第7章 Linux文件与目录管理

本章主要介绍Linux系统里文本和目录的相关操作。有一句话：‘在Linux里一切皆文件’， 是呀文件是Linux的基石，小到一个配置文件，大到一块磁盘都是用文件来控制的。这一部分内容比较基础，有较多基础命令出现，希望你多练习，多使用。用的多了自然而然就熟练了。

绝对路径和相对路径

在Linux中什么是一个文件的路径呢，说白了就是这个文件存在的地方，例如在上一章提到的 `/root/.ssh/authorized_keys` 这就是一个文件的路径。如果你告诉系统这个文件的路径，那么系统就可以找到这个文件。在Linux的世界中，存在着绝对路径和相对路径。

绝对路径：路径的写法一定由根目录 ‘/’写起，例如 `/usr/local/mysql` 这就是绝对路径。

相对路径：路径的写法不是由根目录 ‘/’写起，例如，首先用户进入到/， 然后再进入到home ，命令为 `cd /home` 然后 `cd test` 此时用户所在的路径为 `/home/test` 第一个cd命令后跟 ‘/home’ 第二个cd命令后跟 ‘test’, 并没有斜杠，这个 ‘test’ 是相对于 ‘/home’ 目录来讲的，所以叫做相对路径。

命令：cd

这个命令是用来变更用户所在目录的，后面如果什么都不跟，就会直接到当前用户的根目录下，我们做实验用的是 ‘root’ 账户，所以运行 `cd` 后，会进入root账户的根目录 ‘/root’. 后面跟目录名，则会直接切换到指定目录下：

```
[root@localhost ~]# cd /tmp/
[root@localhost tmp]# pwd
/tmp
[root@localhost tmp]# cd
[root@localhost ~]# pwd
/root
```

`pwd` 这个命令打印出当前所在目录，`cd` 后面只能是目录名，而不能是文件名，如果跟了文件名会报错：

```
[root@localhost ~]# cd /etc/passwd
-bash: cd: /etc/passwd: 不是目录
```

`./` 表示当前目录，`../` 表示当前目录的上一级目录：

```
[root@localhost ~]# cd /usr/local/lib/
[root@localhost lib]# pwd
/usr/local/lib
[root@localhost lib]# cd ./
[root@localhost lib]# pwd
/usr/local/lib
[root@localhost lib]# cd ../
[root@localhost local]# pwd
/usr/local
```

上例中，首先进入到 `/usr/local/lib/` 目录下，然后再进入 `./` 其实还是进入到当前目录下，用 `pwd` 查看当前目录，并没有发生变化，然后再进入 `../` 则是进入到了 `/usr/local/` 目录下，即 `/usr/local/lib` 目录的上一级目录。

命令：mkdir

用来创建目录的，这个命令在上一章节中用到过。 ‘mkdir’ 其实就是make directory的缩写。其语法为mkdir [-mp] [目录名称] ，其中-m, -p为其选项， ‘-m’ 这个选项用来指定要创建目录的权限，不常用，阿铭不做重点解释。 ‘-p’ 这个选项很管用，先来做个试验，你会一目了然的：

```
[root@localhost ~]# mkdir /tmp/test/123
mkdir: 无法创建目录 '/tmp/test/123': 没有那个文件或目录
[root@localhost ~]# mkdir -p /tmp/test/123
[root@localhost ~]# ls /tmp/test
123
```

当我们想创建 `/tmp/test/123` 目录，可是提示不能创建，原因是 `/tmp/test` 目录不存在，你会说，这个Linux怎么这样傻，`/tmp/test` 目录不存在就自动创建不就OK了嘛，的确Linux确实很傻，如果它发现要创建的目录的上一级目录不存在就会报错。然而Linux并不是那么傻，因为它也为我们想好了解决办法，即 ‘-p’ 选项，这个选项可以帮我们创建一大串级联目录，这个选项还有一个好处，那就是当你创建一个已经存在的目录

目录列表

时，不会报错：

```
[root@localhost ~]# ls -ld /tmp/test/123
drwxr-xr-x. 2 root root 4096 5月   9 19:10 /tmp/test/123
[root@localhost ~]# mkdir /tmp/test/123
mkdir: 无法创建目录 '/tmp/test/123': 文件已存在
[root@localhost ~]# mkdir -p /tmp/test/123
[root@localhost ~]# ls -ld /tmp/test/123
drwxr-xr-x. 2 root root 4096 5月   9 19:10 /tmp/test/123
```

在上一章节里，阿铭已经介绍过 `ls` 命令，但是并没有向你介绍它的 '-d' 选项，这个选项是针对目录的，通常都是和 '-l' 同时使用写成 '-ld'. 它可以查看指定目录的属性，比如在本例中，它可以查看 '/tmp/test/123' 目录的创建时间。 `mkdir -p` 后面跟一个已经存在的目录名时，它不会做任何事情，只是不报错而已。

命令：rmdir

用来删除空目录，后面可以是一个也可以是多少，多个的话用空格分隔。该命令阿铭很少使用，因为它只能删除目录，不能删除文件，还有一个命令 `rm` 既可以删除目录又可以删除文件，阿铭用的比较多。rmdir 有一个和mkdir一样的选项 '-p', 同样可以级联删除一大串目录，但是级联的目录中其中一个目录里还有目录或者文件时就不好用了。

```
[root@localhost ~]# ls /tmp/test
123
[root@localhost ~]# rmdir /tmp/test/
rmdir: 删除 '/tmp/test/' 失败: 目录非空
[root@localhost ~]# rmdir /tmp/test/123
[root@localhost ~]# ls /tmp/test
[root@localhost ~]#
```

所以，得出的结论是， 'rmdir' 只能删除空目录，即使加上 '-p' 选项也只能删除一串的空目录，可见这个命令有很大的局限性，偶尔用下还可以。

命令：rm

这个命令是最常用的， 'rm' 同样也有很多选项。你可以通过 `man rm` 来获得详细帮助信息。在这里阿铭只介绍最常用的两个选项。

'-r'：删除目录用的选项，等同于rmdir.

```
[root@localhost ~]# mkdir -p /tmp/test/123
[root@localhost ~]# rm -r /tmp/test/123
rm: 是否删除目录 '/tmp/test/123'? y
```

但是和rmdir不同的是，使用 `rm -r` 删除目录时，会问一下是否删除，如果输入 'y' 则会删除，输入 'n' 则不删除。当然 `rm -r` 也不会向rmdir不能删除非空目录，它是可以删除非空目录的。

'-f' ：表示强制删除，不再提示是否要删除，而是直接就删除了，而后面跟一个不存在的文件或者目录时，也不会报错，如果不加 '-f' 选项会报错。

```
[root@localhost ~]# rm /tmp/test/123/123
rm: 无法删除 '/tmp/test/123/123': 没有那个文件或目录
[root@localhost ~]# rm -f /tmp/test/123/123
```

要删除一个目录时，即使加上 '-f' 选项也会报错，所以删除目录一定要加 '-r' 选项。

```
[root@localhost ~]# rm -f /tmp/test/123
rm: 无法删除 '/tmp/test/123': 是一个目录
[root@localhost ~]# rm -rf /tmp/test/123
```

关于rm，阿铭使用最多便是 '-rf' 两个选项合用了。不管删除文件还是目录都可以。但是方便的同时也要多注意，万一你的手太快后边跟了/那样就会把你的系统文件全部删除的，切记切记。

环境变量PATH

在讲环境变量之前阿铭先介绍一个命令 `which`, 它用来查找某个命令的绝对路径。

```
[root@localhost ~]# which rmdir
/bin/rmdir
[root@localhost ~]# which rm
alias rm='rm -i'
        /bin/rm
[root@localhost ~]# which ls
alias ls='ls --color=auto'
        /bin/ls
```

'rm' 和 'ls' 是两个特殊的命令，使用 `alias` 命令做了别名。我们用的 'rm' 实际上是 'rm -i' 加上 '-i' 选项后，删除文件或者命令时都会问一下是否确定要删除，这样做比较安全。 'alias' 可以设置命令的别名也可以设置文件的别名，阿铭会在以后章节中详细介绍。 'which' 这个命令阿铭平时只用来查询某个命令的绝对路

径，不常使用。

上边提到了alias，也提到了绝对路径的/bin/rm ，然后你意识到没有，为什么我们输入很多命令时是直接打出了命令，而没有去使用这些命令的绝对路径？这是因为环境变量PATH在起作用了。请输入 echo $PATH，这里的echo其实就是打印的意思，而PATH前面的$表示后面接的是变量。

```
[root@localhost ~]# echo $PATH
/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin
```

因为/bin 在PATH的设定中，所以自然就可以找到ls了。如果你将 ls 移动到 /root 底下的话，然后你自己本身也在 /root 底下，但是当你执行 ls 的时候，他就是不理你？怎么办？这是因为 PATH 没有 /root 这个目录，而你又将 ls 移动到 /root 底下了，自然系统就找不到可执行文件了，因此就会告诉你 'command not found!'

```
[root@localhost ~]# mv /bin/ls /root/
[root@localhost ~]# ls
-bash: /bin/ls: 没有那个文件或目录
```

'mv' 用来移动目录或者文件，还可以重命名，稍后讲解。

那么该怎么克服这种问题呢？有两个方法，一种方法是直接将 /root 的路径加入 $PATH 当中！如何增加？可以使用命令 `PATH=$PATH:/root`:

```
[root@localhost ~]# PATH=$PATH:/root
[root@localhost ~]# echo $PATH
/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin:/root
[root@localhost ~]# ls
anaconda-ks.cfg  install.log  install.log.syslog  ls
```

另一种方法就是使用绝对路径：

```
[root@localhost ~]# /root/ls
anaconda-ks.cfg  install.log  install.log.syslog  ls
```

命令：cp

copy的简写，即拷贝。格式为 `cp [选项] [ 来源文件 ] [目的文件]` ，例如我想把test1 拷贝成test2 ，这样即可 cp test1 test2，以下介绍几个常用的选项：

-r：如果你要拷贝一个目录，必须要加-r选项，否则你是拷贝不了目录的, 和 'rm' 类似。

```
[root@localhost ~]# mkdir 123
[root@localhost ~]# cp 123 456
cp: 略过目录'123'
[root@localhost ~]# cp -r 123 456
[root@localhost ~]# ls -l
总用量 44
drwxr-xr-x. 2 root root  4096 5月  10 04:05 123
drwxr-xr-x. 2 root root  4096 5月  10 04:05 456
```

-i ：安全选项，和 'rm' 类似，如果遇到一个存在的文件，会问是否覆盖。在Redhat/CentOS系统中，我们使用的cp其实是cp -i

```
[root@localhost ~]# which cp
alias cp='cp -i'
        /bin/cp
```

下面简单做一个小试验，很快你就会明白-i 选项的作用了。

```
[root@localhost ~]# cd 123
[root@localhost 123]# ls
[root@localhost 123]# touch 111
[root@localhost 123]# touch 222
[root@localhost 123]# cp -i 111 222
cp：是否覆盖 '222'？ n
[root@localhost 123]# echo 'abc' > 111
[root@localhost 123]# echo 'def' > 222
[root@localhost 123]# cat 111 222
abc
def
[root@localhost 123]# /bin/cp 111 222
[root@localhost 123]# cat 111
abc
[root@localhost 123]# cat 222
abc
```

例子中的 'touch' 看字面意思就是 '摸一下'，没错，如果有这个文件，则会改变文件的访问时间，如果没有这个文件就会创建这个文件。前面说过 'echo' 用来打印，在这里所echo的内容 'abc' 和 'def' 并没有显示在屏幕上，而是分别写进了文件 111和222，其写入作用的就是这个大于号 '>' 在linux中这叫做重定向，即把前面产生的输出写入到后面的文件中。在以后的章节中会做详细介绍，这里你要明白它的含义即可。而 'cat' 命

令则是读一个文件，并把读出的内容打印到当前屏幕上。该命令也会在后续章节中详细介绍。

命令: mv

'mv' 是move的简写。格式为 `mv [ 选项 ] [源文件] [目标文件]` 下面介绍几个常用的选项。

-i ：和cp的-i 一样，当目标文件存在时会问用户是否要覆盖。在Redhat/CentOS系统中，我们使用的mv其实是mv -i

该命令有几种情况：

1） 目标文件是目录，而且目标文件不存在；

2） 目标文件是目录，而且目标文件存在；

3） 目标文件不是目录不存在；

4） 目标文件不是目录存在；

目标文件是目录，存在和不存在，移动的结果是不一样的，如果存在，则会把源文件移动到目标文件目录中。否则，就直接移动了，相当于重命名。这样说也许你会觉得有点不好理解，看例子吧。

```
[root@localhost ~]# mkdir dira dirb
[root@localhost ~]# ls
anaconda-ks.cfg  dira  dirb  install.log  install.log.syslog
[root@localhost ~]# mv dira dirc
[root@localhost ~]# ls
anaconda-ks.cfg  dirb  dirc  install.log  install.log.syslog
```

目标文件为目录，并且目标目录不存在，相当于把 'dira' 重命名为 'dirc'.

```
[root@localhost ~]# mv dirc dirb
[root@localhost ~]# ls
anaconda-ks.cfg  dirb  install.log  install.log.syslog
[root@localhost ~]# ls dirb
dirc
```

目标文件为目录，且目标目录存在，则会把 'dirc' 移动到 'dirb' 目录里。

```
[root@localhost ~]# touch filed
[root@localhost ~]# ls
anaconda-ks.cfg  dirb  filed  install.log  install.log.syslog
[root@localhost ~]# mv filed filee
[root@localhost ~]# ls
anaconda-ks.cfg  dirb  filee  install.log  install.log.syslog
[root@localhost ~]# mv filee dirb
[root@localhost ~]# ls
anaconda-ks.cfg  dirb  install.log  install.log.syslog
[root@localhost ~]# ls dirb
dirc  filee
```

目标文件不是目录，且不存在，则会重命名文件。

几个和文档相关的命令

命令: cat

比较常用的一个命令，即查看一个文件的内容并显示在屏幕上，后面可以不加任何选项直接跟文件名，阿铭介绍两个常用的选项：

-n ：查看文件时，把行号也显示到屏幕上。

```
[root@localhost ~]# echo  '111111111' > dirb/filee
[root@localhost ~]# echo  '222222222' >> dirb/filee
[root@localhost ~]# cat dirb/filee
111111111
222222222
[root@localhost ~]# cat -n dirb/filee
     1  111111111
     2  222222222
```

上例中出现了一个 '>>' 这个符号跟前面介绍的 '>' 的作用都是重定向，即把前面输出的东西输入到后边的文件中，只是 '>>' 是追加的意思，而用 '>' 如果文件中有内容则会删除文件中内容，而 '>>' 则不会。

-A ：显示所有东西出来，包括特殊字符

```
[root@localhost ~]# cat -A dirb/filee
111111111$
222222222$
```

命令: tac

和 'cat' 一样，用来把文件的内容显示在屏幕上，只不过是先显示最后一行，然后是倒数第二行，最后显示的是第一行。

```
[root@localhost ~]# tac dirb/filee
222222222
111111111
```

命令: more

也是用来查看一个文件的内容，后面直接跟文件名，当文件内容太多，一屏幕不能占下，而你用 ‘cat’ 肯定是看不前面的内容的，那么使用 ‘more’ 就可以解决这个问题了。当看完一屏后按空格键继续看下一屏。但看完所有内容后就会退出。如果你想提前退出，只需按 ‘q’ 键即可。

命令: less

作用跟more一样，后面直接跟文件名，但比more好在可以上翻，下翻。空格键同样可以翻页，而按 ‘j’ 键可以向下移动（按一下就向下移动一行），按 ‘k’ 键向上移动。在使用more和less查看某个文件时，你可以按一下 ‘/’ 键，然后输入一个word回车，这样就可以查找这个word了。如果是多个该word可以按 ‘n’ 键显示下一个。另外你也可以不按 ‘/’ 而是按 ‘?’ 后边同样跟word来搜索这个word，唯一不同的是， ‘/’ 是在当前行向下搜索，而 ‘?’ 是在当前行向上搜索。

命令: head

‘head’后直接跟文件名，则显示文件的前十行。如果加 -n 选项则显示文件前n行。

```
[root@localhost ~]# head /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
[root@localhost ~]# head -n 1 /etc/passwd
root:x:0:0:root:/root:/bin/bash
[root@localhost ~]# head -n2 /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
```

‘-n’后可以有空格也可以无空格。

命令: tail

和head一样，后面直接跟文件名，则显示文件最后十行。如果加-n 选项则显示文件最后n行。

```
[root@localhost ~]# head -n2 /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
[root@localhost ~]# tail /etc/passwd
nobody:x:99:99:Nobody:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
vcsa:x:69:69:virtual console memory owner:/dev:/sbin/nologin
haldaemon:x:68:68:HAL daemon:/:/sbin/nologin
ntp:x:38:38::/etc/ntp:/sbin/nologin
saslauth:x:499:76:'Saslauthd user':/var/empty/saslauth:/sbin/nologin
postfix:x:89:89::/var/spool/postfix:/sbin/nologin
abrt:x:173:173::/etc/abrt:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
tcpdump:x:72:72::/:/sbin/nologin
[root@localhost ~]# tail -n2 /etc/passwd
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
tcpdump:x:72:72::/:/sbin/nologin
```

-f ： 动态显示文件的最后十行，如果文件是不断增加的，则用-f 选项。如：`tail -f /var/log/messages` 该选项特别特别常用，请熟记。

文件的所属主以及所属组

一个linux目录或者文件，都会有一个所属主和所属组。所属主，即文件的拥有者，而所属组，即该文件所属主所在的一个组。Linux这样设置文件属性的目的是为了文件的安全。例如，test文件的所属主是user0　而test1文件的所属主是user1，那么user1是不能查看test文件的，相应的user0也不能查看test1文件。有时我们也会有这样的需求，让一个文件同时让user0和user1来查看，这怎么实现呢？

这时 ‘所属组’ 就派上用场了。即，创建一个群组users，让user0和user1同属于users组，然后建立一个文件test2，且其所属组为users，那么user0和user1都可以访问test2文件。Linux文件属性不仅规定了所属主和所属组，还规定了所属主（user）、所属组(group)以及其他用户（others）对该文件的权限。你可以通过ls -l 来查看这些属性。

```
[root@localhost ~]# ls -l
```

```
总用量 40
-rw-------. 1 root root   980 5月   7 18:00 anaconda-ks.cfg
drwxr-xr-x. 3 root root  4096 5月  10 05:10 dirb
```

linux文件属性

上例中，用ls –l 查看当前目录下的文件时，共显示了9列内容（用空格划分列），都代表了什么含义呢？

第1列，包含的东西有该文件类型和所属主、所属组以及其他用户对该文件的权限。第一列共11位有的文件是10位，没有最后面的一位。 其中第一位用来描述该文件的类型。上例中，我们看到的类型有 ‘d’, ‘-‘ ，其实除了这两种外还有 ‘l’, ‘b’, ‘c’, ‘s’ 等。

‘d’ 表示该文件为目录；

‘-‘ 表示该文件为普通文件；

‘l’ 表示该文件为链接文件（linux file），上边提到的软链接即为该类型；

```
[root@localhost ~]# ls -l /etc/rc.local
lrwxrwxrwx. 1 root root 13 5月   7 17:54 /etc/rc.local -> rc.d/rc.local
```

上例中，第一列第一位是 ‘l’ 表示该文件为链接文件，稍后阿铭详细介绍。

‘b’ 表示该文件为块设备，比如 /dev/sda 就是这样的文件。

‘c’ 表示该文件为串行端口设备，例如键盘、鼠标。

‘s’ 表示该文件为套接字文件（socket），用于进程间通信。

后边的9位，每三个为一组。均为rwx 三个参数的组合。其中r 代表可读，w代表可写，x代表可执行。前三位为所属主（user）的权限，中间三位为所属组（group）的权限，最后三位为其他非本群组（others）的权限。下面拿一个具体的例子来述说一下。

一个文件的属性为 ‘-rwxr-xr–.’ ，它代表的意思是，该文件为普通文件，文件拥有者可读可写可执行，文件所属组对其可读不可写可执行，其他用户对其只可读。对于一个目录来讲，打开这个目录即为执行这个目录，所以任何一个目录必须要有x权限才能打开并查看该目录。例如一个目录的属性为 ‘drwxr–r–.’ 其所属主为root，那么除了root外的其他用户是不能打开这个目录的。

关于第一列的最后一位的 `.`，阿铭要说一下，之前的CentOS 5 是没有这个点的，这主要是因为新版本的ls把selinux或者acl的属性加进来了，当文件或者目录只使用了selinux context的属性，这里是一个点。如果设置了acl，后面将是一个加号 ‘+’. 关于selinux 和 acl 阿铭不再详细介绍，你只要了解是怎么回事即可，如果你感兴趣可以网上搜索相关知识也可以通过阿铭在本章后面提供的扩展学习链接进行学习。

第2列，表示为链接占用的节点(inode), [1] 为目录时，通常与该目录底下还有多少目录有关系。

第3列，表示该文件的所属主。

第4列，表示该文件的所属组。

第5列，表示该文件的大小。

第6列、第7列和第8列为该文件的最近的修改日期，分别为月份日期以及时间，也就是所谓的mtime.

第9列，文件名。

更改文件的权限

1. 更改所属组 **chgrp**

语法：`chgrp [组名] [文件名]`

```
[root@localhost ~]# groupadd testgroup
[root@localhost ~]# touch test1
[root@localhost ~]# ls -l test1
-rw-r--r-- 1 root root 0 5月  10 08:41 test1
[root@localhost ~]# chgrp testgroup test1
[root@localhost ~]# ls -l test1
-rw-r--r-- 1 root testgroup 0 5月  10 08:41 test1
```

这里用到了 ‘groupadd’ 命令，其含义为增加一个用户组。该命令在以后章节中做详细介绍，你只要知道它是用来增加用户组的即可。除了更改文件的所属组，还可以更改目录的所属组。

```
[root@localhost ~]# ls -l dirb/
总用量 8
drwxr-xr-x. 2 root root 4096 5月  10 05:08 dirc
-rw-r--r--. 1 root root   20 5月  10 05:37 filee
[root@localhost ~]# ls -ld dirb/
drwxr-xr-x. 3 root root 4096 5月  10 05:10 dirb/
[root@localhost ~]# chgrp testgroup dirb
[root@localhost ~]# ls -ld dirb/
drwxr-xr-x. 3 root testgroup 4096 5月  10 05:10 dirb/
[root@localhost ~]# ls -l dirb/
总用量 8
```

```
drwxr-xr-x. 2 root root 4096 5月  10 05:08 dirc
-rw-r--r--. 1 root root   20 5月  10 05:37 filee
```

‘chgrp’命令也可以更改目录的所属组，但是只能更改目录本身，而目录下面的目录或者文件没有更改，要想级联更改子目录以及子文件，有个选项可以实现：

```
[root@localhost ~]# chgrp -R testgroup dirb
[root@localhost ~]# ls -l dirb
总用量 8
drwxr-xr-x. 2 root testgroup 4096 5月  10 05:08 dirc
-rw-r--r--. 1 root testgroup   20 5月  10 05:37 filee
```

‘chgroup’ 命令阿铭使用的不多，因为还有一个命令可以替代。

2. 更改文件的所属主 **chown**

语法： `chown [ -R ] 账户名 文件名` `chown [ -R ] 账户名:组名 文件名`

这里的-R选项只作用于目录，作用是级联更改，即不仅更改当前目录，连目录里的目录或者文件全部更改。

```
[root@localhost ~]# mkdir test // 创建 'test' 目录
[root@localhost ~]# useradd user1 // 创建用户 'user1', 关于 'useradd' 命令会在后续章节介绍。
[root@localhost ~]# touch test/test2 // 在test目录下创建test2文件
[root@localhost ~]# chown user1 test
[root@localhost ~]# ls -l test
总用量 0
-rw-r--r-- 1 root root 0 5月  10 09:00 test2
[root@localhost ~]# ls -ld test   // test目录所属主已经由 'root' 改为 'user1'.
drwxr-xr-x 2 user1 root 4096 5月  10 09:00 test
[root@localhost ~]# ls -l test  // 但是test目录下的test2文件所属主依旧是 'root'.
总用量 0
-rw-r--r-- 1 root root 0 5月  10 09:00 test2
[root@localhost ~]# chown -R user1:testgroup test
[root@localhost ~]# ls -l test
总用量 0
-rw-r--r-- 1 user1 testgroup 0 5月  10 09:00 test2
```

chown -R user1:testgroup test //把test目录以及目录下的文件都修改成所属主为user1, 所属组为testgroup.

3. 改变用户对文件的读写执行权限 **chmod**

在linux中为了方便更改这些权限，linux使用数字去代替rwx, 具体规则为 ‘r’ 等于4, ‘w’ 等于2, ‘x’ 等于1, ‘-‘ 等于0. 举个例子: ‘-rwxrwx—’ 用数字表示就是 ‘770’, 具体是这样来的: ‘rwx’ = 4+2+1=7; ‘rwx’ = 4+2+1=7; ‘- - -‘ = 0+0+0=0.

`chmod 语法： chmod [-R] xyz 文件名` （这里的xyz，表示数字）

‘-R’ 选项作用同chown，级联更改。

值得提一下的是，在linux系统中，默认一个目录的权限为 755，而一个文件的默认权限为644.

```
[root@localhost ~]# ls -ld test
drwxr-xr-x 2 user1 testgroup 4096 5月  10 09:00 test
[root@localhost ~]# ls -l test
总用量 0
-rw-r--r-- 1 user1 testgroup 0 5月  10 09:00 test2
[root@localhost ~]# chmod 750 test
[root@localhost ~]# ls -ld test
drwxr-x--- 2 user1 testgroup 4096 5月  10 09:00 test
[root@localhost ~]# ls -l test/test2
-rw-r--r-- 1 user1 testgroup 0 5月  10 09:00 test/test2
[root@localhost ~]# chmod 700 test/test2
[root@localhost ~]# chmod -R 700 test
[root@localhost ~]# ls -ld test
drwx------ 2 user1 testgroup 4096 5月  10 09:00 test
[root@localhost ~]# ls -l test
总用量 0
-rwx------ 1 user1 testgroup 0 5月  10 09:00 test2
```

如果你创建了一个目录，而该目录不想让其他人看到内容，则只需设置成 ‘rwxr—-’ (740) 即可。‘chmod’ 还支持使用rwx的方式来设置权限。从之前的介绍中我们可以发现，基本上就九个属性分别是(1)user (2)group (3)others, 我们可以使用u, g, o 来代表它们三个的属性，此外， a 则代表 all 亦即全部。阿铭举例来介绍他们的用法：

```
[root@localhost ~]# chmod u=rwx,og=rx test/test2
[root@localhost ~]# ls -l test/test2
-rwxr-xr-x 1 user1 testgroup 0 5月  10 09:00 test/test2
```

这样可以把 'test/test2' 文件权限修改为 'rwxr-xr-x'. 另外还可以针对u, g, o, a增加或者减少某个权限（读，写，执行），例如：

```
[root@localhost ~]# chmod u-x test/test2
[root@localhost ~]# ls -l test
总用量 0
-rw-r-xr-x 1 user1 testgroup 0 5月  10 09:00 test2
[root@localhost ~]# chmod a-x test/test2
[root@localhost ~]# ls -l test/test2
-rw-r--r-- 1 user1 testgroup 0 5月  10 09:00 test/test2
[root@localhost ~]# chmod u+x test/test2
[root@localhost ~]# ls -l test/test2
-rwxr--r-- 1 user1 testgroup 0 5月  10 09:00 test/test2
```

命令: umask

上边也提到了默认情况下，目录权限值为755, 普通文件权限值为644, 那么这个值是由谁规定呢？追究其原因就涉及到了 'umask'.

umask语法： `umask xxx` （这里的xxx代表三个数字）

查看umask值只要输入 'umask' 然后回车。

```
[root@localhost ~]# umask
0022
```

umask预设是0022，其代表什么含义？先看一下下面的规则：

1）若用户建立为普通文件，则预设 '没有可执行权限', 只有'rw'两个权限。最大为666 ('-rw-rw-rw-').

2）若用户建立为目录，则预设所有权限均开放，即777 ('drwxrwxrwx').

umask数值代表的含义为，上边两条规则中的默认值（文件为666，目录为777）需要减掉的权限。所以目录的权限为 `'rwxrwxrwx' - '----w--w-' = 'rwxr-xr-x'`，普通文件的权限为 `'rw-rw-rw-' - '----w--w-' = 'rw-r--r--'`. umask的值是可以自定义的，比如设定umask 为 002，你再创建目录或者文件时，默认权限分别为 `'rwxrwxrwx' - '-------w-' = 'rwxrwxr-x'` 和 `'rw-rw-rw-' - '-------w-' = 'rw-rw-r--'`.

```
[root@localhost ~]# umask 002
[root@localhost ~]# mkdir test2
[root@localhost ~]# ls -ld test2
drwxrwxr-x 2 root root 4096 5月  10 09:44 test2
[root@localhost ~]# touch test3
[root@localhost ~]# ls -l test3
-rw-rw-r-- 1 root root 0 5月  10 09:45 test3
```

可以看到创建的目录权限默认变为775, 而文件默认权限变为664. 然后再把umask改回来。

```
[root@localhost ~]# umask 022
[root@localhost ~]# touch test4
[root@localhost ~]# ls -l test4
-rw-r--r-- 1 root root 0 5月  10 09:45 test4
```

umask 可以在 `/etc/bashrc` 里面更改，预设情况下，root的umask为022，而一般使用者则为002，因为可写的权限非常重要，因此预设会去掉写权限。

4. 修改文件的特殊属性

命令 : chattr

`语法： chattr [+-=][ASaci [文件或者目录名]`

'+-=' : 分别为增加、减少、设定

'A' : 增加该属性后，文件或目录的atime将不可被修改；

'S' : 增加该属性后，会将数据同步写入磁盘中；

'a' : 增加该属性后，只能追加不能删除，非root用户不能设定该属性；

'c' : 自动压缩该文件，读取时会自动解压；

'i' : 增加后，使文件不能被删除、重命名、设定链接接、写入、新增数据；

```
[root@localhost ~]# chattr +i test2
[root@localhost ~]# touch test2/test1
touch: 无法创建'test2/test1': 权限不够
[root@localhost ~]# chattr -i test2
[root@localhost ~]# touch test2/test1
[root@localhost ~]# chattr +i test2
[root@localhost ~]# rm -f test2/test1
rm: 无法删除'test2/test1': 权限不够
```

对 'test2' 目录增加 'i' 权限后，即使是root账户也不能在 'test2' 里创建或删除test1文件。

```
[root@localhost ~]# chattr -i test2
[root@localhost ~]# touch test2/test3
[root@localhost ~]# ls test2
test1  test3
[root@localhost ~]# chattr +a test2
[root@localhost ~]# rm -f test2/test1
rm: 无法删除 'test2/test1': 不允许的操作
[root@localhost ~]# touch test2/test4
[root@localhost ~]# ls test2
test1  test3  test4
```

test2目录增加 ‘a’ 权限后，只可以在里面创建文件，而不能删除文件。文件同样可以适用这些权限。

```
[root@localhost ~]# chattr +a test2/test1
[root@localhost ~]# echo '11111' > test2/test1
-bash: test2/test1: 不允许的操作
[root@localhost ~]# echo '11111' >> test2/test1
[root@localhost ~]# cat test2/test1
11111
[root@localhost ~]# chattr +i test2/test3
[root@localhost ~]# echo '11111' >> test2/test3
-bash: test2/test3: 权限不够
[root@localhost ~]# echo '11111' > test2/test3
-bash: test2/test3: 权限不够
[root@localhost ~]# rm -f test2/test3
rm: 无法删除'test2/test3': 权限不够
```

命令：lsattr

该命令用来读取文件或者目录的特殊权限，语法为 `lsattr  [-aR] [文件/目录名]`

‘-a’：类似与ls 的-a 选项，即连同隐藏文件一同列出；

‘-R’：连同子目录的数据一同列出

```
[root@localhost ~]# lsattr test2
-----a-------e- test2/test1
----i--------e- test2/test3
-------------e- test2/test4
[root@localhost ~]# lsattr -aR test2
----i--------e- test2/.
-----a-------e- test2/test1
-------------e- test2/..
----i--------e- test2/test3
-------------e- test2/test4
```

在linux下搜一个文件

在windows下有一个搜索工具，可以让我们很快的找到一个文件，这是很有用的。然而在linux下搜索功能更加强大。

1. ‘which’ 用来查找可执行文件的绝对路径。

在前面已经用到该命令，需要注意的一点是，which只能用来查找PATH环境变量中出现的路径下的可执行文件。这个命令用的也是蛮多的，有时候我们不知道某个命令的绝对路径，which 一下很容易就知道了。

2. ‘whereis’ 通过预先生成的一个文件列表库去查找跟给出的文件名相关的文件。

`语法：  whereis [-bmsu]  [文件名称]`

‘-b’：只找binary 文件

‘-m’：只找在说明文件manual路径下的文件

‘-s’：只找source来源文件

‘-u’：没有说明档的文件

说明：whereis 阿铭几乎很少用到，如果你感兴趣请深入研究。

3. ‘locate’ 类似于‘whereis’, 也是通过查找预先生成的文件列表库来告诉用户要查找的文件在哪里。

后边直接跟文件名。如果你的linux没有这个命令，请安装软件包 ‘mlocate’, 这个软件包在你的系统安装盘里，后缀名是RPM，随后介绍的find命令会告诉你如何查找这个包。如果你装的CentOS你可以使用这个命令来安装 `yum install -y  mlocate` 前提是你的CentOS能连网。至于yum这个命令如何使用，到后续章节你自然会明白。如果你刚装上这个命令，初次使用会报错。

```
[root@localhost ~] # locate passwd
```

```
locate: can not open `/var/lib/mlocate/mlocate.db': No such file or directory
```

这是因为系统还没有生成那个文件列表库。你可以使用 `updatedb` 命令立即生成（更新）这个库。如果你的服务器上正跑着重要的业务，那么你最好不要去运行这个命令，因为一旦运行，服务器的压力会变大。这个数据库默认情况下每周更新一次。所以你用locate命令去搜索一个文件，正好是在两次更新时间段内，那你肯定是得不到结果的。你可以到/etc/updated.conf 去配置这个数据库生成（更新）的规则。'locate'所搜索到的文件列表，不管是目录名还是文件名，只要包含我们要搜索的关键词，都会列出来，所以'locate'不适合精准搜索，这个命令阿铭使用的也并不多，你只要明白有这么一个工具即可，用到时再去深究其用法吧。

4. 'find' 这个搜索工具是阿铭用的最多的一个，所以请你务必要熟悉它。

语法：`find [路径] [参数]` 下面介绍几个阿铭经常用的参数

'-atime +n/-n'：访问或执行时间大于/小于n天的文件

'-ctime +n/-n'：写入、更改inode属性（例如更改所有者、权限或者链接）时间大于/小于n天的文件

'-mtime +n/-n'：写入时间大于/小于n天的文件

```
[root@localhost ~]# find /tmp/ -mtime -1
/tmp/
/tmp/.ICE-unix
/tmp/test
[root@localhost ~]# find /tmp/ -atime +10
[root@localhost ~]# find /tmp/ -atime +1
/tmp/yum.log
/tmp/.bash_history
```

看到这里，你对这三个time是不是有些晕了，那阿铭就先给你介绍一下这三个time属性。

文件的 Access time也就是 'atime' 是在读取文件或者执行文件时更改的。文件的 Modified time也就是 'mtime' 是在写入文件时随文件内容的更改而更改的。文件的 Change time也就是 'ctime' 是在写入文件、更改所有者、权限或链接设置时随inode的内容更改而更改的。 因此，更改文件的内容即会更改mtime和ctime，但是文件的ctime可能会在 mtime 未发生任何变化时更改，例如，更改了文件的权限，但是文件内容没有变化。 如何获得一个文件的atime mtime 以及ctime ？

'stat' 命令可用来列出文件的 atime、ctime 和 mtime。

```
[root@localhost ~]# stat test/test2
  File: 'test/test2'
  Size: 0               Blocks: 0          IO Block: 4096   普通空文件
Device: 803h/2051d      Inode: 261657      Links: 1
Access: (0744/-rwxr--r--)  Uid: (  500/   user1)   Gid: (  500/testgroup)
Access: 2013-05-10 09:00:36.092000531 +0800
Modify: 2013-05-10 09:00:36.092000531 +0800
Change: 2013-05-10 09:30:58.788996594 +0800
```

atime不一定在访问文件之后被修改，因为：使用ext3文件系统的时候，如果在mount的时候使用了noatime参数那么就不会更新atime的信息。总之，這三個 time stamp 都放在 inode 中。若 mtime, atime 修改inode 就一定會改, 既然 inode 改了, 那 ctime 也就跟着要改了。

阿铭继续'find'常用选项：

'-name filename' 直接查找该文件名的文件，这个选项使用很多。

```
[root@localhost ~]# find . -name test2
./test/test2
./test2
```

'-type filetype' 通过文件类型查找。文件类型在前面部分已经简单介绍过，相信你已经大体上了解了。filetype 包含了 f, b, c, d, l, s 等。

```
[root@localhost ~]# find /tmp/ -type d
/tmp/
/tmp/.ICE-unix
/tmp/test
[root@localhost ~]# find /tmp/ -type f
/tmp/yum.log
/tmp/.bash_history
/tmp/ip.txt
```

linux的文件系统简介

搞计算机的应该都知道windows的系统格式化硬盘时会指定格式，fat 或者 ntfs。而linux的文件系统格式为Ext3，或者Ext4 。早期的linux使用Ext2格式，目前的linux都使用了Ext3， 而CentOS6已经使用了Ext4. Ext2文件系统虽然是高效稳定的。但是，随着Linux系统在关键业务中的应用，Linux文件系统的弱点也渐渐显露出来了，因为Ext2文件系统是非日志文件系统。这在关键行业的应用是一个致命的弱点。Ext3文件系统是直接从Ext2文件系统发展而来，Ext3文件系统带有日志功能，可以跟踪记录文件系统的变化，并将变化内容写入日志，写操作首先是对日志记录文件进行操作，若整个写操作由于某种原因 (如系统掉电) 而中断，系

统重启时，会根据日志记录来恢复中断前的写操作，而且这个过程费时极短。目前Ext3文件系统已经非常稳定可靠。它完全兼容Ext2文件系统。用户可以平滑地过渡到一个日志功能健全的文件系统中来。这实际上了也是ext3日志文件系统初始设计的初衷。

而现在我们使用的Ext4文件系统，较Ext3文件系统又有很多好的特性，最明显的特征是，Ext4支持的最大文件系统容量和单个最大文件大小，详细的区别阿铭不再介绍，这需要你到网上搜一下以丰富你的知识点。

Linux文件系统在windows中是不能识别的，但是在linux系统中你可以挂载的windows的文件系统，linux目前支持MS-DOS，VFAT，FAT，BSD等格式，如果你使用的是Redhat或者CentOS，那么请不要妄图挂载NTFS格式的分区到linux下，因为它不支持NTFS。虽然有些版本的linux支持NTFS，但不建议使用，因为目前的技术还不成熟。但有时，的确会有这方面的需求，你可以安装ntfs-3g软件包来解决这个问题。

Ext3文件系统为早期版本Redhat/CentOS默认使用的文件系统，目前Ext4为默认文件系统，除了Ext3/Ext4文件系统外，有些linux发行版例如SuSE默认的文件系统为reiserFS，Ext3 独特的优点就是易于转换，很容易在 Ext2 和 Ext3 之间相互转换，而具有良好的兼容性，其它优点 ReiserFS 都有，而且还比它做得更好。如高效的磁盘空间利用和独特的搜寻方式都是Ext3 所不具备的，速度上它也不能和 ReiserFS 相媲美，在实际使用过程中，reiserFS 也更加安全高效，据说反删除功能也不错。

ReiserFS 的优势在于，它是基于 B*Tree 快速平衡树这种高效算法的文件系统，例如在处理小于 1k 的文件比 Ext文件系统快10倍。再一个就是ReiserFS空间浪费较少，它不会对一些小文件分配 inode，而是打包存放在同一个磁盘块 (簇) 中，Ext是把它们单独存放在不同的簇上，如簇大小为 4k，那么 2 个 100 字节的文件会占用 2 个簇，ReiserFS则只占用一个。当然ReiserFS也有缺点，就是每升级一个版本，都要将磁盘重新格式化一次。

linux文件类型

在前面的内容中简单介绍了普通文件 '-'，目录 'd' 等，在linux文件系统中，主要有以下几种类型的文件。

1）普通文件(regular file)：就是一般类型的文件，当用 `ls -l` 查看某个目录时，第一个属性为 '-' 的文件就是正规文件，或者叫普通文件。正规文件又可分成纯文字文件(ascii)和二进制文件(binary)。纯文本文件是可以通过cat，more，less等工具直接查看内容的，而二进制文件并不能。例如我们用的命令/bin/ls 这就是一个二进制文件。

2）目录(directory)：这个很容易理解，就是目录，跟windows下的文件夹一个意思，只不过在linux中我们不叫文件夹，而是叫做目录。`ls -l` 查看第一个属性为 'd'.

3）链接文件(link)：`ls -l` 查看第一个属性为 'l'，类似windows下的快捷方式。这种文件在linux中很常见，而且阿铭在日常的系统运维工作中用的很多，所以你要特意留意一下这种类型的文件。在后续章节阿铭会介绍。

4）设备(device)：与系统周边相关的一些档案，通常都集中在 /dev 这个目录之下！通常又分为两种：块(block)设备：就是一些储存数据，以提供系统存取的接口设备，简单的说就是硬盘。例如你的一号硬盘的代码是 /dev/sda1，第一个属性为 'b'；另一种是字符(character)设备：是一些串行端口的接口设备，例如键盘、鼠标等等，第一个属性为 'c'.

Linux文件后缀名

对于后缀名这个概念，相信你不陌生吧。在linux系统中，文件的后缀名并没有具体意义，也就是说，你加或者不加，都无所谓。但是为了容易区分，我们习惯给文件加一个后缀名，这样当用户看到这个文件名时就会很快想到它到底是一个什么文件。例如1.sh，2.tar.gz，my.cnf，test.zip等等，如果你首次接触这些文件，你也许会感到很晕，没有关系，随着学习的深入，你就会逐渐的了解这些文件了。阿铭所列举的几个文件名中1.sh代表它是一个shell script，2.tar.gz 代表它是一个压缩包，my.cnf 代表它是一个配置文件，test.zip 代表它是一个压缩文件。

另外需要你还需知道的是，早期Unix系统文件名最多允许14个字符，而新的Unix或者linux系统中，文件名最长可以到达 256 个字符！

Linux的链接文件

上面阿铭多次提到链接文件的概念，相信你早就想明白到底什么是链接文件。下面阿铭就来介绍一下这个链接文件的知识点。链接文件分为两种，硬链接(hard link)和软链接(symbolic link)。两种链接的本质区别关键点在于inode.

Hard Links：当系统要读取一个文件时，就会先去读inode table，然后再去根据inode中的信息到块区域去将数据取出来。而hard link 是直接再建立一个inode链接到文件放置的块区域。也就是说，进行hard link的时候实际上该文件内容没有任何变化，只是增加了一个指到这个文件的inode，hard link 有两个限制：(1)不能跨文件系统，因为不同的文件系统有不同的inode table；(2) 不能链接目录。

Symbolic Links：跟hard link不同，这个是建立一个独立的文件，而这个文件的作用是当读取这个链接文件时，它会把读取的行为转发到该文件所link的文件上。这样讲，也许比较绕口，那么就来举一个例子。现在有文件a，我们做了一个软链接文件b（只是一个链接文件，非常小），b指向了文件a。当读取b时，那么b就会把读取的动作转发到a上，这样就读取到了文件a。所以，当你删除文件a时，文件b并不会被删除，但是再读取b时，会提示无法打开文件。而，当你删除b时，a是不会有任何影响的。

看样子，似乎 hard link 比较安全，因为即使某一个 inode 被删掉了，只要有任何一个 inode 存在，那么该文件就不会消失不见！不过，不幸的是，由于 Hard Link 的限制太多了，包括无法做目录的link，所以在用途上面是比较受限的！反而是 Symbolic Link 的使用方向较广！那么如何建立软链接和硬链接呢？这就用到了ln 命令。

命令: ln

语法：`ln [-s] [来源文件] [目的文件]`

ln 常用的选项就一个 '-s', 如果不加就是建立硬链接，加上就建立软链接。

```
[root@localhost ~]# mkdir 123
[root@localhost ~]# cd 123
[root@localhost 123]# cp /etc/passwd ./
[root@localhost 123]# ll
总用量 4
-rw-r--r-- 1 root root 1097 5月  10 17:08 passwd
[root@localhost 123]# du -sk
8       .
[root@localhost 123]# ln passwd passwd-hard
[root@localhost 123]# ll
总用量 8
-rw-r--r-- 2 root root 1097 5月  10 17:08 passwd
-rw-r--r-- 2 root root 1097 5月  10 17:08 passwd-hard
[root@localhost 123]# du -sk
8       .
```

上例中的 'll' 命令等同于 'ls -l', 请使用 'which' 命令查看一下。做了硬链接后，虽然两个文件大小都为 '1097', 但是目录的大小并没有变化。

```
[root@localhost 123]# ll
总用量 4
-rw-r--r-- 1 root root 1097 5月  10 17:08 passwd-hard
[root@localhost 123]# rm -f passwd
[root@localhost 123]# du -sk
8       .
```

删除源文件passwd, 空间依旧不变。这说明硬链接只是复制了一份inode信息。

```
[root@localhost ~]# ln 123 456
ln: "123": 不允许将硬链接指向目录
```

硬链接不能用于目录。

```
[root@localhost ~]# mkdir 456
[root@localhost ~]# cd 456
[root@localhost 456]# cp /etc/passwd ./
[root@localhost 456]# ln -s passwd  passwd-soft
[root@localhost 456]# ll
总用量 4
-rw-r--r-- 1 root root 1097 5月  10 17:18 passwd
lrwxrwxrwx 1 root root    6 5月  10 17:19 passwd-soft -> passwd

[root@localhost 456]# head -n1 passwd-soft
root:x:0:0:root:/root:/bin/bash
[root@localhost 456]# head -n1 passwd
root:x:0:0:root:/root:/bin/bash
[root@localhost 456]# rm -f passwd
[root@localhost 456]# head -n1 passwd-soft
head: 无法打开"passwd-soft" 读取数据: 没有那个文件或目录
[root@localhost 456]# ll
总用量 0
lrwxrwxrwx 1 root root 6 5月  10 17:19 passwd-soft -> passwd
```

如果删除掉源文件，则软链接文件不能读取了，而且使用 'll' 查看发现颜色也变了。

```
[root@localhost ~]# ln -s 456 789
[root@localhost ~]# ls -ld 456 789
drwxr-xr-x 2 root root 4096 5月  10 17:22 456
lrwxrwxrwx 1 root root    3 5月  10 17:29 789 -> 456
```

目录是可以软链接的。

知识面扩展学习：http://www.aminglinux.com/bbs/thread-5406-1-1.html

教程答疑: 请移步这里.

欢迎你加入 阿铭学院 和阿铭一起学习Linux，让阿铭成为你Linux生涯中永远的朋友吧！

[1]inode 译成中文就是索引节点，它用来存放档案及目录的基本信息，包含时间信息、文档名、属主以及属组等。Inode是Unix操作系统中的一种数据结构，本质是结构体，inode是随文件系统创建时生成的，它的个数有限。在Linux下，可以通过 `df -i` 来查看各个分区的inode数量。