

Perform uniform mounting with generic NFS

Automatically consolidate many different NFS versions into a uniform mount

Skill Level: Introductory

[Chinmay Soman \(chinmay.soman@in.ibm.com\)](mailto:chinmay.soman@in.ibm.com)

Associate Software Engineer

IBM

[Abhidnya Chirmule \(achirmul@in.ibm.com\)](mailto:achirmul@in.ibm.com)

System Software Engineer

IBM

11 Feb 2009

To efficiently achieve uniform mounting in the presence of multiple, simultaneous NFS version exports, you need a generic NFS mount utility. Learn how a generic NFS mount utility can help reduce handling multiple NFS versions and simplify the management of those versions. The article describes the concept of the generic NFS mount, outlines the advantages and applications of the system, and gives some overall design details.

The Network File System (NFS) is a well-used distributed file system that enables file operations to be performed on a server from remote clients. The server makes its directories or filesystems available to the rest of the world using an operation known as `export`. To access these directories, the client `mounts` the exported directories or filesystems to its local directory hierarchy. Inside the mounted directory, clients access the remote files as if they were stored locally on the machine. Currently, NFS sports three available versions for exporting and mounting of directories or filesystems: versions 2, 3, and 4.

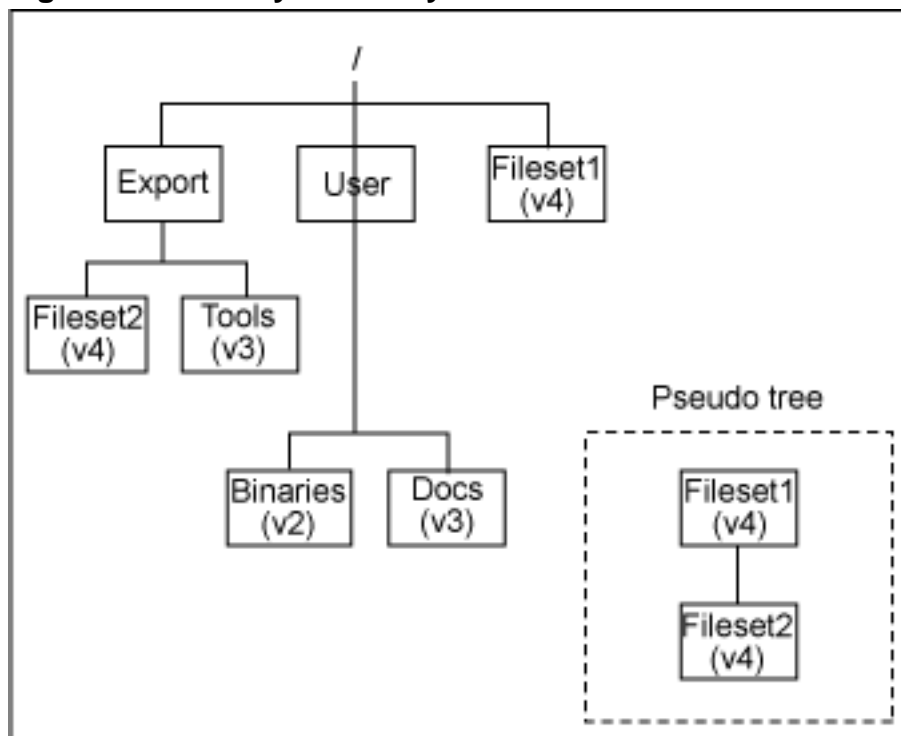
In this article, we show you how to use a generic NFS mount to consolidate exporting and mounting of all existing NFS version into a single, seamless mechanism. Let's consider a scenario where the server has exported directory

entries for all three versions of NFS. Currently, for the client to access all these entries, it has to separately mount each of these entries at different mount points. Although NFS version 4 provides a pseudo-tree mechanism that enables a single mount of all NFSv4 exported entries, it is applicable only to the entries made by that version. The client has to separately mount version 2 and 3 entries along with a single mount for version 4 (in case the pseudo-tree exists).

The generic NFS mount utility is essentially a wrapper for the `mount` command that enables the user to mount all possible exported entries from a particular server using a single command. Since any change on the NFS server is not desirable, internally this wrapper performs separate `mounts` on the client machine that are transparent to the user.

Consider the directory hierarchy on the NFS server as shown in Figure 1:

Figure 1. Directory hierarchy on the server



In this current scenario, the following `mounts` are required on the client machine:

- **One NFSv4 mount:** This mounts the NFSv4 pseudo-tree (*Fileset1* and *Fileset2*). The NFSv4 pseudo-tree feature allows the NFSv4 client to perform only one `mount` operation for all exported entries in the pseudo-tree.
- **Two NFSv3 mounts:** This mounts *Tools* and *Docs*.
- **One NFSv3 mount:** This mounts *Binaries*.

Using the generic NFS mount utility, these mounts are reduced to a single mount. Here is the command:

```
gennfsmount <NFS server> <mountpoint>
```

The advantages of using the generic NFS mount include:

- Users need to do only one mount operation to access all the information from the server; this was not previously supported.
- Consider a server that exports old installation filesets in versions 2 and 3 and new filesets in version 4. Clients can access all these filesets and perform installations using a single operation.
- Searching a file on a particular server and in a particular export directory/filesystem can be enhanced using this generic utility.
- Because the utility handles automatic segregation of version-based NFS exports, it reduces the users' efforts to manage these different mount points.
- NFS administrators can retain the old NFS version exports on the server since they would be available along with the newer version data items in a single `mount` command.

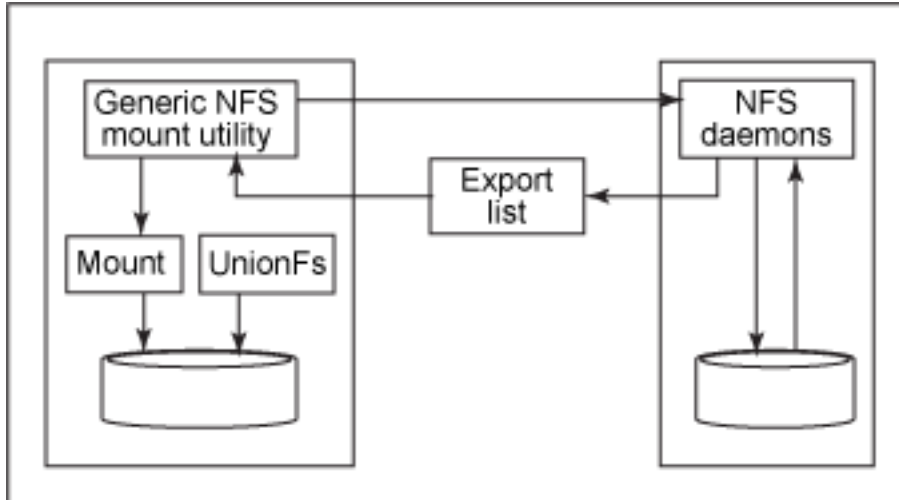
Similar output can be obtained using the `automount` utility, but it requires administrative overhead to configure the `automount` map files for the desired NFS mounts. The `hostmap` feature of `automount` claims to mount all exported entries from a server without necessitating any administrative configuration, but problems have been identified when it comes to mounting NFSv4 entries using `hostmap`. In addition, this operation is performed for all the servers listed in the `/etc/hosts` file. No solution exists for mounting all exported entries of one particular server at a single point.

Overall, the generic NFS mount mechanism is convenience for the user. Next, let's look at some of the design decisions when implementing a generic NFS mounter.

Design implementation details

Let's look at the basic architecture of a generic NFS mount system. The generic NFS mounter internally sends a request to the server asking for all the exported entries.

Figure 2. Generic NFS mount requests all exported entries



On receiving a reply from the server, the algorithm in Listing 1 is followed:

Listing 1. Algorithm for generic NFS mount

```

Start
    Create temporary directories for all versions.

    Initialize list of mount security flavors.
    Embed this list in each internal mount operation.

    For each item in the export-list
    Do
        Mount internally for every NFSv2 and NFSv3 export
        Update internal log
    End for

    Mount internally only once for NFSv4 export.
    Update internal log

Stop
  
```

The `security flavors` is a comma-separated list of security methods (`sys`, `krb5`, `krb5i`, and so on) used during `read/write` operations under the mount point. The list is used to match the security method supported by the server and employed during subsequent system calls under this mount point. A similar mechanism can be employed for the NFS version of the corresponding exported entry.

When all the internal mounts are completed, a unification of these internal directories is done using UnionFS as shown in Listing 2.

Listing 2. Unifying internal directories with UnionFS

```

mount -t unionfs -o
dirs=<temp_dir1>[:<temp_dir2>...]
none <mount-point>
  
```

Remember the [previous scenario](#)? The one NFSv4 mount for v4 pseudo-tree/two

NFSv3 mounts for Tools and Docs/one NFSv3 mount for Binaries were on the client machine and we used the generic NFS mount utility—`gennfsmount <NFS server> <mountpoint>`—to reduce them to a single mount. Now in this case, the following temporary directories are created:

- /tmp/NFSv4
- /tmp/NFSv3/Tools
- /tmp/NFSv3/Docs
- /tmp/NFSv2/Binaries

These are then combined using `unionfs` as in Listing 3:

UnionFS

UnionFS is a file system service for Linux® and FreeBSD that implements a union mount for other file systems by allowing files and directories of separate file systems (known as branches) to be transparently overlaid so that they form a single coherent file system. Contents of directories with the same path will be seen together in a single merged directory in the resultant file system. UnionFS for Linux has two versions:

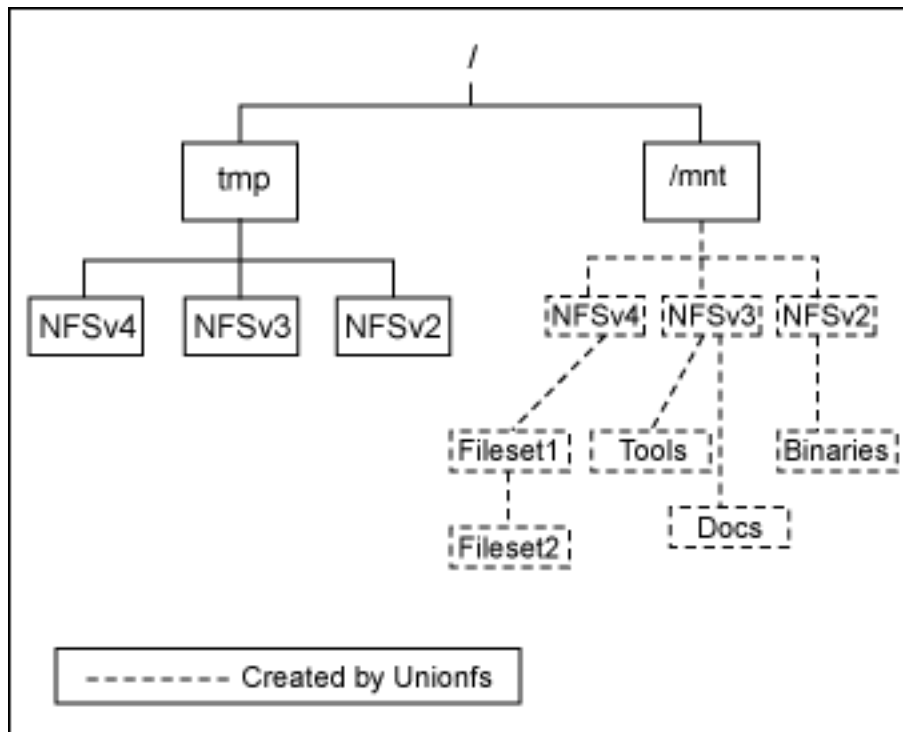
- Version 1.x is a standalone that can be built as a module.
- Version 2.x is newer; it has been redesigned and reimplemented and is included in Andrew Morton's Linux `-mm` tree (so it should get into the main source tree).

Listing 3. Merging using unionfs

```
mount -t unionfs -o dirs=/tmp/NFSv4:/tmp/NFSv3:/tmp/NFSv2 none /mnt
```

This results in the following directory hierarchy:

Figure 3. Hierarchy resulting from unionfs



Next, let's look at how we can use the system.

Using the system

In this scenario, the NFS server exports different entries of different NFS versions. At the client side though, only one mount is performed: the generic NFS mount. Figure 4 shows exported entries at the server.

Figure 4. Exports at the server

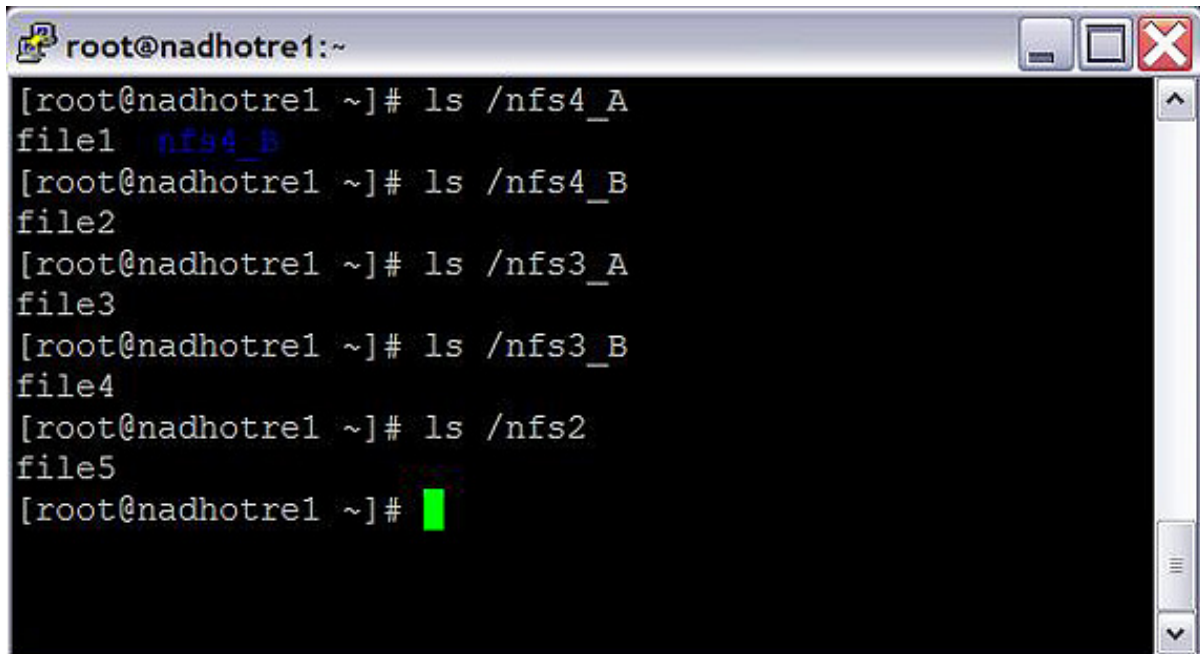
```

root@nadhotre1:~
[root@nadhotre1 ~]# cat /etc/exports
/nfs4_A *(rw,no_root_squash,fsid=0)
/nfs4_A/nfs4_B *(sync,rw,nohide,no_root_squash)
/nfs3_A *(rw,no_root_squash)
/nfs3_B *(rw,no_root_squash)
/nfs2 *(rw,no_root_squash)
[root@nadhotre1 ~]#
[root@nadhotre1 ~]# exportfs -v
/nfs4_A/nfs4_B <world>(rw,wdelay,nohide,no_root_squash,no_subtree_check)
/nfs4_A <world>(rw,wdelay,no_root_squash,no_subtree_check,fsid=0)
/nfs3_A <world>(rw,wdelay,no_root_squash,no_subtree_check)
/nfs3_B <world>(rw,wdelay,no_root_squash,no_subtree_check)
/nfs2 <world>(rw,wdelay,no_root_squash,no_subtree_check)
[root@nadhotre1 ~]#
    
```

Here the server exports five NFS entries with different NFS versions. **nfs4_A** and **nfs4_B** form a NFSv4 pseudo-tree (**/nfs4_A** and **/nfs4_A/nfs4_B**). The remaining are versions 2 and 3 NFS exports.

Figure 5 shows files existing at the server.

Figure 5. Files existing at the server

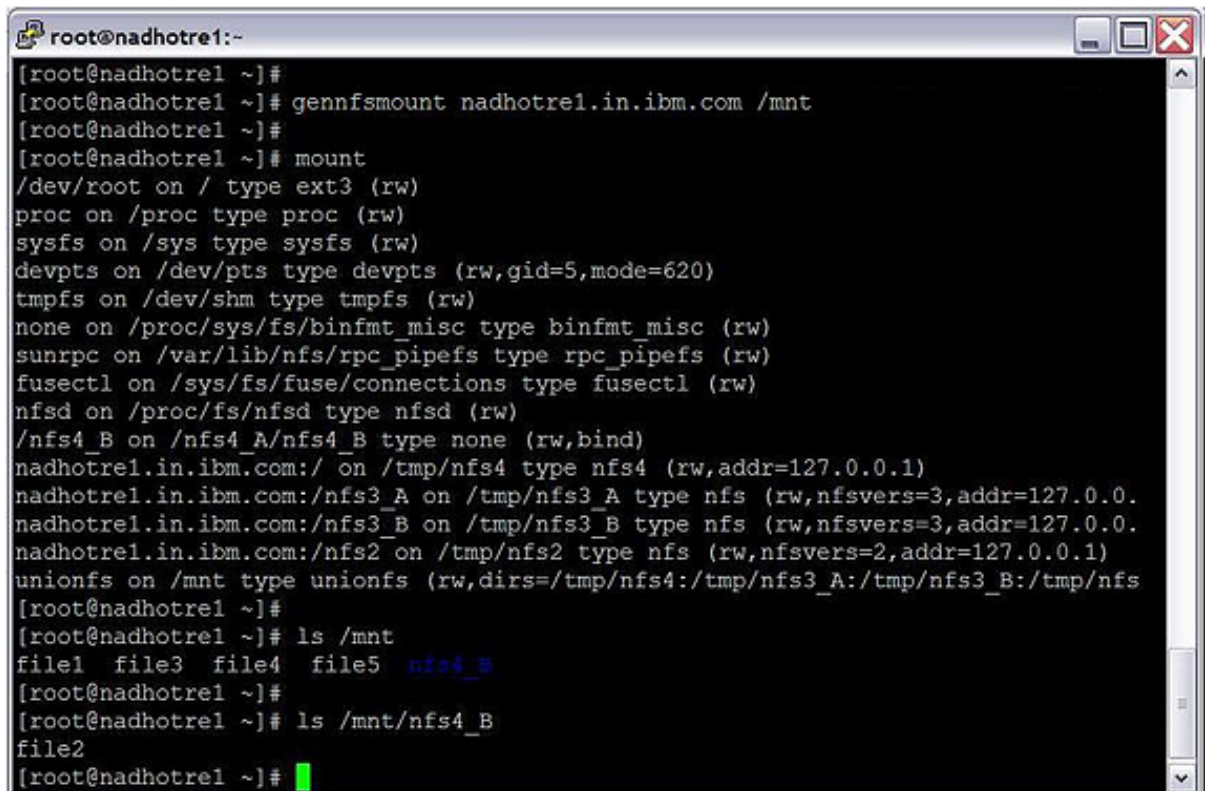
A terminal window titled 'root@nadhotre1:~' with standard window controls (minimize, maximize, close) in the top right. The terminal shows a series of 'ls' commands and their outputs for different NFS export directories. The output for /nfs4_A is 'file1' and 'nfs4_B'. The output for /nfs4_B is 'file2'. The output for /nfs3_A is 'file3'. The output for /nfs3_B is 'file4'. The output for /nfs2 is 'file5'. The prompt is currently at the end of the last command.

```
root@nadhotre1:~  
[root@nadhotre1 ~]# ls /nfs4_A  
file1  nfs4_B  
[root@nadhotre1 ~]# ls /nfs4_B  
file2  
[root@nadhotre1 ~]# ls /nfs3_A  
file3  
[root@nadhotre1 ~]# ls /nfs3_B  
file4  
[root@nadhotre1 ~]# ls /nfs2  
file5  
[root@nadhotre1 ~]#
```

In the current scenario, all these files would be available to the client under different mounted directories through individual mount operations. However, the proposed system enables the user to access all these files in a single hierarchy with a single mount operation.

Figure 6 shows the scenario as seen from the client after the generic NFS mount:

Figure 6. Output of generic NFS mount



```
root@nadhotre1:~#  
[root@nadhotre1 ~]# gennfsmount nadhotre1.in.ibm.com /mnt  
[root@nadhotre1 ~]#  
[root@nadhotre1 ~]# mount  
/dev/root on / type ext3 (rw)  
proc on /proc type proc (rw)  
sysfs on /sys type sysfs (rw)  
devpts on /dev/pts type devpts (rw,gid=5,mode=620)  
tmpfs on /dev/shm type tmpfs (rw)  
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)  
sunrpc on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw)  
fusectl on /sys/fs/fuse/connections type fusectl (rw)  
nfsd on /proc/fs/nfsd type nfsd (rw)  
/nfs4_B on /nfs4_A/nfs4_B type none (rw,bind)  
nadhotre1.in.ibm.com:/ on /tmp/nfs4 type nfs4 (rw,addr=127.0.0.1)  
nadhotre1.in.ibm.com:/nfs3_A on /tmp/nfs3_A type nfs (rw,nfsvers=3,addr=127.0.0.  
nadhotre1.in.ibm.com:/nfs3_B on /tmp/nfs3_B type nfs (rw,nfsvers=3,addr=127.0.0.  
nadhotre1.in.ibm.com:/nfs2 on /tmp/nfs2 type nfs (rw,nfsvers=2,addr=127.0.0.1)  
unionfs on /mnt type unionfs (rw,dirs=/tmp/nfs4:/tmp/nfs3_A:/tmp/nfs3_B:/tmp/nfs  
[root@nadhotre1 ~]#  
[root@nadhotre1 ~]# ls /mnt  
file1 file3 file4 file5 nfs4_B  
[root@nadhotre1 ~]#  
[root@nadhotre1 ~]# ls /mnt/nfs4_B  
file2  
[root@nadhotre1 ~]#
```

As you can see, Figure 6 shows multiple internal mounts performed by the generic NFS mounter. The directory where all the NFS mounts are merged is /mnt.

Summary

We've shown you the architecture and the mechanism behind a generic NFS mounter, a utility that will undoubtedly help the NFS clients by providing easier, one-point access to the files on the NFS server and by offering a more consolidated view of the NFS space.

Resources

Learn

- "[Unionfs: Bringing Filesystems Together](#)" (Linux Journal, December 2004) gives details on how UnionFS works.
- For more on UnionFS, see:
 - "[Remote computing with a Linux application server farm](#)" (February 2007, developerWorks).
 - "[Anatomy of Linux flash file systems](#)" (May 2008, developerWorks).
 - "[Assess system security using a Linux LiveCD](#)" (July 2005, developerWorks).
- The SourceForge "[Linux NFS HOWTO](#)" describes best practices for configuring Linux NFS properly in production environments (including server and client configuration, as well as security and performance tuning).
- In the [developerWorks Linux zone](#), find more resources for Linux developers (including developers who are [new to Linux](#)), and scan our [most popular articles and tutorials](#).
- See all [Linux tips](#) and [Linux tutorials](#) on developerWorks.
- Stay current with [developerWorks technical events and Webcasts](#).

Get products and technologies

- With [IBM trial software](#), available for download directly from developerWorks, build your next development project on Linux.

Discuss

- Get involved in the [developerWorks community](#) through blogs, forums, podcasts, and spaces.

About the authors

Chinmay Soman

Chinmay Soman is a software engineer with IBM Systems and Technology Labs in Pune, India. He is currently involved in CTDB/Panache research activities. He has worked with the NFS team on the development and management of NFS on AIX. He holds a bachelor's degree in Computer Science and Engineering.

Abhidnya Chirmule

Abhidnya Chirmule is a systems software engineer with IBM Systems and Technology Labs in Pune, India. She is currently a member of Scale Out NAS Team. She has worked with the NFS team on the development and management of NFS on AIX. She holds a bachelor's degree in Computer Engineering.

Trademarks

IBM, the IBM logo, ibm.com, DB2, developerWorks, Lotus, Rational, Tivoli, and WebSphere are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. See the current list of [IBM trademarks](#).

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.