or SHA1 hash of the output
-force_volread             = *** only avail for [-copyvolume]

Copy Volume Shadow [raw data]
-copysnaps <vol or image> -out <dir> [-gzip]

Copy files from a volume (** means experimental)
-copyfile [<file1|file2|..> | -pipe] -out <dst> [file opts]
-copydir <folder> -out <dst> [-level <#>] [file opts]
-copyscript <script> [file opts]
-copygroup [grp switches] -out <dst> [file opts] [-vss <#>]

[file/folder copy options]
-image <image file>        = only for unmounted raw NTFS 'dd' images
-filter <*"partial|*.ext"> = filter files
-filter_sqlite             = *** filter on SQLite files
-filter_plist              = *** filter on plist files
-filter_esedb              = *** filter on ESE db files
-filter_start_date <yyyy-mm-dd> = *** files at/greater than date
-filter_stop_date <yyyy-mm-dd> = *** files at/less than date
-ntfsraw                  = *** force raw cluster reads
-ads_rename               = for ADS replace colon with underscore
-gzip                     = compress results
-incl_indx                = *** incl INDX data (NTFS only)

[-copygroup switches (for Windows)]
-pull_sysfiles            = pull $MFT, $UsnJrnl:$J files
-pull_reg_hives           = pull reg hives
-pull_evtlogs             = pull event and other logs
-pull_lnk                 = pull LNK files and Jumplists
-pull_pfs                 = pull prefetch files
-pull_systrash            = pull recycle bin from system vol
-pull_userdbs             = pull various user acct dbs
-pull_browsers            = pull browser artifacts
-pull_all                 = pull all the above
-image <path to image>    = only for unmounted raw NTFS 'dd' images
-sysvol <path to mountpt> = only for mounted volumes understood by OS

[still experimental (for Windows)]
-pull_livestats           = pull network and process/task stats

[-copygroup add volume shadow copy options]
-vss <index>              = target files in specified Vol Shadow
-vssall <partition>       = target files all Vol Shadows in partition

File Utilities (** means experimental)
-expand <file> -out <dst> = decompress file that that used -gzip
-merge -fromfile <f1> -tofile <f2> -tofile_offset <#> [-skip_null_sectors]
-combine_files <file1|file2|...> -out <dst file> = appends to dst file
-md5 [<file> | -pipe]     = compute md5 hash of file or files
-sha1 [<file> | -pipe]    = compute sha1 hash of file or files
-sqlite_log <file>        = *** creates a SQLite log file
-wipefile <file>          = *** erase file
-wipedir <folder>         = *** erase dir/all subdirs
```

Finally, the fifth category was added to aid in the reconstruction of partial images/files and/or resulting collections. The reconstruction option **-merge** is used to take a partial image and merge it to a larger image. This becomes useful when trying to image a drive with bad sectors which may result in getting a piecemeal set of images across sections of the drive. Using the **-merge** option, one can merge the partial fragments to their proper offset and create a semi-complete replica of a bad drive. The other combine file option, allows one to take a number of files and concatenate them together into a final output file. The **-untar** and **-expand** options allow one to use the **dup** tool to undo any tar (tape archive) packing and/or **gzip** compression. Finally, the **-md5** and **-sha1** invoke common hashing algorithms that allow one to either operate on a single file or to pipe in a directory of files to operate on.

## 2.1 Drive Analysis

One can use the **-scandrives** to pull statistics about each of the drives attached to a computer. Below are some of the fields extracted during this operation. Listed below are the various partitions for 3 disks. The disk number is annotated in the first column. The first two disks (drive 0 and 1) have a MBR (Master Boot Record), while the third (drive 2) uses a GPT (GUID Partition Table). Using the data from this table would be a useful place to start prior to imaging a drive or volume. If one wanted to analyze just the gaps of sectors that are outside the partition boundaries, this is easily done by looking at the last two columns and locating entries listed as *unknown* for partition type description and/or if the volume signature is null.

drive_num	disk_guid	disk_sig	type	start_offset	end_offset	num_bytes	vol_guid	vol_sig	format	letter
0	c4f8d4d3-d3ee-4697-aa9a-959f79719e35	775b-a257	GPT	0x1000000	0x064fffff	0x06400000	15963782-6536-4b82-9348-8a8adcd6c6d0		0xee	
0	c4f8d4d3-d3ee-4697-aa9a-959f79719e35	775b-a257	GPT	0x06500000	0x074fffff	0x01000000	f1385ed7-6b4f-4b38-9a0b-f2432e5550d2		0xee	
0	c4f8d4d3-d3ee-4697-aa9a-959f79719e35	775b-a257	GPT	0x07500000	0xe8cf1fffff	0xe8c7d00000	d66c534f-dcbb-4f6d-a4a8-6a1a8245c698		0x07	C:
0	c4f8d4d3-d3ee-4697-aa9a-959f79719e35	775b-a257	GPT	0xe8cf200000	0x1d1507fffff	0xe881600000	728cd83-9855-4d5d-92d7-9438f8b4a6c6		0x07	E:
1	00000000-0000-0000-0000-000000000000	7c69-c138	MBR		0x00	0x01000000	00000000-0000-0000-0000-000000000000		0x00	
1	00000000-0000-0000-0000-000000000000	7c69-c138	MBR	0x01000000	0xee8ffffd	0xee7fffe0	67d66c8b-f17e-11e8-b983-185680762356	fcce-c14a	0x07	D:
1	00000000-0000-0000-0000-000000000000	7c69-c138	MBR	0xee8ffffe0	0xee8fffff	0x0200	00000000-0000-0000-0000-000000000000		0x00	

### 2.1.1 Master Boot Record (MBR) Scan

If desiring to review the MBR, one can quickly scan and extract it using the **-mbr** option. This option does a quick look analysis of the master boot data, and determines if any strange jumps occur. The output includes a hex dump of the MBR data along with the primary partitions. Instead of showing a normal Master Boot Record (MBR), the example below is from a MBR infected by a *MBR Bootkit*. There are a number of these kits in the wild and their purpose is to infect a MBR to gain a foothold on a target computer. One of the reasons this is popular for malware authors is because of the location of the MBR. Since it is kept outside of the Windows partitions/volumes, it is relatively easy to write to this location (eg. Windows allows writes to disk locations outside the partitions without too much difficulty). Also, since the code in the MBR is run as your computer starts up (before Windows), it makes a great place for a virus or rootkit to hide. Even if you reinstall Windows or format the partition in question, a virus infecting the MBR will not be deleted. Other steps need to be taken to restore an infected MBR.

So, after one reinstalls Windows, the computer will still run that same infected MBR code which then can re-introduce a rootkit into the new installation of Windows. Below is an example of one of these infected MBRs and how **dup** reports the data it finds.

```
"cmdline: dup64 -mbr -file e:\testcase\mbr\mbr-inst.bin"
```

```
MBR raw
0000 0000: 33 c0 8e d0 bc 00 76 eb 69 3e d8 be 00 7c bf 00 3....p.i....|..
0000 0010: 06 b9 00 02 fc f3 a4 50 68 1c 06 cb fb b9 04 00 .....Ph.....
0000 0020: bd be 07 80 7e 00 00 7c 0b 0f 85 0e 01 83 c5 10 ....~.|.....
0000 0030: e2 f1 cd 18 88 56 00 55 c6 46 11 05 c6 46 10 00 ....V.U.F...F..
0000 0040: b4 41 bb aa 55 cd 13 5d 72 0f 81 fb 55 aa 75 09 .A..U..}r...U.u
0000 0050: f7 c1 01 00 74 03 fe 46 10 66 60 80 7e 10 00 74 ....t..F.f'.~.t
0000 0060: 26 66 68 00 00 00 00 66 ff 76 08 68 00 00 68 00 &fh....f.v.h..h.
0000 0070: aa 55 8e c0 8e d8 fb fc be 00 7c bf 00 06 b9 00 .U.....|.....
0000 0080: 01 f3 a5 b8 88 06 50 c3 b8 02 02 bb 00 7a b9 1a .....P.....Z..
0000 0090: 00 cd 13 b9 00 04 bb 00 7a 8a 07 34 76 88 07 43 .....z..4v..C
0000 00a0: e2 f7 be be 07 bf be 7d b9 40 00 f3 a4 b8 00 7a .....}.@....z
0000 00b0: 50 c3 e4 8a 56 00 cd 13 5d eb 9e 81 3e fe 7d 55 P..V...].>..}U
0000 00c0: aa 75 6e ff 76 00 e8 8d 00 75 17 fa b0 d1 e6 64 .un.v....u.....d
0000 00d0: e8 83 00 b0 df e6 60 e8 7c 00 b0 ff e6 64 e8 75 .....|.d.u
0000 00e0: 00 fb b8 00 bb cd 1a 66 23 c0 75 3b 66 81 fb 54 ....f.u;f..T
0000 00f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

0000 0100: 6e 67 20 73 79 73 7. 65 d d 4d 69 73 73 6. 6e ..g system...ss..
0000 01a0: 67 20 6f 70 65 72 61 74 69 6e 67 20 73 79 73 74 g operating syst
0000 01b0: 65 6d 00 00 00 63 7b 9a 54 cf 4d 7c 00 00 80 20 em...c{.T.M|...
0000 01c0: 21 00 07 df 13 0c 00 08 00 00 00 20 03 00 00 df !.....
0000 01d0: 14 0c 07 fe ff ff 00 28 03 00 00 d0 fc 03 00 00 .....(.....
0000 01e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000 01f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 55 aa .....U.
```

During analysis, found suspect location  
 0x0007 : eb 69

JMP 69h relative bytes

disk sig	type	start offset	end offset	num bytes	vol sig	type	description
7c4d-cf54	MBR	0x000000000000	0x00000000ffff	0x000000100000	0000-0000	0x00	unkn
7c4d-cf54	MBR	0x000000100000	0x00000064ffff	0x000000640000	0000-0000	0x07	ntfs
7c4d-cf54	MBR	0x000000650000	0x0000007fffff	0x0000007f9a0000	0000-0000	0x07	ntfs

In the above example, **dup** locates a jump instruction at a location in the MBR that is abnormal. In this case, the hex code is "eb 69" which translates to a relative jump instruction of 69 hex bytes. While these types of signature-based scans can be useful, it is more robust and reliable if one could compare the MBR to a known good copy.

For companies with a standard base image of their computers, one can extract the MBR from this known good image and use that to compare it the MBR. Comparing a known good MBR with an unknown MBR can be easily done with any reasonable hex editor. If desiring to do this on a live box, one can use **dup** via the **-mbr\_compare** option, and point to the disk index wishing to analyze with the file containing the known good MBR.

## 2.2 Imaging Disks

There are many 'portable' disk imaging tools available for forensic use for the investigator today. We decided to include this option to **dup** as well, primarily to address some of the shortfalls of disk imaging when encountering disks with bad sectors. Bad sectors are typically caused by a disk head crash, where the head makes contact with the platter of a disk. On occasion, when encountering unrecoverable bad sectors, certain imaging tools will just stop the imaging process. So, with **dup**, we added the ability for the tool to 'try' to continue on. To help with that process, we introduce a sub-option **-read\_timeout** where the argument is the number of seconds to wait before trying the next sector. Since this sub-



option will alter the machine's registry setting for all disk reads, **dup** will restrict the minimum allowable setting to 2 seconds, since going to a lower value may impact read operations for other applications. If this option used, **dup** will store the original setting before changing it, so it can automatically *try* to restore it back to the original after the imaging operation is completed.

Disk sectors that are considered 'unreadable' usually occur in groups and are clustered where the disk head made contact with the platter; therefore, one can take other steps to try to image a disk around these bad sectors. One can focus on targeting specific volumes or locations that are not too affected. Imaging a drive in this way will inevitably result in a handful of segmented files, each with multiple partial images of different sections of the drive, and in some cases, overlapping sections. To help in the reconstruction of these partial images of the disk, one can use the **-merge** option, which allows one to merge one file into another file forcing the merge to be at a particular offset. In this way, the final image can be constructed so the data contents are at the same relative offset as the original drive. For any gaps that are present, the merge operation will set these gaps to zero. This will allow the segmented files, after the merged operation, to be at the proper locations relative to each other.

The two basic options for imaging include **-copydrive** and **-copyvolume**. The first option requires one to pass in the drive number, which the second option requires one to pass in a volume letter. Both options can be tuned in the sense, one can pass in the starting offset (via **-offset <byte offset>** to start imaging from) and the number of bytes to copy (via the **-size <# bytes>**). If one wishes to compress the imaging results on the fly, one can use the **-gzip** option. If one wants to compute MD5 hash or SHA1 hash, one can use the **-md5** and **-sha1** options respectively.

### 2.2.1 Using Multiple Threads to Speed up Imaging

Some of the imagers on the market use multiple threads to speed up the imaging process. **dup** does the same thing. Depending on the options used for imaging, **dup** can spawn 2 to 9 threads to speed up the imaging process. For example, if using a USB-3 connection to image a drive that is relatively fast, one should see a range of 13 up to 17 GBPS speeds. Ultimately the speed will depend on how fast the read I/O is in conjunction with the CPU/memory dedicated to the imaging routine and what additional options were chosen for imaging. For example, using **gzip** compression would slow down the imaging process some. **dup** tries to minimize the amount of memory used during the imaging operation while trying to maximizing the imaging speed.

### 2.2.2 Imaging options

The imaging format available to **dup** does not allow for images in *EO1* or *AFF* formats, but only in the raw 'dd' format. One can compress the 'dd' image by using the **-gzip** option. The compression uses the **zlib** library and will make use of multiple threads to process the raw data in parallel chunks to minimize the processing image of the compression. If an *MD5* or *SHA1* hash is desired during the imaging process, one can also add either the **-md5** or **-sha1** syntax to the command sequence. At the end of the imaging process, **dup** creates a summary file containing the metadata of the device that was imaged. The



filename of the metadata file will be the same name of the raw image created with the extension “meta.txt”.

### 2.2.3 Concatenating Image Segments

For the case where multiple files are generated during an imaging process, either with this imager or with another imager, one can concatenate multiple files together. This is done with the tool’s **-combine\_files** option. This option takes as an argument a quoted argument that can contain multiple filenames that are delimited by a pipe character. The order of the file listing defines the order of how the files are concatenated together; where the first file is the start of the image and the last file is the end of the image. Note: the **-combine\_files** option is strictly for concatenation, whereas the **-merge** option discussed previously is for more complicated file operations where one may need to overlap files fragments or add fragments where there is a gap between them.

### 2.2.4 Working with Encrypted Disks

The **dup** tool uses *disk* semantics for image a disk or volume. What this means is it blindly records the actual bits on the disk, which is what we want; typically, the analyst does not want to translate the data, but preserve the actual raw data.

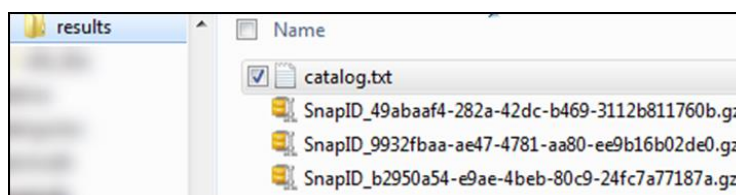
Sometimes, however, this may or may not be what is desired. For example, if the desire volume of data is *encrypted* and if one wanted to image a volume in an *unencrypted* format, one could tell **dup** to use *volume* semantics instead. This allows **dup** to perform reads that get translated by the operating system such that if the underlying data was *encrypted*, the resulting data from the read operation would be *unencrypted*. To do this one would use the option **-force\_volread**. This option is only available for **-copyvolume**.

## 2.3 Copying Volume Snapshots

This is a variation of imaging a volume, in the sense it analyzes each *cluster run* of the Volume Snapshot’s that are present on the volume. It then copies each *cluster run* associated with each Snapshot. *Note: this function does not try to reconstruct clusters from outside the Snapshot cluster run.* The command requires one to point to the desired volume letter or ‘dd’ image. The command syntax is:

```
dup64 -copysnaps e:\images\win8.dblake.vhd -out results -gzip
```

The output produced will be a series of compressed volume snapshots along with a metadata file with the name of “catalog.txt”.



The metadata file will contain statistics about each of the snapshots archived, including creation timestamp, the associated GUIDs, NTFS internals about the snapshot, as well as other information.

```
VSS Store ID:      8dfc93b3-376f-11e3-be88-24fd52
Volume Size:      0xd11200000
Create Time [UTC]: 10/18/13 00:28:29.120
Store HdrInfo Offset: 0x77c52c000
Store Blklist Offset: 0x77c530000
Blkrange Offset list: 0x77c534000
Bitmap Offset:    0x77c6d0000
MFT & sequence#:  0x70000000006034
Allocation Size:  0x000000000
PrevBitmap Offset: 0x77c748000
Snapshot ID:      49abaa4f-282a-42dc-b469-3112b8
Snapshot Set ID:  55d65678-4826-4c27-aea2-34f8d
Type:             0x0000000d
Provider:         0x00000001
Attribute Flags:  0x0042000d [persistent; client
OS Machine:      Bifrost
Service Machine: Bifrost

VSS Store ID:      dc8ffcf4-3a6b-11e3-be8a-24fd5
Volume Size:      0xd11200000
Create Time [UTC]: 10/21/13 19:24:50.879
Store HdrInfo Offset: 0x8365e4000
Store Blklist Offset: 0x8365e8000
```

## 2.4 Copying Files

Targeting a group of files for copying during an incident response can range from easy to hard depending on the tools available to you. Further, if the target machine can be analyzed while it is running, may be viewed as a plus for the client. Doing so, however, will depend on whether the tools you have available can copy files that have been locked down by the operating system. Further, if one has a scripted set of instructions of what files need to be copied, then it can be much easier and less error prone. If the script is not general purpose and cannot handle the differences between WinXP, Win7 or Win10, where some of the artifacts are at different locations, then there is room for error. With **dup**, the tool tries to make the process very simple by detecting the Windows version it is copying from so as to target the proper location for certain artifacts. For example, **dup** knows how to distinguish between the differences in event log directories or user directories that have changed from WinXP to newer Windows versions. Then by using certain key variables, such as **%usersbase%** or **%eventlogs%** within the path, **dup** can figure out the proper directory to analyze.

The file copy operations include: **-copyfile** to copy a single or multiple files, **-copydir** to copy an entire folder and subfolders, **-copyscript** to copy files based on a script, and **-copygroup** to copy common artifacts of files.

### 2.4.1 copyfile

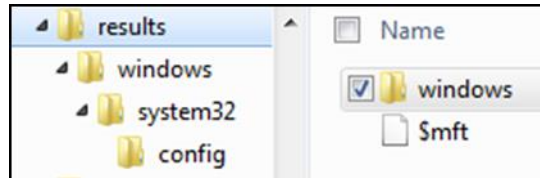
If wanting to copy one or more files, one can use the **-copyfile** option. The argument is the file you wish to copy. The output is specified by the **-out <dst>** syntax. This will be the destination folder where the copied results will be placed. If the destination folder is not present, it will be created.

In addition to copying the specified file, the context of the path of the source file is preserved as well, since this path will be recreated at the destination folder. Below are two examples of copying a file from the root directory and from a nested file in a number of subdirectories.

```
dup64 -copyfile c:\$mft -out c:\dump\results
```

```
dup64 -copyfile c:\windows\system32\config\sam -out c:\dump\results
```

If one looks at the `c:\dump\results` folder, one will see the `$mft` file and a directory structure that corresponds to the `windows\system32\config` subdirectory with the `sam` hive copied there.



If one desires just to collapse the directory structure and copy the file into a single results file, one can use the **-tar** syntax. This option behaves similarly to the `tar` utility in *Unix/Linux*, in the sense that it creates a single file containing the subdirectory information along with the files in those subdirectories. If this option is invoked, **dup** looks at the destination argument passed in to **-out <dst>** and uses that as the `tar` file name by appending `.tar` to the name. Below is an example using this option, and the resulting output file.

```
dup64 -copyfile c:\windows\system32\config\sam -out c:\dump\results2 -tar
```



When using a tool that can look into the internals of the `tar` file, one will be able to see the path, file and any other metadata associated with the file.

Name	Type	Modified	Size	Ratio	Packed	Path
sam	File	2/10/2017 4:56 PM	262,144	0%	262,144	windows\system32\config\

The purpose of these examples is to illustrate that depending on the options used, the behavior of the argument used in **-out <dst>** changes. In the first set of examples, the argument is used as a destination folder. In this last example, the argument is used as the filename.

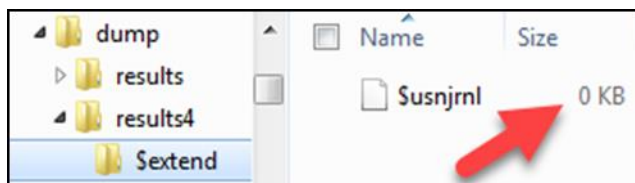
If one desires to copy multiple files, one can do this as well, by specifying the multiple files separated by a pipe `|` character. Below is an example of copying the `$mft` and `$boot` files in one command and the `results3.tar` file.

```
dup -copyfile "c:\$mft|c:\$boot" -out results3 -tar
```

Name	Type	Modified	Size	Ratio	Packed	P
Sboot	File	2/10/2017 5:10 PM	8,192	0%	8,192	
Smft	File	2/10/2017 5:10 PM	223,608,832	0%	223,608,832	

For those cases where you are targeting a file with an alternate data stream (ADS), you have two options. The default, or do-nothing option, will copy the alternate data stream as an ADS file, which means you will need ADS aware tools to actually see the named data stream that was copied. The other alternative is to use the **-ads\_rename** parameter, which replaces the colon used in the ADS name with an underscore, which allows the user to see the copied file without any special tools.

The typical use-case where this is an issue, is with the ADS **\$J** in the file **\$Usnjrnl**. Below is an example using the default (do nothing option) and how the final results are displayed. Notice in the screenshot the file size is zero, but this does not represent the size (or presence of) the alternate data stream of **\$J**.



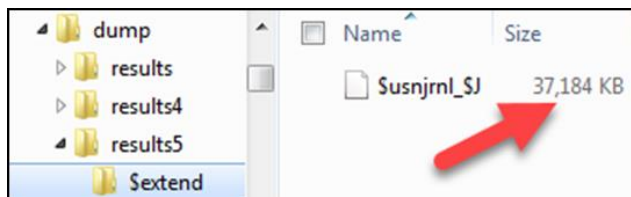
When looking at the results with the Window's built-in **dir** command along with the **/r** option, one can see the alternate data stream and the size of its data.

```
C:\>dir /r c:\dump\results4\sextend\susnjrnl
Directory of c:\dump\results4\sextend
02/10/2017  05:25 PM           0 $usnjrnl
                  37,552,128 $usnjrnl:$J:$DATA
               1 File(s)           0 bytes
               0 Dir(s)  927,531,835,392 bytes free
```

The alternative to the default option shown above example is to use the **-ads\_rename** parameter. Issuing the same command above using this parameter, yields the following:

```
dup64 -copyfile c:\sextend\susnjrnl:$J -out c:\dump\results5 -ads_rename
```

Notice the filename displayed is the actual alternative data stream that was specified as a normal file.



The above example brings up another built-in behavior within **dup**. When **dup** encounters files that have sparse data, the sparse data may or may not be copied. For example, if the copy operation requires *direct cluster access*, such as when **dup** needs to bypass the file system security because the file

is locked by the operating system, **dup** only copies the data that is backed by clusters, since by definition sparse clusters are not backed by physical disk clusters. Alternatively, if the copy operation does not require *direct cluster access*, which means **dup** can make use of the normal filesystem, then it will copy the size provided by the filesystem. In this latter case, the filesystem will return zeros for the sparse data.

If desiring to copy many files, one can use standard input (STDIN) and pipe in the files into **dup**. One can invoke this behavior with the **-pipe** option. This will tell **dup** to take the filenames received by STDIN and copy them to the results directory specified. One can use the **-filter** option in conjunction with the **-pipe** command, which allows one to further refine the files to operate on. (See the examples in the **copydir** section).

When using STDIN for input, one needs an external directory enumerator so that filenames populate the input stream so that **dup** can detect the passed in names and copy them. A good directory enumerator for Windows is the built-in **dir** command using the sub-options **/b /s /a**. These sub-options tell the **dir** command to: pull out the *bare absolute* path/file, *traverse subdirectories* and pull files with any attribute (like those that may be hidden).

```
dir c:\users\*.lnk /b /s /a | dup64 -copyfile -pipe -out c:\dump\results
```

The final option available is the compression option, via the **-gzip** syntax. The Windows build of the **dup** will have the **zlib** library statically compiled into the binary. What this means is, no special shared libraries will need to be present on the target machine for **dup** to invoke the compression routines. The compression library produces the results into a **gzip** compatible format.

## 2.4.2 copydir

Moving forward from copying one or a couple of files, one can also target a whole directory to perform a copy operation. The **-copydir** option takes a source folder as a parameter. To control how deep one traverses to child subdirectories, one uses the **-level** option along with an integer value. This value simply tells **dup** how many subdirectories deep to consider. The **-out** option operates the same way as discussed previously with the *copyfile* use-case.

To provide more control, one can use the **-filter** option to specify various filename and/or extension patterns to include in the copy operation. Below is an example of copying the user hives from the various user directories.

```
dup64 -copydir c:\users -level 5 -filter "ntuser.dat!usrclass.dat" -out c:\dump\results
```

This command will: (a) traverse 5 subdirectories starting with the *c:\users* base directory, (b) locate any files named "*ntuser.dat*" or "*usrclass.dat*", and (c) copy the files, along with their subdirectory context, to the *c:\dump\results* folder.

There are a couple of options that provide for a deeper filtering: (a) to filter SQLite files use **-filter\_sqlite**, (b) *plist* type files (targeting macOS) **-filter\_plist**, (c) ESE database type files (targeting Windows) **-filter\_esedb** and (d) a generic, user-defined filter which relies on the user specifying a set of hexadecimal bytes for the file signature **-filter\_sig**. These filter options allow work by inspecting the contents of the file and not just by the extension of the file.

All the options previously discussed in the prior section, such as **-tar**, **-gzip**, **-ads\_rename**, are applicable here as well.

When building scripts later on, there are a couple of important variables that **dup** can understand. The first is **%usersbase%** and it tells **dup** to look for the user base directory, such as *c:\users* which is used for the new Windows OS's and *c:\documents and settings* for the older WinXP versions. Using this variable in a command will allow the script to be ubiquitous across different versions of Windows. The second variable that **dup** can understand is **%eventlogs%**. This variable either points to *c:\windows\system32\winevt\logs* or *c:\windows\system32\config*, depending on whether the target OS is either a newer or older Windows version, respectively.

For example, one could repeat the above command with a variable and yield the same results.

```
dup64 -copydir %usersbase% -level 5 -filter "ntuser.dat!usrclass.dat" -out c:\dump\results
```

The advantage of doing this makes scripting the above command ubiquitous across all Windows versions. **dup** internally determine which Windows version it is analyzing and then determines where the user's directory is.

#### 2.4.2.1 Use of Wildcard for a folder

One can use the asterisk character '\*' to denote a wildcard for a folder. This allows the directories to be scanned to be more generic. A good example is trying to copy all the Google Chrome files from the various user's accounts. Without knowing the names of the individual accounts before hand one can use a wildcard in the path, like this.

```
dup64 -copydir 'c:\users\*\AppData\Local\Google\Chrome\User Data' -level 8 -out c:\dump\google_results -gzip
```

In this way, all the Google Chrome data will be archived in one zipped file for all the accounts for that computer.

### 2.4.3 copygroup

For the more common artifacts, **dup** has the **-copygroup** option which can take a number of sub-options to identify which artifacts to extract. There are groups for registry hives, event logs, prefetch files, LNK/JumpList, trash entries, and system files. One can use one or more of the sub-options in one session, or just invoke the **-pull\_all** option to tell **dup** to pull the common pre-canned artifacts. There are some other pre-canned artifact options that can be invoked alongside of the **-pull\_all** option, if desired. These include the *live statistics* (artifacts that are in memory, like network stats, processes/tasks



running, etc); this is the **-pull\_livestats** option. If one desires to include these sub-options as well, one just needs to add the **-pull\_<suboption>** next to the **-pull\_all** option to invoke those as well.

As mentioned previously, **dup** will discern which operating system the tool is running on so the proper default directories are targeted for the file groups selected. The other nice aspect about this option is it will spawn multiple instances of the **dup** tool to go after the specific groups. For computers with multiple cores, this will result in a faster copy. Results can either be *tar'd* into a packed archived file or *gzip* into a compressed archived file. Below are the sub-options currently available:

Group Option	Files Targeted
<b>-pull_sysfiles</b>	[Windows] \$MFT, \$Boot, \$LogFile, \$Bitmap, \$BadClus:\$Bad, UsnJrnl:\$J and Shim db files [uses <b>-ads_rename</b> internally] [Linux] <i>certain /etc and /proc files, first level folder files</i> [each user] [macOS] <i>fsevents folders, first level folder files</i> [each user], <i>bash and zsh sessions</i> [each user]
<b>-pull_reghives</b>	[Windows only] User and OS level (system, software, security, etc) registry hives
<b>-pull_evtlogs</b>	[Windows] Event, <i>setupapi</i> and other logs [Linux] <i>/var/log folder</i> [macOS] <i>/var/log folder</i>
<b>-pull_links</b>	[Windows only] LNK and JumpList files
<b>-pull_pfs</b>	[Windows only] <i>prefetch</i> files
<b>-pull_systrash</b>	[Windows] Recycle Bin directory on the system drive. [Linux] <i>.local/share/Trash folder</i> [each user] [macOS] <i>.Trash</i> [each user]
<b>-pull_userdbs</b>	[Windows] ActivitiesCache DB, Push Notifications DB, Outlook, Thunderbird and the main top user-level folder contents, such as: Desktop, Documents, Downloads, Pictures, and Videos [each user] [Linux] configuration and terminal history data [each user] [macOS] configuration and terminal history data [each user]
<b>-pull_browsers</b>	[Windows 7 and later] WebCache DB, Firefox, Edge, Chrome, Brave, Vivaldi and Opera [each user] [Linux]: Firefox, Edge, Chrome, Brave, Vivaldi and Opera [each user] [macOS] Safari, Firefox, Edge, Chrome, Brave, Vivaldi and Opera [each user]
<b>-pull_all</b>	[Windows/Linux/macOS] Invokes all the groups above
<b>-pull_livestats</b>	[Windows/Linux/macOS] Pulls host's system networks statistics and running tasks/processes. Therefore, it is not necessarily useful when extracting artifacts from an image or mounted volume that is not the host system volume.

#### 2.4.3.1 Using copygroup on Volume Shadow Copies

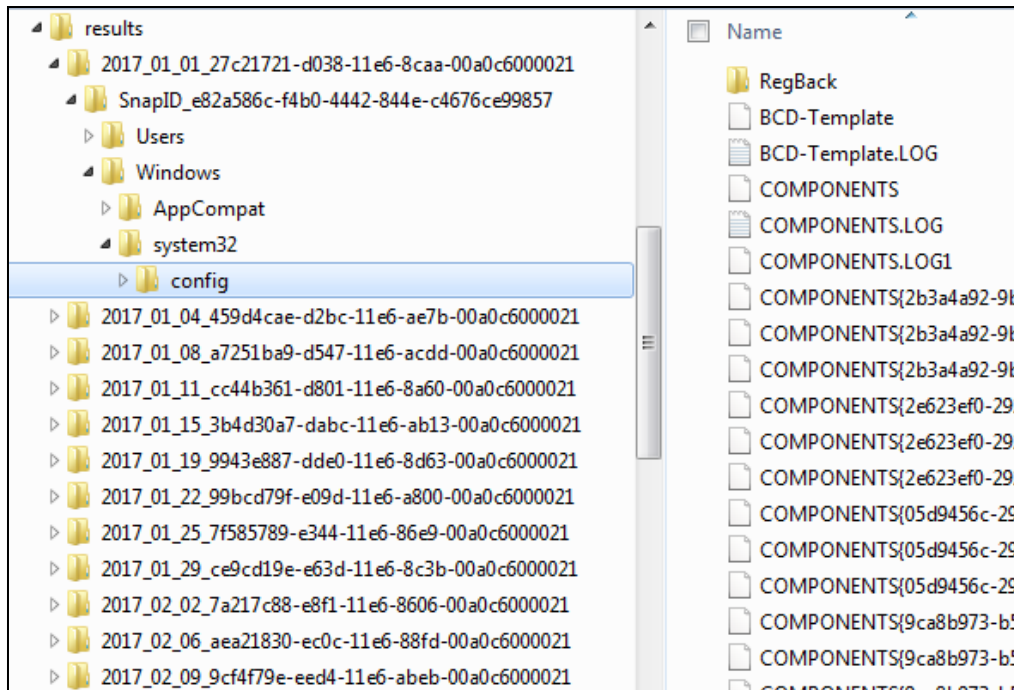
Targeting artifacts in the volume shadow copies is also possible with **dup**. There are two flavors for this. The first option allows one to target a specific volume shadow copy by identifying it by its index. The second option allows one to pull all targeted files from all the volume shadow copies on a specified volume. Both of these options, however, require the volume to be mounted on the system and the volume shadows to be known to the operating system.

To target a specific volume shadow, use the following syntax **-vss <index>**. To target all volume shadows on a volume, use the following: **-vssall <partition letter>**. As an example, if we wanted to target all the registry hives on all the volume shadow copies located on the system volume, one would do the following:



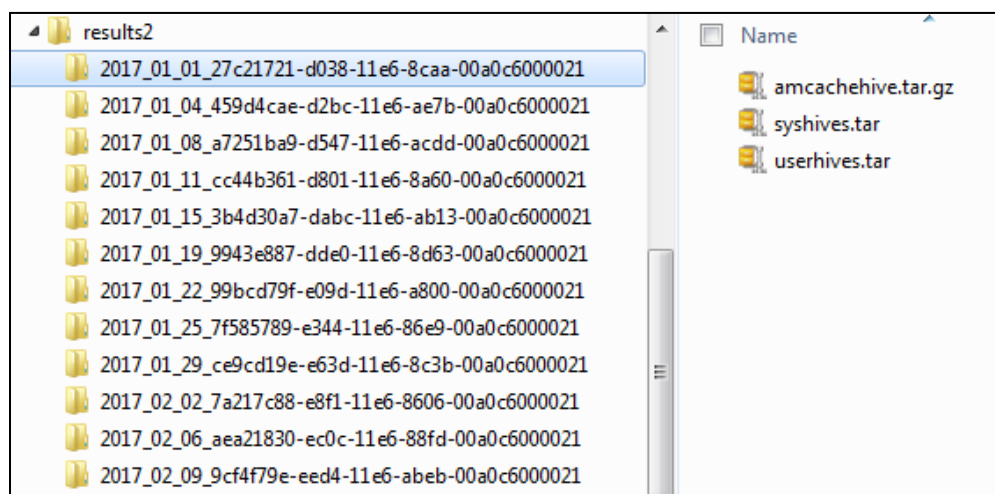
```
dup64 -copygroup -pull_reghives -vssall c: -out c:\dump\results
```

Performing the above command will yield a results folder with a subfolder per volume shadow copy. The primary subfolder names are annotated with the date of the volume shadow copy and it's GUID, followed by another subdirectory of the Snapshot ID.



If desiring a more compact set of results append the **-gzip** option, like so.

```
dup64 -copygroup -pull_reghives -vssall c: -out c:\dump\results2 -gzip
```



### 2.4.3.2 Forcing copygroup to Target a separately Mounted System Volume

The **-copygroup** option was designed to target the system volume of a live system. There are cases, where one may want to use **dup** and the **-copygroup** option to target other system volumes that were imaged from another machine. Currently, **dup** has two options to do this. The first is discussed in a separate section titled “Copying files from a ‘dd’ image”. The second is if one mounts an image of volume as another drive letter. For this case, there is an experimental option **-sysvol <mounted drive letter>** that can be used in conjunction with the **-copygroup** option.

### 2.4.4 Filtering on specific files

Normally, one can find a certain type of file based on its file extension. In these cases, one can use the **-filter <“\*.ext | \*partialname\* | ...”>** option. The wildcard character '\*' is restricted to either before the name or after the name. This is a very limited capability, but suffices on general filtering. For more complex file types where the extension is not uniform for common files used in forensic analysis, a custom filter is provided where the header contents of the file are examined. There are custom filters for some of the more common file types: **-filter\_sqlite**, **-filter\_plist**, **-filter\_esedb**.

### 2.4.5 copyscript

For clients that want to have a repeatable set of artifacts extracted from many endpoints across a network, incorporating well tested commands into a script is a good way to do this. To use a script with **dup**, use the option **-copyscript** followed by the file containing the script.

The script, per se, is just a text file containing a sequence of commands that you wish to perform. **dup** parses a script file using the following rules:

1. General Rules
  - a. The text file containing the script should not have any special characters used in formatting the font. Notepad is a good tool to produce these types of script files, since it does not add special formatting characters or other hidden syntax.
  - b. Each line is parsed separately, so each command needs to reside on its own line.
  - c. A line that starts with two forward slashes (eg. //) is ignored and used for comments
  - d. A blank line is ignored
  - e. Any line not satisfying rule (1c) and (1d) above is assumed to be a command
  - f. All command lines are in CSV format, where the separator is a comma. This applies to commands with parameters. So if a command has a keyword and argument(s), then the keyword is listed, then a comma, then an argument, then another comma, then the next argument. This simple rule allows all the keywords and arguments to be separated.
2. Command Lines [must start with the sequence: **!cmd**, and the entire command must be on one line].

- a. The command sequence can contain the following options, using comma delimiters (in any order):

- **-copyfile, <file to copy>**
- **-copydir, <folder to copy>**
- **-copygroup, <group1>, <group2>, ...,**
- **-out, <destination>**
- **-gzip**
- **-tar**
- **-filter, <\*.ext>**
- **-image, <'dd' of volume image file>**

Below is an example script file that copies various artifacts.

```
// [example1.txt]. Example of a script for the dup utility

// extract the $mft, $boot and change log journal
!cmd, -copyfile, c:\$mft|c:\$boot|c:\$Extend\$\UsnJrnl:$J, ads_rename, -out, c:\results\sysfiles, -gzip

// extract the system and user hives
!cmd, -copygroup, -pull_reghives, -out, c:\results\reghives, -gzip

// extract the event logs
!cmd, -copygroup, -pull_evtlogs, -out, c:\results\evtlogs, -gzip

// extract all the files with a *.dat extension from the c:\users folder
// and scan 7 subfolders deep
!cmd, -copydir, c:\users, -level, 7, -filter, *.dat, -out, c:\results\usersbase_dat_files, -gzip
```

The first **!cmd** line copies three files: *\$mft*, *\$boot* and the USN change log journal. The second and third **!cmd** lines copy the registry hive and event logs, respectively. We could have combined these two lines as follows: **!cmd, -copygroup, -pull\_reghives, -pull\_evtlogs, -out, c:\results\syslogs, -gzip**. The fourth **!cmd** line copies all the files with a **\*.dat** extension from the **c:\users** folder and traverses down 7 subdirectories during the scan/copy operation. To invoke the above script (called *example1.txt*), one issues the following command.

```
dup64 -copyscript example1.txt
```

If one modifies the above script without explicitly specifying a directory, but uses *variables* for artifacts and *relative paths* for outputs, one can make the script generic. Below is an example of taking the above script and doing just that.

```
// [example2.txt] Example of a script for the dup utility using variables and
// relative paths to avoid fixed drive letters

// extract the $mft, $boot and change log journal
!cmd, -copyfile, %systemdrive%\$mft|%systemdrive%\$boot|%systemdrive%\$Extend\UsnJrnl:$J, ads_rename, -out, .\results\sysfiles, -gzip

// extract the system and user hives
!cmd, -copygroup, -pull_reghives, -out, .\results\reghives, -gzip

// extract the event logs
!cmd, -copygroup, -pull_evtlogs, -out, .\results\evtlogs, -gzip

// extract all the files with a *.dat, *.lnk or *ions-ms extension from the .\users folder
// and scan 7 subfolders deep
!cmd, -copydir, %usersbase%, -level, 7, -filter, *.dat|*.lnk|*ions-ms, -out, .\results\userfiles, -gzip
```

Instead of specifying the **c:** drive, one can use the system environment variable **%systemdrive%**. Instead of explicitly listing the **c:\users** folder, **dup** recognizes the variable **%usersbase%** to point to that directory for new Windows operating systems and **c:\Documents and Settings** for the older Windows operating systems. For the output, one could point to a relative path; relative, in this case, is based on the path from where you start from the command prompt. With some simple changes, one now has a general purpose script for any windows box.

As a final example, one can take the above generic script and apply it to a file containing a volume image as opposed to a live volume. This assumes the image that is stored as a file is a 'dd' type image that is bit-for-bit copy of the volume. Let's assume the name of the generic script above is call script2.txt. To invoke the script and tell the script to target an image file, one can do the following:

```
dup64 -copyscript example2.txt -image e:\images\win8.dblake.vhd
```

## 2.5 Working with Linux and macOS – pulling artifacts

If desiring to use **dup** to pull artifacts from *Linux* or *macOS*, one can do this on either: (a) the live *Linux* or *macOS* box, or (b) from a mounted image, where the mounted volume can be either Windows, Linux or macOS.

If targeting the live system, one can use any of these artifact extraction options: **-copyfile**, **-copydir**, **-copyscript**, or **-copygroup**. The sub options **-gzip** and **-sqlite\_log** work as well, if desiring to compress the collected artifacts and/or create an SQLite log file, respectively.

If targeting an offline 'dd' image, one would first need to mount the volume on the host OS. If the host operating system is *Linux*, this can be done this by using the native host *mount* command. A typical example might be:

```
> sudo mount -o loop,ro <location of the image to be mounted> <location of the mount point>
```

If the format of the image cannot be determined using the above command, then use the option **-t <type of format>**, to explicitly identify the format type of the raw image during the mounting process.

Alternatively, if using macOS to mount a raw 'dd' image, then one can either use the native, *hdiutil attach* command. For this to work, the format of the 'dd' image needs to be recognized by macOS, otherwise, one needs to use a 3rd party tool to mount and later dismount the raw image. Here's an example of mounting a NTFS volume on macOS:

```
> hdiutil attach -noverify -noautofsck <path to raw image> -mountpoint <pathname> [-shadow | -readonly]
```

The option *-noverify* skips the verification process to save time. The option *-noautofsck* skips the system checks if an unclean image is detected, and the *-shadow* option creates a shadow file to allow a locked forensic image to mount as if it was read/write (making all changes redirect to the shadow image). This latter option allows Spotlight to index the image (without actually writing on the original image). If indexing is not wanted or required, just use the *-readonly* option. The other useful option is to specify where you want the image mounted via *-mountpoint*. Without this option, the system will mount the image in the root *Volumes* path.

Once the imaged is mounted, one can use *dup's -sysvol <location of the mounted image>* option to target artifacts from the specified image. When the artifacts have been collected, one can then *unmount* the volume, via:

```
> umount <location of the mount point>
```

The above is typical for unmounting devices that were mounted using the '*mount*' command. If using *hdiutil attach* from macOS, one can just use the *hdiutil detach <path of the mountpoint>*.

When mounting an NTFS raw 'dd' system image using the macOS utility *hdiutil*, for some reason the root system files of the mounted volume do not show up. I'm referring to the \$MFT, \$Boot, etc files. For the NTFS case, since *dup* can handle a monolithic native NTFS dd images, one can resort to the *-image <path of the raw NTFS image>* to pull out artifacts. The only downside with using this option is that if the file is compressed (using Windows Overlay Filtering – Wof), then the raw compressed data is returned versus the uncompressed data.

## 2.6 Copying Metadata and Logging

The later versions of *dup*, starting with version 0.37 has the ability to copy additional metadata when copying files and folders. This is part of the logging enhancement that is invoked with the *-sqlite\_log <name of log>* option. The log file that is created is formatted as an SQLite 3 file using the following schema. The 'ref' table is just for internal book keeping. The main two tables of interest are the (a) 'files' table and the (b) 'metadata' table. The 'files' table will record all the attributes associated with the file that was copied. The 'metadata' table is a summary of the run statistics when the dup tool was executed, such as the command that was used, the run time, and other data associated with the files table schema. For each file that was copied, there will be a separate entry in the 'files' table.

files	metaref	integer
	path	varchar
	name	varchar
	extension	varchar
	size	integer
	modified	varchar
	created	varchar
	metachanged	varchar
	accessed	varchar
	inode	integer
	copied	integer
	notes	varchar

ref	table_name	varchar
	last_rowid	integer
	last_update	integer

metadata	runtime	integer
	chunk	integer
	parent	integer
	lic_num	integer
	lic_name	varchar
	cmdline	varchar
	gen_hdrs	varchar
	col_hdrs	varchar
	col_types	varchar
	extra_args	varchar
	host	varchar
	user	varchar
	file	varchar
	file_other	varchar
	output_option	integer
	separator	varchar
	notes	varchar
	toolname	varchar
	artifact_type	varchar
	data_tbl_name	varchar
	row_data_start	integer
	row_data_end	integer

Below is a sample output of the 'file' table

path	name	xtensio	size	modified	created	metachanged	accessed	inode	copied	ntfsraw	notes
c:\user\local\Temp\	.ses	ses	53	2021-...	2021-07-26 20:12:33	2021-07-26 23:20:06	2021-07-26 23:20:06	872178	1	1	NULL
c:\user\local\Temp\	1627389999.srcsafe	srcsafe	584	2021-...	2021-07-27 12:46:39	2021-07-27 12:46:39	2021-07-27 12:46:39	22927	1	1	NULL
c:\users\tzworks\AppData\Local\Temp\	1627391119.srcsafe	srcsafe	584	2021-...	2021-07-27 13:05:19	2021-07-27 13:05:19	2021-07-27 13:05:19	26505	1	1	NULL
c:\users\tzworks\AppData\Local\Temp\	2alm4oyt.cyq	cyq	140...	2021-...	2021-07-27 12:18:10	2021-07-27 12:18:10	2021-07-27 12:46:41	419101	1	1	NULL
c:\users\tzworks\AppData\Local\Temp\	DESKTOP-M8BOQQP...	log	954...	2021-...	2021-07-27 02:36:43	2021-07-27 02:36:43	2021-07-27 02:36:45	873073	1	1	NULL

## 2.6.1 Extracting INDX data from the Folders

If using extra (currently experimental options), such as the **-incl\_indx** option, which pulls the INDX data associated with the folders. This INDX data is stored as a separate file and an 'files' entry is added associated with the INDX data. However, in this particular case, another table will also be added; the schema of this table is shown below.

wisp	metaref	integer
	path	varchar
	name	varchar
	extension	varchar
	is_dir	integer
	size	integer
	modified	varchar
	created	varchar
	metachanged	varchar
	accessed	varchar
	inode	integer
	seqnum	integer
	offset	integer
	notes	varchar

When using **-incl\_indx** option along with the **-sqlite\_log** option, dup will internally parse the INDX data that is extracted and parse out the INDX entries associated with a child file or folder. The associated



metadata will be outputted to the 'wisp' table. This option should, however, be considered experimental.

Below is a sample output from the 'wisp' table. The 'notes' field identifies whether the entry was pulled from the INDX slack, deleted, etc.

path	name	xtensio	size	modified	created	inode	seqnum	offset	notes
c:\us...a\Local\Temp\	PY6E91~1.LOG	LOG	237568	2021-06-06 00:14:06	2021-06-06 00:14:06	113317	2083	5728848	slack, deleted
c:\us...a\Local\Temp\Di...	AP7CE7~1.LOG	LOG	376832	2021-07-15 17:26:03	2021-07-15 16:21:00	4985	1940	86528	NULL
c:\users\tzworks\AppData\Local\Temp\Di...	App_162636606728...	log	376832	2021-07-15 17:26:03	2021-07-15 16:21:00	4985	1940	13864	NULL
c:\users\tzworks\AppData\Local\Temp\	PY6F06~1.LOG	LOG	241664	2021-06-06 00:36:35	2021-06-06 00:36:35	34399	1654	5728960	slack, deleted
c:\users\tzworks\AppData\Local\Temp\	Python 3.9.5 (64-bit...	log	241664	2021-06-06 00:36:35	2021-06-06 00:36:35	34399	1654	5970576	slack, deleted
c:\users\tzworks\AppData\Local\Temp\V...	494618~1.JFM	JFM	16384	2021-06-24 19:02:27	2021-06-24 19:02:27	35387	1640	400	slack, deleted
c:\users\tzworks\AppData\Local\Temp\	Python 3.9.5 (64-bit...	log	81920	2021-06-06 00:37:34	2021-06-06 00:37:34	35387	1621	6097392	slack, deleted

## 2.6.2 Extracting Spotlight Metadata

If running **dup** on macOS, one can invoke an option **-incl\_spotlight** to extract Spotlight metadata that is associated with every file that is copied. This option is still in work and under test. Thus, it should be considered experimental. Below is a sample use of this option targeting the Safari folder.

```
tzworks@TZWorks % ./dup -copydir ~/Library/Safari -level 9 -out ~/Desktop/safari_artifacts -sqlite_log ~/Desktop/dup_log.sqlite -incl_spotlight
```

Since the Spotlight has many possible metadata attributes, only those attributes that are actually associated with the file in question are shown. The output of the Spotlight attributes is reported in the 'files' table in the 'notes' field. Since there are many possible attribute name/value pairs, the output is represented as a quasi-JSON format. In this way, each attribute name is paired with its metadata value. To help the analysis understand some of the metadata values, **dup** will try to interpret each attribute value so that it is human readable (eg. convert timestamps into their date/time format). Below is an example output from the SQLite log.

path	name	extension	size	modified	created	metachanged	accessed	inode	copies	notes
/Users/tzworks/Library/Safari/	History.db-lock	db-lock	0	2020-01-01 00:33:22	2020-01-01 00:33:22	2020-01-01 00:33:22	2020-01-01 00:33:22	419165	1	extra_attri...
/Users/tzworks/Library/Safari/	TopSites.plist	plist	886	2021-01-12 19:20:56	2021-01-12 19:20:56	2021-01-12 19:20:56	2021-01-12 19:56:02	8083895	1	extra_attri...
/Users/tzworks/Library/Safari/	CloudAutoFillCorrectio...	db-wal	0	2021-07-26 19:20:40	2020-01-01 00:33:22	2021-07-26 19:20:40	2021-07-26 19:20:40	419146	1	extra_attri...
/Users/tzworks/Library/Safari/	.DS_Store	DS_Store	6148	2021-02-01 21:41:08	2020-08-13 13:52:28	2021-02-01 21:41:08	2021-02-02 21:33:59	1167338	1	extra_attri...
/Users/tzworks/Library/Safari/	PerSiteZoomPreferenc...	plist	111	2021-01-05 16:12:02	2021-01-05 16:12:02	2021-01-05 16:12:02	2021-01-12 18:42:03	8053096	1	extra_attri...

```
extra_attributes=["kMDItemFSIsExtensionHidden":"0";"kMDItemFSCreationDate":"01/01/20
00:33:22.686";"kMDItemFSFinderFlags":"0";"kMDItemFSHasCustomIcon":"0";"kMDItemFSSize"
:"0";"kMDItemFSIsStationery":"0";"kMDItemFSLabel":"0";"kMDItemFSContentChangeDate":"01
/01/20
00:33:22.686";"kMDItemFSFileTypeCode":"0";"kMDItemFSOwnerGroupID":"20";"kMDItemFSInvisib
le":"0";"kMDItemFSName":"History.db-
lock";"kMDItemFSCreatorCode":"0";"kMDItemFSOwnerUserID":"501";"kMDItemFSNodeCount":"
0"]
```

Spotlight data

## 2.6.3 Metadata table



The other table that may be of interest is the *'metadata'* table. It contains history of what command was used, the time a command was run, the host machine details and which records in the *'files'* table correlate to the command run. Below is an example of running dup twice using the same SQLite log file. One instance was run on a macOS box with an ARM processor and one with an Intel processor.

Since only 2 separate runs were recorded into the log file, there will be 2 records in the *metadata* table. The *'row\_data\_start'* and *'row\_data\_end'* fields in the *'metadata'* table relate to the *'row id'* of the *'files'* table. This provides some traceability which command is associated to which records in the *'files'* table.

metadata table			
runtime	cmdline	host	file
1 132718808558616770	./dup -copydir /Users/davidtomczak/Library/Safari -level 9 -out /U...	Mac-mini.local	/Users/davidtomczak/Desktop/Safari_artifacts [output]
2 132718856734248170	./dup -copydir /Users/tzworks/Library/Safari -level 9 -out /Users/t...	MacBook-Pro.local	/Users/tzworks/Desktop/safari_artifacts [output]

notes	row_data_start	row_data_end
Darwin Kernel Version 20.5.0: Sat May 8 05:10:31 PDT 2021; root:xnu-7195.121.3~9/RELEASE_ARM64_T8101	1	159
Darwin Kernel Version 20.6.0: Wed Jun 23 00:26:31 PDT 2021; root:xnu-7195.141.2~5/RELEASE_X86_64	160	643

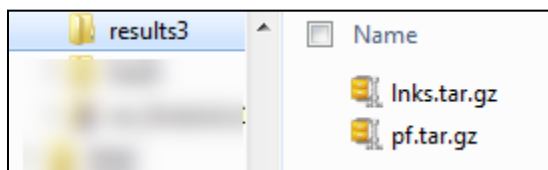
Only a few selected fields were shown in the above example.

## 2.7 Copying files from a 'dd' image

While **dup** was designed to go after mounted partitions during the copy, as mentioned in the previous section, it can also target 'dd' images of NTFS partitions by pointing the tool at the image via the **-image** option. This is useful when you are handed a collected image and you want to pull off certain raw artifacts quickly. The syntax for the copy command is like any previous copy option discussed previously, but appending the **-image <dd image file>** to the command. Let's say you wanted to pull all the LNK and prefetch files from the image, you could issue the following command.

```
dup64 -copygroup -pull_links -pull_pfs -out c:\dump\results3 -image e:\images\win8.dblake.vhd -gzip
```

Notice we have both **-pull\_links** and **-pull\_pfs** in the same command line to instruct **dup** to extract both of these types of files from the specified image file.



The only downside with using this option is that if the file is compressed (using Windows Overlay Filtering – Wof), then the raw compressed data is returned versus the uncompressed data.

## 2.8 File Utilities

There are a number of file utility functions available in **dup**. A list of these can be seen from the command line menu shown below. Each of these will be discussed.

```
File Utilities
-untar <file> -out <dst>      = unpack file that was packed with -tar
-expand <file> -out <dst>     = decompress file that that used -gzip
-merge -fromfile <f1> -tofile <f2> -tofile_offset <#> [-skip_null_sectors]
-combine_files <file1!file2!...> -out <dst file> = appends to dst file
-md5 [<file> ; -pipe]        = compute md5 hash of file or files
-sha1 [<file> ; -pipe]       = compute sha1 hash of file or files
```

### 2.8.1 untar

As background, the term **TAR** stands for (Tape **AR**chive) and was added as a common *Unix* utility, back in the day, when storage of data was done on a tape or any device that stored data in a sequential I/O manner. The resulting file that was created was coined **tarball** and the device that stored the file consisted of no file system of its own. The **TAR** format allowed one to collect many files as a single archive file for backup purposes. As an aside, compression utilities today usually do this sort of operation, in the background, to collect multiple files into a single file prior to compression.

Without getting into the details of the **TAR** format and the variations thereof, the **-untar** option in **dup** undoes the **-tar** operation. Specifically, if one used the **-tar** option when copying multiple files, the **-untar** operation will take the **tarball** and expand the files into the directory structure as the original files.

Normally, one would just use the **tar** utility in Linux or macOS to untar a tarball, however, with Windows this utility is not part of the operating system. Therefore, a limited version is provided with **dup** to undo any **tar** operation that may have been used by **dup**. Since there are various formats for **TAR**, one should note, this option is only for **tarball's** created by the **dup -tar** option.

### 2.8.2 expand (or uncompress)

The **-expand** option will uncompress any file compressed by **dup** that was used with the **-gzip** or **-compress** operation.

### 2.8.3 compress

The **-compress** option is only has basic functionality to compress a file into a **gzip** format. It has two basic options: (a) either on a single file via the **-compress <file>** syntax or the **-compress -pipe** option to tell **dup** to expect filenames to be passed via standard input.

### 2.8.4 merge

If desiring to merge two files into one while starting one file at a specified offset, this option is for you. The arguments for the **-merge** option are the source file you want to merge (**-fromfile**), an existing file

(**-tofile**), and the offset where you want the source file to go within the existing file (**-tofile\_offset**). This offset can be anything, from within the existing file's data, to outside the existing file's size.

For example, if one picked the offset to be within the body of the existing file, the source file will overwrite the existing file starting at the offset specified and only overwrite the number of bytes sized by the source file. If one picked the offset to be greater than the existing file's size, then the existing file will grow, by adding zeros, up to the desired offset, and the rest of the data will be populated with the source file's data.

This functionality, allows one to successfully merge the partial fragments of a file (or disk image) to their proper offset and create a semi-complete replica of a partial file (or bad drive).

### 2.8.5 combine\_files

If desiring to just concatenate two files together, one uses the **-combine\_files** option. One could use the **-merge** operation for concatenating as well, but it is more complex in that it requires one to figure out the size of file one wants to concatenate to and passing that as an argument. Therefore, for concatenation, just use the **-combine\_files** option. The argument is a list of files delimited by pipe characters. The order of the delimited list of files is the order of the concatenation.

### 2.8.6 Hashing Options

We added two common hashing functions (**-md5** and **-sha1**) to this category of options. There are two possible arguments one can use for these operations: (a) a single file in which the case the argument is the filename, or (b) multiple files using the **-pipe** argument, which allows one to send multiple files into standard input (STDIN).

### 2.8.7 Logging Options

The normal logging option is just redirecting the *stdout* into a file. This will generate a list of files that were copied. For a more detailed log, one can use the **-sqlite\_log <log file>** option. This will generate a log that is a SQLite database of all the files copied and some metadata associated with the file; to include, the path, size of the file, inode (if applicable) of the file, and any timestamps associated with the file.

If targeting NTFS folders as well as files, one can ask **dup** to also pull the INDX data associative with each folder that was traversed. The resulting INDX data will be archived in the SQLite log file. The command to do this is **-incl\_indx** along with the **-sqlite\_log <log file>** command.

### 3 *dup* Library Dependencies

From an external library dependency standpoint, starting with v.0.37 version of ***dup***, all the library dependencies have been statically compiled into the binary. This includes the *zlib* and *SQLite* libraries.

### 4 Available Options

Option	Description
<b><i>-scandrives</i></b>	Scan the drives attached to the computer and report the stats on each drive found.
<b><i>-scan_mountpts</i></b>	Scan each volume and the associated mount point for the session.
<b><i>-diskstats</i></b>	For a specified drive, return the statistics of the drive. The syntax is <b><i>-diskstats &lt;drive number&gt;</i></b> .
<b><i>-imagestats</i></b>	For a specified 'dd' image, return the statistics of that drive. The syntax is <b><i>-imagestats &lt;dd image path/file&gt;</i></b> .
<b><i>-vmdk</i></b>	For a specified VMWare VMDK monolithic image, return the statistics of that drive. The syntax is <b><i>-vmdk &lt;file1   file2   ...&gt;</i></b> .
<b><i>-mbr</i></b>	Pull the Master Boot Record (MBR) of the drive. The syntax is <b><i>-mbr &lt;drive number&gt;</i></b> .
<b><i>-mbr_compare</i></b>	Given 2 Master Boot Records compare them with each other and report differences. There are a number of variations for this, where one can compare two disk's, a disk MBR with a file of an MBR, or a disk MBR with an MBR in a 'dd' image. The available syntaxes are:  <b><i>-mbr_compare -disk1 &lt;disk#1&gt; -disk2 &lt;disk#2&gt;</i></b> <b><i>-mbr_compare -disk1 &lt;disk#&gt; -mbrfile &lt;MBR expected data&gt;</i></b> <b><i>-mbr_compare -disk1 &lt;disk#&gt; -image &lt;dd image&gt;</i></b>
<b><i>-copydrive</i></b>	Used to image a drive or a portion thereof. The syntax is <b><i>-copydrive &lt;drive number&gt; -out &lt;dst&gt; [-offset &lt;#&gt;] [-size &lt;#&gt;]</i></b> . The <b><i>-out &lt;dst&gt;</i></b> specifies where to store the image and its name. The <b><i>-offset &lt;#&gt;</i></b> allows one to start the image at a particular offset, and the <b><i>-size &lt;#&gt;</i></b> allows one to limit the number of bytes to copy. The offset and size values should be rounded to the sector size. For those unusual cases where a size is needed that is not on a sector boundary, one can use the

	switch <b>-force_non_sector_size</b> . This will allow to copy some value less than a sector size. If the drive has bad sectors, one can use the options: <b>-retries &lt;#&gt;</b> and <b>-read_timeout &lt;# secs&gt;</b> . This will tell <b>dup</b> to retry reading the bad sector after a specified timeout period.
<b>-copyvolume</b>	Used to image a volume or a portion thereof. The syntax is <b>-copyvolume &lt;partition letter&gt; -out &lt;dst&gt; [-offset &lt;#&gt;] [-size &lt;#&gt;] [-force_volread]</b> . The <b>-out &lt;dst&gt;</b> specifies where to store the image and its name. The <b>-offset &lt;#&gt;</b> allows one to start the image at a particular offset, and the <b>-size &lt;#&gt;</b> allows on to limit the number of bytes to copy. The offset and size values should be rounded to the sector size. The last option <b>-force_volread</b> tells <b>dup</b> to copy using volume semantics versus the default of <i>disk</i> semantics. Where this option is useful, is if the volume is <i>BitLocker</i> encrypted (or other encryption is present), using <i>volume</i> semantics will allow the copy operation to retrieve unencrypted data (as seen from the opened volume perspective); whereas using the <i>disk</i> semantics will cause <b>dup</b> to blindly copy the actual raw data stored on disk (at the volume offset/size), which if <i>BitLocker</i> encrypted, would be an encrypted copy of the data.
<b>-copysnaps</b>	Used to pull the raw data associated with the Volume Shadow snapshots. This option can pull the snapshots from either a mounted volume or a 'dd' image.
<b>-copyfile</b>	Used to copy a specified file. The syntax is <b>-copyfile &lt;filename&gt;</b> . One also needs to define where to copy the file, via the parameter <b>-out &lt;dst&gt;</b> . Other parameters can be added, such as <b>-tar</b> , <b>-gzip</b> , <b>-image</b> , or <b>-filter</b> . See the respective option for an explanation of their purpose. To use this option and receive filenames from standard input to copy, use the <b>-pipe</b> syntax. Below is an example:  <b>dir c:\users\*.lnk /b /s /s   dup -copyfile -pipe -out results</b>
<b>-copydir</b>	Used to copy a specified folder. The syntax is <b>-copydir &lt;folder&gt;</b> . One also needs to define where to copy the file; one can use <b>-out &lt;dst&gt;</b> . One can use the <b>-level &lt;# of levels&gt;</b> to define how many subfolders to copy as well. Other parameters can be added, such as <b>-tar</b> , <b>-gzip</b> , <b>-image</b> , or <b>-filter</b> . See the respective option for an explanation of their purpose.
<b>-copyscript</b>	Used to copy files defined in a script. The syntax is <b>-copyscript &lt;file containing the script&gt;</b> . The script should contain all the relevant arguments including where to copy the files, using <b>-out &lt;dst&gt;</b> . Other parameters can be added, such as <b>-tar</b> , <b>-gzip</b> , <b>-image</b> , or <b>-filter</b> . See the respective option for an explanation of their purpose.
<b>-copygroup</b>	Used to copy a predefined collection of files. One also needs to define

	<p>where to copy the files, using <b>-out &lt;dst&gt;</b>. Other parameters can be added, such as <b>-tar</b>, <b>-gzip</b>, <b>-image</b>, or <b>-filter</b>. See the respective option for an explanation of their purpose.</p> <p>There are separate sub-options that are used for each predefined collection. The available sub-options include:</p> <p><b>-pull_sysfiles</b> collects these files: <i>\$MFT, \$Boot, \$LogFile, \$Bitmap, \$BadClus:\$Bad, UsnJrnl:\$J &amp; Shim db</i> files.</p> <p><b>-pull_reghives</b> collects both user and system level registry hives</p> <p><b>-pull_evtlogs</b> collects event, <i>setupapi</i> and <i>diagnosis</i> logs.</p> <p><b>-pull_inks</b> collects <i>LNK</i> and <i>JumpList</i> files</p> <p><b>-pull_pfs</b> collects <i>prefetch</i> files</p> <p><b>-pull_systrash</b> collects the Recycle Bin directory on the system drive.</p> <p><b>-pull_userdbs</b> collects various DBs including: <i>ActivitiesCache</i> DBs, <i>Push Notification</i> DBs, <i>Outlook</i>, <i>Thunderbird</i>, and the main top user-level folder contents, such as: <i>Desktop</i>, <i>Documents</i>, <i>Downloads</i>, <i>Pictures</i>, and <i>Videos</i>.</p> <p><b>-pull_browsers</b> for Win7 and later: browser artifacts: <i>WebCache</i> DBs, <i>Firefox</i>, <i>Edge</i>, <i>Chrome</i>, <i>Brave</i>, <i>Vivaldi</i> and <i>Opera</i>.</p> <p><b>-pull_all</b> invokes all the above options.</p> <p><b>-sysvol &lt;volume letter&gt;</b> tells <b>dup</b> to target a specified mounted system volume, versus the default live system volume.</p> <p><b>-vss &lt;index&gt;</b> tells <b>dup</b> to target this volume shadow copy</p> <p><b>-vssall &lt;volume letter&gt;</b> tells <b>dup</b> to target all the volume shadow copies it finds on the specified volume.</p> <p><b>-pull_livestats</b> collects network related statistics and running tasks/processes. (This option is not included in the <b>-pull_all</b> category).</p>
<b>-untar</b>	To unpack a file that was packed with the <b>-tar</b> option, use <b>-untar</b> . The syntax is: <b>-untar &lt;file&gt; -out &lt;dst&gt;</b>
<b>-expand</b>	Decompresses a file that was compressed with the <b>-gzip</b> option. The syntax is: <b>-expand &lt;file&gt; -out &lt;dst&gt;</b>
<b>-compress</b>	To compress a single file or a directory of files. Syntax for a single file is: <b>-compress -f &lt;file&gt; -out &lt;destination folder&gt;</b> . For a multiple files, one can use standard input to specify which files to compress. <b>dir &lt;folder of files&gt; /b /s   dup -compress -pipe -out &lt;destination folder&gt;</b>
<b>-merge</b>	This option is to merge one file into another file. It also allows the user to specifically dictate where the file (offset wise) is to be merged. The purpose of this option was to handle situations where partial images were collected and need to be merged into a larger image, where the offsets of the images needed to be preserved. The syntax is <b>-merge -fromfile &lt;file1&gt; -tofile &lt;file2&gt; -tofile_offset &lt;#&gt;</b> . There is an optional switch, <b>-skip_null_sectors</b> , where this tells the operation to only look at sectors that have data. This becomes important during a merge when

	you do not want to overwrite existing data with null data.
<b>-combine_files</b>	To concatenate files in a sequence, use this option. The files to be concatenated or pipe delimited. The syntax is as follows: <b>-combine_files "file1 file2 file3 ..." -out &lt;dst&gt;</b>
<b>-md5</b>	To perform a MD5 hash on a single file or a directory of files. Syntax for a single file is: <b>-md5 -f &lt;file&gt;</b> . For a multiple files, one can use standard input to specify which files to compress. <b>dir &lt;folder of files&gt; /b /s   dup -md5 -pipe</b>
<b>-sha1</b>	To perform a SHA1 hash on a single file or a directory of files. Syntax for a single file is: <b>-sha1 -f &lt;file&gt;</b> . For a multiple files, one can use standard input to specify which files to compress. <b>dir &lt;folder of files&gt; /b /s   dup -sha1 -pipe</b>
<b>-wipefile</b>	Experimental. Zero out the contents of the specified file and delete it. Tries (on a best effort basis) to remove as much metadata associated with the file as well. [designed for Windows]
<b>-wipedir</b>	Experimental. Zero out the contents of all files in the specified folder and child subfolders and deletes them. Tries (on a best effort basis) to remove as much metadata associated with the files as well. [designed for Windows]
<b>-image</b>	Sub-option to specify to target an image of a NTFS disk or volume during a copy operation. If targeting a disk image, one would need to also supply the offset of the system volume using the <b>-offset &lt;#&gt;</b> option. Note, when using this option that if the file is compressed (using Windows Overlay Filtering – Wof), then the raw compressed data is returned versus the uncompressed data.
<b>-filter</b>	Sub-option to filter filenames passed in via STDIN via one of the copy options. The syntax is <b>-filter &lt;"*.ext   *partialname*   ..."&gt;</b> . The wildcard character '*' is restricted to either before the name or after the name.
<b>-filter_sig</b>	Sub-option to filter file signatures passed in via STDIN via one of the copy options. The syntax is <b>-filter_sig &lt;"hex bytes separated by spaces   offset"&gt;</b> . This option is experimental.
<b>-filter_sqlite</b>	Sub-option to filter SQLite files. This option looks at the file header to determine if the file is an SQLite file. The syntax is <b>-filter_sqlite</b>
<b>-filter_plist</b>	Sub-option to filter <i>plist</i> type files. This option looks at the file header to



	determine if the file is a <i>plist</i> file. The syntax is <b>-filter_plist</b> .
<b>-filter_esedb</b>	Sub-option to filter <i>EseDbs</i> type files. This option looks at the file header to determine if the file is an ESE database file. The syntax is <b>-filter_esedb</b> .
<b>-filter_start_date</b>	Sub-option for copying files from a directory to only copy those files with a create or modify date at or greater than the date specified here. The syntax is <b>-filter_start_date &lt;yyyy-mm-dd&gt;</b> .
<b>-filter_stop_date</b>	Sub-option for copying files from a directory to only copy those files with a create or modify date at or less than the date specified here. The syntax is <b>-filter_stop_date &lt;yyyy-mm-dd&gt;</b> .
<b>-sqlite_log</b>	Option to generate an SQLite log database of the files copied by the dup tool. The syntax is <b>-sqlite_log &lt;log file&gt;</b> . If copying files from subfolders, there is a sub-option for recording the folder INDX data ( <b>-incl_indx</b> ).
<b>-tar</b>	Sub-option to be used with a copy command collect all the files copied into one tar file.
<b>-gzip</b>	Sub-option to be used with a copy command to tell the operation to compress the results.
<b>-ntfsraw</b>	Option to force raw cluster reads for NTFS volume
<b>-incl_indx</b>	Experimental option to extract INDX metadata associated with folders.
<b>-incl_spotlight</b>	Experimental option to extract Spotlight attribute metadata associated with a file. Output is rendered in the SQLite log file (therefore, <b>-sqlite_log</b> is required for this option to be used)

## 5 Authentication and the License File

This tool has authentication built into the binary. The primary authentication mechanism is the digital X509 code signing certificate embedded into the binary (Windows and macOS).

The other mechanism is the runtime authentication, which applies to all the versions of the tools (Windows, Linux and macOS). The runtime authentication ensures that the tool has a valid license. The

license needs to be in the same directory of the tool for it to authenticate. Furthermore, any modification to the license, either to its name or contents, will invalidate the license.

## 6 References

1. **zlib** library version 1.2.11, 15 Jan 2017, by Jean-loup Gailly and Mark Adler.
2. <http://tools.ietf.org/html/rfc1950> (**zlib** format), rfc1951 (deflate format) and rfc1952 (gzip format)
3. **ustar** (Uniform Standard Tape ARchive) format - [https://en.wikipedia.org/wiki/Tar\\_\(computing\)](https://en.wikipedia.org/wiki/Tar_(computing))
4. **ntfscopy** – TZWorks NTFS copy utility - [https://tzworks.com/prototype\\_page.php?proto\\_id=9](https://tzworks.com/prototype_page.php?proto_id=9)
5. SQLite library statically linked into tool [Amalgamation of many separate C source files from SQLite version 3.32.3].
6. SQLite documentation [<http://www.sqlite.org>].
7. DB Browser for SQLite [<http://sqlitebrowser.org/>]