



浅谈Inode

No zuo no die why you try

By : BIGBALLON

目录

- Inode 是什么

- Inode 的内容

- Inode 的大小

- Inode 的号码

- 目录文件

- 硬链接和软链接

Inode 是什么

- 理解inode，要从文件储存说起。
- 文件储存在硬盘上，硬盘的最小存储单位叫做“扇区”（Sector），每个扇区储存512字节，相当于0.5KB。
- 操作系统读取硬盘的时候，不会一个个扇区地读取，这样效率太低，而是一次性连续读取多个扇区，即一次性读取一个“块”（block），这种由多个扇区组成的“块”，是文件存取的最小单位，“块”的大小，最常见的是4KB，即连续八个sector组成一个block。
- 文件数据都储存在“块”中，那么很显然，我们还必须找到一个地方储存文件的元信息，比如文件的创建者，文件的创建日期，文件的大小等等，这种储存文件元信息的区域就叫做inode，中文译名为“索引节点”。
- 每一个文件都有对应的inode，里面包含了与该文件有关的一些信息。

Inode 的内容

□ inode包含文件的元信息。具体来说有以下内容：

文件的字节数

文件拥有者的 User ID

文件的 Group ID

文件数据block的位置

文件的读、写、执行权限

链接数。即有多少文件名指向这个inode

文件的时间戳。共有三个：

ctime 指 inode上一次变动的时间。

mtime指 文件内容上一次变动的时间。

atime 指 文件上一次打开的时间。

□ 可以用 **stat** 命令。查看某个文件的inode信息：

```
bigballon@ubuntu:~/Desktop$ stat Hdu1812\ Count\ the\ Tetris\ (polya定理)\.java
File: 'Hdu1812 Count the tetris(polya定理).java'
  Size: 819          Blocks: 8          IO Block: 4096   regular file
Device: 801h/2049d  Inode: 933074     Links: 1
Access: (0766/-rwxrw-rw-)  Uid: ( 1000/bigballon)   Gid: ( 1000/bigballon)
Access: 2015-04-08 05:32:46.989281342 -0700
Modify: 2015-04-02 08:37:46.314262000 -0700
Change: 2015-04-08 05:32:45.973281323 -0700
 Birth: -
```

Inode 的大小

- inode 也会消耗硬盘空间。
- 硬盘格式化时，操作系统自动将硬盘分成两个区域。
 - 一个是 数据区 存放文件数据；
 - 一个是 inode 区 `inode table` 存放 inode 所包含的信息。
- 每个 inode 节点的大小，一般是 128 字节或 256 字节。
inode 节点的总数，在格式化时就给定，一般是每 1KB 或每 2KB 就设置一个 inode。
- 假定在一块 10GB 的硬盘中，每个 inode 节点的大小为 128 字节，每 1KB 就设置一个 inode，那么 inode table 的大小就会达到 128MB，占整块硬盘的 1.28%。

Inode 的大小

□ 使用 **df** 命令 查看每个硬盘分区的inode总数和已经使用的数量。

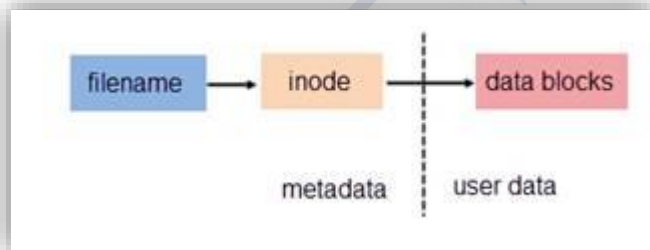
```
bigballon@ubuntu:~/Desktop$ df -i
Filesystem      Inodes    IUsed   IFree  IUse% Mounted on
/dev/sda1       1245184  241854 1003330  20% /
none            126339    2      126337   1% /sys/fs/cgroup
udev            123641    462    123179   1% /dev
tmpfs           126339    466    125873   1% /run
none            126339    5      126334   1% /run/lock
none            126339    6      126333   1% /run/shm
none            126339    25     126314   1% /run/user
```

□ 使用 **sudo dumpe2fs -h /dev/hda | grep "Inode size"** 命令 查看每个inode节点的大小

```
bigballon@ubuntu:~/Desktop$ sudo dumpe2fs -h /dev/sda1 | grep "Inode size"
dumpe2fs 1.42.9 (4-Feb-2014)
Inode size:                256
bigballon@ubuntu:~/Desktop$
```

□ 由于每个文件都必须有一个inode，因此有可能发生inode已经用光，但是硬盘还未存满的情况，这时就无法在硬盘上创建新文件。

Inode 的结构



Inode-index

Inode-no	File name
1	F1
2	D1
⋮	⋮
n	Fn

← Directory

文件系统如何存取文件？

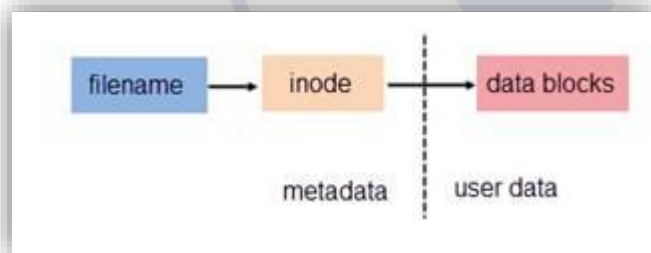
- 1、根据文件名，通过Directory的对应关系，找到文件对应的Inode number
- 2、再根据Inode number读取到文件的Inode table
- 3、再根据Inode table中的Pointer读取到相应的Blocks

Inode-table

Inode-no	File type	permission	Link count	UID	GID	size	Time stamp	pointer
1	-	644	1	500	500				
2	d	755	1	0	0				
⋮	⋮	⋮	⋮	⋮	⋮				
n	-	644	2	501	501				

Inode 的号码

- 每个inode都有一个号码，操作系统用inode号码来识别不同的文件。
- Unix/Linux系统内部不使用文件名，而使用inode号码来识别文件。对于系统来说，文件名只是inode号码便于识别的别称或者绰号。
- 表面上，用户通过文件名，打开文件。实际上，系统内部这个过程分成三步：
 - 首先，系统找到这个文件名对应的inode号码；
 - 其次，通过inode号码获取inode信息；
 - 最后，根据inode信息找到文件数据所在的block 读出数据。



□ Inode 远比文件名重要

目录文件

- Unix/Linux系统中，目录(directory)也是一种文件。打开目录，实际上就是打开目录文件。
- 目录文件的结构非常简单，就是一系列目录项(dirent)的列表。
- 每个目录项，由两部分组成：
所包含文件的文件名，
该文件名对应的inode号码。

□ 使用 ls 命令

ls 命令只列出目录文件中的所有文件名

ls -i 命令列出整个目录文件，即文件名和inode号码：

ls -l 命令列出文件的详细信息

```
bigballon@ubuntu:~/Desktop$ ls
acm  Hdu1812 Count the Tetris(polya定理).java
code justdoit.txt
bigballon@ubuntu:~/Desktop$
```

```
bigballon@ubuntu:~/Desktop$ ls -i
```

```
528256 acm
528255 code
933074 Hdu1812 Count the Tetris(p
948618 justdoit.txt
bigballon@ubuntu:~/Desktop$
```

```
bigballon@ubuntu:~/Desktop$ ls -l
```

```
total 12
drwxrwxr-x 2 bigballon bigballon 4096 Apr  8 07:58 acm
drwxrwxr-x 2 bigballon bigballon 4096 Apr  8 07:57 code
-rwxrw-rw- 1 bigballon bigballon  819 Apr  2 08:37 Hdu1812 Count
-rw-rw-r-- 1 bigballon bigballon    0 Apr  8 07:58 justdoit.txt
```

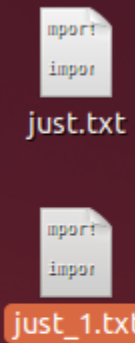
□ 一般情况下，文件名和 inode 号码是“一一对应”关系。每个 inode 号码对应一个文件名。

但是，Unix/Linux 系统允许多个文件名指向同一个 inode 号码。

这意味着，可以用不同的文件名访问同样的内容；对文件内容进行修改，会影响到所有文件名；但是，删除一个文件名，不影响另一个文件名的访问。这种情况就被称为“硬链接” (hard link)。

硬链接

```
bigballon@ubuntu: ~/Desktop
bigballon@ubuntu:~/Desktop$ ls
acm code Hdu1812 Count the Tetris(聚ya定理).java just.txt
bigballon@ubuntu:~/Desktop$ ln just.txt just_1.txt
bigballon@ubuntu:~/Desktop$ ls -i
528256 acm
528255 code
933074 Hdu1812 Count the Tetris(聚ya定理).java
bigballon@ubuntu:~/Desktop$ ls -l
total 20
drwxrwxr-x 2 bigballon bigballon 4096 Apr  8 07:58 acm
drwxrwxr-x 2 bigballon bigballon 4096 Apr  8 07:57 code
-rwxrw-rw- 1 bigballon bigballon  819 Apr  2 08:37 Hdu1812 Count the T
聚ya定理).java
-rw-rw-r-- 2 bigballon bigballon  819 Apr  8 08:25 just_1.txt
-rw-rw-r-- 2 bigballon bigballon  819 Apr  8 08:25 just.txt
bigballon@ubuntu:~/Desktop$
```



948603 just_1.txt
948603 just.txt

2 bigballon bigballon 819 Apr 8 08:25 just_1.txt
2 bigballon bigballon 819 Apr 8 08:25 just.txt

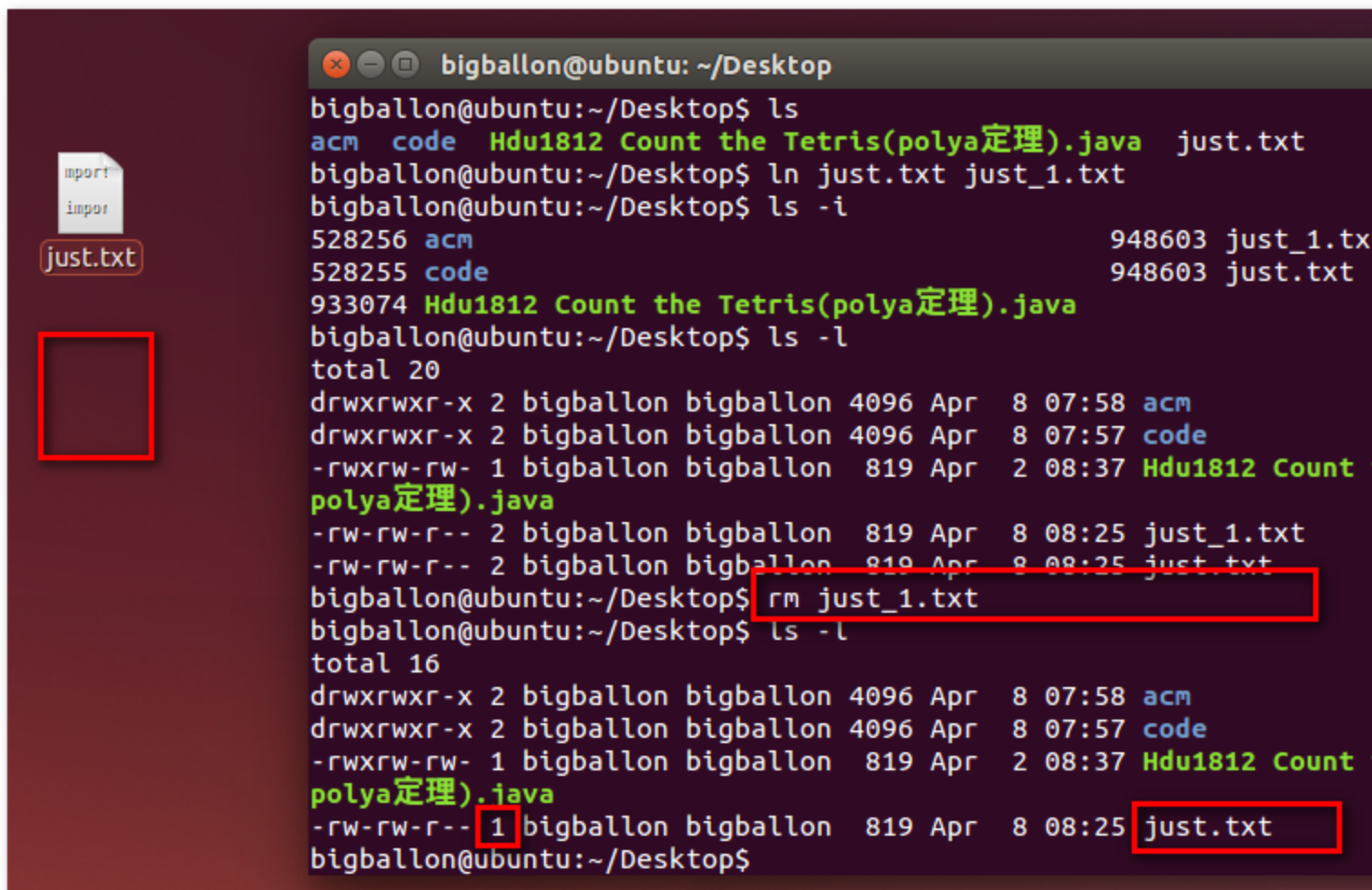
```
just.txt x
import java.util.*;
import java.math.*;
|
public class Main {

    public static void main(String[] args) {

        Scanner cin = new Scanner( System.in );
```

ln 命令创建硬链接

硬链接

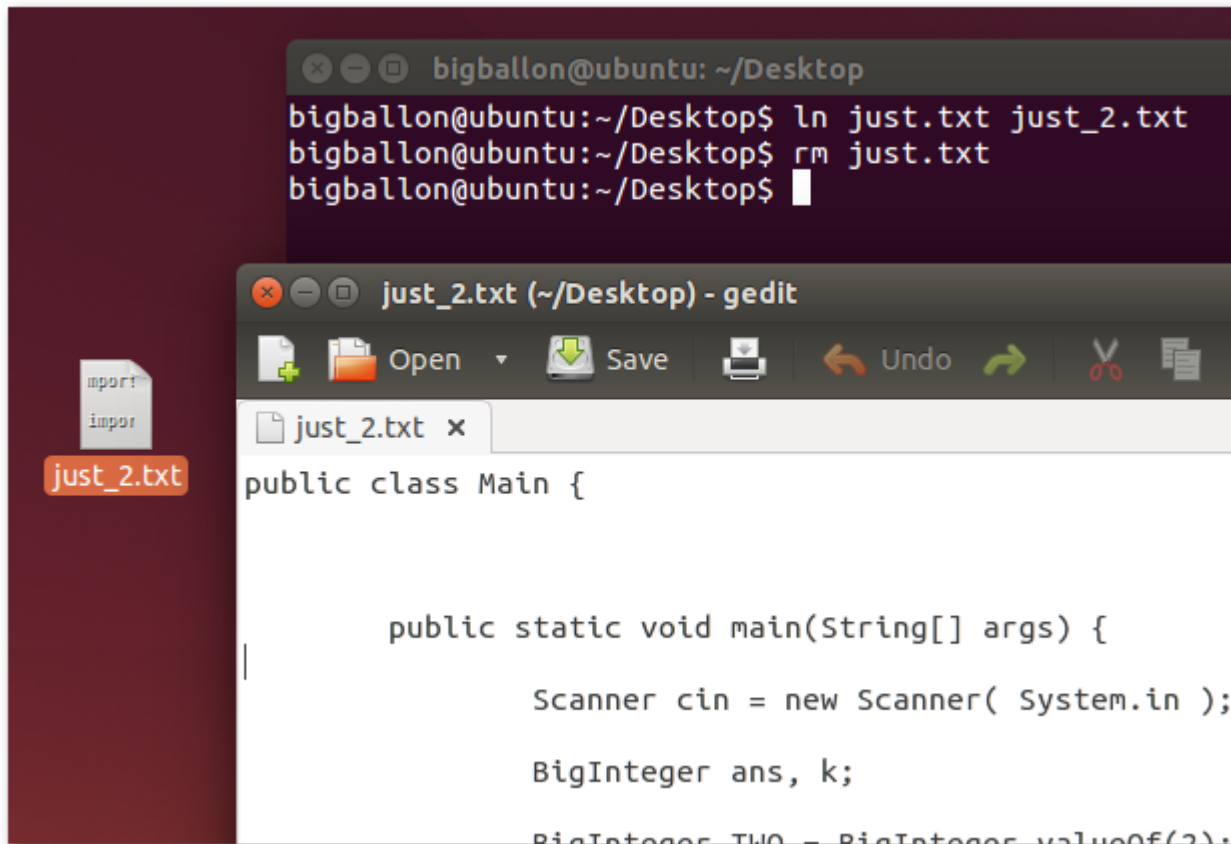


The image shows a terminal window on an Ubuntu system. The user is in the directory ~/Desktop. The terminal output shows the following sequence of commands and results:

```
bigballon@ubuntu: ~/Desktop
bigballon@ubuntu:~/Desktop$ ls
acm code Hdu1812 Count the Tetris(polya定理).java just.txt
bigballon@ubuntu:~/Desktop$ ln just.txt just_1.txt
bigballon@ubuntu:~/Desktop$ ls -li
528256 acm 948603 just_1.tx
528255 code 948603 just.txt
933074 Hdu1812 Count the Tetris(polya定理).java
bigballon@ubuntu:~/Desktop$ ls -l
total 20
drwxrwxr-x 2 bigballon bigballon 4096 Apr 8 07:58 acm
drwxrwxr-x 2 bigballon bigballon 4096 Apr 8 07:57 code
-rwxrw-rw- 1 bigballon bigballon 819 Apr 2 08:37 Hdu1812 Count
polya定理).java
-rw-rw-r-- 2 bigballon bigballon 819 Apr 8 08:25 just_1.txt
-rw-rw-r-- 2 bigballon bigballon 819 Apr 8 08:25 just.txt
bigballon@ubuntu:~/Desktop$ rm just_1.txt
bigballon@ubuntu:~/Desktop$ ls -l
total 16
drwxrwxr-x 2 bigballon bigballon 4096 Apr 8 07:58 acm
drwxrwxr-x 2 bigballon bigballon 4096 Apr 8 07:57 code
-rwxrw-rw- 1 bigballon bigballon 819 Apr 2 08:37 Hdu1812 Count
polya定理).java
-rw-rw-r-- 1 bigballon bigballon 819 Apr 8 08:25 just.txt
bigballon@ubuntu:~/Desktop$
```

Annotations in the image include a red box around the file icon for 'just.txt' on the left, a red box around the 'rm just_1.txt' command, and a red box around the '1' in the final 'ls -l' output, indicating that the hard link 'just_1.txt' has been removed and the original file 'just.txt' now has a link count of 1.

硬链接



The image shows a terminal window and a text editor window. The terminal window displays the following commands and output:

```
bigballon@ubuntu: ~/Desktop
bigballon@ubuntu:~/Desktop$ ln just.txt just_2.txt
bigballon@ubuntu:~/Desktop$ rm just.txt
bigballon@ubuntu:~/Desktop$
```

The text editor window, titled "just_2.txt (~/Desktop) - gedit", shows the following code:

```
public class Main {

    public static void main(String[] args) {

        Scanner cin = new Scanner( System.in );

        BigInteger ans, k;

        BigInteger TWO = BigInteger.valueOf(2);
```

硬链接

```
bigballon@ubuntu:~/Desktop$ ls -l
total 12
drwxrwxr-x 2 bigballon bigballon 4096 Apr  8 07:58 acm
drwxrwxr-x 2 bigballon bigballon 4096 Apr  8 07:57 code
-rwxrw-rw- 1 bigballon bigballon  819 Apr  2 08:37 Hdu1812 Count th
-rw-rw-r-- 1 bigballon bigballon    0 Apr  8 07:58 justdoit.txt
bigballon@ubuntu:~/Desktop$ ls -al
total 20
drwxr-xr-x  4 bigballon bigballon 4096 Apr  8 07:58 .
drwxr-xr-x 16 bigballon bigballon 4096 Apr  8 05:26 ..
drwxrwxr-x  2 bigballon bigballon 4096 Apr  8 07:58 acm
drwxrwxr-x  2 bigballon bigballon 4096 Apr  8 07:57 code
-rwxrw-rw-  1 bigballon bigballon  819 Apr  2 08:37 Hdu1812 Count t
-rw-rw-r--  1 bigballon bigballon    0 Apr  8 07:58 justdoit.txt
```

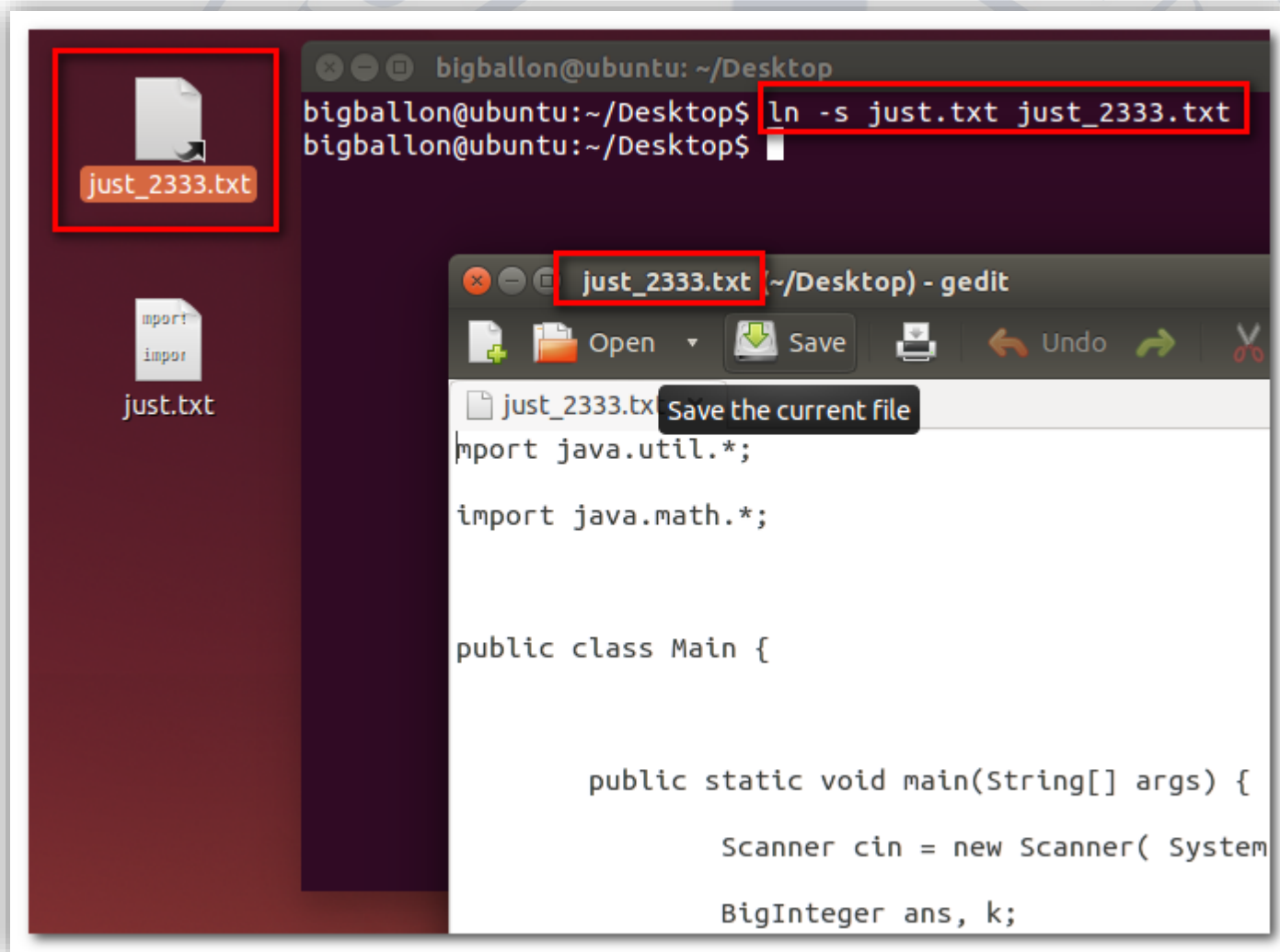
这里顺便说一下目录文件的“链接数”。创建目录时，默认会生成两个目录项“.”和“..”前者的inode号码就是当前目录的inode号码，等同于当前目录的“硬链接”；后者的inode号码就是当前目录的父目录的inode号码，等同于父目录的“硬链接”。所以，任何一个目录的“硬链接”总数，总是等于2加上它的子目录总数（含隐藏目录）。

□ 除了硬链接以外，还有一种特殊情况。

文件A和文件B的inode号码虽然不一样，但是文件A的内容是文件B的路径，读取文件A时，系统会自动将访问者导向文件B，因此，无论打开哪一个文件，最终读取的都是文件B。这时，文件A就称为文件B的“软链接”（soft link）或者“符号链接”（symbolic link）。

这意味着，文件A依赖于文件B而存在，如果删除了文件B，打开文件A就会报错：“No such file or directory”。这是软链接与硬链接最大的不同：文件A指向文件B的文件名，而不是文件B的inode号码，文件B的inode“链接数”不会因此发生变化。

ln -s 命令创建软链接



软链接

有点像... 快捷方式?

```
bigballon@ubuntu: ~/Desktop
bigballon@ubuntu:~/Desktop$ ln -s just.txt just_2333.txt
bigballon@ubuntu:~/Desktop$ rm just_2333.txt
bigballon@ubuntu:~/Desktop$
```

```
bigballon@ubuntu: ~/Desktop
bigballon@ubuntu:~/Desktop$ ln -s just.txt just_2333.txt
bigballon@ubuntu:~/Desktop$ rm just 2333.txt
bigballon@ubuntu:~/Desktop$ ln -s just.txt just_orz.txt
```

just_orz.txt (~/Desktop) - gedit

```
import java.util.*;
```

```
bigballon@ubuntu: ~/Desktop
bigballon@ubuntu:~/Desktop$ ln -s just.txt just_2333.txt
bigballon@ubuntu:~/Desktop$ rm just_2333.txt
bigballon@ubuntu:~/Desktop$ ln -s just.txt just_orz.txt
bigballon@ubuntu:~/Desktop$ rm just.txt
bigballon@ubuntu:~/Desktop$
```

just_orz.txt [Read-Only] (~/Desktop) - gedit

Open Save Undo

just_orz.txt x

Undo the last action

The file "/home/bigballon/Desktop/just_orz.txt" changed on disk.

Reload

Cancel

一点总结

- 一个Inode对应一个文件，而一个文件根据其大小，会占用多块blocks。
- 更为准确的来说，一个文件只对应一个Inode，因为硬链接其实不是创建新文件，只是在Directory中写入了新的对应关系而已。
- 当我们删除文件的时候，只是把Inode标记为可用，文件在block中的内容是没有被清除的，只有在有新的文件需要占用block的时候，才会被覆盖。
- 文件系统如何存取文件

根据文件名，通过Directory里的对应关系，找到文件对应的Inode number
再根据Inode number读取到文件的Inode table
再根据Inode table中的Pointer读取到相应的Blocks

- 硬链接有点像指针，软链接有点像快捷方式

- [百度百科](#)
- [Wiki百科](#)
- [理解inode——阮一峰](#)
- [理解 Linux 的硬链接与软链接](#)
- [一天一点学习Linux之Inode详解](#)



抛砖引玉，谢谢聆听！

No zuo no die why you cry
Thanks for listening