



Linux Administration

The Network File System

Lecturer:
Di Dio Lavore, Galtarossa
Politecnico di Milano
www.polimi.it

© 2003 - Di Dio Lavore, Galtarossa

Summary



-
- Introduction
 - Security and NFS
 - Server-side NFS
 - ▶ The `/etc/exports` file
 - ▶ Common export options
 - ▶ `exportfs` command
 - ▶ `nfsd` daemon
 - Client-side NFS
 - ▶ `mount` command
 - ▶ `umount` command
 - Automatic mounting
 - ▶ `automount`
 - ▶ `amd`
 - `nfsstat`
 - X-based Interfaces

Introduction



- The Network File System (NFS) allows you to share filesystems among computer
- NFS is transparent to users and is “stateless”
- NFS consists of a number of components:
 - ▶ mounting protocol and mount server
 - ▶ daemons that coordinate basic file service
 - ▶ several diagnostic utilities
- A portion of both server-side and client-side software resides in the kernel

Introduction (2)



- NFS runs on top of Sun’s RPC (Remote Procedure Call) protocol
 - ▶ UDP and TCP are supported as the underlying transport protocol but Linux server-side TCP support is experimental
- An NFS server will enforce disk quotas, but users cannot view their quota information unless **rquotad** server is running on the remote server
- A client must explicitly mount an NFS filesystem before using it

NOTE:

NFS server does not keep track of which clients have mounted each filesystem but it simply discloses a secret “cookie” that provides a way for the client to access the mounted directory

Security and NFS



- NFS has great potential to cause security problems but Linux supports a number of features designed to reduce and isolate this type of problem:
 - ▶ access to NFS volumes is granted by the file `/etc/exports`
 - ▶ deny access to the `portmap` service to everyone by editing the `/etc/hosts.deny` file
 - ▶ enable access for trusted hosts and subnets in the `/etc/hosts.allow` file
 - ▶ ...
- As on local filesystem, file-level access control on NFS filesystem is managed according to UID, GID and file permissions
- NFS filesystems should not be exported to nonlocal machines

Security and NFS (2)



- By default, the Linux NFS server intercepts incoming requests made on behalf of UID 0 and changes them to look as if they came from some other user (squashing root)
- A placeholder account named `nobody` is defined specifically to be the pseudo-user as whom a remote root masquerades on an NFS server

NOTE:

The only real effect of UID mapping is to prevent access to files that are owned by root and not readable or writable by the world

Server-side NFS



- A server is usually said to “export” a directory when it makes the directory available for use by other machines
- The process used by clients to mount a filesystem (that is, to learn its secret cookie) is completely separated from the process used to access files:
 - ▶ separate protocol
 - ▶ different daemons (`rpc.mountd` and `rpc.nfsd`)
- On an NFS server these daemons should start when system boots and they should remain running as long as the system is up
 - ▶ different platforms have different names of the NFS server startup script (for example Red Hat uses `/etc/init.d/nfs`)

Server-side NFS (2)



- `rpc.mountd` and `rpc.nfsd` share a single access control database that tells which filesystems should be exported and which clients may mount them:
 - ▶ the operative copy of this database is kept in the `/usr/lib/nfs/xtab` binary file in addition to tables internal to the kernel
 - ▶ `exportfs` is the command to add and modify entries (`-u` option to remove)
- `/etc/exports` is the canonical, human-readable list of exported directories (all filesystems included in this file are exported at boot time)

Server-side : the /etc/exports file



- In this file the clients that may access a given filesystem are presented in a whitespace-separated list
- There are 4 types of client specifications:
 - ▶ hostname (individual hosts)
 - ▶ netgroup (NIS netgroups)
 - ▶ wild cards
 - ▶ IP networks
- Each client is followed by a list of options and there is no way to list multiple clients for a single set of option
- Examples:

```
/share          192.168.0.0/255.255.255.0 (rw)
/share/tmp      tower333 (async)
/usr/share      tower333 (rw,no_root_squash) *.home.net (ro)
```

Export options



- Common export options:
 - ▶ `ro` : exports read-only
 - ▶ `rw` : exports for reading and writing
 - ▶ `root_squash` : maps UID 0 and GID 0 to the values specified by `anonuid` and `anongid`
 - ▶ `no_root_squash` : allows normal access by root
 - ▶ `all_squash` : maps all UIDs AND GIDs to their anonymous versions
 - ▶ `anonuid=nnn` : specifies the UID to which remote roots should be squashed
 - ▶ `anongid=nnn` : specifies the GID to which remote roots should be squashed
 - ▶ `secure` : requires remote access to originate at a privileged port
 - ▶ `insecure` : allows remote access from any port
 - ▶ `noaccess` : prevents access to this dir and its subdirs
 - ▶ `async` : allows asynchronous writes

Common export options (2)



NOTE:

- ▶ Linux's NFS server has the unusual feature of allowing subdirectories of exported directories to be exported with different options and the **noaccess** option is provided to "unexport" subdirectories that you would rather not share

Example:

```
/share          192.168.0.0/255.255.255.0(rw)
/share/private  (noaccess)
```

- ▶ Linux servers may insist that requests come from a privileged port if filesystem is exported with the **secure** option, which is on by default

exportfs command



- The **exportfs** command is used to maintain the current table of exported file systems
- Normally the **/var/lib/nfs/xtab** file is initialized with the list of all file systems named in **/etc/exports**
- Administrators can choose to add and delete individual file systems without modifying **/etc/exports** using **exportfs**
- Any export requests which identify a specific host (rather than a subnet or netgroup etc) are entered directly into the kernel's export table as well as being written to **/var/lib/nfs/xtab**

exportfs command



- Example:
 - ▶ The following adds all directories listed in `/etc/exports` to `/var/lib/nfs/xtab` and pushes the resulting export entries into the kernel:
- ▶ To export the `/share/tmp` directory to host `tower333`, allowing asynchronous writes, one would do this:

```
exportfs -a
```

```
exportfs -o async tower333:/share/tmp
```

nfsd daemon



- Once a client's mount request has been validated by `rpc.mountd`, the client can request various filesystem operations that are handled on the server side by `rpc.nfsd`
- `nfsd` takes a numeric arguments that specifies how many threads to fork
 - ▶ if the number is too low or too high, NFS performance can suffer
 - ▶ `nfsstat` command can check for performance problems
- You can change the number of `nfsd` processes by editing the appropriate startup script in `/etc/init.d` (Red Hat: `nfs`, SuSE: `nfsserver`,...)



- Before an NFS file system can be mounted, it must be properly exported
- **showmount** command can verify that a server has exported its filesystems (`showmount -e hostname`):

```
[root@tower1700]# showmount -e tower1700
Export list for tower1700:
/share 192.168.0.0/255.255.255.0
```

- NFS filesystem are mounted in much the same way as local disk filesystems (**mount** command)
- After mounting, an NFS-mounted filesystem is accessed in the same way as a local filesystem

mount command



- **mount** maps a directory within the existing *file tree*, called the *mount point*, to the root of the newly attached *filesystem*
- The standard form of the command, is:

```
mount -t type device dir
```

- ▶ This tells the kernel to attach the *filesystem* found on device (which is of type *type*) at the directory *dir*
- ▶ The pathname *dir* refers to the root of the *filesystem* on device
- The **mount** command understands the notation *host:directory* and maps the remote *directory* on the remote *host* into a directory within the local file tree
- In the case of an *nfs* mount, device may look like `tower1700.home.net:/share`

mount command (2)



- An example:

```
mount -o rw tower1700.home.net:/share /mnt/nfsshare
```

This command installs the remote filesystem `/share` exported by the remote server `tower1700.home.net` under the path `/mnt/nfsshare` on the local machine

- A list of the *filesystems* that are customarily mounted on a particular system is kept in the `/etc/fstab` file (file system table)
- `mount -a -t nfs` mounts all file systems of the type `nfs` in `fstab`
- The mount can be tested with `df` just as you would test a local filesystem

mount command (3)



- Options are specified with a `-o` flag followed by a comma separated string of options
- Some options are the following:
 - ▶ `intr`: allows users to interrupt blocked operations
 - ▶ `rw`: mount the file system read-write
 - ▶ `ro`: mount the file system read-only
 - ▶ `bg`: if the mount fails, keeps trying it in the background and continues with other mount requests
 - ▶ `tcp`: selects transport via TCP. UDP is the default
 - ▶ `hard`: if a server goes down, causes operations that try to access it to block until the server comes back up
 - ▶ `soft`: if a server goes down, causes operations that try to access it to fail and return an error
 - ▶ ...



- NFS partitions can be unmounted with the **umount** command
 - ▶ it has the same syntax of **mount** command
 - ▶ it detaches the *filesystem* mentioned from the file hierarchy
 - ▶ the *filesystem* is specified by giving the directory where it has been mounted

NOTE:

- ▶ You cannot unmount a *filesystem* that is “busy” (open file...)
- ▶ **lsdf** command finds processes with open files on the filesystem
- ▶ `umount -f` forces the filesystem to be unmounted

Automatic mounting



- You can list mounts that are part of a system’s permanent configuration in the `/etc/fstab` file so that they are mounted automatically at boot time
- Alternatively, **mount** can be handled by an automatic mounting service such as **automount** or **amd**
- Automatic mounting is the best choice because:
 - ▶ in large networks maintaining `/etc/fstab` on a few hundred machines can be tedious
 - ▶ when an important server crashes a copy of the partition can be mounted from a backup server
 - ▶ ...

Automatic mounting (2)



- An automount daemon mounts filesystems when they are referenced and unmounts them when they are no longer needed
 - ▶ it minimizes the number of active mount points
 - ▶ it is transparent to users
 - ▶ it can use a list of “replicated” filesystems
- The automounter mounts a virtual filesystem driver on the directories designated as location for automatic mounting
- The virtual filesystem driver is a kernel-resident driver called *autofs*

automount



- **automount** is a background process that configures a single mount point for *autofs*
- The startup script `/etc/init.d/autofs` parses a *master* file (usually `/etc/auto.master`) and runs **automount** for each of the listed mount points

NOTE:

- ▶ it's typical to see a running instance of **automount** for each automatic mount point that has been configured
- ▶ the **autofs** script tells the kernel how to configure the *autofs* filesystem

automount (2)



- The **autofs** script accepts a single parameter on the command line: `start`, `stop`, `reload`, `restart`, `status`
- The **auto.master** file associates a mount point with a “map”
 - ▶ a map translates the directory name accessed into a command line that **mount** can use to perform the real mount
 - ▶ a map can be a text file or an executable program
- If a map file is executable, it’s assumed to be a script or program that dynamically generates automounting information
 - ▶ it allows you to define a site-wide automount configuration file in a format of your own choice

automount (3)



- When a user references a directory associated to the *autofs* filesystem the **automount** process figures out which filesystem to mount by consulting a map, then performs the mount
- **mount** and **ps** show the *autofs* filesystems and the automount processes they are attached:
 - ▶ `mount`
 - or
 - ▶ `ps auxw | grep automount`

The master file



- The *master* file (usually `/etc/auto.master`) lists the directories that should have *autofs* filesystem mounted on them
 - ▶ it associates a map with each directory
 - ▶ it can specify additional option for the **mount** command
- The path of the directory is specified in this file, not in the map file itself
- An example

```
# Directory      Map              Options
/chimchim       /etc/auto.chim   -hard,bg,intr
```

The map file



- The map file lists the filesystems that have the same path specified in the *master* file
- An example of `/etc/auto.chim` (corresponding to the example above):

```
users      chimchim:/chimchim/users
devel      -soft chimchim:/chimchim/devel
info       -ro  chimchim:/chimchim/info
```

- ▶ This example tells automount that it can mount the directories `/chimchim/users`, `/chimchim/devel` and `/chimchim/info` from the host `chimchim`
- ▶ The paths on `chimchim` and the local host will be identical, but this correspondence is not required

amd



- **amd** is an elaborate riff on the automounter concept

- **amd** offers the following advantages over **automount**:
 - ▶ it sends “keep alive” queries to remote servers at regular intervals and maintains a list of servers that are accessible
 - ▶ it has been ported to over 20 versions of Unix
 - ▶ it offers support for a number of mount types that are not supported by **automount** (for example *union*)
 - ▶ it includes a query-and-manipulation tool (**amq**)
 - ▶ **amd**'s map syntax is more generic than that of **automount**
 - ▶ it is based on the concept that each server has one or more filesystems

amd (2)



- The **amd** map format is flexible and allows the same configuration file to be used on many machines

- Map entries can contain conditionals that activate them only in specific contexts (host or type of machine)



- Linux provides a command called **nfsstat** that can display various statistics kept by the NFS system
- `nfsstat -s` displays statistics for NFS server processes
- `nfsstat -c` shows information related to client-side-operations
- **nfsstat** may reveal problems with dropped packets, fragment reassembly, or network queue overruns that will affects your NFS performance

X-based Interfaces : Webmin (1)



Webmin Index
Module Index
Help..

Edit Export

Export details

Directory to export: /share

Active? Yes No

Export to..

Everyone Host(s) []

WebNFS clients Netgroup []

Network [192.168.0.0] Netmask [255.255.255.0]

Make symbolic links relative? Yes No **Clients must be on secure port?** Yes No

Export security

Access mode Read only Read/write **Deny access to directory?** Yes No

Trust remote users Everyone Everyone except root Nobody

Don't trust UIDs None [] **Don't trust GIDs** None []

Treat untrusted users as Default [] **Treat untrusted groups as** Default []

Save Delete

root logged into Webmin 0.990 on tower1700.home.net (Mandrake Linux 9.0)

X-based Interfaces : Webmin (2)



The screenshot shows two overlapping web browser windows from the Webmin interface. The left window, titled 'Disk and Network Filesystem', displays a table of mounted filesystems. The right window, titled 'Edit Mount', shows configuration options for a network filesystem mount.

Mounted As	Type	Location
/	New Linux Native Filesystem	IDE device A part
/dev/pts	PTS Filesystem	none
/home	New Linux Native Filesystem	IDE device A part
/mnt/WINDOWSSHARE	Windows Networking Filesystem	\\tower333\WIND
/mnt/cdrom	SUPERMOUNT	none
/mnt/cdrom2	SUPERMOUNT	none
/mnt/data	Windows 95 Filesystem	IDE device A part
/mnt/floppy	SUPERMOUNT	none
/mnt/hd	Unknown Type	IDE device B part
/mnt/nfsshare	Network Filesystem	tower1700.home.n
/mnt/winme	Windows 95 Filesystem	IDE device A part
/mnt/zip	SUPERMOUNT	none
/proc	Kernel Filesystem	proc

The 'Edit Mount' window shows the following configuration for the network filesystem mount:

- Mounted As: /mnt/nfsshare
- Size: 5036288 KB / Free: 3514056 KB
- Save Mount?: Save and mount at boot Save Don't save
- Mount now?: Mount Unmount
- NFS Hostname: tower1700.home.net
- NFS Directory: /share

Advanced Mount Options include checkboxes for Read-only, Allow device files, Disallow setuid programs, Retry mounts in background, Timeout, NFS version, Allow user interrupt, Buffer writes to filesystem, Allow execution of binaries, Allow users to mount this filesystem, Return error on timeouts, Number of Retransmissions, NFS Port, and RPC Protocol.

X-based Interfaces : Linuxconf



The screenshot shows two overlapping graphical windows from the Linuxconf interface. The top window, 'Filesystem configurator', provides a menu of actions. The bottom window, 'NFS volume', shows a table of existing NFS mounts. The 'Volume specification' window is open, showing options for configuring a new NFS volume.

The 'Filesystem configurator' window offers the following actions:

- Access local drive
- Access nfs volume
- Configure swap files and partitions
- Set quota defaults
- Check some file permissions

The 'NFS volume' window displays the following table:

Source	Mount point	FsType	Status
tower1700.home.net/share	/mnt/nfsshare	nfs	

The 'Volume specification' window shows the 'Options' tab with the following settings:

- Read only
- User mountable
- Mountable by device owner
- Not mount at boot time
- No program allowed to execute
- No special device file support
- No setuid programs allowed
- User quota enabled
- Group quota enabled

Reference



- “UNIX System Administration Handbook Third Edition” by Evi Nemeth, Garth Snyder, Scott Seebass, Trent R. Hein
- “Linux Administration Handbook” by Evi Nemeth, Garth Snyder, Trent R. Hein
- **amd** command
www.cs.columbia.edu/~ezk/am-utils

System Commands



- `mount` - mount a file system
- `umount` - unmount file systems
- `df` - report filesystem disk space usage
- `exportfs` - maintain list of NFS exported file systems
- `showmount` - show mount information for an NFS server
- `nfsstat` - print NFS statistics
- `ps` - report process status