NAME

exports − NFS server export table

DESCRIPTION

The file */etc/exports* contains a table of local physical file systems on an NFS server that are accessible to NFS clients. The contents of the file are maintained by the server's system administrator.

Each file system in this table has a list of options and an access control list. The table is used by **exportfs**(8) to give information to **mountd**(8).

The file format is similar to the SunOS *exports* file. Each line contains an export point and a whitespace-separated list of clients allowed to mount the file system at that point. Each listed client may be immediately followed by a parenthesized, comma-separated list of export options for that client. No whitespace is permitted between a client and its option list.

Also, each line may have one or more specifications for default options after the path name, in the form of a dash ("−") followed by an option list. The option list is used for all subsequent exports on that line only.

Blank lines are ignored. A pound sign ("#") introduces a comment to the end of the line. Entries may be continued across newlines using a backslash. If an export name contains spaces it should be quoted using double quotes. You can also specify spaces or other unusual character in the export name using a backslash followed by the character code as three octal digits.

To apply changes to this file, run **exportfs**−ra or restart the NFS server.

Machine Name Formats

NFS clients may be specified in a number of ways:

single host
: You may specify a host either by an abbreviated name recognized be the resolver, the fully qualified domain name, an IPv4 address, or an IPv6 address. IPv6 addresses must not be inside square brackets in /etc/exports lest they be confused with character-class wildcard matches.

netgroups
: NIS netgroups may be given as *@group*. Only the host part of each netgroup members is consider in checking for membership. Empty host parts or those containing a single dash (−) are ignored.

wildcards
: Machine names may contain the wildcard characters * and *?*, or may contain character class lists within [square brackets]. This can be used to make the *exports* file more compact; for instance, **.cs.foo.edu* matches all hosts in the domain *cs.foo.edu*. As these characters also match the dots in a domain name, the given pattern will also match all hosts within any subdomain of *cs.foo.edu*.

IP networks
: You can also export directories to all hosts on an IP (sub-) network simultaneously. This is done by specifying an IP address and netmask pair as *address/netmask* where the netmask can be specified in dotted-decimal format, or as a contiguous mask length. For example, either '/255.255.252.0' or '/22' appended to the network base IPv4 address results in identical subnetworks with 10 bits of host. IPv6 addresses must use a contiguous mask length and must not be inside square brackets to avoid confusion with character-class wildcards. Wildcard characters generally do not work on IP addresses, though they may work by accident when reverse DNS lookups fail.

RPCSEC_GSS security

You may use the special strings "gss/krb5", "gss/krb5i", or "gss/krb5p" to restrict access to clients using rpcsec_gss security. However, this syntax is deprecated; on linux kernels since 2.6.23, you should instead use the "sec=" export option:

sec=
: The sec= option, followed by a colon-delimited list of security flavors, restricts the export to clients using those flavors. Available security flavors include sys (the default--no cryptographic security), krb5 (authentication only), krb5i (integrity protection), and krb5p (privacy protection). For the purposes of security flavor negotiation, order counts: preferred flavors should be listed

first. The order of the sec= option with respect to the other options does not matter, unless you want some options to be enforced differently depending on flavor. In that case you may include multiple sec= options, and following options will be enforced only for access using flavors listed in the immediately preceding sec= option. The only options that are permitted to vary in this way are ro, rw, no_root_squash, root_squash, and all_squash.

General Options

exportfs understands the following export options:

secure This option requires that requests originate on an Internet port less than IPPORT_RESERVED (1024). This option is on by default. To turn it off, specify *insecure*.

rw Allow both read and write requests on this NFS volume. The default is to disallow any request which changes the filesystem. This can also be made explicit by using the *ro* option.

async This option allows the NFS server to violate the NFS protocol and reply to requests before any changes made by that request have been committed to stable storage (e.g. disc drive).

Using this option usually improves performance, but at the cost that an unclean server restart (i.e. a crash) can cause data to be lost or corrupted.

sync Reply to requests only after the changes have been committed to stable storage (see *async* above).

In releases of nfs-utils up to and including 1.0.0, the *async* option was the default. In all releases after 1.0.0, *sync* is the default, and *async* must be explicitly requested if needed. To help make system administrators aware of this change, **exportfs** will issue a warning if neither *sync* nor *async* is specified.

no_wdelay

This option has no effect if *async* is also set. The NFS server will normally delay committing a write request to disc slightly if it suspects that another related write request may be in progress or may arrive soon. This allows multiple write requests to be committed to disc with the one operation which can improve performance. If an NFS server received mainly small unrelated requests, this behaviour could actually reduce performance, so *no_wdelay* is available to turn it off. The default can be explicitly requested with the *wdelay* option.

nohide This option is based on the option of the same name provided in IRIX NFS. Normally, if a server exports two filesystems one of which is mounted on the other, then the client will have to mount both filesystems explicitly to get access to them. If it just mounts the parent, it will see an empty directory at the place where the other filesystem is mounted. That filesystem is "hidden".

Setting the *nohide* option on a filesystem causes it not to be hidden, and an appropriately authorised client will be able to move from the parent to that filesystem without noticing the change.

However, some NFS clients do not cope well with this situation as, for instance, it is then possible for two files in the one apparent filesystem to have the same inode number.

The *nohide* option is currently only effective on *single host* exports. It does not work reliably with netgroup, subnet, or wildcard exports.

This option can be very useful in some situations, but it should be used with due care, and only after confirming that the client system copes with the situation effectively.

The option can be explicitly disabled with *hide*.

crossmnt

This option is similar to *nohide* but it makes it possible for clients to move from the filesystem marked with crossmnt to exported filesystems mounted on it. Thus when a child filesystem "B" is mounted on a parent "A", setting crossmnt on "A" has the same effect as setting "nohide" on B.

no_subtree_check

This option disables subtree checking, which has mild security implications, but can improve reliability in some circumstances.

If a subdirectory of a filesystem is exported, but the whole filesystem isn't then whenever a NFS request arrives, the server must check not only that the accessed file is in the appropriate filesystem (which is easy) but also that it is in the exported tree (which is harder). This check is called the *subtree_check*.

In order to perform this check, the server must include some information about the location of the file in the "filehandle" that is given to the client. This can cause problems with accessing files that are renamed while a client has them open (though in many simple cases it will still work).

subtree checking is also used to make sure that files inside directories to which only root has access can only be accessed if the filesystem is exported with *no_root_squash* (see below), even if the file itself allows more general access.

As a general guide, a home directory filesystem, which is normally exported at the root and may see lots of file renames, should be exported with subtree checking disabled. A filesystem which is mostly readonly, and at least doesn't see many file renames (e.g. /usr or /var) and for which subdirectories may be exported, should probably be exported with subtree checks enabled.

The default of having subtree checks enabled, can be explicitly requested with *subtree_check*.

From release 1.1.0 of nfs-utils onwards, the default will be *no_subtree_check* as subtree_checking tends to cause more problems than it is worth. If you genuinely require subtree checking, you should explicitly put that option in the **exports** file. If you put neither option, **exportfs** will warn you that the change is pending.

insecure_locks

no_auth_nlm

This option (the two names are synonymous) tells the NFS server not to require authentication of locking requests (i.e. requests which use the NLM protocol). Normally the NFS server will require a lock request to hold a credential for a user who has read access to the file. With this flag no access checks will be performed.

Early NFS client implementations did not send credentials with lock requests, and many current NFS clients still exist which are based on the old implementations. Use this flag if you find that you can only lock files which are world readable.

The default behaviour of requiring authentication for NLM requests can be explicitly requested with either of the synonymous *auth_nlm*, or *secure_locks*.

mountpoint=path

mp This option makes it possible to only export a directory if it has successfully been mounted. If no path is given (e.g. *mountpoint* or *mp*) then the export point must also be a mount point. If it isn't then the export point is not exported. This allows you to be sure that the directory underneath a mountpoint will never be exported by accident if, for example, the filesystem failed to mount due to a disc error.

If a path is given (e.g. *mountpoint*=/path or *mp*=/path) then the nominated path must be a mountpoint for the exportpoint to be exported.

fsid=num|root|uuid

NFS needs to be able to identify each filesystem that it exports. Normally it will use a UUID for the filesystem (if the filesystem has such a thing) or the device number of the device holding the filesystem (if the filesystem is stored on the device).

As not all filesystems are stored on devices, and not all filesystems have UUIDs, it is sometimes necessary to explicitly tell NFS how to identify a filesystem. This is done with the *fsid=* option.

For NFSv4, there is a distinguished filesystem which is the root of all exported filesystem. This is specified with *fsid=root* or *fsid=0* both of which mean exactly the same thing.

Other filesystems can be identified with a small integer, or a UUID which should contain 32 hex digits and arbitrary punctuation.

Linux kernels version 2.6.20 and earlier do not understand the UUID setting so a small integer must be used if an fsid option needs to be set for such kernels. Setting both a small number and a UUID is supported so the same configuration can be made to work on old and new kernels alike.

nordirplus

This option will disable READDIRPLUS request handling. When set, READDIRPLUS requests from NFS clients return NFS3ERR_NOTSUPP, and clients fall back on READDIR. This option affects only NFSv3 clients.

refer=path@host[+host][:path@host[+host]]

A client referencing the export point will be directed to choose from the given list an alternative location for the filesystem. (Note that the server must have a mountpoint here, though a different filesystem is not required; so, for example, *mount --bind* /path /path is sufficient.)

replicas=path@host[+host][:path@host[+host]]

If the client asks for alternative locations for the export point, it will be given this list of alternatives. (Note that actual replication of the filesystem must be handled elsewhere.)

User ID Mapping

nfsd bases its access control to files on the server machine on the uid and gid provided in each NFS RPC request. The normal behavior a user would expect is that she can access her files on the server just as she would on a normal file system. This requires that the same uids and gids are used on the client and the server machine. This is not always true, nor is it always desirable.

Very often, it is not desirable that the root user on a client machine is also treated as root when accessing files on the NFS server. To this end, uid 0 is normally mapped to a different id: the so-called anonymous or *nobody* uid. This mode of operation (called ‘root squashing’) is the default, and can be turned off with *no_root_squash*.

By default, **exportfs** chooses a uid and gid of 65534 for squashed access. These values can also be overridden by the *anonuid* and *anongid* options. Finally, you can map all user requests to the anonymous uid by specifying the *all_squash* option.

Here’s the complete list of mapping options:

root_squash

Map requests from uid/gid 0 to the anonymous uid/gid. Note that this does not apply to any other uids or gids that might be equally sensitive, such as user *bin* or group *staff* .

no_root_squash
Turn off root squashing. This option is mainly useful for diskless clients.

all_squash
Map all uids and gids to the anonymous user. Useful for NFS-exported public FTP directories, news spool directories, etc. The opposite option is *no_all_squash*, which is the default setting.

anonuid and *anongid*
These options explicitly set the uid and gid of the anonymous account. This option is primarily useful for PC/NFS clients, where you might want all requests appear to be from one user. As an example, consider the export entry for **/home/joe** in the example section below, which maps all requests to uid 150 (which is supposedly that of user joe).

Extra Export Tables

After reading */etc/exports* **exportfs** reads files under */etc/exports.d.* directory as extra export tables. **exportfs** regards only a file which name is ended with *.exports* and not started with . as an extra export file. A file which name is not met this condition is just ignored. The format for extra export tables is the same as */etc/exports*

EXAMPLE

```
# sample /etc/exports file
/               master(rw) trusty(rw,no_root_squash)
/projects       proj*.local.domain(rw)
/usr            *.local.domain(ro) @trusted(rw)
/home/joe       pc001(rw,all_squash,anonuid=150,anongid=100)
/pub            *(ro,insecure,all_squash)
/srv/www        −sync,rw server @trusted @external(ro)
/foo            2001:db8:9:e54::/64(rw) 192.0.2.0/24(rw)
/build          buildhost[0-9].local.domain(rw)
```

The first line exports the entire filesystem to machines master and trusty. In addition to write access, all uid squashing is turned off for host trusty. The second and third entry show examples for wildcard hostnames and netgroups (this is the entry ‘@trusted’). The fourth line shows the entry for the PC/NFS client discussed above. Line 5 exports the public FTP directory to every host in the world, executing all requests under the nobody account. The *insecure* option in this entry also allows clients with NFS implementations that don’t use a reserved port for NFS. The sixth line exports a directory read-write to the machine ’server’ as well as the ‘@trusted’ netgroup, and read-only to netgroup ‘@external’, all three mounts with the ‘sync’ option enabled. The seventh line exports a directory to both an IPv6 and an IPv4 subnet. The eighth line demonstrates a character class wildcard match.

FILES

/etc/exports /etc/exports.d

SEE ALSO

exportfs(8), **netgroup**(5), **mountd**(8), **nfsd**(8), **showmount**(8).