TZWorks LLC

## Windows $MFT and NTFS Metadata Extractor Tool (ntfswalk)

***ntfswalk*** is a command line a tool that traverses a specified NTFS volume reading all MFT entries and pulling predefined statistics as it runs.

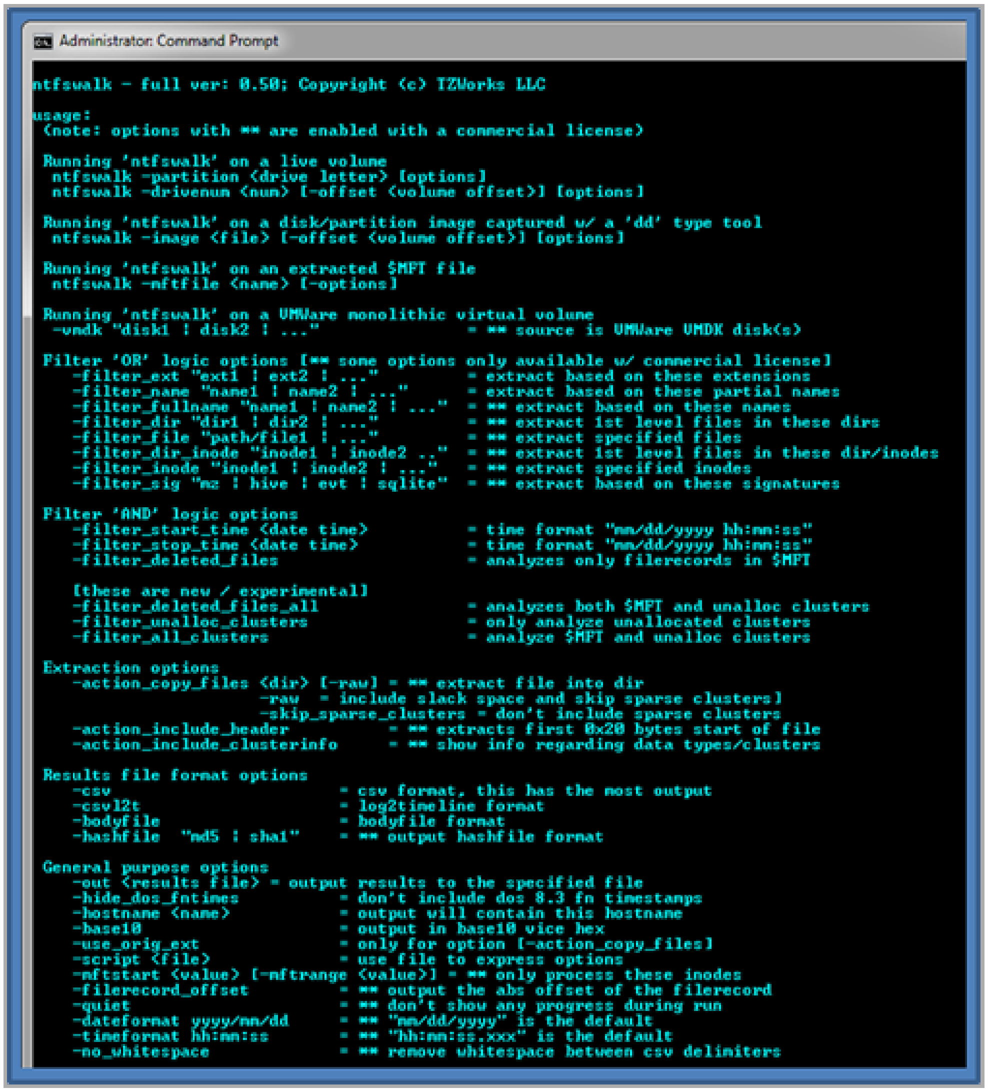
Originally the NTFS engine was designed as a widget for other applications to help pull data out from targeted categories of files on NTFS partitions. After successfully using the functionality in other tools, it was determined that the utility in making a standalone tool would be helpful in debugging and understanding the internals of any NTFS volume. This new tool, coined ***ntfswalk***, is named after its ability to walk an entire NTFS volume and output each MFT entry it encounters.

Designed to work with live NTFS partitions, there is also functionality for traversing NTFS images created with the 'dd' utility (as well as some versions of VMWare VMDK files). There are options to filter on file extension, timestamp range, binary signature, partial filenames and directory contents. For the files found, one can list the summary metadata, extract the header bytes, or extract the entire file contents into a designated directory. Since the engine is Windows API agnostic, there are compiled versions for Windows, Linux and Mac OS X.

If targeting a volume mounted on a live Windows system, one needs to be run ***ntfswalk*** with administrator privileges.

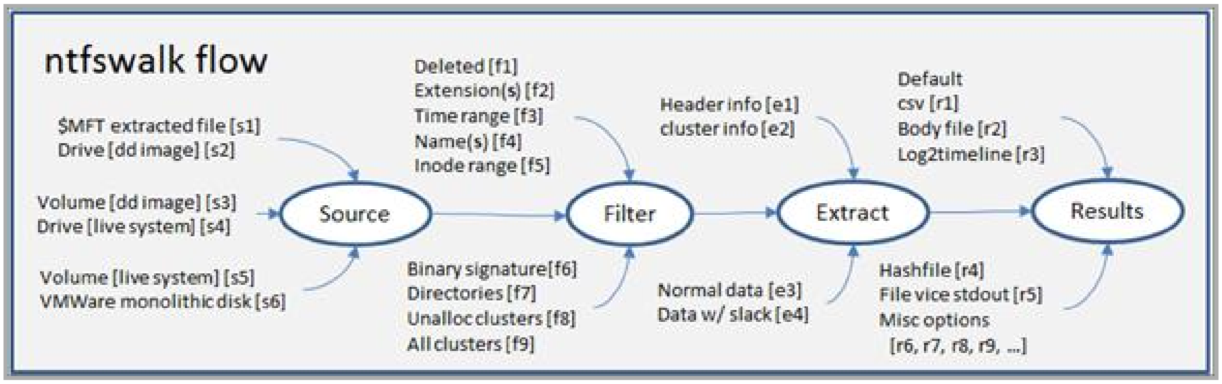
### How to Use *ntfswalk*

***ntfswalk*** has a number of command line switches, and for the occasional user, it can be confusing which options can be used together and which cannot. Below is a screenshot of the menu options that are displayed when running the tool without any arguments.



```
ntfswalk - full ver: 0.50; Copyright (c) TZWorks LLC

usage:
 (note: options with ** are enabled with a commercial license)

 Running 'ntfswalk' on a live volume
  ntfswalk -partition <drive letter> [options]
  ntfswalk -drivenum <num> [-offset <volume offset>] [options]

 Running 'ntfswalk' on a disk/partition image captured w/ a 'dd' type tool
  ntfswalk -image <file> [-offset <volume offset>] [options]

 Running 'ntfswalk' on an extracted $MFT file
  ntfswalk -mftfile <name> [-options]

 Running 'ntfswalk' on a VMWare monolithic virtual volume
  -vmdk "disk1 | disk2 | ..."                = ** source is VMWare VMDK disk(s)

 Filter 'OR' logic options [** some options only available w/ commercial license]
    -filter_ext "ext1 | ext2 | ..."         = extract based on these extensions
    -filter_name "name1 | name2 | ..."      = extract based on these partial names
    -filter_fullname "name1 | name2 | ..."  = ** extract based on these names
    -filter_dir "dir1 | dir2 | ..."         = ** extract 1st level files in these dirs
    -filter_file "path/file1 | ..."         = ** extract specified files
    -filter_dir_inode "inode1 | inode2 .."  = ** extract 1st level files in these dir/inodes
    -filter_inode "inode1 | inode2 | ..."   = ** extract specified inodes
    -filter_sig "mz | hive | evt | sqlite"  = ** extract based on these signatures

 Filter 'AND' logic options
    -filter_start_time <date time>          = time format "mm/dd/yyyy hh:mm:ss"
    -filter_stop_time <date time>           = time format "mm/dd/yyyy hh:mm:ss"
    -filter_deleted_files                   = analyzes only filerecords in $MFT

    [these are new / experimental]
    -filter_deleted_files_all               = analyzes both $MFT and unalloc clusters
    -filter_unalloc_clusters                = only analyze unallocated clusters
    -filter_all_clusters                    = analyze $MFT and unalloc clusters

 Extraction options
    -action_copy_files <dir> [-raw] = ** extract file into dir
                       -raw  = include slack space and skip sparse clusters]
                       -skip_sparse_clusters = don't include sparse clusters
    -action_include_header          = ** extracts first 0x20 bytes start of file
    -action_include_clusterinfo     = ** show info regarding data types/clusters

 Results file format options
    -csv                       = csv format, this has the most output
    -csvl2t                    = log2timeline format
    -bodyfile                  = bodyfile format
    -hashfile  "md5 | sha1"    = ** output hashfile format

 General purpose options
    -out <results file> = output results to the specified file
    -hide_dos_fntimes          = don't include dos 8.3 fn timestamps
    -hostname <name>           = output will contain this hostname
    -base10                    = output in base10 vice hex
    -use_orig_ext              = only for option [-action_copy_files]
    -script <file>             = use file to express options
    -mftstart <value> [-mftrange <value>] = ** only process these inodes
    -filerecord_offset         = ** output the abs offset of the filerecord
    -quiet                     = ** don't show any progress during run
    -dateformat yyyy/mm/dd     = ** "mm/dd/yyyy" is the default
    -timeformat hh:mm:ss       = ** "hh:mm:ss.xxx" is the default
    -no_whitespace             = ** remove whitespace between csv delimiters
```

To help understand the various options, one can break the architecture into four main areas: (a) source of the data, (b) filter that can be applied, (c) extraction options, and (d) output format.



Starting with the first area, this identifies which data sources ***ntfswalk*** can handle for input. Various input types include: (i) an extracted $MFT file, (ii) a 'dd' image of a drive or volume, (iii) a drive or volume currently in use on a live system, or (iv) a VMWare single volume disk.

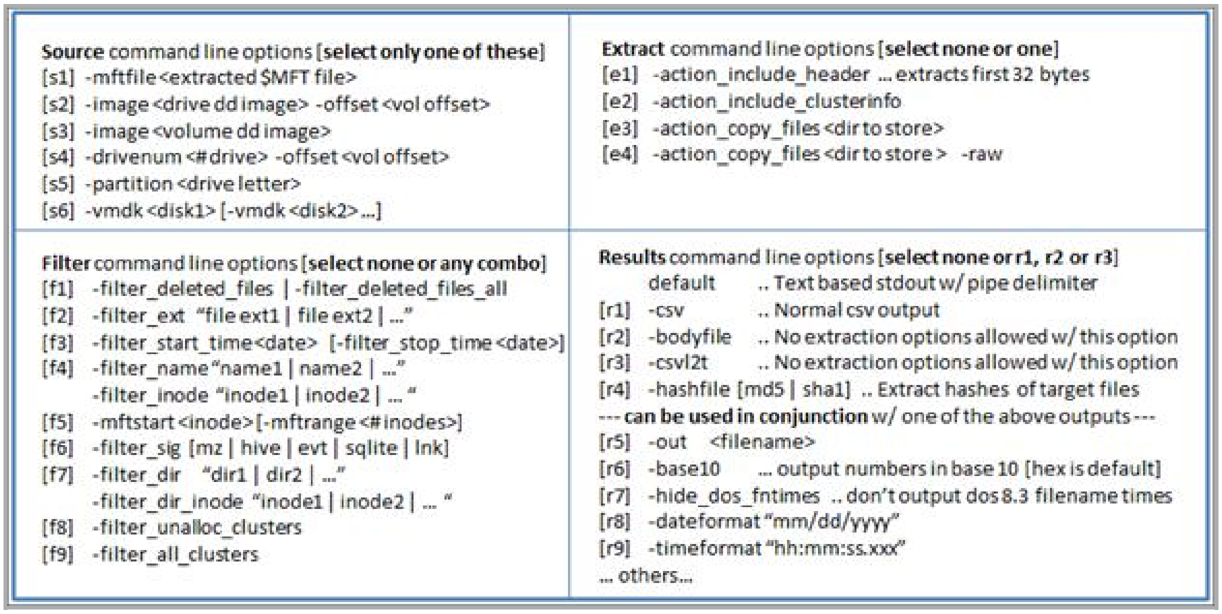
The second area shown above is filtering. This defines what files (or MFT entries) are analyzed and displayed to the user. One can filter on deleted files/folders, extensions, partial names, and binary signatures. For binary signatures, currently ***ntfswalk*** allows one to find: registry hives, event logs, SQLite3 databases, or portable executable files. Also in this area one can choose to analyze all *unallocated clusters* instead of the normal *allocated clusters*, or to pull files from a specified directory.

The third area in the diagram are the extraction options. Whatever option is chosen, at a minimum, ***ntfswalk*** will produce a results file. This results file will contain much of the metadata one needs for forensic analysis. For more detailed analysis, one can add extra data to the results, including: (a) the bytes in the header for each file or (b) the cluster run information. To physically extract the contents of the file, one can specify an archive directory as well as whether to include slack data or not. If one does extract the file data, ***ntfswalk*** will compute the MD5 hash of the file and annotate this data to the results file as well.

The fourth area allows one to select how one wishes to see the results. As mentioned above, even if one only wishes to extract data to a directory, there will be a results file that logs all the files passing the filter tests. The default output is plain text, which by itself, has reasonable formatting when viewed in notepad and word wrap is turned off. The other formats are geared for spreadsheet analysis or other post processing tools. Typically, any data containing numbers is defaulted as hexadecimal; however, there is an option to transform the output into base10 notation, if desired. As an add-on to ***ntfswalk***, is the ability to generate a hashset type results file.

### The Command Line options for the above

The syntax for each of the options that correlate to the above ***ntfswalk*** flow diagram is shown in the figure below. The figure also identifies which options can be used in combination with others. Therefore, one can select: (a) one source of input, (b) none or any combination of filters, (c) none or one extraction option and (d) one type of format for the output results.



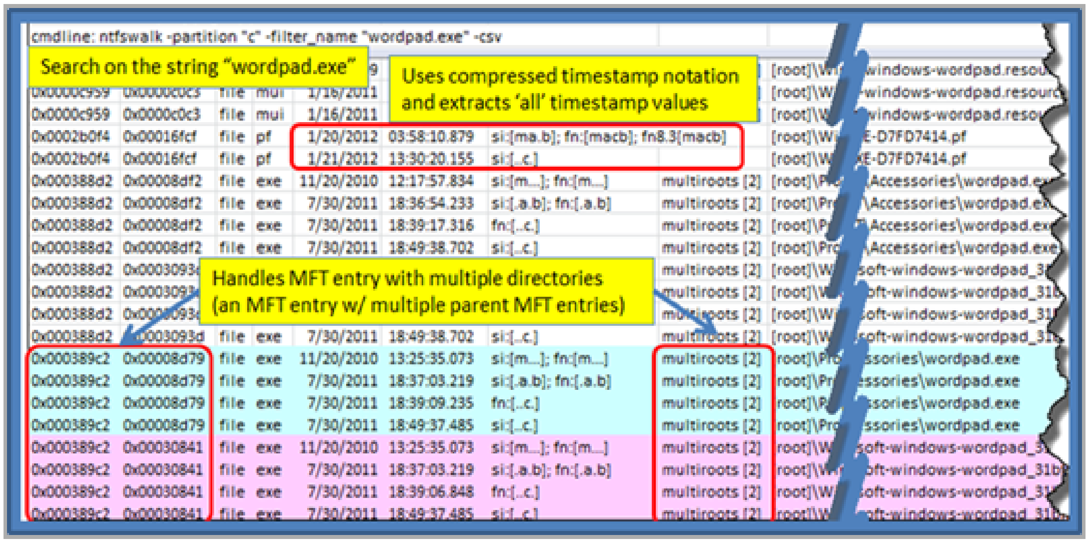
```
Source command line options [select only one of these]
[s1]  -mftfile <extracted $MFT file>
[s2]  -image <drive dd image>  -offset <vol offset>
[s3]  -image <volume dd image>
[s4]  -drivenum <#drive>  -offset <vol offset>
[s5]  -partition <drive letter>
[s6]  -vmdk <disk1> [-vmdk <disk2> ...]

Filter command line options [select none or any combo]
[f1]   -filter_deleted_files  | -filter_deleted_files_all
[f2]   -filter_ext  "file ext1 | file ext2 | ..."
[f3]   -filter_start_time <date>  [-filter_stop_time <date>]
[f4]   -filter_name "name1 | name2 | ..."
       -filter_inode  "inode1 | inode2 | ... "
[f5]   -mftstart <inode> [-mftrange <#inodes>]
[f6]   -filter_sig  [mz | hive | evt | sqlite | lnk]
[f7]   -filter_dir    "dir1 | dir2 | ..."
       -filter_dir_inode  "inode1 | inode2 | ... "
[f8]   -filter_unalloc_clusters
[f9]   -filter_all_clusters

Extract command line options [select none or one]
[e1]  -action_include_header  ... extracts first 32 bytes
[e2]  -action_include_clusterinfo
[e3]  -action_copy_files <dir to store>
[e4]  -action_copy_files <dir to store >   -raw

Results command line options [select none or r1, r2 or r3]
       default        .. Text based stdout w/ pipe delimiter
[r1]   -csv           .. Normal csv output
[r2]   -bodyfile      .. No extraction options allowed w/ this option
[r3]   -csvl2t        .. No extraction options allowed w/ this option
[r4]   -hashfile [md5 | sha1]  .. Extract hashes  of target files
--- can be used in conjunction w/ one of the above outputs ---
[r5]   -out    <filename>
[r6]   -base10        ... output numbers in base 10 [hex is default]
[r7]   -hide_dos_fntimes  .. don't output dos 8.3 filename times
[r8]   -dateformat "mm/dd/yyyy"
[r9]   -timeformat "hh:mm:ss.xxx"
... others...
```

### Understanding the Output

Lets say you wanted to search all the names in a live volume that contained the string "wordpad.exe" and store the output into CSV format. That way you could double click on the resulting CSV file and Excel could easily open the file. The syntax would be the following for scanning the 'c' partition and redirecting the output to some results file:

```
    ntfswalk -partition c -filter_name "wordpad.exe" -csv > results.csv
```

When examining the results.csv file, one would see *prefetch*, *mui* and *exe* entries all containing the string *wordpad.exe*. Since the *prefetch* entry has a name longer then the DOS 8.3 length, the normal windows system would have a set of timestamps for the long filename as well as a set of timestamps for the 8.3 version of the filename. Many of these timestamps are duplications, and thus, by using the compressed *macb* timestamp notation, one can show all the pertinent data without taking too much room, as is highlighted below. Also highlighted, are entries where there are multiple parent directories for one MFT entry (in this case, there are 2 parents for *wordpad.exe*). This means that *wordpad.exe* as a single MFT entry, has two *hard links* to separate directories.



cmdline: ntfswalk -partition "c" -filter_name "wordpad.exe" -csv

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 9 | | | | | | | [root]\Wi windows-wordpad.resou |
| 0x0000c959 | 0x0000c0c3 | file | mui | 1/16/2011 | | | | [root]\W -windows-wordpad.resourc |
| 0x0000c959 | 0x0000c0c3 | file | mui | 1/16/2011 | | | | [root]\W windows-wordpad.reso |
| 0x0002b0f4 | 0x00016fcf | file | pf | 1/20/2012 | 03:58:10.879 | si:[ma.b]; fn:[macb]; fn8.3[macb] | | [root]\Wi E-D7FD7414.pf |
| 0x0002b0f4 | 0x00016fcf | file | pf | 1/21/2012 | 13:30:20.155 | si:[..c.] | | [root]\W XE-D7FD7414.pf |
| 0x000388d2 | 0x00008df2 | file | exe | 11/20/2010 | 12:17:57.834 | si:[m...]; fn:[m...] | multiroots [2] | [root]\Pr \Accessories\wordpad.ex |
| 0x000388d2 | 0x00008df2 | file | exe | 7/30/2011 | 18:36:54.233 | si:[.a.b]; fn:[.a.b] | multiroots [2] | [root]\Pr \Accessories\wordpad.ex |
| 0x000388d2 | 0x00008df2 | file | exe | 7/30/2011 | 18:39:17.316 | fn:[..c.] | multiroots [2] | [root]\P \Accessories\wordpad.exe |
| 0x000388d2 | 0x00008df2 | file | exe | 7/30/2011 | 18:49:38.702 | si:[..c.] | multiroots [2] | [root]\Pro \Accessories\wordpad.exe |
| 0x000388d2 | 0x0003093 | | | | | | multiroots [2] | [root]\W soft-windows-wordpad_3 |
| 0x000388d2 | 0x0003093 | | | | | | multiroots [2] | [root]\W oft-windows-wordpad_31 |
| 0x000388d2 | 0x0003093 | | | | | | multiroots [2] | [root]\W soft-windows-wordpad_31 |
| 0x000388d2 | 0x0003093d | file | exe | 7/30/2011 | 18:49:38.702 | si:[..c.] | multiroots [2] | [root]\W soft-windows-wordpad_31 |
| 0x000389c2 | 0x00008d79 | file | exe | 11/20/2010 | 13:25:35.073 | si:[m...]; fn:[m...] | multiroots [2] | root]\Pro ssories\wordpad.exe |
| 0x000389c2 | 0x00008d79 | file | exe | 7/30/2011 | 18:37:03.219 | si:[.a.b]; fn:[.a.b] | multiroots [2] | root]\Pr essories\wordpad.exe |
| 0x000389c2 | 0x00008d79 | file | exe | 7/30/2011 | 18:39:09.235 | fn:[..c.] | multiroots [2] | root]\P ssories\wordpad.exe |
| 0x000389c2 | 0x00008d79 | file | exe | 7/30/2011 | 18:49:37.485 | si:[..c.] | multiroots [2] | root]\Pro ssories\wordpad.exe |
| 0x000389c2 | 0x00030841 | file | exe | 11/20/2010 | 13:25:35.073 | si:[m...]; fn:[m...] | multiroots [2] | root]\W soft-windows-wordpad_3 |
| 0x000389c2 | 0x00030841 | file | exe | 7/30/2011 | 18:37:03.219 | si:[.a.b]; fn:[.a.b] | multiroots [2] | root]\W oft-windows-wordpad_31b |
| 0x000389c2 | 0x00030841 | file | exe | 7/30/2011 | 18:39:06.848 | fn:[..c.] | multiroots [2] | root]\W soft-windows-wordpad_31 |
| 0x000389c2 | 0x00030841 | file | exe | 7/30/2011 | 18:49:37.485 | si:[..c.] | multiroots [2] | root]\W oft-windows-wordpad_31 |

Search on the string "wordpad.exe"

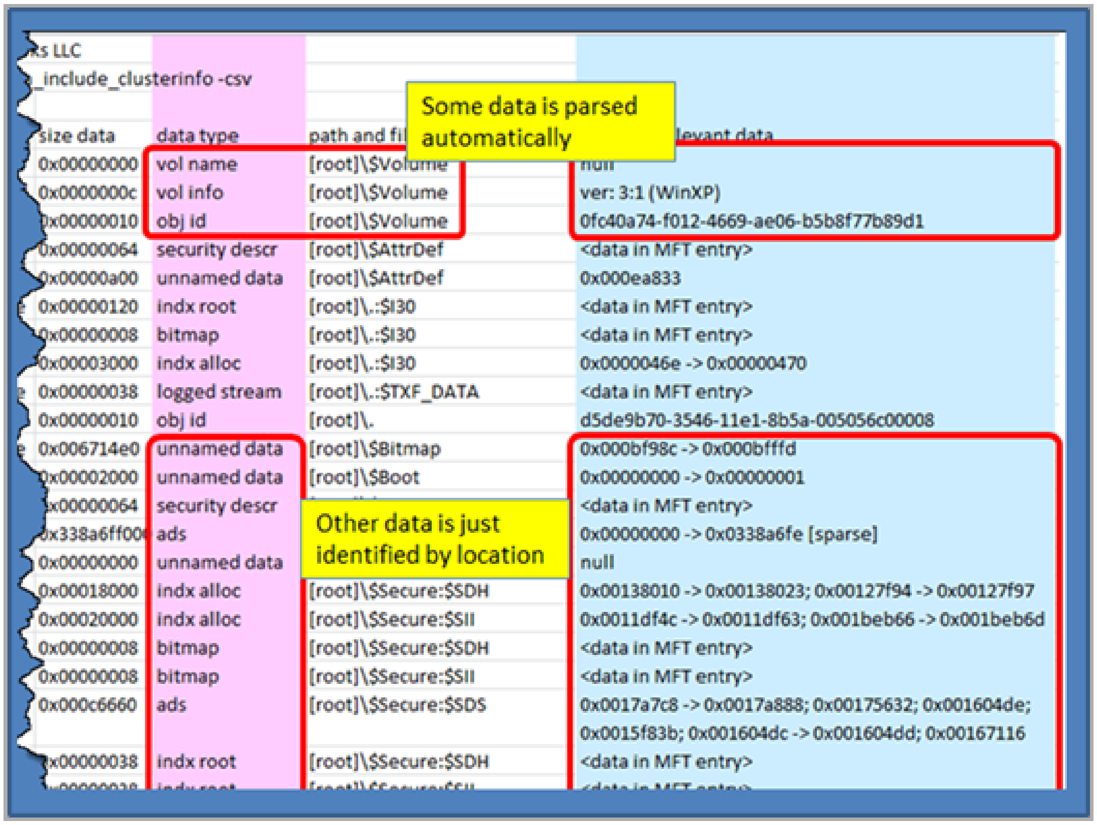
Uses compressed timestamp notation and extracts 'all' timestamp values

Handles MFT entry with multiple directories (an MFT entry w/ multiple parent MFT entries)

Other data that can be extracted from ***ntfswalk*** include cluster information. By using the option *[-action_include_clusterinfo]*, one can view all the cluster information available for each attribute that contains data. Below is an example:

```
    ntfwalk -partition c -action_include_clusterinfo -csv > results.csv
```

The figure shows a snapshot of a sample output. After trimming out some of the rows/cols, one can see the data type, filename and the location where the data resides. For those datasets that are easily parsed, such as the volume information or object identifier, ***ntfswalk*** shows the interpreted data. For other entries, the cluster information is shown, if applicable.



s LLC

_include_clusterinfo -csv

| size data | data type | path and fil | levant data |
|---|---|---|---|
| 0x00000000 | vol name | [root]\$Volume | null |
| 0x0000000c | vol info | [root]\$Volume | ver: 3:1 (WinXP) |
| x00000010 | obj id | [root]\$Volume | 0fc40a74-f012-4669-ae06-b5b8f77b89d1 |
| 0x00000064 | security descr | [root]\$AttrDef | <data in MFT entry> |
| x00000a00 | unnamed data | [root]\$AttrDef | 0x000ea833 |
| 0x00000120 | indx root | [root]\.:$I30 | <data in MFT entry> |
| x00000008 | bitmap | [root]\.:$I30 | <data in MFT entry> |
| 0x00003000 | indx alloc | [root]\.:$I30 | 0x0000046e -> 0x00000470 |
| 0x00000038 | logged stream | [root]\.:$TXF_DATA | <data in MFT entry> |
| 0x00000010 | obj id | [root]\. | d5de9b70-3546-11e1-8b5a-005056c00008 |
| 0x006714e0 | unnamed data | [root]\$Bitmap | 0x000bf98c -> 0x000bfffd |
| x00002000 | unnamed data | [root]\$Boot | 0x00000000 -> 0x00000001 |
| x00000064 | security descr |  | <data in MFT entry> |
| x338a6ff00 | ads |  | 0x00000000 -> 0x0338a6fe [sparse] |
| 0x00000000 | unnamed data |  | null |
| 0x00018000 | indx alloc | [root]\$Secure:$SDH | 0x00138010 -> 0x00138023; 0x00127f94 -> 0x00127f97 |
| 0x00020000 | indx alloc | [root]\$Secure:$SII | 0x0011df4c -> 0x0011df63; 0x001beb66 -> 0x001beb6d |
| 0x00000008 | bitmap | [root]\$Secure:$SDH | <data in MFT entry> |
| 0x00000008 | bitmap | [root]\$Secure:$SII | <data in MFT entry> |
| 0x000c6660 | ads | [root]\$Secure:$SDS | 0x0017a7c8 -> 0x0017a888; 0x00175632; 0x001604de;<br>0x0015f83b; 0x001604dc -> 0x001604dd; 0x00167116 |
| x00000038 | indx root | [root]\$Secure:$SDH | <data in MFT entry> |

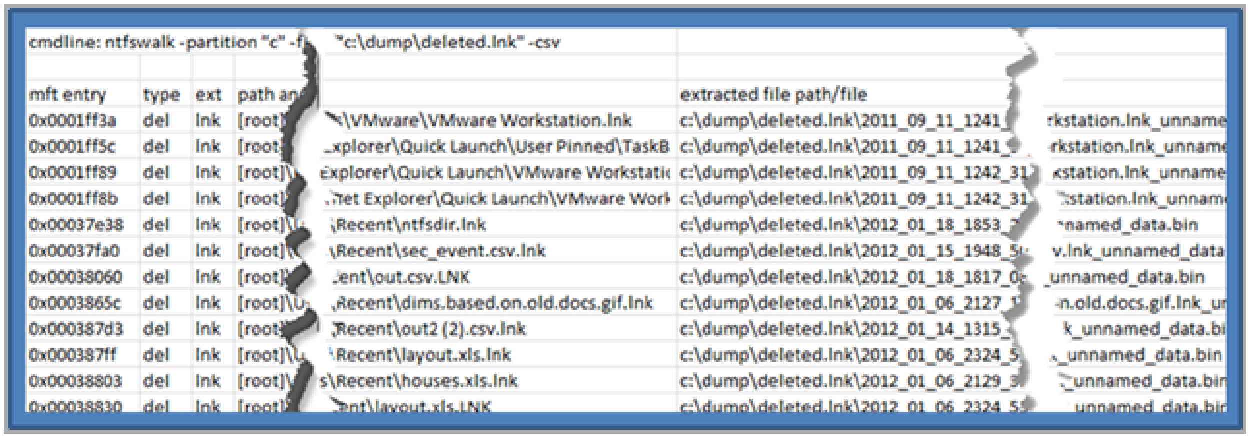
Some data is parsed automatically

Other data is just identified by location

As a third example, if one wishes to extract cluster data associated with a MFT entry, one can use the *[-action_copy_files <directory to store extracted files>]*. The syntax below shows we want to enumerate only those *deleted* files that have an extension of *lnk*. As part of the copy, we tell ***ntfswalk*** to copy each of the clusters associated with these resulting files to a *dump* directory. The syntax of the command is:

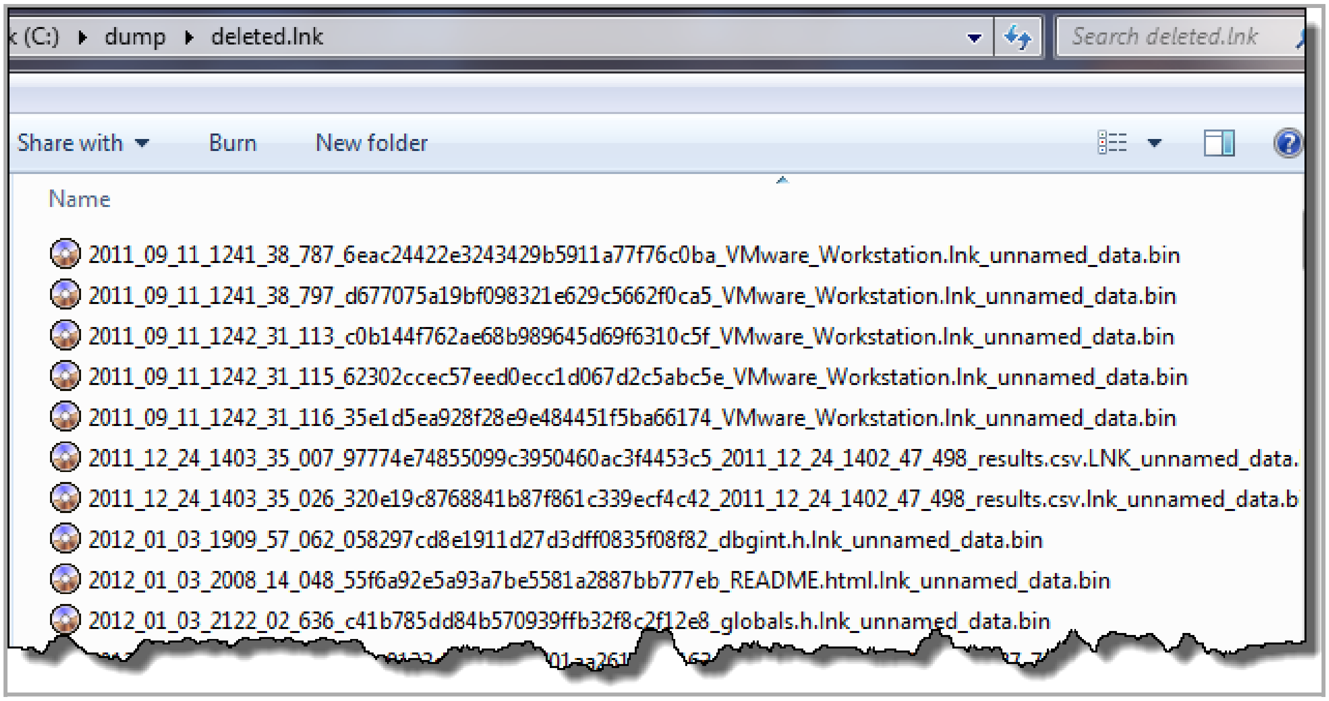
```
    ntfwalk -partition c -filter_deleted_files -filter_ext "lnk" \
        -action_copy_files c:\dump\deleted.lnk -csv > results.csv
```

The first figure shows each MFT entry and the associated path/name of the extracted file. The second figure shows the output of the extracted files. The syntax of the extracted file uses <last modify date>_<md5 hash>_<filename w/ extension>_<data type>.bin



cmdline: ntfswalk -partition "c" -f "c:\dump\deleted.lnk" -csv

| mft entry | type | ext | path an |  | extracted file path/file |  |
|---|---|---|---|---|---|---|
| 0x0001ff3a | del | lnk | [root] | \VMware\VMware Workstation.lnk | c:\dump\deleted.lnk\2011_09_11_1241_ | rkstation.lnk_unname |
| 0x0001ff5c | del | lnk | [root | xplorer\Quick Launch\User Pinned\TaskB | c:\dump\deleted.lnk\2011_09_11_1241_ | rkstation.lnk_unname |
| 0x0001ff89 | del | lnk | [root]\ | Explorer\Quick Launch\VMware Workstati | c:\dump\deleted.lnk\2011_09_11_1242_31 | kstation.lnk_unname |
| 0x0001ff8b | del | lnk | [root] | net Explorer\Quick Launch\VMware Work | c:\dump\deleted.lnk\2011_09_11_1242_31 | station.lnk_unnam |
| 0x00037e38 | del | lnk | [root]\ | \Recent\ntfsdir.lnk | c:\dump\deleted.lnk\2012_01_18_1853_ | named_data.bin |
| 0x00037fa0 | del | lnk | [root]\ | \Recent\sec_event.csv.lnk | c:\dump\deleted.lnk\2012_01_15_1948_5 | v.lnk_unnamed_data |
| 0x00038060 | del | lnk | [root] | ent\out.csv.LNK | c:\dump\deleted.lnk\2012_01_18_1817_0 | unnamed_data.bin |
| 0x0003865c | del | lnk | [root]\U | Recent\dims.based.on.old.docs.gif.lnk | c:\dump\deleted.lnk\2012_01_06_2127_1 | n.old.docs.gif.lnk_u |
| 0x000387d3 | del | lnk | [root | Recent\out2 (2).csv.lnk | c:\dump\deleted.lnk\2012_01_14_1315_ | k_unnamed_data.bi |
| 0x000387ff | del | lnk | [root]\ | \Recent\layout.xls.lnk | c:\dump\deleted.lnk\2012_01_06_2324_5 | _unnamed_data.bin |
| 0x00038803 | del | lnk | [root]\ | s\Recent\houses.xls.lnk | c:\dump\deleted.lnk\2012_01_06_2129_3 | unnamed_data.bin |
| 0x00038830 | del | lnk | [root] | ent\layout.xls.LNK | c:\dump\deleted.lnk\2012_01_06_2324_5 | unnamed_data.bin |



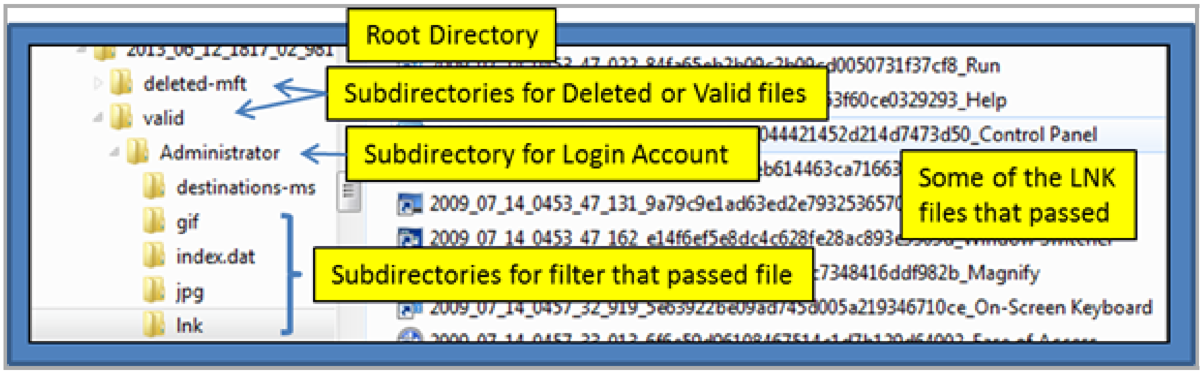
2011_09_11_1241_38_787_6eac24422e3243429b5911a77f76c0ba_VMware_Workstation.lnk_unnamed_data.bin
2011_09_11_1241_38_797_d677075a19bf098321e629c5662f0ca5_VMware_Workstation.lnk_unnamed_data.bin
2011_09_11_1242_31_113_c0b144f762ae68b989645d69f6310c5f_VMware_Workstation.lnk_unnamed_data.bin
2011_09_11_1242_31_115_62302ccec57eed0ecc1d067d2c5abc5e_VMware_Workstation.lnk_unnamed_data.bin
2011_09_11_1242_31_116_35e1d5ea928f28e9e484451f5ba66174_VMware_Workstation.lnk_unnamed_data.bin
2011_12_24_1403_35_007_97774e74855099c3950460ac3f4453c5_2011_12_24_1402_47_498_results.csv.LNK_unnamed_data.
2011_12_24_1403_35_026_320e19c8768841b87f861c339ecf4c42_2011_12_24_1402_47_498_results.csv.lnk_unnamed_data.b
2012_01_03_1909_57_062_058297cd8e1911d27d3dff0835f08f82_dbgint.h.lnk_unnamed_data.bin
2012_01_03_2008_14_048_55f6a92e5a93a7be5581a2887bb777eb_README.html.lnk_unnamed_data.bin
2012_01_03_2122_02_636_c41b785dd84b570939ffb32f8c2f12e8_globals.h.lnk_unnamed_data.bin

## Copying files during the session

Extracting files is a common need, especially when gathering critical data from an incident response request. By using the *-action_copy_files <root directory> [-raw] [-skip_sparse_clusters]* one can direct the files copied to a root directory, but also indicate whether you want file slack (*-raw*option) or to skip sparse clusters (*-skip_sparse_clusters* option).

During the copy operation, various subdirectories within the root directory will be created automatically to store the extracted files. The subdirectories are based on: (a) derived user account, (b) whether the file was deleted or not, and (c) what filter caused the file to be passed. Below is a directory hierarchy that was created based on the ***ntfswalk*** command:

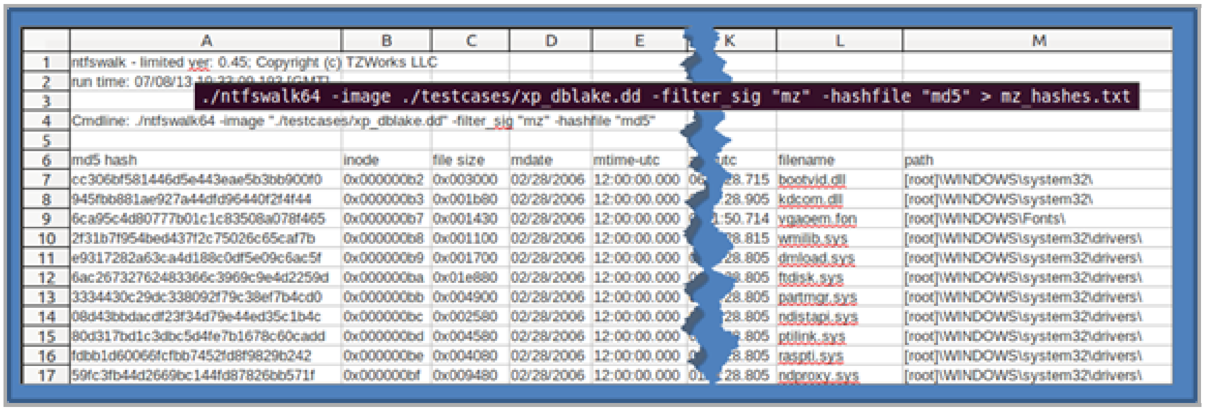
```
ntfswalk -partition c  -filter_ext ".gif | .jpg | .lnk" \
       -filter_name "index.dat | destinations ms" \
       -action_copy_files 2013_06_12_1817_02_981
```


Root Directory
Subdirectories for Deleted or Valid files
Subdirectory for Login Account
Some of the LNK files that passed
Subdirectories for filter that passed file
2013_06_12_1817_02_981
deleted-mft
valid
Administrator
destinations-ms
gif
index.dat
jpg
lnk
d0050731f37cf8_Run
3f60ce0329293_Help
044421452d214d7473d50_Control Panel
b614463ca71663
2009_07_14_0453_47_131_9a79c9e1ad63ed2e7932536570
2009_07_14_0453_47_162_e14f6ef5e8dc4c628fe28ac893
7348416ddf982b_Magnify
2009_07_14_0457_32_919_5e63922be09ad745d005a219346710ce_On-Screen Keyboard

## Generating Hashsets on Target File types

There are a number of excellent tools available on the Internet that perform hashing and creating hash sets. While ***ntfswalk*** was not originally designed to generate hash sets, it does have the ability to hash any desired target file. The main difference between ***ntfswalk***'s approach to that of a normal hash tool, is ***ntfswalk*** accesses the file contents of the file at the cluster level directly, whereas many other hashing tools do not. This becomes more important when considering your target machine may be infected with malware, and whether the actual file contents that are viewed have been masked by malicious software.

Using the switch *-hashfile [md5 | sha1]* will invoke the hashset option. The hashing routine will only target files with data and only the 'unnamed' data streams vice any alternate data streams. Filtering on executable type files is a good way to generate a hashset on any exe, dynamic link library or driver file. Below is an example of running the option on a Linux box targeting a 'dd' image of a NTFS volume:



```
./ntfswalk64 -image ./testcases/xp_dblake.dd -filter_sig "mz" -hashfile "md5" > mz_hashes.txt
```

|  | A | B | C | D | E | K | L | M |
|---|---|---|---|---|---|---|---|---|
| 1 | ntfswalk - limited ver: 0.45; Copyright (c) TZWorks LLC |  |  |  |  |  |  |  |
| 2 | run time: 07/08/13 |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |
| 4 | Cmdline: ./ntfswalk64 -image "./testcases/xp_dblake.dd" -filter_sig "mz" -hashfile "md5" |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |
| 6 | md5 hash | inode | file size | mdate | mtime-utc | utc | filename | path |
| 7 | cc306bf581446d5e443eae5b3bb900f0 | 0x000000b2 | 0x003000 | 02/28/2006 | 12:00:00.000 | 28.715 | bootvid.dll | [root]\WINDOWS\system32\ |
| 8 | 945fbb881ae927a44dfd96440f2f4f44 | 0x000000b3 | 0x001b80 | 02/28/2006 | 12:00:00.000 | .28.905 | kdcom.dll | [root]\WINDOWS\system32\ |
| 9 | 6ca95c4d80777b01c1c83508a078f465 | 0x000000b7 | 0x001430 | 02/28/2006 | 12:00:00.000 | 1:50.714 | vgaoem.fon | [root]\WINDOWS\Fonts\ |
| 10 | 2f31b7f954bed437f2c75026c65caf7b | 0x000000b8 | 0x001100 | 02/28/2006 | 12:00:00.000 | 8.815 | wmilib.sys | [root]\WINDOWS\system32\drivers\ |
| 11 | e9317282a63ca4d188c0df5e09c6ac5f | 0x000000b9 | 0x001700 | 02/28/2006 | 12:00:00.000 | 28.805 | dmload.sys | [root]\WINDOWS\system32\drivers\ |
| 12 | 6ac26732762483366c3969c9e4d2259d | 0x000000ba | 0x01e880 | 02/28/2006 | 12:00:00.000 | .805 | ftdisk.sys | [root]\WINDOWS\system32\drivers\ |
| 13 | 3334430c29dc338092f79c38ef7b4cd0 | 0x000000bb | 0x004900 | 02/28/2006 | 12:00:00.000 | 28.805 | partmgr.sys | [root]\WINDOWS\system32\drivers\ |
| 14 | 08d43bbdacdf23f34d79e44ed35c1b4c | 0x000000bc | 0x002580 | 02/28/2006 | 12:00:00.000 | 28.805 | ndistapi.sys | [root]\WINDOWS\system32\drivers\ |
| 15 | 80d317bd1c3dbc5d4fe7b1678c60cadd | 0x000000bd | 0x004580 | 02/28/2006 | 12:00:00.000 | .805 | ptilink.sys | [root]\WINDOWS\system32\drivers\ |
| 16 | fdbb1d60066fcfbb7452fd8f9829b242 | 0x000000be | 0x004080 | 02/28/2006 | 12:00:00.000 | 8.805 | raspti.sys | [root]\WINDOWS\system32\drivers\ |
| 17 | 59fc3fb44d2669bc144fd87826bb571f | 0x000000bf | 0x009480 | 02/28/2006 | 12:00:00.000 | 28.805 | ndproxy.sys | [root]\WINDOWS\system32\drivers\ |

In the example above, ***ntfswalk*** scanned the contents of every file to see whether it was an executable (independent of what the extension was) or not. If it determined that a PE (or 16 bit version of a exe/dll) signature was present, it computed the MD5 hash of the contents. As you can imagine, this process takes some time depending on the size of the volume you are analyzing.

### For more information

If you would like more information about ***ntfswalk***, contact us via email.

## Downloads

|  | 32-bit Version | 64-bit Version |  |
|---|---|---|---|
| **Windows:** | ntfswalk32.v.0.51.win.zip | ntfswalk64.v.0.51.win.zip | md5/sha1 |
| **Linux:** | ntfswalk32.v.0.51.lin.tar.gz | ntfswalk64.v.0.51.lin.tar.gz | md5/sha1 |
| **Mac OS X:** | ntfswalk.v.0.51.osx.tar.gz | ntfswalk.v.0.51.osx.tar.gz | md5/sha1 |

**32bit apps can run in a 64bit linux distribution if "ia32-libs" (and dependencies) are present.*

Terms of Use | Disclaimer | Contact