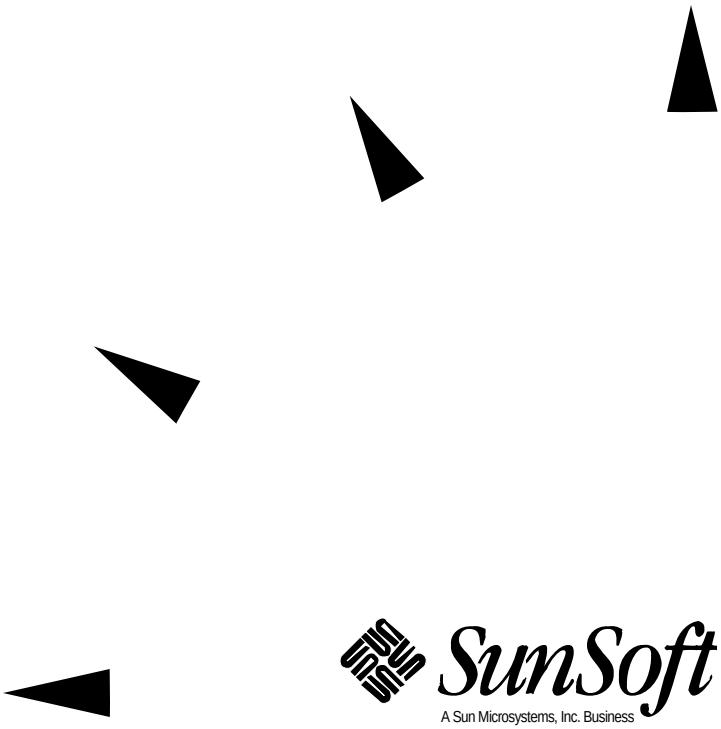
NFS Administration Guide



2550 Garcia Avenue
Mountain View, CA 94043
U.S.A.

SunSoft
A Sun Microsystems, Inc. Business

Contents

Figures

Tables

Preface

NFS Administration Guide presents the administrative tasks required for the successful operation of the SunSoft™ NFS® distributed file system. This resource-sharing product allows you to share files and directories among a number of computers on a network.

Also included in this manual is how to set up and use autofs (formerly called the automounter) to automatically mount and unmount NFS file systems.

This book is organized into explanatory background material and task-oriented instructions.

Who Should Use This Book

This book is intended for the system administrator whose responsibilities include setting up and maintaining NFS systems. Though much of the book is directed toward the experienced system administrator, it also contains information useful to novice administrators and other readers who may be new to the Solaris™ platform.

How This Book Is Organized

Chapter 1, "Solaris NFS Environment," provides an overview of the Solaris NFS environment and autofs.

Chapter 2, "NFS Administration," provides information on how to set-up NFS servers. It assumes you are using NIS or NIS+ as your name service.

Chapter 3, "Setting Up and Maintaining NFS Security," presents background information on the security features of NFS software, as well as fundamental procedures for setting up and maintaining NFS security.

Chapter 4, "NFS Troubleshooting," describes problems that may occur on machines using NFS services. It contains procedures for tracking NFS problems. Background and reference sections are also included.

Chapter 5, "Using Autofs," provides procedures for setting up and using autofs. It also includes background, reference, and troubleshooting sections.

Appendix A, "NFS Tunables," lists several parameters that can be changed to improve the NFS service. It includes instructions for making these changes.

Related Books

- *NIS+ and DNS Setup and Configuration Guide*
- *System Administration Guide, Volume I*
- *System Administration Guide, Volume II*
- *TCP/IP and Data Communications Administration Guide*

What Typographic Changes Mean

The following table describes the typographic changes used in this book.

Table P-1 Typographic Conventions

Typeface or Symbol	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% You have mail.`

Table P-1 Typographic Conventions

Typeface or Symbol	Meaning	Example
AaBbCc123	What you type, contrasted with on-screen computer output	`machine_name% su` `Password:`
AaBbCc123	Command-line placeholder: replace with a real name or value	To delete a file, type `rm` *filename*.
AaBbCc123	Book titles, new words or terms, or words to be emphasized	Read Chapter 6 in *User's Guide*. These are called *class* options. You *must* be root to do this.

Shell Prompts in Command Examples

The following table shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

Table P-2 Shell Prompts

Shell	Prompt
C shell prompt	`machine_name%`
C shell superuser prompt	`machine_name#`
Bourne shell and Korn shell prompt	`$`
Bourne shell and Korn shell superuser prompt	`#`

Solaris NFS Environment 1

This chapter provides an overview of the NFS environment. It includes a short introduction to networking, a description of the NFS service and a discussion of the concepts necessary to understand the NFS system.

An Introduction to Networking

The NFS software depends on many levels of networking software. Each network program can be associated with one or more network protocols. Each protocol simply defines how messages or data are to be exchanged. This section presents a short description of the networking fundamentals that are necessary to fully utilize the NFS product.

Protocol Layers

Some network protocols are structured as a series of layers, sometimes referred to collectively as a *protocol stack*. Each layer is designed for a specific purpose and exists on both the sending and receiving hosts. Each is designed so that a specific layer on one machine sends or receives exactly the same object sent or

received by its *peer process* on another machine. These activities take place independently from what is going on in layers above or below the layer under consideration. In other words, each layer on a host acts independently of other layers on the same machine and in concert with the same layer on other hosts.

OSI Reference Model

The Open Systems Interconnection (OSI) Reference Model is the basis of some commercially available network service architectures. Most network protocols developed independently conform loosely to the model. The Transmission Control Protocol/Internet Protocol (TCP/IP) is an example. For more information on TCP/IP, see *TCP/IP and Data Communications Administration Guide*.

The OSI Reference Model is a convenient framework for networking concepts. Basically, data are added to a network by a sender. The data is transmitted along a communication connection and is delivered to a receiver. To do this, a variety of networking hardware and software must work together.

Industry standards have been or are being defined for each layer of the reference model. Users of a service interface standard should be able to ignore the protocol and any other implementation details of the layer.

The OSI model describes network activities as having a structure of seven layers, each of which has one or more protocols associated with it. The layers represent data-transfer operations common to all types of data transfers among cooperating networks.

The protocol layers of the OSI Reference Model are traditionally listed from the top (layer 7) to the bottom (layer 1), as shown in Table 1-1.

Table 1-1 The Open Systems Interconnect Reference Model

Layer No.	Layer Name	Description
7	Application	Consists of standard communication services and applications that everyone can use.
6	Presentation	Ensures that information is delivered to the receiving machine in a form that it can understand.
5	Session	Manages the connections and terminations between cooperating computers.
4	Transport	Manages the transfer of data and assures that received and transmitted data are identical.
3	Network	Manages data addressing and delivery between networks.
2	Data Link	Handles the transfer of data across the physical network.
1	Physical	Specifies the physical characteristics of the hardware connections between hosts and networks.

Each protocol layer performs services for the layer above it. The OSI definition of the protocol layers provides designers some freedom of implementation. For example, some applications skip the presentation and session layers to interface directly with the transport layer.

The operations defined by the OSI model are purely conceptual and not unique to any particular network protocol suite. For example, the OSI network protocol suite implements all seven layers of the OSI Reference Model. TCP/IP uses some of OSI model layers and combines others. Other network protocols, such as SNA, add an eighth layer.

Network Layer

This layer, also known as the Ethernet layer, is responsible for machine-to-machine communications. It determines the path a transmission must take, based on the receiving machine's IP address. Besides message routing, it also translates from logical to physical addresses and provides error detection.

Transport Layer

The transport layer controls the flow of data on the network and assures that received and transmitted data are identical. TCP/IP or UDP may be used to enable communications between application programs running on separate machines.

TCP Protocol

TCP enables applications to communicate with each other as though connected by a physical circuit. TCP sends data in a form that appears to be transmitted in a character-by-character fashion, rather than in discrete packets. This transmission consists of a starting point, which opens the connection, the entire transmission in byte order, and an ending point, which closes the connection.

TCP makes an attempt to confirm that a packet has reached its destination by establishing an end-to-end connection between sending and receiving hosts. TCP is therefore considered a “reliable, connection-oriented” protocol.

UDP Protocol

UDP provides datagram delivery service. It does not provide any means of verifying that connection was ever achieved between receiving and sending hosts. Because UDP eliminates the processes of establishing and verifying connections, applications that send small amounts of data often use it rather than TCP. UDP is a “connectionless” protocol.

Application Layer

The application layer defines standard Internet services and network applications that anyone can use. These services work with the transport layer to send and receive data. There are many applications layer protocols, some of which you probably already use. Some of the protocols include:

- Standard TCP/IP services such as the `ftp`, `tftp`, and `telnet` commands
- UNIX “r” commands, such as `rlogin` and `rsh`
- Name services, such as NIS+ and Domain Name System (DNS)
- File services, such as the NFS system

About the NFS Environment

The NFS environment is a service that enables computers of different architectures running different operating systems to share file systems across a network. The NFS software has been implemented on many platforms ranging from the MS-DOS® to the VMS® operating systems.

The NFS environment can be implemented on different operating systems because it defines an abstract model of a file system, rather than an architectural specification. Each operating system applies the NFS model to its file system semantics. This means that file system operations like reading and writing function as though they are accessing a local file.

The benefits of NFS software are that it:

- Allows multiple computers to use the same files, so the same data can be accessed by everyone on the network
- Reduces storage costs by having computers share applications instead of needing local disk space for each user application
- Provides data consistency and reliability because all users can read the same set of files
- Makes mounting of file systems transparent to users
- Makes accessing remote files transparent to users
- Supports heterogeneous environments
- Reduces system administration overhead

The NFS system makes the physical location of the file system irrelevant to the user. You can use the NFS system to enable users to see all the relevant files, regardless of location. Instead of placing copies of commonly used files on every system, the NFS software allows you to place one copy on one computer's disk and have all other systems access it across the network. Under NFS operation, remote file systems are indistinguishable from local ones.

NFS Version 2

SunOS releases prior to Solaris 2.5 support Version 2 of the NFS protocol. The current release supports both the Version 2 and Version 3 protocols.

1

NFS Version 3

An implementation of NFS Version 3 software is a new feature of the Solaris 2.5 release. Several changes have been made to improve interoperability and to improve performance. To take full advantage of these improvements, the software should be running on the NFS servers and clients.

This version allows for safe asynchronous writes on the server, which improves performance by allowing the server to cache client write requests in memory. The client does not need to wait for the server to commit the changes to disk, so the response time is faster. Also, the server can batch the requests, which improves the response time on the server.

All operations in NFS Version 3 bring over the file attributes, which are stored in the local cache. Since the cache is updated more often, the need to do a separate operation to update this data happens less often. Specifically, the number of RPC calls to the server is reduced, improving performance.

The process for verifying file access permissions has been improved. In particular, in Version 2 a message reporting a “write error” or a “read error” would be generated if users tried to copy a remote file that they did not have permissions to. In Version 3, the permissions are checked before the file is opened, so the error is reported as an “open error.”

NFS Version 3 software removes the 8-Kbyte transfer size limit. Clients and servers will negotiate whatever transfer size they support, rather than be restricted by the 8-Kbyte limit that was imposed in Version 2. The Solaris 2.5 implementation defaults to a 32-Kbyte transfer size.

NFS ACL Support

Access Control List (ACL) support has been added in this release. The ACL software provides a more precise way to set file access permissions than is available through normal UNIX file permissions. Although performance may not be improved with this addition, access to files can be restricted further, which could tighten security.

NFS over TCP

The default transport protocol for the NFS protocol has been changed to TCP, which will help performance on slow networks. TCP provides congestion control and error recovery.

Network Lock Manager

The Solaris 2.5 release also includes an improved version of the network lock manager, which provides UNIX record locking and PC file sharing for NFS files. The locking mechanism is now more reliable for NFS files, and so, commands like `ksh` and `mail`, which use locking, are less likely to hang.

NFS Servers and Clients

The terms *client* and *server* are used to describe the roles that a computer plays when sharing file systems. If a file system resides on a computer's disk and that computer makes the file system available to other computers on the network, then that computer acts as a server. The computers that are accessing that file system are said to be clients. NFS software enables any given computer to access any other computer's file systems and, at the same time, to provide access to its own file systems. A computer may play the role of client, server, or both at any given time on a network.

A server can provide files to a diskless client, a computer that has no local disk. A diskless client relies completely on the server for all its file storage. A diskless client can act only as a client—never as a server.

Clients access files on the server by mounting the server's shared file systems. When a client mounts a remote file system, it does not make a copy of the file system; rather, the mounting process uses a series of remote procedure calls that enable the client to access the file system transparently on the server's disk. The mount looks like a local mount and users type commands as if the file systems were local.

Once a file system has been shared on a server through NFS operation, it can be accessed from a client. NFS file systems may be mounted automatically with autofs.

1

NFS File Systems

The objects that can be shared through the NFS software include any whole or partial directory tree or a file hierarchy—including a single file. A computer cannot share a file hierarchy that overlaps one that is already shared. Peripheral devices such as modems and printers cannot be shared.

In most UNIX system environments, a file hierarchy that can be shared corresponds to a file system or to a portion of a file system; however, NFS software works across operating systems, and the concept of a file system may be meaningless in other, non-UNIX environments. Therefore, the term *file system* is used throughout this guide to refer to a file or file hierarchy that can be shared and mounted over the NFS environment.

Autofs

File systems shared through NFS software can be mounted using automatic mounting. Autofs, a client-side service, is a file system structure that provides advanced automatic mounting. The file system structure is created by `automount`. This program notifies the automount daemon, `automountd`, when mounting or unmounting needs to be done. The daemon runs in the background, mounting and unmounting remote directories on an as-needed basis.

Whenever a user on a client computer running `automountd` tries to access a remote file or directory, the daemon mounts the file system to which that file or directory belongs. This remote file system remains mounted for as long as it is needed. If the remote file system is not accessed for a certain period of time, it is automatically unmounted.

No mounting is done at boot time, and the user no longer has to know the superuser password to mount a directory; users need not use the `mount` and `umount` commands. The autofs service mounts and unmounts file systems as required without any intervention on the part of the user.

Mounting some file hierarchies with `automountd` does not exclude the possibility of mounting others with `mount`. A diskless computer *must* mount `/` (root), `/usr`, and `/usr/kvm` through the `mount` command and the `/etc/vfstab` file.

More specific information about the autofs service is given in Chapter 5, “Using Autofs.”

Autofs Features

Autofs works with file systems specified in the local name space. This information can be maintained in NIS, NIS+, or local files.

The name space data can specify several remote locations for a particular file. This way, if one of the servers is down, the autofs service can try to mount from another computer. To specify which servers are preferred for each file system in the maps, each server can be assigned a weighting factor.

1

NFS Administration 2

This chapter provides an introduction to the NFS commands. This chapter also provides information on how to perform such NFS administration tasks as setting up NFS servers, adding new file systems to share, unsharing file systems, displaying shared local file systems, and displaying mounted file systems. It assumes you are using NIS or NIS+ as your name service.

Your responsibilities as an NFS administrator depend on your site's requirements and the role of your computer on the network. You may be responsible for all the computers on your local network, in which case you may be responsible for the major tasks involved in NFS administration:

- Determining which computers, if any, should be dedicated servers
- Which should act as both servers and clients
- Which should be clients only

2

Maintaining a server once it has been set up involves the following tasks:

- Sharing and unsharing file systems as necessary
- Modifying administrative files to update the lists of file systems your computer shares or mounts automatically
- Checking the status of the network
- Diagnosing and fixing NFS-related problems as they arise – see to Chapter 4, “NFS Troubleshooting”
- Setting up maps for autofs – see Chapter 5, “Using Autofs”

Remember, a computer can be both a server and a client—sharing local file systems with remote computers and mounting remote file systems.

NFS Files

Several ASCII files are needed to support NFS activities on any computer. Table 2-1 lists these files and their function.

Table 2-1 NFS ASCII files

File Name	Function
`/etc/vfstab`	Defines file systems to be mounted locally (see the `vfstab(4)` man page)
`/etc/mnttab`	Lists file systems that are currently mounted including automounted directories (see the `mnttab(4)` man page); do not edit this file
`/etc/rmtab`	Lists file systems remotely mounted by NFS clients (see the `rmtab(4)` man page); do not edit this file
`/etc/default/fs`	Lists the default file system type for local file systems
`/etc/dfs/dfstab`	Lists the local resources to be shared
`/etc/dfs/fstypes`	Lists the default file system types for remote file systems
`/etc/dfs/sharetab`	Lists the resources (local and remote) that are shared (see the `sharetab(4)` man page); do not edit this file

The first entry in `/etc/dfs/fstypes` is often used as the default file system type for remote file systems. This entry defines the NFS file system type as the default.

There is only one entry in `/etc/default/fs`: the default file system type for local disks. The file system types that are supported on a client or server can be determined by checking the files in `/kernel/fs`.

NFS Daemons

To support NFS activities, several daemons are started when a system goes into run level 3 or multiuser mode. Two of these daemons (`mountd` and `nfsd`) are run on systems that are NFS servers. The automatic startup of the server daemons depends on the existence of entries labeled with the NFS file system type in `/etc/dfs/sharetab`.

The other two daemons (`lockd` and `statd`) are run on NFS clients to support NFS file locking. These daemons must also run on the NFS servers.

`lockd`

This daemon supports record locking operations on NFS files. It will send locking requests from the client to the NFS server. On the NFS server, it will start local locking. The daemon is normally started without any options. The command syntax is:

`lockd` [`-g` *graceperiod*] [`-t` *timeout*]

where *graceperiod* selects the number of seconds that the clients have to reclaim locks after the server reboots, and *timeout* selects the number of seconds to wait before retransmitting a lock request to the remote server. The default value for *graceperiod* is 45 seconds. Reducing this value means that NFS clients can resume operation more quickly after a server reboot, but it increases the chances that a client might not be able to recover all of its locks. The default value for *timeout* is 15 seconds. Decreasing the *timeout* value can improve response time for NFS clients on a noisy network, but it can cause additional server load by increasing the frequency of lock requests.

`mountd`

This is an RPC server that handles file system mount requests from remote systems. It checks `/etc/dfs/sharetab` to determine which file systems are available for remote mounting and which systems are allowed to do the remote mounting. There are no options to select with this daemon.

`nfsd`

This daemon handles other client file system requests. The command syntax is:

`nfsd` [`-a`] [`-p` *protocol*] [`-t` *device*] [`-c` *#_conn*] [*nservers*]

where `-a` indicates to start a daemon over all available transports, *protocol* selects a specific protocol to run the daemon over, *device* chooses a specific transport for the daemon, `-c` *#_conn* selects the maximum number of connections per connection-oriented transport, and *nservers* is the maximum number of concurrent requests that a server can handle. The default value for *#_conn* is unlimited. The default value for *nservers* is 1, but the startup scripts select 16.

Unlike older versions of this daemon, `nfsd` does not spawn multiple copies to handle concurrent requests. Checking the process table with `ps` will only show one copy of the daemon running.

`statd`

This daemon works with `lockd` to provide crash and recovery functions for the lock manager. It keeps track of the clients that hold locks on a NFS server. If a server crashes, upon rebooting `statd` on the server will contact `statd` on the client. The client `statd` can then attempt to reclaim any locks on the server. The client `statd` will also inform the server `statd` when a client has crashed, so that the client's locks on the server can be cleared. There are no options to select with this daemon.

NFS Commands

These commands must be run as root to be fully effective, but requests for information can be made by all users:

- `mount` – See page 15.
- `mountall` – See page 21.
- `setmnt` – See page 28.
- `share` – See page 22.
- `shareall` – See page 26.
- `showmount` – See page 27.
- `umount` – See page 20.
- `umountall` – See page 21.
- `unshare` – See page 26.

- `unshareall` – See page 27.

mount

With this command, you can attach a named file system, either local or remote, to a specified mount point. For more information, see the `mount(1M)` man page.

Used without arguments, `mount` displays a list of file systems that are currently mounted on your computer. The syntax is:

`mount [ -F` *FSType* `] [` *generic_options* `] [ -o` *specific_options* `]` *resource mount_point*

where -F *FSType* specifies the file system type to be accessed, *generic_options* selects options that are not specific to the type of file system that regulates how the file system will be accessed, `-o` *specific_options* indicates access options that are specific to each file system type, *resource* is the name of the file system to be accessed, and *mount_point* is the place on the local file system to which the file system will be attached.

Many types of file systems are included in the standard Solaris installation. For a complete description of all of the file system types, see *System Administration Guide, Volume I*. Some available options are listed in Table 2-2.

Table 2-2 FStype Options for `mount`

FSType Selection	File System Type Description
`cachefs`	Cache file system using local disks to store information from a remote system for faster access and less net traffic
`hsfs`	High Sierra file system used on CD-ROMs
`nfs`	Default SunOS distributed file system
`pcfs`	PC file system created on a DOS system
`s5fs`	System V file system used by PC versions of UNIX
`tmpfs`	Temporary file system using local memory for the file system
`ufs`	UNIX file system is the SunOS default for local disks

The `mount` command includes options that are file system specific and some that are not. The options that are not file system specific are sometimes called the generic options. These options are shown in Table 2-3.

Table 2-3 Generic Options for `mount`

Generic Option	Description
`-m`	Mounts without creating an entry in `/etc/mnttab`
`-r`	Mounts read-only
`-o`	Includes FSType-specific options in a comma separated list
`-O`	Overlays mount on top of file system already mounted

The list of options that may be included with the -o flag are different for each file system type. Each file system type has a specific mount man page which lists these options. The man page for NFS file systems is `mount_nfs(1M)`, for UFS file systems it is `mount_ufs(1M)`, and so forth.

`mount` *Options for NFS File Systems*

Listed below are some of the most commonly used options that can follow the `–o` flag when mounting an NFS file system.

`bg|fg`

If the first mount attempt fails, retry in the background (`bg`) or foreground (`fg`). The default is `fg`, which is the best selection for file systems that must be available. It prevents further processing until the mount is complete. `bg` is a good selection for file systems that are not critical, because the client will do other processing while waiting for the mount request to complete.

`intr|nointr`

Allow (`intr`) or disallow (`nointr`) keyboard interruption to kill a process that is hung while waiting for a hard-mounted file system. Default is `intr`. The `intr` option makes it possible for clients to interrupt applications which may be waiting for a remote mount.

`noac`

Suppress attribute caching, so attributes are always up-to-date. This allows for the mail delivery and mail user agents to lock mail files properly.

`proto=`*netid*

Select the transport protocol to be used. The value for *netid* must be listed in `/etc/netconfig`. Possible values are `udp` or `tcp`. TCP will be selected by default.

`quota|noquota`

Execution of `quota` is enabled or disabled (see the `quota(1M)` man page). If the file system is on a remote server that has `quota` enabled, `quota` will be run regardless of how it is mounted locally. Running `quota` is the default.

`rsize=#`

Select the read buffer size in bytes. The default is 8 Kbytes for Version 2 and 32 Kbytes for Version 3.

`wsize=#`

Select the write buffer siz in bytes. The default is 8 Kbytes for Version 2, and 32 Kbytes for Version 3.

Warning – Third party Ethernet cards, can not always support the 8-Kbyte buffer size so it may be necessary to select a proper `rsize` and `wsize` value.

`rw|ro`

The file system is to be mounted read-write or read-only. The default is read-write, which is the appropriate option for remote home directories, mail-spooling directories or other file systems that will be need to be changed by the users. The read-only option is appropriate for directories that should not be changed by the users, for example, shared copies of the man pages should not be writable by the users.

`soft|hard`

An NFS file system mounted with the `soft` option will return an error if the server does not respond. The `hard` option will cause the mount to continue to retry until the server responds. The default is `hard` which should be used for most file systems. Applications frequently do not check return values from `soft` mounted file systems, which can make the application fail or can lead to corrupted files. Even if the application does check, routing problems and other conditions can still confuse the application or lead to file

corruption if the soft option is used. In most cases the `soft` option should not be used. If a file system is unavailable, an application using this file system may hang if the `hard` option is selected until the file system becomes available.

`suid|nosuid`

`setuid` execution on the file system is allowed or disallowed. Allowing suid execution is the default. This is a potential security problem.

`vers=#`

Select the version of the NFS protocol to be used (either 2 or 3). Version 3 will automatically be selected if it is supported by the client and the server. This option can be used to force a specific version.

Additional `mount` *Options*

The mount command includes some options that do not require a file system type to be specified in the command line. These are described below.

`-a [` *mount_points* `]`

Attempts to mount the selected *mount_points* in parallel if possible. If *mount_points* are not given then all entries in `/etc/vfstab` that should be mounted at boot are tried. If the `-F` *nfs* option is included then all file system that are listed as `nfs` type in `/etc/vfstab` are mounted. Selecting `-F` *ufs* mounts only the local file systems.

`-p`

Prints a list of the mounted file systems in a format appropriate for the `/etc/vfstab` file. This is useful for adding entries to the file without mistakes. This option cannot be used with others.

`-v`

Displays a more verbose listing of the currently mounted file systems. The information is much like that displayed when no options are used, but it includes the type of file system. This option cannot be used with others.

`-V`

Echoes the command line but will not execute it. This command can be useful when verifying configurations and needed command lines.

Using the `mount` *Command*

Both of these commands will mount an NFS file system from the server `bee` read-only:

```
# mount -F nfs -r bee:/export/share/man /usr/man
```

```
# mount -F nfs -o -ro bee:/export/share/man /usr/man
```

This command will force the man pages from the server `bee` to be mounted on the local system even if `/usr/man` has already been mounted on:

```
# mount -F nfs -O bee:/export/share/man /usr/man
```

Use the `mount` command with no arguments to display file systems mounted on a client.

```
% mount
/ on /dev/dsk/c0t3d0s0 read/write/setuid on Tues Jan 24 13:20:47 1995
/usr on /dev/dsk/c0t3d0s6 read/write/setuid on Tues Jan 24 13:20:47 1995
/proc on /proc read/write/setuid on Tues Jan 24 13:20:47 1995
/dev/fd on fd read/write/setuid on Tues Jan 24 13:20:47 1995
/tmp on swap read/write on Tues Jan 24 13:20:51 1995
/opt on /dev/dsk/c0t3d0s5 setuid/read/write on Tues Jan 24 13:20:51 1995
/home/kathys on bee:/export/home/bee7/kathys
   intr/noquota/nosuid/remote on Tues Jan 24 13:22:13 1995
```

umount

This command allows you to remove a remote file system that is currently mounted. The syntax of the command is:

`umount [ -o` *specific_options* `]` *resource* `|` *mount_point*

where `-o` *specific_options* indicates options that are specific to each file system type, *resource* is the name of the file system to be disconnected, and *mount_point* is the place on the local file system to which the file system is attached.

Like `mount`, `umount` supports the `-o` and `-V` options to allow for extra options and for testing. The extra options selected with the `-o` option should not be necessary to uniquely identify the file system if entries are kept current in `/etc/mnttab`. The `-a` option may also be used, but it behaves differently. If *mount_points* are included with the `-a` option, then those file systems are unmounted. If no mount points are included, then an attempt is made to unmount all file systems listed in `/etc/mnttab`, except for the "required" file systems, such as `/`, `/usr`, `/var`, `/proc`, `/dev/fd`, and `/tmp`.

Since the file system is already mounted and should have any entry in `/etc/mnttab` there is no need to include a flag for the file system type.

The command will not succeed if the file system is in use. For instance, if a user has used `cd` to get access to a file system, the file system will be busy until they change their working directory. The `umount` command may hang temporarily if the NFS server is unreachable.

Using the `umount` *Command*

This example unmounts a file system mounted on `/usr/man`:

```
# umount /usr/man
```

This example displays the results of running `umount -a -V`:

```
# umount -a -V
umount /home/kathys
umount /opt
umount /home
umount /net
```

mountall

Use this command to mount all file systems specified in a file system table. The syntax of the command is:

`mountall [ -F` *FSType* `] [ -l | -r ] [` *file_system_table* `]`

where `-F` *FSType* specifies the file system type to be accessed, `-l` indicates that only local file systems are to be used, `-r` indicates that only remote file systems are to be used, and *file_system_table* selects an alternate to `/etc/vfstab`.

Using the `mountall` *Command*

These two examples are equivalent:

```
# mountall -F nfs
```

```
# mountall -F nfs -r
```

umountall

Use this command to unmount a group of file systems. The syntax of the command is:

`umountall [ -k ] [ -s ] [ -F` *FSType* `] [ -l | -r ]`

`umountall [ -k ] [ -s ] [ -h` *host* `]`

where `-k` indicates that the `fuser -k` *mount_point* command should be used to kill any processes associated with the *mount_point*, `-s` indicates that unmount is not to be performed in parallel, `-F` *FSType* specifies the file system type to be

unmounted, `-l` specifies that only local file systems are to be used, and `-r` specifies that only remote file systems are to be used. The `-h` *host* option indicates that all file systems from the named host should be unmounted. The `-h` option can not be combined with `-F`, `-l` or `-r`.

Using the `umountall` *Command*

This command unmounts all file systems that are mounted from remote hosts:

```
# umountall -r
```

This command unmounts all file systems currently mounted from the server `bee`:

```
# umountall -h bee
```

`share`

With this command, you can make a local file system on an NFS server available for mounting. You can also use the `share` command to display a list of the file systems on your system that are currently shared. The command has this syntax:

`share [ -F` *FSType* `] [ -o` *specific_options* `][ -d` *description* `]` *pathname*

where `-F` *FSType* indicates the type of file system that is to be shared, `-o` *specific_options* is a comma-separated list of options that regulates how the file system is shared, *description* is a comment that describes the file system to be shared, and *pathname* is the full name of the file system to be shared, starting at root (`/`). If the `-F` option is not used, then `/etc/dfs/fstypes` is checked to determine the default file system type (normally this is set to NFS).

The NFS server must be running for the `share` command to work. The NFS server software is started automatically during boot if there is an entry in `/etc/dfs/dfstab`. The command will not report an error if the NFS server software is not running, so you must check this yourself.

The objects that can be shared include any directory tree, but each file system hierarchy is limited by the disk slice or partition that the file system is located on. For instance, sharing the root (/) file system would not also share `/usr`, unless they are on the same disk partition or slice. Normal installation places root on slice 0 and `/usr` on slice 6. Also, sharing `/usr` would not share any other local disk partitions that are mounted on subdirectories of `/usr`.

A file system can not be shared that is part of a larger file system that is already shared. For example, if `/usr` and `/usr/local` are on one disk slice, then `/usr` can be shared or `/usr/local` can be shared, but if both need to be shared with different share options then `/usr/local` will need to be moved to a separate disk slice.

Warning – It is possible to gain access to a file system which is shared read-only through the file handle of a file system that is shared read-write if the two file systems are on the same disk slice. It is more secure to place those file systems that need to be read-write on a separate partition or disk slice than the file systems that you need to share read-only.

`share` *Options*

The options that can be included with the `-o` flag are:

`rw|ro`

The *pathname* file system is shared read-write or read-only to all clients

`rw=client[:client]...`

The file system is shared read-write to the listed clients only. All other requests are denied. Netgroup names can be used instead of client names in most cases. Netgroup names may not be used in `rw=list` if there is also a `ro=list` in the option string. This same type of entry can be used with the `ro` option.

`share` *Options for NFS File Systems*

The options that can be used with NFS file systems only include:

`anon=`*uid*

where *uid* is used to select the user ID of unauthenticated users. If *uid* is set to -1, access is denied to unauthenticated users. Authentication is explained further in Chapter 3, “Setting Up and Maintaining NFS Security.” Root access can be granted by setting `anon=0`, but this will allow unauthenticated users to have root access, so use the `root` option instead.

`root=host[:host]....`

Root access is given to the hosts in the list. By default, no remote host is given root access. Netgroup names can not be used with the `root` option.

Caution – Granting root access to other hosts has far-reaching security implications; use the `root=` option with extreme caution.

`nosuid`

This option signals that all attempts to enable the `setuid` or `setgid` mode should be ignored.

`aclok`

This option allows the NFS server to do access control for NFS Version 2 clients (running SunOS 2.4 or earlier releases). Without this option minimal access is given to all clients. With this option maximal access is given to the clients. For instance, with `aclok` set on the server, if anyone has read permissions, then everyone does. See *System Administration Guide, Volume I* for more information about ACLs.

Note – To take advantage of ACLs, it is best to have clients and servers run software that supports the NFS Version 3 and NFS_ACL protocols. If the software only supports the NFS Version 3 protocol, then clients will get correct access, but will not be able to manipulate the ACLs. If the software supports the NFS_ACL protocol, then the client will get correct access and the ability to manipulate the ACLs. The Solaris 2.5 release supports both protocols.

Using the `share` *Command*

To list the file systems that are shared on the local system, use the share command without options. The `-F` *FSType* option used alone lists all of the file systems of that specific type that are being shared.

```
# share
-       /export/share/man   ro   ""
-       /usr/src   rw=eng   ""
```

The command below provides read-only access for most systems but allows read-write access for `rose` and `lilac`:

```
# share -F nfs -o ro,rw=rose:lilac /usr/src
```

In the next example, read-only access is assigned to any host in the `eng` netgroup. The client `rose` is specifically given read-write access.

```
# share -F nfs -o ro=eng,rw=rose /usr/src
```

If both the `ro=list` and `rw=list` options are used, only the `ro` entry may include a netgroup name. In the previous example, `rw` is used to add write access for only one host. For more information about netgroups, see *System Administration Guide, Volume I.*

Note – You cannot specify both `rw` and `ro` without arguments, and you cannot specify the same client in the `rw=list` and the `ro=list`. If no read-write option is specified, the default is read-write for all clients.

To share one file system with multiple clients, all options must be entered on the same line, since multiple invocations of the `share` command on the same object will “remember” only the last command run. This command will allow read-write access to three client systems, but only `rose` and `tulip` are given access to the file system as root.

```
# share -F nfs -o rw=rose:lilac:tulip,root=rose:tulip /usr/src
```

unshare

This command allows you to make a previously available file system unavailable for mounting by clients. The syntax of the command is:

`unshare [ -F` *FSType* `] [ -o` *specific_options* `] [` *pathname* `]`

where `-F` *FSType* indicates the type of file system that is to be made unavailable, `-o` *specific-options* is a comma-separated list of options specific to the file system, and *pathname* is the full name of the file system.

The `unshare` command can be used to unshare any file system—whether the file system was shared explicitly with the `share` command or automatically through `/etc/dfs/dfstab`. If you use the `unshare` command to unshare a file system that you shared through the `dfstab` file, remember that it will be shared again when you exit and re-enter run level 3. The entry for this file system must be removed from the `dfstab` file if the change is to continue.

When you unshare an NFS file system, access from clients with existing mounts is inhibited. The file system may still be mounted on the client, but the files will not be accessible.

Using the `unshare` *Command*

This command unshares a specific file system:

```
# unshare /usr/src
```

shareall

This command allows for multiple file systems to be shared. When used with no options, the command will share all entries in `/etc/dfs/dfstab`. The syntax of the command is:

`shareall [ -F` *FSType*`[,`*FSType*`...]] [` *file* `]`

where `-F` FSType is a list of file system types defined in `/etc/dfs/fstypes` and *file* is a name of a file that lists share command lines. If *file* is not included `/etc/dfs/dfstab` is checked. If a "`-`" is used for file, then share commands may be entered from standard input.

Using the shareall *Command*

This command shares all file systems listed in a local file:

```
# shareall /etc/dfs/special_dfstab
```

unshareall

This command will make all currently shared resources unavailable. This is the command syntax:

unshareall [-F *FSType*[,*FSType*...]

where -F *FSType* is a list of file system types defined in /etc/dfs/fstypes. This flag allows you to choose only certain types of file systems to be unshared. The default file system type is defined in /etc/dfs/fstypes. To choose specific file systems, use the unshare command.

Using the unshareall *Command*

This example should unshare all NFS type file systems:

```
# unshareall -F nfs
```

showmount

This command displays all the clients that have remotely mounted file systems that are shared from an NFS server, or only the file systems that are mounted by clients, or the shared file systems with the client access information. The command syntax is:

showmount [-ade] [*hostname*]

where -a prints a list all of the remote mounts (each entry includes the client name and the directory), -d prints a list of the directories that are remotely mounted by clients, -e prints a list of the files shared (or exported), and *hostname* selects the NFS server to gather the information from. If *hostname* is not specified then the local host is queried.

Using the showmount *Command*

This command lists all clients and the directory that they have mounted.

```
# showmount -a bee
lilac:/export/share/man
lilac:/usr/src
rose:/usr/src
tulip:/export/share/man
```

This command lists the directories that have been mounted.

```
# showmount -d bee
/export/share/man
/usr/src
```

This command lists file systems that have been shared.

```
# showmount -e bee
/usr/src           (everyone)
/export/share/man  eng
```

setmnt

This command creates an /etc/mnttab table. The table is consulted by the mount and umount commands. Generally, there is no reason to run this command by hand; it is run automatically when a system is booted.

Other Useful Commands

These commands can be useful when troubleshooting NFS problems.

nfsstat

This command can be used to gather statistical information about NFS and RPC connections. The syntax of the command is:

```
nfsstat [ -cmnrsz ]
```

where `-c` displays client side information, `-m` displays statistics for each NFS mounted file system, `-n` specifies that NFS information is to be displayed (both client and server side), `-r` displays RPC statistics, `-s` displays the server side information, and `-z` specifies that the statistics should be set to zero. If no options are supplied on the command line, the `-cnrs` options are used.

Gathering server side statistics can be very important for debugging problems when new software or hardware are added to the computing environment. Running this command, at least once a week, and storing the numbers will provide a good history of previous performance.

Using the `nfsstat` *Command*

```
# nfsstat -s

Server rpc:
Connection oriented:
calls     badcalls   nullrecv   badlen     xdrcall    dupchecks  dupreqs
11420263   0          0          0          0          1428274    19
Connectionless:
calls     badcalls   nullrecv   badlen     xdrcall    dupchecks  dupreqs
14569706   0          0          0          0          953332     1601

Server nfs:
calls      badcalls
24234967   226
Version 2: (13073528 calls)
null       getattr    setattr    root       lookup     readlink   read
138612 1%  1192059 9% 45676 0%   0 0%      9300029 71% 9872 0%   1319897 10%
wrcache    write      create     remove     rename     link       symlink
0 0%       805444 6%  43417 0%   44951 0%   3831 0%    4758 0%    1490 0%
mkdir      rmdir      readdir    statfs
2235 0%    1518 0%    51897 0%   107842 0%
Version 3: (11114810 calls)
null       getattr    setattr    lookup     access     readlink   read
141059 1%  3911728 35% 181185 1% 3395029 30% 1097018 9% 4777 0%   960503 8%
write      create     mkdir      symlink    mknod      remove     rmdir
763996 6%  159257 1%  3997 0%    10532 0%   26 0%      164698 1%  2251 0%
rename     link       readdir    readdir+   fsstat     fsinfo     pathconf
53303 0%   9500 0%    62022 0%   79512 0%   3442 0%    34275 0%   3023 0%
commit
73677 0%

Server nfs_acl:
Version 2: (1579 calls)
null       getacl     setacl     getattr    access
0 0%       3 0%       0 0%       1000 63%   576 36%
Version 3: (45318 calls)
null       getacl     setacl
0 0%       45318 100% 0 0%
```

This is an example of NFS server statistics. The first five lines deal with RPC and the rest of them report NFS activities. In both sets of statistics knowing the

average number of badcalls / calls and the number of calls / week, can help identify when something is going wrong. The badcalls value reports the number of bad messages from a client and can point out network hardware problems.

Some of the connections generate write activity on the disks. A sudden increase in these statistics could indicate trouble and should be investigated. For NFS Version 2 statistics, the connections to pay special attention to are: setattr, write, create, remove, rename, link, symlink, mkdir, and rmdir. For NFS Version 3 statistics the value to watch is commit. If the commit level is high in one NFS server as compared to another almost identical one, check to make sure that the NFS clients have enough memory. The number of commit operations on the server go up when clients do not have resources available.

pstack

This command displays a stack trace for each process. It must be run by `root`. It can be used to determine where a process is hung, The only option allowed with this command is the `PID` of the process that you want to check (see the `proc(1)` man page).

Using the `pstack` *Command*

The example below is checking the `nfsd` process that is running.

```
# /usr/proc/bin/pstack 243
243:    /usr/lib/nfs/nfsd -a 16
 ef675c04 poll     (24d50, 2, ffffffff)
 000115dc ???????? (24000, 132c4, 276d8, 1329c, 276d8, 0)
 00011390 main     (3, effff14, 0, 0, ffffffff, 400) + 3c8
 00010fb0 _start   (0, 0, 0, 0, 0, 0) + 5c
```

It shows that the process is waiting for a request. This is a normal response for a system that is not handling much NFS activity. If the stack shows that the process is still in poll after a request is made, it is possible that the process is hung. Please follow the instructions in “How to Restart NFS Services” on page 57 to fix this problem. Review the instructions in “NFS Troubleshooting Procedures” on page 53 to fully verify that your problem is a hung program.

rpcinfo

This command generates information about the RPC service running on a system. It can also be used to change the RPC service. There are many options available with this command (see the `rpcinfo(1M)` man page). This is a shortened synopsis for some of the options that can be used with the command:

`rpcinfo [ -m | -s ] [` *hostname* `]`

`rpcinfo [ -t | -u ] [` *hostname* `] [` *progname* `]`

where `-m` displays a table of statistics of the `rpcbind` operations, `-s` displays a concise list of all registered RPC programs, `-t` displays the RPC programs that use TCP, `-u` displays the RPC programs that use UDP, *hostname* selects the hostname of the server you need information from, and *progname* selects the RPC program to gather information about. If no value is given for *hostname*, then the local hostname is used. The RPC program number can be substituted for *progname*, but many will remember the name and not the number. The `-p` option can be used in place of the `-s` option on those systems which do not run the NFS Version 3 software.

The data generated by this command can include:

- the RPC program number
- the version number for a specific program
- the transport protocol that is being used
- the name of the RPC service
- the owner of the RPC service

Using the `rpcinfo` *Command*

This example gathers information on the RPC services running on a server. The text generated by the command is filtered by the sort command to make it more readable. Several lines listing RPC services have been deleted from the example.

```
% rpcinfo -s bee |sort -n
   program version(s) netid(s)                          service     owner
    100000  2,3,4     udp,tcp,ticlts,ticotsord,ticots  portmapper  superuser
    100001  4,3,2     ticlts,udp                       rstatd      superuser
    100002  3,2       ticots,ticotsord,tcp,ticlts,udp  rusersd     superuser
    100003  3,2       tcp,udp                          nfs         superuser
    100005  3,2,1     ticots,ticotsord,tcp,ticlts,udp  mountd      superuser
    100008  1         ticlts,udp                       walld       superuser
    100011  1         ticlts,udp                       rquotad     superuser
    100012  1         ticlts,udp                       sprayd      superuser
    100021  4,3,2,1   ticots,ticotsord,ticlts,tcp,udp  nlockmgr    superuser
    100024  1         ticots,ticotsord,ticlts,tcp,udp  status      superuser
    100026  1         ticots,ticotsord,ticlts,tcp,udp  bootparam   superuser
    100029  2,1       ticots,ticotsord,ticlts          keyserv     superuser
    100068  4,3,2     tcp,udp                          cmsd        superuser
    100078  4         ticots,ticotsord,ticlts          kerbd       superuser
    100083  1         tcp,udp                          -           superuser
    100087  11        udp                              adm_agent   superuser
    100088  1         udp,tcp                          -           superuser
    100089  1         tcp                              -           superuser
    100099  1         ticots,ticotsord,ticlts          pld         superuser
    100101  10        tcp,udp                          event       superuser
    100104  10        udp                              sync        superuser
    100105  10        udp                              diskinfo    superuser
    100107  10        udp                              hostperf    superuser
    100109  10        udp                              activity    superuser
    .
    .
    100227  3,2       tcp,udp                          -           superuser
    100301  1         ticlts                           niscachemgr superuser
    390100  3         udp                              -           superuser
1342177279  1,2       tcp                              -           14072
```

This example shows how to gather information about a particular RPC service using a particular transport on a server.

```
% rpcinfo -t bee mountd
program 100005 version 1 ready and waiting
program 100005 version 2 ready and waiting
program 100005 version 3 ready and waiting
% rpcinfo -u bee nfs
program 100003 version 2 ready and waiting
program 100003 version 3 ready and waiting
```

The first example checks the mountd service running over TCP. The second example checks the NFS service running over UDP.

snoop

This command is often used to watch for packets on the network. It must be run as `root`. It is a good way to make sure that the network hardware is functioning on both the client and the server. There are many options available (see the `snoop(1M)` man page). A shortened synopsis of the command is given below:

`snoop [ -d` *device* `] [ -o` *filename* `] [ host` *hostname*]

where `-d` *device* specifies the local network interface, `-o` *filename* will store all of the captured packets into the named file, and *hostname* indicates to only display packets going to and from a specific host.

The `-d` *device* option is very useful on those servers which have multiple network interfaces. There are many other expressions that can be used besides setting the host. A combination of command expressions with `grep` can often generate data that is specific enough to be useful.

When troubleshooting, make sure that packets are going to and from the host that you expect them too. Also, look for error messages. Saving the packets to a file can make it much easier to review the data.

truss

This command can be used to see if a process is hung. It must be run by `root`. There are many options that can be used with this command (see the `truss(1)` man page). A shortened syntax of the command is:

`truss [ -t` *syscall* `] -p` *pid*

where `-t` *syscall* selects system calls to trace, and `-p` *pid* indicates the PID of the process to be traced. The *syscall* may be a comma-separated list of system calls to be traced. Also, starting *syscall* with a ! selects to exclude the system calls from the trace.

Using the `truss` *Command*

The example below shows that the process is waiting for another request for service.

```
# /usr/bin/truss -p 243
poll(0x00024D50, 2, -1)         (sleeping...)
```

This is a normal response for a system that is not handling much NFS activity. If the response does not change after a request has been made, it is possible that the process is hung. Please follow the instructions in “How to Restart NFS Services” on page 57 to fix the hung program. Review the instructions in “NFS Troubleshooting Procedures” on page 53 to fully verify that your problem is a hung program.

Automatic File System Sharing

Servers provide access to their file systems by sharing them over the NFS environment. You specify which file systems are to be shared with the `share` command and/or the `/etc/dfs/dfstab` file.

Entries in the `/etc/dfs/dfstab` file are shared automatically whenever you start NFS server operation. You should set up automatic sharing if you need to share the same set of file systems on a regular basis. For example, if your computer is a server that supports diskless clients, you need to make your clients’ root directories available at all times.

The `dfstab` file lists all the file systems that your server shares with its clients and controls which clients may mount a file system. If you want to modify `dfstab` to add or delete a file system or to modify the way sharing is done, simply edit the file with any supported text editor (such as `vi`). The next time the computer enters run level 3, the system reads the updated `dfstab` to determine which file systems should be shared automatically.

Each line in the `dfstab` file consists of a `share` command—the same command you would enter at the command line prompt to share the file system. The `share` command is located in `/usr/sbin`.

1. **Edit the `/etc/dfs/dfstab` file.**
   Add one entry to the file for each file system that you want to have shared automatically. Each entry must be on a line by itself in the file and uses this syntax:

```
share [-F nfs] [-o specific-options] [-d description] pathname
```

2. **Make sure that the NFS software is running on the server.**
   If this is the first `share` command or set of `share` commands that you have initiated, it is likely that the NFS daemons are not running. The following commands kill the daemons and restart them.

```
# /etc/init.d/nfs.server stop
# /etc/init.d/nfs.server start
```

This ensures that NFS software is now running on the servers and will restart automatically when the server is at run level 3 during boot.

At this point, set up your autofs maps so clients can access the file systems you've shared on the server. See "Setting Up Autofs Maps" on page 65.

Mounting at Boot Time

If you want to mount file systems at boot time instead of using autofs maps, follow this procedure. Although this procedure must be followed for all local file systems, it is not recommended for remote file systems, because it must be completed on every client.

♦ **Edit the `/etc/vfstab` file.**

Entries in the `/etc/vfstab` file have the following syntax:

```
special  fsckdev  mountp  fstype  fsckpass  mount-at-boot  mntopts
```

Example of a `vfstab` *entry*

You want a client computer to mount the `/var/mail` directory on the server `wasp`. You would like it mounted as `/var/mail` on the client. You want the client to have read-write access. Add the following entry to the client's `vfstab` file.

```
wasp:/var/mail - /var/mail nfs - yes rw
```

Warning – NFS Servers should not have NFS `vfstab` entries because of a potential dead-lock. The NFS server software is started after the entries in `/etc/vfstab` are checked, so that if you have two servers go down at the same time which are mounting file systems from the other, each system could hang as the systems reboot.

Mounting on the Fly

To manually mount a file system during normal operation, run the `mount` command as superuser:

```
# mount -F nfs -r -o bee:/export/share/stuff /mnt
```

In this case the `/export/share/stuff` file system from the server `bee` is mounted on read-only `/mnt` on the local system. This will allow for temporary viewing of the file system. The file system can be unmounted with `umount` or by rebooting the local host.

2

Mounting with the Automounter

Chapter 5, “Using Autofs,” includes the specific instructions for establishing and supporting mounts with the automounter. Without any changes to the generic system, remote file systems should be accessible through the `/net` mount point. To mount the `/export/share/stuff` file system from the previous example, all you would need to do is:

```
% cd /net/bee/export/share/stuff
```

Since the automounter allows all users to mount file systems, root access is not required. It also provides for automatic unmounting of file systems, so there is no need to unmount file systems after you are done.

Setting Up and Maintaining NFS Security

3

This chapter discusses Secure NFS. It includes a description of the transactions between a server and a client using DES authentication and presents the administration procedures that are needed to set up and administer Secure NFS.

Secure NFS

The NFS environment is a powerful and convenient way to share file systems on a network of different computer architectures and operating systems. However, the same features that make sharing file systems through NFS operation convenient also pose some security problems. An NFS server authenticates a file request by authenticating the computer making the request, but not the user when using UNIX authentication. When using UNIX authentication, a client user can run `su` and impersonate the owner of a file. If DES authentication is used, the NFS server will authenticate the user, making this sort of impersonation much harder.

Given root access and knowledge of network programming, anyone is capable of introducing arbitrary data into the network and picking up any data from the network. The most dangerous attacks are those involving the introduction of data, such as impersonating a user by generating the right packets or

recording “conversations” and replaying them later. These attacks affect data integrity. Attacks involving passive eavesdropping—merely listening to network traffic without impersonating anybody—are not as dangerous, since data integrity is not compromised. Users can protect the privacy of sensitive information by encrypting data that goes over the network.

A common approach to network security problems is to leave the solution to each application. A better approach is to implement a standard authentication system at a level that covers all applications.

The Solaris operating system includes an authentication system at the level of remote procedure call (RPC)—the mechanism on which NFS operation is built. This system, known as Secure RPC, greatly improves the security of network environments and provides additional security to services such as the NFS system. When the NFS system uses the facilities provided by Secure RPC, it is known as Secure NFS.

Secure RPC

Secure RPC is fundamental to Secure NFS. The goal of Secure RPC is to build a system at least as secure as a time-sharing system (one in which all users share a single computer). A time-sharing system authenticates a user through a login password. With Data Encryption Standard (DES) authentication, the same is true. Users can log in on any remote computer just as they can on a local terminal, and their login passwords are their passports to network security. In a time-sharing environment, the system administrator has an ethical obligation not to change a password in order to impersonate someone. In Secure RPC, the network administrator is trusted not to alter entries in a database that stores *public keys*.

You need to be familiar with two terms to understand an RPC authentication system: *credentials* and *verifiers*. Using ID badges as an example, the credential is what identifies a person: a name, address, birthday, and so on. The verifier is the photo attached to the badge: you can be sure the badge has not been stolen by checking the photo on the badge against the person carrying it. In RPC, the client process sends both a credential and a verifier to the server with each RPC request. The server sends back only a verifier because the client already knows the server’s credentials.

RPC's authentication is open ended, which means that a variety of authentication systems may be plugged into it. Currently, there are three systems: UNIX, DES, and KERB (for Kerberos Version 4).

When UNIX authentication is used by a network service, the credentials contain the client's hostname, UID, GID, and group-accesslist, but the verifier contains nothing. Because there is no verifier, a superuser could deduce appropriate credentials, using commands such as `su`. Another problem with UNIX authentication is that it assumes all computers on a network are UNIX computers. UNIX authentication breaks down when applied to other operating systems in a heterogeneous network.

To overcome the problems of UNIX authentication, Secure RPC uses DES authentication—a scheme that employs verifiers, yet allows Secure RPC to be general enough to be used by most operating systems.

DES Authentication

DES authentication uses the Data Encryption Standard (DES) and Diffie-Hellman public-key cryptography to authenticate both users and computers in the network. DES is a standard encryption mechanism; Diffie-Hellman public-key cryptography is a cipher system that involves two keys: one public and one secret. The public and secret keys are stored in the namespace. NIS stores the keys in the `publickey` map, and NIS+ stores the keys in the cred table. These maps contain the public key and secret key for all potential users. See *System Administration Guide, Volume I*, for more information on how to set up the maps and tables.

The security of DES authentication is based on a sender's ability to encrypt the current time, which the receiver can then decrypt and check against its own clock. The time stamp is encrypted with DES. There are two requirements for this scheme to work:

- The two agents must agree on the current time
- The sender and receiver must be using the same encryption key

If a network runs a time-synchronization program, then the time on the client and the server is synchronized automatically. If a time synchronization program is not available, time stamps can be computed using the server's time instead of the network time. The client asks the server for the time before starting the RPC session, then computes the time difference between its own clock and the server's. This difference is used to offset the client's clock when

computing time stamps. If the client and server clocks get out of sync to the point where the server begins to reject the client's requests, the DES authentication system on the client resynchronizes with the server.

The client and server arrive at the same encryption key by generating a random *conversation key*, also known as the *session key*, and then using public-key cryptography to deduce a *common key*. The common key is a key that only the client and server are capable of deducing. The conversation key is used to encrypt and decrypt the client's time stamp; the common key is used to encrypt and decrypt the conversation key.

KERB Authentication

Kerberos is an authentication system developed at MIT. Encryption in Kerberos is based on DES.

Kerberos works by authenticating the user's login password. A user types the `kinit` command, which obtains a ticket that is valid for the time of the session (or eight hours, the default session time) from the authentication server. When the user logs out, the ticket may be destroyed using the `kdestroy` command.

The Kerberos server software is available from MIT Project Athena, and is not part of the SunOS software. SunOS software provides

- Routines used by the client to create, acquire, and verify tickets
- An authentication option to Secure RPC
- A client-side daemon, `kerbd`

See *System Administration Guide, Volume I,* for more details.

AUTH_DES Client-Server Session

This section describes the series of transactions in a client-server session using DES authorization (AUTH_DES).

Generating the public and secret keys

Sometime prior to a transaction, the administrator runs a program, either `newkey` or `nisaddcred` that generates a public key and a secret key. (Each user has a unique public key and secret key.) The public key is stored in a public database; the secret key is stored in encrypted form, in the same database. To change the key pair, use the `chkey` command.

Running `keylogin`

Normally, the login password is identical to the secure RPC password. In this case, a `keylogin` is not required. If the passwords are different, the users have to log in, and then do a `keylogin` explicitly.

The `keylogin` program prompts the user for a secure RPC password and uses the password to decrypt the secret key. The `keylogin` program then passes the decrypted secret key to a program called the *keyserver*. (The keyserver is an RPC service with a local instance on every computer.) The keyserver saves the decrypted secret key and waits for the user to initiate a secure RPC transaction with a server.

If the passwords are the same, the login process passes the secret key to the keyserver. If the passwords are required to be different and the user must always run `keylogin`, then the `keylogin` program may be included in the user's environment configuration file, such as `~/.login`, `~/.cshrc`, or `~/.profile`, so that it runs automatically whenever the user logs in.

Generating the Conversation Key

When the user initiates a transaction with a server:

1. The keyserver randomly generates a conversation key.
2. The kernel uses the conversation key to encrypt the client's time stamp (among other things).
3. The keyserver looks up the server's public key in the public-key database (see the `publickey(4)` man page).
4. The keyserver uses the client's secret key and the server's public key to create a common key.
5. The keyserver encrypts the conversation key with the common key.

First Contact with the Server

The transmission including the encrypted time stamp and the encrypted conversation key is then sent to the server. The transmission includes a credential and a verifier. The credential contains three components:

- The client's net name
- The conversation key, encrypted with the common key
- A "window," encrypted with the conversation key

The window is the difference the client says should be allowed between the server's clock and the client's time stamp. If the difference between the server's clock and the time stamp is greater than the window, the server would reject the client's request. Under normal circumstances this will not happen because the client first synchronizes with the server before starting the RPC session.

The client's verifier contains:

- The encrypted time stamp
- An encrypted verifier of the specified window, decremented by 1

The window verifier is needed in case somebody wants to impersonate a user and writes a program that, instead of filling in the encrypted fields of the credential and verifier, just stuffs in random bits. The server will decrypt the conversation key into some random key and use it to try to decrypt the window and the time stamp. The result will be random numbers. After a few thousand trials, however, there is a good chance that the random window/time stamp pair will pass the authentication system. The window verifier makes guessing the right credential much more difficult.

Decrypting the Conversation Key

When the server receives the transmission from the client:

1. The keyserver local to the server looks up the client's public key in the publickey database.
2. The keyserver uses the client's public key and the server's secret key to deduce the common key—the same common key computed by the client. (Only the server and the client can calculate the common key because doing so requires knowing one secret key or the other.)
3. The kernel uses the common key to decrypt the conversation key.
4. The kernel calls the keyserver to decrypt the client's time stamp with the decrypted conversation key.

Storing Information on the Server

After the server decrypts the client's time stamp, it stores four items of information in a credential table:

- The client's computer name

- The conversation key
- The window
- The client's time stamp

The server stores the first three items for future use. It stores the time stamp to protect against replays. The server accepts only time stamps that are chronologically greater than the last one seen, so any replayed transactions are guaranteed to be rejected.

Note – Implicit in these procedures is the name of the caller, who must be authenticated in some manner. The keyserver cannot use DES authentication to do this because it would create a deadlock. To solve this problem, the keyserver stores the secret keys by UID and grants requests only to local root processes.

Verifier Returned to the Client

The server returns a verifier to the client, which includes:

- The index ID, which the server records in its credential cache
- The client's time stamp minus 1, encrypted by conversation key

The reason for subtracting 1 from the time stamp is to ensure that the time stamp is invalid and cannot be reused as a client verifier.

Client Authenticates the Server

The client receives the verifier and authenticates the server. The client knows that only the server could have sent the verifier because only the server knows what time stamp the client sent.

Additional Transactions

With every transaction after the first, the client returns the index ID to the server in its second transaction and sends another encrypted time stamp. The server sends back the client's time stamp minus 1, encrypted by the conversation key.

You should be aware of the following points if you plan to use Secure RPC:

- If a server crashes when no one is around (after a power failure for example), all the secret keys that are stored on the system are wiped out. Now no process is able to access secure network services or mount an NFS

file system. The important processes during a reboot are usually run as root, so things would work if root's secret key were stored away, but nobody is available to type the password that decrypts it. `keylogin -r` allows root to store the clear secret key in `/etc/.rootkey` which `keyserv` reads.

- Some systems boot in single-user mode, with a root login shell on the console and no password prompt. Physical security is imperative in such cases.
- Diskless computer booting is not totally secure. Somebody could impersonate the boot server and boot a devious kernel that, for example, makes a record of your secret key on a remote computer. Secure NFS provides protection only after the kernel and the keyserver are running. Before that, there is no way to authenticate the replies given by the boot server. This could be a serious problem, but it requires a sophisticated attack, using kernel source code. Also, the crime would have evidence. If you polled the network for bootservers, you would discover the devious bootserver's location.
- Most setuid programs are owned by root; if root's secret key is stored in `/etc/.rootkey`, these programs behave as they always have. If a setuid program is owned by a user, however, it may not always work. For example, if a setuid program is owned by `dave` and `dave` has not logged into the computer since it booted, then the program would not be able to access secure network services.
- If you log in to a remote computer (using `login`, `rlogin`, or `telnet`) and use `keylogin` to gain access, you give away access to your account. This is because your secret key gets passed to that computer's keyserver, which then stores it. This is only a concern if you don't trust the remote computer. If you have doubts, however, don't log in to a remote computer if it requires a password. Instead, use the NFS environment to mount file systems shared by the remote computer. As an alternative, you can use `keylogout` to delete the secret key from the keyserver.
- If a home directory is shared with the `-o secure` or `-o kerberos` options, then remote logins can be a problem. If the `/etc/hosts.equiv` or `~/.rhosts` files are set to not prompt for a password, the login will succeed, but the user will not be able to access their home directory since no authentication has occurred locally. If the user is prompted for a password, then as long as the password matches the network password, the user will have access to their home directory.

Administering Secure NFS

To use Secure NFS, all the computers you are responsible for must have a domain name. A *domain* is an administrative entity, typically consisting of several computers, that joins a larger network. If you are running NIS+, you should also establish the NIS+ name service for the domain. See *NIS+ and DNS Setup and Configuration Guide.*

With UNIX authentication a universal flat name space is assumed. UIDs are normally unique within a domain but may not be unique across domains. A problem with this scheme is that UIDs clash when domains are linked across the network. Another problem with UNIX authentication has to do with superusers; with UNIX authentication, the superuser ID (UID 0) is in effect assigned one per computer, not one per domain. Therefore, a domain can have multiple superusers—all with the same UNIX UID.

DES authentication corrects these problems by using netnames. A *netname* is a string of printable characters created by concatenating the name of the operating system, a user ID, and a domain name. For example, a UNIX system user with a user ID of 508 in the domain `eng.acme.com` would be assigned the following netname `unix.508@eng.acme.com`. Because user IDs must be unique within a domain and because domain names must be unique on a network, this scheme produces a unique netname for every user.

To overcome the problem of multiple superusers per domain, netnames are assigned to computers as well as to users. A computer's netname is formed much like a user's—by concatenating the name of the operating system and the computer name with the domain name. The root user on a UNIX computer named `hal` in the domain `eng.acme.com` would have the netname `unix.hal@eng.acme.com`.

▼ How to Set Up Secure NFS

1. **Assign your domain a domain name, and make the domain name known to each computer in the domain.**
   See the *NIS+ and FNS Administration Guide* if you are using NIS+ as your name service.

2. **Establish public keys and secret keys for your clients' users using the** `newkey` **or** `nisaddcred` **command, and have each user establish his or her own secure RPC password using the** `chkey` **command.**

Note – For information about these commands, see the `newkey(1M)`, the `nisaddcred(1M)`, and the `chkey(1)` man pages.

When public and secret keys have been generated, the public and encrypted secret keys are stored in the `publickey` database.

3. **Verify that the name service is responding. If you are running NIS+, type the following:**

```
# nisping -u
Last updates for directory eng.acme.com. :
Master server is eng-master.acme.com.
        Last update occurred at Mon Jun  5 11:16:10 1995

Replica server is eng1-replica-replica-58.acme.com.
        Last Update seen was Mon Jun  5 11:16:10 1995
```

If you are running NIS, verify that the `ypbind` daemon is running.

4. **Verify that the** `keyserv` **daemon (the keyserver) is running, type the following:**

```
# ps -ef | grep keyserv
root   100    1 16   Apr 11 ?    0:00 /usr/sbin/keyserv
root  2215 2211  5 09:57:28 pts/0 0:00 grep keyserv
```

If it isn't running, to start the keyserver, type the following:

```
# /usr/sbin/keyserv
```

5. **Run** `keylogin` **to decrypt and store the secret key.**
Usually, the login password is identical to the network password. In this case, `keylogin` is not required. If the passwords are different, the users have to log in, and then do a `keylogin`. You still need to use the `keylogin -r` command as root to store the decrypted secret key in `/etc/.rootkey`.

Note – `keylogin -r` will only need to be run if the root secret key changes or `/etc/.rootkey` is lost.

6. **Edit the** `/etc/dfs/dfstab` **file and add the** `secure` **option to the appropriate entries (for DES authentication).**

```
# share -F nfs -o secure /export/home
```

For KERB authentication, add the `kerberos` option.

```
# share -F nfs -o kerberos /export/home
```

7. **Edit the** `auto_master` **data to include** `secure` **as a mount option in the appropriate entries (for DES authentication):**

```
/home          auto_home     -nosuid,secure
```

For KERB authentication, add the `kerberos` option.

```
/home          auto_home     -nosuid,kerberos
```

Note – With Version 2 NFS software, if a client does not mount as `secure` a file system that is shared as secure, users have access as user `nobody`, rather than as themselves. With Version 3, the secure flag will be inherited from the NFS server, so there is no need for the clients to specify `kerberos` or `secure`. The users will have access to the files as themselves.

When you reinstall, move, or upgrade a computer, remember to save `/etc/.rootkey` if you don't establish new keys or change them for root. If you do delete `/etc/.rootkey`, you can always type:

```
# keylogin -r
```

3

NFS Troubleshooting 4

This chapter describes problems that may occur on computers using NFS services. It contains procedures for fixing and tracking NFS problems. A reference section is also included. If you want to skip the background information that explains NFS internals and proceed directly to step-by-step instructions, use the following table to find the page where instructions for specific tasks begin.

Strategies for NFS Troubleshooting

When tracking down an NFS problem, keep in mind that there are three main points of possible failure: the server, the client, and the network. The strategy outlined in this section tries to isolate each individual component to find the one that is not working. In all cases, the `mountd` and `nfsd` daemons must be running on the server for remote mounts to succeed.

Note – The `mountd` and `nfsd` daemons start automatically at boot time only if there are NFS share entries in the `/etc/dfs/dfstab` file. Therefore, `mountd` and `nfsd` must be started manually when setting up sharing for the first time.

The `intr` option is set by default for all mounts. If a program hangs with a "server not responding" message, it can be killed with the keyboard interrupt Control-c.

When the network or server has problems, programs that access hard-mounted remote files will fail differently than those that access soft-mounted remote files. Hard-mounted remote file systems cause the client's kernel to retry the requests until the server responds again. Soft-mounted remote file systems cause the client's system calls to return an error after trying for a while. Because these errors may result in unexpected application errors, soft mounting is not recommended.

When a file system is hard mounted, a program that tries to access it hangs if the server fails to respond. In this case, the NFS system displays the following message on the console.

```
NFS server hostname not responding still trying
```

When the server finally responds, the following message appears on the console.

```
NFS server hostname ok
```

A program accessing a soft-mounted file system whose server is not responding will generate the following message:

```
NFS operation failed for server hostname: error # (error_message)
```

Note – Because of possible errors, do not soft-mount file systems with read-write data or file systems from which executables will be run. Writable data could be corrupted if the application ignores the errors. Mounted executables may not load properly and can fail.

NFS Troubleshooting Procedures

To determine where the NFS service has failed, it is necessary to follow several procedures to isolate the failure. The following items need to be checked:

- Can the client reach the server?
- Can the client contact the NFS services on the server?
- Are the NFS services running on the server?

In the process of checking these items it may become apparent that other portions of the network are not functioning, such as the name service or the physical network hardware. Debugging procedures for the NIS+ name service are found in *NIS+ and FNS Administration Guide*. Also, during the process it may become obvious that the problem isn't at the client end (for instance, if you get at least one trouble call from every subnet in your work area). In this case, it is much more timely to assume that the problem is the server or the network hardware near the server, and start the debugging process at the server not at the client.

▼ How to Check Connectivity on a NFS Client

1. **Make sure that the NFS server is reachable from the client. On the client, type the following command.**

```
% /usr/sbin/ping bee
bee is alive
```

   If the command reports that the server is alive, remotely check the NFS server (see "How to Remotely Check the NFS Server" on page 54).

2. **If the server is not reachable from the client, make sure that the local name service is running. For NIS+ clients type the following:**

```
% /usr/lib/nis/nisping -u
Last updates for directory eng.acme.com. :
Master server is eng-master.acme.com.
        Last update occurred at Mon Jun  5 11:16:10 1995

Replica server is eng1-replica-58.acme.com.
        Last Update seen was Mon Jun  5 11:16:10 1995
```

3. **If the name service is running, but the server is not reachable from the client, run the `ping` command from another client.**
   If the command run from a second client fails, see "How to Verify the NFS Service on the Server" on page 55.

4. **If the server is reachable from the second client, use `ping` to check connectivity of the first client to other systems on the local net.**
   If this fails, check the networking software configuration on the client (`/etc/netmasks`, `/etc/nsswitch.conf`, and so forth).

5. **If the software is correct, check the networking hardware.**
   Try moving the client onto a second net drop.

▼ How to Remotely Check the NFS Server

1. **Check that the server's `nfsd` processes are responding. On the client, type the following command.**

```
% /usr/bin/rpcinfo -u bee nfs
program 100003 version 2 ready and waiting
program 100003 version 3 ready and waiting
```

   If the server is running, it prints a list of program and version numbers. Using the `-t` option will test the TCP connection. If this fails, skip to "How to Verify the NFS Service on the Server" on page 55.

2. **Check that the server's `mountd` is responding, by typing the following command.**

```
% /usr/bin/rpcinfo -u bee mountd
program 100005 version 1 ready and waiting
program 100005 version 2 ready and waiting
program 100005 version 3 ready and waiting
```

   Using the `-t` option will test the TCP connection. If either fails, skip to "How to Verify the NFS Service on the Server" on page 55.

3. **Check the local autofs service, if it is being used:**

```
% cd /net/wasp
```

Choose a `/net` or `/home` mount point that you know should work properly. If this doesn't work, then as root on the client, type the following to restart the autofs service.

```
# /etc/init.d/autofs stop
# /etc/init.d/autofs start
```

4. **Verify that file system is shared as expected on the server.**

```
% /usr/sbin/showmount -e bee
/usr/src          eng
/export/share/man  (everyone)
```

Check the entry on the server and the local mount entry for errors. Also check the name space. In this instance, if the first client is not in the `eng` netgroup, then they would not be able to mount the `/usr/src` file system.

Make sure to check the entries in all of the local files that include mounting information. The list includes `/etc/vfstab` and all of the `/etc/auto_*` files.

▼ How to Verify the NFS Service on the Server

1. **Log onto the server as root.**

2. **Make sure that the server can reach the clients.**

```
# ping lilac
lilac is alive
```

3. **If the client is not reachable from the server, make sure that the local name service is running. For NIS+ clients type the following:**

```
% /usr/lib/nis/nisping -u
Last updates for directory eng.acme.com. :
Master server is eng-master.acme.com.
        Last update occurred at Mon Jun  5 11:16:10 1995

Replica server is eng1-replica-58.acme.com.
        Last Update seen was Mon Jun  5 11:16:10 1995
```

4. **If the the name service is running, check the networking software configuration on the server (**`/etc/netmasks`**,** `/etc/nsswitch.conf`**, and so forth).**

5. **Type the following command to check whether the** `nfsd` **daemon is running.**

```
# rpcinfo -u localhost nfs
program 100003 version 2 ready and waiting
program 100003 version 3 ready and waiting
# ps -ef | grep nfsd
root   232   1 0   Apr 07 ?    0:01 /usr/lib/nfs/nfsd -a 16
root  3127 24621 09:32:57 pts/3 0:00 grep nfsd
```

Also use the `-t` option with `rpcinfo` to check the TCP connection. If these commands fail, restart the NFS service (see "How to Restart NFS Services" on page 57).

6. **Type the following command to check whether the `mountd` daemon is running.**

```
# /usr/bin/rpcinfo -u localhost mountd
program 100005 version 1 ready and waiting
program 100005 version 2 ready and waiting
program 100005 version 3 ready and waiting
# ps -ef | grep mountd
root   145   1 0   Apr 07 ?   21:57 /usr/lib/autofs/automountd
root   234   1 0   Apr 07 ?    0:04 /usr/lib/nfs/mountd
root  3084 24621 09:30:20 pts/3 0:00 grep mountd
```

Also use the `-t` option with `rpcinfo` to check the TCP connection. If these commands fail, restart the NFS service (see "How to Restart NFS Services" on page 57).

7. **Type the following command to check whether the `rpcbind` daemon is running.**

```
# /usr/bin/rpcinfo -u localhost rpcbind
program 100000 version 1 ready and waiting
program 100000 version 2 ready and waiting
program 100000 version 3 ready and waiting
```

If `rpcbind` seems to be hung, either reboot the server or follow the steps in "How to Warm Start rpcbind" on page 58.

▼ How to Restart NFS Services

♦ **To enable daemons without rebooting, become superuser and type the following commands.**

```
# /etc/init.d/nfs.server stop
# /etc/init.d/nfs.server start
```

This will stop the daemons and restart them, if there is an entry in `/etc/dfs/dfstab`.

▼ How to Warm Start `rpcbind`

If the NFS server can not be rebooted because of work in progress, it is possible to restart `rpcbind` without having to restart all of the services which use RPC by completing a warm start as described below.

1. **As root on the server, get the PID for `rpcbind`.**
   Run `ps` to get the PID (which will be the value in the second column).

```
# ps -ef |grep rpcbind
    root   115     1  0   May 31 ?        0:14 /usr/sbin/rpcbind
    root 13000  6944  0 11:11:15 pts/3    0:00 grep rpcbind
```

2. **Send a SIGTERM signal to the `rpcbind` process.**
   In this example, term is the signal that is to be sent and 115 is the PID for the program (see the `kill(1)` man page). This will cause rpcbind to create a list of the current registered services in `/tmp/portmap.file` and `/tmp/rpcbind.file`.

```
# kill -s term 115
```

Note – If the `rpcbind` process is not killed with the `-s` option, then a warm start of `rpcbind` is not possible.

3. **Restart `rpcbind`.**
   Do a warm restart of the command so that the files created by the kill command are consulted, so that the process resumes without requiring that all of the RPC services be restarted (see the `rpcbind(1M)`man page).

```
# /usr/sbin/rpcbind -w
```

Common NFS Error Messages

`mount: ... server not responding:RPC_PMAP_FAILURE - RPC_TIMED_OUT`

The server sharing the file system you are trying to mount is down or unreachable, at the wrong run level, or its `rpcbind` is dead or hung.

`mount: ... server not responding: RPC_PROG_NOT_REGISTERED`

mount registered with rpcbind, but the NFS mount daemon mountd is not registered.

`mount: ... No such file or directory`

Either the remote directory or the local directory does not exist. Check the spelling of the directory names. Run `ls` on both directories.

`mount: ...: Permission denied`

Your computer name may not be in the list of clients or netgroup allowed access to the file system you want to mount. Use `showmount -e` to verify the access list.

`NFS server` *hostname* `not responding still trying`

If programs hang while doing file-related work, your NFS server may be dead. This message indicates that NFS server *hostname* is down or that there is a problem with the server or with the network. Start with “How to Check Connectivity on a NFS Client” on page 53.

`NFS fsstat failed for server` *hostname*`: RPC: Authentication error`

This error can be caused by many situations. One of most difficult to debug is when this occurs because a user is in too many groups. Currently a user may be in as many as 16 groups but no more if they are accessing files through NFS mounts. If a user must have the functionality of being in more than 16 groups and if Solaris 2.5 is running on the NFS server and the NFS clients, then use ACLs to provide the needed access privileges.

4

Using Autofs 5

This chapter tells you how to use autofs, a new implementation of automatic mounting. It includes some common scenarios for use of autofs and how to design autofs maps to best meet your needs for accessing file systems.

How Autofs Works

Autofs is a client-side service. When a client attempts to access a file system that is not presently mounted, the autofs file system intercepts the request and calls `automountd`, to mount the requested directory. The `automountd` daemon locates the directory, mounts it within autofs, and replies. On receiving the reply, autofs allows the waiting request to proceed. Subsequent references to the mount are redirected by the autofs—no further participation is required by `automountd`.

Three components that work together to accomplish automatic mounting are:

- The `automount` command
- The autofs file system
- The `automountd` daemon

The `automount` command, called at system startup time, reads the master map file `auto_master` to create the initial set of autofs mounts. These autofs mounts are not automatically mounted at startup time. They are points under which file systems will be mounted in the future.

Once the autofs mounts are set up, they can trigger file systems to be mounted under them. For example, when autofs receives a request to access a file system that is not currently mounted, autofs calls `automountd`, which actually mounts the requested file system.

With this new implementation of automatic mounting the `automountd` daemon is completely independent from the `automount` command. Because of this separation, it's possible to add, delete, or change map information without first having to stop and start the `automountd` daemon process. Once the file system is mounted, further access does not require any action from `automountd`.

After initially mounting autofs mounts, the `automount` command is used to keep autofs mounts as necessary by comparing the list of mounts in the `auto_master` map with the list of mounted file systems in the mount table file `/etc/mnttab` (formerly `/etc/mtab`) and making the appropriate changes. This allows system administrators to change mount information within `auto_master` and have those changes used by the autofs processes without having to stop and restart the autofs daemons.

Unlike `mount`, `automount` does not read the `/etc/vfstab` file (which is specific to each computer) for a list of file systems to mount. The `automount` command is controlled within a domain and on computers through the namespace or local files.

This is a simplified overview of how autofs works:

The automount daemon `automountd` starts at boot time from the `/etc/init.d/autofs` script. This script also runs the `automount` command, which reads the master map (see "How Autofs Navigates Through the Network (Maps)" on page 69) and installs autofs mount points.

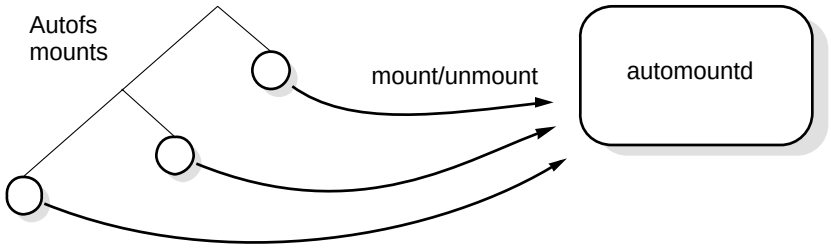


Figure 5-1 `/etc/init.d/autofs` Script Starts `automount`

Autofs is a kernel file system that supports automatic mounting and unmounting.

When a request is made to access a file system at an autofs mount point:

1. Autofs intercepts the request.
2. Autofs sends a message to the `automountd` for the requested file system to be mounted.
3. `automountd` locates the file system information in a map and performs the mount.
4. Autofs allows the intercepted request to proceed.
5. Autofs unmounts the file system after five minutes of inactivity.

Note – Mounts managed through the autofs service should not be manually mounted or unmounted. Even if the operation is successful, the autofs service will not know that the object has been unmounted resulting in possible inconsistency. A reboot will clear all of the autofs mount points, if the service is confused by manual interaction.

5

Autofs Programs

automount

This command installs autofs mount points and associates the information in the automaster files with each mount point. The syntax of the command is:

`automount [ -t` *duration* `] [ -v ]`

where `-t` *duration* sets the time, in seconds, that a file system is to remain mounted, and `-v` selects the verbose mode. Running this command in the verbose mode allows for easier troubleshooting.

If not specifically set, the value for duration is set to 5 minutes. In most circumstances this is a good value, however, on systems which have many automounted file systems, it can be necessary to increase the duration value. In particular, if a server has many users active checking the automounted file systems every five minutes can be inefficient. Checking the autofs file systems every 1800 seconds (or 30 minutes) could be more optimal. By not unmounting the file systems every 5 minutes, it is possible that `/etc/mnttab` which is checked by `df` can get very large. The output from `df` can be filtered by using the `-F` option (see the `df(1B)` man page) or by using `egrep` to help fix this problem.

Another factor to consider is that changing the duration also changes how quickly changes to the automounter maps will be reflected.

automountd

This daemon handles the mount and unmount requests from the autofs service. The syntax of the command is:

`automountd [ -Tv ] [ -D` *name=value* `]`

where `-T` selects to display each RPC call to standard output, `-v` selects to log all status messages to the console, and `-D` *name=value* substitutes *value* for the automount map variable indicated by *name*. The `-T` option is only recommended for troubleshooting.

Setting Up Autofs Maps

Autofs uses three types of maps:

- Master maps
- Direct maps
- Indirect maps

Master Maps

The `auto_master` map associates a directory with a map. It is a master list specifying all the maps that autofs should know about.

Each line in the master map `/etc/auto_master` has the following syntax:

mount-point *map-name* [*mount-options*]

`mount-point`

mount-point is the full (absolute) path name of a directory. If the directory does not exist, autofs creates it if possible. If the directory exists and is not empty, mounting on it hides its contents. In this case, autofs issues a warning message.

`map-name`

map-name is the map autofs uses to find directions to locations, or mount information. If the name is preceded by a slash (/), autofs interprets the name as a local file. Otherwise, autofs searches for the mount information using the search specified in the name service switch configuration file.

`mount-options`

mount-options is an optional, comma-separated list of options that apply to the mounting of the entries specified in *map-name*, unless the entries in *map-name* list other options. The *mount-options* are the same as those for a standard NFS `mount`, except that `bg` (background) and `fg` (foreground) do not apply. See mount on page 15.

A line beginning with # is a comment. Everything that follows until the end of the line is ignored.

To split long lines into shorter ones, put a backslash (\ ) at the end of the line.

The notation `/-` as a mount point indicates that the map in question is a direct map, and no particular mount point is associated with the map as a whole.

Direct Maps

A direct map is an automount point. With a direct map, there is a direct association between a mount point on the client and a directory on the server. Direct maps have a full path name and indicate the relationship explicitly. Lines in direct maps have the following syntax:

key [*mount-options*] *location*

`key`

key is the path name of the mount point in a direct map.

`mount-options`

mount-options are the options you want to apply to this particular mount. They are required only if they differ from the map default.

`location`

location is the location of the file system, specified (one or more) as *server:pathname.*

Warning – The *pathname* should not include an automounted mount point, it should be the actual absolute path to the files system. For instance, the location of a home directory should be listed as *server*`:/export/home/`*username* not as *server*`:/home/`*username*.

As in the master map, a line beginning with # is a comment. All the text that follows until the end of the line is ignored. Put a backslash at the end of the line to split long lines into shorter ones.

Of all the maps, the entries in a direct map most closely resemble, in their simplest form, the corresponding entries in `/etc/vfstab` (`vfstab` contains a list of all file systems to be mounted). An entry that appears in `/etc/vfstab` as:

```
dancer:/usr/local - /usr/local/tmp nfs - yes ro
```

appears in a direct map as:

```
/usr/local/tmp     -ro     dancer:/usr/local
```

This is a typical `/etc/auto_direct` map:

```
/usr/local          -ro \
    /bin                        ivy:/export/local/sun4 \
    /share                      ivy:/export/local/share \
    /src                        ivy:/export/local/src
/usr/man            -ro         oak:/usr/man \
                                rose:/usr/man \
                                willow:/usr/man
/usr/games          -ro         peach:/usr/games
/usr/spool/news     -ro         pine:/usr/spool/news \
                                willow:/var/spool/news
```

There are important but previously unmentioned features in this map. See ”How Autofs Selects the Nearest Read-Only Files for Clients (Multiple Locations)“ on page 76 and ”Multiple Mounts“ on page 73.

Indirect Maps

An indirect map uses a substitution value of a key to establish the association between a mount point on the client and a directory on the server. Indirect maps are useful for accessing specific file systems, like home directories. The `auto_home` map is an example of an indirect map.

Lines in indirect maps have the following general syntax:

key [*mount-options*] *location*

`key`

key is a simple name (no slashes) in an indirect map.

`mount-options`

The *mount-options* are the options you want to apply to this particular mount. They are required only if they differ from the map default.

`location`

location is the location of the file system, specified (one or more) as *server*:*pathname*.

Warning – The *pathname* should not include an automounted mount point, it should be the actual absolute path to the files system. For instance, the location of a directory should be listed as *server*`:/usr/local` not as *server*`:/net/`*server*`/usr/local`.

As in the master map, a line beginning with # is a comment. All the text that follows until the end of the line is ignored. Put a backslash (\) at the end of the line to split long lines into shorter ones.

Table 5-1 showed an `auto_master` map that contained the entry:

```
/home      auto_home
```

`auto_home` is the name of the indirect map that contains the entries to be mounted under `/home`. A typical `auto_home` map might contain:

```
david                     willow:/export/home/david
rob                       cypress:/export/home/rob
gordon                    poplar:/export/home/gordon
rajan                     pine:/export/home/rajan
tammy                     apple:/export/home/tammy
jim                       ivy:/export/home/jim
linda        -rw,nosuid   peach:/export/home/linda
```

As an example, assume that the previous map is on host `oak`. If user `linda` has an entry in the password database specifying her home directory as `/home/linda`, then whenever she logs into computer `oak`, autofs mounts the directory `/export/home/linda` residing on the computer `peach`. Her home directory is mounted read-write, `nosuid`.

Assume the following conditions occur: User `linda`'s home directory is listed in the password database as `/home/linda`. Anybody, including Linda, has access to this path from any computer set up with the master map referring to the map in the previous example.

Under these conditions, user `linda` can run `login` or `rlogin` on any of these computers and have her home directory mounted in place for her.

Furthermore, now `linda` can also type the following command:

```
% cd ~david
```

autofs mounts David's home directory for her (if all permissions allow).

Note – There is no concatenation of options between the automounter maps. Any options added to an automounter map will override all options listed in maps that are searched earlier. For instance, options included in the `auto_master` map would be overwritten by corresponding entries in any other map.

On a network without a name service, you have to change all the relevant files (such as `/etc/passwd`) on all systems on the network to accomplish this. With NIS, make the changes on the NIS master server and propagate the relevant databases to the slave servers. On a network running NIS+, propagating the relevant databases to the slave servers is done automatically after the changes are made.

How Autofs Navigates Through the Network (Maps)

Autofs searches a series of maps to navigate its way through the network. Maps are files that contain information such as the password entries of all users on a network or the names of all host computers on a network; that is, network-wide equivalents of UNIX administration files. Maps are available locally or through a network name service like NIS or NIS+. You create maps to meet the needs of your environment using the Solstice System Management Tools. See "Modifying How Autofs Navigates the Network (Modifying Maps)" on page 81.

How Autofs Starts the Navigation Process (Master Map)

The `automount` command reads the master map at system startup. Each entry in the master map is a direct or indirect map name, its path, and its mount options, as shown in Figure 5-2. The specific order of the entries is not important. `automount` compares entries in the master map with entries in the mount table to generate a current list.

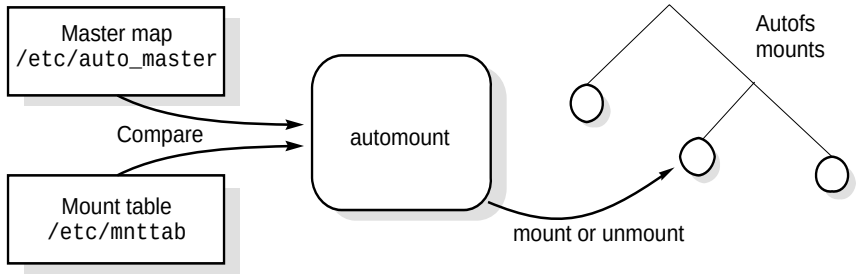


Figure 5-2 Master Map

For example, Table 5-1 shows what a typical `auto_master` file would contain:

Table 5-1 `auto_master` File Contents

Mount Point	Map	Mount options
/-	auto_direct	-ro
/home	auto_home	-nosuid
/net	-hosts	-nosuid

Autofs recognizes some special mount points and maps, which are explained in the following sections.

Mount Point /–

In Table 5-1, the mount point `/-` tells autofs not to associate the entries in `auto_direct` with any specific mount point. Indirect maps use mount points. Direct maps use mount points specified in the named map. (Remember, in a direct map the key, or mount point, is a full path name.)

A NIS or NIS+ `auto_master` file can have only one direct map entry because the mount point must be a unique value in the name space. An `auto_master` file that is a local file can have any number of direct map entries, as long as they do not overlap.

Mount Point `/home`

The mount point `/home` is the directory under which the entries listed in `/etc/auto_home` (an indirect map) are to be mounted.

Mount Point `/net`

Autofs mounts under the directory `/net` all the entries in the special map `-hosts`. This is a built-in map that uses only the `hosts` database. For example, if the computer `gumbo` is in the `hosts` database and it exports any of its file systems, the command

```
% cd /net/gumbo
```

changes the current directory to the root directory of the computer `gumbo`. Note that autofs can mount only the *exported* file systems of host `gumbo`; that is, those on a server available to network users as opposed to those on a local disk. Therefore, all the files and directories on `gumbo` may not be available through `/net/gumbo`.

Note – Autofs checks the server's export list only at mount time. Once a server's file systems are mounted, autofs does not check with the server again until the server's file systems are unmounted and then remounted. Therefore, newly exported file systems will not be seen until the file systems on the server are unmounted or remounted.

The Mount Process

When you issue the command in the previous example, autofs performs the following steps:

1. `ping`s the server's mount service to see if it's alive.

2. Requests the list of exported file systems from the server.

3. Sorts the exported list according to length of path name:

```
/usr/src
/export/home
/usr/src/sccs
/export/root/client
```

   This sorting ensures that the mounting is done in the required order (that is, `/usr/src` is done before `/usr/src/sccs`).

4. Proceeds down the list, mounting all the file systems at mount points.

Note that autofs has to mount all the file systems that the server in question exports. Even if the request is as follows.

```
% ls /net/gumbo/usr/include
```

Autofs mounts all of `gumbo`'s exported systems, not just `/usr`.

If autofs is running on NFS servers, any maps that refer to file systems on the server should be checked for file name paths that pass through an autofs mount point. This causes an access through the loopback file system (LOFS) which cannot be exported. The entry would appear as follows.

```
brent   creole:/home/brent
```

Replace it with this entry:

```
brent   creole:/export/home/creole/brent
```

In previous releases, the `mount` daemon on the server would follow the automounter's symbolic link at `/home/brent` and find the exported file system. With autofs, the mount daemon finds a loopback mount at `/home/brent` that is not exportable so the client will not be able to mount it.

Note – Check existing maps to make sure that the *server:/path* portion of the map entry does not refer to an autofs mount point.

The unmounting that occurs after a certain amount of time is from the bottom up (reverse order of mounting). If one of the directories at the top is busy, autofs has to remount the unmounted file systems and try again later.

The `-hosts` special map provides a convenient way for users to access directories in many different hosts without having to use `rlogin` or `rsh`. They no longer have to modify `/etc/vfstab` files or mount the directories individually as superuser.

With the `/net` method of access, the server name is in the path and is location-dependent. If you want to move an exported file system from one server to another, the path may no longer work. Also, because all exported file systems need to be mounted, using only one of those file systems is inefficient. Instead, you should set up an entry in a map specifically for the file system you want rather than use `/net`.

Note – Autofs runs on all computers and supports `/net` and `/home` (automounted home directories) by default. These defaults may be overridden by entries in the NIS `auto.master` map or NIS+ `auto_master` table, or by local editing of the `/etc/auto_master` file.

Multiple Mounts

A map entry can describe multiple mounts. Multiple mounts enable users to access file systems from different locations. By having the same applications on several servers, users have an alternate source for that application if a particular server is down. Also, autofs can choose the quickest path for users (clients). Consider the first entry in ”Direct Maps“ on page 66:

```
/usr/local   -ro \
    /bin                 ivy:/export/local/sun3 \
    /share               ivy:/export/local/share\
    /src                 ivy:/export/local/src
```

This is actually one long entry split into four lines using the backslash with the continuation lines indented with spaces or tabs. This entry mounts `/usr/local/bin`, `/usr/local/share`, and `/usr/local/src` from the server `ivy`, with the read-only option. The entry could also read:

```
/usr/local \
   /bin      -ro          ivy:/export/local/sun3 \
   /share    -rw,secure   willow:/usr/local/share\
   /src      -ro          oak:/home/jones/src
```

where the options are different and more than one server is used. The previous example is equivalent to three separate entries, for example:

```
/usr/local/bin   -ro                   ivy:/export/local/sun3
/usr/local/share -rw,secure            willow:/usr/local/share
/usr/local/src   -ro                   oak:/home/jones/src
```

Multiple mounting guarantees that all three directories are mounted when you refer to one of them. If the entries are listed as separate mounts, then each of the directories is mounted only as needed. The first (multiple mount) case is accomplished with a single autofs mount at `/usr/local`. The second (single mounts) case results in three independent autofs mounts.

The mount root is a path relative to a direct autofs mount, or relative to the directory under an indirect autofs mount. This path describes where each file system should be mounted beneath an autofs mount point. This mount point theoretically should be specified by.

```
parsley      /    -ro     veg:/usr/greens
```

But in practice, the mount point is not specified because in the case of a single mount as in the previous example, the location of the mount point is *at* the mount root or "`/`." So instead of the previous example, you would use this entry:

```
parsley      -ro          veg:/usr/greens
```

The mount-point specification is important in a multiple-mount entry. Autofs must have a mount point for each mount. When the entry specifies that one mount occur within another, the entry becomes a hierarchical mount, which is a special case of multiple mounts.

Note – A hierarchical mount can be a problem if the server for the root of the file system goes down. Because the unmounting has to proceed through the mount root, which also cannot be unmounted while its server is down, any attempt to unmount the lower branches fails.

The mount points used here for the file system are `/`, `/bin`, `/share`, and `/src`. Note that these mount point paths are relative to the *mount* root, not the host's *file system* root.

```
/usr/local \
     /       -rw          peach:/export/local \
     /bin    -ro          ivy:/export/local/sun3 \
     /share  -rw          willow:/usr/local/share \
     /src    -ro          oak:/home/jones/src
```

The first entry in the previous example has `/` as its mount point. It is mounted *at* the mount root. The first mount of a file system does not need to be at the mount root. Autofs issues `mkdir` commands to build a path to the first mount point if it is not at the mount root.

In these mount option examples:

```
/usr/local \
  /bin       -ro          ivy:/export/local/$CPU \
  /share     -ro          willow:/usr/local/share \
  /src       -ro          oak:/home/jones/src
```

all three mounts share the same options. You can change this to:

```
/usr/local   -ro\
    /bin                  ivy:/export/local/sun4 \
    /share                willow:/usr/local/share \
    /src                  oak:/home/jones/src
```

Administration is easier if there is only one set of mount options common to all mounts in the entry. If one of the mount points needs a different specification, you can write:

```
/usr/local    -ro\
    /bin                    ivy:/export/local/sun4 \
    /share    -rw,secure    willow:/usr/local/share \
    /src                    oak:/home/jones/src
```

You may want different mount options for some of the mounts; for example, to enable clients to update the files on one mount but not on the others.

How Autofs Selects the Nearest Read-Only Files for Clients (Multiple Locations)

In the example of a direct map, which was:

```
/usr/local                -ro \
    /bin                                ivy:/export/local/sun4\
    /share                              ivy:/export/local/share\
    /src                                ivy:/export/local/src
/usr/man                  -ro           oak:/usr/man \
                                        rose:/usr/man \
                                        willow:/usr/man
/usr/games                -ro           peach:/usr/games
/usr/spool/news           -ro           pine:/usr/spool/news \
                                        willow:/var/spool/news
```

the mount points `/usr/man` and `/usr/spool/news` list more than one location (three for the first, two for the second). This means users can mount from any of the replicated locations. This procedure makes sense only when you mount a file system that is read-only, since you must have some control over the locations of files you write or modify. You don't want to modify files on one server on one occasion and, minutes later, modify the "same" file on another server. The benefit is that the best available server will be mounted automatically without any effort required by the user.

A good example of this is man pages. In a large network, more than one server may export the current set of manual pages. Which server you mount them from does not matter, as long as the server is running and exporting its file systems. In the previous example, multiple mount locations are expressed as a list of mount locations in the map entry.

```
/usr/man -ro oak:/usr/man rose:/usr/man willow:/usr/man
```

You could also enter this as a comma-separated list of servers, followed by a colon and the path name (as long as the path name is the same for all the replicated servers).

```
/usr/man -ro oak,rose(1),willow(2):/usr/man
```

Here you can mount the man pages from the servers `oak`, `rose`, or `willow`. The numbers in parentheses indicate a weighting. Servers without a weighting have a value of zero (most likely to be selected). The higher the weighting value, the lower the chance the server will be selected.

Note – Server proximity is more important than weighting. A server on the same network segment as the client is more likely to be selected than a server on another network segment, regardless of the weighting factors assigned.

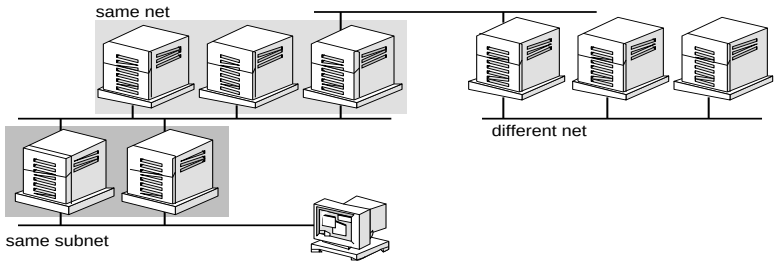


Figure 5-3 Server Proximity

This redundancy is used once at mount time to select one server from which to mount. Autofs does not check the status of the mounted-from server once the mount occurs, so this is not useful as a fail-over service. Multiple locations are very useful in an environment where individual servers may not be exporting their file systems temporarily.

This feature is particularly useful in a large network with many subnets. Autofs chooses the nearest server and therefore confines NFS network traffic to a local network segment. In servers with multiple network interfaces, list the host name associated with each network interface as if it were a separate server. Autofs selects the nearest interface to the client.

Variables in a Map Entry

You can create a client-specific variable by prefixing a dollar sign ($) to its name. This helps you to accommodate different architecture types accessing the same file system location. You can also use curly braces to delimit the name of the variable from appended letters or digits. Table 5-2 shows the predefined map variables.

Table 5-2 Predefined Map Variables

Variable	Meaning	Derived From	Example
`ARCH`	Architecture type	`uname -m`	`sun4c`
`CPU`	Processor Type	`uname -p`	`sparc`
`HOST`	Host name	`uname -n`	`dinky`
`OSNAME`	Operating system name	`uname -s`	`SunOS`
`OSREL`	Operating system release	`uname -r`	`5.4`
`OSVERS`	Operating system version (version of the release)	`uname -v`	`FCS1.0`

You can use variables anywhere in an entry line except as a *key*. For instance, if you have a file server exporting binaries for SPARC and x86 architectures from `/usr/local/bin/sparc` and `/usr/local/bin/x86`, respectively, you can have the clients mount through a map entry like the following:

```
/usr/local/bin   -ro      server:/usr/local/bin/$CPU
```

Now the same entry on all the clients applies for all architectures.

Maps That Refer to Other Maps

A map entry +*mapname* used in a file map causes `automount` to read the specified map as if it were included in the current file. If *mapname* is not preceded by a slash, then autofs treats the map name as a string of characters and uses the name service switch policy to find it. If the path name is an absolute path name, then `automount` looks for a local map of that name. If the map name starts with a dash (`-`), `automount` consults the appropriate built-in map.

This name service switch file contains an entry for autofs, which contains the order in which the name services are searched. The file below is an example of a name service switch file:

```
#
# /etc/nsswitch.nis:
#
# An example file that could be copied over to /etc/nsswitch.conf;
# it uses NIS (YP) in conjunction with files.
#
# "hosts:" and "services:" in this file are used only if the /etc/netconfig
# file contains "switch.so" as a nametoaddr library for "inet" transports.
# the following two lines obviate the "+" entry in /etc/passwd and /etc/group.
passwd:         files nis
group:          files nis

# consult /etc "files" only if nis is down.
hosts:          nis [NOTFOUND=return] files
networks:       nis [NOTFOUND=return] files
protocols:      nis [NOTFOUND=return] files
rpc:            nis [NOTFOUND=return] files
ethers:         nis [NOTFOUND=return] files
netmasks:       nis [NOTFOUND=return] files
bootparams:     nis [NOTFOUND=return] files
publickey:      nis [NOTFOUND=return] files
netgroup:       nis
automount:      files nis
aliases:        files nis
# for efficient getservbyname() avoid nis
services:       files nis
```

For example, you can have a few entries in your local `/etc/auto_home` map for the most commonly accessed home directories, and use the switch to fall back to the NIS map for other entries.

```
bill                            cs.csc.edu:/export/home/&
bonny                           cs.csc.edu:/export/home/&
```

Note – If your `/etc/auto_home` or other local maps have the execute bit set, then `autofs` tries to execute it to obtain a map entry, instead of reading it. autofs logs errors to the console when the map is accessed.

To fix the problem, reset the execute bit:

```
# chmod -x /etc/auto_home
```

After consulting the included map, `automount` continues scanning the current map if no match is found. This means you can add more entries after a + entry.

```
bill                        cs.csc.edu:/export/home/&
bonny                       cs.csc.edu:/export/home/&
+auto_home
*           -nosuid         &:/export/home/&
```

The map included can be a local file (remember, only local files can contain + entries) or a built-in map:

```
+auto_home_finance          # NIS+ map
+auto_home_sales            # NIS+ map
+auto_home_engineering      # NIS+ map
+/etc/auto_mystuff          # local map
+auto_home                  # NIS+ map
+-hosts                     # built-in hosts map
```

The wildcard means a match is found. Therefore, the wildcard should be the last entry in all cases, because autofs does not continue consulting the map after finding a wildcard.

Note – + entries cannot be used in NIS+ or NIS maps.

Modifying How Autofs Navigates the Network (Modifying Maps)

You can modify, delete, or add entries to maps to meet the needs of your environment. As applications and other file systems that users require change their location, the maps must reflect those changes. You can modify autofs maps at any time. Whether your modifications take effect the next time `automountd` mounts a file system depends on which map you modify and what kind of modification you make.

Administrative Tasks Involving Maps

Listed below are the different administrative tasks you may need to perform involving maps in order to change your autofs environment.

- ”How to Modify Indirect Maps“ on page 83
- ”How to Modify Direct Maps“ on page 83
- ”How to Modify the Master Map“ on page 82
- ”Avoiding Mount-Point Conflicts“ on page 84

Table 5-3 describes the types of maps and their uses.

Table 5-3 Types of Maps and Their Uses

Type of Map	Use
Master	Associates a directory with a map
Direct	Directs autofs to specific file systems
Indirect	Directs autofs to reference-oriented file systems

Table 5-4 describes how to make changes to your autofs environment based on your name service.

Table 5-4 Map Maintenance

Name Service	Method
Local files	text editor
NIS	`make` files
NIS+	`nistbladm`

Table 5-5 tells you when to run the `automount` command depending on the modification you have made to the type of map. For example, if you've made an addition or a deletion to a direct map, you need to run the `automount` command to allow the change take effect; however, if you've modified an existing entry, you do not need to run autofs to make the change take effect.

Table 5-5 When to Run the `automount` Command

Type of Map	Restart `automount`?	
	Addition or Deletion	Modification
`auto_home`	N	N
`auto_master`	Y	Y
`direct`	Y	N
`indirect`	N	N

Modifying the Maps

The following procedures assume that you are using NIS+ as your name service.

▼ How to Modify the Master Map

1. **Using the `nistbladm` command, make the changes you want to the master map.**
   See *NIS+ and FNS Administration Guide.*
2. **For each client, become superuser by typing `su` at a prompt and then your superuser password.**

3. **For each client, run the** `automount` **command to ensure the changes you made take effect.**

4. **Notify your users of the changes.**
   Notification is required so that the users can also run the `automount` command as superuser on their own computers.

The `automount` command consults the master map whenever it is run.

▼ How to Modify Indirect Maps

♦ **Using the** `nistbladm` **command, make the changes you want to the indirect map.**
See *NIS+ and FNS Administration Guide.*

The change takes effect the next time the map is used, which is the next time a mount is done.

▼ How to Modify Direct Maps

1. **Using the** `nistbladm` **command, add or delete the changes you want to the direct map.**
   See *NIS+ and FNS Administration Guide.*

2. **If you added or deleted a mount point entry in Step 1, run the** `automount` **command.**

3. **Notify your users of the changes.**
   Notification is required so that the users can also run the `automount` command as superuser on their own computers.

Note – If you simply modify or change the contents of an existing direct map entry, you do not need to run the `automount` command.

For example, suppose you modify the `auto_direct` map so that the `/usr/src` directory is now mounted from a different server. If `/usr/src` is not mounted at this time, the new entry takes effect immediately when you try to access `/usr/src`. If `/usr/src` is mounted now, you can wait until the auto-unmounting takes place, and then access it. If this is not satisfactory, you

can unmount with the `umount` command and then access `/usr/src`. The mounting will now be done from the new server. If you deleted the entry, you would have to run the `automount` command for the deletion to take effect.

Note – Because of the additional steps, and because they do not take up as much room in the mount table as direct maps, use indirect maps whenever possible. They are easier to construct, and less demanding on the computers' file systems.

Avoiding Mount-Point Conflicts

If you have a local disk partition mounted on `/src` and you also want to use autofs to mount other source directories, you may encounter a problem. If you specify the mount point `/src`, then autofs hides the local partition whenever you try to reach it.

You need to mount the partition somewhere else; for example, on `/export/src`. You would the need an entry in `/etc/vfstab` like

```
/dev/dsk/d0t3d0s5 /dev/rdsk/c0t3d0s5 /export/src ufs 3 yes -
```

and this entry in `auto_src`

```
terra   terra:/export/src
```

where `terra` is the name of the computer.

Default Autofs Behavior

Booting invokes autofs using the `/etc/init.d/autofs` script and looks for the master `auto_master` map (subject to the rules discussed below).

Autofs uses the name service specified in the automount entry of the `/etc/nsswitch.conf` file. If NIS+ is specified, as opposed to local files or NIS, all map names are used as is. If NIS is selected and autofs cannot find a

map that it needs, but finds a map that contains one or more underscores, the underscores are changed to dots. Then autofs looks up the map again, as shown in Figure 5-4.

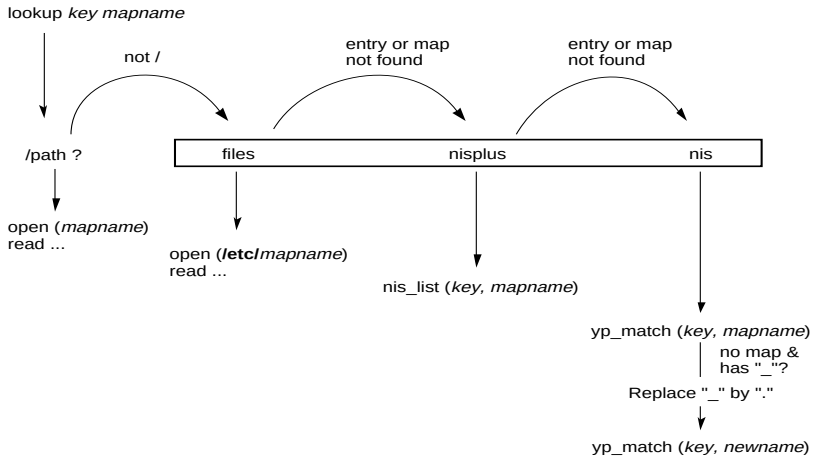


Figure 5-4 How Autofs Uses the Name Service

The screen activity would look like the following example.

```
$ more /etc/auto_master
# Master map for autofs
#
+auto_master
/net            -hosts          -nosuid
/home           auto_home

$ ypmatch brent auto_home
Can't match key brent in map auto_home.  Reason: no such map in
server's domain.

$ ypmatch brent auto.home
diskus:/export/home/diskus1/&
$
```

If “files” is selected as the name service, all maps are assumed to be local files in the `/etc` directory. Autofs interprets a map name that begins with a slash (/) as local, regardless of which name service it uses.

Autofs Reference

The rest of this chapter describes more advanced autofs features and topics.

Metacharacters

Autofs recognizes some characters as having a special meaning. Some are used for substitutions, some to protect other characters from the autofs map parser.

Ampersand (&)

If you have a map with many subdirectories specified, as in the following, consider using string substitutions.

```
john                    willow:/home/john
mary                    willow:/home/mary
joe                     willow:/home/joe
able                    pine:/export/able
baker                   peach:/export/baker
```

You can use the ampersand character (&) to substitute the key wherever it appears. If you use the ampersand, the map above changes to:

```
john                    willow:/home/&
mary                    willow:/home/&
joe                     willow:/home/&
able                    pine:/export/&
baker                   peach:/export/&
```

You could also use key substitutions in a direct map, in situations like this:

```
/usr/man                willow,cedar,poplar:/usr/man
```

which you can also write as:

```
/usr/man              willow,cedar,poplar:&
```

Notice that the ampersand substitution uses the whole key string, so if the key in a direct map starts with a / (as it should), the slash is carried over, and you could not do, for example, the following:

```
/progs          &1,&2,&3:/export/src/progs
```

because autofs would interpret it as:

```
/progs          /progs1,/progs2,/progs3:/export/src/progs
```

Asterisk (*)

The catchall substitute character, the asterisk (*), can be used to match any key. The `/export` file system from all hosts could be mounted through this map entry.

```
*                       &:/export
```

Each ampersand is substituted by the value of any given key. Autofs interprets the asterisk as an end-of-file character.

Special Characters

If you have a map entry that contains special characters, you may have to mount directories whose names confuse the autofs map parser. The autofs parser is sensitive to names containing colons, commas, spaces, and so on. These names should be enclosed in double quotations, as in the following:

```
/vms    -ro     vmsserver:"rc0:dk1"
/mac    -ro     gator:/"Mr Disk"
```

Accessing Non-NFS File Systems

Autofs can also mount files other than NFS files. Autofs mounts files on removable media, such as diskettes or CD-ROM. Normally, you would mount files on removable media using the Volume Manager. The examples below are given to show how this mounting could be done through autofs. The Volume Manager and autofs do not work together, so normally these entries would not be used without first deactivating the Volume Manager.

Instead of mounting a file system from a server, you put the media in the drive and reference it from the map. If you want to access non-NFS file systems and you are using autofs, see the following procedures. For more information about Volume Manager, see *System Administration Guide, Volume I.*

▼ How to Access CD-ROM Applications

Note – Use this procedure if you are *not* using Volume Manager.

♦ **Specify the CD-ROM file system type as follows:**

```
hsfs     -fstype=hsfs,ro     :/dev/sr0
```

The CD-ROM device you wish to mount must appear as a name following a colon.

▼ How to Access Diskettes Containing Data in PC-DOS Files

Note – Use this procedure if you are *not* using Volume Manager.

♦ **Specify the diskette file system type as follows:**

```
pcfs     -fstype=pcfs     :/dev/diskette
```

Accessing NFS File Systems Using CacheFS

The cache file system (CacheFS) is a generic nonvolatile caching mechanism that improves the performance of certain file systems by utilizing a small, fast, local disk.

You can improve the performance of the NFS environment by using CacheFS to cache data from an NFS file system on a local disk.

▼ How to Start CacheFS

♦ **Run the `cfsadmin` command to create a cache directory on the local disk.**

```
# cfsadmin -c /var/cache
```

Add this line to the master map to cache home directories:

```
/home auto_home -fstype=cachefs,cachedir=/var/cache,backfstype=nfs
```

Note – Options that are included in maps that are searched later will override options that are set in maps that are searched earlier. The last options that are found are the ones that are used. In the example above, a specific entry added to the `auto_home` map would need to include the options listed in the master maps.

Common Tasks and Procedures

This section describes some of the most common tasks you may encounter in your own environment. Recommended procedures are included for each scenario to help you configure autofs to best meet your clients' needs.

Note – Use the Solstice System Management Tools or see *NIS+ and FNS Administration Guide* to perform the tasks discussed in this section.

5

How to Set Up Different Architectures to Access a Shared Name Space

You need to assemble a shared name space for local executables, and applications, such as spreadsheet tools and word-processing packages. The clients of this name space use several different workstation architectures that require different executable formats. Also, some workstations are running different releases of the operating system.

1. **Create the `auto_local` map with the `nistbladm` command.**
   See the *NIS+ and FNS Administration Guide*.

2. **Choose a single, site-specific name for the shared name space so that files and directories that belong to this space are easily identifiable.**

   For example, if you choose `/usr/local` as the name, then the path `/usr/local/bin` is obviously a part of this name space.

3. **For ease of user community recognition, create an autofs indirect map and mount it at `/usr/local`. Set up the following entry in the NIS+ (or NIS) `auto_master` map:**

```
/usr/local     auto_local     -ro
```

   Note that the `ro` mount option implies that clients will not be able to write to any files or directories.

4. **Export the appropriate directory on the server.**

5. **Include a `bin` entry in the map.**
   Your directory structure looks like this:

```
bin     aa:/export/local/bin
```

   To satisfy the need to serve clients of different architectures, you need references to the `bin` directory to be directed to different directories on the server, depending on the clients' architecture type.

6. **To serve clients of different architectures, change the entry by adding the autofs `CPU` variable.**

```
bin     aa:/export/local/bin/$CPU
```

Note – For SPARC clients, make executables available under `/export/local/bin/sparc` on the server. For x86 clients, use `/export/local/bin/i386`.

▼ How to Support Incompatible Client Operating System Versions

1. **Combine the architecture type with a variable that determines the operating system type of the client.**
   The autofs `OSREL` variable can be combined with the `CPU` variable to form a name that determines both CPU type and OS release.

2. **Create the following map entry.**

```
bin     aa:/export/local/bin/$CPU$OSREL
```

For SPARC clients running version 5.1 of the operating system, you need to export `/export/local/bin/sparc5.1` from the server and similarly export for other releases. Since operating systems attempt to preserve backward compatibility with executable formats, assume that the OS release is not a factor and eliminate it from future examples.

So far, you have set up an entry for a single server `aa`. In a large network, you want to replicate these shared files across several servers. Each server should have a close network proximity to the clients it serves so that NFS traffic is confined to local network segments.

▼ How to Replicate Shared Files Across Several Servers

♦ **Modify the entry to create the list of all replica servers as a comma-separated list:**

```
bin     aa,bb,cc,dd:/export/local/bin/$CPU
```

Autofs chooses the nearest server. If a server has several network interfaces, then list each interface. Autofs chooses the nearest interface to the client, avoiding unnecessary routing of NFS traffic.

```
bin     aa,bb-68,bb-72,dd:/export/local/bin/$CPU
```

Several shared files may not have an architecture dependency. A good example of this is shell scripts. You can locate these shared files under `/usr/local/share` with an independent map entry like this:

```
share     aa,bb-68,bb-72,dd:/export/local/share
```

To ensure that scripts refer to local executables, use architecture-independent paths either fully qualified or relative (for example, `/usr/local/bin/frotz` or `../bin/frotz`).

Similarly, other applications may have their own wrapper scripts for handling client dependencies. You can also set up these scripts with their own map entries.

```
frame   pp,qq:/export/local/frame/3.0
valid   pp,rr,tt:/export/local/valid
lotus   pp,qq,zz:/export/local/lotus
```

The servers can use the same `/usr/local` map as the clients. Users who work on the server will see the same shared name space under `/usr/local`.

If the server's autofs notices that a directory under `/usr/local` is available on the server under `/export/local`, it will loopback mount the directory so that it appears under `/usr/local`. Servers must not mount local disks on or under `/usr/local`.

▼ How to Set Up a Common View of /home Directory Structure

You would like every user in the network to be able to locate their own, or anyone else's home directory under /home. This view should be common across all computers, whether client or server.

Every Solaris installation comes with a pre-installed master map: /etc/auto_master.

```
# Master map for autofs
#
+auto_master
/net        -hosts      -nosuid
/home       auto_home
```

A map for auto_home is also preinstalled under /etc.

```
# Home directory map for autofs
# +auto_home
```

Except for a reference to an external auto_home map, this map is empty. If the directories under /home are to be common to all computers, then do not modify this /etc/auto_home map. All home directory entries should appear in the name service files, either NIS or NIS+.

Users should not be permitted to run setuid executables from their home directories because without a restriction any user could have superuser privileges on any computer.

▼ How to Apply Security Restrictions

♦ **Create the following entry in the name service auto_master file, either NIS or NIS+:**

```
    /home      auto_home      -nosuid
```

This entry overrides the entry for /home in the local /etc/auto_master file (see the previous example) because the +auto_master reference to the external name service map occurs before the /home entry in the file.

Note – Do not mount the home directory disk partitions on or under `/home` on the server.

▼ How to Set Up Home Directory Servers

1. **Mount home directory partitions under `/export/home`.**
   This directory is reserved for autofs.

   If there are several partitions, mount them under separate directories, for example, `/export/home1`, `/export/home2`, and so on.

2. **Use the Solstice System Management Tools to create and maintain the `auto_home` map.**
   Whenever you create a new user account, type the location of the user's home directory in the `auto_home` map. Map entries can be simple, for example:

```
rusty       dragon:/export/home1/&
gwenda      dragon:/export/home1/&
charles     sundog:/export/home2/&
rich        dragon:/export/home3/&
```

Note the use of the & (ampersand) to substitute the map key. This is an abbreviation for the second occurrence of `rusty` in the following example.

```
rusty      dragon:/export/home1/rusty
```

With the `auto_home` map in place, users can refer to any home directory (including their own) with the path `/home/`*user* where *user* is their login name. This common view of all home directories is valuable when logging into another user's computer. Autofs there mounts your home directory for you. Similarly if you run a remote windowing system client on another computer, the client program has the same view of the `/home` directory as you do on the computer providing the windowing system display.

This common view also extends to the server. Using the previous example, if `rusty` logs into the server `dragon`, autofs there provides direct access to the local disk by loopback mounting `/export/home1/rusty` onto `/home/rusty`.

Users do not need to be aware of the real location of their home directories. If `rusty` needs more disk space and needs to have his home directory relocated to another server, only `rusty`'s entry in the `auto_home` map needs to be changed to reflect the new location. Everyone else can continue to use the `/home/rusty` path.

▼ How to Consolidate Project-Related Files Under `/ws`

You are the administrator of a large software development project. You want to make all project-related files available under a directory called `/ws`. This directory is to be common across all workstations at the site.

1. **Add an entry for the `/ws` directory to the site `auto_master` map, either NIS or NIS+.**

```
/ws     auto_ws     -nosuid
```

The contents of the `/ws` directory are determined by the `auto_ws` map.

2. **Add the `-nosuid` option as a precaution.**
   This option prevents users from running `setuid` programs that may exist in any workspaces.

The `auto_ws` map is organized so that each entry describes a subproject. Your first attempt yields a map that looks like the following:

```
compiler    alpha:/export/ws/&
windows     alpha:/export/ws/&
files       bravo:/export/ws/&
drivers     alpha:/export/ws/&
man         bravo:/export/ws/&
tools       delta:/export/ws/&
```

The ampersand (&) at the end of each entry is just an abbreviation for the entry key. For instance, the first entry is equivalent to:

```
compiler     alpha:/export/ws/compiler
```

This first attempt provides a map that looks simple, but it turns out to be inadequate. The project organizer decides that the documentation in the `man` entry should be provided as a subdirectory under each subproject. Also, each subproject requires subdirectories to describe several versions of the software. Each of these subdirectories must be assigned to an entire disk partition on the server.

Modify the entries in the map as follows:

```
compiler \
    /vers1.0          alpha:/export/ws/&/vers1.0 \
    /vers2.0          bravo:/export/ws/&/vers2.0 \
    /man              bravo:/export/ws/&/man
windows \
    /vers1.0          alpha:/export/ws/&/vers1.0 \
    /man              bravo:/export/ws/&/man
files \
    /vers1.0          alpha:/export/ws/&/vers1.0 \
    /vers2.0          bravo:/export/ws/&/vers2.0 \
    /vers3.0          bravo:/export/ws/&/vers3.0 \
    /man              bravo:/export/ws/&/man
drivers \
    /vers1.0          alpha:/export/ws/&/vers1.0 \
    /man              bravo:/export/ws/&/man
tools \
    /                 delta:/export/ws/&
```

Although the map now appears to be much bigger, it still contains only the five entries. Each entry is larger because it contains multiple mounts. For instance, a reference to `/ws/compiler` requires three mounts for the `vers1.0`, `vers2.0`, and `man` directories. The backslash at the end of each line tells autofs that the entry is continued onto the next line. In effect, the entry is one long line, though line breaks and some indenting have been used to make it more readable. The `tools` directory contains software development tools for all subprojects, so it is not subject to the same subdirectory structure. The `tools` directory continues to be a single mount.

This arrangement provides the administrator with much flexibility. Software projects are notorious for consuming large amounts of disk space. Through the life of the project you may be required to relocate and expand various disk partitions. As long as these changes are reflected in the `auto_ws` map, the users do not need to be notified since the directory hierarchy under `/ws` is not changed.

Since the servers `alpha` and `bravo` view the same autofs map, any users who log into these computers will find the `/ws` name space as expected. These users will be provided with direct access to local files through loopback mounts instead of NFS mounts.

Troubleshooting Autofs

Occasionally, you may encounter problems with autofs. This section should make the problem-solving process easier. It is divided into two subsections.

This section presents a list of the error messages autofs generates. The list is divided in two parts:

- Error messages generated by the verbose (`-v`) option of `automount`
- Error messages that may appear at any time

Each error message is followed by a description and probable cause of the message.

When troubleshooting, start the autofs programs with the verbose (`-v`) option, otherwise you may experience problems without knowing why.

The following paragraphs are labeled with the error message you are likely to see if autofs fails, and a description of what the problem may be.

Error Messages Generated by `automount -v`

`bad key` *`key`* `in direct map` *`mapname`*

While scanning a direct map, autofs has found an entry key without a prefixed `/`. Keys in direct maps must be full path names.

`bad key` *`key`* `in indirect map` *`mapname`*

While scanning an indirect map autofs has found an entry key containing a `/`. Indirect map keys must be simple names—not path names.

`can't mount` *server*`:`*pathname: reason*

The mount daemon on the server refuses to provide a file handle for *server*:*pathname*. Check the export table on *server*.

`couldn't create mount point` *mountpoint*: *reason*

Autofs was unable to create a mount point required for a mount. This most frequently occurs when attempting to hierarchically mount all of a server's exported file systems. A required mount point may exist only in a file system that cannot be mounted (it may not be exported) and it cannot be created because the exported parent file system is exported read-only.

`leading space in map entry` *entry* `text in` *mapname*

Autofs has discovered an entry in an automount map that contains leading spaces. This is usually an indication of an improperly continued map entry, for example:

```
fake
/blat   frobz:/usr/frotz
```

In this example, the warning is generated when autofs encounters the second line because the first line should be terminated with a backslash (\).

mapname`: Not found`

The required map cannot be located. This message is produced only when the `-v` option is used. Check the spelling and path name of the map name.

`remount` *server*`:`*pathname* `on` *mountpoint*`: server not responding`

Autofs has failed to remount a file system it previously unmounted.

`WARNING:` *mountpoint* `already mounted on`

Autofs is attempting to mount over an existing mount point. This means there is an internal error in autofs (an anomaly).

Miscellaneous Error Messages

`dir` *mountpoint* `must start with '/'`

Automounter mount point must be given as full path name. Check the spelling and path name of the mount point.

`hierarchical mountpoints:` *pathname1* `and` *pathname2*

Autofs does not allow its mount points to have a hierarchical relationship. An autofs mount point must not be contained within another automounted file system.

`host` *server* `not responding`

Autofs attempted to contact but received no response.

hostname`: exports:` *rpc_err*

Error getting export list from *hostname*. This indicates a server or network problem.

`map` *mapname*`, key` *key*`: bad`

The map entry is malformed, and autofs cannot interpret it. Recheck the entry; perhaps there are characters in it that need escaping.

mapname`:` *nis_err*

Error in looking up an entry in an NIS map. This may indicate NIS problems.

`mount of` *server*`:`*pathname* `on` *mountpoint:reason*

Autofs failed to do a mount. This may indicate a server or network problem.

mountpoint`: Not a directory`

Autofs cannot mount itself on *mountpoint* because it's not a directory. Check the spelling and path name of the mount point.

`nfscast: cannot send packet:` *reason*

Autofs cannot send a query packet to a server in a list of replicated file system locations.

`nfscast: cannot receive reply:` *reason*

Autofs cannot receive replies from any of the servers in a list of replicated file system locations.

`nfscast:select:` *reason*

All these error messages indicate problems attempting to `ping` servers for a replicated file system. This may indicate a network problem.

`pathconf: no info for` *server*`:`*pathname*

Autofs failed to get pathconf information for *pathname* (see the `fpathconf(2)` man page).

`pathconf:` *server*`: server not responding`

Autofs is unable to contact the mount daemon on *server* that provides the information to `pathconf()`.

Other Errors with Autofs

If the /etc/auto* files have the execute bit set, then the automounter will try to execute the maps, which will create messages like:

```
/etc/auto_home: +auto_home: not found
```

In this case, the auto_home file has incorrect permissions. Each entry in the file will generate an error message much like this one. The permissions to the file should be reset by typing the following command:

```
# chmod 644 /etc/auto_home
```

NFS Tunables A

Several parameters can be set which can improve the functioning of the NFS service. These parameters can be defined in `/etc/system`, which is read during the boot process. Each parameter can be identified by the name of the kernel module that it is in and a symbol name which identifies it.

Warning – The names of the symbols, the modules that they are resident in, and the default values can change between releases. Check the documentation for the version of SunOS that you are running, before making changes or applying values from previous releases.

Table A-1 on page 102 lists the parameters that are part of the nfs module. Table A-2 on page 104 lists the parameters that are part of the nfssrv module. “How to Set the Value of a Kernel Parameter” on page 105 shows how to change these parameters. See the `system(4)` man page for information about the `/etc/system` file.

A

.

Table A-1 NFS Parameters for the nfs Module

Symbol Name	Description	Default Setting
authdes_win	This symbol controls how much clock skew will be allowed between the server and clients when using AUTH_DES.	Defaults to 300 seconds
authkerb_win	This symbol controls how much clock skew will be allowed between the server and clients when using AUTH_KERB.	Defaults to 300 seconds
nfs_acl_cache	This symbol controls whether ACLs are cached on clients which are using the NFS_ACL protocol.	Defaults to off (0). This can probably be safely enabled (1) and probably will be in the next release of Solaris.
nfs_do_symlink_cache	This symbol controls whether symbolic links are cached for file systems mounted using NFS Version 2.	Defaults to on (1). This may be disabled (0) if something like amd is to be used on the system. Client system performance may be reduced if this is disabled.
nfs3_do_symlink_cache	This symbol controls whether symbolic links are cached for file systems mounted using NFS Version 3.	Defaults to on (1). This may be disabled (0) but client system performance may be reduced.
nfs_dynamic	This symbol controls whether dynamic retransmission support is used for file systems mounted using NFS Version 2.	Defaults to on (1). It can be turned off (0) safely, with possible interoperability problems with servers which are slow or can not support full 8k read or write transfers.
nfs3_dynamic	This symbol controls whether dynamic retransmission support is used for file systems mounted using NFS Version 3.	Defaults to off (0). Do not change this.
nfs_lookup_neg_cache	This symbol controls whether failed lookup requests are cached for file systems mounted using NFS Version 2.	Defaults to off (0). This can probably be safely enabled (1) but may negatively impact normal directory name caching.
nfs3_lookup_neg_cache	This symbol controls whether failed lookup requests are cached for file systems mounted using NFS Version 3.	Defaults to off (0). This can probably be safely enabled (1) but may negatively impact normal directory name caching.

Table A-1 NFS Parameters for the nfs Module

Symbol Name	Description	Default Setting
nfs_max_threads	This symbol controls the maximum number of async threads started per file system mounted using NFS Version 2.	Defaults to 8. Since this number effects the number of threads per file system, on a clients with many file systems a large change could severely degrade performance.
nfs3_max_threads	This symbol controls the maximum number of async threads started per file system mounted using NFS Version 3.	Defaults to 8. Since this number effects the number of threads per file system, on a client with many file systems a large change could several degrade performance
nfs3_max_transfer_size	This symbol controls the NFS Version 3 client file blocksize.	Defaults to 32k bytes. Strongly recommend that it not be changed.
nfs_nra	This symbol controls the number of readahead blocks that are read for file systems mounted using NFS Version 2.	Defaults to 1. 4 is actually a much better value, but does result in increased memory utilization on the client.
nfs3_nra	This symbol controls the number of readahead blocks that are read for file systems mounted using NFS Version 3.	Defaults to 1. 2 is actually a much better value, but does result in increased memory utilization on the client.
nrnode	This symbol controls the number of NFS rnodes that are cached.	The value assigned to this symbol is configured at boot time and scales to match the server. This can be set to 1 to disable caching.

A

Table A-1 NFS Parameters for the nfs Module

Symbol Name	Description	Default Setting
`nfs_shrinkreaddir`	This symbol controls whether over the wire NFS Version 2 READDIR requests are shrunk to 1024 bytes. Certain very old NFS Version 2 servers could not correctly handle READDIR requests larger than 1024 bytes.	Defaults to off (0), which means to not reduce the READDIR requests. This can be safely enabled (1) but may negatively impact performance while reading directories.
nfs_write_error_interval	This symbol controls how often NFS ENOSPC write error messages are logged. Its units are in seconds.	Defaults to 5.
nfs_write_error_to_cons_only	This symbol controls whether NFS write error messages are logged to the system console or to the system console and syslog.	Defaults to off (0), which means to log all NFS write error messages to the system console and syslog. Enabling (1) this functionality means that most NFS write error messages will only printed on the system console.

Table A-2 NFS Parameters for the nfssrv Module

Symbol Name	Description	Default Setting
nfs_portmon	This symbol controls whether the NFS server will do filtering of requests based on the IP port number. It uses the Berkeley notion of reserved port numbers.	Defaults to off (0). It can be enabled (1), but problems with interoperability may appear.
nfsreadmap	This symbol is no longer active. Map reads are no longer implemented. It is left to ease transitions.	Defaults to off (0).
rfs_write_async	This symbol controls whether the NFS Version 2 server will use write clustering in order to safely increase write throughput.	Defaults to on (1). It can be disabled (0), but performance may be reduced.

▼ How to Set the Value of a Kernel Parameter

1. **Become root.**

2. **Edit the /etc/system file and add a line to set the parameter.**
   Each entry should follow this form:

   set *module*:*symbol*=*value*

   where *module* is the name of the kernel module which contains the required parameter, *symbol* is the name of the parameter, and *value* is the numerical value to assign to the parameter. For example:

   ```
   set nfs:nfs_nra=4
   ```

   would change the number of readahead blocks that are read for file systems mounted using NFS Version 2.

3. **Reboot the system.**

A

Index

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z