Sharing Data

Traditional UNIX permissions and NFSv4 Access Control Lists

Traditional UNIX permissions

Linux is a multi-user OS that is based on the Unix concepts of file ownership and permissions to provide security at the file system level. Each file is owned by a single user, a single group, and has its own access permissions. The traditional UNIX file system object permission model defines three permission classes called *owner, group, and world*:

- The o*wner* **i**s the user account that has primary power over the file or directory, allowing it to do things like change the file's permissions. The owner of a file is the account used to create the file, though files can be reassigned by the root account to a different owner using the chown command.
- The g*roup* has its own set of access permissions to the file. When creating a file, the group is set to the default group of the user account used to create the file, though files can be reassigned to a different group by the owner using the chgrp command.

- The w*orld* (sometimes referred to as o*ther*) covers everyone else -- any accounts that are not the owner or a member of the group associated with the file.

There are three basic permissions defined as *read (r), write (w), and execute (x)* associated to each of these classes

- The *read* permission grants the ability to read a file. When set for a directory, this permission grants the ability to read the names of files in the directory, but not to find out any further information about them such as contents, file type, size, ownership, permissions.
- The *write* permission grants the ability to modify a file. When set for a directory, this permission grants the ability to modify entries in the directory, which includes creating files, deleting files, and renaming files. Note that this requires that *execute* is also set; without it, the write permission is meaningless for directories.
- The *execute* permission grants the ability to execute a file. This permission must be set for executable programs, in order to allow the operating system to run them. When set for a directory, the execute permission is interpreted as the *search* permission: it grants the ability to access file contents and meta-information if its name is known, but not list files inside the directory, unless *read* is set also.

To see a file's permissions settings, the -l option for the ls command provides information that includes permissions in the first column.

```
edington@xbio:~>ls -l

drwxr-xr-x  3 edington hpc        423 Nov  4  2011 test_data

-rw-r--r--  1 edington hpc         28 Nov  6  2015 TESTFILE

-rwxr-xr-x  3 edington hpc        423 Nov  4  2011 myscript.sh
```

A “d” at the beginning of a line indicates the file is a directory, a “-” at the beginning of a line indicates that it is a regular file. The rest of the permissions block is broken up into three-character groups, corresponding in order to owner's read, write, and execute; group's read, write, and execute; and world's read, write, and execute. A dash in any position indicates that the relevant entity category does not have that permission.

You can see the group memberships for a user with the id command:

```
id edington

uid=1020(edington) gid=3010(hpc) groups=3010(hpc),4(adm),2006(database)
```

Access Control Lists

Traditional UNIX permissions are very limited in the access they can provide. UNIX permissions can only be set for the owner, one group and everyone else (other). Access Control Lists (ACLs) are more recent in origin and are universally used on Microsoft Windows based file systems. They are complex but capable of more detailed fine-tuning of permissions than the traditional Unix permissions. ACLs can assign permissions to the owner, the group and to multiple specific users and groups of users.

On Unix and Linux based systems POSIX ACLs are the standard but other variants exist such as NFSv4 ACLs which work slightly differently. NFSv4 ACLs provide finer granularity than typical POSIX read/write/execute permissions and are similar to SAMBA (or CIFS) ACLs. We are recommending NFSv4 ACLs because the syntax is sane, recursion is supported and it is recommended for IBM SAMBA.

Traditional NFSv4 ACL syntax

An ACL is a list of permissions associated with a file or directory and consists of one or more Access Control Entries (ACEs). An ACE is an individual entry in an access control list, and describes the permissions for an individual user or group of users. An ACL can have zero or more ACEs.

A single NFSv4 ACE is written as an ace_spec, which is a colon-delimited 4-field string in the following format:

[type]:[flags]:[principal]:[permissions]

All parts are required for every operation though the [flags] section may be empty. To get a list of what all the mask bits mean type "nfs4_getfacl" or read the nfs4_acl(5) manpage http://man7.org/linux/man-pages/man5/nfs4_acl.5.html. Here are the most common options for an ACE.

- *[type]*: There are four types of ACEs, each represented by a single character. An ACE must have exactly one type. The only useful one for us is Allow. Because NFSv4 ACLs are "default-deny", the use of Deny ACEs should be avoided entirely in most cases. Deny ACEs can be confusing and complicated. Unlike POSIX ACLs and CIFS ACLs, the ordering of ACEs within NFSv4 ACLs affects how they are evaluated.

  A Allow - allow principal to perform actions requiring permissions.

- *[flags]*: There are three kinds of ACE flags: group, inheritance, and administrative.

  GROUP FLAG - can be used in any ACE

  g group - indicates that principal represents a group instead of a user.

  INHERITANCE FLAGS - can only be used in a directory ACE Don’t use these unless you know exactly what you are doing. They are complicated.

  fdi — These flags collectively indicate that the ACE is to be inherited. If set, any file created below the directory will have the given ACE, but without the inheritance flags. Additionally, any directory created below one with these flags will have the ACE, and will also have a second ACE with the inheritance flags. Inheritance flags are recursive. Check this...I think this only works if the ACL is put on the directory before any files or subdirectories are created.

- [principal] — The principal is the entity to which the ACE refers. A principal is either a named user (e.g., `myuser@mskcc.org') or group (provided the group flag is also set), or one of three special principals: `OWNER@', `GROUP@', and `EVERYONE@', which are, respectively, analogous to the POSIX user/group/other distinctions used in, e.g., chmod(1).

- [permissions] —Permissions are the symbols that come at the end. NFSv4 ACLs provide a variety of different ACE permissions (13 for files, 14 for directories), each represented by a single character.

'r'	read-data / list-directory
'w'	write-data / create-file
'a'	append-data / create-subdirectory
'x'	execute
'd'	delete
'D'	delete-child (directories only)
't'	read-attrs
'T'	write-attrs
'n'	read-named-attrs
'N'	write-named-attrs
'c'	read-ACL
'C'	write-ACL
'o'	write-owner
'y'	synchronize

Examples of NFSv4 ACL commands

To read an ACL, use nfs4_getfacl on a file in a nfsv4-mounted directory use nfs4_getfacl.

```
nfs4_getfacl /mnt/nfsv4/my-file.txt
```

Need more local examples here

To modify an acl, use nfs4_getfacl on a file in a nfsv4-mounted directory:

```
nfs4_setfacl -e /mnt/nfsv4/my-file.txt
```

which will allow you to edit the acl in a text editor; edit the acl and save the result, and the modified acl will be set on the file when you exit. Alternatively, you can give the ACEs in a file

More local examples here.

Where to set NFSv4 ACLs

hpc-acl01.mskcc.org is a VM with NFSv4 mounts of GPFS file systems. If you would like to manage ACLs please contact us and we can provide access and a NFSv4 mount of your data.

The users and groups for the ACLs come from HPC LDAP. HPC is responsible for maintaining them at this time.

Moving data with NFSv4 ACLs

User space utilities cp and tar are capable of transferring the NFSv4 ACLs using extended attributes. The CentOS7 default rsync version 3.1.2 is not. We do not have this rsync version on any of our systems at this time and ACLs will not be backed up. Rsync version 3.1.3 has added filter options that will allow this to work in future when we upgrade to RHEL 8 or CentOS 8.

```
cp --preserve=xattr

tar --xattrs
```

All HPC GPFS file systems support NFSv4 ACLs. Isilon /ifs and local directories such as /scratch do not. If you copy a file with --preserve=xattr to another filesystem that does not support NFFv4 ACLs they will be mapped to POSIX ACLs. if the original ACL is representable as a POSIX ACL, then the ACL returned should represent equivalent permissions to the one set. If not, then the ACL returned should have permissions that are stricter than those requested. See the getacl and setacl man pages for more documentation.

Caveats

The ls -l command does not show a “+” at the end of the permissions when a file has an attached NFSv4 ACL. https://bugzilla.redhat.com/show_bug.cgi?id=1269951

ACLs should only be set on an NFSv4 mount point or from a SAMBA mount on a Windows (Mac?)

Do not use ACL inheritance. Recursion works for NFSv4 ACLS, use that.

NFSv4 ACLs in GPFS

GPFS has support for NFSv4 ACLs but the syntax is different. While traditional NFSv4 ACLs display one entry per line, each GPFS representation of a NFSv4 ACL entry is three lines. The fields are in a completely different order and they include special key words. This is to confuse everyone. While you can view ACLs in GPFS we do not recommend that you set them using GPFS commands. Read on if you want to understand how to view NFSv4 ACLs in GPFS. Here is the description of NFSv4 ACLs from the spectrum scale documentation at https://www.ibm.com/support/knowledgecenter/STXKQY_5.0.4/com.ibm.spectrum.scale.v5r04.doc/bl1adm_nfsv4syn.htm

The first line has several parts separated by colons (':').

- The first part identifies the user or group.
- The second part displays a rwxc translation of the permissions that appear on the subsequent two lines.
- The third part is the ACL type. NFS V4 provides both an *allow* and *deny* type.
- ***allow***

- Means to allow (or permit) those permissions that have been selected with an 'X'.

- ***deny***

- Means to not allow (or deny) those permissions that have been selected with an 'X'.

- The fourth and final part is a list of flags indicating *inheritance*.
- Valid flag values are:
- **DirInherit**

- Indicates that the ACL entry should be included in the initial ACL for subdirectories created in this directory (as well as the current directory).

- **FileInherit**

- Indicates that the ACL entry should be included in the initial ACL for files created in this directory.

- **Inherited**
- Indicates that the current ACL entry was derived from inherit entries in an NFS v4 ACL of the parent directory.
- **InheritOnly**
- Indicates that the current ACL entry should *not* apply to the directory, but *should* be included in the initial ACL for objects created in this directory.
- **NoPropagateInherit**
- Indicates that the ACL entry should be included in the initial ACL for subdirectories created in this directory but not further propagated to subdirectories created below *that* level.

As in traditional ACLs, users and groups are identified by specifying the type and name. For example, group:staff or user:bin. NFS V4 provides for a set of special names that are not associated with a specific local UID or GID. These special names are identified with the keyword special followed by the NFS V4 name. These names are recognized by the fact that they end with the character '@'. For example, special:owner@ refers to the owner of the file, special:group@ the owning group, and special:everyone@ applies to all users.

The next two lines provide a list of the available access permissions that may be *allowed* or *denied*, based on the ACL type specified on the first line. A permission is selected using an 'X'. Permissions that are not specified by the entry should be left marked with '-' (minus sign).

These are examples of NFS V4 ACLs.

- An ACL entry that explicitly allows **READ**, **EXECUTE** and **READ_ATTR** to the **staff** group on a file is similar to this:
- group:staff:r-x-:allow (X)READ/LIST (-)WRITE/CREATE (-)APPEND/MKDIR (-)SYNCHRONIZE (-)READ_ACL (X)READ_ATTR (-)READ_NAMED (-)DELETE   (-)DELETE_CHILD (-)CHOWN (X)EXEC/SEARCH (-)WRITE_ACL (-)WRITE_ATTR (-)WRITE_NAMED

- A Directory ACL is similar to this. It may include *inherit* ACL entries that do not apply to the directory itself, but instead become the initial ACL for any objects created within the directory.special:

- group@:----:deny:DirInherit:InheritOnly (X)READ/LIST (-)WRITE/CREATE (-)APPEND/MKDIR (-)SYNCHRONIZE (-)READ_ACL (X)READ_ATTR (-)READ_NAMED (-)DELETE   (-)DELETE_CHILD (-)CHOWN (X)EXEC/SEARCH (-)WRITE_ACL (-)WRITE_ATTR (-)WRITE_NAMED
- A complete NFS V4 ACL is similar to this:#NFSv4 ACL#owner:smithj#group:staffspecial:owner@:rwxc:allow:FileInherit (X)READ/LIST (X)WRITE/CREATE (X)APPEND/MKDIR (-)SYNCHRONIZE (X)READ_ACL (X)READ_ATTR (-)READ_NAMED (X)DELETE   (X)DELETE_CHILD (X)CHOWN (X)EXEC/SEARCH (X)WRITE_ACL (X)WRITE_ATTR (-)WRITE_NAMEDspecial:owner@:rwxc:allow:DirInherit:InheritOnly (X)READ/LIST (X)WRITE/CREATE (X)APPEND/MKDIR (-)SYNCHRONIZE (X)READ_ACL (X)READ_ATTR (-)READ_NAMED (X)DELETE   (X)DELETE_CHILD (X)CHOWN (X)EXEC/SEARCH (X)WRITE_ACL (-)WRITE_ATTR (-)WRITE_NAMEDuser:smithj:rwxc:allow (X)READ/LIST (X)WRITE/CREATE (X)APPEND/MKDIR (-)SYNCHRONIZE (X)READ_ACL (X)READ_ATTR (-)READ_NAMED (X)DELETE   (X)DELETE_CHILD (X)CHOWN (X)EXEC/SEARCH (X)WRITE_ACL (-)WRITE_ATTR (-)WRITE_NAMED

Using mmsetfacl is potentially damaging as they will overwrite the posix permissions and have no append or delete capability. --check this.

References

https://wiki.linux-nfs.org/wiki/index.php/ACLs

https://lwn.net/Articles/661357/

https://www.thegeekdiary.com/unix-linux-access-control-lists-acls-basics/

http://www.citi.umich.edu/projects/nfsv4/linux/using-acls.html

http://www.citi.umich.edu/projects/nfsv4/jallison-acl-mapping/img1.html

https://www.eecs.utk.edu/resources/it/eecs-it-knowledge-base/nfsv4-acls/