**NTFS Partition Boot Sector**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Byte | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| 0x000 | Jump Instruction | | | OEM ID | | | | | | | | **B/S** | | **S/C** | RS | |
| 0x010 | Zero | | | Unused | | MD | Zero | | S/T | | Num Heads | | Hidden Sectors | | | |
| 0x020 | Unused | | | | | | | | Total Sectors | | | | | | | |
| 0x030 | **$MFT Cluster Number** | | | | | | | | $MFTMirr Cluster Number | | | | | | | |
| 0x040 | Clusters / File Record | | | | C/IB | Unused | | | Volume Serial Number | | | | | | | |
| 0x050 | Checksum | | | |  | | | | | | | | | | | |
| 0x060 | Bootstrap Code | | | | | | | | | | | | | | | |
| … | (0x1AA bytes) | | | | | | | | | | | | | | | |
| 0x1F0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | ESM | |

|  |  |  |
| --- | --- | --- |
| B/S : Bytes per Sector  S/C : Sectors per Cluster  RS : Reserved Sectors | MD : Media Descriptor  S/T : Sectors per Track | C/IB : Clusters per Index Buffer  ESM : End of Sector Marker |

When a volume is formatted with NTFS, the first 16 sectors in the partition are allocated for the $Boot metadata file. The first sector is a boot sector, which is laid out in the table above, and the following 15 sectors are part of the boot sector’s initial program loader (IPL). A copy of the boot sector is also stored in the very last sector of an NTFS partition. Below represents a high level of how this looks on disk.

|  |  |  |  |
| --- | --- | --- | --- |
| Cluster 0 | 1 - 15 | … | N - 1 |
| Boot Sector | IPL | Data | Copy of Boot Sector |

**Master File Table**

The Master File Table (MFT) is a set (array) of FILE records, and these records describe each file on disk. Its location is provided in the Boot Sector and is not predefined. This is so if there are bad sectors in its normal location it can be moved. If the Boot Sector and its copy are both corrupted, then there is no conventional way to mount the disk in NTFS. To calculate the MFT location is as follows (input values from Boot Sector):

MFT\_Byte\_Offset = $MFT\_Cluster\_Number \* Sectors\_per\_Cluster \* Bytes\_per\_Sector

Once the MFT location is obtained, we can grab the first entry and determine how the MFT is laid out on disk. Each FILE record contains a Record Header followed by a list of Attributes, which are terminated by an End Marker (0xFFFFFFFF).

|  |
| --- |
| **FILE Record** |
| Record Header |
| Attribute 1 |
| Attribute 2 |
| … |
| Attribute N |
| End Marker (0xFFFFFFFF) |

It is variable in size, but is commonly 1024 bytes (2 sectors) long. The allocated size is given in the Record Header. The offset of the first attribute, calculated from the beginning of the File Record, is also stored here. Below is the structure of a Record Header. Note the magic number (integrity check)—this appears at the beginning of each FILE Record.

**Record Header**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Byte | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| 0x000 | Magic number ‘FILE’ | | | | UpdSeqOff | | UpdSeqLen | | $LogFile Sequence Number (LSN) | | | | | | | |
| 0x010 | Seq Num | | HL Count | | **Attr. Offset** | | Flags | | FILE Record Real Size | | | | FILE Record Alloc’d Size | | | |
| 0x020 | Base FILE Record Reference | | | | | | | | Next AttrId | | Padding | | MFT Record Number | | | |

**Attributes**

There are many different types of attributes. They all start with a Standard Attribute Header followed by their contents. A Standard Attribute header determines the attribute’s length and whether its data is resident (stored in the attribute) or non-resident (stored elsewhere on disk).

Note: If the header contains a name (optional), then it immediately follows—also given by the nameOffset variable.

**Resident Attribute Header**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Byte | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| 0x00 | Attribute Type | | | | Length (including header) | | | | NRF | NL | Name Off. | | Flags | | AttributeID | |
| 0x01 | Attribute Data Length | | | | AttrOffset | | IF | Pad. |  |  |  |  |  |  |  |  |

**Non-Resident Attribute Header**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Byte | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| 0x00 | Attribute Type | | | | Length (including header) | | | | NRF | NL | Name Off. | | Flags | | AttributeID | |
| 0x10 | Starting VCN | | | | | | | | Last VCN | | | | | | | |
| 0x20 | CRO | | CUS | | Padding | | | | Allocated Size | | | | | | | |
| 0x30 | Real Size | | | | | | | | Initialized Stream Size | | | | | | | |

|  |  |
| --- | --- |
| NRF : Non-resident flag (0x00 = Resident; 0x01 = Non-Resident)  NL : Name Length  IF : Indexed Flag | CRO : Cluster Run Offset  CUS : Compression Unit Size |

**Cluster Run**

Following the Non-Resident Attribute Header (and optional name) is a cluster run list. Each run is represented by a starting (offset) cluster and its length. The starting (offset) cluster is coded as an offset to the starting cluster of the *previous* run. The first byte is the header, which is parsed by taking the lower nibble (4 bits) as the number of bytes for the length and the higher nibble as the number of bytes for the offset. Following the header is the length and offset. The list is terminated when the header is zero.

Example 1 – Normal, Unfragmented File

Run: 21 18 34 56 00

Regrouped: 21 18 34 56 – 00

|  |  |
| --- | --- |
| **Run 1** |  |
| Header | 0x21 – 1 byte length, 2 byte offset |
| Length | 0x18 (1 byte) |
| Offset | 0x5634 (2 bytes) |
| **Run 2** |  |
| Header | 0x00 – the end |

Example 2 – Normal, Fragmented File

Run: 31 38 73 25 34 32 14 01 E5 11 02 31 42 AA 00 03 00

Regrouped: 31 38 73 25 34 - 32 14 01 E5 11 02 - 31 42 AA 00 03 – 00

|  |  |
| --- | --- |
| **Run 1** |  |
| Header | 0x31 – 1 byte length, 3 byte offset |
| Length | 0x38 |
| Offset | 0x342573 |
| **Run 2** |  |
| Header | 0x32 – 2 byte length, 3 byte offset |
| Length | 0x114 |
| Offset | 0x363758 (0x211E5 relative to 0x342573) |
| **Run 3** |  |
| Header | 0x31 – 1 byte length, 3 byte offset |
| Length | 0x42 |
| Offset | 0x393802 (0x300AA relative to 0x363758) |
| **Run 4** |  |
| Header | 0x00 – the end |

TODO:

Scrambled File

Sparse File

Compressed File

**Attribute Types**

The following list are the most common attributes that we’ll be working with for recovering data. There are many others, but these are the most important.

|  |  |
| --- | --- |
| **Type** | **Name** |
| 0x10 | $STANDARD\_INFORMATION |
| 0X30 | $FILE\_NAME |
| 0X80 | $DATA |
| 0X90 | $INDEX\_ROOT |
| 0XA0 | $INDEX\_ALLOCATION |
| 0XB0 | $BITMAP |

$STANDARD\_INFORMATION contains DOS Permissions and file times

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Byte | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| 0x00 | C Time – File Creation | | | | | | | | A Time – File Altered | | | | | | | |
| 0x10 | M Time – MFT Changed | | | | | | | | R Time – File Read | | | | | | | |
| 0x20 | DOS File Permissions | | | | Max Num Versions | | | | Version Number | | | | Class ID | | | |
| 0x30 | Owner ID | | | | Security ID | | | | Quota Charged | | | | | | | |
| 0x40 | Update Sequence Number (USN) | | | | | | | |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Byte | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| 0x00 | FILE record number | | | | | | Seq Num | | C Time – File Creation | | | | | | | |
| 0x10 | A Time – File Altered | | | | | | | | M Time – MFT Changed | | | | | | | |
| 0x20 | R Time – File Read | | | | | | | | Allocated size of file | | | | | | | |
| 0x30 | Real size of file | | | | | | | | Flags | | | | Reparse | | | |
| 0x40 | FL | FN | File name in Unicode (1 char = 2 bytes) | | | | | | | | | | | | | |

FL : Filename length in characters

FN : Filename namespace

$DATA

The Data attribute is typically non-resident, unless it’s small enough (few hundred bytes) then it will be resident. For the MFT, the data is always non-resident, so the cluster runs must be parsed. This gives the layout.

**Sample Data Recovery Tutorial**

00000000 EB 52 90 4E 54 46 53 20 20 20 20 00 02 08 00 00 .R.NTFS .....

00000010 00 00 00 00 00 F8 00 00 3F 00 FF 00 00 08 00 00 ........?.......

00000020 00 00 00 00 80 00 00 00 DE 5A F3 00 00 00 00 00 .........Z......

00000030 00 00 0C 00 00 00 00 00 02 00 00 00 00 00 00 00 ................

00000040 F6 00 00 00 01 00 00 00 38 4C 14 40 59 14 40 0C ........8L.@Y.@.

00000050 00 00 00 00 FA 33 C0 8E D0 BC 00 7C FB 68 C0 07 .....3.....|.h..

00000060 1F 1E 68 66 00 CB 88 16 0E 00 66 81 3E 03 00 4E ..hf......f.>..N

00000070 54 46 53 75 15 B4 41 BB AA 55 CD 13 72 0C 81 FB TFSu..A..U..r...

00000080 55 AA 75 06 F7 C1 01 00 75 03 E9 DD 00 1E 83 EC U.u.....u.......

00000090 18 68 1A 00 B4 48 8A 16 0E 00 8B F4 16 1F CD 13 .h...H..........

000000A0 9F 83 C4 18 9E 58 1F 72 E1 3B 06 0B 00 75 DB A3 .....X.r.;...u..

000000B0 0F 00 C1 2E 0F 00 04 1E 5A 33 DB B9 00 20 2B C8 ........Z3... +.

000000C0 66 FF 06 11 00 03 16 0F 00 8E C2 FF 06 16 00 E8 f...............

000000D0 4B 00 2B C8 77 EF B8 00 BB CD 1A 66 23 C0 75 2D K.+.w......f#.u-

000000E0 66 81 FB 54 43 50 41 75 24 81 F9 02 01 72 1E 16 f..TCPAu$....r..

000000F0 68 07 BB 16 68 52 11 16 68 09 00 66 53 66 53 66 h...hR..h..fSfSf

00000100 55 16 16 16 68 B8 01 66 61 0E 07 CD 1A 33 C0 BF U...h..fa....3..

00000110 0A 13 B9 F6 0C FC F3 AA E9 FE 01 90 90 66 60 1E .............f`.

00000120 06 66 A1 11 00 66 03 06 1C 00 1E 66 68 00 00 00 .f...f.....fh...

00000130 00 66 50 06 53 68 01 00 68 10 00 B4 42 8A 16 0E .fP.Sh..h...B...

00000140 00 16 1F 8B F4 CD 13 66 59 5B 5A 66 59 66 59 1F .......fY[ZfYfY.

00000150 0F 82 16 00 66 FF 06 11 00 03 16 0F 00 8E C2 FF ....f...........

00000160 0E 16 00 75 BC 07 1F 66 61 C3 A1 F6 01 E8 09 00 ...u...fa.......

00000170 A1 FA 01 E8 03 00 F4 EB FD 8B F0 AC 3C 00 74 09 ............<.t.

00000180 B4 0E BB 07 00 CD 10 EB F2 C3 0D 0A 41 20 64 69 ............A di

00000190 73 6B 20 72 65 61 64 20 65 72 72 6F 72 20 6F 63 sk read error oc

000001A0 63 75 72 72 65 64 00 0D 0A 42 4F 4F 54 4D 47 52 curred...BOOTMGR

000001B0 20 69 73 20 63 6F 6D 70 72 65 73 73 65 64 00 0D is compressed..

000001C0 0A 50 72 65 73 73 20 43 74 72 6C 2B 41 6C 74 2B .Press Ctrl+Alt+

000001D0 44 65 6C 20 74 6F 20 72 65 73 74 61 72 74 0D 0A Del to restart..

000001E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

000001F0 00 00 00 00 00 00 8A 01 A7 01 BF 01 00 00 55 AA ..............U.

This is the first 0x200 bytes of the NTFS partition. The necessary information to calculate the MFT location are highlighted (Bytes per Sector, Sectors per Cluster, and MFT Cluster Number, respectively).

Bytes per Sector = 0x0200

Sectors per Cluster = 0x08

MFT Cluster Number = 0x00000000000C0000

MFT\_Byte\_Offset = $MFT\_Cluster\_Number \* Sectors\_per\_Cluster \* Bytes\_per\_Sector

0xC0000 \* 0x08 \* 0x200 = 0xC0000000

Navigating on disk to byte location 0xC0000000 (relative to start of NTFS partition), we get the following data.

C0000000 46 49 4C 45 30 00 03 00 E6 45 42 00 00 00 00 00 FILE0....EB.....

C0000010 01 00 01 00 38 00 01 00 A0 01 00 00 00 04 00 00 ....8...........

C0000020 00 00 00 00 00 00 00 00 07 00 00 00 00 00 00 00 ................

C0000030 04 00 00 00 00 00 00 00 **10** 00 00 00 **60** 00 00 00 ............`...

C0000040 00 00 18 00 00 00 00 00 48 00 00 00 18 00 00 00 ........H.......

C0000050 B0 B4 50 10 31 4D D2 01 B0 B4 50 10 31 4D D2 01 ..P.1M....P.1M..

C0000060 B0 B4 50 10 31 4D D2 01 B0 B4 50 10 31 4D D2 01 ..P.1M....P.1M..

C0000070 06 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

C0000080 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 ................

C0000090 00 00 00 00 00 00 00 00 **30** 00 00 00 **68** 00 00 00 ........0...h...

C00000A0 00 00 18 00 00 00 03 00 4A 00 00 00 18 00 01 00 ........J.......

C00000B0 05 00 00 00 00 00 05 00 B0 B4 50 10 31 4D D2 01 ..........P.1M..

C00000C0 B0 B4 50 10 31 4D D2 01 B0 B4 50 10 31 4D D2 01 ..P.1M....P.1M..

C00000D0 B0 B4 50 10 31 4D D2 01 00 40 00 00 00 00 00 00 ..P.1M...@......

C00000E0 00 40 00 00 00 00 00 00 06 00 00 00 00 00 00 00 .@..............

C00000F0 04 03 24 00 4D 00 46 00 54 00 00 00 00 00 00 00 ..$.M.F.T.......

C0000100 **80** 00 00 00 **48** 00 00 00 01 00 40 00 00 00 06 00 ....H.....@.....

C0000110 00 00 00 00 00 00 00 00 BF 00 00 00 00 00 00 00 ................

C0000120 40 00 00 00 00 00 00 00 00 00 0C 00 00 00 00 00 @...............

C0000130 00 00 0C 00 00 00 00 00 00 00 0C 00 00 00 00 00 ................

C0000140 32 C0 00 00 00 0C 00 00 **B0** 00 00 00 **50** 00 00 00 2...........P...

C0000150 01 00 40 00 00 00 05 00 00 00 00 00 00 00 00 00 ..@.............

C0000160 01 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ........@.......

C0000170 00 20 00 00 00 00 00 00 08 10 00 00 00 00 00 00 . ..............

C0000180 08 10 00 00 00 00 00 00 31 01 FF FF 0B 11 01 FF ........1.......

C0000190 00 00 00 00 00 00 00 00 **FF FF FF FF** 00 00 00 00 ................

As you can see, the entry starts off with the magic FILE number. The offset to the first attribute is located at 0x14 and is two bytes long.

Offset to first attribute = 0x38

The attributes listed are as follows:

$STANDARD\_INFORMATION (0x10 type, 0x60 bytes long)

$FILE\_NAME (0x30 type, 0x68 bytes long)

$DATA (0x80 type, 0x48 bytes long)

$BITMAP (0xB0 type, 0x50 bytes long)

0xFFFFFFFF signifies the end of the attribute list

Focusing on $DATA first, we can see that it is non-resident. The following byte is 0x00 so it does not have a name, meaning the cluster runs start at 0x40 (size of non-resident header).

Cluster Run: 32 C0 00 00 00 0C 00

Grouped Run: 32 C0 00 00 00 0C – 00

|  |  |
| --- | --- |
| Run 1 |  |
| Header | 0x32 – 2 byte length, 3 byte offset |
| Length | 0xC0 |
| Offset | 0xC0000 |
| Run 2 |  |
| Header | 0x00 – the end |

The offset 0xC0000 matches the MFT Cluster Number located in the Boot Sector, which makes sense because we are retrieving the location of the MFT from its entry. The length is 0xC0 clusters long. As the MFT grows, it will expand either contiguously if it can, so only the length component in run 1 will change, or it will fragment and thus add more cluster runs.

Now let’s look at the $BITMAP attribute. This keeps track of the active state for each MFT entry in bit form. The attribute is non-resident so we need to get the cluster runs and go there.

Cluster Run: 31 01 FF FF 0B 11 01 FF 00

Grouped Run: 31 01 FF FF 0B – 11 01 FF – 00

|  |  |
| --- | --- |
| Run 1 |  |
| Header | 0x31 – 1 byte length, 3 byte offset |
| Length | 0x01 |
| Offset | 0xBFFFF |
| Run 2 |  |
| Header | 0x11 – 1 byte length, 1 byte offset |
| Length | 0x01 |
| Offset | 0xC00FE (0xFF relative to 0xBFFFF) |
| Run 3 |  |
| Header | 0x00 – the end |

So the MFT Bitmap is located in clusters 0xBFFFF and 0xC00FE. Let’s look at the first 0x100 bytes of the 0xBFFFF cluster.

BFFFF000 FF FF 00 FF FF FF FF FF FF FF FF FF FF FF FF FF ................

BFFFF010 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................

BFFFF020 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................

BFFFF030 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................

BFFFF040 03 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

BFFFF050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

BFFFF060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

BFFFF070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

BFFFF080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

BFFFF090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

BFFFF0A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

BFFFF0B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

BFFFF0C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

BFFFF0D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

BFFFF0E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

BFFFF0F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

Let’s also parse the $FILE\_NAME. The attribute is resident, so we’ll need to move 0x18+0x40 = 0x58 from the beginning of it to get the file name length.

Filename length = 0x04

Filename namespace = 0x03

Unicode filename (2\*0x4 = 0x8) : 24 00 4D 00 46 00 54 00

ASCII filename : $MFT

**Using the Tool**

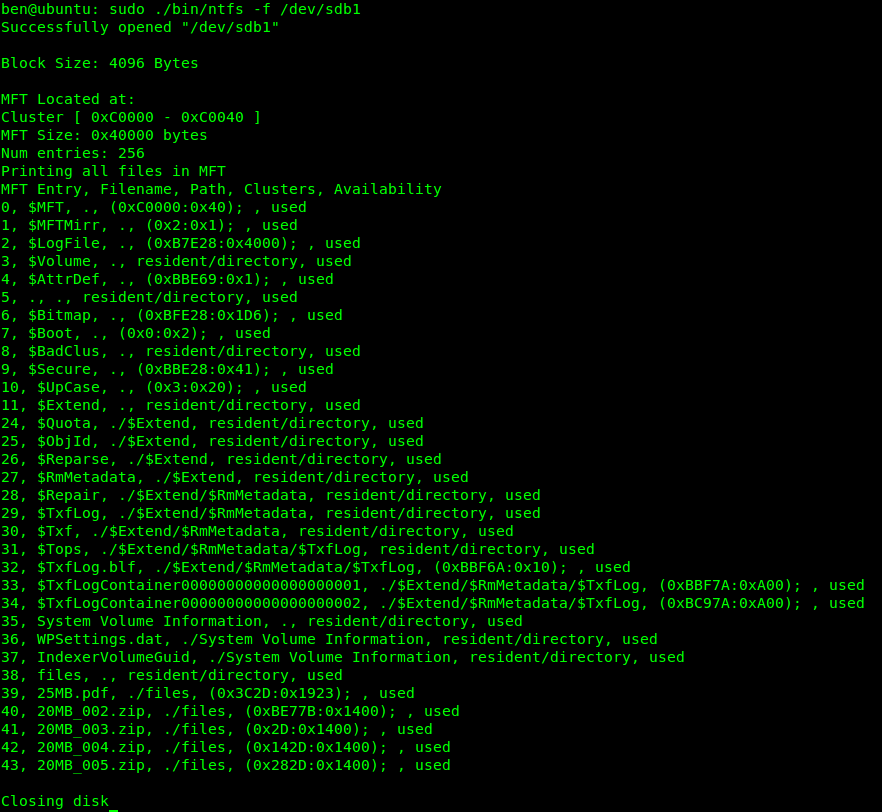
1. Navigate to <digital-forensics-dir>/src/linux
2. Run “make ntfs”
   1. Dependencies: c++/g++ compiler
3. To execute, run “sudo ./bin/ntfs [options] <partition>
4. Output will be in CSV format most of the time

**Commands/Options**

1. Listing all files (-f/-l)
   1. To list all files: sudo ./bin/ntfs -f <partition>
   2. To list all deleted files: sudo ./bin/ntfs -l <partition>
   3. Output will be (comma delimited)
      1. MFT Entry : The index in the MFT table
      2. Filename : Filename of the entry
      3. Path : File path of the entry
      4. Clusters : either cluster runs (offset:length); or resident
      5. Availability : used or free with recoverability percentage
2. Printing a directory (-d)
   1. To print a directory: sudo ./bin/ntfs -d <mft-idx> <partition>
   2. Displays directory index and parent directory index
   3. Output will be (comma delimited)
      1. MFT Entry : The index in the MFT table
      2. isDir? : YES = directory entry, NO = file entry
      3. Filename : Filename of the entry
3. Printing all meta-data of entry (-e)
   1. To print an MFT Entry: sudo ./bin/ntfs -e <entry-index> <partition>
   2. Displays all relevant attributes
   3. Very verbose, mostly helpful for further analysis or debugging
4. Interactive mode
   1. Command: sudo ./bin/ntfs <partition>
   2. Same as (3), but can keep entering indexes to print

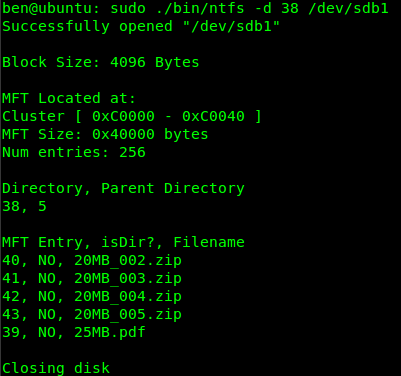
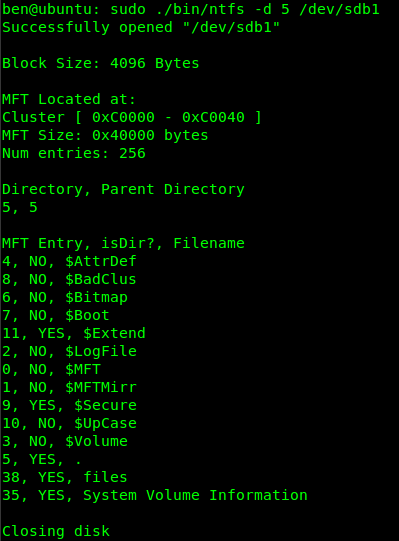
**Sample Outputs**

Command: sudo ./bin/ntfs -f /dev/sdb1



Command: sudo ./bin/ntfs -d 5 /dev/sdb1

Command: sudo ./bin/ntfs -d 38 /dev/sdb1



Experiments

1. Format flash drive to NTFS
2. Copy a few 20MB files (entries 39-43)



1. Delete a file (entry 39). Data can still be recovered.



1. Copy a few 5MB files (entries 39, 44, 45). Notice entry 39 is immediately used along with the newly freed space from the old entry 39.



1. Instead of copying a few smaller files, copy a file larger than 20MB (entry 39). Notice entry 39 is used immediately again, but the freed space is not. NTFS does not want to fragment so it chooses a contiguous space. 25MB.pdf immediately follows 20MB\_005.zip (entry 43) on the disk.

