Original ACL related man pages

NAME

getfacl - get file access control lists

SYNOPSIS

```
getfacl [-dRLPvh] file ...

getfacl [-dRLPvh] -
```

DESCRIPTION

For each file, getfacl displays the file name, owner, the group, and the Access Control List (ACL). If a directory has a default ACL, getfacl also displays the default ACL. Non-directories cannot have default ACLs.

If getfacl is used on a file system that does not support ACLs, getfacl displays the access permissions defined by the traditional file mode permission bits.

The output format of getfacl is as follows:

```
 1: # file: somedir/
 2: # owner: lisa
 3: # group: staff
 4: user::rwx
 5: user:joe:rwx     #effective:r-x
 6: group::rwx     #effective:r-x
 7: group:cool:r-x
 8: mask:r-x
 9: other:r-x
10: default:user::rwx
11: default:user:joe:rwx    #effective:r-x
12: default:group::r-x
13: default:mask:r-x
14: default:other:---
```

Lines 4, 6 and 9 correspond to the user, group and other fields of the file mode permission bits. These three are called the base ACL entries. Lines 5 and 7 are named user and named group entries. Line 8 is the effective rights mask. This entry limits the effective rights granted to all groups and to named users. (The file owner and others permissions are not affected by the effective rights mask; all other entries are.) Lines 10--14 display the default ACL associated with this directory. Directories may have a default ACL. Regular files never have a default ACL.

The default behavior for getfacl is to display both the ACL and the default ACL, and to include an effective rights comment for lines where the rights of the entry differ from the effective rights.

If output is to a terminal, the effective rights comment is aligned to column 40. Otherwise, a single tab character separates the ACL entry and the effective rights comment.

The ACL listings of multiple files are separated by blank lines. The output of getfacl can also be used as input to setfacl.

PERMISSIONS

Process with search access to a file (i.e., processes with read access to the containing directory of a file) are also granted read access to the file's ACLs. This is analogous to the permissions required for accessing the file mode.

OPTIONS

--access

Display the file access control list.

-d, --default

Display the default access control list.

--omit-header
Do not display the comment header (the first three lines of each file's output).

--all-effective
Print all effective rights comments, even if identical to the rights defined by the ACL entry.

--no-effective
Do not print effective rights comments.

--skip-base
Skip files that only have the base ACL entries (owner, group, others).

-R, --recursive
List the ACLs of all files and directories recursively.

-L, --logical
Logical walk, follow symbolic links. The default behavior is to follow symbolic link arguments, and to skip symbolic links encountered in subdirectories.

-P, --physical
Physical walk, skip all symbolic links. This also skips symbolic link arguments.

--tabular
Use an alternative tabular output format. The ACL and the default ACL are displayed side by side. Permissions that are ineffective due to the ACL mask entry are displayed capitalized. The entry tag names for the ACL_USER_OBJ and ACL_GROUP_OBJ entries are also displayed in capital letters, which helps in spotting those entries.

--absolute-names
Do not strip leading slash characters (`/'). The default behavior is to strip leading slash characters.

--version
Print the version of getfacl and exit.

--help
Print help explaining the command line options.

-- End of command line options. All remaining parameters are interpreted as file names, even if they start with a dash character.

- If the file name parameter is a single dash character, getfacl reads a list of files from standard input.

CONFORMANCE TO POSIX 1003.1e DRAFT STANDARD 17
If the environment variable POSIXLY_CORRECT is defined, the default behavior of getfacl changes in the following ways: Unless otherwise specified, only the ACL is printed. The default ACL is only printed if the -d option is given. If no command line parameter is given, getfacl behaves as if it was invoked as ``getfacl -''.

AUTHOR
Andreas Gruenbacher, <a.gruenbacher@computer.org>.

Please send your bug reports and comments to the above address.

SEE ALSO
setfacl(1), acl(5)

NAME
setfacl - set file access control lists

SYNOPSIS
setfacl [-bkndRLPvh] [{-m|-x} acl_spec] [{-M|-X} acl_file] file ...

setfacl --restore=file

DESCRIPTION

This utility sets Access Control Lists (ACLs) of files and directories. On the command line, a sequence of commands is followed by a sequence of files (which in turn can be followed by another sequence of commands, ...).

The options -m, and -x expect an ACL on the command line. Multiple ACL entries are separated by comma characters (`,'). The options -M, and -X read an ACL from a file or from standard input. The ACL entry format is described in Section ACL ENTRIES.

The --set and --set-file options set the ACL of a file or a directory. The previous ACL is replaced. ACL entries for this operation must include permissions.

The -m (--modify) and -M (--modify-file) options modify the ACL of a file or directory. ACL entries for this operation must include permissions.

The -x (--remove) and -X (--remove-file) options remove ACL enries. Only ACL entries without the perms field are accepted as parameters, unless POSIXLY_CORRECT is defined.

When reading from files using the -M, and -X options, setfacl accepts the output getfacl produces. There is at most one ACL entry per line. After a Pound sign (`#'), everything up to the end of the line is treated as a comment.

If setfacl is used on a file system which does not support ACLs, setfacl operates on the file mode permission bits. If the ACL does not fit completely in the permission bits, setfacl modifies the file mode permission bits to reflect the ACL as closely as possible, writes an error message to standard error, and returns with an exit status greater than 0.

PERMISSIONS

The file owner and processes capable of CAP_FOWNER are granted the right to modify ACLs of a file. This is analogous to the permissions required for accessing the file mode. (On current Linux systems, root is the only user with the CAP_FOWNER capability.)

OPTIONS

-b, --remove-all
Remove all extended ACL entries. The base ACL entries of the owner, group and others are retained.

-k, --remove-default
Remove the Default ACL. If no Default ACL exists, no warnings are issued.

-n, --no-mask
Do not recalculate the effective rights mask. The default behavior of setfacl is to recalculate the ACL mask entry, unless a mask entry was explicitly given. The mask entry is set to the union of all permissions of the owning group, and all named user and group entries. (These are exactly the entries affected by the mask entry).

--mask
Do recalculate the effective rights mask, even if an ACL mask entry was explicitly given. (See the -n option.)

-d, --default
All operations apply to the Default ACL. Regular ACL entries in the input set are promoted to Default ACL entries. Default ACL entries in the input set are discarded. (A warning is issued if that happens).

--restore=file
Restore a permission backup created by `getfacl -R' or similar. All permissions of a complete directory subtree are restored using this mechanism. If the input contains owner comments or group comments, and setfacl is run by root, the owner and owning group of all files are restored as well. This option cannot be mixed with other options except `--test'.

--test
Test mode. Instead of changing the ACLs of any files, the resulting ACLs are listed.

-R, --recursive
Apply operations to all files and directories recursively. This option cannot be mixed with `--restore'.

-L, --logical
Logical walk, follow symbolic links. The default behavior is to follow symbolic link arguments, and to skip symbolic links encountered in subdirectories. This option cannot be mixed with `--restore'.

-P, --physical
Physical walk, skip all symbolic links. This also skips symbolic link arguments. This option cannot be mixed with `--restore'.

--version
Print the version of setfacl and exit.

--help
Print help explaining the command line options.

-- End of command line options. All remaining parameters are interpreted as file names, even if they start with a dash.

- If the file name parameter is a single dash, setfacl reads a list of files from standard input.

ACL ENTRIES

The setfacl utility recognizes the following ACL entry formats (blanks inserted for clarity):

[d[efault]:] [u[ser]:]uid [:perms]
Permissions of a named user. Permissions of the file owner if uid is empty.

[d[efault]:] g[roup]:gid [:perms]
Permissions of a named group. Permissions of the owning group if gid is empty.

[d[efault]:] m[ask][:] [:perms]
Effective rights mask

[d[efault]:] o[ther][:] [:perms]
Permissions of others.

Whitespace between delimiter characters and non-delimiter characters is ignored.

Proper ACL entries including permissions are used in modify and set operations. (options -m, -M, --set and --set-file). Entries without the perms field are used for deletion of entries (options -x and -X).

For uid and gid you can specify either a name or a number.

The perms field is a combination of characters that indicate the permissions: read (r), write (w), execute (x), execute only if the file is a directory or already has execute permission for some user (X). Alternatively, the perms field can be an octal digit (0-7).

AUTOMATICALLY CREATED ENTRIES

Initially, files and directories contain only the three base ACL entries for the owner, the group, and others. There are some rules that need to be satisfied in order for an ACL to be valid:

* The three base entries cannot be removed. There must be exactly one entry of each of these base entry types.

* Whenever an ACL contains named user entries or named group objects, it must also contain an effective rights mask.

* Whenever an ACL contains any Default ACL entries, the three Default ACL base entries (default owner, default group, and default others) must also exist.

* Whenever a Default ACL contains named user entries or named group objects, it must also contain a default effective rights mask.

To help the user ensure these rules, setfacl creates entries from existing entries under the following conditions:

* If an ACL contains named user or named group entries, and no mask entry exists, a mask entry containing the same permissions as the group entry is created. Unless the -n option is given, the permissions of the mask entry are further adjusted to include the union of all permissions affected by the mask entry. (See the -n option description).

* If a Default ACL entry is created, and the Default ACL contains no owner, owning group, or others entry, a copy of the ACL owner, owning group, or others entry is added to the Default ACL.

* If a Default ACL contains named user entries or named group entries, and no mask entry exists, a mask entry containing the same permissions as the default Default ACL's group entry is added. Unless the -n option is given, the permissions of the mask entry are further adjusted to inclu de the union of all permissions affected by the mask entry. (See the -n option description).

EXAMPLES

Granting an additional user read access
```
setfacl -m u:lisa:r file
```

Revoking write access from all groups and all named users (using the effective rights mask)
```
setfacl -m m::rx file
```

Removing a named group entry from a file's ACL
```
setfacl -x g:staff file
```

Copying the ACL of one file to another
```
getfacl file1 | setfacl --set-file=- file2
```

Copying the access ACL into the Default ACL
```
getfacl -a dir | setfacl -d -M- dir
```

CONFORMANCE TO POSIX 1003.1e DRAFT STANDARD 17

If the environment variable POSIXLY_CORRECT is defined, the default behavior of setfacl changes as follows: All non-standard options are disabled. The ``default:'' prefix is disabled. The -x and -X options also accept permission fields (and ignore them).

AUTHOR

Andreas Gruenbacher, <a.gruenbacher@computer.org>.

Please send your bug reports, suggested features and comments to the above address.

SEE ALSO

getfacl(1),chmod(1),umask(1), acl(5)

NAME

chacl - change the access control list of a file or directory

SYNOPSIS

```
chacl acl pathname...
chacl -b acl dacl pathname...
chacl -d dacl pathname...
chacl -R pathname...
chacl -D pathname...
chacl -B pathname...
chacl -l pathname...
chacl -r pathname...
```

DESCRIPTION

chacl is an IRIX-compatibility command, and is maintained for those users who are familiar with its use from either XFS or IRIX. Refer to the SEE ALSO section below for a description of tools which conform more closely to the (withdrawn draft) POSIX 1003.1e standard which describes Access Control Lists (ACLs).

chacl changes the ACL(s) for a file or directory. The ACL(s) specified are applied to each file in the pathname arguments.

Each ACL is a string which is interpreted using the acl_from_text(3) routine. These strings are made up of comma separated clauses each of which is of the form, tag:name:perm. Where tag can be:

"user" (or "u")
indicating that the entry is a user ACL entry.

"group" (or "g")
indicating that the entry is a group ACL entry.

"other" (or "o")
indicating that the entry is an other ACL entry.

"mask" (or "m")
indicating that the entry is a mask ACL entry.

name is a string which is the user or group name for the ACL entry. A null name in a user or group ACL entry indicates the file's owner or file's group. perm is the string "rwx" where each of the entries may be replaced by a "-" indicating no access of that type, e.g. "r-x", "--x", "---".

OPTIONS

-b Indicates that there are two ACLs to change, the first is the file access ACL and the second the directory default ACL.

-d Used to set only the default ACL of a directory.

-R Removes the file access ACL only.

-D Removes directory default ACL only.

-B Remove all ACLs.

-l Lists the access ACL and possibly the default ACL associated with the specified files or directories. This option was added during the Linux port of XFS, and is not IRIX compatible.

-r Set the access ACL recursively for each subtree rooted at path-name(s). This option was also added during the Linux port of XFS, and is not compatible with IRIX.

EXAMPLES

A minimum ACL:

```
chacl u::rwx,g::r-x,o::r-- file
```

The file ACL is set so that the file's owner has "rwx", the file's

```
    group has read and execute, and others have read  only access to the
    file.

    An ACL that is not a minimum ACL, that is, one that specifies a user or
    group other than the file's owner or owner's group, must contain a mask
    entry:

  chacl u::rwx,g::r-x,o::r--,u:bob:r--,m::r-x file1 file2

    To set the default and access ACLs on newdir to be the same as on old-
    dir, you could type:

  chacl -b `chacl -l olddir | \
    sed -e 's/.*\[//' -e 's#/# #' -e 's/]$//'` newdir

CAUTIONS
    chacl can replace the existing ACL. To add or delete entries, you must
    first do chacl -l to get the existing ACL, and use the output to form
    the arguments to chacl.

    Changing the permission bits of a file will change the file access ACL
    settings  (see chmod(1)).  However, file creation mode masks (see
    umask(1)) will not affect the access ACL settings of files created
    using directory default ACLs.

    ACLs are filesystem extended attributes and hence are not typically
    archived or restored using the conventional archiving utilities.  See
    attr(5) for more information about extended attributes and see xfsdump(8)
    for a method of backing them up under XFS.

SEE ALSO
   getfacl(1),setfacl(1),chmod(1),umask(1), acl_from_text(3), acl(5),
    xfsdump(8)
```

Currently help from NFSv4_acl commands Man pages coming summer CY05.

bfields@puzzle:~$ nfs4_getfacl -h
Useage: nfs4_getfacl [-f <outputfile>] <path>
Prints the NFSv4 acl entry for the given path.

Permission letter mapping:

r - NFS4_ACE_READ_DATA
w - NFS4_ACE_WRITE_DATA
a - NFS4_ACE_APPEND_DATA
x - NFS4_ACE_EXECUTE
d - NFS4_ACE_DELETE
l - NFS4_ACE_LIST_DIRECTORY
f - NFS4_ACE_ADD_FILE
s - NFS4_ACE_ADD_SUBDIRECTORY
n - NFS4_ACE_READ_NAMED_ATTRS
N - NFS4_ACE_WRITE_NAMED_ATTRS
D - NFS4_ACE_DELETE_CHILD
t - NFS4_ACE_READ_ATTRIBUTES
T - NFS4_ACE_WRITE_ATTRIBUTES
c - NFS4_ACE_READ_ACL
C - NFS4_ACE_WRITE_ACL
o - NFS4_ACE_WRITE_OWNER

y - NFS4_ACE_SYNCHRONIZE
bfields@puzzle:~$ nfs4_getfacl FOO
1: A::OWNER@:rwatTcCy
2: D::OWNER@:x
3: A:g:GROUP@:rtcy
4: D:g:GROUP@:waxTC
5: A::EVERYONE@:rtcy
6: D::EVERYONE@:waxTC

stat

NAME

stat, fstat, lstat - get file status

SYNOPSIS

#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

int stat(const char *_file_name_, struct stat *_buf_);
int fstat(int _filedes_, struct stat *_buf_);
int lstat(const char *_file_name_, struct stat *_buf_);

DESCRIPTION

These functions return information about the specified file. You do not need any access rights to the file to get this information but you need search rights to all directories named in the path leading to the file.

stat stats the file pointed to by *file_name* and fills in *buf*.

lstat is identical to **stat**, except in the case of a symbolic link, where the link itself is stat-ed, not the file that it refers to.

fstat is identical to **stat**, only the open file pointed to by *filedes* (as returned by **open**(2)) is stat-ed in place of *file_name*.

They all return a *stat* structure, which contains the following fields:

```
struct stat {
    dev_t         st_dev;      /* device */
    ino_t         st_ino;      /* inode */
    mode_t        st_mode;     /* protection */
    nlink_t       st_nlink;    /* number of hard links */
    uid_t         st_uid;      /* user ID of owner */
    gid_t         st_gid;      /* group ID of owner */
```

```
    dev_t         st_rdev;     /* device type (if inode device)
*/
    off_t         st_size;     /* total size, in bytes */
    blksize_t     st_blksize;  /* blocksize for filesystem I/O */
    blkcnt_t      st_blocks;   /* number of blocks allocated */
    time_t        st_atime;    /* time of last access */
    time_t        st_mtime;    /* time of last modification */
    time_t        st_ctime;    /* time of last change */
};
```

The value *st_size* gives the size of the file (if it is a regular file or a symlink) in bytes. The size of a symlink is the length of the pathname it contains, without trailing NUL.

The value *st_blocks* gives the size of the file in 512-byte blocks. (This may be smaller than *st_size*/512 e.g. when the file has holes.) The value *st_blksize* gives the "preferred" blocksize for efficient file system I/O. (Writing to a file in smaller chunks may cause an inefficient read-modify-rewrite.)

Not all of the Linux filesystems implement all of the time fields. Some file system types allow mounting in such a way that file accesses do not cause an update of the *st_atime* field. (See `noatime' in **mount**(8).)

The field *st_atime* is changed by file accesses, e.g. by **execve**(2), **mknod**(2), **pipe**(2), **utime**(2) and **read**(2) (of more than zero bytes). Other routines, like **mmap**(2), may or may not update *st_atime*.

The field *st_mtime* is changed by file modifications, e.g. by **mknod**(2), **truncate**(2), **utime**(2) and **write**(2) (of more than zero bytes). Moreover, *st_mtime* of a directory is changed by the creation or deletion of files in that directory. The *st_mtime* field is *not* changed for changes in owner, group, hard link count, or mode.

The field *st_ctime* is changed by writing or by setting inode information (i.e., owner, group, link count, mode, etc.).

The following POSIX macros are defined to check the file type:

S_ISREG(m)
is it a regular file?
S_ISDIR(m)
directory?
S_ISCHR(m)
character device?
S_ISBLK(m)
block device?
S_ISFIFO(m)
fifo?
S_ISLNK(m)
symbolic link? (Not in POSIX.1-1996.)
S_ISSOCK(m)
socket? (Not in POSIX.1-1996.)

The following flags are defined for the *st_mode* field:

bitmask for the file type bitfield
socket

symbolic link
regular file
block device
directory
character device
fifo
set UID bit
set GID bit (see below)
sticky bit (see below)
mask for file owner permissions
owner has read permission
owner has write permission
owner has execute permission
mask for group permissions
group has read permission
group has write permission
group has execute permission
mask for permissions for others
others have read permission
others have write permisson
others have execute permissior

The set GID bit (S_ISGID) has several special uses: For a directory it indicates that BSD semantics is to be used for that directory: files created there inherit their group ID from the directory, not from the effective gid of the creating process, and directories created there will also get the S_ISGID bit set. For a file that does not have the group execution bit (S_IXGRP) set, it indicates mandatory file/record locking. The `sticky' bit (S_ISVTX) on a directory means that a file in that directory can be renamed or deleted only by the owner of the file, by the owner of the directory, and by root.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ERRORS

EBADF
filedes is bad.
ENOENT
A component of the path *file_name* does not exist, or the path is an empty string.
ENOTDIR
A component of the path is not a directory.
ELOOP
Too many symbolic links encountered while traversing the path.
EFAULT
Bad address.
EACCES
Permission denied.
ENOMEM
Out of memory (i.e. kernel memory).
ENAMETOOLONG

File name too long.

CONFORMING TO

The **stat** and **fstat** calls conform to SVr4, SVID, POSIX, X/OPEN, BSD 4.3. The **lstat** call conforms to 4.3BSD and SVr4. SVr4 documents additional **fstat** error conditions EINTR, ENOLINK, and EOVERFLOW. SVr4 documents additional **stat** and **lstat** error conditions EACCES, EINTR, EMULTIHOP, ENOLINK, and EOVERFLOW. Use of the *st_blocks* and *st_blksize* fields may be less portable. (They were introduced in BSD. Are not specified by POSIX. The interpretation differs between systems, and possibly on a single system when NFS mounts are involved.)

POSIX does not describe the S_IFMT, S_IFSOCK, S_IFLNK, S_IFREG, S_IFBLK, S_IFDIR, S_IFCHR, S_IFIFO, S_ISVTX bits, but instead demands the use of the macros S_ISDIR(), etc. The S_ISLNK and S_ISSOCK macros are not in POSIX.1-1996, but both will be in the next POSIX standard; the former is from SVID 4v2, the latter from SUSv2.

Unix V7 (and later systems) had S_IREAD, S_IWRITE, S_IEXEC, where POSIX prescribes the synonyms S_IRUSR, S_IWUSR, S_IXUSR.

OTHER SYSTEMS

Values that have been (or are) in use on various systems:

	description
	mask for file type
	SCO out-of-service inode, BSD unknown type
	SVID-v2 and XPG2 have both 0 and 0100000 f
	fifo (named pipe)
	character special (V7)
	multiplexed character special (V7)
	directory (V7)
	XENIX named special file
	with two subtypes, distinguished by st_rdev va
	XENIX semaphore subtype of IFNAM
	XENIX shared data subtype of IFNAM
	block special (V7)
	multiplexed block special (V7)
	regular (V7)
	VxFS compressed
	network special (HP-UX)
	symbolic link (BSD)
	Solaris shadow inode for ACL (not seen by use
	socket (BSD; also "S_IFSOC" on VxFS)
	Solaris door
	BSD whiteout (not used for inode)
	`sticky bit': save swapped text even after use
	reserved (SVID-v2)
	On non-directories: don't cache this file (SunO

On directories: restricted deletion flag (SVID-v
set group ID on execution (V7)
for directories: use BSD semantics for propaga
SysV file locking enforcement (shared w/ S_IS
set user ID on execution (V7)
directory is a context dependent file (HP-UX)

A sticky command appeared in Version 32V AT&T UNIX.

SEE ALSO

chmod(2), **chown**(2), **readlink**(2), **utime**(2)