HowTo ASIX file ACLs

Documentació

Aquest document ha estat elaborant utilitzant com a eina de treball un sistema GNU/Linux Fedora 20.

- Documentació de les pàgines man de les ordres.
- Fedora Documentation: Fedora 14, Storage Administration Guide, Chapter 15: Access Control Lists
- Access Control Lists in Linux (SUSE) By Andreas Gruenbacher

$ apropos acl

```
acl (5)          - Access Control Lists
chacl (1)        - change the access control list of a file or directory
getfacl (1)      - get file access control lists
setfacl (1)      - set file access control lists
sharesec (1)     - Set or get share ACLs
slapacl (8)      - Check access to a list of attributes.
```

```
.k5login (5)         - Kerberos V5 acl file for host access.
k5login (5)          - Kerberos V5 acl file for host access.

getcifsacl (1)       - Userspace helper to display an ACL in a security descriptor for
    Common Internet File System (CIFS)
setcifsacl (1)       - Userspace helper to alter an ACL in a security descriptor for
    Common Internet File System (CIFS)
smbcacls (1)         - Set or get ACLs on an NT file or directory names
```

Ordres de gestió de les ACLs de fitxers:

```
getfacl   setfacl  stat  chmod  umask
```

File Access Control Lists

Definició de wikipèdia de “Access control List: Filesystem ACLs”:

Filesystem ACLs

In the 1990s the ACL and RBAC models were extensively tested and used to administrate file permissions. A filesystem ACL is a data structure (usually a table) containing entries that specify individual user or group rights to specific system objects such as programs, processes, or files. These entries are known as access control entries (ACEs) in the Microsoft Windows NT,[2] OpenVMS, Unix-like, and Mac OS X operating systems. Each accessible object contains an identifier to its ACL. The privileges or permissions determine specific access rights, such as whether a user can read from, write to, or execute an object. In some implementations, an ACE can control whether or not a user, or group of users, may alter the ACL on an object.
Most of the Unix and Unix-like operating systems (e.g. Linux,[3] BSD, or Solaris) support POSIX.1e ACLs, based on an early POSIX draft that was abandoned. Many of them, for example AIX, FreeBSD,[4] Mac OS X beginning with version 10.4 ("Tiger"), or Solaris with ZFS filesystem,[5] support NFSv4 ACLs, which are part of the NFSv4 standard. There are two experimental implementations of NFSv4 ACLs for Linux: NFSv4 ACLs support for Ext3 filesystem[6] and recent Richacls,[7] which brings NFSv4 ACLs support for Ext4 filesystem.

Contingut del paquet RPM **acl**:

```
$ rpm -ql acl
/usr/bin/chacl
/usr/bin/getfacl
/usr/bin/setfacl
/usr/share/doc/acl-2.2.51
```

```
/usr/share/doc/acl-2.2.51/CHANGES.gz
/usr/share/doc/acl-2.2.51/COPYING
/usr/share/doc/acl-2.2.51/COPYING.LGPL
/usr/share/doc/acl-2.2.51/PORTING
/usr/share/doc/acl-2.2.51/README
/usr/share/locale/de/LC_MESSAGES/acl.mo
/usr/share/locale/es/LC_MESSAGES/acl.mo
/usr/share/locale/fr/LC_MESSAGES/acl.mo
/usr/share/locale/gl/LC_MESSAGES/acl.mo
/usr/share/locale/pl/LC_MESSAGES/acl.mo
/usr/share/locale/sv/LC_MESSAGES/acl.mo
/usr/share/man/man1/chacl.1.gz
/usr/share/man/man1/getfacl.1.gz
/usr/share/man/man1/setfacl.1.gz
/usr/share/man/man5/acl.5.gz
```

Definitions

Extret del document suse_acces_control_lists_in_linux

User class The conventional POSIX permission concept uses three classes of users for assigning permissions in the file system: the owner, the owning group, and other users. Three permission bits can be set for each user class, giving permission to read (r), write (w), and execute (x). An introduction to the user concept in Linux is provided in the User Guide in the section Users and Access Permissions.

Access ACL The user and group access permissions for all kinds of file system objects (files and directories) are determined by means of access ACLs.

Default ACL Default ACLs can only be applied to directories. They determine the permissions a file system object inherits from its parent directory when it is created.

ACL entry Each ACL consists of a set of ACL entries. An ACL entry contains a type (see Table B.1 on the following page), a qualifier for the user or group to which the entry refers, and a set of permissions. For some entry types, the qualifier for the group or users is undefined.

Extret del man acl:

> An ACL consists of a set of ACL entries. An ACL entry specifies the access permissions on the associated object for an individual user or a group of users as a combination of read, write and search/execute permissions.
>
> An ACL entry contains an entry tag type, an optional entry tag qualifier, and a set of permissions. We use the term qualifier to denote the entry tag qualifier of an ACL entry.

The qualifier denotes the identifier of a user or a group, for entries with tag types of ACL_USER or ACL_GROUP, respectively. Entries with tag types other than ACL_USER or ACL_GROUP have no defined qualifiers.

The following entry tag types are defined:

ACL_USER_OBJ
The ACL_USER_OBJ entry denotes access rights for the file owner.

ACL_USER
ACL_USER entries denote access rights for users identified by the entry's qualifier.

ACL_GROUP_OBJ
The ACL_GROUP_OBJ entry denotes access rights for the file group.

ACL_GROUP
ACL_GROUP entries denote access rights for groups identified by the entry's qualifier.

ACL_MASK
The ACL_MASK entry denotes the maximum access rights that can be granted by entries of type ACL_USER, ACL_GROUP_OBJ, or ACL_GROUP.

ACL_OTHER
The ACL_OTHER entry denotes access rights for processes that do not match any other entry in the ACL.

When an access check is performed, the ACL_USER_OBJ and ACL_USER entries are tested against the effective user ID. The effective group ID, as well as all supplementary group IDs are tested against the ACL_GROUP_OBJ and ACL_GROUP entries.

Exemple

```
$ cd /opt/m11/
$ mkdir acldir
$ chmod 750 acldir

# groupadd m11videojocs
# groupadd m11piscina
# groupadd m11users

# useradd -g m11videojocs -G m11users -d /var/tmp/ramon ramon
# useradd -g m11piscina   -G m11users -d /var/tmp/natalia natalia
# useradd -g m11users  -G m11piscina,m11videojocs -d /var/tmp/xavi xavi
# useradd -g m11users -d /var/tmp/noemi noemi
```

$ getfacl acldir/

```
# file: acldir/
# owner: ecanet
# group: inf
user::rwx
group::r-x
other::---

$ setfacl -m user:natalia:rwx,group:m11videojocs:rwx acldir/

$ getfacl acldir/
# file: acldir/
# owner: ecanet
# group: inf
user::rwx
user:natalia:rwx
group::r-x
group:m11videojocs:rwx
mask::rwx
other::---

$ ls -ld acldir/
drwxrwx---+ 2 ecanet inf 4096 ene 29 11:42 acldir/
```

Observar que els permisos que mostra *ls* llistats per a group són rwx però en realitat això és la màscara. Segons sigui el grup s'aplicarà la intersecció dels permisos amb la màscara. Així per al grup *m11videojocs* els permisos són rwx, però per al grup *inf* (group owner) són r-x.

Mask

This *mask* entry is set auto-matically to reduce all entries in the ***group class*** to a common denominator. Furthermore, setfacl automatically adapts existing mask entries to the settings modified, provided you do not deactivate this feature with -n.
mask defines the maximum effective access permissions for all entries in the ***group class***. This includes: ***named user***, ***named group***, and ***owning group***.

Observar que si amb *chmod* es pretén canviar el permís w de grup què passa. En realitat es canvien els permisos de la màscara i queda r-x. Ara tant els named user, owner group i named group estan afectats amb la nova màscara (es fa la intersecció de permisos).

```
$ chmod g-w acldir
$ ls -dl acldir
drwxr-x---+ 2 ecanet inf 4096  3 feb 19:08 acldir
```

```
$ getfacl acldir
# file: acldir/
# owner: ecanet
# group: inf
user::rwx
user:natalia:rwx                #effective:r-x
group::r-x
group:m11videojocs:rwx          #effective:r-x
mask::r-x
other::---
```

El procés per tornar a assignar permisos ala màscara de grup és idèntic. És a dir, *chmod* posa/treu els permisos al propietari (user owner), a la màscara (mask) i a others.

```
$ chmod u-w,g+w,o+rx dirjocs/
$ ls -ld dirjocs/
dr-xrwxr-x+ 2 ecanet ecanet 4096  3 feb 19:08 dirjocs/

$ getfacl dirjocs/
# file: dirjocs/
# owner: ecanet
# group: ecanet
user::r-x
user:pere:rwx
group::r-x
group:videoclub:rwx
mask::rwx
other::r-x
```

Funcionament de mask:

Els permisos ACL a fitxers i directoris estenen els permisos de GNU/Linux a *named users* i *named groups*. Els permisos del propietari (*user owner*) de l'element i els permisos de *other* són sempre independents de ***mask***.

En assignar a un fitxer amb permisos estàndard un permís ACL de tipus named automàticament es genera una entrada de tipus *mask* amb la **unió** dels permisos de les entrades que són subordinades a *mask*. Així per exemple:

```
# 1) un element amb permisos únix estàndard

$ getfacl file.txt
# file: file.txt
```

```
# owner: ecanet
# group: inf
user::rwx
group::r-x
other::--

# 2) Assignar permisos a named users i named groups

$ setfacl -m user:pere:rw-,user:marta:r--,u:ramon:-w-,g:sys:r-x,g:admin:rwx  file.txt

$ getfacl file.txt
# file: file.txt
# owner: ecanet
# group: inf
user::rwx
user:pere:rw-
user:marta:r--
user:ramon:-w-
group::r-x
group:sys:r-x
group:admin:rwx
mask:rwx
other::--

# 3) llistar els permisos estàndard (però no ho són realment)

$ ls -l file.txt
-rwxrwx---+ 2 ecanet inf 4096  3 feb 19:08 file.txt
```

S'ha generat automàticament una entrada ***mask***, amb quins permisos? Amb la **unió** (suma) dels permisos de totes les entrades de tipus *named user*, *group owner* i *named group*. ës a dir: (rw-) + (r--) + (-w-) + (r-x) + (rwx) generen una unió de ***rwx***.

Quan existeix una màscara aquesta actua com a permisos GNU/Linux de grup, és la que es llista amb la ordre ***ls*** i són els permisos de la *mask* els que es modifiquen amb la ordre ***chmod***.

Quina és la funció de la màscara?
Com actua?

La màscara de permisos actual com a un super-interruptor, com una 'màscara'!. És a dir, a un *named user*, *group owner* o *named group* se li aplica la ***intersecció*** del seu permís amb la *mask*.

Així en l'exemple anterior pere disposarà de permís rw- perquè és la interseció dels seus permisos (rw-) i de la mask (rwx). De fet en tots els casos posats en l'exemple el permís efectiu és el propi permís perquè la *mask* és rwx.
La *mask* actua en el cas de pere de la següent manera, pere té r, la permet la *mask*? si, llavors pere té r. Pere té w, la permet la *mask*? si, llavors pere té w. Pere no té x per tant no cal mirar la *mask* (que encara que tingui x no li concedeix, cal recordar que parlem de la intersecció).

funció

La funció de la *mask* és atorgar un interruptor general per ***apagar*** permisos, no per concedir-los. Els permisos de named user, group owner i named group han d'estar concedits individualment i la *mask* els ha de permetre per ser ***efectius***.

La mask no pot atorgar permisos que l'element individual no tingui.

Llavors, quina és la vertadera finalitat de mask? És apagar permisos, per exemple si la *mask* ***desactiva*** la w tots els elements deixaran de tenir com a permís efectiu la w independentment de que el tinguin activat individualment.

Observem el següent exemple:

```
# 4) desactivar el permís w del grup  amb chmod. En realitat modifica la mask

$ chmod g-w file.txt

$ ls -l file.txt
-rwxr-x---+ 2 ecanet inf 4096  3 feb 19:08 file.txt

$ getfacl file.txt
# file: file.txt
# owner: ecanet
# group: inf
user::rwx
user:pere:rw-          #effective:r--
user:marta:r--
user:ramon:-w-         #effective:---
group::r-x
group:sys:r-x
group:admin:rwx    #effective:r-x
mask:r-x
other::--
```

Observar que aquells elements (*named user*, *group owner* i *named group*) que tenen els permisos propis amb la w el permís efectiu no disposa de la w.

Per exemple en el cas de pere que té la r individualment i la *mask* també, la intersecció li concedeix la r. Pere també té la w individualment, però la *mask* no la té, de manera que la intersecció és no disposar de la *mask*. Finalment pere no té la x individualment de manera que tant si la *mask* la té com si no pere no la tindrà.

Conclusió

La mask permet desactivar permisos però activar-los (per activar-los també cal el permís individual).
La mask permet desactivar d'un sol cop a **tots** els named user, group owner i named group algun dels permisos r, w o x de cop, a tots ells.

Mirem per exemple el següent cas inventat:

```
# 5) Un element te assignades multitud de regles named user i named group, amb una mascara rwx

$ getfacl file.txt
# file: file.txt
# owner: ecanet
# group: inf
user::rwx
user:pere:rw-
user:marta:r--
user:ramon:-w-
user:mireia:rw-
user:jaume:rwx
user:jordi:rw-
group::r-x
group:sys:r-x
group:admin:rwx
group:jardi2:rw-
group:carnicers3:rwx
mask:rwx
other::--

# 6) Si es modifica la mask a r-x (eliminant w) de cop tots els named users, named group i el mateix group owner deixen de poder usar la w

$ chmod g-w file.txt
```

Default Acls

Es poden establir ACLs per defecte únicament a directòris. Els elements d'aquests directòri hereten aquesta ACL per defecte. Els fitxers l'hereten com a acl d'accés. Els directòris l'hereten com a ACL d'ac´es i com a ACL per defecte.

Definició:

Directories can be equipped with a special kind of ACL — a default ACL. The default ACL defines the access permissions all objects under this directory inherit when they are created. A default ACL affects subdirectories as well as files.

Efectes:

Basically, the permissions in a default ACL are handed down in two ways:

- A subdirectory inherits the default ACL of the parent directory both as its own default ACL and as an access ACL.
- A file inherits the default ACL as its own access ACL

All system calls that create file system objects use a mode parameter that defines the access permissions for the newly created file system object:

- If the parent directory does not have a default ACL, an intersection of the permissions defined in the mode parameter and those in the current umask is formed and assigned to the object.
- If a default ACL exists for the parent directory, the permission bits are determined according to the intersection of the value of the mode parameter and the permissions defined in the default ACL and assigned to the object.

Exemple:

```
$ setfacl -d -m group:djungle:r-x mydir

$ getfacl mydir
# file: mydir
# owner: tux
# group: project3
user::rwx
```

```
user:jane:rwx
group::r-x
group:djungle:rwx
mask::rwx
other::---
default:user::rwx
default:group::r-x
default:group:djungle:r-x
default:mask::r-x
default:other::---

$mkdir mydir/mysubdir
$getfacl mydir/mysubdir
# file: mydir/mysubdir
# owner: tux
# group: project3
user::rwx
group::r-x
group:djungle:r-x
mask::r-x
other::---
default:user::rwx
default:group::r-x
default:group:djungle:r-x
default:mask::r-x
default:other::---

$ touch mydir/myfile
$ ls -l mydir/myfile
-rw-r-----+ ... tux project3 ... mydir/myfile

$ getfacl mydir/myfile

# file: mydir/myfile
# owner: tux
# group: project3
user::rw-
group::r-x                # effective:r--
group:djungle:r-x     # effective:r--
mask::r--
other::---
```