setfacl

设置文件访问控制列表

补充说明

setfacl命令 是用来在命令行里设置ACL（访问控制列表）。在命令行里，一系列的命令跟随以一系列的文件名。

选项

```
-b,--remove-all：删除所有扩展的acl规则，基本的acl规则(所有者，群组，其他）将被保留。
-k,--remove-default：删除缺省的acl规则。如果没有缺省规则，将不提示。
-n：--no-mask：不要重新计算有效权限，setfacl默认会重新计算ACL mask，除非mask被明确的制定。
--mask：重新计算有效权限，即使ACL mask被明确指定。
-d：--default：设定默认的acl规则。
--restore=file：从文件恢复备份的acl规则（这些文件可由getfacl -R产生）。通过这种机制可以恢复整个目录树的acl规则。此参数不能和除--test以外的任何参数一同执行。
--test：测试模式，不会改变任何文件的acl规则，操作后的acl规格将被列出。
-R：--recursive：递归的对所有文件及目录进行操作。
-L：--logical：跟踪符号链接，默认情况下只跟踪符号链接文件，跳过符号链接目录。
-P：--physical：跳过所有符号链接，包括符号链接文件。
--version：输出setfacl的版本号并退出。
--help：输出帮助信息。
--：标识命令行参数结束，其后的所有参数都将被认为是文件名
-：如果文件名是-，则setfacl将从标准输入读取文件名。
```

- 选项-m和-x后边跟以acl规则。多条acl规则以逗号(,)隔开。选项-M和-X用来从文件或标准输入读取acl规则。
- 选项--set和--set-file用来设置文件或目录的acl规则，先前的设定将被覆盖。
- 选项-m(--modify)和-M(--modify-file)选项修改文件或目录的acl规则。
- 选项-x(--remove)和-X(--remove-file)选项删除acl规则。

当使用-M、-X选项从文件中读取规则时，setfacl接受getfacl命令输出的格式。每行至少一条规则，以#开始的行将被视为注释。

当在不支持ACLs的文件系统上使用setfacl命令时，setfacl将修改文件权限位。如果acl规则并不完全匹配文件权限位，setfacl将会修改文件权限位使其尽可能的反应acl规则，并会向standard error发送错误消息，以大于0的状态返回。

权限

文件的所有者以及有CAP_FOWNER的用户进程可以设置一个文件的acl（在目前的linux系统上，root用户是唯一有CAP_FOWNER能力的用户）

ACL规则

setfacl命令可以识别以下的规则格式：

```
[d[efault]:] [u[ser]:]uid [:perms]   指定用户的权限，文件所有者的权限（如果uid没有指定）。
[d[efault]:] g[roup]:gid [:perms]    指定群组的权限，文件所有群组的权限（如果gid未指
```

```
定）
[d[efault]:] m[ask][:] [:perms]      有效权限掩码
[d[efault]:] o[ther] [:perms]        其他的权限
```

恰当的acl规则被用在修改和设定的操作中，对于uid和gid可以指定一个数字，也可指定一个名字。perms域是一个代表各种权限的字母的组合：读-r写-w执行-x，执行只适合目录和一些可执行的文件。pers域也可设置为八进制格式。

自动创建的规则

最初的，文件目录仅包含3个基本的acl规则。为了使规则能正常执行，需要满足以下规则。

- 3个基本规则不能被删除。
- 任何一条包含指定的用户名或群组名的规则必须包含有效的权限组合。
- 任何一条包含缺省规则的规则在使用时，缺省规则必须存在。

ACL的名词定义

先来看看在ACL里面每一个名词的定义，这些名词我大多从man page上摘下来虽然有些枯燥,但是对于理解下面的内容还是很有帮助的。

ACL是由一系列的Access Entry所组成的，每一条Access Entry定义了特定的类别可以对文件拥有的操作权限。Access Entry有三个组成部分。Entry tag type, qualifier (optional), permission。

我们先来看一下最重要的Entry tag type。它有以下几个类型：

```
ACL_USER_OBJ：相当于Linux里file_owner的permission
ACL_USER：定义了额外的用户可以对此文件拥有的permission
ACL_GROUP_OBJ：相当于Linux里group的permission
ACL_GROUP：定义了额外的组可以对此文件拥有的permission
ACL_MASK：定义了ACL_USER, ACL_GROUP_OBJ和ACL_GROUP的最大权限 (这个我下面还会专门讨论)
ACL_OTHER：相当于Linux里other的permission
```

让我们来据个例子说明一下，下面我们就用getfacl命令来查看一个定义好了的ACL文件：

```
[root@localhost ~]# getfacl ./test.txt
# file: test.txt
# owner: root
# group: admin
user::rw-
user:john:rw-
group::rw-
group:dev:r--
mask::rw- other::r--
```

前面三个以#开头的定义了文件名，file owner和group这些信息没有太大的作用，接下来我们可以用--omit-header来省略掉。

```
user::rw-        定义了ACL_USER_OBJ, 说明file owner拥有read and write
permission
user:john:rw-    定义了ACL_USER,这样用户john就拥有了对文件的读写权限,实现了我们一开始要
达到的目的
```

```
group::rw-      定义了ACL_GROUP_OBJ,说明文件的group拥有read and write
permission
group:dev:r--   定义了ACL_GROUP,使得dev组拥有了对文件的read permission
mask::rw-       定义了ACL_MASK的权限为read and write
other::r--      定义了ACL_OTHER的权限为read
```

从这里我们就可以看出ACL提供了我们可以定义特定用户和用户组的功能，那么接下来我们就来看一下如何设置一个文件的ACL

如何设置ACL文件

首先我们还是要讲一下设置ACL文件的格式，从上面的例子中我们可以看到每一个Access Entry都是由三个被：号分隔开的字段所组成，第一个就是Entry tag type

```
user   对应了ACL_USER_OBJ和ACL_USER
group  对应了ACL_GROUP_OBJ和ACL_GROUP
mask   对应了ACL_MASK
other  对应了ACL_OTHER
```

第二个字段称之为qualifier也就是上面例子中的john和dev组，它定义了特定用户和拥护组对于文件的权限。这里我们也可以发现只有user和group才有qualifier其他的都为空。第三个字段就是我们熟悉的permission了。它和Linux的permission一样定义，这里就不多讲了。

下面我们就来看一下怎么设置test.txt这个文件的ACL让它来达到我们上面的要求。

一开始文件没有ACL的额外属性：

```
[root@localhost ~]# ls -l
-rw-rw-r-- 1 root admin 0 Jul 3 22:06 test.txt

[root@localhost ~]# getfacl --omit-header ./test.txt
user::rw- group::rw- other::r--
```

我们先让用户john拥有对test.txt文件的读写权限：

```
[root@localhost ~]# setfacl -m user:john:rw- ./test.txt
[root@localhost ~]# getfacl --omit-header ./test.txt
user::rw-
user:john:rw-
group::rw-
mask::rw-
other::r--
```

这时我们就可以看到john用户在ACL里面已经拥有了对文件的读写权。这个时候如果我们查看一下linux的permission我们还会发现一个不一样的地方。

```
[root@localhost ~]# ls -l ./test.txt
-rw-rw-r--+ 1 root admin 0 Jul 3 22:06 ./test.txt
```

在文件permission的最后多了一个+号，当任何一个文件拥有了ACL_USER或者ACL_GROUP的值以后我们就可以称它为ACL文件，这个+号就是用来提示我们的。我们还可以发现当一个文件拥有了ACL_USER或

者ACL_GROUP的值时ACL_MASK同时也会被定义。

接下来我们来设置dev组拥有read permission

```
[root@localhost ~]# setfacl -m group:dev:r-- ./test.txt
[root@localhost ~]# getfacl --omit-header ./test.txt
user::rw-
user:john:rw-
group::rw-
group:dev:r--
mask::rw-
other::r--
```

到这里就完成了我们上面讲到的要求，是不是很简单呢。

ACL_MASK和Effective permission

这里需要重点讲一下ACL_MASK，因为这是掌握ACL的另一个关键，在Linux file permission里面大家都知道比如对于rw-rw-r--来说, 当中的那个rw-是指文件组的permission. 但是在ACL里面这种情况只是在ACL_MASK不存在的情况下成立。如果文件有ACL_MASK值，那么当中那个rw-代表的就是mask值而不再是group permission了。

让我们来看下面这个例子：

```
[root@localhost ~]# ls -l
-rwxrw-r-- 1 root admin 0 Jul 3 23:10 test.sh
```

这里说明test.sh文件只有file owner: root拥有read, write, execute/search permission，admin组只有read and write permission，现在我们想让用户john也对test.sh具有和root一样的permission

```
[root@localhost ~]# setfacl -m user:john:rwx ./test.sh
[root@localhost ~]# getfacl --omit-header ./test.sh
user::rwx user:john:rwx
group::rw-
mask::rwx
other::r--
```

这里我们看到john已经拥有了rwx的permission，mask值也被设定为rwx，那是因为它规定了ACL_USER，ACL_GROUP和ACL_GROUP_OBJ的最大值，现在我们再来看test.sh的Linux permission，它已经变成了：

```
[root@localhost ~]# ls -l
-rwxrwxr--+ 1 root admin 0 Jul 3 23:10 test.sh
```

那么如果现在admin组的用户想要执行test.sh的程序会发生什么情况呢？它会被permission deny，原因在于实际上admin组的用户只有read and write permission，这里当中显示的rwx是ACL_MASK的值而不是group的permission

所以从这里我们就可以知道，如果一个文件后面有+标记，我们都需要用getfacl来确认它的permission，以免发生混淆。

下面我们再来继续看一个例子，假如现在我们设置test.sh的mask为read only，那么admin组的用户还会

有write permission吗？

```
[root@localhost ~]# setfacl -m mask::r-- ./test.sh
[root@localhost ~]# getfacl --omit-header ./test.sh
user::rwx
user:john:rwx   #effective:r--
group::rw-      #effective:r--
mask::r--
other::r--
```

这时候我们可以看到ACL_USER和ACL_GROUP_OBJ旁边多了个#effective:r--这是什么意思呢？让我们再来回顾一下ACL_MASK的定义。它规定了ACL_USER、ACL_GROUP_OBJ和ACL_GROUP的最大权限。那么在我们这个例子中他们的最大权限也就是read only虽然我们这里给ACL_USER和ACL_GROUP_OBJ设置了其他权限，但是他们真正有效果的只有read权限。

这时我们再来查看test.sh的Linux file permission时它的group permission也会显示其mask的值(i.e. r--)

```
[root@localhost ~]# ls -l
-rwxr--r--+ 1 root admin 0 Jul 3 23:10 test.sh
```

Default ACL

上面我们所有讲的都是Access ACL也就是对文件而言。下面我简单讲一下Default ACLDefault ACL是指对于一个目录进行Default ACL设置，并且在此目录下建立的文件都将继承此目录的ACL

同样我们来做一个试验说明，比如现在root用户建立了一个dir目录：

```
[root@localhost ~]# mkdir dir
```

他希望所有在此目录下建立的文件都可以被john用户所访问，那么我们就应该对dir目录设置Default ACL

```
[root@localhost ~]# setfacl -d -m user:john:rw ./dir
[root@localhost ~]# getfacl --omit-header ./dir
user::rwx
group::rwx
other::r-x
default:user::rwx
default:user:john:rwx
default:group::rwx
default:mask::rwx
default: other::r-x
```

这里我们可以看到ACL定义了default选项john用户拥有了default的read, write, excute/search permission所有没有定义的default都将从file permission里copy过来，现在root用户在dir下建立一个test.txt文件。

```
[root@localhost ~]# touch ./dir/test.txt
[root@localhost ~]# ls -l ./dir/test.txt
-rw-rw-r--+ 1 root root 0 Jul 3 23:46 ./dir/test.txt

[root@localhost ~]# getfacl --omit-header ./dir/test.txt
```

```
user::rw-
user:john:rw-
group::rwx #effective:rw-
mask::rw-
other::r--
```

这里我们看到在dir下建立的文件john用户自动就有了read and write permission

ACL相关命令

前面的例子中我们都注意到了getfacl命令是用来读取文件的ACL，setfacl是用来设定文件的Acess ACL，这里还有一个chacl是用来改变文件和目录的Access ACL and Default ACL，它的具体参数大家可以去看man page，我只想提及一下`chacl -B`。它可以彻底删除文件或者目录的ACL属性(包括Default ACL)，比如你即使用了`setfacl -x`删除了所有文件的ACL属性，那个+号还是会出现在文件的末尾，所以正确的删除方法应该是用`chacl -B`用cp来复制文件的时候我们现在可以加上`-p`选项。这样在拷贝文件的时候也将拷贝文件的ACL属性，对于不能拷贝的ACL属性将给出警告。

mv命令将会默认地移动文件的ACL属性，同样如果操作不允许的情况下会给出警告。

需要注意的几点

如果你的文件系统不支持ACL的话，你也许需要重新mount你的file system

```
mount -o remount, acl [mount point]
```

如果用chmod命令改变Linux file permission的时候相应的ACL值也会改变，反之改变ACL的值，相应的file permission也会改变。