NFS Server

**Contents:**

**• Executive Summary:**

**• Directories and IP addresses used in these examples**

**• Get the Daemons Running**

**• Configure the NFS Server**

**• Mounting an NFS Share on a Client**

**• Overview**

**• 1: Check the Daemons on the Server**

**• 2: Eyeball the Syntax**

**• 3: Carefully read error messages and develop a symptom description**

**• 4: If it mounts but can't access, check permissions, gid's and uid's**

**• 5: If there are still problems, disable firewalls or log firewalls**

**• 6: If there are still problems, investigate the server's DNS, host name resolution, etc**

**• Summary**

# Executive Summary:

NFS is the best thing since sliced bread. It stands for Network File System. NFS is a file and directory sharing mechanism native to Unix and Linux.  
  
NFS is conceptually simple. On the server (the box that is allowing others to use its disk space), you place a line in /etc/exports to enable its use by clients. This is called *sharing*. For instance, to share /home/myself for both read and write on subnet 192.168.100, netmask 255.255.255.0, place the following line in /etc/exports on the server:

/home/myself 192.168.100.0/24(rw)

To share it read only, change the (rw)to (ro).  
  
On the client wanting to access the shared directory, use the mount command to access the share:

mkdir /mnt/test  
mount -t nfs -o rw 192.168.100.85:/home/myself /mnt/test

The preceding must be performed by user root, or else as a sudo command. Another alternative is to place the mount in /etc/fstab. That will be discussed later in this document.  
  
As mentioned, NFS is conceptually simple, but in practice you'll encounter some truly nasty Overview:

* Non-running NFS or portmap daemons
* Differing uid's for the same usernames
* NFS hostile firewall
* Defective DNS causes timeouts

To minimize troubleshooting time, quickcheck for each of these problems. Each of these Overview is explained in detail in this document.

# Directories and IP addresses used in these examples

The following settings are used in examples on this page:

* Server (computer donating disk space) Settings
  + FQDN hostname = myserver.domain.cxm
  + IP address = 192.168.100.85
  + Netmask = 255.255.255.0
  + RedHat ISO container directory = /data/rh8iso
  + Mandrake RPM container directory = /data/mand9iso
* Client (computer using the donated disk space) settings
  + FDQN hostname = mydesk.domain.cxm
  + IP address = 192.168.100.2
  + Netmask = 255.255.255.0

Please make note of these settings so that you're not confused in the examples.

# Get the Daemons Running

You can't run NFS without the server's portmap and NFS daemons running. This article discusses how to set them to run at boot, and how to check that they're currently running. First you'll use the chkconfig program to set the portmap, nfs and mountd daemons to run at boot. Then you'll check whether these daemons are running. Finally, whether they're running or not, you'll restart these daemons.

## Checking the portmap Daemon

In order to run NFS, the portmap daemon must run. Check for automatic running at boot with the following command:

|  |
| --- |
| [root@myserver root]# **chkconfig --list portmap** portmap 0:off 1:off 2:off 3:on 4:on 5:on 6:off [root@myserver root]# |

Note that in the preceding example, runlevels 3, 4, and 5 say "on". That means that at boot, for runlevels 3, 4 and 5, the portmap daemon is started automatically. If either 3, 4 or 5 say "off", turn them on with the following command:

chkconfig portmap on

Now check that the portmapper is really running, using the ps and grep commands:

|  |
| --- |
| [root@myserver root]# **ps ax | grep portmap**  3171 ? S 0:00 portmap  4255 pts/0 S 0:00 grep portmap You have new mail in /var/spool/mail/root [root@myserver root]# |

The preceding shows the portmap daemon running at process number 3171.

## Checking the NFS Daemon

Next perform the exact same steps for the NFS daemon. Check for automatic run at boot:

|  |
| --- |
| [root@myserver root]# chkconfig --list nfs nfs 0:off 1:off 2:on 3:on 4:on 5:on 6:off [root@myserver root]# |

If either of runlevels 3, 4 or 5 say "off", turn all 3 on with the following command:

chkconfig nfs on

And check that the NFS and the mountd daemons are running as follows:

ps ax | grep nfs  
ps ax | grep mountd

You might get several different different nfs daemons -- that's OK.

## Restarting the Daemons

When learning or troubleshooting, there's nothing so wonderful as a known state. On that theory, restart the daemons before proceeding. Or if you want a truly known state, reboot. Always restart the portmap daemon BEFORE restarting the NFS daemon. Here's how you restart the two daemons:

service portmap restart  
service nfs restart

You'll see messages on the screen indicating if the startups were successful. If they are not, troubleshoot. If they are, continue.  
  
Note that the mountd daemon is started by restarting nfs.  
  
You don't need to restart these daemons every time. Now that you've enabled the daemons on reboot, you can safely assume they're running (unless there's an NFS problem -- then don't make this assumption). From now on the only time you need to restart the NFS daemon is when you change /etc/exports. Theoretically you should never need to restart the portmap daemon unless there's a problem.

## Summary

NFS requires a running portmap daemon and NFS daemon. Use chkconfig to make sure these daemons run at boot, make sure they're running, and the first time, restart the portmap daemon and then the NFS daemon just to be sure you've achieved a known state.

# Configure the NFS Server

Configuring an NFS server is as simple as placing a line in the /etc/exports file. That line has three pieces of information:

1. The directory to be shared (exported)
2. The computer, NIS group, hostname, domain name or subnet allowed to access that directory
3. Any options, such as ro or rw, or several other options

There's one line for each directory being shared. The general syntax is:

directory\_being\_shared subnet\_allowed\_to\_access(options)

Here's an example:

/home/myself 192.168.100.0/24(ro)

In the preceding example, the directory being shared is /home/myself, the subnet is 192.168.100.0/24, and the options are ro(read only). The subnet can also be a single host, in which case there would be an IP address with no bitmask (the /24 in the preceding example). Or it can be an NIS netgroup, in which case the IP address is replaced with @groupname. You can use a wildcard such as ? or \* to replace part of a Fully Qualified Domain Name (FQDN). An example would be \*.accounting.domain.cxm. Do not use wildcards in IP addresses, as they are intermittent in IP addresses.  
  
There are two kinds of options: General options and User ID Mapping options. Read on...

## General Options

Many options can go in the parentheses. If more than one, they are delimited by commas. Here are the common options:

|  |  |  |
| --- | --- | --- |
| **Option** | **What it does** | **Comment** |
| ro | Read Only | The share cannot be written. This is the default. |
| rw | Read Write | The share can be written. |
| secure | Ports under 1024 | Requires that requests originate on a port less  than IPPORT\_RESERVED (1024). This is the default. |
| insecure | Negation of secure |  |
| async | Reply before disk write | Replies to requests before the data is written to disk. This improves performance, but results in lost data if the server goes down. |
| sync | Reply only after disk write | Replies to the NFS request only after all data has been written to disk. This is much safer than async, and is the default in all nfs-utils versions after 1.0.0. |
| no\_wdelay | Write disk as soon as possible | NFS has an optimization algorithm that delays disk writes if NFS deduces a likelihood of a related write request soon arriving. This saves disk writes and can speed performance. **BUT...** If NFS deduced wrongly, this behavior causes delay in every request, in which case this delay should be eliminated. That's what the no\_wdelay option does -- it eliminates the delay. In general, no\_wdelay is recommended when most NFS requests are small and unrelated. |
| wdelay | Negation of no\_wdelay | This is the default. |
| nohide | Reveal nested directories | 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. |
| hide | Negation of nohide | This is the default |
| subtree\_check | Verify requested file is in exported tree | This is the default. Every file request is checked to make sure that the requested file is in an exported subdirectory. If this option is turned off, the only verification is that the file is in an exported filesystem. |
| no\_subtree\_check | Negation of subtree\_check | Occasionally, subtree checking can produce problems when a requested file is renamed while the client has the file open. If many such situations are anticipated, it might be better to set no\_subtree\_check. One such situation might be the export of the /home filesystem. Most other situations are best handed with subtree\_check. |
| secure\_locks | Require authorization for lock requests | This is the default. Require authorization of all locking requests. |
| insecure\_locks | Negation of secure\_locks | Some NFS clients don't send credentials with lock requests, and hence work incorrectly with secure\_locks., in which case you can only lock world-readable files. If you have such clients, either replace them with better ones, or use the insecure\_locks option. |
| auth\_nlm | Synonym for secure\_locks |  |
| no\_auth\_nlm | Synonym for secure\_locks |  |

## User ID Mapping Options

In an ideal world, the user and group of the requesting client would determine the permissions of the data returned. We don't live in an ideal world. Two real-world problems intervene:

1. You might not trust the root user of a client with root access to the server's files.
2. The same username on client and server might have different numerical ID's

Problem 1 is conceptually simple. John Q. Programmer is given a test machine for which he has root access. In no way does that mean that John Q. Programmer should be able to alter root owned files on the server. Therefore NFS offers *root squashing*, a feature that maps uid 0 (root) to the anonymous (nfsnobody) uid, which defaults to -2 (65534 on 16 bit numbers).  
  
So when John Q. Programmer mounts the share, he can access only what the anonymous user and group can access. That means files that are world readable or writeable, or files that belong to either user nfsnobody or group nfsnobodyand allow access by the user or group. One way to do this is to export a chmod 777 directory (booooooo). A better way is to export a directory belonging to user nfsnobody or group nfsnobody, and permissioning accordingly. Now root users from other boxes can write files in that directory, and read the files they write, but they can't read or write files created by root on the server itself.  
  
Now that you know what root squashing is, how do you enable or disable it on a per-share basis? If you want to enable root squashing, that's simple, because it's the default. If you want to disable it, so that root on any box operates as root within the mounted share, disable it with the no\_root\_squash option as follows:

/data/foxwood 192.168.100.0/24(rw,no\_root\_squash)

If, for documentation purposes or to guard against a future change in the default, you'd like to explicitly specify root squashing, use the root\_squash option.  
  
Perhaps you'd like to change the default anonymous user or group on a per-share basis. That way the client's root user can access files within the share as a specific user, let's say user myself. No problem. Use the anonuid or anongid option. The following example uses the anongid option to access the share as group myself., assuming that on the server group myself has gid 655:

/data/wekiva 192.168.100.0/24(rw,anongid=655)

The preceding makes the client's root user group 655 (which happens to be group myself on share /data/wekiva. Files created by the client's root user are user and group 655, but files modified by the client's root are group 655, and a different user.  
  
Now imagine that instead of mapping incoming client root requests to the anonymous user or group, you want ALL incoming NFS requests to be mapped to the anonomous user or the anonymous group. To accomplish that you use the all\_squashoption, as follows:

/data/altamonte 192.168.100.0/24(rw,all\_squash)

You can combine the all\_squash option with the anonuidandanongid options to make directories accessible as if the incoming request was from that user or that group. The one problem with that is that, for NFS purposes, it makes the share world readable and/or world writeable, at least to the extent of which hosts are allowed to mount the share.  
  
We'll get into this subject a little bit more when discussing the Gotcha concerning different user and group id numbers.  
  
The following table lists the User ID Mapping Options:

|  |  |  |
| --- | --- | --- |
| **Option** | **What it does** | **Comment** |
| root\_squash | Convert incoming requests from user root to the anonymous uid and gid. | This is the default. |
| no\_root\_squash | Negation of root\_squash |  |
| anonuid | Set anonymous user id to a specific id | The id is a number, not a name. This number can be obtained by this command on the server:  grep myself /etc/passwd  Where myself is the username whose uid you want to find. |
| anongid | Set anonymous group id to a specific id | The id is a number, not a name. This number can be obtained by this command on the server:  grep myself /etc/group  Where myself is the name of the group whose uid you want to find. |
| all\_squash | Convert incoming requests, from ALL users, to the anonymous uid and gid. | Remember that this gives all incoming users the same set of rights to the share. This may not be what you want. |

# Mounting an NFS Share on a Client

Mounting an NFS share on a client can be simple. At its simplest it might look like this:

mount -t nfs -o ro 192.168.100.85:/data/altamonte /mnt/test

The English translation of the preceding is this: mount type (-t) nfs with options (-o) read only (ro) server 192.168.100.85's directory /data/altamonteat mount point /mnt/test. What usually changes is the comma delimited list of options (-o). For instance, NFS typically performs better with rsize=8192and wsize=8192. These are the read and write buffer sizes, and it's been found that in general 8192 performs better than the default 4096. Thehard option keeps the request alive even if the server goes down, whereas the softoption enables the mount to time out if the server goes down. The hardoption has the advantage that whenever the server comes back up, the file activity continues where it left off.  
  
Besides these and a few other NFS specific options, there are filesystem independent options such as async/sync/dirsync, atime/noatime, auto/noauto, defaults,dev/nodev, exec/noexec, \_netdev, remount, ro, rw, suid/nosuid, user/nouser.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Option** | **Action** | **Default?** | **Comment** | **Negation option** |
| async | All I/O done asynchronously | Y | Better performance, more possiblity of corruption when things crash. Do not use when the same file is being modified by different users. | sync |
| sync | All I/O done synchronously | N | Less likelihood of corruption, less likelihood of overwrite by other users. | async |
| dirsync | All I/O to directories done synchronously | N |  |  |
| atime | Update inode access time for each  access. | Y |  | noatime |
| auto | Automatic mounting. | Y | Can be mounted with the -a option. Mounted at boot time. | noauto |
| defaults | Shorthand for default options. |  | rw,suid,dev,exec,auto,nouser,async. |  |
| dev | Device | Y | Interpret character or block special devices on the  file system. | nodev |
| exec | Permit execution of binaries. | Y |  | noexec |
| \_netdev | Device requires network. |  | The  device holding the filesystem requires network access. Do not mount until the network has been enabled. |  |
| remount | Remount a mounted system. |  | Used to change the mount flags, especially to toggle between rw and ro. |  |
| ro | Allow only read access. | N | Used to protect the mounted filesystem from writes. Even if the filesystem is writeable by the user, and is exported writeable, this still protects it. | rw |
| rw | Allow both read and write. | Y | Allow writing to the filesystem, assuming that the system is writeable by the user and has been exported writeable. | ro |
| suid | Allow set-user-identifier and/or set-group-identifier bits to take effect. | Y |  | nosuid |
| user | Allow mounting by ordinary user. | N | When used in /etc/fstab, this allows mounting by an ordinary user. Only the user performing the mount can unmount it. | nouser |
| users | Allow mounting and dismounting by arbitrary user. | N | When used in /etc/fstab, this allows mounting by an ordinary user. Any user can unmount it at any time, regardless of who initially mounted it. |  |

### /etc/fstab syntax

Like any other mount, NFS mounting can be done in /etc/fstab. The advantages to placing it in /etc/fstab are:

* It can be mounted automatically (auto) either with mount -a or on boot.
* It can easily be configured to be mountable by ordinary users (user or users).
* The mount is documented in /etc/fstab.

The disadvantages to placing a mount in /etc/fstab are:

* /etc/fstab can become cluttered by too many mounts.
* The mountpoint cannot be used for different filesystems.

The following example shows an NFS mount:

192.168.100.85:/home/myself /mnt/test nfs users,noauto,rw 0 0

The preceding is a typical example. Just like other /etc/fstab mounts, NFS mounts in /etc/fstab have 6 columns, listed in order as follows:

1. The filesystem to be mounted (192.168.100.85:/home/myself)
2. The mountpoint (/mnt/test)
3. The type of the filesystem (nfs)
4. The options (users,noauto,rw)
5. Frequency to be dumped (a backup method)  (0)
6. Order in which to be fsck'ed at boot time.  (0). The root filesystem should have a value of 1 so it gets fsck'ed first. Others should have 2 or more so they get fsck'ed later. A value of 0 means don't perform the fsck at all.

## Summary

The server exports a share, but to use it the client must mount that share. The mount is performed with a mount command, like this:

mount -t nfs -o rw 192.168.100.85:/data/altamonte /mnt/test

That same mount can be performed in /etc/fstab with the following syntax:

192.168.100.85:/data/altamonte /mnt/test nfs rw 0 0

There are many mount options that can be used, and those are listed in this article.

# Overview

If you've worked with NFS, you know it's not that simple. Often times the mount fails, times out, or takes so long as to discourage use. Sometimes the mount succeeds but the data is inaccessible. These problems can be a bear to troubleshoot.  
  
To make troubleshooting easier this article lists the usual causes of NFS failure, ways to quickly check whether these problems are the cause, and methods to overcome these problems. Here are the typical causes of NFS problems:

* The portmap or nfs daemons are not running
* Syntax error on client mount command or server /etc/exports
  + A space between the mount point and the (rw) causes the (rw) to be ignored.
* Problems with permissions, uid's and gid's
* Firewalls filtering packets necessary for NFS. The offending firewall is typically on the server, but it could also be on the client.
* Bad DNS on server (including /etc/resolv.conf on the server).

|  |
| --- |
| **!! WARNING !!**  Always restart the nfs service after making a change to /etc/exports. Otherwise your changes will not be recognized, leading you down a long and winding dead end. |

|  |  |
| --- | --- |
| **Cause category** | **Symptom** |
| The portmap or nfs daemons are not running | Typically, failure to mount |
| Syntax error on client mount command or server's /etc/exports | Typically, failure to mount or failure to write enable. A space between the mount point and the (rw) causes the share to be read-only -- a frustrating and hard to diagnose problem. |
| Problems with permissions, uid's and gid's | Mounts OK, but access to the data is impossible or not as specified |
| Firewalls filtering packets necessary for NFS | Mount failures, timeouts, excessively slow mounts, or intermittent mounts |
| Bad DNS on server | Mount failures, timeouts, excessively slow mounts, or intermittent mounts |

Here's your predefined diagnostic:

1. Check the daemons on the server
2. Eyeball the syntax of the client mount command and the server /etc/exports. Pay particular attention that the mountpoint is NOT separated from the parenthasized options list, because a space between the mountpoint and the opening paren causes the options to be ignored.
3. Carefully read error messages and develop a symptom description
4. If the symptom involves successful mounts but you can't correctly access the data, check permissions, gid's and uid's. Correct as necessary.
5. If there are still problems, disable firewalls or log firewalls.
6. If there are still problems, investigate the server's DNS, host name resolution, etc.

For maximum diagnostic speed, quickly check that the portmap and nfs daemons are running on the server. If not, investigate why not. Next, eyeball the syntax on the client's mount command and the server's /etc/exports file. Look for not only bad syntax, but wrong information such as wrong IP addresses, wrong filesystem directories, and wrong mountpoints. If you find bad syntax, correct it. These two steps should take no more than 3 minutes, and will find the root cause in many cases.  
  
Next, carefully read the error message, and formulate a symptom description. Try to determine whether the mount has succeeded. If the mount succeeded but you can't access the data, it's likely a problem with permissions, uid's or gid's. Investigate that. If the mount succeeds but it's slow, investigate firewalls and DNS. A healthy NFS system should mount instantaneously. By the time you lift your finger off the Enter key, the mount should have been completed. If it takes more than one second, there's a problem that bears investigation.  
  
The hardest problems are those in which you experience mount failures, timeouts, excessively slow mounts, or intermittent mounts. In such situations, it's likely either a firewall problem or a server DNS problem. Investigate those.  
  
Each of these problem categories is discussed in an article later in this document.

# 1: Check the Daemons on the Server

This will take you all of a minute. Perform the following 2 commands on the server:

ps ax | grep portmap  
ps ax | grep nfs

If either shows nothing (or if it shows just the grep command), that server is not running. Investigate why. Start by seeing if it's even set to run at boot:

/sbin/chkconfig --list portmap  
/sbin/chkconfig --list nfs

Each command will output a line showing the run levels at which the command is on. If either one is not on at any runlevel between 3 and 5 inclusive, turn it on with one or both of these commands:

/sbin/chkconfig portmap on  
/sbin/chkconfig nfs on

The preceding commands set it to fire at boot, but do not run the daemon. You must run them manually:

service portmap restart  
service nfs restart

Always restart the portmap daemon before restarting the nfs daemon, because NFS needs the portmapper to function. If either of those commands fails or produces an error message, investigate.  
  
**IMPORTANT NOTE:** Even if the daemons were both running when you investigated, restart them both anyway. First, you might see an error message. Second, it's always nice to achieve a known state. Restarting these two daemons should take a minute. That one minute is a tiny price to pay for the peace of mind you achieve knowing that there's no undiscovered problem with the daemons.  
  
If NFS fails to start, investigate the syntax in /etc/exports, and possibly comment out everything in that file, and try another restart. If that changes the symptom, divide and conquer. If restarting NFS takes a huge amount of time, investigate the server's DNS.

# 2: Eyeball the Syntax

If the daemons work, eyeball the syntax of the mount command on the client and the /etc/exports file on the server. Obviously, if you use the wrong syntax (or wrong IP addresses or directories) in your mount command, the mount fails. You needn't take a great deal of time -- just verify that the syntax is correct and you're using the correct IP addresses, directories and mount points. Correct as necessary, and retest.  
  
Pay SPECIAL attention to make sure there is no space between the mountpoint and the opening paren of the options list. A space between them causes the options to be ignored -- clearly not what you want. If you can't figure out why a mount is read-only, even though the client mount command specifies read-write and the server's directory is clearly read-write with the correct user and group (not a number, but an actual name), suspect this intervening space.

|  |
| --- |
| **!! WARNING !!**  Always restart the nfs service after making a change to /etc/exports. Otherwise your changes will not be recognized, leading you down a long and winding dead end. |

# 3: Carefully read error messages and develop a symptom description

The first two steps were general maintenance -- educated guesses designed to yield quick solutions. If they didn't work, it's time to buckle down and troubleshoot. The first step is to read the error message, and learn more about it. You might want to check the system logs (start with /var/log/messages) in case relevent messages were written.  
  
Try several mounts and umounts, and note exactly what the malfunction looks like:

* Does the mount produce an error message?
* Does the mount time out?
* Does the mount appear to hang forever (more than 5 minutes)?
* Does the mount appear to succeed, but the data can't be seen, read or written as expected?
* Does the symptom change over time, or with reboots?

The more you learn and record about the symptom, the better your chances of quickly and accurately solving the problem.

# 4: If it mounts but can't access, check permissions, gid's and uid's

Generally speaking, the permissions on the server don't affect the mounting or unmounting of the NFS share. But they very much affect whether such a share can be seen, executed, read or written. Often the cause is obvious. If the directory is owned by root, permissioned 700, it obviously can't be read and written by user myself.  This type of problem is easy to diagnose and fix.  
  
Tougher are root squashing problems. You access an NFS share as user root, and yet you can't see the mounted share or its contents. You need to remember this is probably happening because on the server you're operating not as root, but as the anonomous user. A quick test can be done by changing the server's export to export to a no\_root\_squash and single IP address (for security). If the problem goes away, it's a root squashing problem. Either access it as non-root, or change the ownership of the directory and contents to the anonomous gid or uid.  
  
By far the toughest problems are caused by non-matching uid's and gid's. Let's say you share your home directory on the server, and you log in as yourself on the client and mount that share. It mounts ok (we'll assume you used su -c or sudo to mount it), but you can't read the data -- permission denied!  
  
That's crazy. The directory you're sharing is owned by myself, and you're logged into the client as myself, and yet you don't have permission to read. What's up?  
  
It turns out that under the hood, NFS requests contain numeric uid's and gid's, but not actual usernames or groupnames. What that means is that if user myself is uid 555 on the server, but uid 600 on the client, you're trying to access files owned by uid 555 when you're uid 600. That means your only rights to the mounted material are permissions granted to "other" -- not to "user" or "group".  
  
The best solution to this problem is to create a system in which all boxes on your network have the same uid for each username and the same gid for each groupname. This can be accomplished either by attention to detail, by using NIS to assign users and groups, or by using some other authentication scheme yielding global users and groups.  
  
If you cannot have a single uid for all instances of a username, suboptimal steps must be taken. In some instances you could make the directory and files world-readable, thereby enabling all users to read it. It could also be made world-writeable, but that's always a bad idea. It could be mounted all\_squash with a specific anonuid and/or a specific anongid to cure the problem, but once again, at least from the NFS viewpoint, that's equivalent to making it world readable or writeable.  
  
If you have problems accessing mounts, always check the gid's and uid's on both sides and make sure they match. If they don't, find a way of fixing it. Sometimes it's as simple as editing /etc/passwd and /etc/group to change the numeric ID's on one or both sides. Remember that if you do that, you need to perform the proper chown command on any files that were owned or grouped by the owner and/or group that you renumbered. A dead giveaway are files that are listed with numbers rather than names for group and user.

# 5: If there are still problems, disable firewalls or log firewalls

Many supposed NFS problems are really problems with the firewall. In order for your NFS server to successfully serve NFS shares, its firewall must enable the following:

* ICMP Type 3 packets
* Port 111, the Portmap daemon
* Port 2049, NFS
* The port(s) assigned to the mountd daemon

The easiest way to see whether your problem resides in the firewall is to completely open up the client and server firewalls and anything in between. For details on how to manipulate iptables see the [May 2003 Linux Productivity Magazine](http://www.troubleshooters.com/lpm/200305/200305.htm).  
  
Note that opening up firewalls is appropriate only if you're disconnected from the Internet, or if you're in a very un-hostile environment. Even so, you should open up the firewalls for a very short time (less than 5 minutes). If in doubt, instead of opening the firewalls, insert logging statements in IPTables to show what packets are being rejected during NFS mounts, and take action to enable those ports. For details on IPTables diagnostic logging, see the [May 2003 Linux Productivity Magazine](http://www.troubleshooters.com/lpm/200305/200305.htm).  
  
The mountd daemon ports are especially problematic, because they're normally assigned by the portmap daemon, and vary from NFS restart to NFS restart. The /etc/rc.d/init.d/nfs script can be changed to nail down the mountd daemon to a specific port, which then enables you to pinhole a specific port. The [A Somewhat Practical Server Firewall](http://www.troubleshooters.com/lpm/200305/200305.htm#_A_Somewhat_Practical_Server_Firewall) article in the [May 2003 Linux Productivity Magazine](http://www.troubleshooters.com/lpm/200305/200305.htm). explains how to do this.  
  
If for some reason you don't want to nail down the port, your only other alternatives are to create a firewall enabling a huge range of ports in the 30000's, or to create a master NFS restart script which does the following:

1. Use the rcpinfo program to find all ports used by mountd.
2. Issue iptables commands to find the rule numbers for those ports.
3. Issue iptables commands to delete all rules on those ports.
4. Restart NFS
5. Use the rcpinfo program to find all ports used by mountd.
6. Issue iptables commands to insert rules for those ports where the rules for those ports used to be.

One technique that might make that easier is to create a user defined chain just to hold mountd rules. In that case you'd simply empty that chain, restart NFS, use rpcinfo to find the port numbers, and add the proper rules using the iptables -A command.  
  
It bears repeating that the [May 2003 Linux Productivity Magazine](http://www.troubleshooters.com/lpm/200305/200305.htm) details how to create an NFS friendly firewall.

# 6: If there are still problems, investigate the server's DNS, host name resolution, etc

Bad forward and reverse name resolution can mess up any server app, including NFS. Like other apps, bad DNS most often results in very slow performance or timeouts. Be sure to check your /etc/resolv.conf and make sure you're querying the correct DNS server. Check your DNS server with DNSwalk or DNS lint or another suitable utility.

# Summary

NFS is wonderful. It's a convenient and lightning fast way to use a network. Although it's not particularly secure, its security can be beefed up with firewalls. Its security can also be strengthened by authentication schemes.  
  
Although conceptually simple, NFS often requires overcoming troubleshooting challenges before a working system is achieved. Here's a handy predefined diagnostic:

1. Check the daemons on the server
2. Eyeball the syntax of the client mount command and the server /etc/exports
3. Carefully read error messages and develop a symptom description
4. If the symptom involves successful mounts but you can't correctly access the data, check permissions, gid's and uid's. Correct as necessary.
5. If there are still problems, disable firewalls or log firewalls.
6. If there are still problems, investigate the server's DNS, host name resolution, etc.

If you suspect firewall problems are stopping your NFS, see the [May 2003 Linux Productivity Magazine](http://www.troubleshooters.com/lpm/200305/200305.htm) , which details IPTables and how to create an NFS-friendly firewall.