Linux NFS Client/Server Lab
By Michael Ambeguia

Purpose: Having a collaborative file sharing ability is a useful utility for companies to have. NFS is one way companies can implement this functionality into their IT environment. NFS allows companies to share documents for editing and reading across the network from a single server. The purpose of this lab is to gain hands-on experience implementing this commonly used Linux service. This lab will demonstrate how to configure and run an NFS server on a Linux server and how to set up the client devices as well. Other pertinent topics such as how to secure NFS and troubleshoot its issue are demonstrated as well.

Topics covered:

1. NFS server configuration
2. NFS client configuration
3. Managing NFS exports and mounts
4. Troubleshooting NFS issues
5. Understanding NFS permissions and security

Sections:

Section #1 Introduction to NFS:

1.1a. What is NFS? How does it work?

NFS is a service that allows a system to share directories and files over the network. Users that access the remote files using NFS access files as though they were located locally on the filesystem.

This is how NFS works:

Client Request:

1. The client device requests to perform an action on a file such as opening, reading, writing, or closing a file.
2. The NFS client translates the request into an RPC (Remote Procedure Call) and sends the RPC over the network to the NFS server.

Server Processing:

1. The NFS server receives the RPC request and processes it. This involves interpreting the request and performing the appropriate file system operations.
2. The server accesses its local file system to perform the requested operation, such as reading from or writing to the file.

3. The server prepares a response, which includes the result of the operation (e.g., file data for a read request) or an acknowledgment.
4. The server sends the response back to the NFS client over the network.

Client Response Handling:

1. The NFS client receives the server's response and processes it.
2. The client application completes the file operation based on the response received from the server.

Let's take a look at how this would work in a hypothetical scenario:

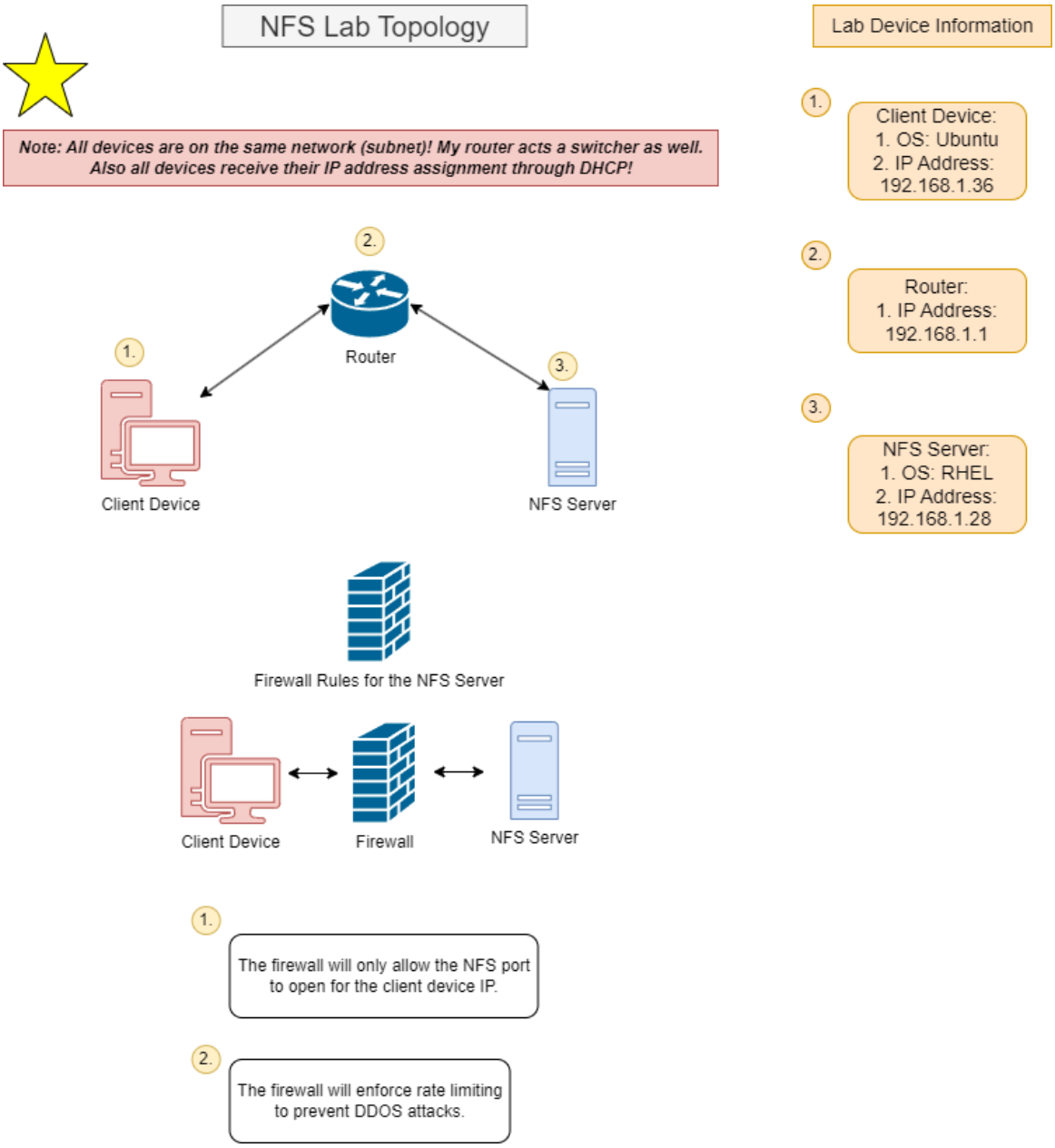
Say an Ubuntu client requests to read the file testfile.txt on an NFS server on the network it is on. The Client device will convert this read request into an RPC message and send it across the network to the NFS server. When the NFS server receives the RPC message from the Ubuntu client it will first interpret the RPC message. Once the NFS server finds out that the Ubuntu client requested to read testfile.txt, it will perform that action on its own filesystem. Once the read operation is done the NFS server creates a response to send back to the client, and this response will be the read data from testfile.txt. The client will receive the response back and will be able to view the data from testfile.txt. ***NFS works by having the server perform the requested task, and sending the result back to the client device. It might appear as though you are actually viewing the file or writing to it, but it is all an illusion since the server is the one performing the task.***

1.1b. Why is NFS used?

NFS is used to provide a convenient way for users to access files on a remote server. The benefits of using NFS are the following:

1. Workstations and client devices won't have to use up disk space storing commonly used files. You can have a centralized storage location for key files used by multiple departments or groups.
2. NFS works across different systems. NFS can work across Linux, MacOS, and Windows OS making it a great service to use in a mixed computing environment.
3. NFS allows a company to control their data. Unlike cloud file sharing services like Google Cloud or DropBox the company does not have to store their data on a third party company's server. This makes NFS a useful service for sensitive industries like defense, government, and healthcare where strict data security rules must be implemented.

1.1d. Lab Topology:

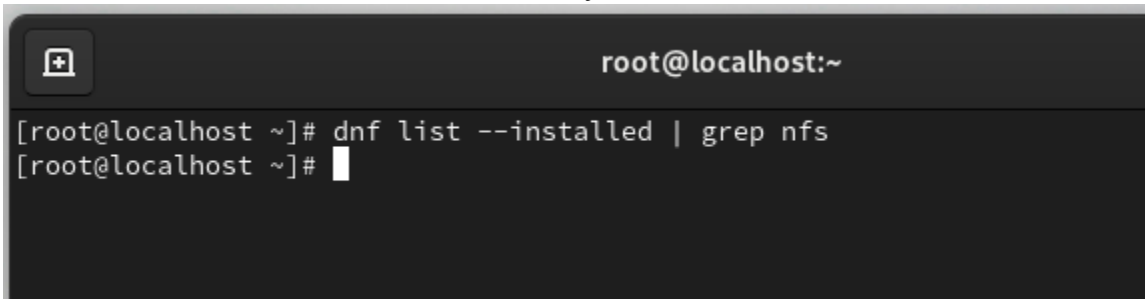


In this lab I will use my RHEL vm as the NFS server and my Ubuntu vm as the NFS client. These devices are on my local network on the same subnet. For communication between the devices my router is used as a switch since there is no need for routing in between networks.

Section #2 NFS Server Setup:

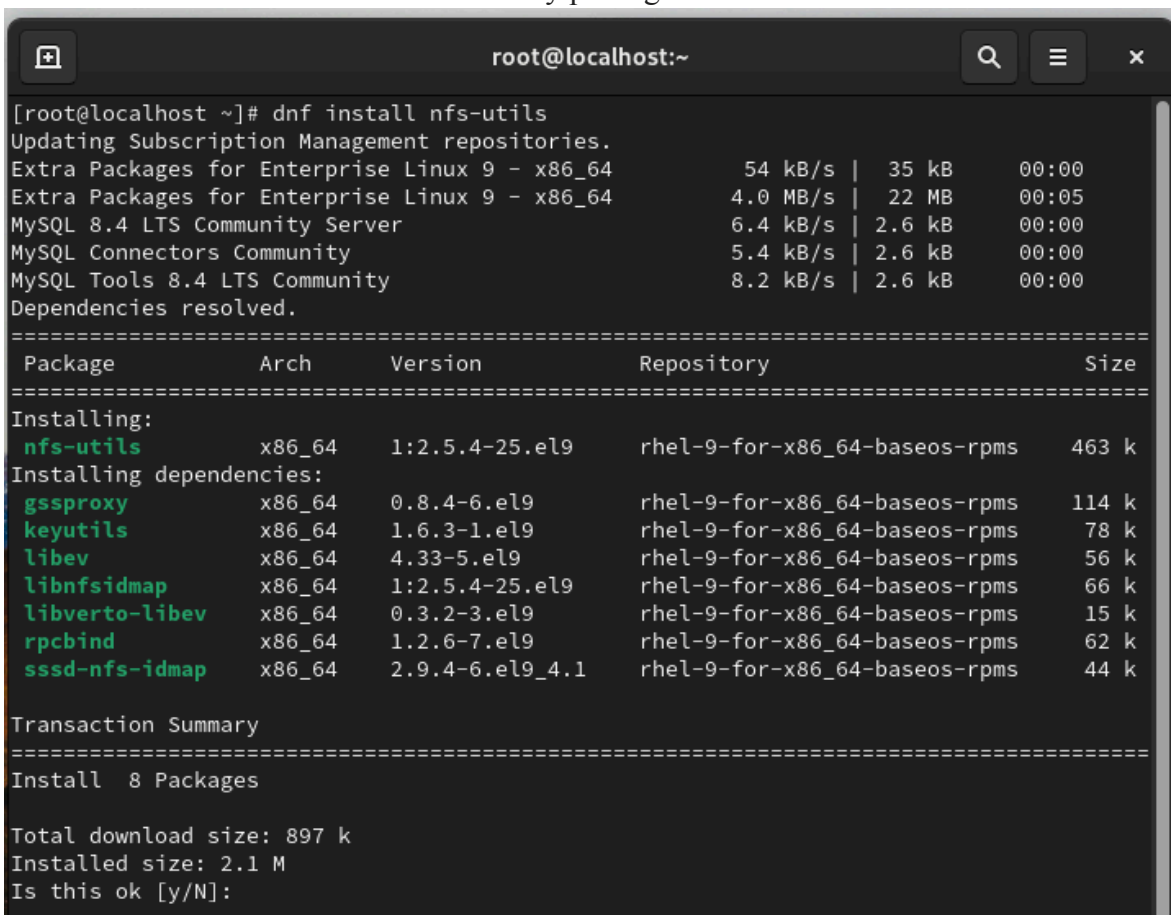
2.1. Make sure necessary packages are installed and start the NFS Service. Set it to run at boot time:

2.1a I will first check if NFS is installed on my RHEL vm. If it is not installed I will install it.

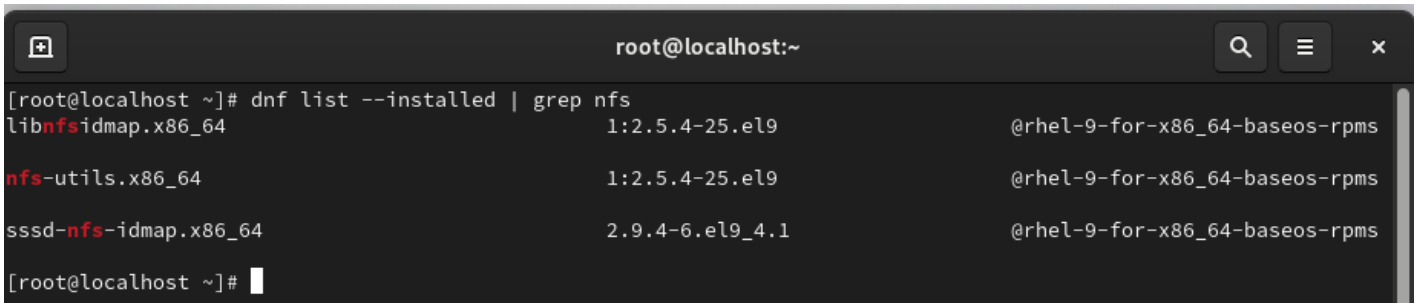


I checked if any packages related to NFS are installed, and there are none installed.

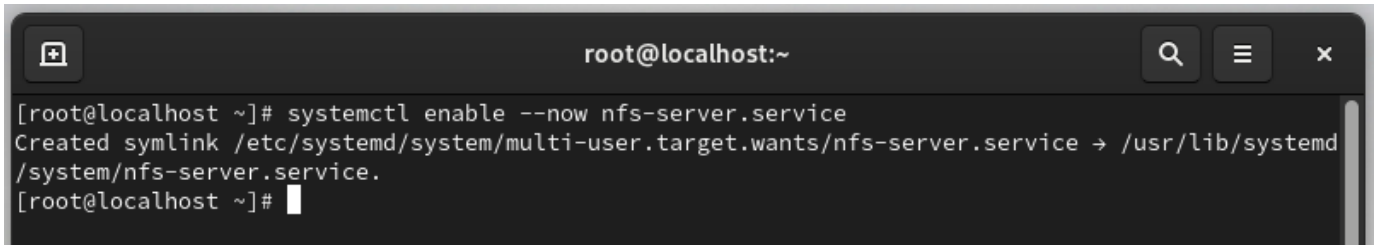
2.1b. Now I will install NFS. The necessary package is called nfs-utils.



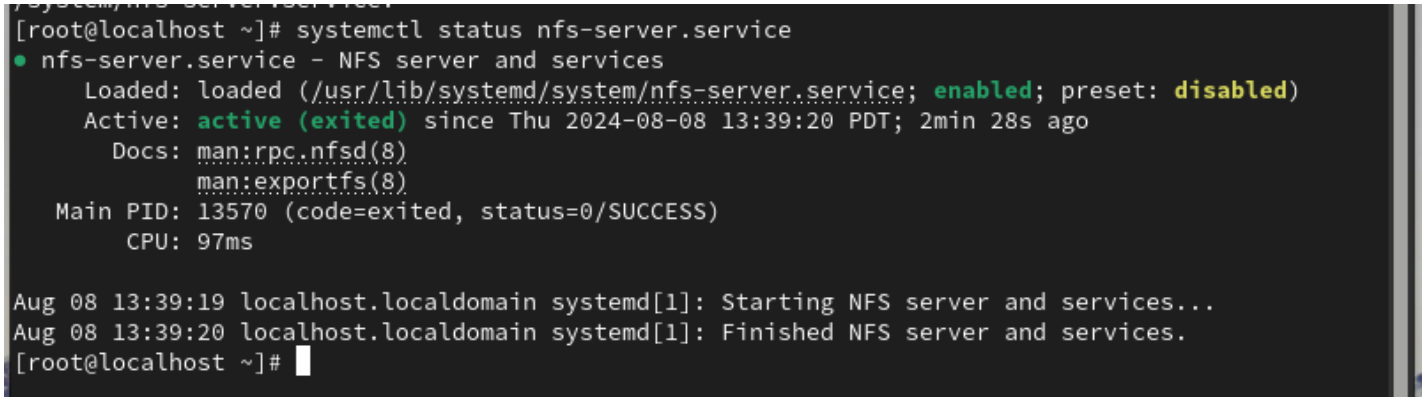
Verify the packages were installed.



2.1c Start the NFS service and ensure that it runs at boot time.

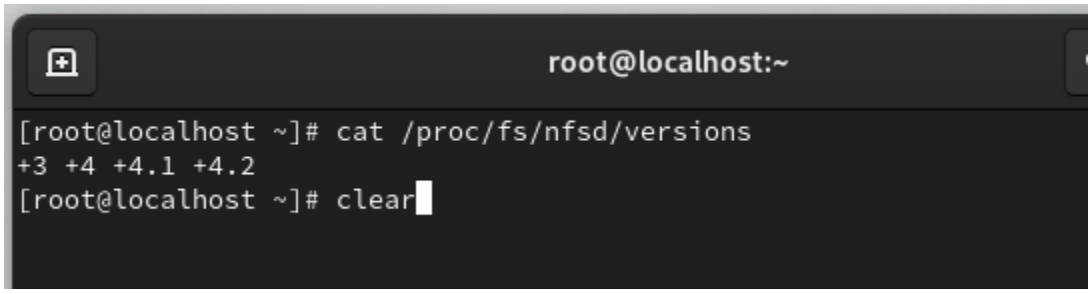


I used systemctl enable –now to not only enable the NFS service to start running when the vm boots but to also make the service start running right away as well during the current boot.



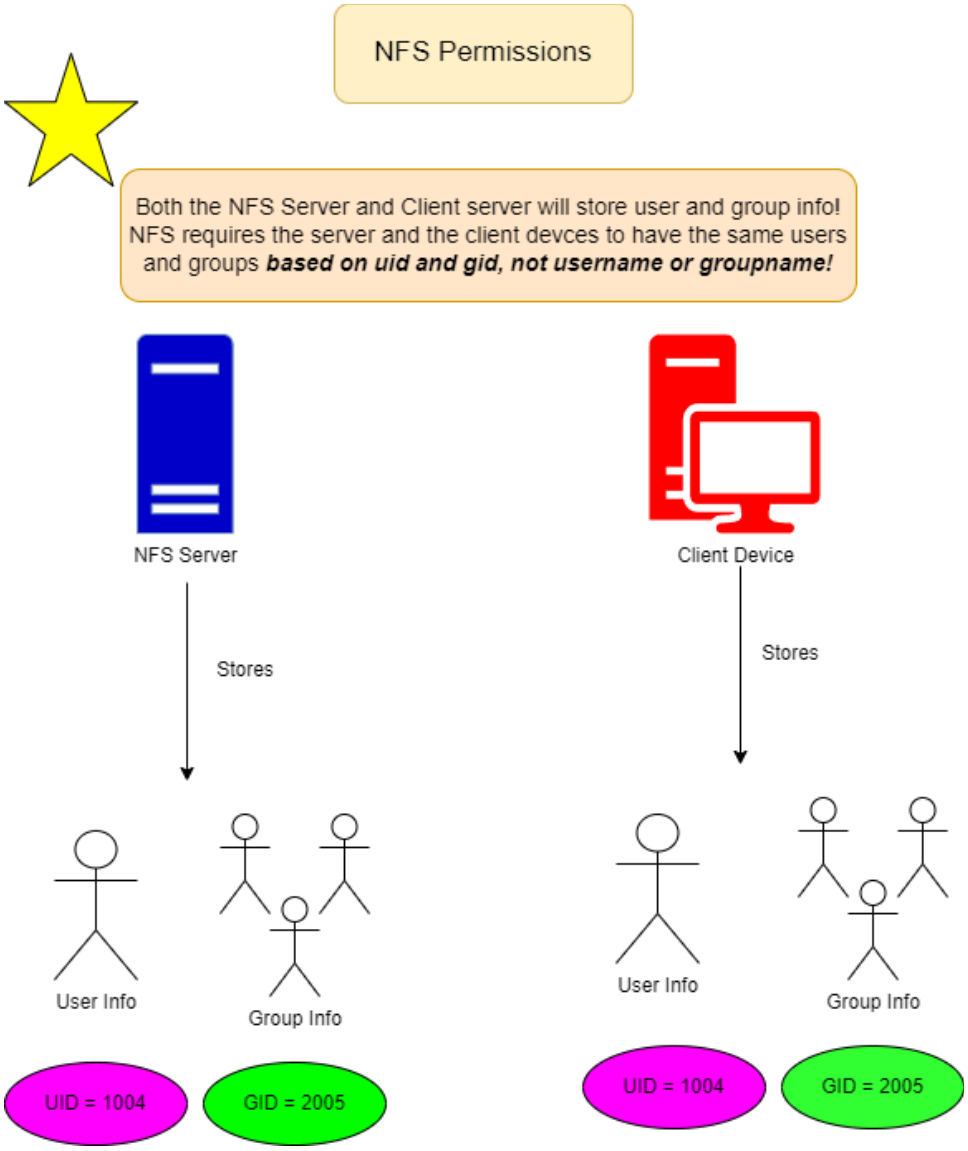
I then used systemctl status to check if NFS is enabled and currently running, and it is.

2.2. Check the versions of NFS supported by the server:



2.3. Create the necessary users and groups for NFS to function.

How do NFS user permissions work?

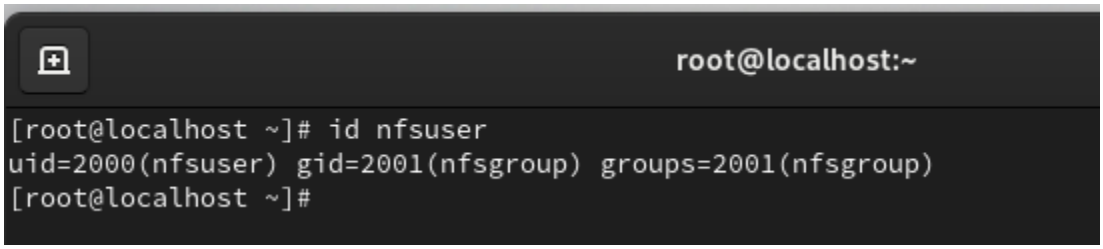


NFS requires the server and the client to both have users and groups with the same uids and gids. This mapping allows the server to know who can access the share on the client device and what permissions they have for the share contents.

```
root@localhost:~
[root@localhost ~]# useradd nfsuser && groupadd nfsgroup
[root@localhost ~]# usermod -u 2000 nfsuser  && groupmod -g 2001 nfsgroup
[root@localhost ~]# usermod -g nfsgroup nfsuser
[root@localhost ~]# passwd nfsuser
Changing password for user nfsuser.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
[root@localhost ~]#
```

Make sure you specify the uid and gid for the user and group! I also added the nfsuser to the nfsgroup and made nfsgroup their primary group.

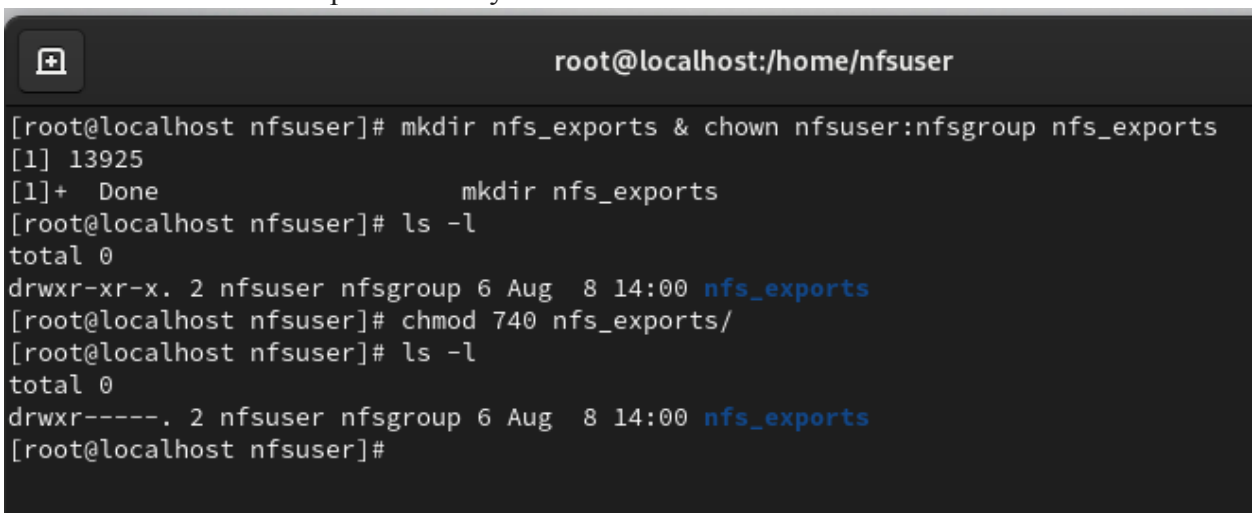
Verify the user and group info:



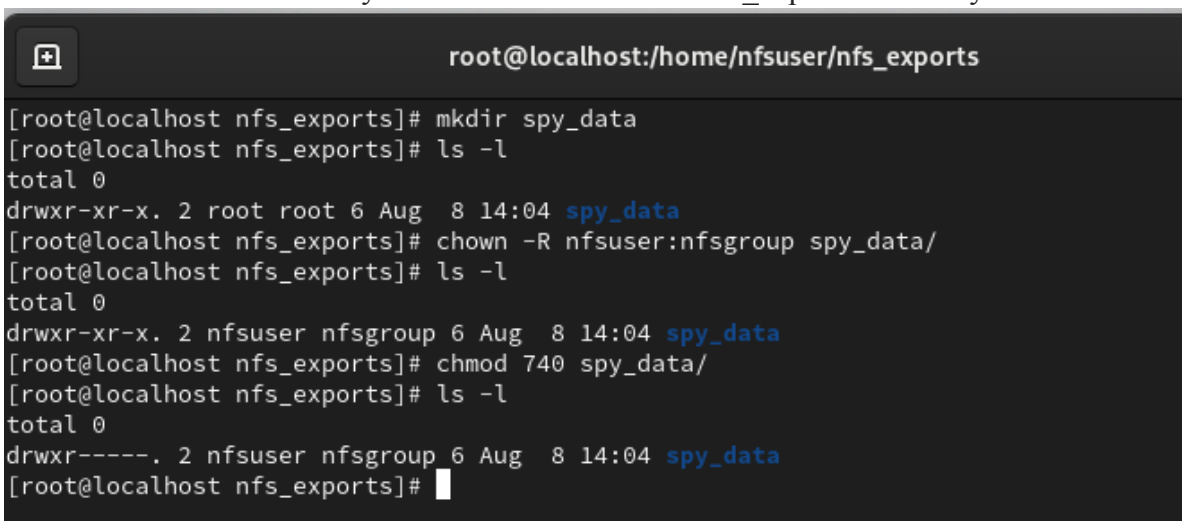
2.4. Create directories and files you want to share through NFS along with necessary ownership and permissions:

This lab will assume that the nfsuser will be the user owner of the nfs export directories and their files. Nfsgroup will be the group owner as well.

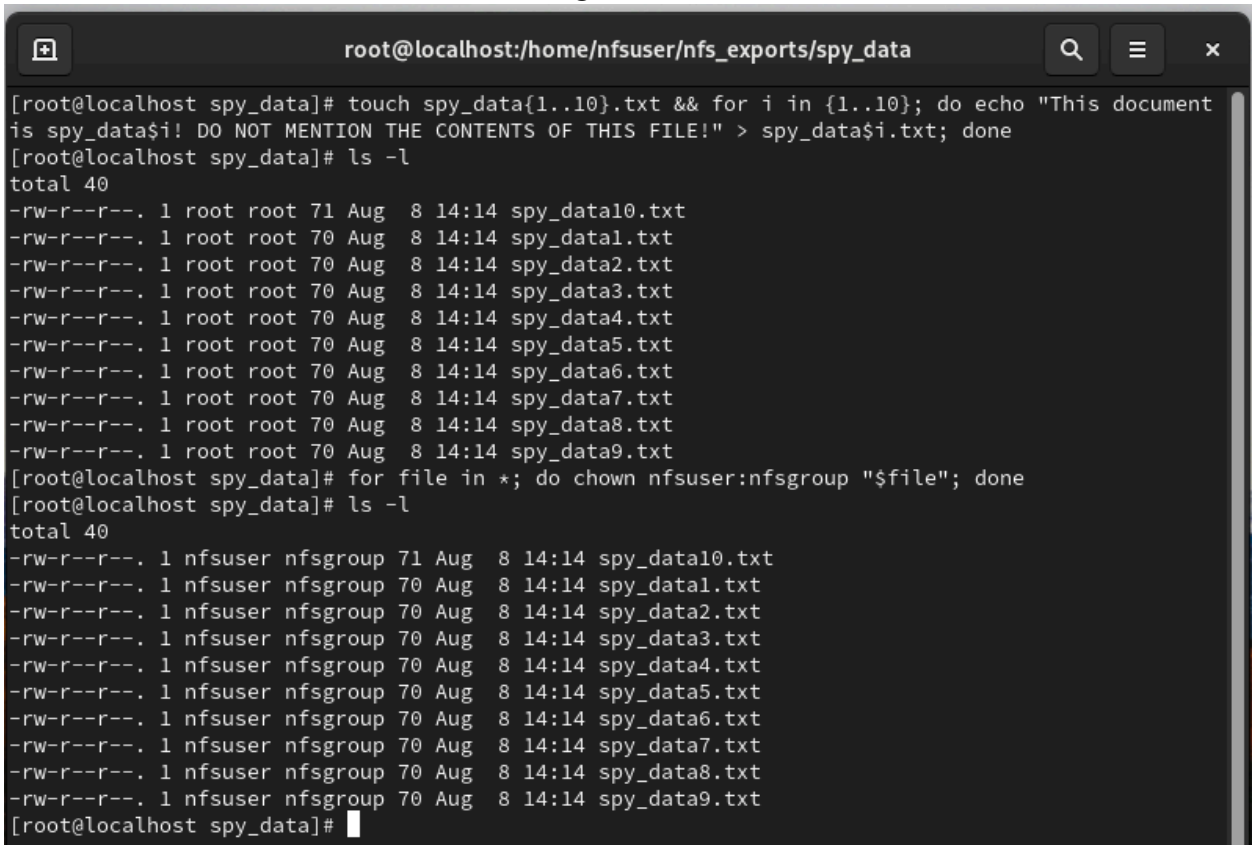
2.4a Create the root export directory.



2.4b Create a subdirectory under the /home/nfsuser/nfs_exports/ directory.



2.4c Create the files that will be shared through nfs.



I used some bash scripting to make the process a lot easier. My first script creates 10 spy_data.txt files numbered from 1-10. It then loops through each file and appends some text to them. The second script changes the permissions for these 10 files using a loop structure as well.

Now I will change the permissions for these files:

```
[root@localhost spy_data]# for file in *; do chmod 740 "$file"; done
[root@localhost spy_data]# ls -l
total 40
-rwxr-----. 1 nfsuser nfsgroup 71 Aug  8 14:14 spy_data10.txt
-rwxr-----. 1 nfsuser nfsgroup 70 Aug  8 14:14 spy_data1.txt
-rwxr-----. 1 nfsuser nfsgroup 70 Aug  8 14:14 spy_data2.txt
-rwxr-----. 1 nfsuser nfsgroup 70 Aug  8 14:14 spy_data3.txt
-rwxr-----. 1 nfsuser nfsgroup 70 Aug  8 14:14 spy_data4.txt
-rwxr-----. 1 nfsuser nfsgroup 70 Aug  8 14:14 spy_data5.txt
-rwxr-----. 1 nfsuser nfsgroup 70 Aug  8 14:14 spy_data6.txt
-rwxr-----. 1 nfsuser nfsgroup 70 Aug  8 14:14 spy_data7.txt
-rwxr-----. 1 nfsuser nfsgroup 70 Aug  8 14:14 spy_data8.txt
-rwxr-----. 1 nfsuser nfsgroup 70 Aug  8 14:14 spy_data9.txt
[root@localhost spy_data]#
```

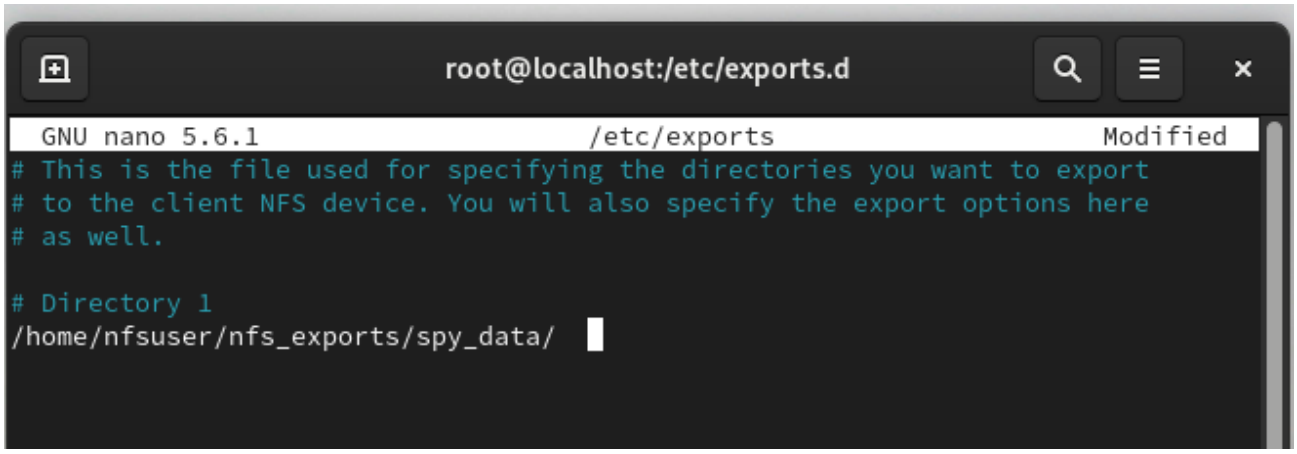
Lastly, my third bash script loops through the files in the current directory and changes their permissions from 644 to 740.

Section #3 Configuring NFS Exports:

3.1. Edit the /etc/exports file to share the created files and directories:

In order for NFS to work on the server you must specify what directory paths you want to share with clients in the /etc/exports configuration file. In this configuration file you will also specify the clients you want each path to be shared with and the export options you want to use.

3.1a. Add the directory path you want to share:



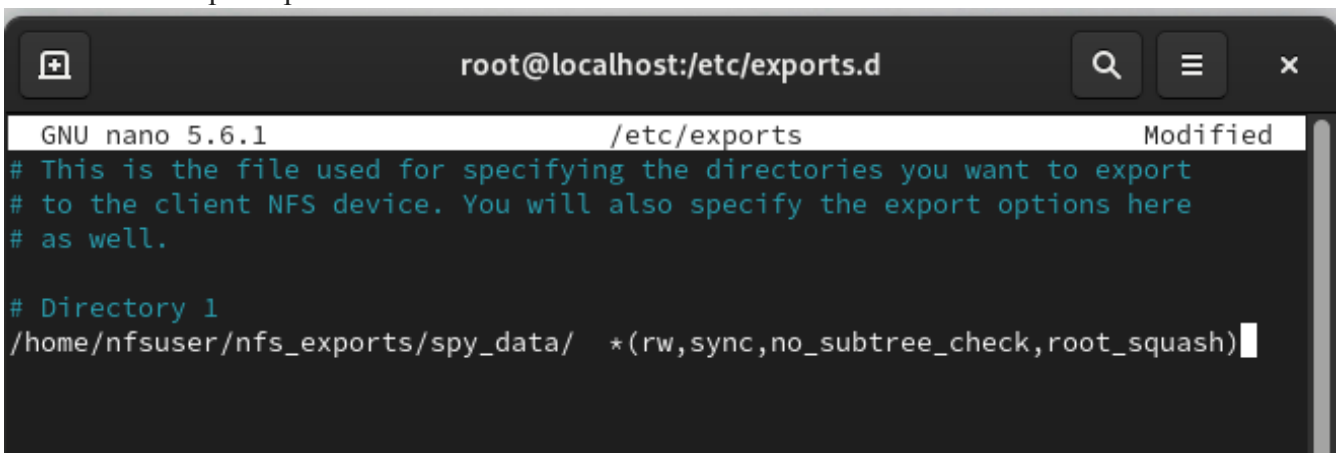
3.1b. Add the necessary export options for the directories:
NFS export options:

Export Option	Purpose	Example
rw	RW only grants clients read and write permissions for the files exported to the client device.	Say test1.txt was exported with rw, but it had rwx. On the client device users will only have rw. It is kind of like a netmask for limiting permissions.
ro	RO only grants clients the ability to read the contents of the files and directories exported to them.	Say test1.txt was exported with only ro. On the client device the users will only be able to read the contents of the file and its directory.
no_subtree_check	No_subtree_check will not check the permissions of the full directory tree above the file a client is trying to access. This setting will force NFS to assume that the user has permissions for the directory and subdirectories.	Say test1.txt is under the /exports/data/ directory. No_subtree_check will simply tell NFS to not check if the user on the client has permissions for the full directory tree, and to only check permissions for the files. ***No_subtree_check will help improve nfs performance, but it comes with security risks.***
subtree_check	Subtree_check will actually perform the full directory tree permissions check to ensure that a user is not accessing something they shouldn't be accessing.	For instance, say test1.txt has the /data/tests/test1.txt path. With subtree check NFS will check if users have access to the /data/tests subdirectory and if they have access to the parent /data directory.

root_squash	Root squash will cause client users with root privileges (root/ users in sudo group) to have their privileges “squashed” or simply ignored when accessing NFS shares. Since privileged users could typically enter any directory they want, having root_squash will prevent abuse of this privilege.	Say on a client device user admin with sudo privileges wants to view files in the /data/secret_files/ NFS share. The NFS share is configured to technically only allow nfsuser to access it. Admin would try to access the share using sudo -i to gain root privileges, but root_squash will prevent this from allowing Admin to access the NFS share.
no_root_squash	No_root_squash will allow users on the client device to maintain their privileged permissions. Sp users that have sudo privileges can use the privileges to access the NFS share even if they shouldn't be allowed to.	For example say on a client device a user named testuser has sudo privileges. If the client device has an NFS mounted to it, the testuser can access the files in the NFS share directory even if they do not map to a uid on the server.
sync	Sync will only confirm write operations once the data has been written to the physical storage device (ssd, hard drive). This is vital for preventing data losses. The only downside is that sync can cause NFS write operations to appear slower since the server will only acknowledge the write operation if it successfully writes data to the disk.	
async	Async will confirm the write operation once the write data is stored on the server memory (ram). Async is faster than sync, but at the cost of guaranteeing that the data is actually written on the physical storage device. If	

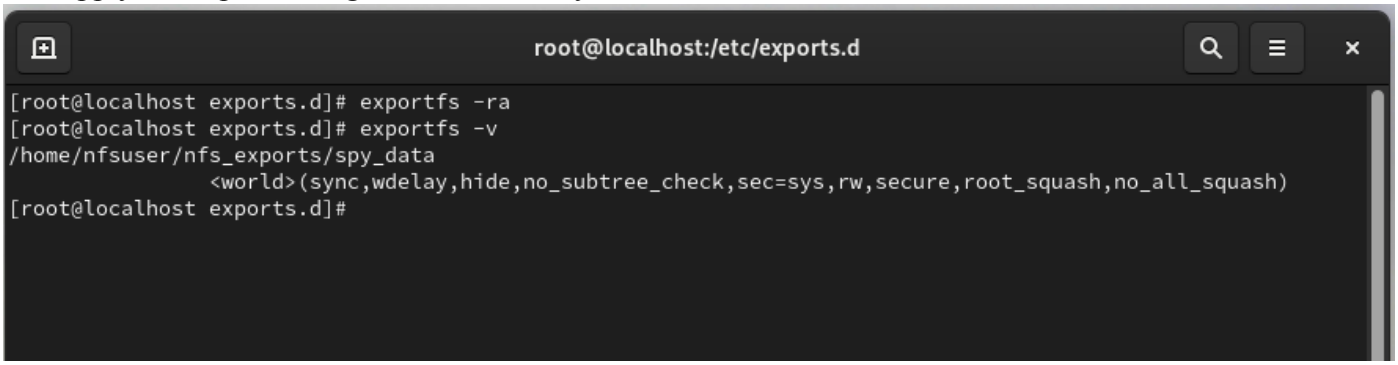
	something happens to the server such as a shutdown, restart, or crash, the data that was meant to be written would be lost.	

These are the export options I have chosen:



The * symbol represents any networked device.

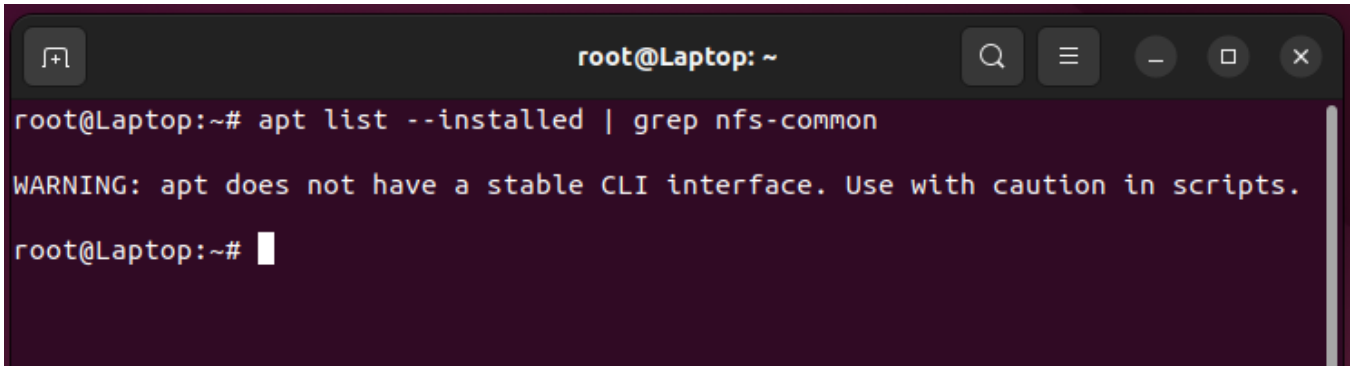
3.2. Apply the export configuration and verify it as well:



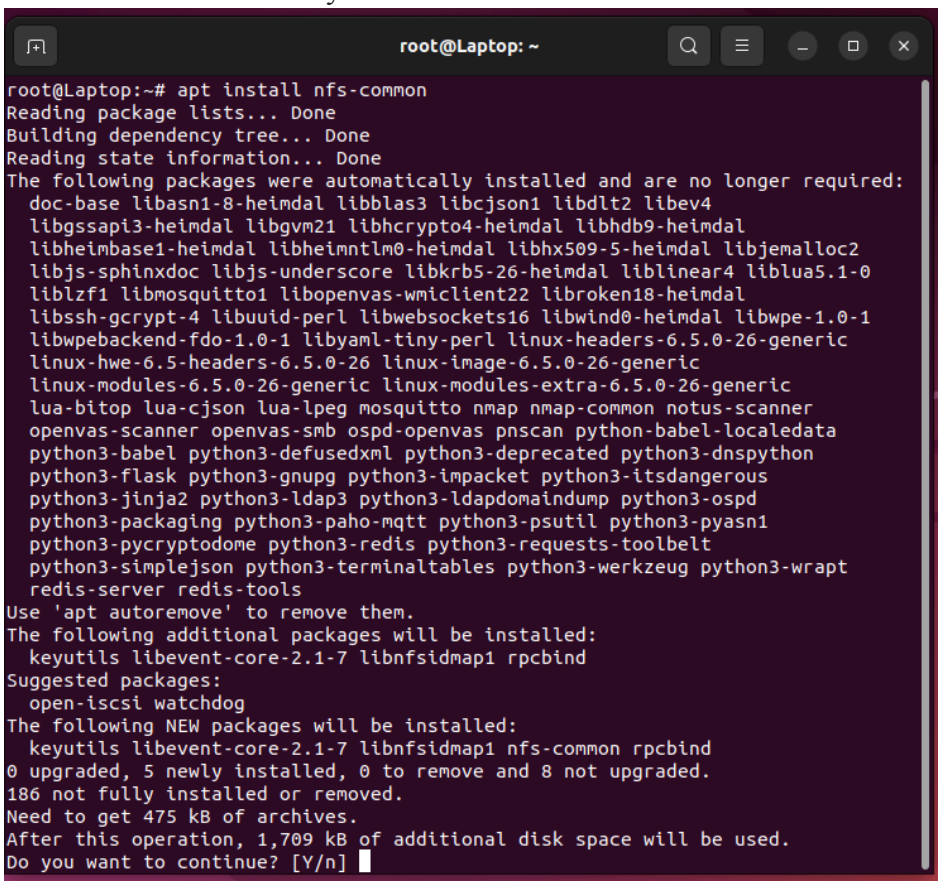
The flags I used for the exportfs command are r,a, and v. The -r flag reloads export settings. The -a flag exports all directories listed on the /etc/exports file. Lastly, -v is used to verify what you are exporting to the client.

Section #4 NFS Client Setup:

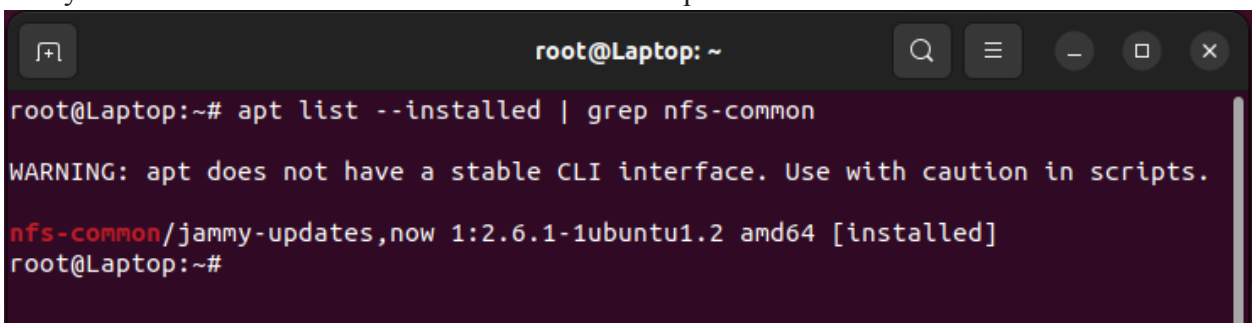
4.1. Check if the NFS packages are installed on the client device. If not, install them:

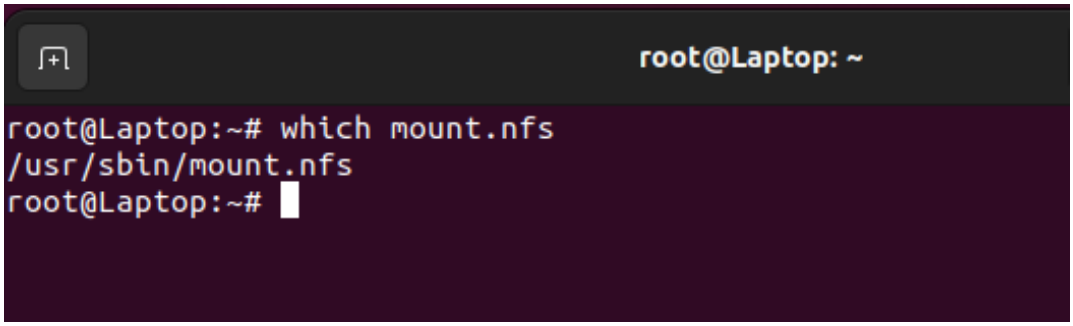


NFS is not installed on my Ubuntu client device. I will have to install it.



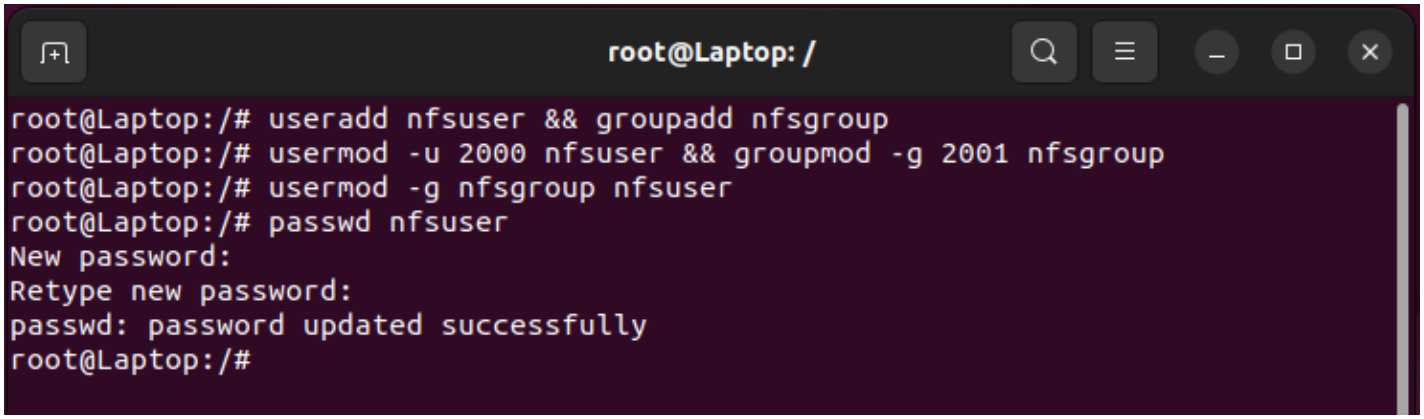
Verify that NFS is installed and client commands are present:





4.2. Create the necessary user and group, ***ensuring that the uid and gid matches the ones on the NFS server.***

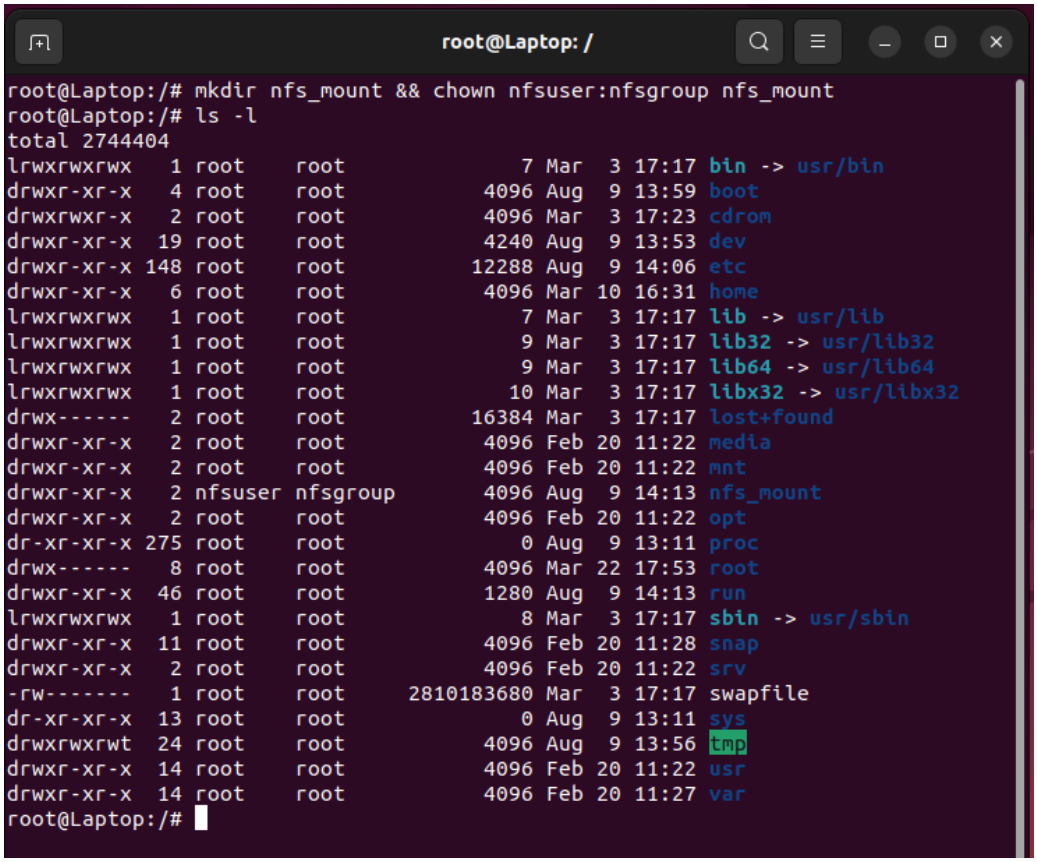
Note that the names for groups and users on the client and servers do not have to match, only the gids and uids!



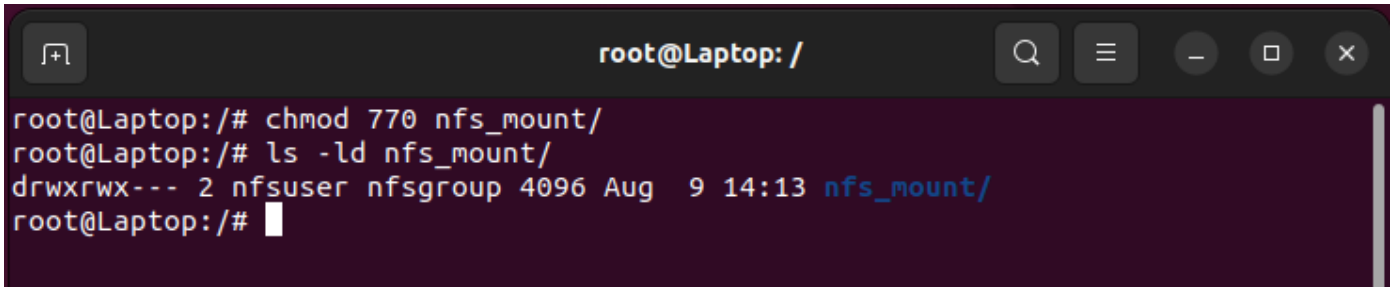
4.3. Mount the NFS share:

4.3a. Create a mount point to mount the share on:

Since NFS is literally named Network File System, like any file system on Linux it must be mounted on a directory so you can access its data.The mount point I will create will be directly under / (root). It is going to be called nfs_mount and it will be owned by nfsuser and the nfsgroup (not root even though it is at the top of the filesystem hierarchy).

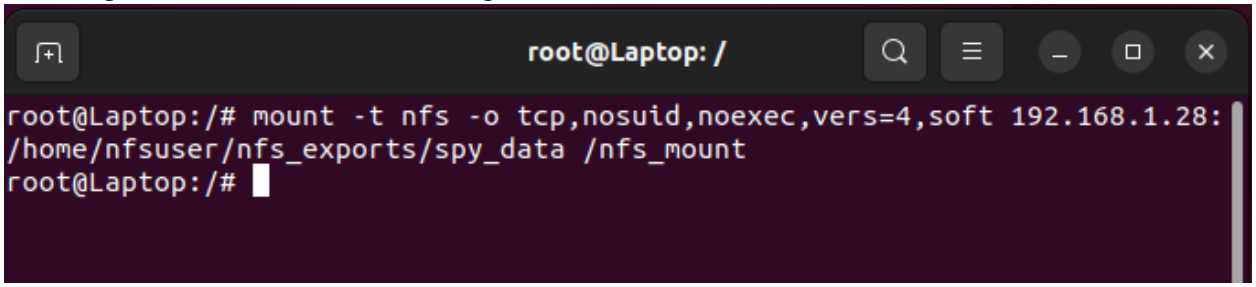


I will now change the permissions for the mount from 755 to 770 since I want nfsuser and nfsgroup to both have rwx permissions for the mount point.



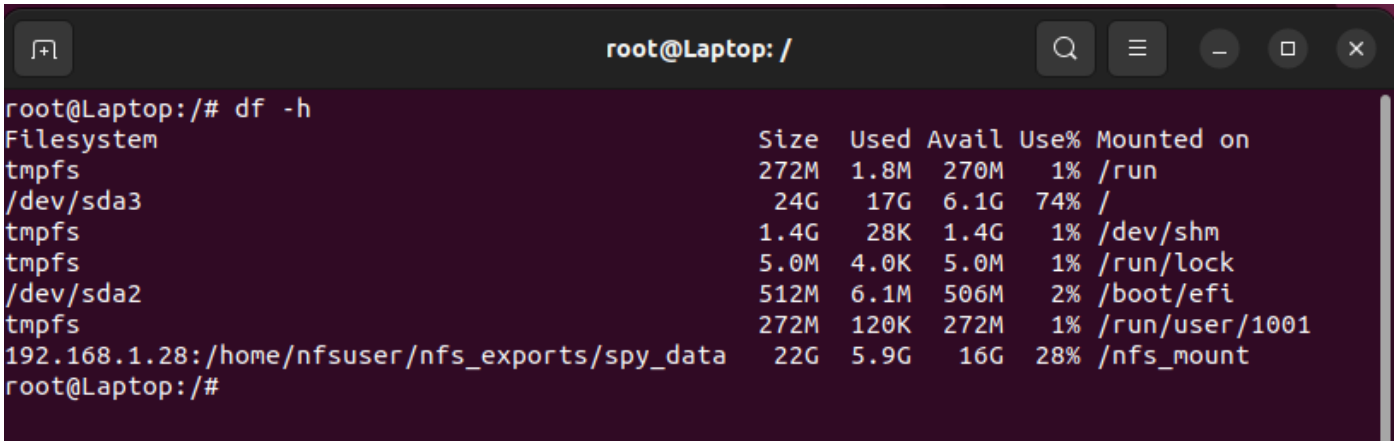
4.3b. Mount the NFS share:

I will now mount the NFS export onto the /nfs_mount/ directory. Here are some of the NFS mount options. These are the mount options I have chosen:



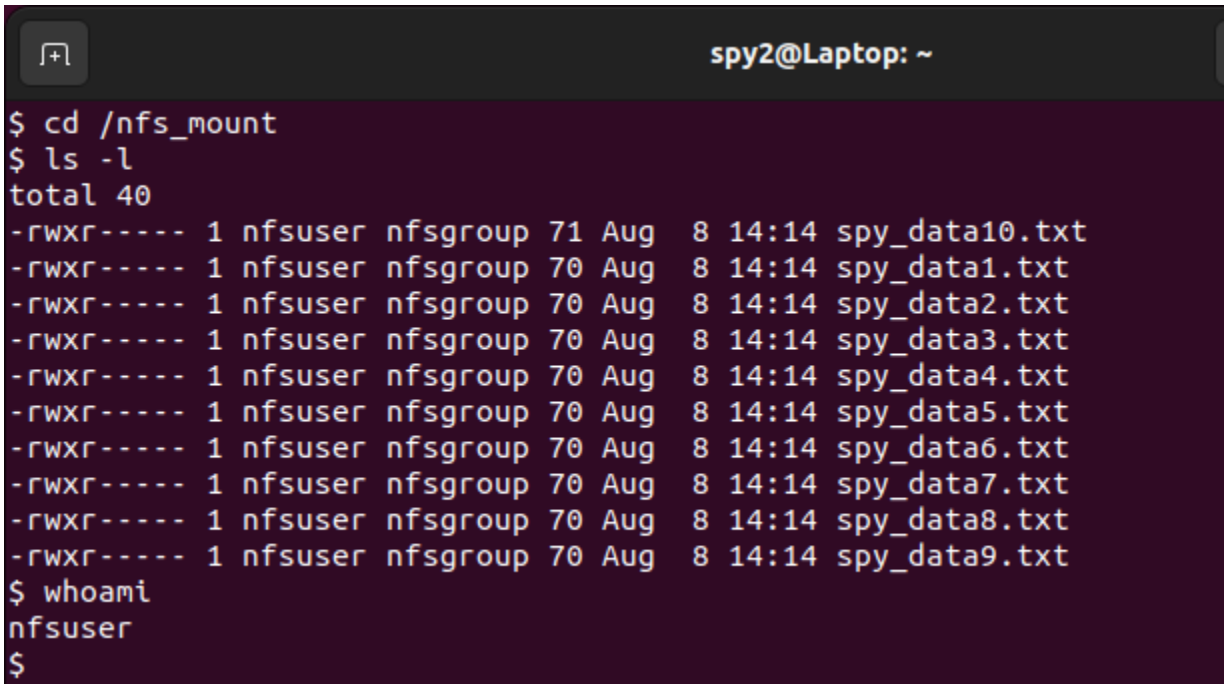
I chose the tcp, nosuid, noexec, vers=4, and soft options. The tcp option will specify that this client should communicate with the NFS server using the tcp protocol. Nosuid means that files in the share with the setuid bit set won't execute with elevated permissions. Noexec prevents binary files from executing on the device. Vers=4 is used to specify the NFS version I want to use. Lastly soft is an option that specifies if the server is unreachable the client will try a few times to reconnect before displaying an error and giving up.

4.3c. Verify the mount:

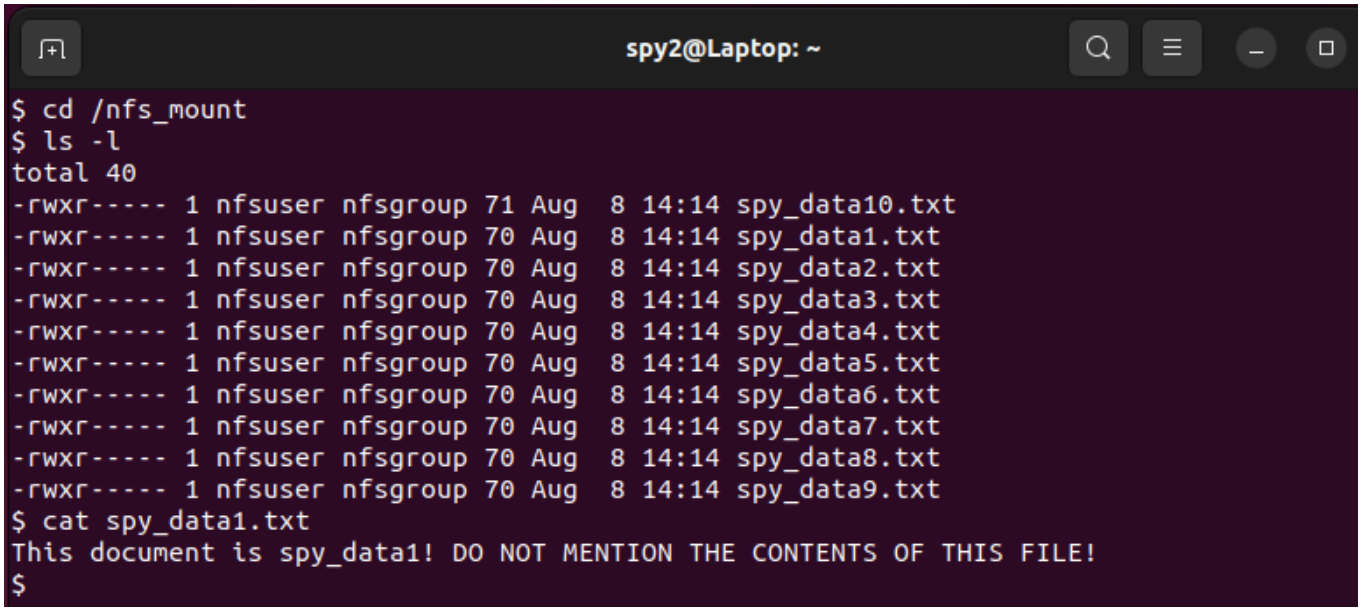


4.4. Test actions against the NFS share. Try to read/ write to it and create files in it:

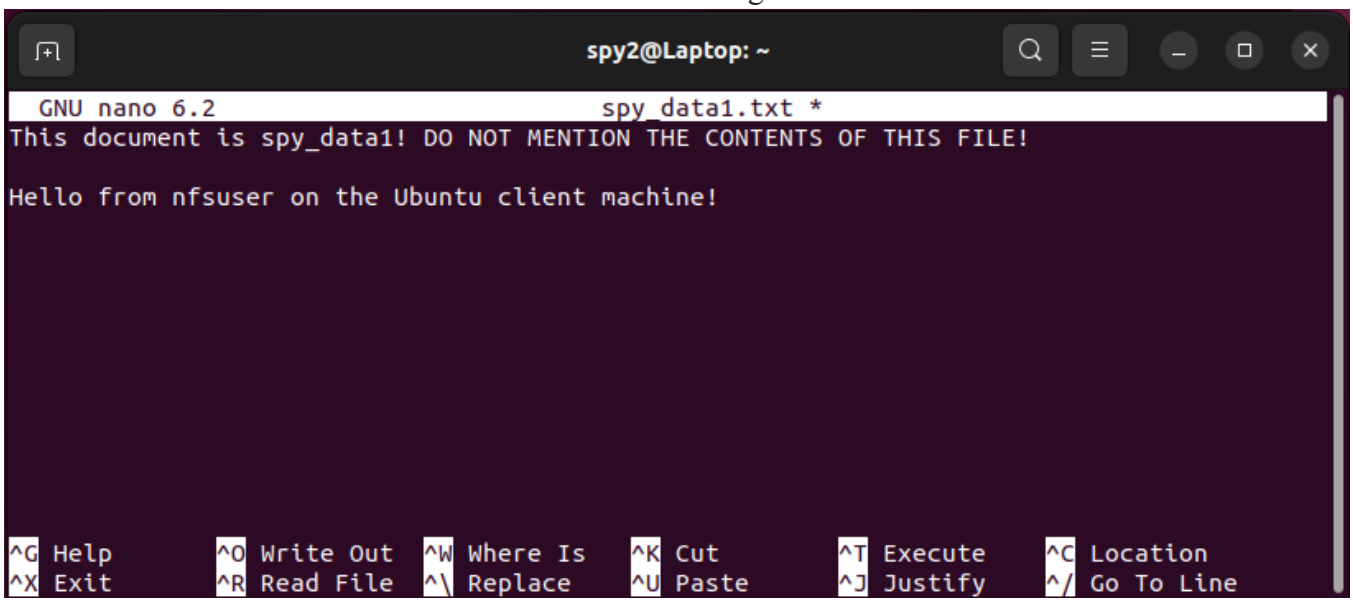
4.4a. Try to access the NFS share as nfsuser on the client:

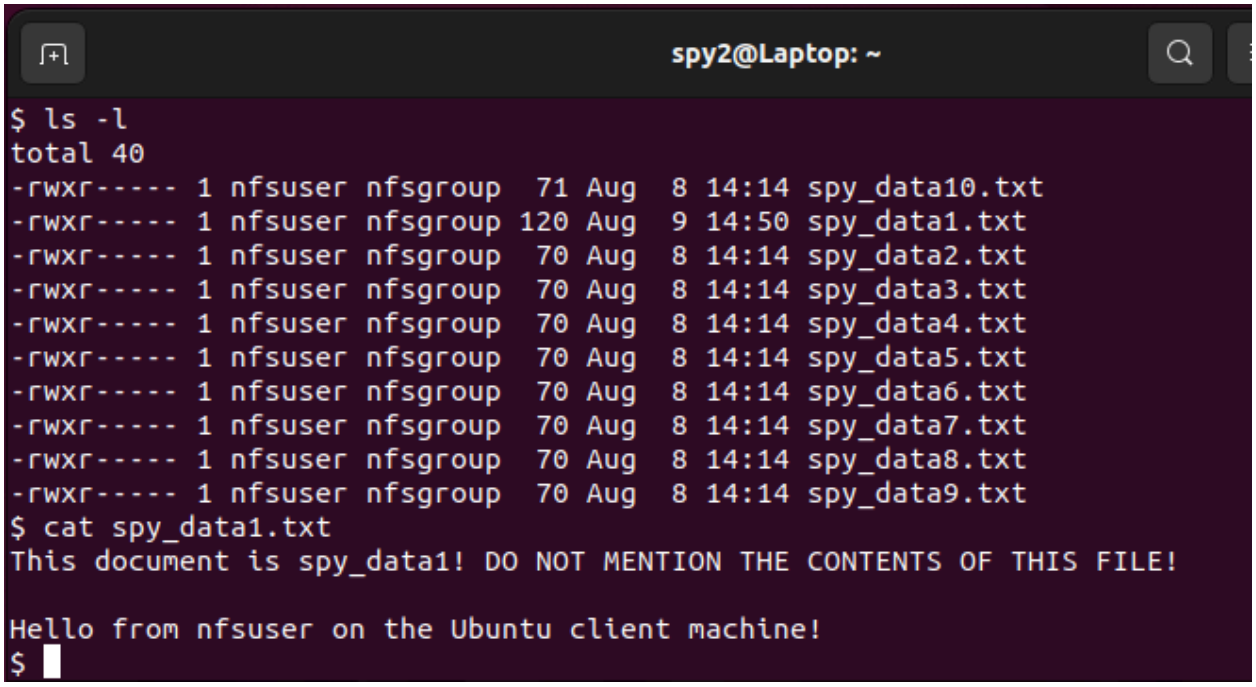


4.4b. Try to read and write to files in the NFS Share:



I can read the contents of the files! Now time to test writing to them!





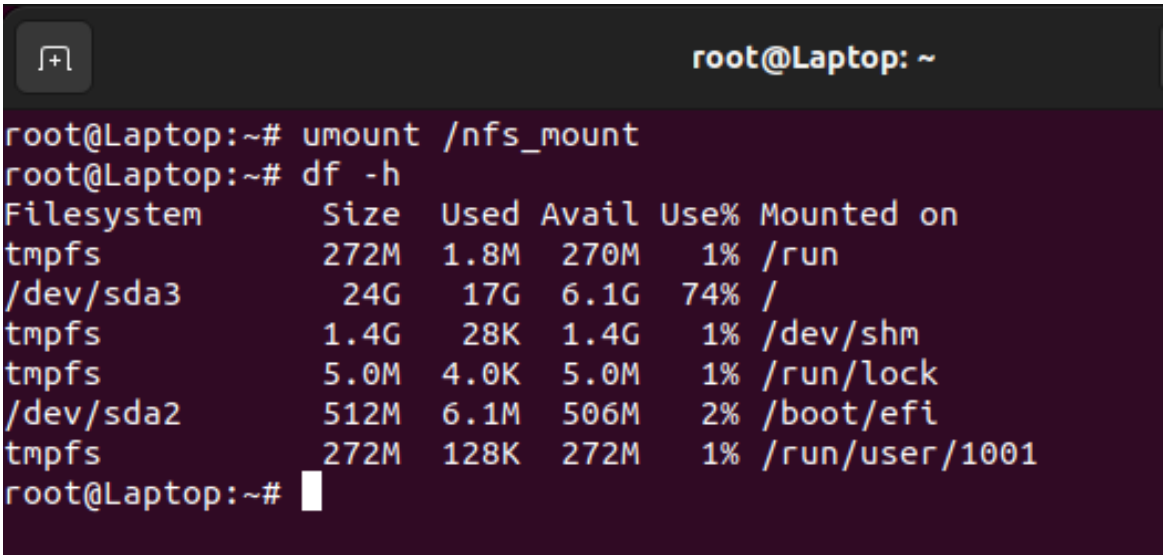
I was able to write to the file on the NFS server!

4.5. Configure the client to automount the NFS share on boot with Autofs:

Note that the /nfs_mount/ directory used as the mount point before in section 4.4 is not going to be used anymore! I deleted it after demonstrating manually mounting NFS shares.

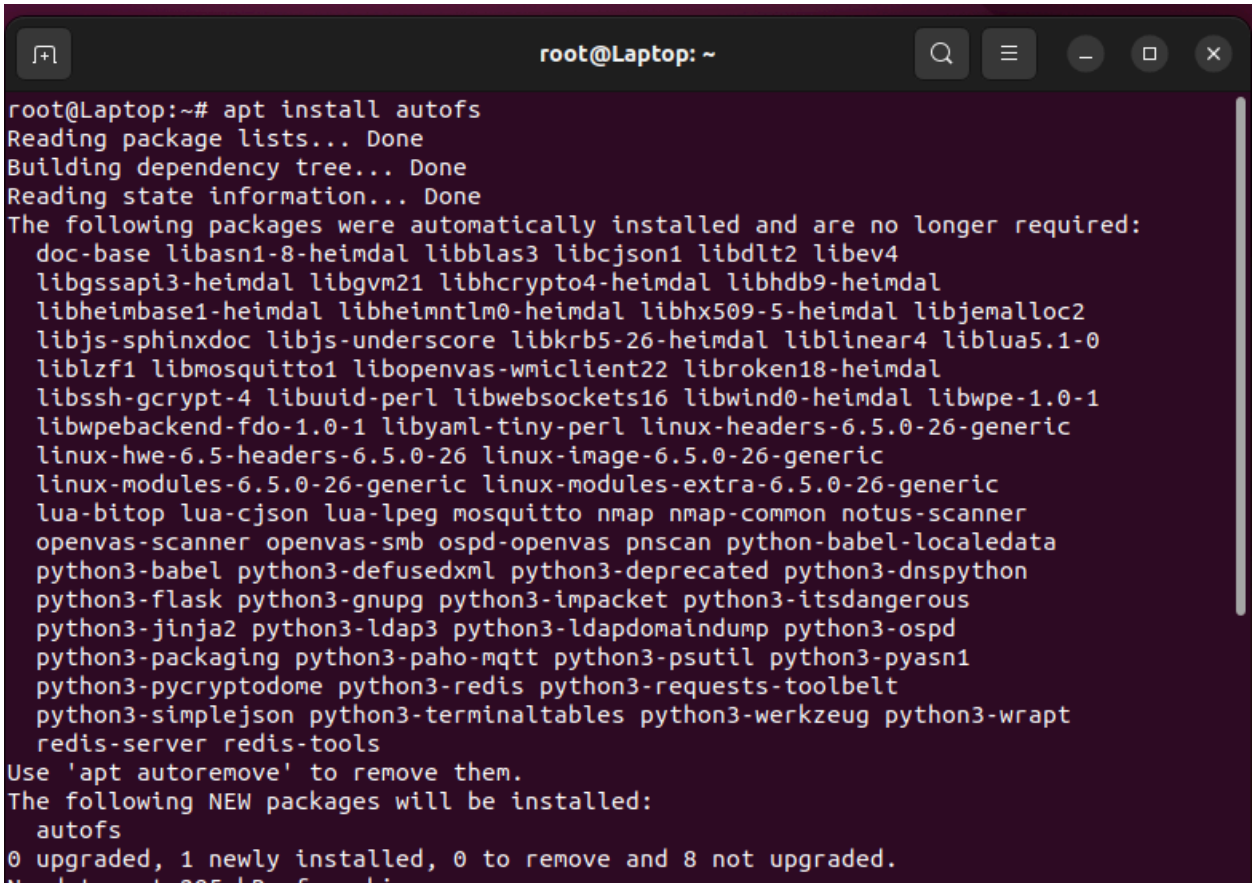
4.5a. Unmount the NFS share:

Before configuring NFS to mount automatically you need to unmount the export.

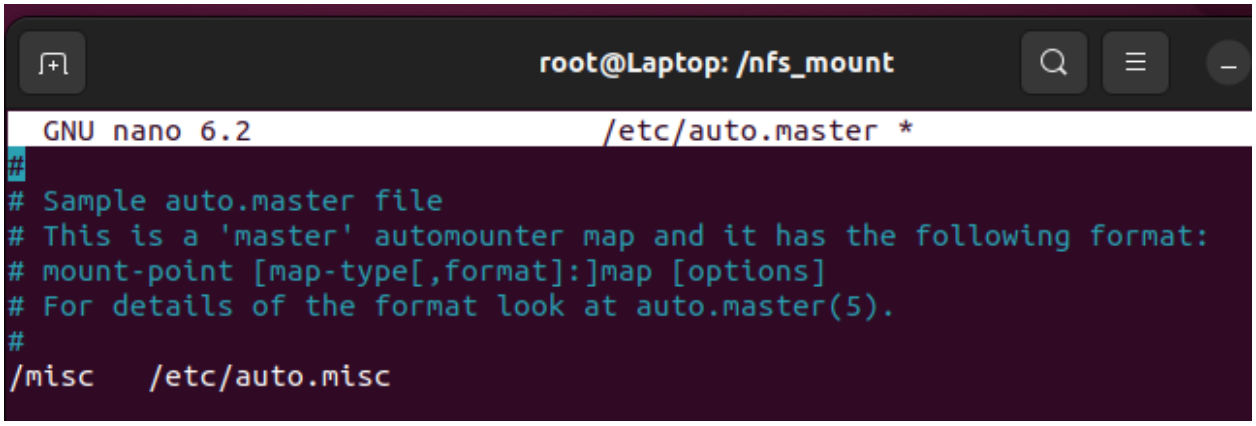


4.5b. Install autofs if it is not installed already:

Autofs is not installed, so I installed it onto the Ubuntu client:



4.5c. Edit the /etc/auto.master file, adding the mount point, and /etc/auto.nfs to define the nfs shares. You can also include a timeout period for limiting how long the nfs export will remain mounted on the client device without interaction or access. I am using a new mount point, /misc.



4.5d. Create the spy_data subdirectory under /misc/ directory. Then edit the /etc/auto.misc file.

Create the /misc/spy_data subdirectory:

```
root@Laptop: /
root@Laptop:/# mkdir -p /misc/spy_data/
root@Laptop:/# ls -l misc/spy_data/
total 0
root@Laptop:/#
```

Edit the /etc/ auto.misc file:

```
root@Laptop: ~
  GNU nano 6.2                                  /etc/auto.misc *
#
# This is an automounter map and it has the following format
# key [ -mount-options-separated-by-comma ] location
# Details may be found in the autofs(5) manpage

cd              -fstype=iso9660,ro,nosuid,nodev :/dev/cdrom

# the following entries are samples to pique your imagination
#linux          -ro,soft                ftp.example.org:/pub/linux
#boot           -fstype=ext2            :/dev/hda1
#floppy         -fstype=auto            :/dev/fd0
#floppy         -fstype=ext2            :/dev/fd0
#e2floppy       -fstype=ext2            :/dev/fd0
#jaz            -fstype=ext2            :/dev/sdc1
#removable      -fstype=ext2            :/dev/hdd
spy_data        -fsytpe=nfs,soft,tcp    192.168.1.28:/home/nfsuser/nfs_exports/spy_data
```

4.5e. Restart the autofs service:

```
root@Laptop: ~
root@Laptop:~# systemctl restart autofs
root@Laptop:~#
```

4.6f. Test the mount to ensure you can access the files.

As you can see, as the nfsuser I am able to list the contents of the spy_data NFS share. You can even see that the NFS export is mounted on my device.

```
spy2@Laptop: /nfs_mount
$ ls -l /misc/spy_data
total 40
-rwxr----- 1 nfsuser nfsgroup  71 Aug  8 14:14 spy_data10.txt
-rwxr----- 1 nfsuser nfsgroup 120 Aug  9 14:50 spy_data1.txt
-rwxr----- 1 nfsuser nfsgroup  70 Aug  8 14:14 spy_data2.txt
-rwxr----- 1 nfsuser nfsgroup  70 Aug  8 14:14 spy_data3.txt
-rwxr----- 1 nfsuser nfsgroup  70 Aug  8 14:14 spy_data4.txt
-rwxr----- 1 nfsuser nfsgroup  70 Aug  8 14:14 spy_data5.txt
-rwxr----- 1 nfsuser nfsgroup  70 Aug  8 14:14 spy_data6.txt
-rwxr----- 1 nfsuser nfsgroup  70 Aug  8 14:14 spy_data7.txt
-rwxr----- 1 nfsuser nfsgroup  70 Aug  8 14:14 spy_data8.txt
-rwxr----- 1 nfsuser nfsgroup  70 Aug  8 14:14 spy_data9.txt
$ whoami
nfsuser
$ df -h
Filesystem                                      Size  Used Avail Use% Mounted on
tmpfs                                           272M  1.7M  270M   1% /run
/dev/sda3                                        24G   16G  7.6G  67% /
tmpfs                                           1.4G   28K  1.4G   1% /dev/shm
tmpfs                                           5.0M  4.0K  5.0M   1% /run/lock
/dev/sda2                                       512M  6.1M  506M   2% /boot/efi
tmpfs                                           272M  132K  272M   1% /run/user/2000
192.168.1.28:/home/nfsuser/nfs_exports/spy_data  22G  5.9G   16G  28% /misc/spy_data
$
```

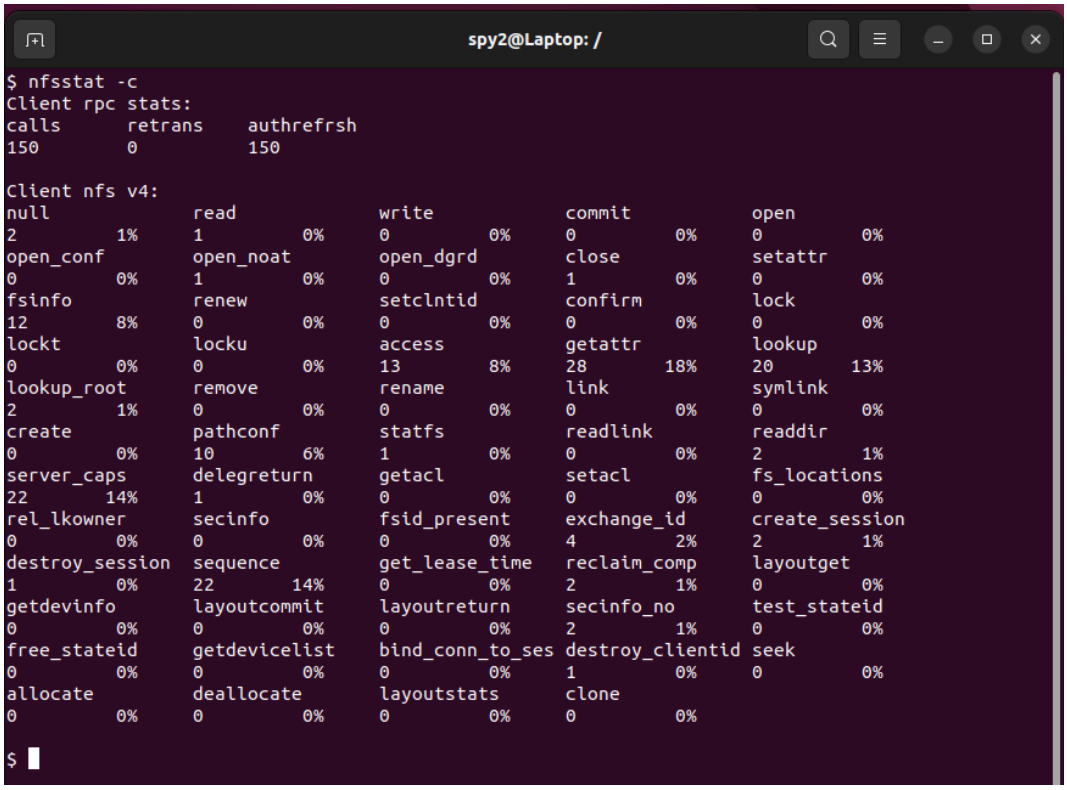
I can even view the contents of the files!

```
spy2@Laptop: /
$ cd /misc/spy_data/
$ ls -l
total 40
-rwxr----- 1 nfsuser nfsgroup  71 Aug  8 14:14 spy_data10.txt
-rwxr----- 1 nfsuser nfsgroup 120 Aug  9 14:50 spy_data1.txt
-rwxr----- 1 nfsuser nfsgroup  70 Aug  8 14:14 spy_data2.txt
-rwxr----- 1 nfsuser nfsgroup  70 Aug  8 14:14 spy_data3.txt
-rwxr----- 1 nfsuser nfsgroup  70 Aug  8 14:14 spy_data4.txt
-rwxr----- 1 nfsuser nfsgroup  70 Aug  8 14:14 spy_data5.txt
-rwxr----- 1 nfsuser nfsgroup  70 Aug  8 14:14 spy_data6.txt
-rwxr----- 1 nfsuser nfsgroup  70 Aug  8 14:14 spy_data7.txt
-rwxr----- 1 nfsuser nfsgroup  70 Aug  8 14:14 spy_data8.txt
-rwxr----- 1 nfsuser nfsgroup  70 Aug  8 14:14 spy_data9.txt
$ cat spy_data1.txt
This document is spy_data1! DO NOT MENTION THE CONTENTS OF THIS FILE!

Hello from nfsuser on the Ubuntu client machine!
$
```

Section #5 NFS Performance and Security:

6.1. Monitor NFS performance using various tools:

6.1a Use the nfsstat to view client statistics:



6.1b. Use nfsstat to view server statistics:

```
Server nfs v4 operations:
op0-unused       op1-unused       op2-future       access           close
0         0%     0         0%     0         0%     15        3%     1         0%
commit           create           delegpurge       delegreturn      getattr
0         0%     0         0%     0         0%     1         0%     108      23%
getfh            link             lock             lockt            locku
18        3%     0         0%     0         0%     0         0%     0         0%
lookup           lookup_root      nverify          open             openattr
20        4%     0         0%     0         0%     1         0%     0         0%
open_conf        open_dgrd        putfh            putpubfh         putrootfh
0         0%     0         0%     114      25%     0         0%     4         0%
read             readdir          readlink         remove           rename
1         0%     2         0%     0         0%     0         0%     0         0%
renew            restorefh        savefh           secinfo          setattr
0         0%     0         0%     0         0%     0         0%     0         0%
setcltid         setcltidconf     verify           write            rellockowner
0         0%     0         0%     0         0%     0         0%     0         0%
bc_ctl           bind_conn        exchange_id      create_ses       destroy_ses
0         0%     0         0%     4         0%     2         0%     1         0%
free_stateid     getdirdeleg      getdevinfo       getdevlist       layoutcommit
0         0%     0         0%     0         0%     0         0%     0         0%
layoutget        layoutreturn     secinfononam     sequence         set_ssv
0         0%     0         0%     2         0%     156      34%     0         0%
test_stateid     want_deleg       destroy_clid     reclaim_comp     allocate
0         0%     0         0%     1         0%     2         0%     0         0%
copy             copy_notify      deallocate       ioadvise         layouterror
0         0%     0         0%     0         0%     0         0%     0         0%
layoutstats      offloadcancel    offloadstatus    readplus         seek
0         0%     0         0%     0         0%     0         0%     0         0%
write_same
0         0%

[administrator@localhost ~]$
```

6.1d Use showmount to see the clients mounting to the server and to see the exports available on the server:

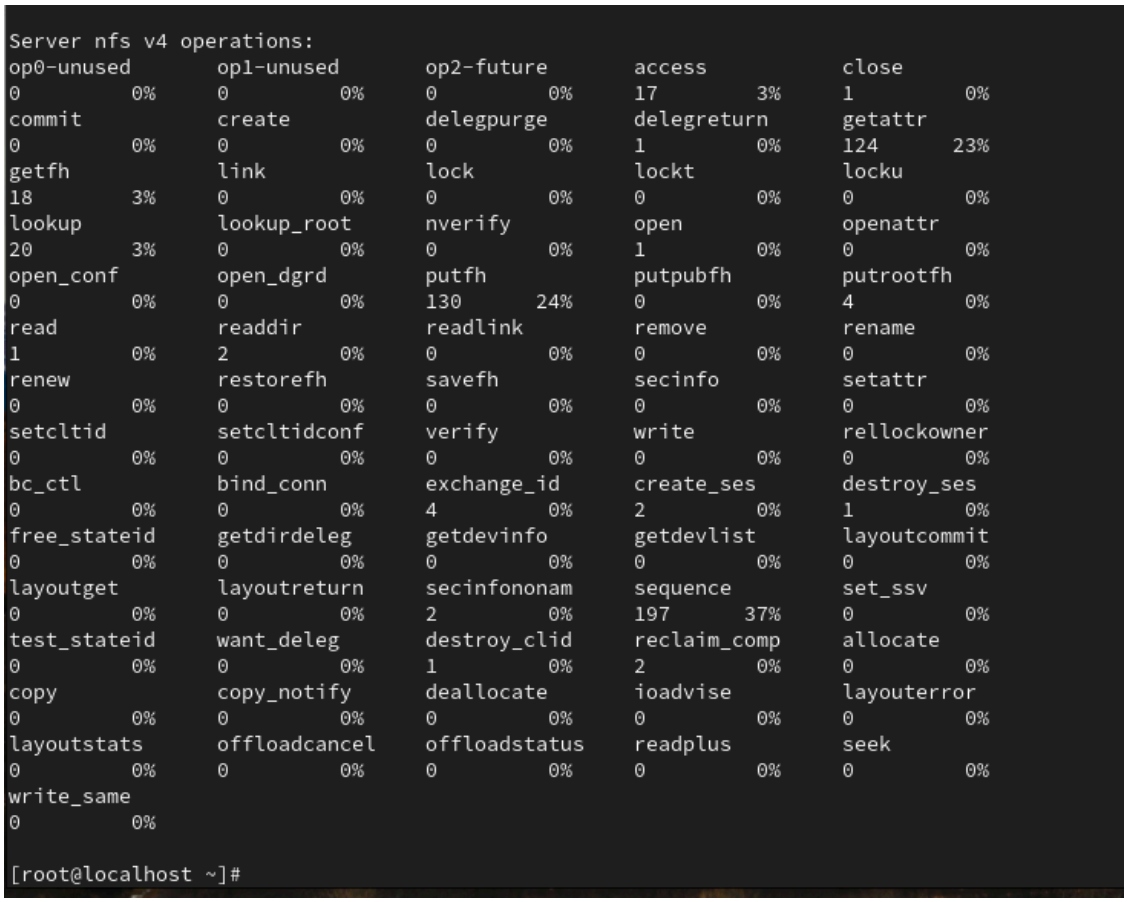
```
root@localhost:~
[root@localhost ~]# showmount --exports
Export list for localhost.localdomain:
/home/nfsuser/nfs_exports/spy_data 192.168.1.36
[root@localhost ~]#
```

\

6.2. Verify the latest version of NFS is being used:

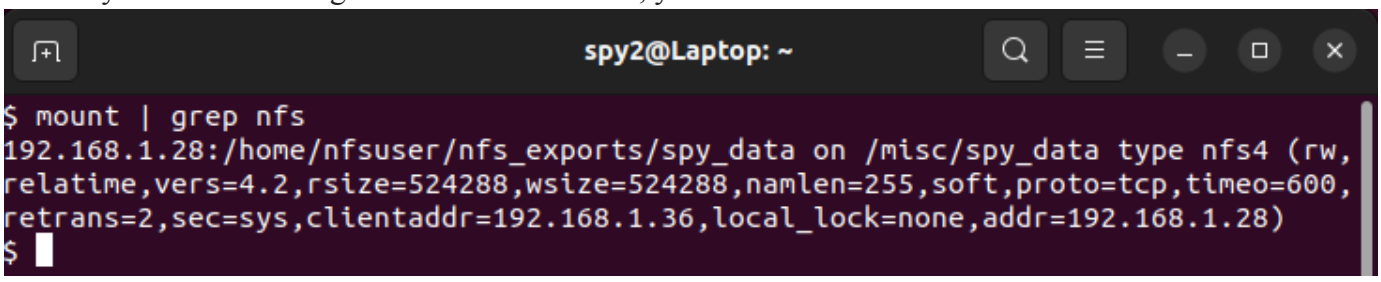
You should use the latest version of NFS so that you are not susceptible to vulnerabilities and have access to the latest features.

To verify that you are currently using the latest version of NFS on the client you can use the nfsstat -s command:



```
Server nfs v4 operations:
op0-unused       op1-unused       op2-future       access           close
0          0%    0          0%    0          0%    17         3%    1          0%
commit           create           delegpurge       delegreturn      getattr
0          0%    0          0%    0          0%    1          0%    124       23%
getfh            link             lock             lockt            locku
18         3%    0          0%    0          0%    0          0%    0          0%
lookup           lookup_root      nverify          open             openattr
20         3%    0          0%    0          0%    1          0%    0          0%
open_conf        open_dgrd        putfh            putpubfh         putrootfh
0          0%    0          0%    130       24%    0          0%    4          0%
read             readdir          readlink         remove           rename
1          0%    2          0%    0          0%    0          0%    0          0%
renew            restorefh        savefh           secinfo          setattr
0          0%    0          0%    0          0%    0          0%    0          0%
setcltid         setcltidconf     verify           write            rellockowner
0          0%    0          0%    0          0%    0          0%    0          0%
bc_ctl           bind_conn        exchange_id      create_ses       destroy_ses
0          0%    0          0%    4          0%    2          0%    1          0%
free_stateid     getdirdeleg      getdevinfo       getdevlist       layoutcommit
0          0%    0          0%    0          0%    0          0%    0          0%
layoutget        layoutreturn     secinfononam     sequence         set_ssv
0          0%    0          0%    2          0%    197       37%    0          0%
test_stateid     want_deleg       destroy_clid     reclaim_comp     allocate
0          0%    0          0%    1          0%    2          0%    0          0%
copy             copy_notify      deallocate       ioadvise         layouterror
0          0%    0          0%    0          0%    0          0%    0          0%
layoutstats      offloadcancel    offloadstatus    readplus         seek
0          0%    0          0%    0          0%    0          0%    0          0%
write_same
0          0%

[root@localhost ~]#
```

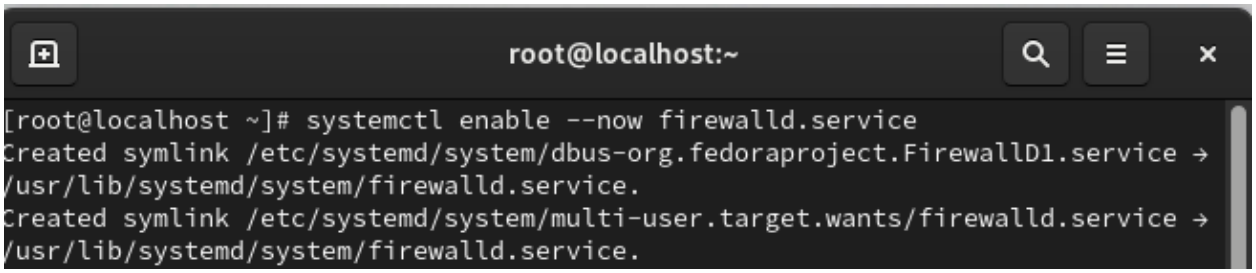
To verify the version being used on the client side, you can use the mount command as follows:



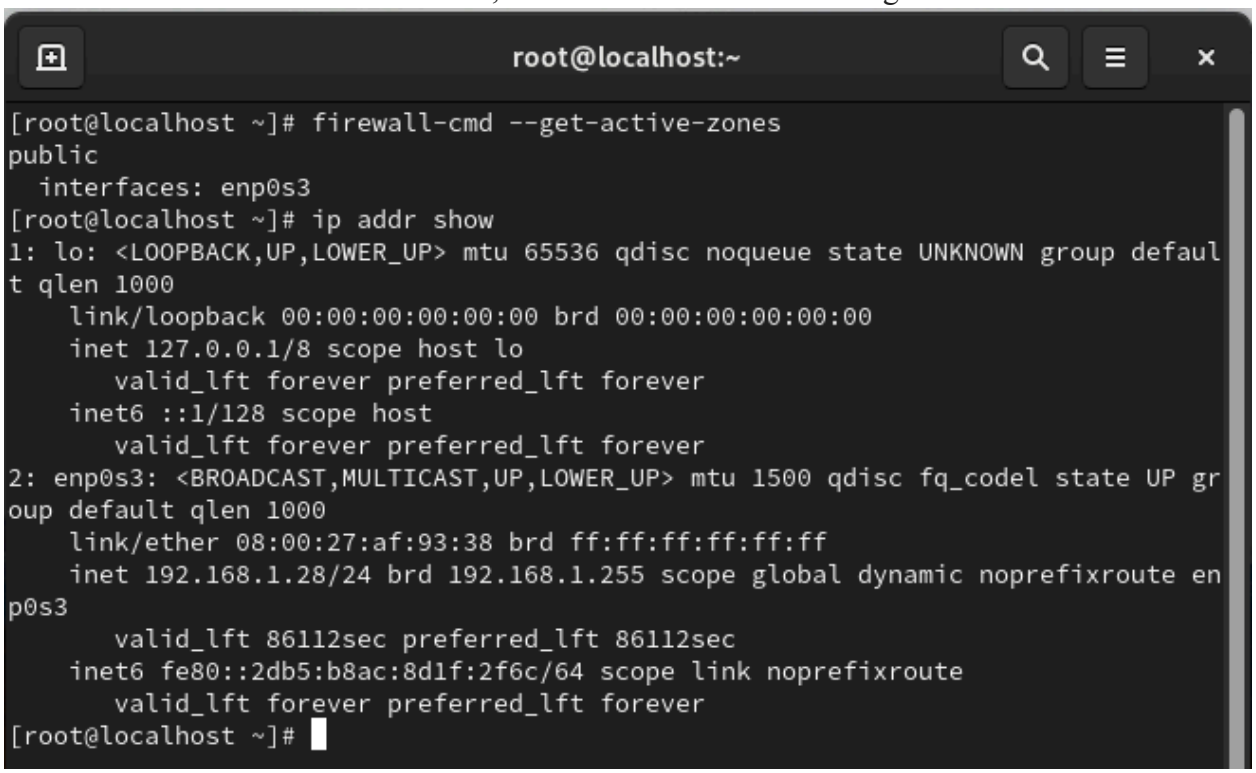
```
spy2@Laptop: ~

$ mount | grep nfs
192.168.1.28:/home/nfsuser/nfs_exports/spy_data on /misc/spy_data type nfs4 (rw,
relatime,vers=4.2,rsize=524288,wsize=524288,namlen=255,soft,proto=tcp,timeo=600,
retrans=2,sec=sys,clientaddr=192.168.1.36,local_lock=none,addr=192.168.1.28)
$
```

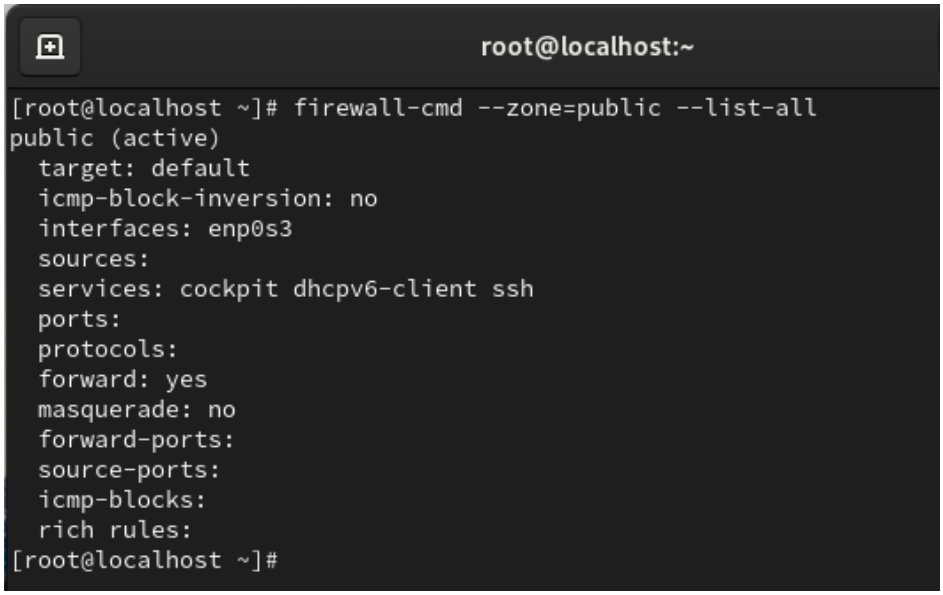
6.3 Set up a firewall for the NFS Server. The firewall will restrict access to the NFS server from only the client device, it will enforce rate limiting to prevent DDoS attacks, and lastly it will log connection attempts to the server.

6.3a. Enable firewalld on the NFS server:



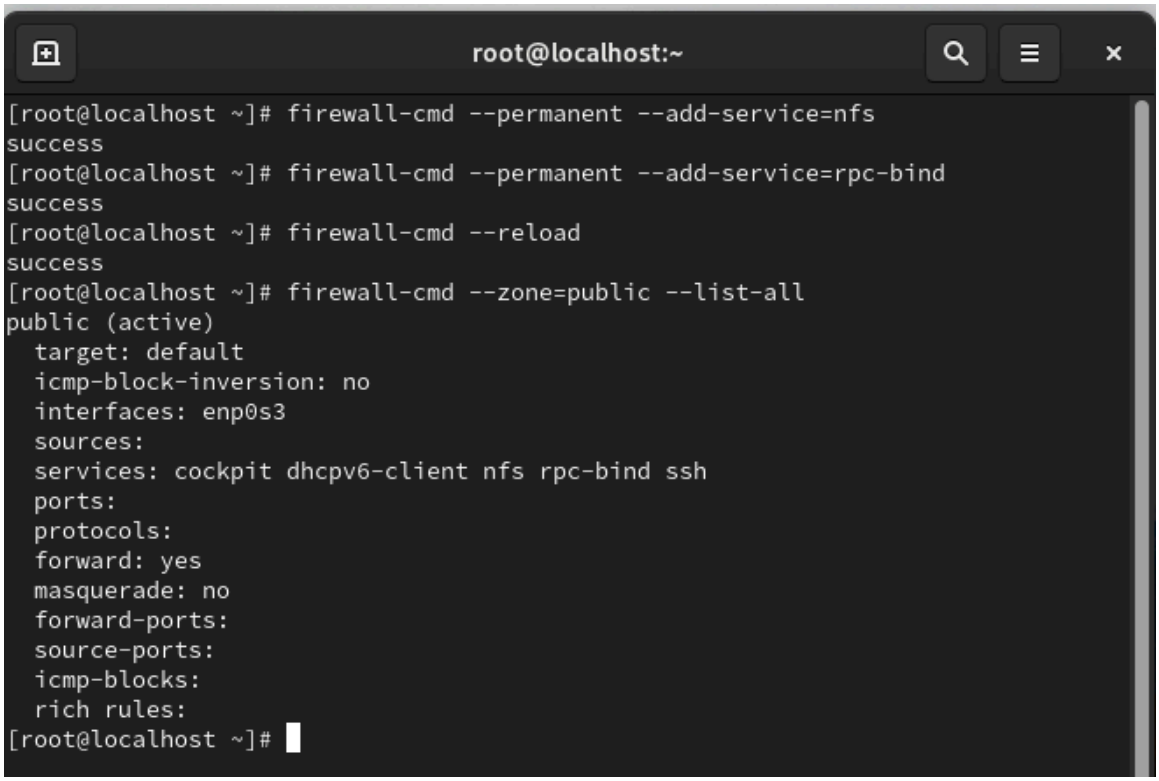
6.3b. Check what the current zone is, and what the zone is enforcing:





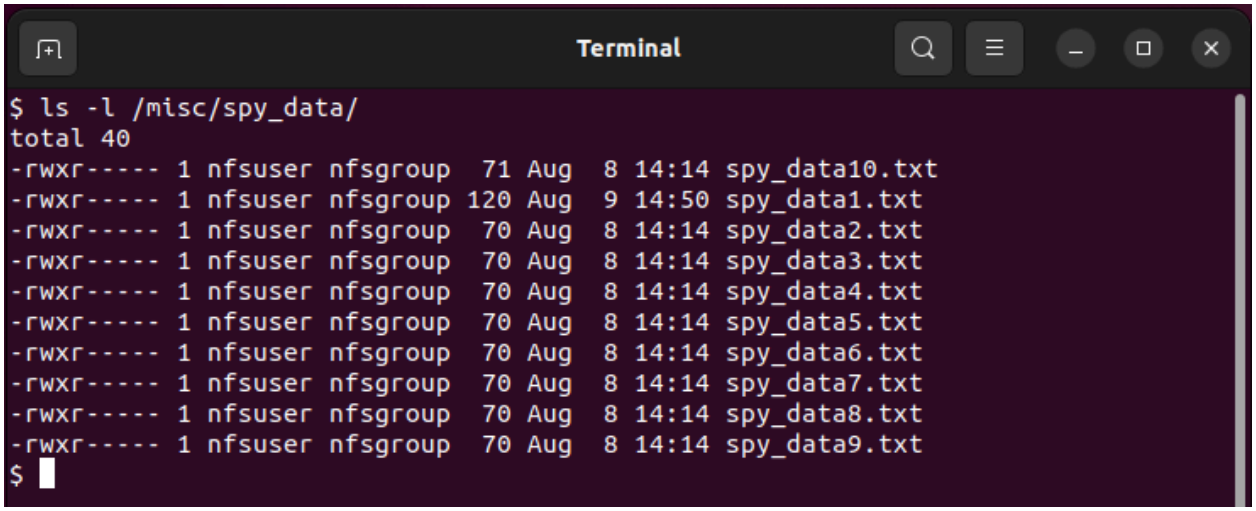
The RHEL server currently has the public zone set as the default zone. The only services this zone is configured to allow so far are cockpit, dhcpv6-client, and ssh.

6.3c. Open the NFS and RPC Bind service ports in the public zone:

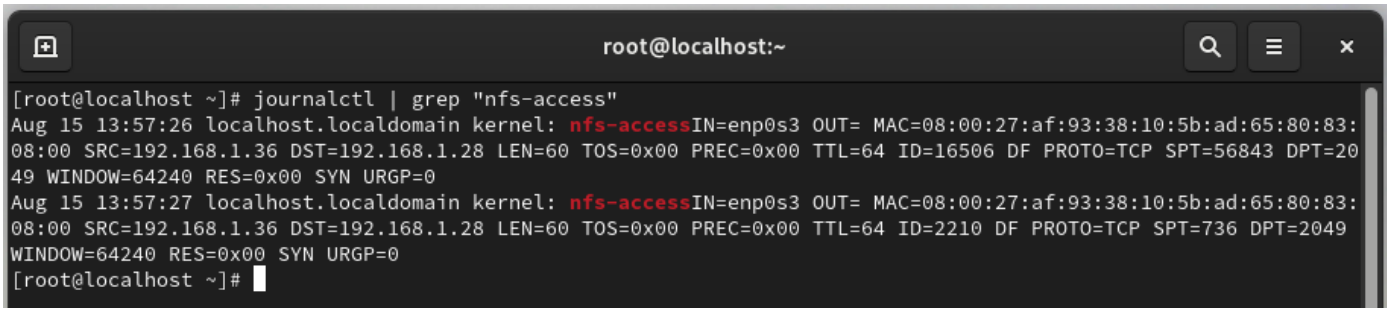


6.3d. Create a rich rule that will implement rate-limiting, only allow NFS connections to the server to occur from a designated client, and logging:

6.3e Test the firewall on the client to ensure that it works:



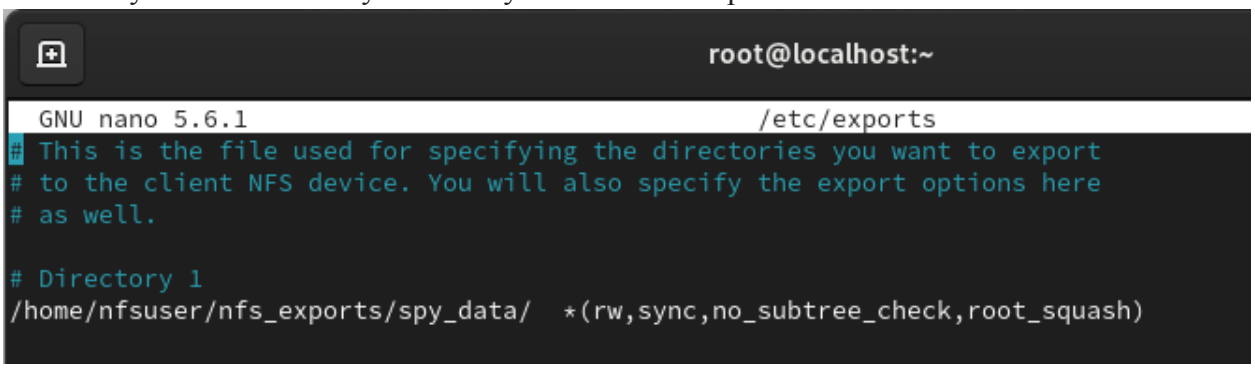
On the client I was able to access the NFS shares after the firewall was implemented!



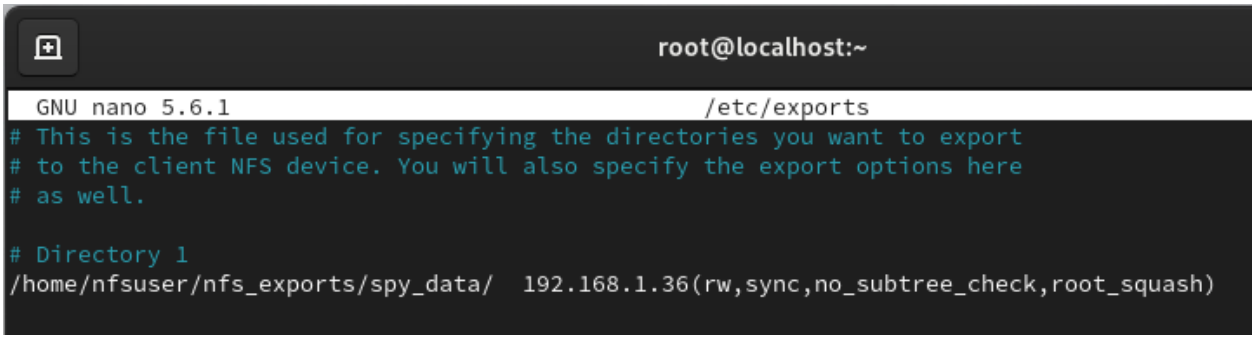
On the NFS server I can even view the connection information by using journalctl and looking for the log prefix nfs-access.

6.4. Specify specific network address that can mount the NFS exports:

Before I was using the * directive for specifying the network devices that can mount the exports. The * directives means that any network host can mount the exports. But this is not acceptable for security reasons since anyone can try to mount the exports. /

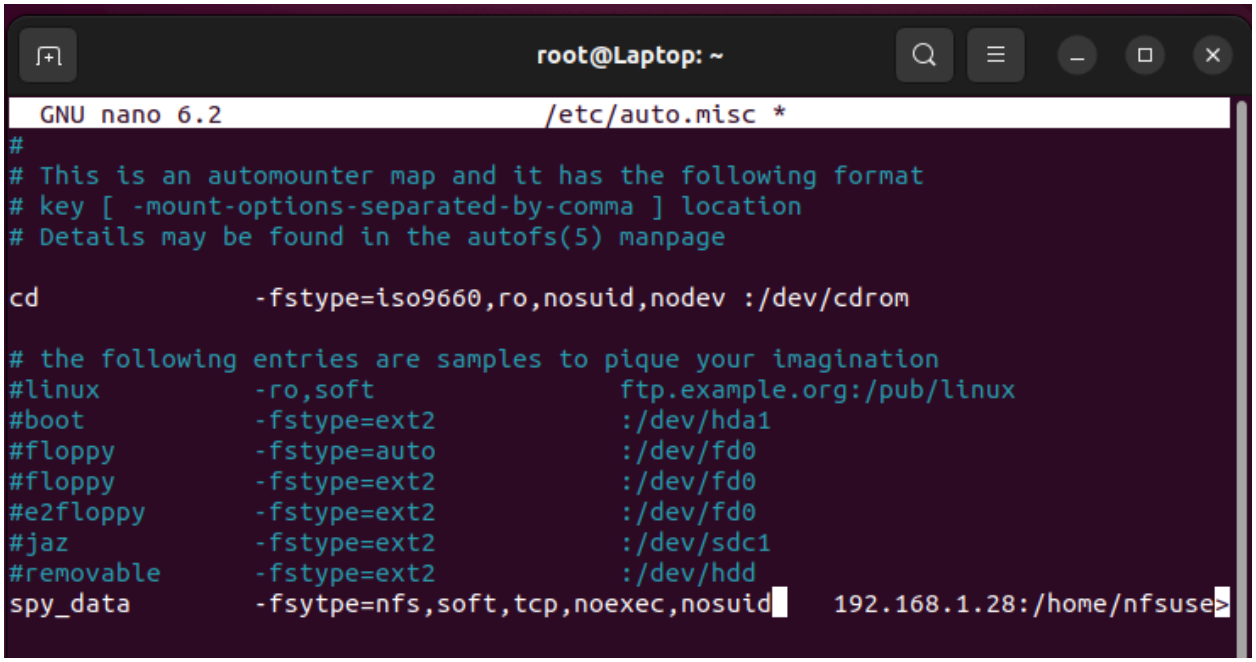


Now I will specify the exact IP address of the client device I want to allow mounting the export on:



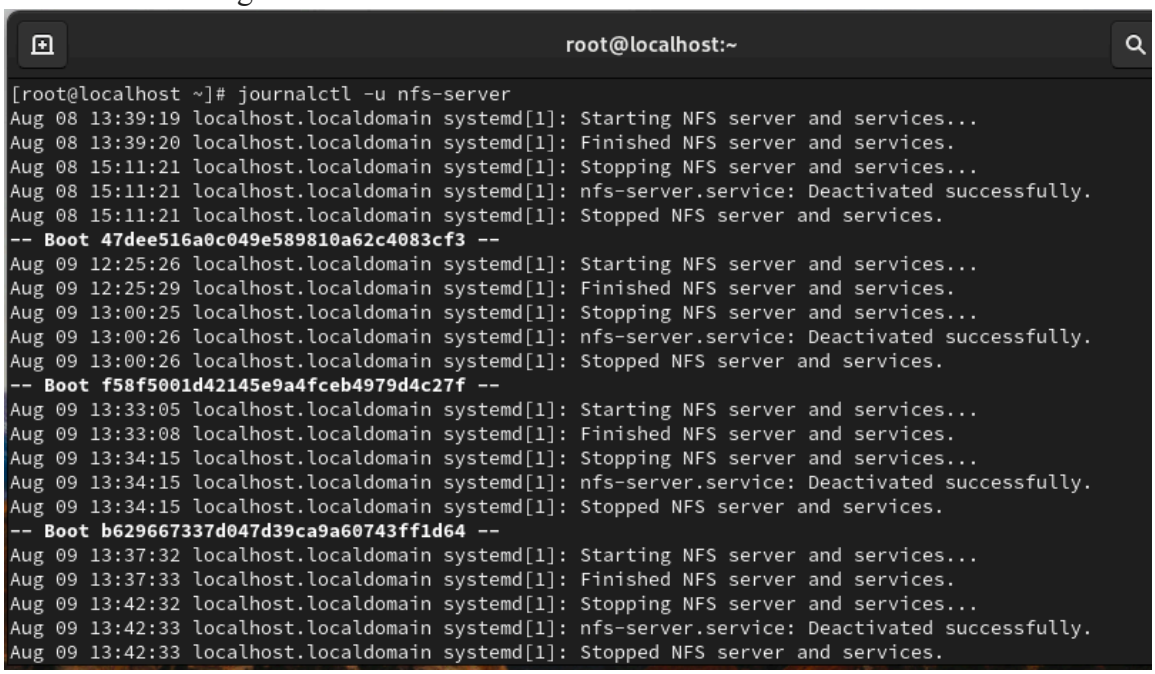
Now only the client device with the IP address 192.168.1.36 can mount the export!

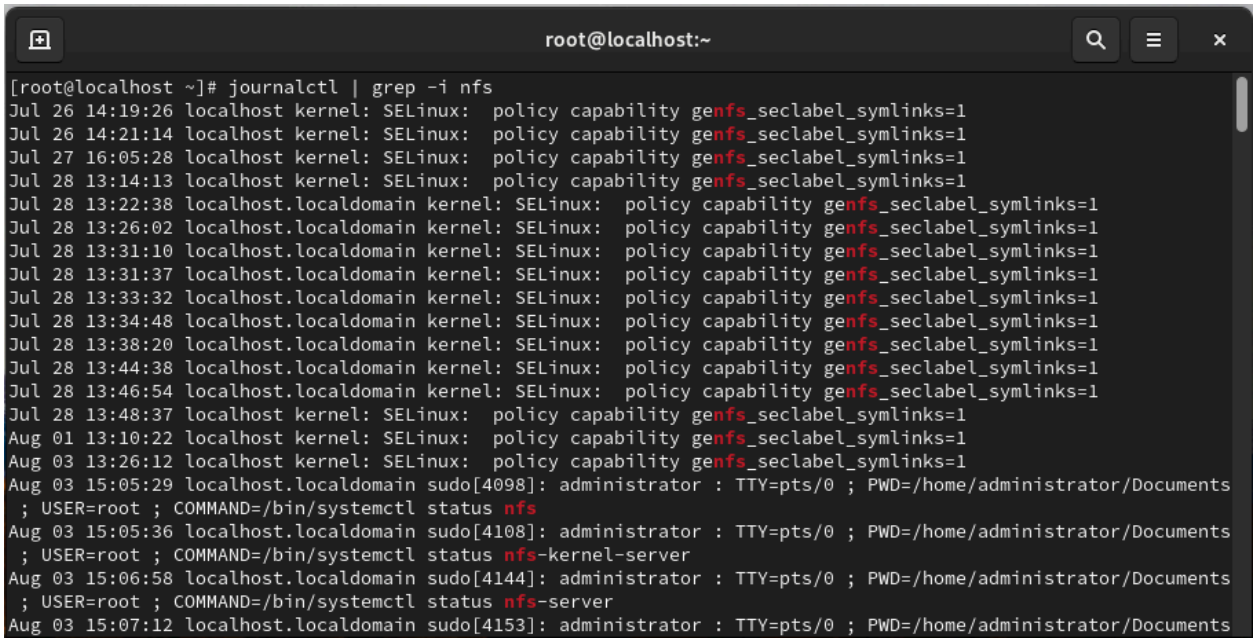
6.5. Use stricter mount options on the client side.



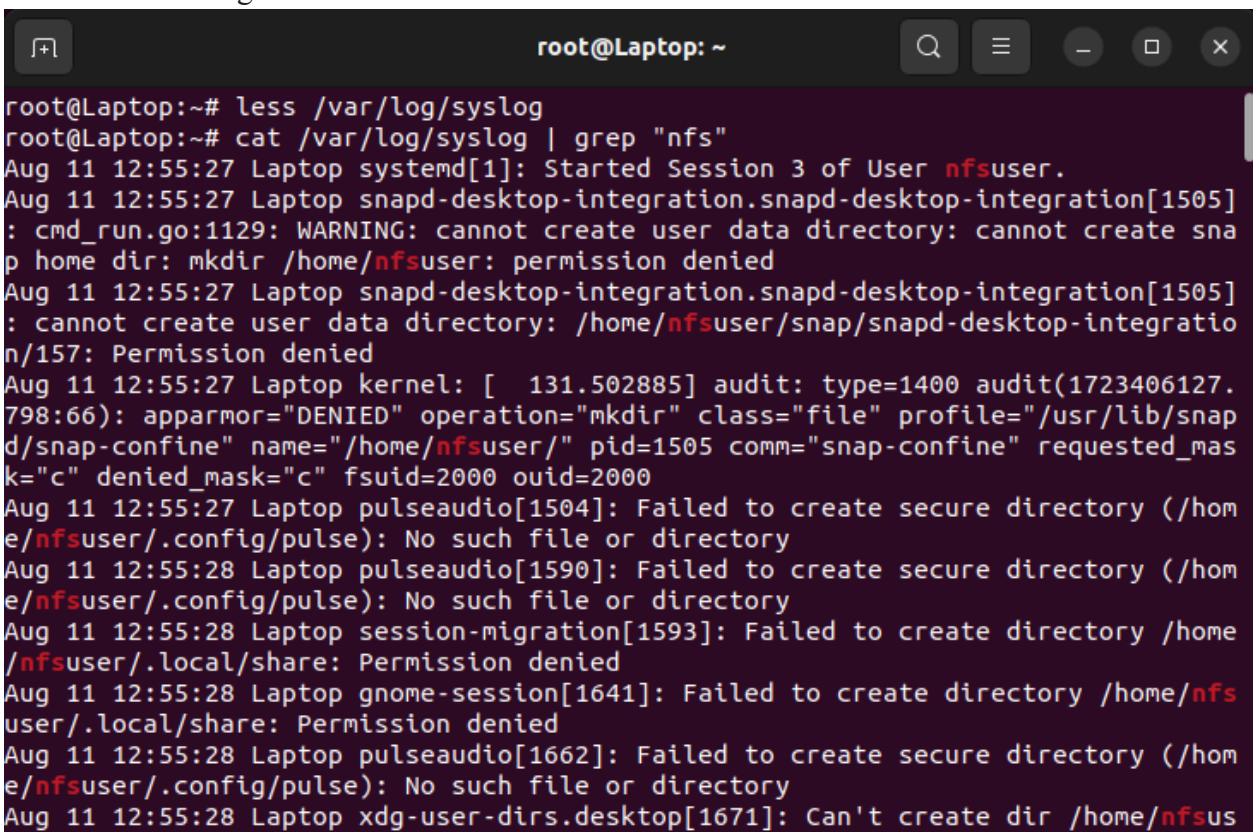
Section #6 Troubleshooting NFS:

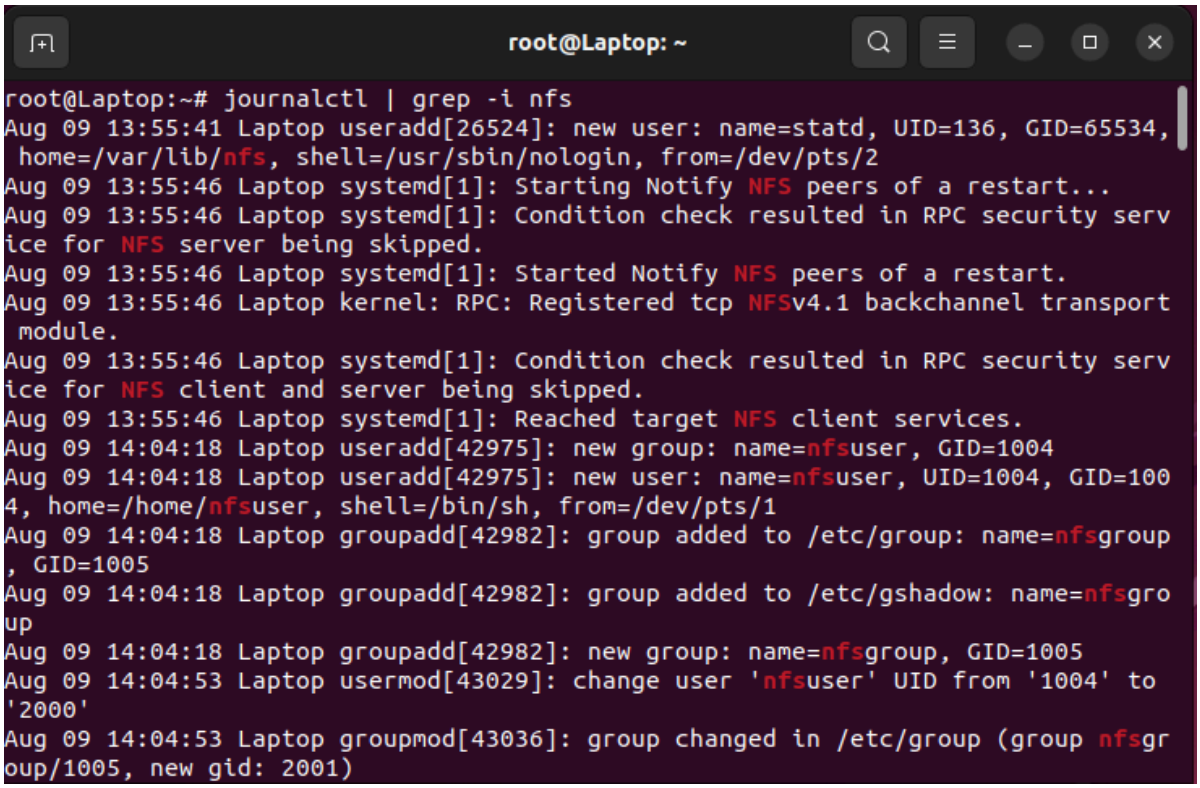
7.1. Check NFS logs on the server:





7.2 Check NFS logs on the client:



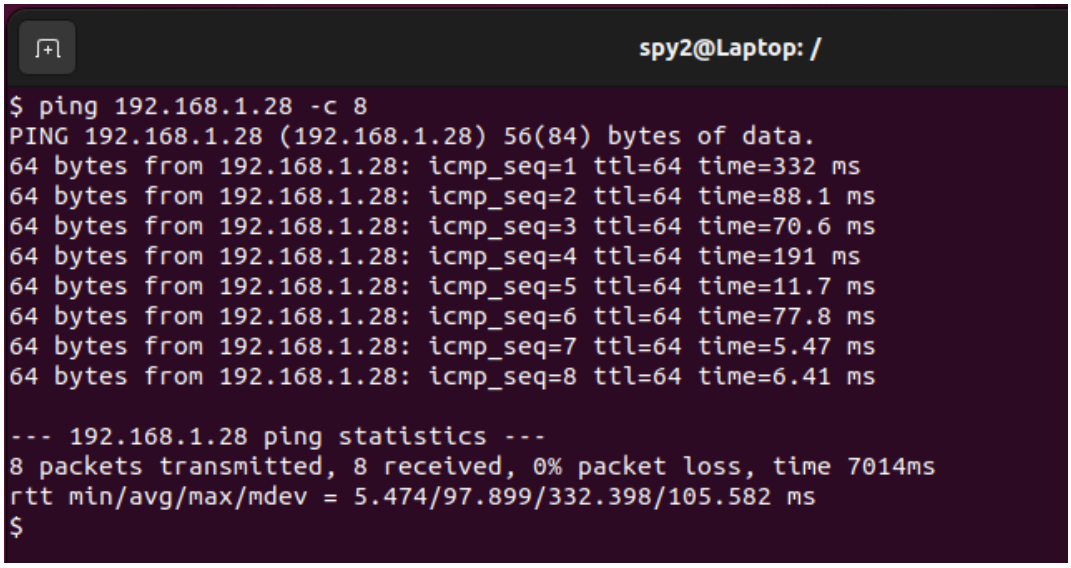


7.3. Check the network connection between the client and server:

Since NFS is a networking service, issues with network connections can occur. Luckily there are plenty of ways you can troubleshoot such issues.

7.3a. Use ping to test if the client and server can reach each other:

- Client to server:



I sent 8 icmp ping packets from the client device to the server. The packets were transmitted with no loss so that means that the client can reach the server over the network.

- Server to client:

```
[administrator@localhost ~]$ ping 192.168.1.36 -c 8
PING 192.168.1.36 (192.168.1.36) 56(84) bytes of data.
64 bytes from 192.168.1.36: icmp_seq=1 ttl=64 time=5.19 ms
64 bytes from 192.168.1.36: icmp_seq=2 ttl=64 time=6.81 ms
64 bytes from 192.168.1.36: icmp_seq=3 ttl=64 time=5.74 ms
64 bytes from 192.168.1.36: icmp_seq=4 ttl=64 time=6.33 ms
64 bytes from 192.168.1.36: icmp_seq=5 ttl=64 time=6.08 ms
64 bytes from 192.168.1.36: icmp_seq=6 ttl=64 time=5.78 ms
64 bytes from 192.168.1.36: icmp_seq=7 ttl=64 time=5.12 ms
64 bytes from 192.168.1.36: icmp_seq=8 ttl=64 time=5.89 ms

--- 192.168.1.36 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7020ms
rtt min/avg/max/mdev = 5.117/5.865/6.812/0.524 ms
[administrator@localhost ~]$
```

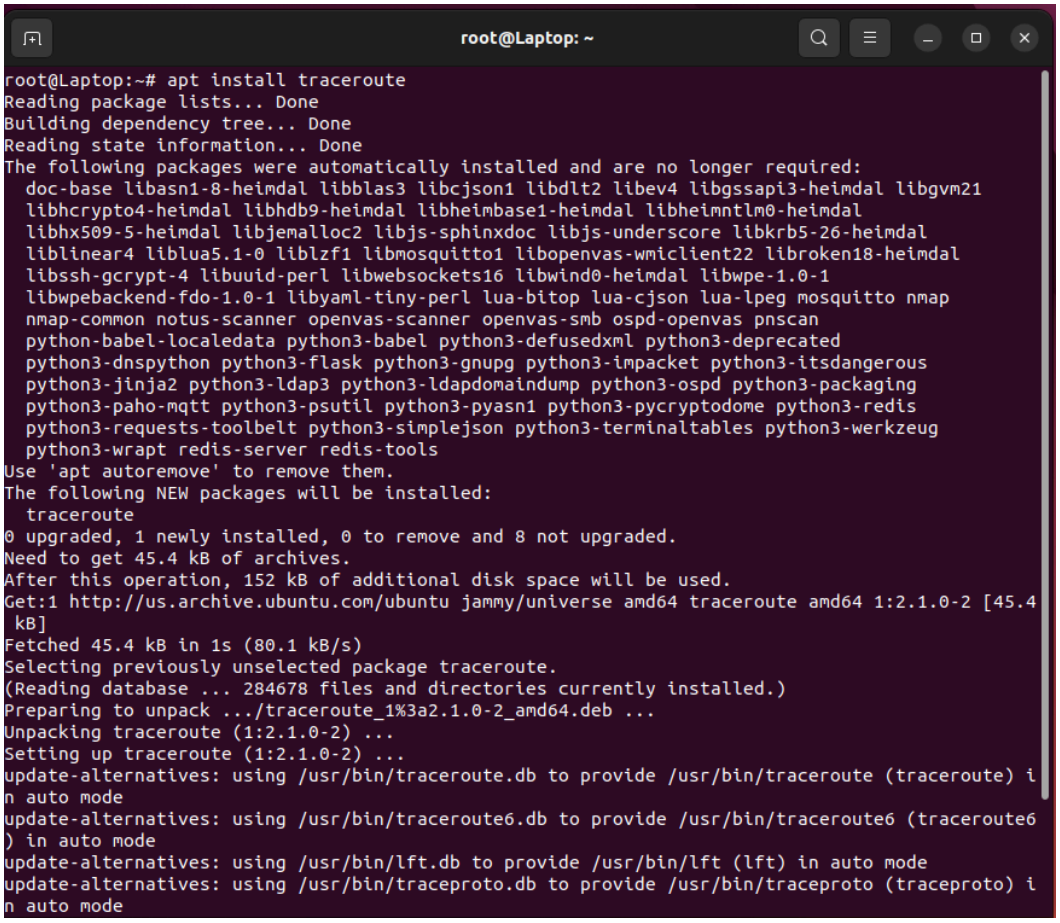
I sent 8 icmp ping packets from the server to the client device. The packets were transmitted with no loss so that means that the server can reach the client over the network.

7.3b. Use traceroute if you suspect there is an issue reaching a remote NFS server:

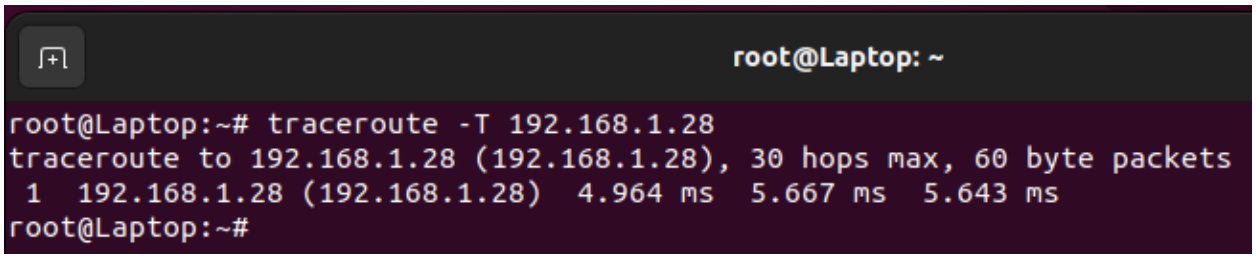
Note that in this lab the server and client are on the same network/subnet so there really is no routing to a remote device. However, my router acts as a switcher to allow devices on the same subnet to communicate.

- Client to server:

I needed to install traceroute on my client device:

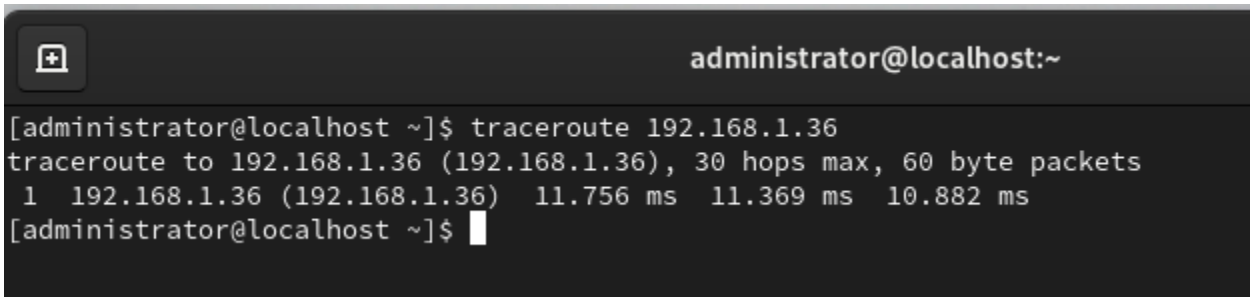


Now time to see how the client device reaches the server:



My client device only needed one hop to reach the NFS server. This makes sense since the client device and server are on the same subnet on my local network. But if the server was located in another city,state, or even country the number of hops would greatly increase.

- Server to client:



Again, only one hop was needed for the server to reach the client device.

7.3c Use netstat on the server to see if the port for NFS is open and working:

Using netstat I can see the connection between the client (192.168.1.36) and the server (192.168.1.28) taking place over tcp port 2049 (which is the default NFS port). This is god since that means that the correct port is being used by the NFS service and connections between the client and server can occur.