# Chapter 6
# Deleted File Recovery in FAT

**Learning Objectives**
The objectives of this chapter are to:

- Understand the principles of data recovery
- Understand the principles of file creation and file deletion in FAT file system
- Understand how to recover deleted files based on remaining file system metadata, particularly through manually recovering the deleted files in a FAT file system

In this chapter, you'll start to explore how file recovery works. It is very common to recover deleted or lost files during an investigation. When a file is deleted by an Operating System, the file content (data) stored in the file system is not destroyed immediately, remaining intact until it is overwritten by other files. Furthermore, many file systems don't completely destroy file system meta-data of the deleted file. It becomes, thus, highly likely and straightforward to perform file recovery based on residual metadata after deletion. In the chapter, we'll show you what happens to a file when it is created and deleted in FAT file system. Then, you'll learn how to recover a deleted file based on residual file system metadata remaining after deletion in FAT file systems.

## 6.1 Principles of File Recovery

Digital devices such as cellular phones, PDAs, laptops, desktops and a myriad of data storage devices pervade many aspects of life in today's society. The digitization of data and its resultant ease of storage, retrieval and distribution have revolutionized

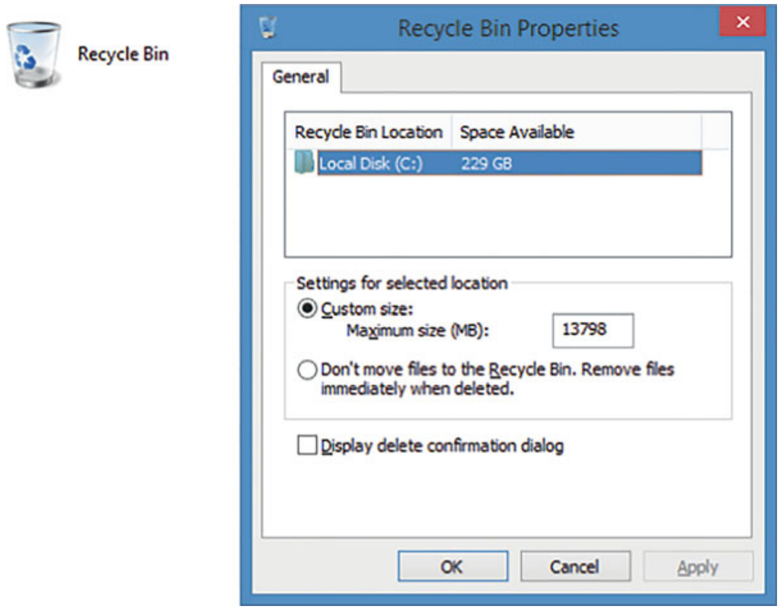
X. Lin, *Introductory Computer Forensics*,
https://doi.org/10.1007/978-3-030-00581-8_6

our lives in many ways. For example, we have witnessed the success of Internet business (e-commerce or online commerce), allowing people to buy products, without visiting a real store. Unfortunately, the digital age has also given rise to digital crime where criminals use digital devices in the commission of unlawful activities like hacking, identity theft, embezzlement, child pornography, theft of trade secrets, etc. Increasingly, digital devices like computers, cell phones, cameras, etc. are found at crime scenes during a criminal investigation. Consequently, there is a growing need for investigators to search digital devices for data evidence including emails, photos, video, text messages, transaction log files, etc., which can assist in the reconstruction of a crime and identification of the perpetrator.

Unfortunately, it is also very often that criminals try to hide their wrongdoings by deleting these digital evidences, which could indict them in the court. Nevertheless, if these digital evidences haven't been overwritten or zeroed-out, the data may still be there, and can be recovered. Actually, when a file is permanently deleted in a file system, the file system no longer provides any means for retrieving the file and marks the data units (clusters or blocks) previously allocated to hold the deleted file content as unallocated hence available for reusing by other files. It is worth noting that many modern Operating Systems provide the user with a means to recover deleted files, particularly from Recycle Bin due to the fact that once a file is deleted from your computer, it's actually just moved to the Recycle Bin, a predefined special file directory, where it's temporarily stored until the Recycle Bin is emptied. In this book, however, we will consider scenarios where there is no ways for the user to gain access to the deleted files using regular file-browsing tools in Operating Systems. Although the file appears to have been erased, its data is still largely intact until it is overwritten by another file.

The Recycle Bin only stores files deleted from hard drives, not from removable media, such as USB Flash Drives or memory cards. Nor does it store files deleted from network drives. The actual location of the Recycle Bin may vary depending on the type of Operating System and file system used. On FAT file systems (typically Windows 98 and prior), it is located in Drive:\RECYCLED. In the NTFS filesystem (Windows 2000, XP, NT), it is Drive:\RECYCLER. On Windows Vista and Windows 7, it is Drive:\$Recycle.Bin folder. For example, Fig. 6.1 shows the Recycle Bin in Windows 7, which is located in a hidden directory named \$Recycle.Bin\%SID%, where $RECYCLE.BIN is a system and hidden folder and %SID% is the security identifier (SID) of the user that performed the deletion. A Recycle Bin can also be configured to remove files immediately when deleted, as shown in Fig. 6.1, to enable the check box provided.

Also, data could be lost or missing due to many other reasons, for example, hardware failure, accidentally deleting a file (or folder) or formatting a disk partition by mistake. Therefore, it is important and crucial to recover files that have been deleted or lost. For example, when the Enron scandal [1] surfaced in October 2001, top executives deleted thousands of e-mails and digital documents in an effort to



| Recycle Bin Location | Space Available |
|---|---|
| Local Disk (C:) | 229 GB |

Settings for selected location

(● Custom size:) Maximum size (MB): 13798

○ Don't move files to the Recycle Bin. Remove files immediately when deleted.

☐ Display delete confirmation dialog

**Fig. 6.1** The Recycle Bin in Windows 7

cover up their fraud. As one of the decade's most fascinating criminal trials against corporate giant Enron, it was successful largely due to the digital evidence in the form of over 200,000 emails and office documents recovered from computers at their offices. Digital forensics or computer forensics is an increasingly vital part of law enforcement investigations and is also useful in the private sector for disaster recovery plans for commercial entities that rely heavily on digital data, where data recovery plays an important role in the computer forensics field. Therefore, data recovery has become one popular task for digital investigator.

Data recovery can be done mainly on two possible copies of deleted or lost data, the primary (original) copy and backup copy. Data recovery effort should first be made on the primary (original) copy because of two reasons. One is that the primary copy has current or up-to-date data. Second, the backup copy may not exist in many cases. For example, if there is no contingency plan in an organization, there could be no backup copy available.

The most common form of inaccessible data is different types of deleted or lost files, including photos, documents, source code files, photos, videos, audio, email files (e.g., Microsoft Outlook Data Files (.pst)) etc. However, data recovery can be the recovery of any other forms of data which cannot be accessed in a normal way. For example, nowadays, mobile devices are widely used, and also can be used for many different applications. Consider social networking and mobile chat apps: There is a vast amount of information that can be shared (and potentially stored on your device) via these two categories of applications alone, such as messages and geographic locations (GPS information). The above information could be of extreme importance to a forensic investigation, whether a victim or a perpetrator is being investigated. Many of these applications use SQLite, an open source, embedded relational database, to manage their data. As a result, data recovery could mean the recovery of deleted database records. Nevertheless, deleted file recovery is the focus of this chapter, particularly recovering deleted files in FAT file systems.

There are various different approaches to recover deleted files. According to whether or not file system data has been used for recovery, data recovery techniques can be classified into two categories: Residual file system metadata based approaches and file carving.

When a file is deleted, the operating system updates some file system data so the file system no longer provides any means for the user to access the deleted file. However, in many cases, in addition to file data left intact, some important file system metadata information about the deleted file is not completely wiped out. Obviously, it is very straightforward to recover deleted files by using remaining file metadata information. Nevertheless, these traditional recovery methods that make use of file system structure presented on storage devices become ineffective when the file system structure is corrupted or damaged, a task easily accomplished by a savvy criminal or disgruntled employee with freely available tools. A more sophisticated data recovery solution which does not rely of this file system structure is therefore necessary. These new and sophisticated solutions are collectively known as file carving. File carving is the technique of recovering files from a block of binary data without using any information available in the file system structure.

In this book, we distinguish two terms "file deletion" and "file wipeout", although they are sometimes used interchangeably. File wipeout refers to the file contents completely destroyed, for example, overwritten, or zeroed-out. It means the file has been physically removed. Once file contents are physically destroyed in this way, the deleted file cannot be recovered. It is also known as secure deletion.

## 6.2 File Creation and Deletion in FAT File Systems

We now look at what happens to a file when it is created and deleted, particularly in FAT file systems. For ease of presentation, we will always discuss the scenarios about files in the root directory. This is because as to files in the other directories, the only extra effort is that we would need to locate the directory which contains the file we are looking for. This can be done by going through the entire file path, starting from the top-most directory (or root directory) to its next sub level until the directory containing the file is reached.

### 6.2.1 *File Creation*

The file system is an integral component of the operating system of a digital device. It is composed of files of many different types with varying sizes and functions. There are system files, executable program files, text files, image files, etc. The file system organizes and manages these files and keeps track of various properties like the file name, security permissions, created and last accessed dates, deleted status, data clusters on disk where the file is stored, information about unused space on the device, etc.

When a file is newly created, the Operating System is responsible for searching the file system for the best location to store the file to ensure fast and efficient retrieval later on. Usually, the Operating System first searches its unallocated space for a block of consecutive clusters large enough to hold the entire file. However, free space is often broken into chunks over time as files are created and deleted and so the largest chunk of unallocated space may not be large enough to hold the entire file. In such cases, the file must be broken into smaller pieces so that each piece can be stored in a separate chunk of free space on the device. Such a file is said to be fragmented, as shown by the example (file *2015_16.xlsx*) in Fig. 6.2.

Next, we will explain the steps for creating a file in a FAT file system. As shown in Fig. 6.2, a FAT file system is divided into three areas, including reserved area, FAT table(s), and data area for the file content. All the files are organized in a hierarchical structure, starting with the top most folder, root folder or directory. A directory in a FAT file system is represented by Directory Table (DIR), containing an entry for each file or sub-directory including its name, extension, meta-data (e.g. created date and time, etc.), permissions, size, and most importantly the address of the starting cluster for the file. The FAT area contains indexed entries for each data block on the disk indexed by the block number. Each index contains one of the following four entries [2, 3]:

- Unused (0x0000)
- Bad cluster (0xFFF7)
- Address of next cluster in use by a file
- Last cluster in a file (0xFFF8–0xFFFF)

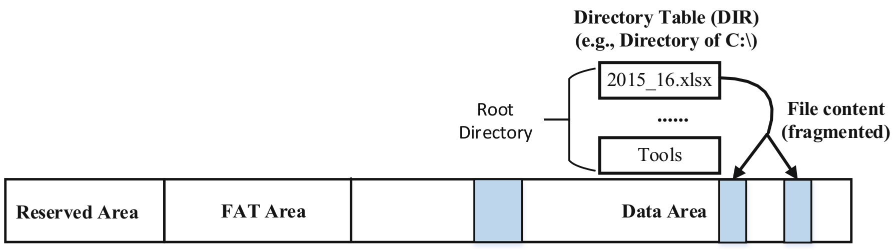


**Fig. 6.2** Layout of FAT file system

Most of the time, the terms Directory and Folder are used interchangeably to describe a location for storing files and/or other directories (subdirectories) on your computer, however there is one distinct feature that separates these terms. Directory is a term in file system, while a Folder is a term used by GUI in an Operating System like Windows. So, sometimes, folders are not physical directories, for example printers. For ease of presentation, hereafter unless otherwise specified the terms Directory and Folder are used interchangeably.

In an FAT file system, the procedure for the file creation can be illustrated as below:

- First, the Operating System checks if the file system has sufficient disk space to store the contents of the newly created file. If not, an "Insufficient disk space" error message appears and the file creation fails. Otherwise, a certain number of clusters will be allocated to the file, and their status becomes allocated and unavailable to other files or folders.
- Second, according to the file path, an entry of the directory that contains the file will be assigned to the file, where some important information about the file, including its name, extension, created date/time, permissions, and size. Most importantly, the address of the starting cluster for the file, will be filled.
- Third, the chain of clusters that are allocated to the file will be created in the FAT table of the file system. In FAT file system, each cluster has a corresponding FAT table entry with the same sequence number, for example, FAT table entry 2 stands for cluster 2. Also, each FAT entry holds either the address of next cluster in use by the file or a special mark (e.g., 0xfffffff in FAT32), indicating the last cluster in a file (or the end-of-cluster chain). In the example shown in Fig. 6.3, the newly created file *file 1.dat* occupies clusters 8–10. FAT table entry 8 represents cluster 8, and it contains a value 9, indicating that the next cluster in use by the file is cluster 9, whereas FAT table entry 10 contains EOF, indicating that cluster 10 is the last cluster allocated to the file.

### *6.2.2 File Deletion*

When a file is deleted in a FAT file system, the operating system only updates the DIR entry where the first character of the file name (or the first byte of the directory entry) is set to a special character (0xe5 in hex). The directory entry becomes a deletion entry. The character **0xE5** indicates to the system that the directory entry is available for use by a new entry, but no other change is made to this directory entry. In other words, the rest of the file name as well as meta-data (created date/time, permissions, size, and most importantly the address of the starting cluster for the file) is kept intact. At the same time, all the FAT entries corresponding to the deleted file are zeroed out to indicate that the corresponding clusters are available for use. However, the operating system does not erase the actual contents of the data clusters.

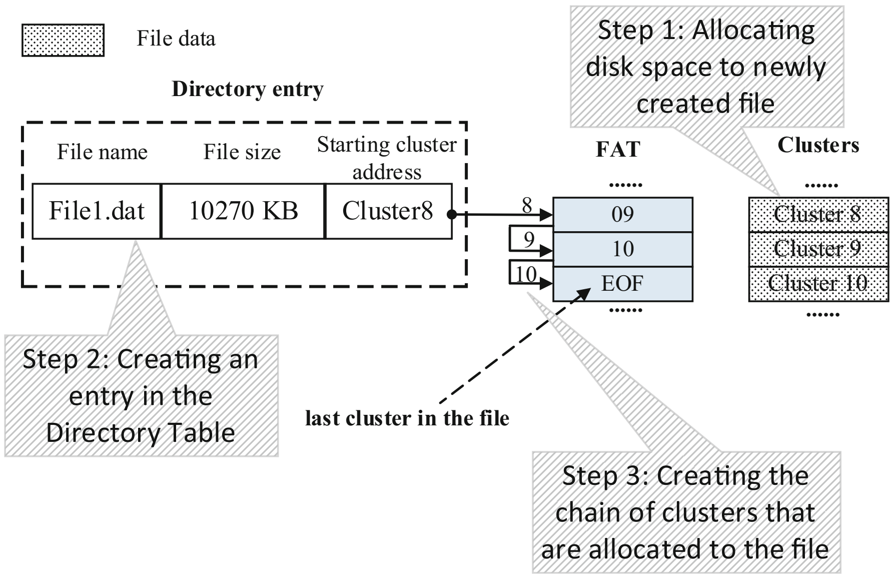

File data
Directory entry
File name
File size
Starting cluster address
File1.dat
10270 KB
Cluster8
Step 1: Allocating disk space to newly created file
FAT
Clusters
8
9
10
09
10
EOF
Cluster 8
Cluster 9
Cluster 10
Step 2: Creating an entry in the Directory Table
last cluster in the file
Step 3: Creating the chain of clusters that are allocated to the file

**Fig. 6.3** File creation

In other words, the data of the file remains untouched in the data area. As shown in Fig. 6.4b, it shows its relevant remaining information in the system after *File.txt* (in the example Fig. 6.4a) is deleted from the system. The two cardinal changes are: "*File.txt*" updates to "*_ile.txt*" and the FAT cluster chain are wiped out in directory entry.

## 6.3 Deleted File Recovery in FAT File Systems

Obviously, it is very straightforward to recover deleted files by using remaining file metadata information in a FAT file system given that most of files are stored in contiguous blocks. First, we scan the entire file system one directory entry at a time and compile a list of entries with a deletion marker (e.g. 0xE5). Then, we simply change the first character of the file name back to its original one from 0xE5. Then, we put back the cluster chain into FAT since we know the starting cluster address from the DIR entry as well as the file is stored on consecutive disk blocks. The detailed recovery process will be elaborated on below.

Assume that a file is stored in a hard disk contiguously and without fragmentation, which happens to be the most common scenario for file storage, i.e., contiguous storage allocation. In order words, the list of clusters in use by a file will progress in a linear fashion (for examples 26, 27, 28, 29 if a file occupies 4 clusters, starting with cluster 26.). It is worth noting that in this chapter, we only consider the above scenario. But when a deleted file is fragmented, then it becomes very challenging to recover it. The state-of-the-art of this will be elaborated on in Chap. 9 on file carving.

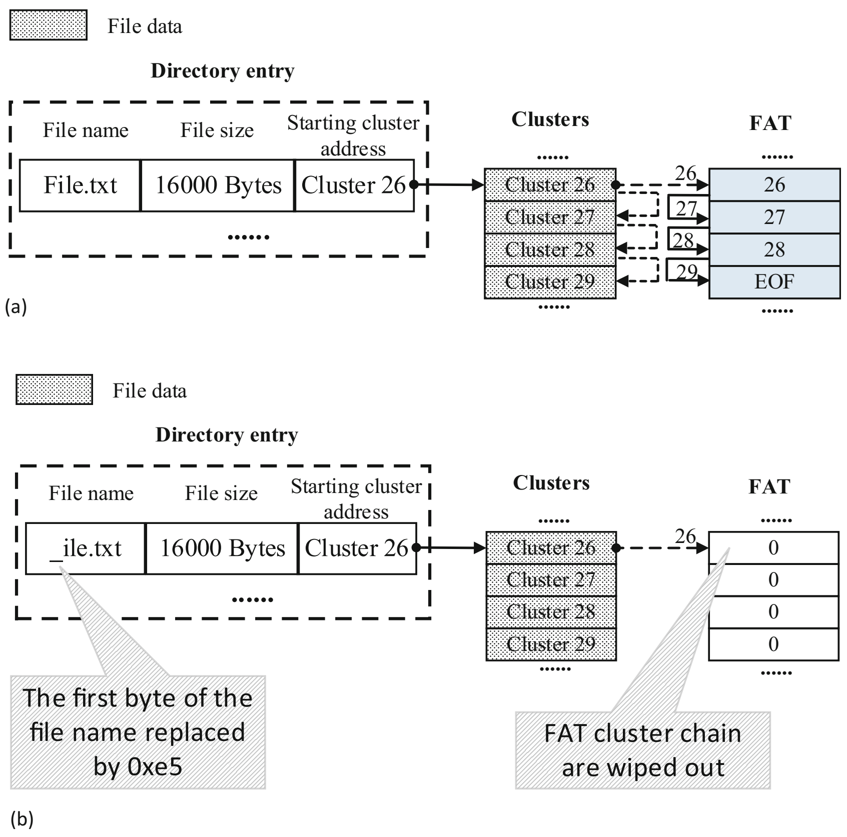


**Fig. 6.4** Illustration of file deletion in FAT file systems. (**a**) Relationship between the directory entry structures, clusters, and FAT structure before file deleted. (**b**) Relationship between the directory entry structures, clusters, and FAT structure after file deleted

Now, if you want to recover this deleted file, then you need to do a number of things. Suppose that we know the cluster size.

Firstly, search the DIR which contains the deleted file, and according to the remaining file name, locate the directory entry, which represents the deleted file.

Secondly, locate the file name in the directory entry and alter that first character from 0xE5 to its original one or any legal value.

Thirdly, obtain the file size and the address of the first cluster which has been allocated to the file by parsing out the directory entry. Then, determine the number of clusters allocated to the file based on the file size and the cluster size. In the example shown in Fig. 6.4a, the file size is 16,000 Bytes. Suppose that the cluster size is 4 KB. Then, we have

$$\begin{aligned}\text{No of clusters} &= \text{ceil}(\text{ the file size/the cluster size }) = \text{ceil}(16000\text{ Bytes}/4\text{ KB})\\ &= \text{ceil}(3.91) = 4\end{aligned}$$

where ceil(.) is the ceiling function. Since the starting cluster address is 26, we now know that clusters 26,27,28,29 are allocated to the deleted file. For cued recall, we assume that files are stored continuously.

Finally, the linked-list of used clusters for that file needs to be chained back together into the FAT table, starting from the first cluster defined in the directory entry representing the file and progressing in a linear fashion until the last cluster with the corresponding FAT entry filled with end-of-clusterchain marker, i.e., 0xfffffff in FAT32. Based on the addresses of the clusters used by the file, we can locate these corresponding FAT entries, i.e., FAT entries 26, 27, 28, 29, which need to be updated. Basically, every FAT entry must be inserted with a value which is the address of the next cluster allocated to the file, where the last FAT entry will be updated with a special flag or end-of-clusterchain marker, indicating that it is the last cluster for the file or the end of the file.

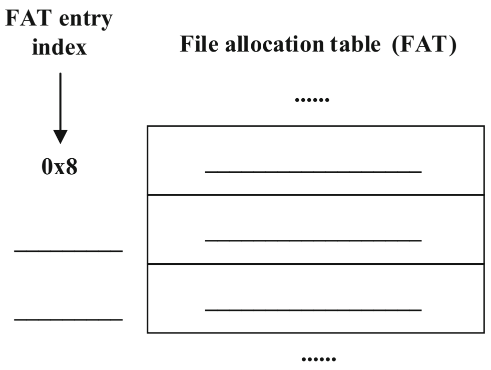
**Review Questions**

1. After a file is deleted in FAT file system, the first character of the file name is replaced with a special character with hexadecimal code __________.
2. In a FAT32 file system, the exact size of each entry in the File Allocation Table is __________ bytes.
3. In FAT file systems, the size of a directory entry is ___________ bytes.
4. Does FAT file system allow random access to a file? __________ (Yes or No)
5. The first cluster in a FAT file system is cluster ________________.
6. Assume that a file named file.txt is stored contiguously on disk and each cluster is 4 KB. Consider the following directory entry pointing to the file *file.txt*.

**Directory entry**

| File name | File size | Starting cluster address |
|---|---|---|
| FILE.TXT | 10270 Bytes | Cluster 8 |

Fill in all the blank FAT entries with proper values in the figure below, particularly the missing index numbers for the entries and their stored values. Assume a value of 0xfffffff in a FAT entry indicates the end of chain of clusters occupied by a file. (Note that in the real FAT file system, an entry may use multiple bytes, for example each entry uses 2 and 4 bytes (16 and 32 bits) for FAT16 and FAT32, respectively. The value of each entry can be stored in different ways, mainly big endian and little endian, depending on which computing system you use, Big-Endian machine or Little-Endian machine; we are simply using the actual value for simplicity in this question.)



## 6.4 Practice Exercise

The objective of this exercise is to recover a deleted file based on residual file system metadata remaining after deletion in FAT file systems.

### *6.4.1 Setting Up the Exercise Environment*

For this exercise, you will use a disk image named "thumbimage_fat.dd" provided in the book and will need to upload this disk image to Forensics Workstation you have built up in Chap. 3. Also, you need to extract a partition (or the only partition) from the disk image, and the partition has been formatted with an FAT file system.

### *6.4.2 Exercises*

**Part A: File System Layer Analysis**

Now analyze the file system image by answering the following questions using the *fsstat* command in TSK:

**Q1.** What version of FAT is it? (FAT12, FAT16, or FAT32).

**Q2.** Where is the File Allocation Table (FAT) 0 located? (Hint: You would need to figure out the start position and the size (or end position) of the FAT area.)

**Q3.** Where is the root directory located (sector address, and cluster address)? (Hint: You would need to figure out the start position and the size (or end position) of the root directory.)

**Q4.** What is the cluster size (in bytes)?

**Q5.** Where is the data area (in sectors and clusters)?

**Part B: Mounting and Unmounting File Systems**
Write down the command(s) used for the exercises below:

**Q6.** Change into the */mnt* directory, and make a directory called "forensics".

..........................................................................................................................

..........................................................................................................................

**Q7.** Mount the extracted partition into */mnt/forensics* with read-write access.

..........................................................................................................................

To mount a file system (either the name of the device file for the file system or a partition (file system) image) to a directory (mount point), making it available to the computer system, for example, a partition image hda.dd will be mounted as shown below:

```
# mount –o rw hda.dd < mount_point>
```

In the above example, the option "-o rw" indicates that the partition image hda.dd should be mounted with read-write access. Also, make sure that the directory where the file system will be mounted (the mount point) exists before you execute the *mount* command.

**Q8.** Change into the "/mnt/forensics" directory and then delete a file named readme.txt by using the rm command.

..........................................................................................................................

For a file system, any file operation may not be immediately applied to the physical file associated to it. Therefore, in order to make sure that the readme.txt is indeed deleted, you can use the command *sync* to flush file system buffers and force changes to disk in Linux.

**Q9.** Unmounting the extracted partition.

..........................................................................................................................

To manually unmount file systems, type:

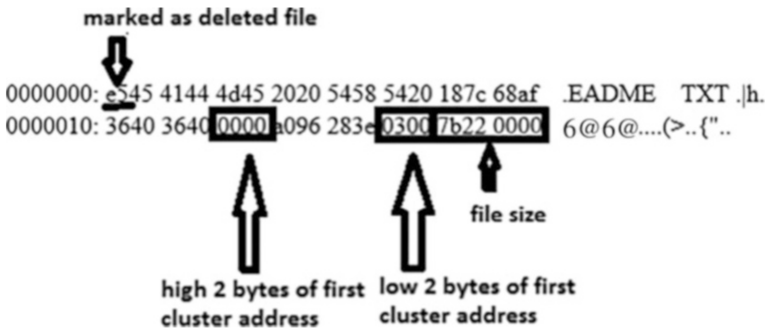
```
# umount < mount_point >
```

Before unmounting a disk partition, you have to ensure that the partition is not accessed. Otherwise, you will get a "device is busy" error.

**Part C: Recovering the Deleted File "Readme.txt"**

After the file "readme.txt" has been deleted, you can discover that its directory entry has been marked as deleted, particularly with its first character (byte 0 of the entry) replaced with 0xE5, by looking at the root directory.

- Sort through the root directory and locate the entry pointing to the deleted "readme.txt" file, like the one below:



```
0000000: e545 4144 4d45 2020 5458 5420 187c 68af  .EADME   TXT .|h.
0000010: 3640 3640 0000 a096 283e 0300 7b22 0000  6@6@....(>..{"..
```

marked as deleted file
file size
high 2 bytes of first cluster address
low 2 bytes of first cluster address

- Referring to Table 5.2 in Chap. 5, parse the root directory entry and obtain the following information, including first cluster address and file size. Since both are multiple byte values, the little endian conversion applies here since we are using a Little-Endian machine.

  **Q10**. Address of the first cluster: ________________, **Q11.** file size: ________________________

- Replace the first character of the root directory entry, 0xE5, with its original one "R" or any legal value. Note that efficient editing of large data files could work in a way like the following, especially there are only few places which need to be altered: First, extract these data areas to be modified; then, make changes to these extracted data and replace the ones in the original data file with the revised version. For more how-to details, please refer to the **Helpful tips** section.
- Determine how many clusters are allocated to the file and what are they?

  **Q12**. Clusters: ____________________________

- Based on the addresses of the clusters above, locate these corresponding FAT entries and insert proper values into them. Note that since we are using FAT32, each FAT entry uses four bytes. Therefore, the little endian conversion applies here too. We need to flip the order of bytes when inserting the value into the FAT entry. Fill in the following as many FAT entries as necessary with proper values, particularly the missing index numbers for the entries and their stored values in right byte order (Fig. 6.5).

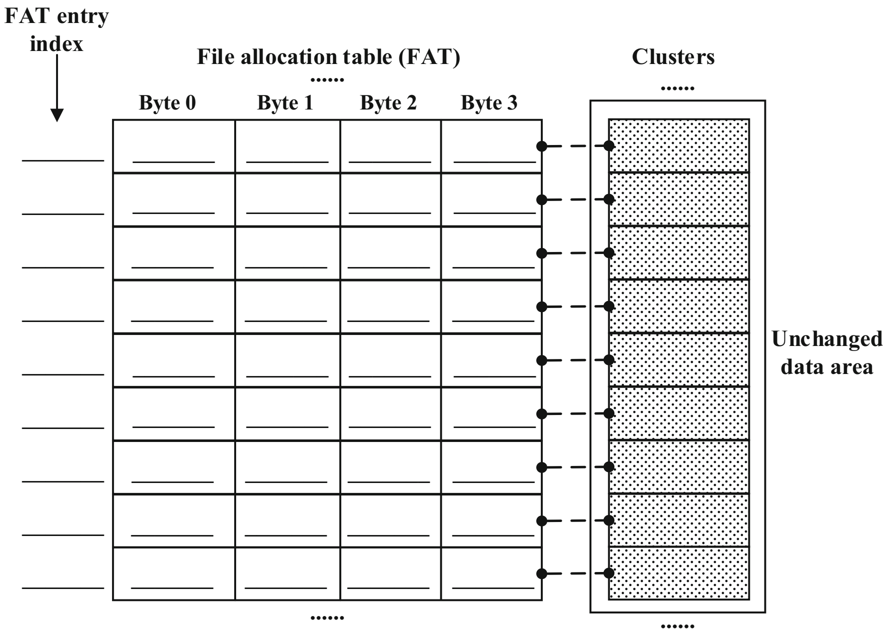


**Fig. 6.5** Restore the cluster chain in the File Allocation Table while recovering the deleted file "readme.txt"

Once you have successfully completed this lab, you can verify the successful recovery of the deleted file by mounting the modified partition image and seeing whether or not you can view the "readme.txt" properly.

## 6.5 Helpful Tips

(a) Endian order

Depending on which computing system you use, you will have to consider the byte order in which multibyte numbers are stored, particularly when you are writing those numbers to a file. The two orders are called "Little Endian" and "Big Endian".

When you read or write multiple byte data from/to a binary data file, the big or little endian conversion applies depending on what kind of byte order is used by the computing machine, Big-Endian machine or Little-Endian machine.

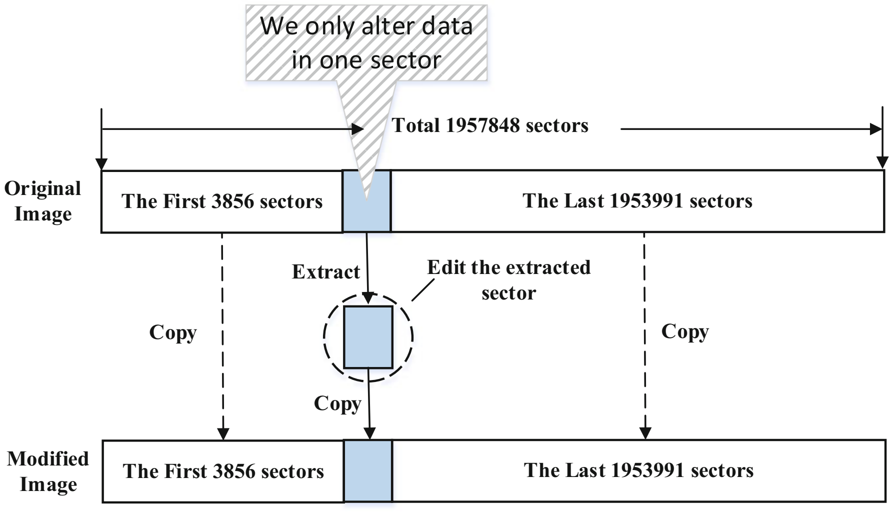


**Fig. 6.6** The best practices of editing large image files

(b) A practical approach to recover the deleted readme.txt file

In order to recover a deleted file in FAT file system, we have to modify several areas.

However, in practice, we need to work on a disk image, which could be very large. It is hard to edit the whole image file. In reality, we only need to edit a small disk area. Therefore, a good approach is to locate the area which we will work on, and then extract and save it into a small image file. Afterwards, we can edit the small image file and make any necessary changes for file recovery. Once all changes are complete, we can create a new disk partition image by integrating two images, the original image and the modified small image, as shown in Fig. 6.6.

Suppose you want to work on FAT table 0 to restore the cluster chain of the readme.txt file.

You can extract the first sector of FAT 0 by [FAT 0: Sectors 6316–7253]

```
dcfldd if = fatimage.dd bs = 512 skip = 6316 count = 1 of = fat0.dd
```

where “fatimage.dd” is the extracted file system image, and “fat0.dd” is the file storing the extracted sector. Note that the size of “fat0.dd” is only one sector, which is very small comparing with the original file system image.

Then, we can use a hex editor (e.g., ghex) to edit “fat0.dd” file.

The following is the snippet of the first sector of FAT 0:

```
[root@localhost lab8]# xxd fat0.dd
0000000: f8ff ff0f ffff ffff ffff ff0f 0000 0000 ................
0000010: 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000020: 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000030: 0000 0000 0000 0000 0000 0000 0000  0000 ................
0000040: 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000050: 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000060: 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000070: 0000 0000 0000 0000 000 0 0000 0000 0000 ................
0000080: 0000 0000 0000 0000 0000 0000 0000 0000 ................
......
```

Now, we have to put back the cluster chain, which is wiped out during the file deletion. Since the readme.txt file has 8827 bytes and the cluster size is 1024 bytes, no of clusters allocated to "readme.txt" is

$$\mathrm{ceil}(8827/1024) = 9.$$

Further, it is worth noting that in this exercise, we only consider a scenario where a file is stored in a hard disk contiguously and without fragmentation. In order words, the list of used clusters will progress in a linear fashion and clusters 3, 4, 5, 6, 7, 8, 9, 10 and 11 (total 9 clusters) are occupied by "readme.txt". Therefore, the cluster chain looks like the following (remember that each FAT entry has 32 bits or 4 bytes in a FAT32).

FAT entry 3 contains "0x00000004", which means the next occupied cluster is cluster 4

Similarly, we have

FAT entry 4 contains "0x00000005", which means the next occupied cluster is cluster 5
FAT entry 5 contains "0x00000006", which means the next occupied cluster is cluster 6
FAT entry 6 contains "0x00000007", which means the next occupied cluster is cluster 7
FAT entry 7 contains "0x00000008", which means the next occupied cluster is cluster 8
FAT entry 8 contains "0x00000009", which means the next occupied cluster is cluster 9
FAT entry 9 contains "0x0000000a", which means the next occupied cluster is cluster 10
FAT entry 10 contains "0x0000000b", which means the next occupied cluster is cluster 11

until FAT entry 11, where cluster 11 is the last cluster allocated to readme.txt. Therefore, FAT entry 11 contains a special flag of the end of file, “0x0fffffff”, which means cluster 11 is the last cluster allocated to “readme.txt”.

As a result, you should see the following:

```
[root@localhost softwares]# xxd fat0.dd
0000000: f8ff ff0f ffff ffff ffff ff0f 0400 0000  ................
0000010: 0500 0000 0600 0000 0700 0000 0800 0000  ................
0000020: 0900 0000 0a00 0000 0b00 0000 ffff ff0f  ................
0000030: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000040: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000050: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000060: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000070: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000080: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

Next, suppose you want to work on root directory to change back the first character of the file name.

You can extract the first sector of root directory by [Root Directory: 8192–8193]

```
dcfldd if = fatimage.dd bs = 512 skip = 8192 count = 1 of = rootdir.dd
```

where “fatimage.dd” is the extracted file system image.

Then, we can use ghex to edit rootdir.dd file, which is very small.

The following is the snippet of the first sector of root directory:

```
[root@localhost softwares]# xxd rootdir.dd
0000000: e545 4144 4d45 2020 5458 5420 187c 68af  .EADME  TXT .|h.
0000010: 3640 3640 0000 a096 283e 0300 7b22 0000  6@6@....(>..{"..
0000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000030: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000040: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000050: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000060: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000070: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000080: 0000 0000 0000 0000 0000 0000 0000 0000  ................
......
```

Change “0xe5” located at byte offset 00 to “0x52”, which is “R”.

Finally, now it is time to resemble the images we've already made changes on. Note that there are total 248,223 sectors (Total Range: 0–248,222) in this FAT file system.

```
dcfldd if = fatimage.dd bs = 512 skip = 0 count = 6316 of = recover.dd
dcfldd if = fat0.dd bs = 512 skip = 0 count = 1 > > recover.dd
dcfldd if = fatimage.dd bs = 512 skip = 6317 count = 1875 > > recover.dd
dcfldd if = rootdir.dd bs = 512 skip = 0 count = 1 > > recover.dd
dcfldd if = fatimage.dd bs = 512 skip = 8193 count = 240,030 > > recover.dd
```

where "fatimage.dd" is the original file system image after file deletion and recover. dd is the resulted image after you successfully recover the deleted "readme.txt" file.

## References

1. X. Lin, C. Zhang, T. Dule, On Achieving Encrypted File Recovery. Forensics in Telecommunications, Information, and Multimedia (e-Forensics), 2010
2. File Allocation Table. http://en.wikipedia.org/wiki/File_Allocation_Table
3. FAT. http://www.win.tue.nl/~aeb/linux/fs/fat/fat-1.html