Undeletion

Kavan Patel(121023), Snehal Panchal (121055), Somi Jha (121057)

30 April 2015

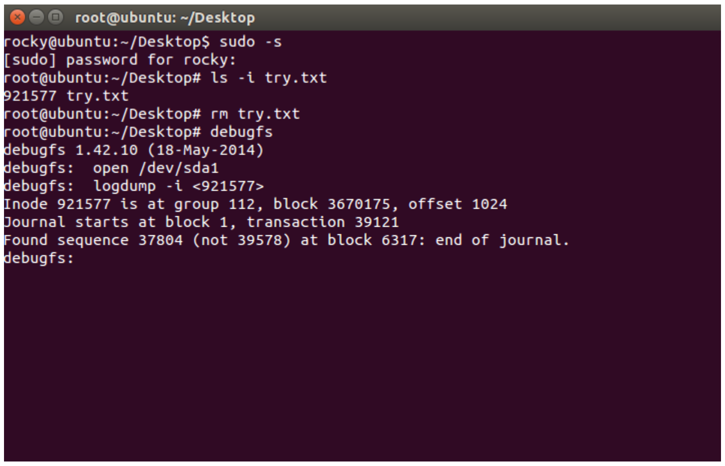
1 Introduction

Many a times it happens that while working on our computers, we accidentally delete some data or files which are important to us. The recovery of that deleted file can be done through **Undeletion**.
Undeletion is an unique feature for retrieving files of our computer system which have been removed from a file system by file deletion. But after deletion, a file generally reside in the hard drive and that is the reason that we can recover those files. Although undeletion can protect users from losing data but it can also pose some security risks since intentionally deleted data also remains in the drive and it can be undeleted. Now as we are aware that files that are deleted from our PC, do not get deleted permanently from our hard drive. But why is it so?

It is so because, when a file is deleted only the pointer to that particular file is deleted not the data which makes it difficult for us to locate the position of the file. The data remains intact in its original position that it existed before. We can take advantage of the fact that the data still exists on the hard drive, while the pointers have been deleted. When we search for a particular file that has been deleted, we will get some pointers of the files that have been deleted recently. We can then use these pointers to locate the exact location of the deleted file. Once if we are able to locate the deleted data, then it would be easy for us to recover the data using the information that is available and after executing undeletion successfully we can access the undeleted file like before. But if for a long time the operating system couldn't locate the file or if we restart or reboot our machine, the operating system marks the disk space as reusable and empty. This disk space could get overwritten, when new data is saved on to the hard drive, thereby causing permanent loss of data.

2 How Deletion works?

As we create, delete or modify data on the system, there are different operations done by the operating system, on the file system, whenever a file is deleted, modified or created.



```
rocky@ubuntu:~/Desktop$ sudo -s
[sudo] password for rocky:
root@ubuntu:~/Desktop# ls -i try.txt
921577 try.txt
root@ubuntu:~/Desktop# rm try.txt
root@ubuntu:~/Desktop# debugfs
debugfs 1.42.10 (18-May-2014)
debugfs:  open /dev/sda1
debugfs:  logdump -i <921577>
Inode 921577 is at group 112, block 3670175, offset 1024
Journal starts at block 1, transaction 39121
Found sequence 37804 (not 39578) at block 6317: end of journal.
debugfs:
```

Figure 1: Before Deletion
http://www.slashroot.in/how-does-file-deletion-work-linux

In the below diagram it the inode structure of the file music.mp3 is shown and how data block address are linked with the help of direct block pointers and indirect block pointers. Direct Block pointers point to the blocks where the data is located. Whereas indirect block pointers will point to the blocks which will in turn point to the blocks where data is located. Direct block pointers are inside the inode block shown above and stores the block address of the data in it. But indirect,double indirect and triple indirect block pointers are just pointers pointing to the blocks which in turn point to data blocks, Which is evident from the above figure.

Now if we delete that music.mp3 file, all the pointers which were heading us towards the the data are lost and the blocks will have 0 value but the data remains intact.

3 How Undeletion works?

Generally, after deletion the filesystem marks the space that was earlier holded by the deleted file as empty. With these filesystems, the undelete program looks at the file allocation table and copies the deleted file's data to another storage unit, but then the user could also lose other deleted files that are needed to them.

When files are deleted from a temporary storage location, undeleting the data becomes more difficult, or even impossible. In a File Allocation Table (FAT) file system, whenever a file is deleted, the file system keeps the directory entry with all the information, including physical location, name, and length. This directory entry is used by file allocation table to undelete a file.
Though data recovery seems possible through the programs we create or the

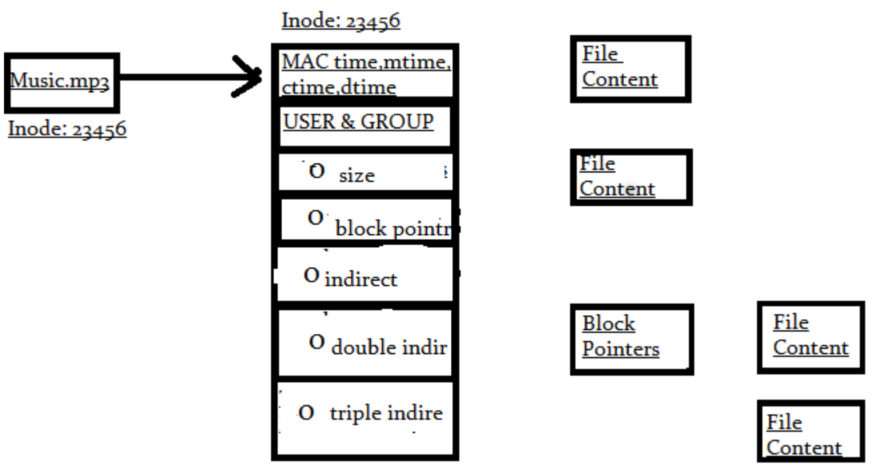


Figure 2: After Deletion
http://www.slashroot.in/how-does-file-deletion-work-linux

softwares we have, it may not be possible to recover deleted files unless the following criterias are met:

1. The deleted file entry exists in the folder or directory and has not been overwritten by another file.
2. Data storage drive sectors have not been overwritten by other files.
3. Files are not fragmented.

In Linux filesystem an inode consists of all information about file including filesize, timestamp, memory address etc. When a file is deleted the link to inode is removed but the file is not removed from memory. However, if a process is having a file open it will be possible to retrieve the deleted file.

If a process still has the file open, the data is available somewhere on the system, even though according to the directory listing the file already appears to be lost or deleted.To know where to go, we need to get the id of the process that has the file open, and the file descriptor.

4 Different Approaches

4.1 Debugfs with logdump

We tried this approach and we also retrieved the file, but the file we got was corrupted. Therefore we stopped following this approach. The steps are as follows:

# gedit try.txt
We created a file called try.txt

# ls -i try.txt
This command gives us the inode number of try.txt. An inode is a data structure used to represent file system in linux.

# rm try.txt
We are removing the file we created

# debugfs:
This command is to open debugfs file

# debugfs: open /dev/sda1
This will open device sda1. Every system has a particular filesystem and it opens on a particular device. In our case, we have ext4 filesystem and sda1 device.

# debugfs: logdump -i ¡inode number¿
This command will dump the file with this inode number and we will get some information about the file as an output. There we will get the block number on which the file is residing

# dd if=/dev/sda1 of=try.txt bs=4096 count=1 skip=block number

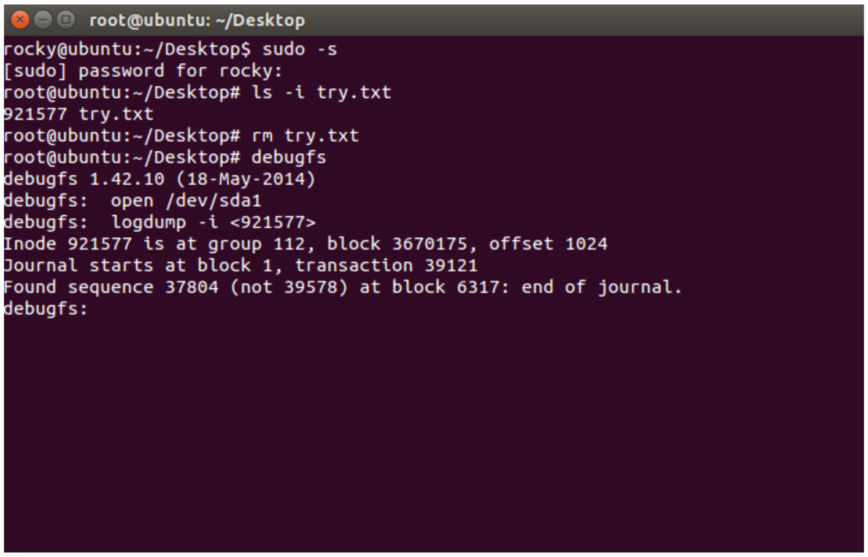
Here dd stands for device drivers. So this commands mean that if the input file is in following device driver then output file will be try.txt. That is the information will be dumped in this file. bs is the block size and count = 1 means that we need only one file and we are skipping all the block numbers and focusing on the block number we want.

Fig 3 shows above approach.
We also didn't go with this approach because we need to remember the inode number of the file, which is not possible for any user who accidentally deletes a file

4.2 Debugfs with stat

debugfs is a simple to use file specially designed for debugging purposes based on RAM. It exists as a simple way to make information available to user space. It is an interactive file system debugger and can be used to examine and change the state of an ext2,ext3 and ext4 file system.
Now we followed this approach as follows:

# gedit try.txt
We created a file called try.txt



```
root@ubuntu: ~/Desktop
rocky@ubuntu:~/Desktop$ sudo -s
[sudo] password for rocky:
root@ubuntu:~/Desktop# ls -i try.txt
921577 try.txt
root@ubuntu:~/Desktop# rm try.txt
root@ubuntu:~/Desktop# debugfs
debugfs 1.42.10 (18-May-2014)
debugfs:  open /dev/sda1
debugfs:  logdump -i <921577>
Inode 921577 is at group 112, block 3670175, offset 1024
Journal starts at block 1, transaction 39121
Found sequence 37804 (not 39578) at block 6317: end of journal.
debugfs:
```

Figure 3: Using debugfs : logump

# ls -i try.txt
This command gives us the inode number of try.txt. An inode is a data structure used to represent file system in linux.

# rm try.txt
We are removing the file we created

# debugfs:
This command is to open debugfs file

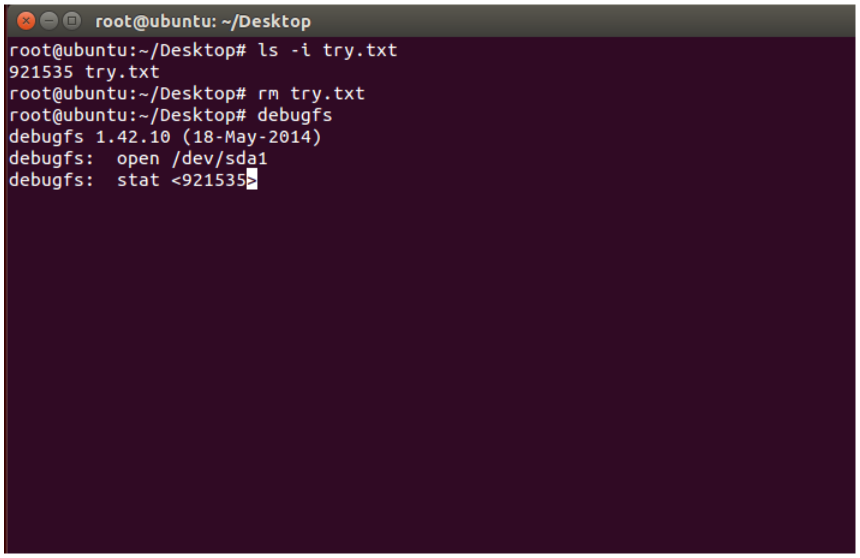
# debugfs: open /dev/sda1
This will open device sda1. Every system has a particular filesystem and it opens on a particular device. In our case, we have ext4 filesystem and sda1 device.

# debugfs: stat ¡inode number¿
stat displays file or file system status. Using stat we will get the block number.

So basically, we tried to get block number of deleted file using stat command (Fig 4). But we failed (Fig 5) as the output of command didn't showed block number.

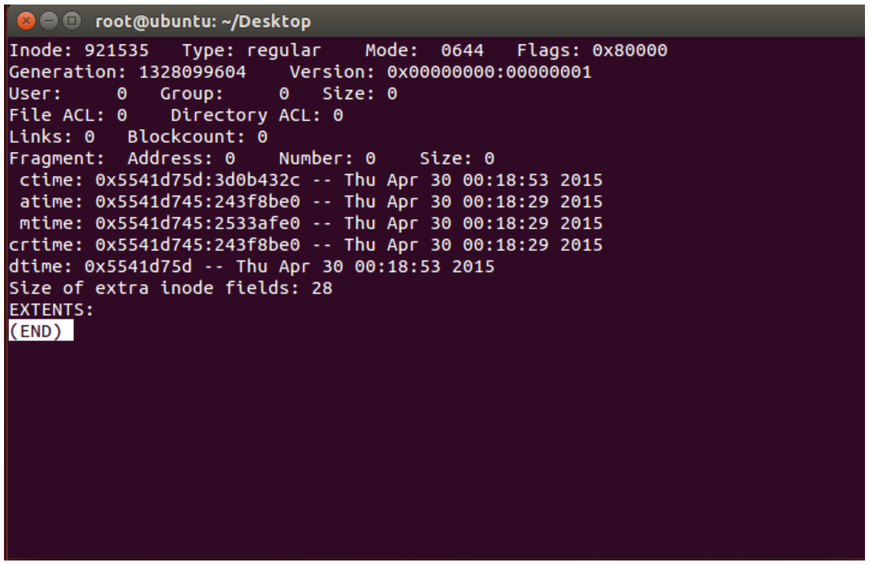
So we tried to get block number of target file before its deletion and it worked.

Well limitation of this approach is that we extract some file information be-



```
root@ubuntu:~/Desktop# ls -i try.txt
921535 try.txt
root@ubuntu:~/Desktop# rm try.txt
root@ubuntu:~/Desktop# debugfs
debugfs 1.42.10 (18-May-2014)
debugfs:  open /dev/sda1
debugfs:  stat <921535>
```

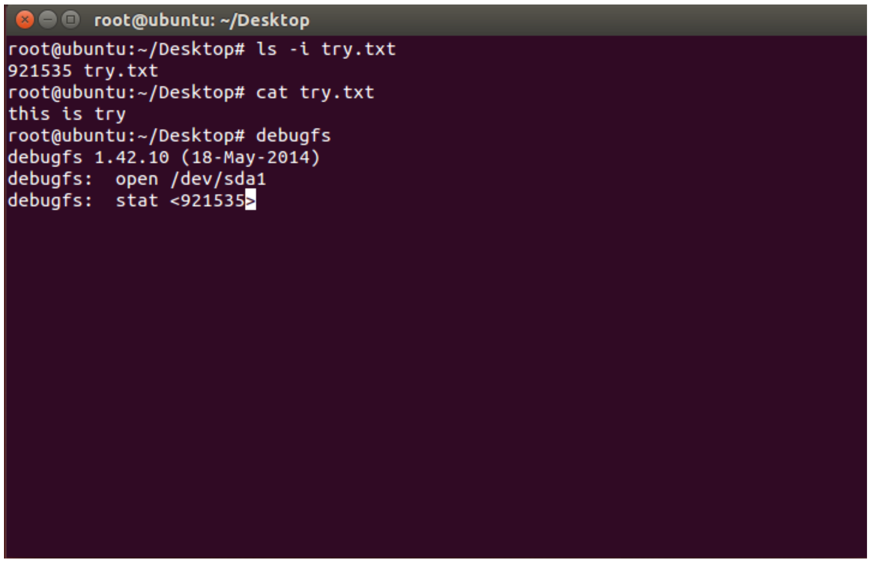
Figure 4: Using debugfs : stat



```
Inode: 921535   Type: regular    Mode:  0644   Flags: 0x80000
Generation: 1328099604    Version: 0x00000000:00000001
User:     0   Group:     0   Size: 0
File ACL: 0    Directory ACL: 0
Links: 0   Blockcount: 0
Fragment:  Address: 0    Number: 0    Size: 0
 ctime: 0x5541d75d:3d0b432c -- Thu Apr 30 00:18:53 2015
 atime: 0x5541d745:243f8be0 -- Thu Apr 30 00:18:29 2015
 mtime: 0x5541d745:2533afe0 -- Thu Apr 30 00:18:29 2015
crtime: 0x5541d745:243f8be0 -- Thu Apr 30 00:18:29 2015
dtime: 0x5541d75d -- Thu Apr 30 00:18:53 2015
Size of extra inode fields: 28
EXTENTS:
(END)
```

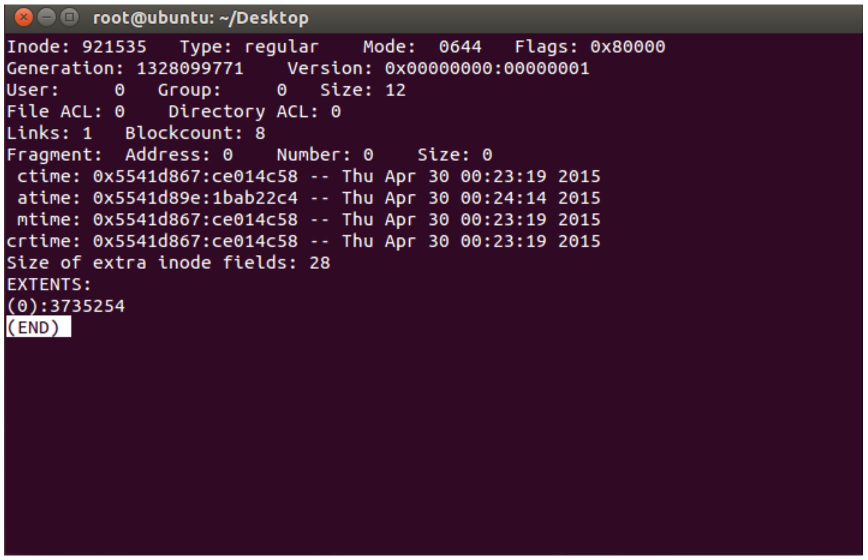
Figure 5: Output of debugfs:stat

fore its deletion, which might not be the real world case.



```
root@ubuntu:~/Desktop# ls -i try.txt
921535 try.txt
root@ubuntu:~/Desktop# cat try.txt
this is try
root@ubuntu:~/Desktop# debugfs
debugfs 1.42.10 (18-May-2014)
debugfs:  open /dev/sda1
debugfs:  stat <921535>
```

Figure 6: Using debugfs : stat

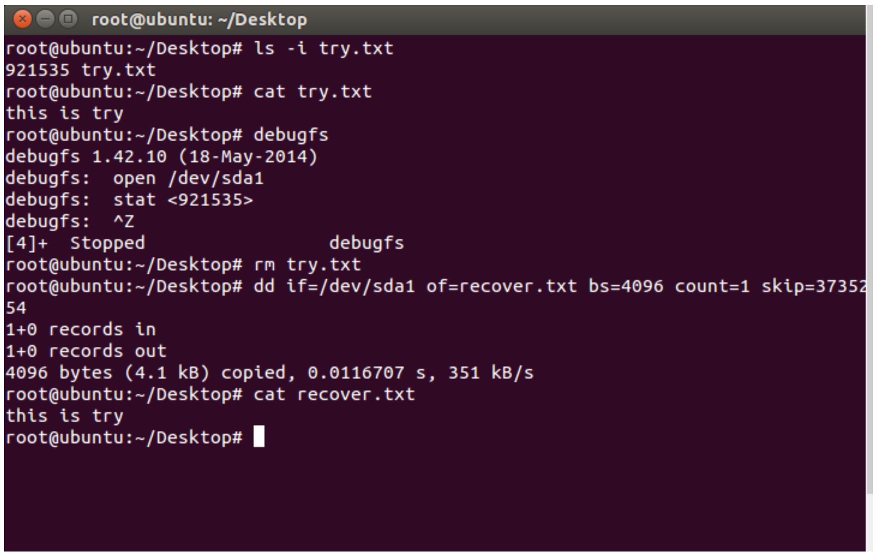


```
Inode: 921535   Type: regular    Mode:  0644   Flags: 0x80000
Generation: 1328099771    Version: 0x00000000:00000001
User:     0   Group:     0   Size: 12
File ACL: 0    Directory ACL: 0
Links: 1   Blockcount: 8
Fragment:  Address: 0    Number: 0    Size: 0
 ctime: 0x5541d867:ce014c58 -- Thu Apr 30 00:23:19 2015
 atime: 0x5541d89e:1bab22c4 -- Thu Apr 30 00:24:14 2015
 mtime: 0x5541d867:ce014c58 -- Thu Apr 30 00:23:19 2015
crtime: 0x5541d867:ce014c58 -- Thu Apr 30 00:23:19 2015
Size of extra inode fields: 28
EXTENTS:
(0):3735254
(END)
```

Figure 7: Getting file information before deletion

4.3 Using \proc

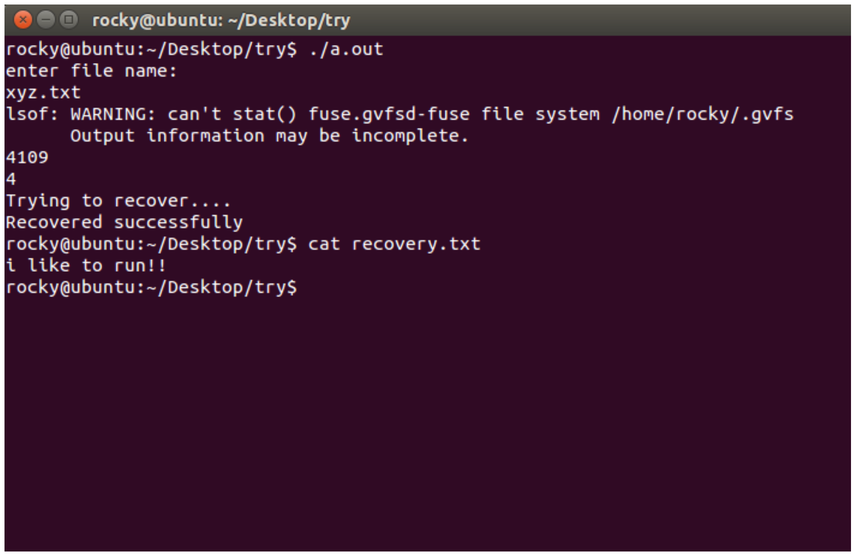
As stated earlier deleted files if open somewhere can be recovered by getting its process_id and file descriptor. Here after deletion of file we are searching for any running process which has that file open. Here if there is a process containing



```
root@ubuntu: ~/Desktop
root@ubuntu:~/Desktop# ls -i try.txt
921535 try.txt
root@ubuntu:~/Desktop# cat try.txt
this is try
root@ubuntu:~/Desktop# debugfs
debugfs 1.42.10 (18-May-2014)
debugfs:  open /dev/sda1
debugfs:  stat <921535>
debugfs:  ^Z
[4]+  Stopped                 debugfs
root@ubuntu:~/Desktop# rm try.txt
root@ubuntu:~/Desktop# dd if=/dev/sda1 of=recover.txt bs=4096 count=1 skip=37352
54
1+0 records in
1+0 records out
4096 bytes (4.1 kB) copied, 0.0116707 s, 351 kB/s
root@ubuntu:~/Desktop# cat recover.txt
this is try
root@ubuntu:~/Desktop# 
```

Figure 8: Retrieved file

file we will get its process_id and file descriptor as well. And it will be having a link to its inode number. Hence with this approach we can recover a delete file under condition that file is open in some or the other process.



```
rocky@ubuntu: ~/Desktop/try
rocky@ubuntu:~/Desktop/try$ ./a.out
enter file name:
xyz.txt
lsof: WARNING: can't stat() fuse.gvfsd-fuse file system /home/rocky/.gvfs
      Output information may be incomplete.
4109
4
Trying to recover....
Recovered successfully
rocky@ubuntu:~/Desktop/try$ cat recovery.txt
i like to run!!
rocky@ubuntu:~/Desktop/try$
```

Figure 9: Retrieved file by C code

5 User Manual

We finally followed the last approach and we are providing here with the user manual of the third approach:

```
# gedit abc.txt
```
Create a random file and write something so that after recovery we can check whether we have undeleted the correct file or not

```
# rm abc.txt
```
Remove or delete the file you created

```
# less abc.txt
```
Like cat less also displays the file information but unlike cat, it holds the file and if we want to exit we can by using q.

```
# open new terminal
```

```
# cc.undelete.c
```
This is the C code that we have written. So, to undelete your stuff you first need to compile this code.

```
# ./a.out
```
Run the code. You'll get a message that the file has been recovered.

6 Limitation

We can successfully retrieve the file we deleted, but there are some limitations too:
1. The user has to remember the exact file name which they have deleted accidentally.
2. There should be a process having target file open within it, in order to retrieve.
3. If the system is rebooted or restarted then we can't recover the deleted file.

7 Future Ideas

We can try this approach and putting them in a shell script and then calling that shell script through a system call. Then user just have to write undelete ”file$_{n}ame$”$torecoverdeletedfile.$

References

http://www.techopedia.com/definition/9831/undelete
http://www.file-recovery-software.org/how-does-undelete-software-work.html
http://www.slashroot.in/how-does-file-deletion-work-linux
http://www.linux.org/threads/undelete-files-on-linux-systems.4316/
http://www.cyberciti.biz/tips/linux-ext3-ext4-deleted-files-recovery-howto.html