## Linux 的文件与目录

现代操作系统为解决信息能独立于进程之外被长期存储引入了文件，文件作为进程创建信息的逻辑单元可被多个进程并发使用。在 UNIX 系统中，操作系统为磁盘上的文本与图像、鼠标与键盘等输入设备及网络交互等 **I/O 操作设计了一组通用 API，使他们被处理时均可统一使用字节流方式。换言之，UNIX 系统中除进程之外的一切皆是文件，而 Linux 保持了这一特性。**为了便于文件的管理，Linux 还引入了目录（有时亦被称为文件夹）这一概念。**目录使文件可被分类管理，且目录的引入使 Linux 的文件系统形成一个层级结构的目录树**。[清单 1.](https://www.ibm.com/developerworks/cn/linux/l-cn-hardandsymb-links/index.html#listing1)所示的是普通 Linux 系统的顶层目录结构，其中 /dev 是存放了设备相关文件的目录。

##### 清单 1. Linux 系统的顶层目录结构

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | /              根目录  ├── bin     存放用户二进制文件  ├── boot    存放内核引导配置文件  ├── dev     存放设备文件  ├── etc     存放系统配置文件  ├── home    **用户主目录**  ├── lib     动态共享库  ├── lost+found  文件系统恢复时的恢复文件  ├── media   可卸载存储介质挂载点  ├── mnt     **文件系统临时挂载点**  ├── opt     附加的应用程序包  ├── proc    系统内存的映射目录，提供内核与进程信息  ├── root    root 用户主目录  ├── sbin    存放系统二进制文件  ├── srv     存放服务相关数据  ├── sys     sys 虚拟文件系统挂载点  ├── tmp   **存放临时文件**  ├── usr     存放用户应用程序  └── var     存放邮件、系统日志等变化文件 |

Linux 与其他类 UNIX 系统一样并不区分文件与目录：**目录是记录了其他文件名的文件。**使用命令 mkdir 创建目录时，若期望创建的目录的名称与现有的文件名（或目录名）重复，则会创建失败。

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | # ls -F /usr/bin/zi\*  /usr/bin/zip\*       /usr/bin/zipgrep\*  /usr/bin/zipnote\*  /usr/bin/zipcloak\*  /usr/bin/zipinfo\*  /usr/bin/zipsplit\*    # mkdir -p /usr/bin/zip  mkdir: cannot create directory `/usr/bin/zip': File exists |

Linux 将设备当做文件进行处理，[清单 2.](https://www.ibm.com/developerworks/cn/linux/l-cn-hardandsymb-links/index.html#listing2)展示了如何打开设备文件 /dev/input/event5 并读取文件内容。文件 event5 表示一种输入设备，其可能是鼠标或键盘等。查看文件 /proc/bus/input/devices 可知 event5 对应设备的类型。设备文件 /dev/input/event5 使用 read() 以字符流的方式被读取。结构体 input\_event 被定义在内核头文件 linux/input.h 中。

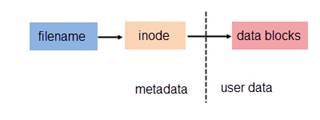
##### 清单 2. 打开并读取设备文件

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | int fd;  struct input\_event ie;  fd = open("/dev/input/event5", O\_RDONLY);  read(fd, &ie, sizeof(struct input\_event));  printf("type = %d  code = %d  value = %d\n",              ie.type, ie.code, ie.value);  close(fd); |

## 硬链接与软链接的联系与区别

我们知道文件都有**文件名与数据**，这在 Linux 上被分成两个部分：**用户数据 (user data) 与元数据 (metadata)。用户数据，**即**文件数据块 (data block)**，数据块是**记录文件真实内容的地方**；而**元数据则是文件的附加属性**，如**文件大小**、**创建时间、所有者**等信息。在 Linux 中，**元数据中的 inode 号**（inode 是文件元数据的一部分但其并不包含文件名，inode 号即索引节点号）**才是文件的唯一标识而非文件名**。文件名仅是为了方便人们的记忆和使用，系统或程序通过 inode 号寻找正确的文件数据块。[图 1.](https://www.ibm.com/developerworks/cn/linux/l-cn-hardandsymb-links/index.html#fig1)展示了程序通过文件名获取文件内容的过程。

##### 图 1. 通过文件名打开文件



##### 清单 3. 移动或重命名文件

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | # stat /home/harris/source/glibc-2.16.0.tar.xz   File: `/home/harris/source/glibc-2.16.0.tar.xz'   Size: 9990512      Blocks: 19520      IO Block: 4096   regular file  Device: 807h/2055d      Inode: 2485677     Links: 1  Access: (0600/-rw-------)  Uid: ( 1000/  harris)   Gid: ( 1000/  harris)  ...  ...  # mv /home/harris/source/glibc-2.16.0.tar.xz /home/harris/Desktop/glibc.tar.xz  # ls -i -F /home/harris/Desktop/glibc.tar.xz  2485677 /home/harris/Desktop/glibc.tar.xz |

在 Linux 系统中查看 inode 号可使用命令 stat 或 ls -i（若是 AIX 系统，则使用命令 istat）。[清单 3.](https://www.ibm.com/developerworks/cn/linux/l-cn-hardandsymb-links/index.html" \l "listing3)中使用命令 **mv 移动**并重命名文件 glibc-2.16.0.tar.xz，其**结果不影响文件的用户数据及 inode 号，文件移动前后 inode 号均为**：2485677。

为解决文件的共享使用，Linux 系统引入了两种链接：**硬链接 (hard link) 与软链接（又称符号链接，即 soft link 或 symbolic link）**。链接为 Linux 系统解决了文件的共享使用，还带来了隐藏文件路径、增加权限安全及节省存储等好处。**若一个 inode 号对应多个文件名，则称这些文件为硬链接。**换言之，**硬链接就是同一个文件使用了多个别名**（见 [图 2.](https://www.ibm.com/developerworks/cn/linux/l-cn-hardandsymb-links/index.html#fig2)hard link 就是 file 的一个别名，他们有共同的 inode）。**硬链接可由命令 link 或 ln 创建**。如下是对文件 oldfile 创建硬链接。

|  |  |
| --- | --- |
| 1  2 | link oldfile newfile  ln oldfile newfile |

由于硬链接是有着相同 inode 号仅文件名不同的文件，因此硬链接存在以下几点特性：

* 文件有相同的 inode 及 data block；
* 只能对已存在的文件进行创建；
* 不能交叉文件系统进行硬链接的创建；
* 不能对目录进行创建，只可对文件创建；
* 删除一个硬链接文件并不影响其他有相同 inode 号的文件。

##### 清单 4. 硬链接特性展示

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | # ls -li  total 0    // 只能对已存在的文件创建硬连接  # link old.file hard.link  link: cannot create link `hard.link' to `old.file': No such file or directory    # echo "This is an original file" > old.file  # cat old.file  This is an original file  # stat old.file   File: `old.file'   Size: 25           Blocks: 8          IO Block: 4096   regular file  Device: 807h/2055d      Inode: 660650      Links: 2  Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)  ...  // 文件有相同的 inode 号以及 data block  # link old.file hard.link | ls -li  total 8  660650 -rw-r--r-- 2 root root 25 Sep  1 17:44 hard.link  660650 -rw-r--r-- 2 root root 25 Sep  1 17:44 old.file    // 不能交叉文件系统  # ln /dev/input/event5 /root/bfile.txt  ln: failed to create hard link `/root/bfile.txt' => `/dev/input/event5':  Invalid cross-device link    // 不能对目录进行创建硬连接  # mkdir -p old.dir/test  # ln old.dir/ hardlink.dir  ln: `old.dir/': hard link not allowed for directory  # ls -iF  660650 hard.link  657948 old.dir/  660650 old.file |

**文件 old.file 与 hard.link 有着相同的 inode 号：660650 及文件权限，inode 是随着文件的存在而存在，因此只有当文件存在时才可创建硬链接，即当 inode 存在且链接计数器（link count）不为 0 时。inode 号仅在各文件系统下是唯一的，当 Linux 挂载多个文件系统后将出现 inode 号重复的现象**（如 [清单 5.](https://www.ibm.com/developerworks/cn/linux/l-cn-hardandsymb-links/index.html#listing5)所示，文件 t3.jpg、sync 及 123.txt 并无关联，却有着相同的 inode 号），**因此硬链接创建时不可跨文件系统。**设备文件目录 /dev 使用的文件系统是 devtmpfs，而 /root（与根目录 / 一致）使用的是磁盘文件系统 ext4。[清单 5.](https://www.ibm.com/developerworks/cn/linux/l-cn-hardandsymb-links/index.html" \l "listing5)展示了使用命令 df 查看当前系统中挂载的文件系统类型、各文件系统 inode 使用情况及文件系统挂载点。

##### 清单 5. 查找有相同 inode 号的文件

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | # df -i --print-type  Filesystem     Type       Inodes  IUsed    IFree IUse% Mounted on  /dev/sda7      ext4      3147760 283483  2864277   10% /  udev           devtmpfs   496088    553   495535    1% /dev  tmpfs          tmpfs      499006    491   498515    1% /run  none           tmpfs      499006      3   499003    1% /run/lock  none           tmpfs      499006     15   498991    1% /run/shm  /dev/sda6      fuseblk  74383900   4786 74379114    1% /media/DiskE  /dev/sda8      fuseblk  29524592  19939 29504653    1% /media/DiskF    # find / -inum 1114  /media/DiskE/Pictures/t3.jpg  /media/DiskF/123.txt  /bin/sync |

值得一提的是，Linux 系统存在 inode 号被用完但磁盘空间还有剩余的情况。我们创建一个 5M 大小的 ext4 类型的 mo.img 文件，并将其挂载至目录 /mnt。然后我们使用一个 shell 脚本将挂载在 /mnt 下 ext4 文件系统的 indoe 耗尽（见清单 6.）。

##### 清单 6. 测试文件系统 inode 耗尽但仍有磁盘空间的情景

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | # dd if=/dev/zero of=mo.img bs=5120k count=1  # ls -lh mo.img  -rw-r--r-- 1 root root 5.0M Sep  1 17:54 mo.img  # mkfs -t ext4  -F ./mo.img  ...  OS type: Linux  Block size=1024 (log=0)  Fragment size=1024 (log=0)  Stride=0 blocks, Stripe width=0 blocks  1280 inodes, 5120 blocks  256 blocks (5.00%) reserved for the super user  ...  ...  Writing superblocks and filesystem accounting information: done    # mount -o loop ./mo.img /mnt  # cat /mnt/inode\_test.sh  #!/bin/bash    for ((i = 1; ; i++))  do     if [ $? -eq 0 ]; then         echo  "This is file\_$i" > file\_$i     else         exit 0     fi  done    # ./inode\_test.sh  ./inode\_test.sh: line 6: file\_1269: No space left on device    # df -iT /mnt/; du -sh /mnt/  Filesystem     Type Inodes IUsed IFree IUse% Mounted on  /dev/loop0     ext4   1280  1280     0  100% /mnt  1.3M    /mnt/ |

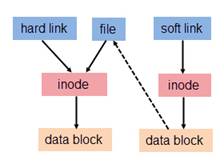
**硬链接不能对目录创建是受限于文件系统的设计（见**[**清单 4.**](https://www.ibm.com/developerworks/cn/linux/l-cn-hardandsymb-links/index.html#listing4)**对目录创建硬链接将失败）**。现 Linux 文件系统中的目录均隐藏了两个个特殊的目录：当前目录（.）与父目录（..）。查看这两个特殊目录的 inode 号可知其实这两目录就是两个硬链接（注意目录 /mnt/lost+found/ 的 inode 号）。若系统允许对目录创建硬链接，则会产生目录环。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | # ls -aliF /mnt/lost+found  total 44  11 drwx------ 2 root root 12288 Sep  1 17:54 ./  2 drwxr-xr-x 3 root root 31744 Sep  1 17:57 ../    # stat  /mnt/lost+found/   File: `/mnt/lost+found/'   Size: 12288        Blocks: 24         IO Block: 1024   directory  Device: 700h/1792d      Inode: 11          Links: 2  Access: (0700/drwx------)  Uid: (    0/    root)   Gid: (    0/    root)  Access: 2012-09-01 17:57:17.000000000 +0800  Modify: 2012-09-01 17:54:49.000000000 +0800  Change: 2012-09-01 17:54:49.000000000 +0800  Birth: - |

**软链接与硬链接不同，若文件用户数据块中存放的内容是另一文件的路径名的指向，则该文件就是软连接。**软链接就是一个普通文件，只是数据块内容有点特殊。**软链接有着自己的 inode 号以及用户数据块（见**[**图 2.**](https://www.ibm.com/developerworks/cn/linux/l-cn-hardandsymb-links/index.html#fig2)**）。因此软链接的创建与使用没有类似硬链接的诸多限制：**

* 软链接有自己的文件属性及权限等；
* 可对不存在的文件或目录创建软链接；
* 软链接可交叉文件系统；
* 软链接可对文件或目录创建；
* 创建软链接时，链接计数 i\_nlink 不会增加；
* 删除软链接并不影响被指向的文件，但若被指向的原文件被删除，则相关软连接被称为死链接（即 dangling link，若被指向路径文件被重新创建，死链接可恢复为正常的软链接）。

##### 图 2. 软链接的访问



##### 清单 7. 软链接特性展示

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | # ls -li   total 0     // 可对不存在的文件创建软链接   # ln -s old.file soft.link   # ls -liF   total 0   789467 lrwxrwxrwx 1 root root 8 Sep  1 18:00 soft.link -> old.file     // 由于被指向的文件不存在，此时的软链接 soft.link 就是死链接   # cat soft.link   cat: soft.link: No such file or directory     // 创建被指向的文件 old.file，soft.link 恢复成正常的软链接   # echo "This is an original file\_A" >> old.file   # cat soft.link   This is an original file\_A     // 对不存在的目录创建软链接   # ln -s old.dir soft.link.dir   # mkdir -p old.dir/test   # tree . -F --inodes   .  ├── [ 789497]  old.dir/  │   └── [ 789498]  test/  ├── [ 789495]  old.file  ├── [ 789495]  soft.link -> old.file  └── [ 789497]  soft.link.dir -> old.dir/ |

当然软链接的用户数据也可以是另一个软链接的路径，其解析过程是递归的。但需注意：软链接创建时原文件的路径指向使用绝对路径较好。使用相对路径创建的软链接被移动后该软链接文件将成为一个死链接（如下所示的软链接 a 使用了相对路径，因此不宜被移动），因为链接数据块中记录的亦是相对路径指向。

|  |  |
| --- | --- |
| 1  2  3  4  5 | $ ls -li  total 2136  656627 lrwxrwxrwx 1 harris harris       8 Sep  1 14:37 a -> data.txt  656662 lrwxrwxrwx 1 harris harris       1 Sep  1 14:37 b -> a  656228 -rw------- 1 harris harris 2186738 Sep  1 14:37 data.txt 6 |

## 链接相关命令

在 Linux 中查看当前系统已挂着的文件系统类型，除上述使用的命令 df，还可使用命令 mount 或查看文件 /proc/mounts。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | # mount  /dev/sda7 on / type ext4 (rw,errors=remount-ro)  proc on /proc type proc (rw,noexec,nosuid,nodev)  sysfs on /sys type sysfs (rw,noexec,nosuid,nodev)  ...  ...  none on /run/shm type tmpfs (rw,nosuid,nodev) |

命令 ls 或 stat 可帮助我们区分软链接与其他文件并查看文件 inode 号，但较好的方式还是使用 find 命令，其不仅可查找某文件的软链接，还可以用于查找相同 inode 的所有硬链接。（见清单 8.）

##### 清单 8. 使用命令 find 查找软链接与硬链接

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | // 查找在路径 /home 下的文件 data.txt 的软链接  # find /home -lname data.txt  /home/harris/debug/test2/a    // 查看路径 /home 有相同 inode 的所有硬链接  # find /home -samefile /home/harris/debug/test3/old.file  /home/harris/debug/test3/hard.link  /home/harris/debug/test3/old.file    # find /home -inum 660650  /home/harris/debug/test3/hard.link  /home/harris/debug/test3/old.file    // 列出路径 /home/harris/debug/ 下的所有软链接文件  # find /home/harris/debug/ -type l -ls  656662 0 lrwxrwxrwx 1 harris harris 1 Sep 1 14:37 /home/harris/debug/test2/b -> a  656627 0 lrwxrwxrwx 1 harris harris 8 Sep 1 14:37 /home/harris/debug/test2/a ->  data.txt  789467 0 lrwxrwxrwx 1 root root 8 Sep 1 18:00 /home/harris/debug/test/soft.link ->  old.file  789496    0 lrwxrwxrwx   1 root     root            7 Sep  1 18:01  /home/harris/debug/test/soft.link.dir -> old.dir |

系统根据磁盘的大小默认设定了 inode 的值（见清单 9.），如若必要，可在格式文件系统前对该值进行修改。如键入命令 mkfs -t ext4 -I 512/dev/sda4，将使磁盘设备 /dev/sda4 格式成 inode 大小是 512 字节的 ext4 文件系统。

##### 清单 9. 查看系统的 inode 值

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | // 查看磁盘分区 /dev/sda7 上的 inode 值  # dumpe2fs -h /dev/sda7 | grep "Inode size"  dumpe2fs 1.42 (29-Nov-2011)  Inode size:              256    # tune2fs -l /dev/sda7 | grep "Inode size"  Inode size:              256 |

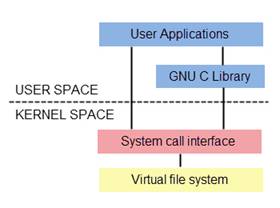
## Linux VFS

Linux 有着极其丰富的文件系统，大体上可分如下几类：

1. 网络文件系统，如 nfs、cifs 等；
2. 磁盘文件系统，如 ext4、ext3 等；
3. 特殊文件系统，如 proc、sysfs、ramfs、tmpfs 等。

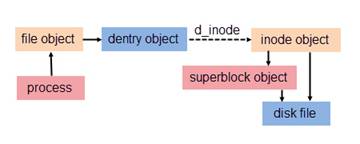
实现以上这些文件系统并在 Linux 下共存的基础就是 Linux VFS（Virtual File System 又称 Virtual Filesystem Switch），即虚拟文件系统。VFS 作为一个通用的文件系统，抽象了文件系统的四个基本概念：**文件、目录项 (dentry)、索引节点 (inode) 及挂载点，**其在内核中为用户空间层的文件系统提供了相关的接口（见 [图 3.](https://www.ibm.com/developerworks/cn/linux/l-cn-hardandsymb-links/index.html#fig3)所示 VFS 在 Linux 系统的架构）。VFS 实现了 open()、read() 等系统调并使得 cp 等用户空间程序可跨文件系统。VFS 真正实现了上述内容中：在 Linux 中除进程之外一切皆是文件。

##### 图 3. VFS 在系统中的架构



Linux VFS 存在四个基本对象：超级块对象 (superblock object)、索引节点对象 (inode object)、目录项对象 (dentry object) 及文件对象 (file object)。超级块对象代表一个已安装的文件系统；索引节点对象代表一个文件；目录项对象代表一个目录项，如设备文件 event5 在路径 /dev/input/event5 中，其存在四个目录项对象：/ 、dev/ 、input/ 及 event5。文件对象代表由进程打开的文件。这四个对象与进程及磁盘文件间的关系如图 4. 所示，其中 d\_inode 即为硬链接。为文件路径的快速解析，Linux VFS 设计了目录项缓存（Directory Entry Cache，即 dcache）。

##### 图 4. VFS 的对象之间的处理



## Linux 文件系统中的 inode

在 Linux 中，索引节点结构存在于系统内存及磁盘，其可区分成 VFS inode 与实际文件系统的 inode。VFS inode 作为实际文件系统中 inode 的抽象，定义了结构体 inode 与其相关的操作 inode\_operations（见内核源码 include/linux/fs.h）。

##### 清单 10. VFS 中的 inode 与 inode\_operations 结构体

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | struct inode {     ...     const struct inode\_operations   \*i\_op; // 索引节点操作     unsigned long           i\_ino;      // 索引节点号     atomic\_t                i\_count;    // 引用计数器     unsigned int            i\_nlink;    // 硬链接数目     ...  }    struct inode\_operations {     ...     int (\*create) (struct inode \*,struct dentry \*,int, struct nameidata \*);     int (\*link) (struct dentry \*,struct inode \*,struct dentry \*);     int (\*unlink) (struct inode \*,struct dentry \*);     int (\*symlink) (struct inode \*,struct dentry \*,const char \*);     int (\*mkdir) (struct inode \*,struct dentry \*,int);     int (\*rmdir) (struct inode \*,struct dentry \*);     ...  } |

如清单 10. 所见，每个文件存在两个计数器：i\_count 与 i\_nlink，即引用计数与硬链接计数。结构体 inode 中的 i\_count 用于跟踪文件被访问的数量，而 i\_nlink 则是上述使用 ls -l 等命令查看到的文件硬链接数。或者说 i\_count 跟踪文件在内存中的情况，而 i\_nlink 则是磁盘计数器。当文件被删除时，则 i\_nlink 先被设置成 0。文件的这两个计数器使得 Linux 系统升级或程序更新变的容易。系统或程序可在不关闭的情况下（即文件 i\_count 不为 0），将新文件以同样的文件名进行替换，新文件有自己的 inode 及 data block，旧文件会在相关进程关闭后被完整的删除。

##### 清单 11. 文件系统 ext4 中的 inode

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | struct ext4\_inode {     ...     \_\_le32  i\_atime;        // 文件内容最后一次访问时间     \_\_le32  i\_ctime;        // inode 修改时间     \_\_le32  i\_mtime;        // 文件内容最后一次修改时间     \_\_le16  i\_links\_count;  // 硬链接计数     \_\_le32  i\_blocks\_lo;    // Block 计数     \_\_le32  i\_block[EXT4\_N\_BLOCKS];  // 指向具体的 block     ...  }; |

清单 11. 展示的是文件系统 ext4 中对 inode 的定义（见内核源码 fs/ext4/ext4.h）。其中三个时间的定义可对应与命令 stat 中查看到三个时间。i\_links\_count 不仅用于文件的硬链接计数，也用于目录的子目录数跟踪（目录并不显示硬链接数，命令 ls -ld 查看到的是子目录数）。由于文件系统 ext3 对 i\_links\_count 有限制，其最大数为：32000（该限制在 ext4 中被取消）。尝试在 ext3 文件系统上验证目录子目录及普通文件硬链接最大数可见 [清单 12](https://www.ibm.com/developerworks/cn/linux/l-cn-hardandsymb-links/index.html#listing12). 的错误信息。因此实际文件系统的 inode 之间及与 VFS inode 相较是有差异的。

##### 清单 12. 文件系统 ext3 中 i\_links\_count 的限制

|  |  |
| --- | --- |
| 1  2  3  4  5 | # ./dirtest.sh  mkdir: cannot create directory `dir\_31999': Too many links    # ./linkcount.sh  ln: failed to create hard link to `old.file': Too many links |

# 它是用来干什么的？

**你可以将链接简单地理解为 Windows 中常见的快捷方式**（或是 OS X 中的替身），Linux 中常用它来解决一些库版本的问题，**通常也会将一些目录层次较深的文件链接到一个更易访问的目录中。**在这些用途上，我们通常会使用到**软链接**（也称**符号链接**）。

1. 为test文件创建硬链接如下：

shell-test:/u/nanaal/shell$ cp -l test test1

shell-test:/u/nanaal/shell$ ln -l test test2

shell-test:/u/nanaal/shell$ ls -li

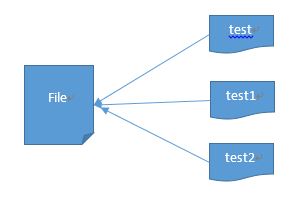
total 12

4298659818 -rw-rw-r--. 3 nanaal nanaal 6 Apr 19 06:24 test

4298659818 -rw-rw-r--. 3 nanaal nanaal 6 Apr 19 06:24 test1

4298659818 -rw-rw-r--. 3 nanaal nanaal 6 Apr 19 06:24 test2

命令ls显示的第一列为文件索引，在文件系统每个文件都有一个唯一的索引值。从上面的信息可以知道，test、test1、test2共享一份文件，且链接计数为3（ls显示结果的第三列）。根据上面信息，硬链接可表示为：



Note：

硬链接的所有文件必须在一个硬盘上，如test在磁盘/dev/sda1上面，那么test1和test2也必须在/dev/sda1上面。

1. 为test文件创建软链接如下：

shell-test:/u/nanaal/shell$ cp -s test test3

shell-test:/u/nanaal/shell$ ln -s test test4

shell-test:/u/nanaal/shell$ ls -li

total 12

4298659818 -rw-rw-r--. 3 nanaal nanaal 6 Apr 19 06:24 test

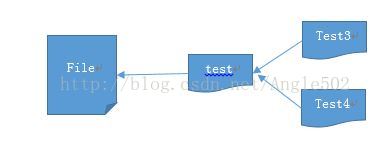
4298659818 -rw-rw-r--. 3 nanaal nanaal 6 Apr 19 06:24 test1

4298659818 -rw-rw-r--. 3 nanaal nanaal 6 Apr 19 06:24 test2

4298103063 lrwxrwxrwx. 1 nanaal nanaal 4 Apr 19 06:43 test3 -> test

4304030281 lrwxrwxrwx. 1 nanaal nanaal 4 Apr 19 06:43 test4 -> test

从ls命令的输出结果可以看到， test3和test4的文件索引值均与test的索引值不同，即test3和test4均是新的文件，但是文件大小比test小，且链接计数为1. 软链接文件test3、test4只保存了源文件test的文件信息，并没有对test文件指定的数据进行拷贝。根据上面信息，软链接可表示为如下：



Note:

软链接的所有文件可以保存在不同的磁盘上。如test在磁盘/dev/sda1上面，test3和test4可以保存在/dev/sda2上面。

1. 修改文件

1) 修改源文件test

2) 修改硬链接文件test1或test2

3) 修改软链接文件test3或test4

修改上面的文件，会发现所有的文件均跟着修改了，这这方面，软链接和硬链接没有区别。

1. 删除文件

1)删除硬链接文件

shell-test:/u/nanaal/shell$ rm -rf test1

shell-test:/u/nanaal/shell$ ls -li

total 8

4298659818 -rw-rw-r--. 2 nanaal nanaal 6 Apr 19 06:24 test

4298659818 -rw-rw-r--. 2 nanaal nanaal 6 Apr 19 06:24 test2

4298103063 lrwxrwxrwx. 1 nanaal nanaal 4 Apr 19 06:43 test3 -> test

4304030281 lrwxrwxrwx. 1 nanaal nanaal 4 Apr 19 06:43 test4 -> test

**从ls的输出可以看到，删除文件test1后，test和test2的链接计数由3变成了2. 文件test和test2可以正常阅读。**

2) 删除软链接文件

shell-test:/u/nanaal/shell$ ls -li

total 8

4298659818 -rw-rw-r--. 2 nanaal nanaal 6 Apr 19 06:24 test

4298659818 -rw-rw-r--. 2 nanaal nanaal 6 Apr 19 06:24 test2

4304030281 lrwxrwxrwx. 1 nanaal nanaal 4 Apr 19 06:43 test4 -> test

**从ls的输出来看，删除文件test3，对其他文件没有任何影响。且文件test和test4可以正常阅读。**

3) 删除源文件

shell-test:/u/nanaal/shell$ rm -rf test

shell-test:/u/nanaal/shell$ ls -li

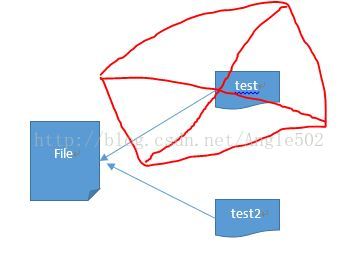
total 4

4298659818 -rw-rw-r--. 1 nanaal nanaal 6 Apr 19 06:24 test2

4304030281 lrwxrwxrwx. 1 nanaal nanaal 4 Apr 19 06:43 test4 -> test

**从ls的输出可以看出，删除文件test后，test2的链接计数由2变为1，可以阅读。文件test4属性没有任何变化，但是已经不能阅读。硬链接和软链接在这点的不同，可以表示如下：**

硬链接文件删除操作：



软链接文件删除操作：

