NFS µ-HOWTO
for Slackware Linux

Lew Pitcher

lew.pitcher@digitalfreehold.ca

Revision 91, Friday, January 13, 2017

Author's Forward

This document provides a brief summary of the activities necessary to configure NFS on a Linux system. In writing this document, I assume that you have already read the *"NFS Howto"* (at http://tldp.org/HOWTO/NFS-HOWTO/index.html or `/usr/doc/Linux-HOWTOs/NFS-HOWTO`) and are familiar with Network File System terminology and requisites. This document ***is not*** a primer on NFS; it is a cheat-sheet summary, biased to Slackware, for those who know NFS, but do not configure it often.

Overview

There are two systems involved:

- the ***server*** provides the storage space, and `exports` it to clients over the network
- the ***client*** `mounts` the ***server's*** exported storage space, so that it occupies part of the ***client's*** filesystem tree.

On the ***server***:

- Set up `/etc/exports` to name server directories to be exported to NFS clients. See `exports(5)` for format
- Set up `/etc/hosts.allow` to designate client systems permitted to access NFS exports. See `hosts_access(5)` for format
- Set up `/etc/hosts.deny` to designate systems not permitted to access NFS exports. See `hosts_access(5)` for format
- Start the server-side NFS services. On Slackware, you can run `/etc/rc.d/rc.nfsd start` (which is usually started via `/etc/rc.d/rc.inet2`)

On the ***client***:

- wait until all the server's NFS services are started
- start the client-side NFS services. On Slackware, you can run `/etc/rc.d/rc.rpc start` (which is usually started via `/etc/rc.d/rc.inet2`)
- mount remote NFS filesystem(s) to local mount-point(s). See `mount(8)` for format
- optionally, edit `/etc/fstab` to add NFS mounts. See `fstab(5)` and `nfs(5)` for format

Security:

- Unless specifically configured otherwise, the server will use the client UID/GID in file access requests. To avoid problems, ensure that the client UIDs and GIDs match the server UIDs and GIDs. This may require that

client and server `/etc/passwd` entries match, for users using NFS, or the use of NIS to serve network-wide consistent UIDs.
- Client UID `0` / GID `0` is a special case. Normally, the server maps the client "root" user to a known, unprivileged user. You change this on the server `/etc/exports` side.
- Read the `exports(5)` *"User ID Mapping"* section for details on how UID and GID are handled; look for the `root_squash / no_root_squash / all_squash` options to `/etc/exports`

NFS Server-side Configuration

/etc/exports

`exports(5)` says:

> *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.*

The 'export point' named in the server's `/etc/exports` file will be the fully-qualified pathname to a server-side directory that the server wishes to make accessible to one or more NFS clients.

The client system named in the `/etc/exports` file can be one of
- a single IP address
  eg: `192.169.0.5`
- a range of IP addresses specified in net-address/netmask format
  eg: `192.168.0.0/24`
- a single hostname, such as an abbreviated name recognized by the resolver, or a fully-qualified domain name
  eg: `guest`
  eg: `guest.this.lan`
- a "wildcard" domain name, which is a node name containing the wildcard characters '*' and/or '?'
  eg: `*.this.lan` (matches all .this.lan hosts)

eg: `gues?.this.lan` (matches `guest.this.lan` and `guess.this.lan`)

A simple `/etc/exports` example:

```
/home/guest guest.this.lan(rw,sync)
```

which exports the server's `/home/guest` subdirectory to the client system named `guest.this.lan`. The server will grant both read and write requests from the client `(rw)`, and will reply to the client only after the changes have been committed to stable storage `(sync)`.

Multiple clients (and their applicable options) can be given in one `/etc/exports` entry:

```
/home/guest guest.this.lan(rw,sync) guess.this.lan(ro)
```

Options may include:

- **`secure`** *"requests originate on an Internet port less than IPPORT_RESERVED (1024). This option is on by default. To turn it off, specify* **`insecure"`**.
- **`rw`** will *"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"*.
- **`sync`** will cause NFS to *"reply to requests only after the changes have been committed to stable storage"*.
- **`root_squash`** will *"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_subtree_check`** *"disables subtree checking, which has mild security implications, but can improve reliability in some circumstances. 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`**".

`/etc/hosts.deny` & `/etc/hosts.allow`

When servicing client requests, the NFS server first looks through `/etc/hosts.allow` for a rule that ***permits*** that specific client to access the NFS services. If NFS finds a matching rule here, the NFS server permits the client access to the NFS resource.

However, if no such "allow" rule is found in `/etc/hosts.allow`, the NFS server looks through `/etc/hosts.deny` for a rule that ***denies*** that specific client from

access to NFS. If NFS finds a matching rule here, the NFS server denies the client access to the NFS resource; if no "deny" rule is found, then the NFS server grants the client access to the resource.

A typical installation configures `/etc/hosts.deny` to deny services to ALL hosts, and configures `/etc/hosts.allow` with specific exceptions to the blanket rule.

Example /etc/hosts.allow

```
# Only permit hosts on local network (192.168.1.0 - 192.168.1.255) access to NFS
portmap: 192.168.1.
lockd: 192.168.1.
mountd: 192.168.1.
rquotad: 192.168.1.
statd: 192.168.1.
```

Example /etc/hosts.deny

```
# Deny access to NFS services to all hosts, except for those explicitly permitted by hosts.allow
portmap:ALL
lockd:ALL
mountd:ALL
rquotad:ALL
statd:ALL
```

Server-side daemons

1. Start the common RPC *portmap* and *statd* services.
   1. `/sbin/rpc.portmap`
   2. `/sbin/rpc.statd`

2. Start the NFS server daemons
   1. Load the `nfsd.ko` kernel module and mount the `nfsd` filesystem
      `/sbin/modprobe nfsd`
      `/sbin/mount -t nfsd nfsd /proc/fs/nfs`
   2. Re-export all exported directories, to sync `/var/lib/nfs/etab` with `/etc/exports`
      `/usr/sbin/exportfs -r`
   3. Start the RPC Remote Quota Server Daemon
      `/usr/sbin.rpc.rquotad`
   4. Start the Network File Server Daemon, with 8 NFS server threads
      `/usr/sbin.rpc.nfsd 8`
   5. Start the NFS Mount Daemon
      `/usr/sbin.rpc.mountd`

Slackware Notes:

On Slackware, the `portmap` and `statd` services run from `/etc/rc.d/rc.rpc start` (which is executed by `/etc/rc.d/rc.inet2`). However, the preparatory mount of `/proc/fs/nfs` and the export of exported directories, along with the launching of the NFS server daemons `rpc.quotad`, `rpc.nfsd`, and `rpc.mountd`,

run from /etc/rc.d/rc.nfsd start, which is also executed by /etc/rc.d/rc.inet2.

1. To prepare initscripts for execution:
   ```
   chmod u+x /etc/rc.d/rc.rpc
   chmod u+x /etc/rc.d/rc.nfsd
   ```
2. To manually start the initscripts on an already booted system:
   ```
   cd / ; /etc/rc.d/rc.rpc
   cd / ; /etc/rc.d/rc.nfsd
   ```

NFS Client-side Configuration

Client-side daemons

Start the common RPC `portmap` and `statd` services.

1. `/sbin/rpc.portmap`
2. `/sbin/rpc.statd`

Slackware Notes:

On Slackware, the `portmap` and `statd` services run from `/etc/rc.d/rc.rpc start` (which is executed by `/etc/rc.d/rc.inet2`).

1. To prepare the initscript for execution:
   ```
   chmod u+x /etc/rc.d/rc.rpc
   ```
2. To manually start the initscript on an already booted system:
   ```
   cd / ; /etc/rc.d/rc.rpc start
   ```

Client-side Mounts

Like filesystems on local storage devices, you mount NFS exported filesystems onto local mount-points before you use them. You need

1. a local mount-point; a directory over which you will mount the NFS filesystem,
2. the fully-qualified domain name of the NFS server, and
3. the full path of the filesystem exported by the NFS server
4. permission to mount filesystems

First, ensure that the local mount-point exists, and create it if it doesn't

```
[ -d /some/local/directory ] || mkdir -p /some/local/directory
```

Next, mount the NFS remote filesystem on that local mount-point

```
mount -t nfs nfs.server.name:/path/to/exported/filesystem /some/local/directory
```

Optionally, you can build an entry in **`/etc/fstab`** to perform the mount as the system starts (run by `/etc/rc.d/rc.M`)

```
nfs.server.name:/path/to/exported/filesystem /some/local/directory nfs defaults
```

Once the NFS filesystem is mounted on the client system, it will appear to the client as any other local filesystem. Note that this means that the local filesystem will treat file ownership and permissions on the NFS volume the same way it

would on any other local filesystem, including the testing and assignment of file UIDs and GIDs. Systems using NFS must either synchronize their UID/GID assignments, or ensure through other means that file ownership issues are accounted for.