Fat32 notes

hexedit is your friend!!

You can mount fat32.img as a real filesystem on Mac or Linux VM

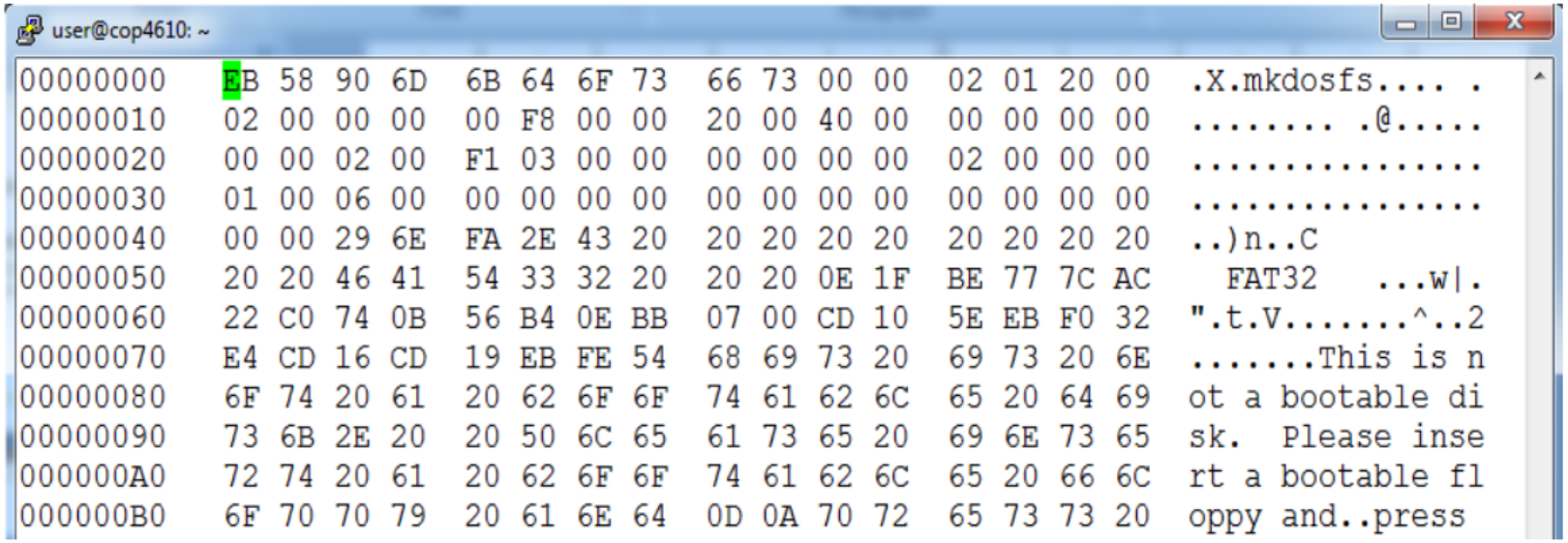
- On Linux do the following commands

```
sudo mount -o loop fat32.img /mnt
cd /mnt
```

  - Once the file is mounted, you can go into the /mnt directory and use all your normal file system commands like ls, cat, cd, etc.

FileSystems are binary structures

- You can't inspect them with a text editor
- You need a hexadecimal file editor, like hexedit.
  - available for Windows, Mac, and Linux
  - displays content in 3 columns: offsets, hex, and ASCII
  - it can modify a file, so be careful!! (Linux *hexdump* can't modify)



Endian-ness count!

- FAT32 structures are little-endian

What is the value of this number??

user@cop4610: ~

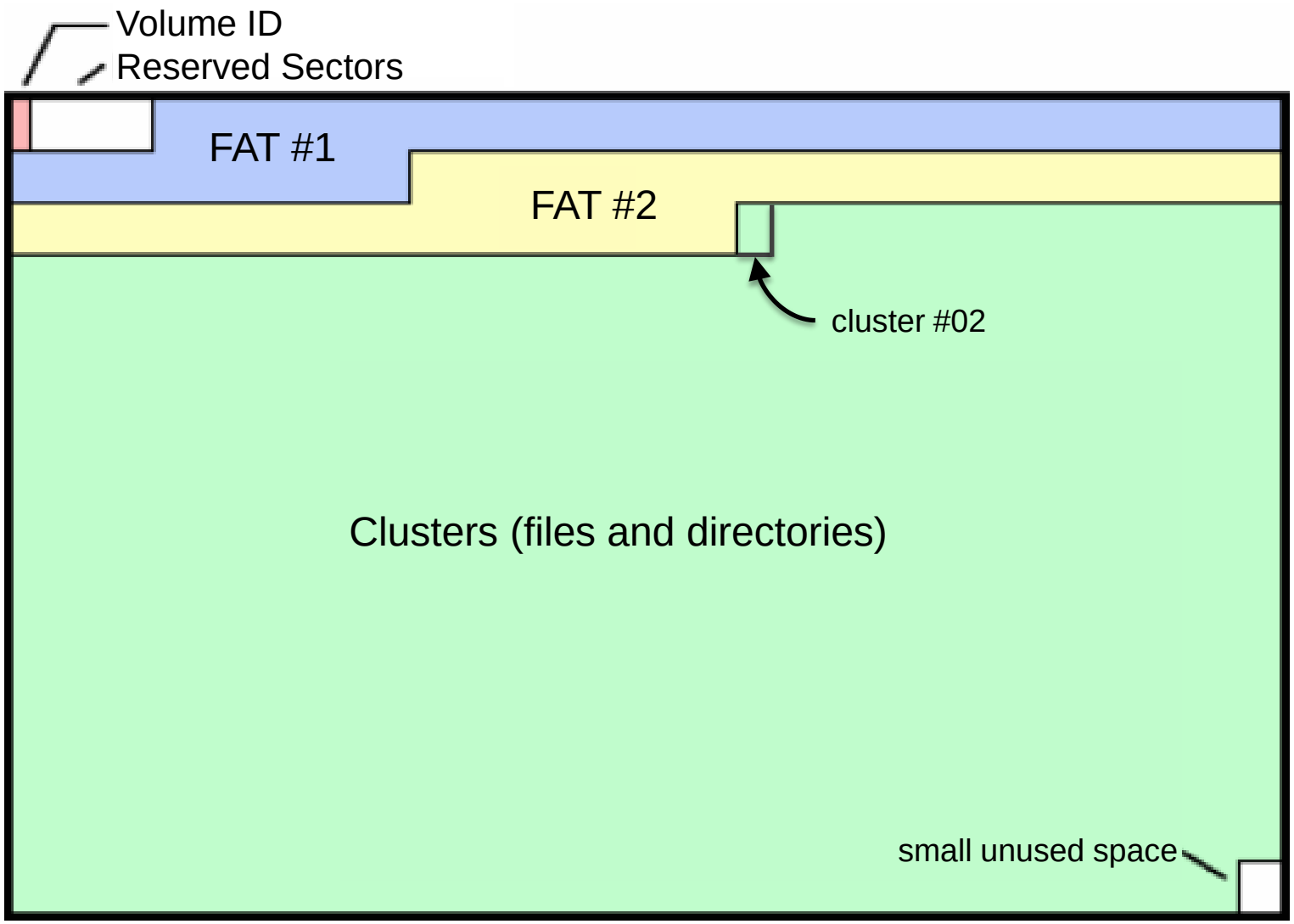
```
00000000   EB 58 90 6D   6B 64 6F 73   66 73 00 00   02 01 20 00   .X.mkdosfs.... .
00000010   02 00 00 00   00 F8 00 00   20 00 40 00   00 00 00 00   ........ .@.....
00000020   00 00 02 00   F1 03 00 00   00 00 00 00   02 00 00 00   ................
00000030   01 00 06 00   00 00 00 00   00 00 00 00   00 00 00 00   ................
00000040   00 00 29 6E   FA 2E 43 20   20 20 20 20   20 20 20 20   ..)n..C
00000050   20 20 46 41   54 33 32 20   20 20 0E 1F   BE 77 7C AC     FAT32   ...w|.
00000060   22 C0 74 0B   56 B4 0E BB   07 00 CD 10   5E EB F0 32   ".t.V.......^..2
00000070   E4 CD 16 CD   19 EB FE 54   68 69 73 20   69 73 20 6E   .......This is n
00000080   6F 74 20 61   20 62 6F 6F   74 61 62 6C   65 20 64 69   ot a bootable di
00000090   73 6B 2E 20   20 50 6C 65   61 73 65 20   69 6E 73 65   sk.  Please inse
000000A0   72 74 20 61   20 62 6F 6F   74 61 62 6C   65 20 66 6C   rt a bootable fl
000000B0   6F 70 70 79   20 61 6E 64   0D 0A 70 72   65 73 73 20   oppy and..press
```

Some FAT Terminology

- Partition = one entire file system
- sector/block = native unit of data on drive
- 2 ways to specify a block
  - CHS = (Old) cluster-head-sector addressing (packed bit fields)
  - LBA = logical block addressing (this is what we use)
- boot sector = first sector of partition
- cluster = group of blocks allocated as one unit
- FAT = file allocation table (linked list)

- FAT and partition meta-data are NOT in clusters!!
- Clusters start after the last FAT table!!

FAT32 Partition Layout



FAT Structure

The *File Allocation Table* (*FAT*) is a contiguous number of sectors immediately following the area of reserved sectors. It represents a list of entries that map to each cluster on the volume. Each entry records one of five things:

- the cluster number of the next cluster in a chain
- a special *end of cluster-chain* (*EOC*) entry that indicates the end of a chain
  - In FAT32, the EOC is indicated by a value in the range 0x0FFFFFF8 to 0x0FFFFFFF
- a special entry to mark a bad cluster
  - In FAT32, 0x0FFFFFF7
- a zero to note that the cluster is unused

XXXXXXXX	XXXXXXXX	00000009	00000004
00000005	00000007	00000000	00000008
FFFFFFFF	0000000A	0000000B	00000011
0000000D	0000000E	FFFFFFFF	00000010
00000012	FFFFFFFF	00000013	00000014
00000015	00000016	FFFFFFFF	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000

Root Directory:
2, 9, A, B, 11

File #1:
3, 4, 5, 7, 8

File #2:
C, D, E

File #3:
F, 10, 12, 13, 14, 15, 16

FAT Structure

Because each entry in the FAT table directly corresponds to a comparably positioned cluster on the disk, the math is straightforward to convert:

- From entry in the FAT table to position on disk
- From position on disk to entry in the FAT table.

XXXXXXXX	XXXXXXXX	00000009	00000004
00000005	00000007	00000000	00000008
FFFFFFFF	0000000A	0000000B	00000011
0000000D	0000000E	FFFFFFFF	00000010
00000012	FFFFFFFF	00000013	00000014
00000015	00000016	FFFFFFFF	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000

Root Directory:
2, 9, A, B, 11

File #1:
3, 4, 5, 7, 8

File #2:
C, D, E

File #3:
F, 10, 12, 13, 14, 15, 16

Important Boot Sector Information

- Size of each region
  - BPB_BytesPerSector
  - BPB_SecPerCluster
  - BPB_RsvdSecCnt
  - BPB_NumFATS
  - BPB_FATSz32
- Root Directory
  - BPB_RootClus
- Warning: this list is not exhaustive

Hex Dump of Boot Sector

sectors per cluster

# of FATs

bytes per sector

sectors before FAT #1

```
EB 58 90 6D  6B 64 6F 73  66 73 00 00  02 01 20 00  .X.mkdosfs.... .
02 00 00 00  00 F8 00 00  20 00 40 00  00 00 00 00  ........ .@.....
00 00 02 00  F1 03 00 00  00 00 00 00  02 00 00 00  ................
01 00 06 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
```

sectors per FAT

cluster # of root directory

Field	Microsoft's Name	Offset	Size	Value
Bytes Per Sector	BPB_BytsPerSec	0x0B	16 Bits	Always 512 Bytes
Sectors Per Cluster	BPB_SecPerClus	0x0D	8 Bits	1,2,4,8,16,32,64,128
Number of Reserved Sectors	BPB_RsvdSecCnt	0x0E	16 Bits	Usually 0x20
Number of FATs	BPB_NumFATs	0x10	8 Bits	Always 2
Sectors Per FAT	BPB_FATSz32	0x24	32 Bits	Depends on disk size
Root Directory First Cluster	BPB_RootClus	0x2C	32 Bits	Usually 0x00000002

Finding the Root Directory

1. Figure out the ***root directory cluster number*** from the boot sector, not the FAT (see previous slide)
2. Figure out where the root directory starts in the data region
3. Read in the root directory structure located at the first sector of the root directory cluster
4. Does the root directory span more than 1 cluster? Find out by looking up the root directory's ***next cluster number*** in the FAT.
   - Convert from root cluster to FAT entry
   - Check the FAT entry for the root cluster. FAT will give you either the next cluster number in the directory or the ***End of Cluster Chain*** value (0x0FFFFFF8)
   - If not EoCC, proceed to the next cluster

Finding Files and Directories

- Each directory is made up of one or more ***directory entries*** that contain
  - file/subdirectory name (maybe long and short forms)
  - attributes
  - first cluster number
  - more …

Directory Entries

- List names of files and subdirectories in a directory
- 2 entry types
  - Long-name directory entry (one or more)
    - Starts with 'A' (0x41)
    - precede corresponding short-name directory entry
    - give a file name in Unicode 16-bit-per-character encoding
  - Short-name directory entry
    - limits name size to 8 bytes with 3-byte file extension: ALL CAPS
    - compatibility with previous FAT versions
    - 32 bytes total size
    - other important metadata: attributes, file size, first cluster number, etc.
    - If DIR_Name[0] == 0xE5, then the entry is free (no current file or directory name in this entry)
    - if DIR_Name[0] == 0x00, then the entry is free AND there are no allocated directory entries after this one.

FAT32 Short-Name Directory Layout

32-byte directory layout; short filename version

Name	Offset (byte)	Size (bytes)	Description
DIR_Name	0	11	Short Name
DIR_Attr	11	1	File Attributes (More on it later)
DIR_NTRes	12	1	Reserved for Windows NT
DIR_CrtTimeTenth	13	1	Millisecond stamp at file creation time
DIR_CrtTime	14	2	Time file was created
DIR_CrtDate	16	2	Date file was created

Name	Offset (byte)	Size (bytes)	Description
DIR_LstAccDate	18	2	Last access date
DIR_FstClusHI	20	2	High word of this entry's first cluster number
DIR_WrtTime	22	2	Time of last write
DIR_WrtDate	24	2	Date of last write
DIR_FstClusLO	26	2	Low word of this entry's first cluster number
DIR_FileSize	28	4	32-bit DWORD holding this file's size in bytes

DIR_Attr bit flags:

Bit	7	6	5	4	3	2	1	0
Attribute	Reserved. Set to 0		Archive	Directory	Volume ID	System	Hidden	Read-only

Hex Dump of a Root Directory

- Pink entries are long name entries.
  - Start with “A”, low nibble of DIR_Attr is F.
- Blue entries are short-name directory entries

Attrib.

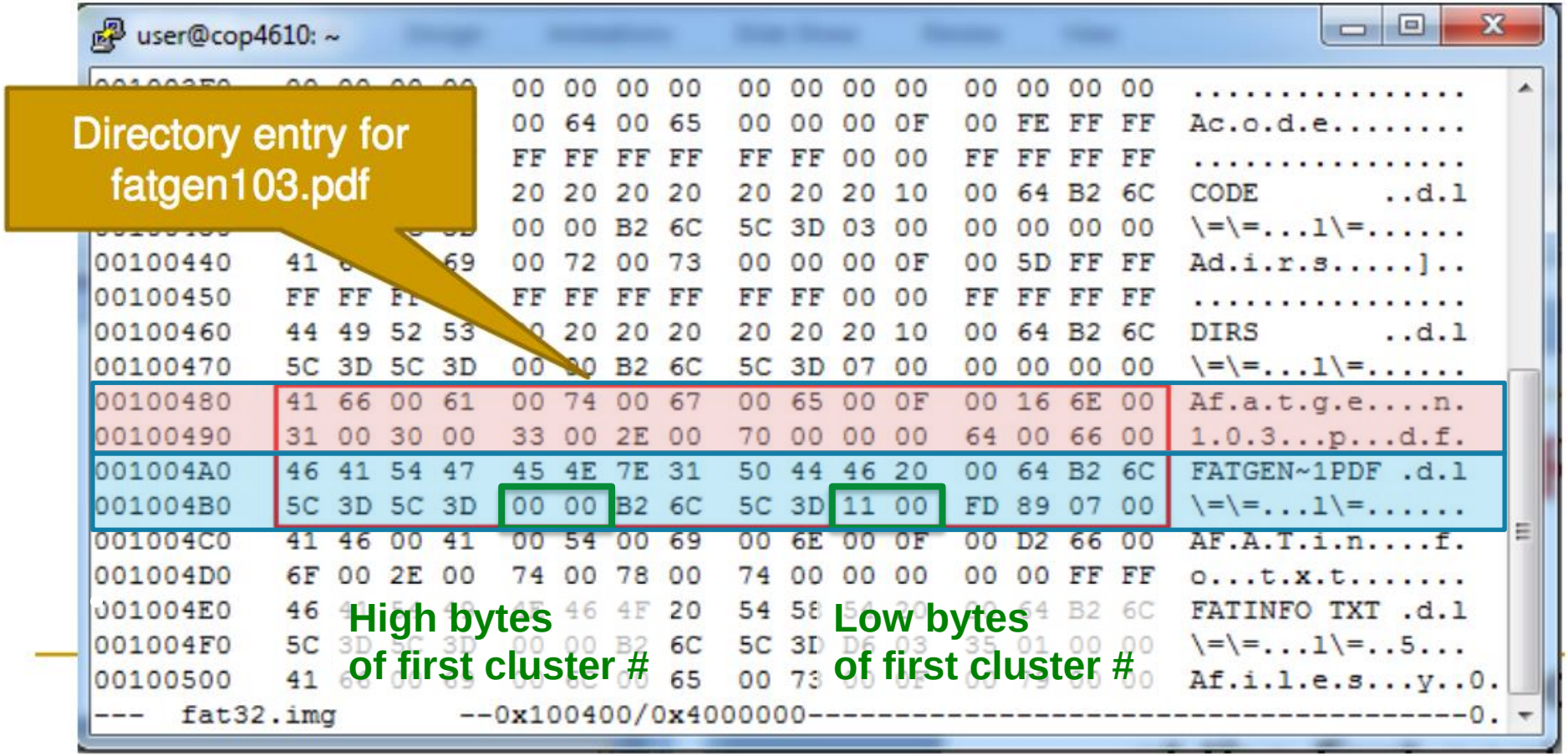
```
43 48 55 43  4B 4C 45 53  20 20 20 08  00 00 53 3E  CHUCKLES   ...S>
87 44 87 44  00 00 53 3E  87 44 00 00  00 00 00 00  .D.D..S>.D......
41 66 00 73  00 69 00 6E  00 66 00 0F  00 0F 6F 00  Af.s.i.n.f....o.
2E 00 74 00  78 00 74 00  00 00 00 00  FF FF FF FF  ..t.x.t.........
46 53 49 4E  46 4F 20 20  54 58 54 20  00 64 BB 36  FSINFO  TXT .d.6
87 44 87 44  00 00 BB 36  87 44 03 00  60 01 00 00  .D.D...6.D..`...
41 65 00 6D  00 70 00 74  00 79 00 0F  00 6D 2E 00  Ae.m.p.t.y...m..
74 00 78 00  74 00 00 00  FF FF 00 00  FF FF FF FF  t.x.t...........
45 4D 50 54  59 20 20 20  54 58 54 20  00 00 C0 36  EMPTY   TXT ...6
87 44 87 44  00 00 C0 36  87 44 00 00  00 00 00 00  .D.D...6.D......
41 63 00 6F  00 6E 00 73  00 74 00 0F  00 0A 2E 00  Ac.o.n.s.t......
74 00 78 00  74 00 00 00  FF FF 00 00  FF FF FF FF  t.x.t...........
43 4F 4E 53  54 20 20 20  54 58 54 20  00 00 C3 36  CONST   TXT ...6
87 44 87 44  00 00 C3 36  87 44 04 00  3F B0 00 00  .D.D...6.D..?...
E5 73 00 65  00 63 00 72  00 65 00 0F  00 AE 74 00  .s.e.c.r.e....t.
2E 00 74 00  78 00 74 00  00 00 00 00  FF FF FF FF  ..t.x.t.........
E5 45 43 52  45 54 20 20  54 58 54 20  00 00 C5 36  .ECRET  TXT ...6
87 44 87 44  00 00 C5 36  87 44 5D 00  FC 00 00 00  .D.D...6.D].....
```

0xE5 = free entry

First Clusters

File Sizes

Finding files: Example



File lives at cluster 0x11. Do math to get the byte address of the beginning of cluster 0x11 and go there…

```
user@cop4610: ~
001021F0   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
00102200   25 50 44 46  2D 31 2E 34  0A 25 C3 A4  C3 BC C3 B6  %PDF-1.4.%......
00102210   C3 9F 0A 32  20 30 20 6F  62 6A 0A 3C  3C 2F 4C 65  ...2 0 obj.<</Le
00102220   6E 67 74 68  20 33 20 30  20 52 2F 46  69 6C 74 65  ngth 3 0 R/Filte
00102230   72 2F 46 6C  61 74 65 44  65 63 6F 64  65 3E 3E 0A  r/FlateDecode>>.
00102240   73 74 72 65  61 6D 0A 78  9C ED 3D C9  8E 1C B9 72  stream.x..=....r
00102250   F7 FA 8A 3A  0F 50 6D EE  99 04 84 06  7A 35 E0 DB  ...:.Pm.....z5..
```

Does the file continue after this cluster?

- Look up cluster 0x11 in the FAT (cluster 17)

Continues to cluster 0x12=18!

```
user@cop4610: ~
00004000   F8 FF FF 0F  FF FF FF 0F  F8 FF FF 0F  FF FF FF 0F  ................
00004010   FF FF FF 0F  FF FF FF 0F  FF FF FF 0F  FF FF FF 0F  ................
00004020   FF FF FF 0F  FF FF FF 0F  FF FF FF 0F  FF FF FF 0F  ................
00004030   FF FF FF 0F  FF FF FF 0F  FF FF FF 0F  FF FF FF 0F  ................
00004040   FF FF FF 0F  12 00 00 00  13 00 00 00  14 00 00 00  ................
00004050   15 00 00 00  16 00 00 00  17 00 00 00  18 00 00 00  ................
00004060   19 00 00 00  1A 00 00 00  1B 00 00 00  1C 00 00 00  ................
```

What about Sub-directories?

- ATTR_Directory flag is set in the directory entry
- Treated just like a file in terms of cluster allocation
- Clusters contain 32 bytes directory entries for the files and directories under this directory

Machine Endianness

- The endianness of a given machine determines in what order a group of bytes are handled (ints, shorts, long longs)
  - Big-endian – most significant byte first
  - Little-endian – least significant byte first
- This is important to understand for this project, since FAT32 is always formatted as little-endian

FAT32 Endianness

- The following are a few cases where endianness matters in your project:
  - Reading in multi-byte values from the FAT32 image
  - Reading in shorts from a FAT32 image
  - Combining multiple shorts to form a single integer from the FAT32 image
  - Interpreting directory entry attributes

Visualizing Endianness
Value = 13371337 (0x00CC07C9)

index	0	1	2	3
little endian	0xC9	0x07	0xCC	0x00
big endian	0x00	0xCC	0x07	0xC9

Next Steps

- Take a look at the FAT32 specification file from Microsoft
  - Somewhat confusing – just like the real world
- Take a look at the image file
  - You *may* mount it in Linux, but that doesn't contribute to your project