How To Set Up an NFS Mount on Ubuntu

The server that shares its directories will be referred to as the **host** and the server that mounts these directories as the **client**. You will need to use the IP address for both.

Step 1 — Downloading and Installing the Components

You'll begin by installing the necessary components on each server.

On the Host

On the **host** server, install the `nfs-kernel-server` package, which will allow you to share your directories. Since this is the first operation that you're performing with `apt` in this session, refresh your local package index before the installation:

```
sudo apt update
sudo apt install nfs-kernel-server
```

Once these packages are installed, switch to the **client** server.

On the Client

On the **client** server, you need to install a package called `nfs-common`, which provides NFS functionality without including any server components. Again, refresh the local package index prior to installation to ensure that you have up-to-date information:

```
sudo apt update
sudo apt install nfs-common
```

Now that both servers have the necessary packages, you can start configuring them.

Step 2 — Creating the Share Directories on the Host

You're going to share two separate directories, with different configuration settings, in order to illustrate two key ways that NFS mounts can be configured with respect to superuser access.

Superusers can do anything anywhere on their system. However, NFS-mounted directories are not part of the system on which they are mounted, so by default, the NFS server refuses to perform operations that require superuser privileges. This default restriction means that superusers on the **client** cannot write files as **root**, reassign ownership, or perform any other superuser tasks on the NFS mount.

Sometimes, however, there are trusted users on the **client** system who need to perform these actions on the mounted file system but who have no need for superuser access on the **host**. You can configure the NFS server to allow this, although it introduces an element of risk, as such a user *could* gain root access to the entire **host** system.

Example 1: Exporting a General Purpose Mount

In the first example, you'll create a general-purpose NFS mount that uses default NFS behavior to make it difficult for a user with root privileges on the **client** machine to interact with the **host** using those **client** superuser privileges. You might use something like this to store files which were uploaded using a content management system or to create space for users to easily share project files.

First, make the share directory:

```
//host
sudo mkdir /var/nfs/general -p
```

Since you're creating it with `sudo`, the directory is owned by the **host**'s **root** user:

```
//host
ls -dl /var/nfs/general
```

```
Output
drwxr-xr-x 2 root root 4096 Apr 17 23:51 /var/nfs/general
```

NFS will translate any **root** operations on the **client** to the `nobody:nogroup` credentials as a security measure. Therefore, you need to change the directory ownership to match those credentials.

```
//host
sudo chown nobody:nogroup /var/nfs/general
```

```
Output
drwxr-xr-x 2 nobody nogroup 4096 Apr 17 23:51 /var/nfs/general
```

You're now ready to export this directory.

Example 2: Exporting the Home Directory

In the second example, the goal is to make user home directories stored on the **host** available on **client** servers, while allowing trusted administrators of those **client** servers the access they need to conveniently manage users.
To do this, you'll export the `/home` directory. Since it already exists, you don't need to create it. You won't change the permissions, either. If you *did*, it could lead to a range of issues for anyone with a home directory on the **host** machine.

Step 3 — Configuring the NFS Exports on the Host Server

Next, you'll dive into the NFS configuration file to set up the sharing of these resources.

On the **host** machine, open the `/etc/exports` file in your text editor with **root** privileges:

```
sudo nano /etc/exports
```

The file has comments showing the general structure of each configuration line. The syntax is as follows:

/etc/exports

```
directory_to_share    client(share_option1,...,share_optionN)
```

You'll need to create a line for each of the directories that you plan to share. Be sure to change the `client_ip` placeholder shown here to your actual IP address:

/etc/exports

```
/var/nfs/general    client_ip(rw,sync,no_subtree_check)
/home               client_ip(rw,sync,no_root_squash,no_subtree_check)
```

Here, you're using the same configuration options for both directories with the exception of `no_root_squash`. Take a look at what each of these options mean:

- `rw`: This option gives the **client** computer both read and write access to the volume.
- `sync`: This option forces NFS to write changes to disk before replying. This results in a more stable and consistent environment since the reply reflects the actual state of the remote volume. However, it also reduces the speed of file operations.
- `no_subtree_check`: This option prevents subtree checking, which is a process where the **host** must check whether the file is actually still available in the exported tree for every request. This can cause many problems when a file is renamed while the **client** has it opened. In almost all cases, it is better to disable subtree checking.
- `no_root_squash`: By default, NFS translates requests from a **root** user remotely into a non-privileged user on the server. This was intended as security feature to prevent a **root** account on the **client** from using the file system of the **host** as **root**. `no_root_squash` disables this behavior for certain shares.

When you are finished making your changes, save and close the file. Then, to make the shares available to the clients that you configured, restart the NFS server with the following command:

```
//host
sudo systemctl restart nfs-kernel-server
```

Before you can actually use the new shares, however, you'll need to be sure that traffic to the shares is permitted by firewall rules.

Step 4 — Adjusting the Firewall on the Host (Ubuntu 22 only)

Don't work with the Firewall and **skip Step 4 completely if you're using Ubuntu 20**.

First, check the firewall status to see if it's enabled and, if so, to see what's currently permitted:

```
//host
sudo ufw status
```

```
Output
Status: active
```

```
To                         Action      From
--                         ------      ----
OpenSSH                    ALLOW       Anywhere
OpenSSH (v6)               ALLOW       Anywhere (v6)
```

On your system, only SSH traffic is being allowed through, so you'll need to add a rule for NFS traffic.

With many applications, you can use `sudo ufw app list` and enable them by name, but `nfs` is not one of those. However, because `ufw` also checks `/etc/services` for the port and protocol of a service, you can still add NFS by name. Best practice recommends that you enable the most restrictive rule that will still allow the traffic you want to permit, so rather than enabling traffic from just anywhere, you'll be specific.
Use the following command to open port `2049` on the **host**, being sure to substitute your **client** IP address:

```
//host
sudo ufw allow from client_ip to any port nfs
```

You can verify the change by typing:

```
//host
sudo ufw status
```

You should see traffic allowed from port `2049` in the output:

```
Output
Status: active

To                         Action      From
--                         ------      ----
OpenSSH                    ALLOW       Anywhere
2049                       ALLOW       203.0.113.24
OpenSSH (v6)               ALLOW       Anywhere (v6)
```

This confirms that UFW will only allow NFS traffic on port `2049` from your **client** machine.

Step 5 — Creating Mount Points and Mounting Directories on the Client

Now that the **host** server is configured and serving its shares, you'll prepare your **client**.
In order to make the remote shares available on the **client**, you need to mount the directories on the **host** that you want to share to empty directories on the **client**.
Note: If there are files and directories in your mount point, they will become hidden as soon as you mount the NFS share. To avoid the loss of important files, be sure that if you mount in a directory that already exists that the directory is empty.

You'll create two directories for your mounts:

```
//client
sudo mkdir -p /nfs/general
sudo mkdir -p /nfs/home
```

Now that you have a location to put the remote shares and you've opened the firewall, you can mount the shares using the IP address of your **host** server:

```
//client
sudo mount host_ip:/var/nfs/general /nfs/general
sudo mount host_ip:/home /nfs/home
```

These commands will mount the shares from the host computer onto the **client** machine. You can double-check that they mounted successfully in several ways. You can check this with a `mount` or `findmnt` command, but `df -h` provides a more readable output:

```
//client
df -h
```

```
Output
Filesystem                   Size  Used Avail Use% Mounted on
tmpfs                        198M  972K  197M   1% /run
/dev/vda1                     50G  3.5G   47G   7% /
tmpfs                        989M     0  989M   0% /dev/shm
tmpfs                        5.0M     0  5.0M   0% /run/lock
/dev/vda15                   105M  5.3M  100M   5% /boot/efi
tmpfs                        198M  4.0K  198M   1% /run/user/1000
10.124.0.3:/var/nfs/general   25G  5.9G   19G  24% /nfs/general
10.124.0.3:/home              25G  5.9G   19G  24% /nfs/home
```

Both of the shares you mounted appear at the bottom. Because they were mounted from the same file system, they show the same disk usage. To see how much space is actually being used under each mount point, use the disk usage command `du` and the path of the mount. The `-s` flag provides a summary of usage rather than displaying the usage for every file. The `-h` prints human-readable output.

For example:

```
//client
du -sh /nfs/home
```

```
Output
36K    /nfs/home
```

This shows us that the contents of the entire home directory is using only 36K of the available space.

Step 6 — Testing NFS Access

Next, test access to the shares by writing something to each of them.

Example 1: The General Purpose Share

First, write a test file to the `/var/nfs/general` share:

```
//client
sudo touch /nfs/general/general.test
```

Then, check its ownership:

```
//client
ls -l /nfs/general/general.test
```

```
Output
-rw-r--r-- 1 nobody nogroup 0 Apr 18 00:02 /nfs/general/general.test
```

Because you mounted this volume without changing NFS's default behavior and created the file as the **client** machine's **root** user via the `sudo` command, ownership of the file defaults to `nobody:nogroup`. **client** superusers won't be able to perform typical administrative actions, like changing the owner of a file or creating a new directory for a group of users, on this NFS-mounted share.

Example 2: The Home Directory Share

To compare the permissions of the General Purpose share with the Home Directory share, create a file in `/nfs/home` the same way:

```
//client
sudo touch /nfs/home/home.test
```

Then look at the ownership of the file:

```
//client
ls -l /nfs/home/home.test
```

```
Output
-rw-r--r-- 1 root root 0 Apr 18 00:03 /nfs/home/home.test
```

You created `home.test` as **root** using the `sudo` command, exactly the same way you created the `general.test` file. However, in this case it is owned by **root** because you overrode the default behavior when you specified the `no_root_squash` option on this mount. This allows your **root** users

on the **client** machine to act as **root** and makes the administration of user accounts much more convenient. At the same time, it means you don't have to give these users root access on the **host**.

Step 7 — Mounting the Remote NFS Directories at Boot

You can mount the remote NFS shares automatically at boot by adding them to `/etc/fstab` file on the **client**.

Open this file with root privileges in your text editor:

```
//client
sudo nano /etc/fstab
```

At the bottom of the file, add a line for each of your shares. They will look like this:

/etc/fstab

```
. . .
host_ip:/var/nfs/general    /nfs/general   nfs auto,nofail,noatime,nolock,intr,tcp,actimeo=1800 0 0
host_ip:/home               /nfs/home      nfs auto,nofail,noatime,nolock,intr,tcp,actimeo=1800 0 0
```

Note: You can find more information about the options you are specifying here in the NFS man page. You can access this by running the following command:

```
man nfs
```

The **client** will automatically mount the remote partitions at boot, although it may take a few moments to establish the connection and for the shares to be available.

Step 8 — Unmounting an NFS Remote Share

If you no longer want the remote directory to be mounted on your system, you can unmount it by moving out of the share's directory structure and unmounting, like this:

```
//client
cd ~
sudo umount /nfs/home
sudo umount /nfs/general
```

Take note that the command is named `umount` not `unmount` as you may expect.

This will remove the remote shares, leaving only your local storage accessible:

```
//client
df -h
```

```
Output
```

```
Filesystem      Size  Used Avail Use% Mounted on
tmpfs           198M  972K  197M   1% /run
/dev/vda1        50G  3.5G   47G   7% /
tmpfs           989M     0  989M   0% /dev/shm
tmpfs           5.0M     0  5.0M   0% /run/lock
/dev/vda15      105M  5.3M  100M   5% /boot/efi
tmpfs           198M  4.0K  198M   1% /run/user/1000
```

If you also want to prevent them from being remounted on the next reboot, edit `/etc/fstab` and either delete the line or comment it out by placing a `#` character at the beginning of the line. You can also prevent auto-mounting by removing the `auto` option, which will allow you to still mount it manually.