NFS Administration Guide

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303
U.S.A.

Part No: 805–3479–10
October 1998

Contents

Preface

The *NFS Administration Guide* presents the administrative tasks required for the successful operation of the NFS™ distributed file system. This resource-sharing product enables 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 are new to the Solaris™ platform.

How This Book Is Organized

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

Chapter 2 provides information on how to set up NFS servers with NIS or NIS+ as your name service.

Chapter 3 presents background information on the security features of the NFS service as well as fundamental procedures for setting up and maintaining NFS security.

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

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

Appendix A lists several parameters that you can change to improve the NFS service. It includes instructions for making these changes.

Related Books

This is a list of related documentation that is refered to in this book.

- *Solaris Naming Administration Guide*
- *Solaris Naming Setup and Configuration Guide*
- *System Administration Guide, Volume I*
- *System Administration Guide, Volume II*
- *TCP/IP and Data Communications Administration Guide*

Ordering Sun Documents

The SunDocs℠ program provides more than 250 manuals from Sun Microsystems, Inc. If you live in the United States, Canada, Europe, or Japan, you can purchase documentation sets or individual manuals using this program.

For a list of documents and how to order them, see the catalog section of SunExpress™ Internet site at `http://www.sun.com/sunexpress`.

What Typographic Changes Mean

Table P–1 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.`

`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

Table P–2 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	`#`

PART I Introduction

This part provides an overview of the services provided in the NFS environment.

CHAPTER **1**

Solaris NFS Environment

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 environment.

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. The NFS service 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 can 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.

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

NFS File Systems

The objects that can be shared with the NFS service 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 support works across operating systems, and the concept of a file system might be meaningless in other, non-UNIX environments. Therefore, the term *file system* used throughout this guide refers to a file or file hierarchy that can be shared and mounted over the NFS environment.

About the NFS Environment

The NFS service enables computers of different architectures running different operating systems to share file systems across a network. NFS support 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 the NFS service are that it:

- Allows multiple computers to use the same files, so everyone on the network can access the same data
- 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 service makes the physical location of the file system irrelevant to the user. You can use the NFS implementation 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 service enables 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 almost indistinguishable from local ones.

NFS Version 2

Version 2 was the first version of the NFS protocol in wide use. It continues to be available on a large variety of platforms. Solaris releases prior to Solaris 2.5 support version 2 of the NFS protocol.

NFS Version 3

An implementation of NFS version 3 protocol was a new feature of the Solaris 2.5 release. Several changes have been made to improve interoperability and to improve performance. To take advantage of these improvements, the version 3 protocol must be running on both 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 NFS version 3 operations return the file attributes, which are stored in the local cache. Because the cache is updated more often, the need to do a separate operation to update this data arises less often. Therefore, the number of RPC calls to the server is reduced, improving performance.

The process for verifying file access permissions has been improved. In particular, version 2 would generate a message reporting a "write error" or a "read error" 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."

The NFS version 3 implementation removes the 8-Kbyte transfer size limit. Clients and servers 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 was added in the Solaris 2.5 release. ACLs provide a finer-grained mechanism to set file access permissions than is available through standard UNIX file permissions. NFS ACL support provides a method of changing and viewing ACL entries from a Solaris NFS client to a Solaris NFS server.

NFS Over TCP

The default transport protocol for the NFS protocol was changed to the transport control protocol (TCP) in the Solaris 2.5 release, which helps performance on slow networks and wide-area networks. TCP provides congestion control and error recovery. NFS over TCP works with version 2 and version 3. Prior to 2.5, the default NFS protocol was user datagram protocol (UDP).

Network Lock Manager

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

NFS Large File Support

The Solaris 2.6 release of the NFS version 3 protocol was changed to correctly manipulate files larger than 2 Gbytes. The NFS version 2 protocol and the Solaris 2.5 implementation of the version 3 protocol cannot handle files larger than 2 Gbytes.

NFS Client Failover

Dynamic failover of read-only file systems was added in the Solaris 2.6 release. It provides a high level of availability for read-only resources that are already replicated, such as man pages, AnswerBook™ documentation, and shared binaries. Failover can occur anytime after the file system is mounted. Manual mounts can now list multiple replicas, much like the automounter allowed in previous releases. The automounter has not changed, except that failover need not wait until the file system is remounted.

Kerberos Support for the NFS Environment

Support for Kerberos V4 clients was included in the Solaris 2.0 release. In release 2.6, the `mount` and `share` commands were altered to support NFS mounts using Kerberos V5 authentication. Also, the `share` command was changed to allow for multiple authentication flavors to different clients.

WebNFS Support

The Solaris 2.6 release also included the ability to make a file system on the Internet accessible through firewalls, using an extension to the NFS protocol. One of the advantages to using the WebNFS™ protocol for Internet access is that the service is built as an extension of the NFS version 3 and version 2 protocol, which is very reliable. Soon, applications will be written to utilize this new file system access protocol. Also, an NFS server provides greater throughput under a heavy load than HyperText Transfer Protocol (HTTP) access to a Web server. This can decrease the amount of time required to retrieve a file. In addition, the WebNFS implementation provides the ability to share these files without the administrative overhead of an anonymous `ftp` site.

RPCSEC_GSS Security Flavor

A new security flavor, called RPCSEC_GSS, is supported in the Solaris 7 release. This flavor uses the standard GSS-API interfaces to provide authentication, integrity and privacy, as well as allowing for support of multiple security mechanisms. Currently, the mechanisms to use these new security flavors are not integrated into the Solaris release.

Solaris 7 Extensions for NFS Mounting

Included in the Solaris 7 release are extensions to the `mount` and `automountd` command which allow for the mount request to use the public file handle instead of the MOUNT protocol. This is the same access method that the WebNFS service uses. By circumventing the MOUNT protocol, the mount can occur through a firewall. In addition, because fewer transactions need to occur between the server and client, the mount should occur much faster.

The extensions also allow for NFS URLs to be used instead of the standard pathname. Also, the `–public` option can be used with the `mount` command and the automounter maps to force the use of the public file handle.

About Autofs

File systems shared through the NFS service can be mounted using automatic mounting. Autofs, a client-side service, is a file system structure that provides automatic mounting. The autofs file system is initialized by `automount`, which is run automatically when a system is booted. The automount daemon, `automountd`, runs continuously, 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.

Mounting need not be 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.

Chapter 5 gives more specific information about the autofs service.

Autofs Features

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

A fully multithreaded version of `automountd` was included in the Solaris 2.6 release. This enhancement makes autofs more reliable and allows for concurrent servicing of multiple mounts, which prevents the service from hanging if a server is unavailable.

The new `automountd` also provides better on-demand mounting. Previous releases would mount an entire set of file systems if they were hierarchically related. Now only the top file system is mounted. Other file systems related to this mountpoint are mounted when needed.

The autofs service now support browsability of indirect maps. This allows a user to see what directories could be mounted, without having to actually mount each one of the file systems. A –nobrowse option has been added to the autofs maps, so that large file systems, such as /net and /home, are not automatically browsable. Also, autofs browsability can be turned off on each client by using the –n option with automount.

PART II

All About NFS Services

This part of the manual describes most of the the NFS services and how to administer them. The next part covers autofs.

CHAPTER **2**

NFS Administration

This chapter provides information on how to perform such NFS administration tasks as setting up NFS services, adding new file systems to share, mounting file systems, using the Secure NFS system, or using the WebNFS functionality. The last part of the chapter includes troubleshooting procedures and a list of many of the NFS error messages and their meanings.

Your responsibilities as an NFS administrator depend on your site's requirements and the role of your computer on the network. You might be responsible for all the computers on your local network, in which case you might be responsible for determining these configuration items:

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

Maintaining a server after 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
- Setting up maps for autofs

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

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. Most file system sharing should be done automatically, the only time that manual sharing should occur is during testing or troubleshooting.

The `dfstab` file lists all the file systems that your server shares with its clients and controls which clients can 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, 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 type at the command-line prompt to share the file system. The `share` command is located in `/usr/sbin`.

▼ How to Set Up Automatic File-System Sharing

1. **Edit the `/etc/dfs/dfstab` file.**

   Add one entry to the file for each file system that you want to be automatically shared. 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. **Check that the NFS service 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 service 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 that clients can access the file systems you have shared on the server. See "Setting Up Autofs" on page 73.

Mounting File Systems

You can mount file systems in several ways. They can be mounted automatically when the system is booted, on demand from the command line, or through the automounter. The automounter provides many advantages to mounting at boot time or mounting from the command line, but many situations require a combination of all three.

▼ How to Mount at Boot Time

If you want to mount file systems at boot time instead of using autofs maps, follow this procedure. Although you must follow this procedure 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 from the server `wasp`. You would like the file system to be mounted as `/var/mail` on the client and 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
```

Caution - NFS servers should not have NFS `vfstab` entries because of a potential deadlock. The NFS service is started after the entries in `/etc/vfstab` are checked, so that if two servers that are mounting file systems from each other fail at the same time, each system could hang as the systems reboot.

▼ How to Mount From the Command Line

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

```
# mount -F nfs -o ro bee:/export/share/local /mnt
```

In this case, the `/export/share/local` file system from the server `bee` is mounted on read-only `/mnt` on the local system. Mounting from the command line allows for temporary viewing of the file system. You can unmount the file system with `umount` or by rebooting the local host.

Caution - Starting with the 2.6 release, all versions of the `mount` command will not warn about invalid options. The command silently ignores any options that cannot be interpreted. Make sure you verify all of the options that were used, to prevent unexpected behavior.

▼ How to Mount With the Automounter

Chapter 5 includes the specific instructions for establishing and supporting mounts with the automounter. Without any changes to the generic system, clients should be able to access remote file systems through the `/net` mount point. To mount the `/export/share/local` file system from the previous example, all you need to do is type:

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

Because 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 finished.

Setting Up NFS Services

This section discusses some of the tasks necessary to initialize or use NFS services.

▼ How to Start the NFS Services

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

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

This starts the daemons if there is an entry in `/etc/dfs/dfstab`.

▼ How to Stop the NFS Services

♦ **To disable daemons without rebooting, become superuser and type the following command.**

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

▼ How to Disable Large Files on an NFS Server

1. **Check to be sure no large files exist on the file system.**

   Here is an example of a command that you can run to locate large files:

```
# cd /export/home1
# find . -xdev -size +2000000 -exec ls -l {} \;
```

   If there are large files on the file system, you must remove or move them to another file system.

2. **Unmount the file system.**

```
# umount /export/home1
```

3. **Reset the file system state if the file system has been mounted using** `–largefiles`**.**

   `fsck` resets the file system state if no large files exist on the file system:

```
# fsck /export/home1
```

4. **Mount the file system using** `–nolargefiles`**.**

```
# mount -F ufs -o nolargefiles /export/home1
```

You can do this from the command line, but to make the option more permanent, add an entry like the following into `/etc/vfstab`:

```
/dev/dsk/c0t3d0s1 /dev/rdsk/c0t3d0s1 /export/home1  ufs  2  yes  nolargefiles
```

Note - Previous versions of the Solaris operating environment cannot use large files. Check to be sure clients of the NFS server are running at least version 2.6 if the clients need to access large files.

▼ How to Use Client-Side Failover

♦ **On the NFS client, mount the file system using the** `–ro` **option.**

You can do this from the command line, through the automounter, or by adding an entry to `/etc/vfstab` that looks like:

```
bee,wasp:/export/share/local  -  /usr/local  nfs  -  no  -o ro
```

This syntax has been allowed by the automounter in earlier releases, but the failover was not available while file systems were mounted, only when a server was being selected.

Note - Servers that are running different versions of the NFS protocol can not be mixed using a command line or in a vfstab entry. Mixing servers supporting NFS V2 and V3 protocols can only be done with autofs, in which case the best subset of version 2 or version 3 servers is used.

▼ How to Disable Mount Access for One Client

1. **Edit** `/etc/dfs/dfstab`**.**

The first example allows mount access to all clients in the `eng` netgroup except the host named `rose`. The second example allows mount access to all clients in the `eng.sun.com` DNS domain except for `rose`.

```
share -F nfs -o ro=-rose:eng /export/share/man
share -F nfs -o ro=-rose:.eng.sun.com /export/share/man
```

For additional information on access lists, see "Setting Access Lists With the `share` Command" on page 52.

2. **Run the `shareall` command.**

   The NFS server does not use changes to `/etc/dfs/dfstab` until the file systems are shared again or until the server is rebooted.

   ```
   # shareall
   ```

▼ How to Mount an NFS File System Through a Firewall

1. **Become superuser.**

2. **Manually mount the file system, using a command like:**

   ```
   # mount -F nfs -o public bee:/export/share/local /mnt
   ```

   In this example the file system `/export/share/local` is mounted on the local client using the public file handle. An NFS URL can be used instead of the standard pathname. If the public file handle is not supported by the server `bee`, the mount operation will fail.

Note - This procedure requires that the file system on the NFS server be shared using the public option and any firewalls between the client and the server allow TCP connections on port `2049`. Starting with the 2.6 release, all file systems that are shared allow for public file handle access.

▼ How to Mount an NFS File System Using an NFS URL

1. **Become superuser.**

2. **Manually mount the file system, using a command such as:**

```
# mount -F nfs nfs://bee:3000/export/share/local /mnt
```

In this example, the `/export/share/local` file system is being mounted from the server `bee` using NFS port number `3000`. The port number is not required and by default uses the standard NFS port number of `2049`. You can include the public option with an NFS URL, if you want. Without the public option, the MOUNT protocol is used if the public file handle is not supported by the server. The public option will force the use of the public file handle, and the mount will fail if the public file handle is not supported.

Administering the Secure NFS System

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

You can configure the Secure NFS environment to use either Diffie-Hellman or Kerberos Version 4 authentication or a combination of the two. “Managing System Security (Overview)” in *System Administration Guide, Volume II* discusses these authentication services.

▼ How to Set Up a Secure NFS Environment With DH Authentication

1. **Assign your domain a domain name, and make the domain name known to each computer in the domain.**

   See the *Solaris Naming 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. **To 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 the daemon isn't running, start the keyservet by typing 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 - You only need to run `keylogin -r` if the root secret key changes or `/etc/.rootkey` is lost.

6. **Edit the** `/etc/dfs/dfstab` **file and add the** `–sec=dh` **option to the appropriate entries (for Diffie-Hellman authentication).**

```
share -F nfs -o sec=dh /export/home
```

7. **Edit the** `auto_master` **data to include** `–sec=dh` **as a mount option in the appropriate entries (for Diffie-Hellman authentication):**

```
/home auto_home -nosuid,sec=dh
```

Note - With 2.5 and earlier Solaris releases, 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 2 on later releases, the NFS server refuses access if the security modes do not match, unless `–sec=none` is included on the `share` command line. With version 3, the mode is inherited from the NFS server, so there is no need for the clients to specify `–sec=krb4` or `–sec=dh`. The users have access to the files as themselves.

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

```
# keylogin -r
```

▼ How to Set Up a Secure NFS Environment With KERB Authentication

1. **Edit the** `/etc/dfs/dfstab` **file and add the** `sec=krb4` **option to the appropriate entries.**

```
# share -F nfs -o sec=krb4 /export/home
```

2. **Edit the** `auto_master` **data to include** `sec=krb4` **as a mount option.**

```
/home auto_home -nosuid,sec=krb4
```

Note - With 2.5 and earlier Solaris releases, 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 2 on later releases, the NFS server refuses access if the security modes do not match, unless `–sec=none` is included on the `share` command line. With version 3, the mode is inherited from the NFS server, so there is no need for the clients to specify `–sec=krb4` or `–sec=dh`. The users have access to the files as themselves.

WebNFS Administration Tasks

This section provides instructions for administering the WebNFS system. The following tasks are discussed.

- "Planning for WebNFS Access" on page 23
- "How to Enable WebNFS Access" on page 24
- "How to Browse Using an NFS URL" on page 25
- "How to Enable WebNFS Access Through a Firewall" on page 25

Planning for WebNFS Access

To use the WebNFS functionality, you first need an application capable of running and loading an NFS URL (for example, `nfs://server/path`). The next step is to choose the file system that will be exported for WebNFS access. If the application is web browsing, often the document root for the web server is used. Several factors need to be considered when choosing a file system to export for WebNFS access.

1. Each server has one public file handle that by default is associated with the server's root file system. The path in an NFS URL is evaluated relative to the directory with which the public file handle is associated. If the path leads to a file or directory within an exported file system, then the server provides access. You can use the `–public` option of the `share` command to associate the public file handle with a specific exported directory. Using this option allows URLs to be relative to the shared file system rather than to the servers' root file system. By default the public file handle points to the root file system, but this file handle does not allow web access unless the root file system is shared.

2. The WebNFS environment allows users who already have mount privileges to access files through a browser regardless of whether the file system is exported using the –public option. Because users already have access to these files through the NFS setup, this should not create any additional security risk. You only need to share a file system using the –public option if users who cannot mount the file system need to use WebNFS access.
3. File systems that are already open to the public make good candidates for using the –public option, like the top directory in an ftp archive or the main URL directory for a web site.
4. You can use the –index option with the share command to force the loading of an HTML file instead of listing the directory when an NFS URL is accessed.

   After a file system is chosen, review the files and set access permissions to restrict viewing of files or directories as needed. Establish the permissions as appropriate for any NFS file system that is being shared. For many sites, 755 permissions for directories and 644 permissions for files provides the correct level of access.

   Additional factors need to be considered if both NFS and HTTP URLs are going to be used to access one Web site. These are described in "WebNFS Limitations With Web Browser Use" on page 66.

▼ How to Enable WebNFS Access

Starting with the 2.6 release, by default all file systems that are available for NFS mounting are automatically available for WebNFS access. The only time that this procedure needs to be followed is on servers that do not already allow NFS mounting, if resetting the public file handle is useful to shorten NFS URLs, or if the –index option is required.

1. **Edit the /etc/dfs/dfstab file.**

   Add one entry to the file for the file system that you want to have shared automatically. The –index tag is optional.

```
share -F nfs -o ro,public,index=index.html /export/ftp
```

2. **Check that the NFS service 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 and restart the daemons.

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

3. **Share the file system.**

   After the entry is in `/etc/dfs/dfstab`, the file system can be shared by either rebooting the system or by using the `shareall` command. If the NFS daemons were restarted in step 2, then this command does not need to be run because the script runs the command.

   ```
   # shareall
   ```

4. **Verify that the information is correct.**

   Run the `share` command to check that the correct options are listed:

```
# share
-       /export/share/man   ro   ""
-       /usr/src     rw=eng   ""
-       /export/ftp    ro,public,index=index.html  ""
```

▼ How to Browse Using an NFS URL

Browsers capable of supporting WebNFS access should provide access using an NFS URL that looks something like:

```
nfs://server<:port>/path
```

server is the name of the file server, *port* is the port number to use (the default value is `2049`), and *path* is the path to the file. Path can either be relative to the public file handle or relative to the root file system on the server.

Note - In most browsers, the URL service type (for example, `nfs` or `http`) is remembered from one transaction to the next, unless a URL that includes a different service type is loaded. When using NFS URLs, if a reference to an HTTP URL is loaded, then subsequent pages are loaded using the HTTP protocol instead of the NFS protocol, unless the URLs specify an NFS URL.

▼ How to Enable WebNFS Access Through a Firewall

You can enable WebNFS access for clients that are not part of the local subnet by configuring the firewall to allow a TCP connection on port `2049`. Just allowing access for `httpd` does not allow NFS URLs to be used.

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, you can kill it with the keyboard interrupt Control-c.

When the network or server has problems, programs that access hard-mounted remote files 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 awhile. Because these errors can result in unexpected application errors and data corruption, avoid soft-mounting.

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 generates 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 are run. Writable data could be corrupted if the application ignores the errors. Mounted executables might not load properly and can fail.

NFS Troubleshooting Procedures

To determine where the NFS service has failed, you need to follow several procedures to isolate the failure. Check for the following items:

- 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 might become apparent that other portions of the network are not functioning, such as the name service or the physical network hardware. The *Solaris Naming Administration Guide* contains debugging procedures for the NIS+ name service. Also, during the process it might 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 an NFS Client

1. **Check 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 28).

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, make sure that the client has received the correct host information by typing the following:**

```
% /usr/bin/getent hosts bee
129.144.83.117 bee.eng.acme.com
```

4. **If the host information is correct, 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 30.

5. **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).

6. **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 NFS services have started on the NFS server by typing the following command:**

```
% rpcinfo -s bee|egrep 'nfs|mountd'
 100003  3,2    tcp,udp                          nfs     superuser
 100005  3,2,1  ticots,ticotsord,tcp,ticlts,udp  mountd  superuser
```

   If the daemons have not been started, see "How to Restart NFS Services" on page 32.

2. **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 tests the TCP connection. If this fails, skip to "How to Verify the NFS Service on the Server" on page 30.

3. **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 tests the TCP connection. If either attempt fails, skip to "How to Verify the NFS Service on the Server" on page 30.

4. **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
```

5. **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 that client would not be able to mount the `/usr/src` file system.

Check all entries that include mounting informtion in all of the local files. The list includes `/etc/vfstab` and all the `/etc/auto_*` files.

▼ How to Verify the NFS Service on the Server

1. **Log on to the server as `root`.**

2. **Check 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 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   2462  1  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 32).

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 2462 1 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 32).

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 32.

▼ 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 stops the daemons and restarts them, if there is an entry in `/etc/dfs/dfstab`.

▼ How to Warm-Start `rpcbind`

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

1. **As root on the server, get the PID for** `rpcbind`**.**

   Run `ps` to get the PID (which is 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 causes `rpcbind` to create a list of the current registered services in `/tmp/portmap.file` and `/tmp/rpcbind.file`.

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

Note - If you do not kill the `rpcbind` process with the `–s term` option, then you cannot complete a warm start of `rpcbind` and must reboot the server to restore service.

3. **Restart** `rpcbind`**.**

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

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

▼ How to Identify Which Host Is Providing NFS File Service

♦ **Run the `nfsstat` command with the `–m` option to gather current NFS information.**

The name of the current server is printed after “`currserver=`”.

```
% nfsstat -m
/usr/local from bee,wasp:/export/share/local
 Flags: vers=3,proto=tcp,sec=sys,hard,intr,llock,link,synlink,
  acl,rsize=32768,wsize=32678,retrans=5
 Failover: noresponse=0, failover=0, remap=0, currserver=bee
```

▼ How to Verify Options Used With the `mount` Command

In the Solaris 2.6 release and in any versions of the `mount` command that were patched after the 2.6 release, no warning is issued for invalid options. The following procedure helps determine whether the options that were supplied either on the command line or through `/etc/vfstab` were valid.

For this example, assume that the following command has been run:

```
# mount -F nfs -o ro,vers=2 bee:/export/share/local /mnt
```

1. **Run the nfsstat command to verify the options.**

```
# nfsstat -m
/mnt from bee:/export/share/local
Flags:  vers=2,proto=tcp,sec=sys,hard,intr,dynamic,acl,rsize=8192,wsize=8192,
        retrans=5
```

(continued)

```
```

Notice that the file system from `bee` has been mounted with the protocol version set to `2`. Unfortunately, the `nfsstat` command does not display information about all of the options, but using the `nfsstat` command is the most accurate way to verify the options.

2. **Check the entry in `/etc/mnttab`.**

   The `mount` command does not allow invalid options to be added to the mount table, so verifying that the options listed in the file match those listed on the command line is a way to check those options not reported by the `nfsstat` command.

```
# grep bee /etc/mnttab
bee:/export/share/local /mnt nfs ro,vers=2,dev=2b0005e 859934818
```

NFS Error Messages

This section shows an error message followed by a description of the conditions which should create the error and at least one way of fixing the problem.

`Bad argument specified with index option - must be a file`

You must include a file name with the `–index` option. You cannot use directory names.

`Cannot establish NFS service over /dev/`*tcp*`: transport setup problem`

This message is often created when the services information in the name space has not been updated. It can also be reported for UDP. To fix this problem, you must update the services data in the name space. For NIS+ the entries should be:

```
nfsd nfsd tcp 2049 NFS server daemon
nfsd nfsd ucp 2049 NFS server daemon
```

For NIS and `/etc/services`, the entries should be:

```
nfsd    2049/tcp    nfs    # NFS server daemon
nfsd    2049/ucp    nfs    # NFS server daemon
```

`Cannot use index option without public option`

Include the `public` option with the index option in the share command. You must define the public file handle for the `–index` option to work.

Note - The Solaris 2.5.1 release required that the public file handle be set using the share command. A change in the Solaris 2.6 release sets the public file handle to be `/` by default, this error message is no longer relevant.

`Could not use public filehandle in request to` *server*

This message is displayed if the public option is specified but the NFS server does not support the public file handle. In this case, the mount will fail. To remedy this situation, either try the mount request without using the public file handle or reconfigure the NFS server to support the public file handle.

`NOTICE: NFS3: failing over from` *host1* `to` *host2*

This message is displayed on the console when a failover occurrs. It is an advisory message only.

filename`: File too large`

An NFS version 2 client is trying to access a file that is over 2 Gbytes.

`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 might 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 mount: ignoring invalid option "`*-option*`"`

The *-option* flag is not valid. Refer to the **`mount_nfs`**(1M) man page to verify the required syntax.

Note - This error message is not displayed when running any version of the `mount` command included in a Solaris release from 2.6 to the current release or in earlier versions that have been patched.

`nfs mount: NFS can't support "nolargefiles"`

An NFS client has attempted to mount a file system from an NFS server using the `–nolargefiles` option. This option is not supported for NFS file system types.

`nfs mount: NFS V2 can't support "largefiles"`

The NFS version 2 protocol cannot handle large files. You must use version 3 if access to large files is required.

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

If programs hang while doing file-related work, your NFS server might be dead. This message indicates that NFS server *hostname* is down or that there is a problem with the server or with the network. If failover is being used, then *hostname* is a list of servers. Start with "How to Check Connectivity on an NFS Client" on page 27.

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

This error can be caused by many situations. One of the most difficult to debug is when this occurs because a user is in too many groups. Currently a user can 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 at least Solaris 2.5 is running on the NFS server and the NFS clients, then use ACLs to provide the needed access privileges.

`port` *number* `in nfs URL not the same as port` *number* `in port option`

The port number included in the NFS URL must match the port number included with the `–port` option to mount. If the port numbers do not match, the mount will fail. Either change the command to make the port numbers the same or do not specify the port number that is incorrect. There usually is no reason to specify the port number both in the NFS URL and with the `–port` option.

`replicas must have the same version`

For NFS failover to function properly, the NFS servers that are replicas must support the same version of the NFS protocol. Mixing version 2 and version 3 servers is not allowed.

`replicated mounts must be read-only`

NFS failover does not work on file systems that are mounted read-write. Mounting the file system read-write increases the likelihood that a file will change. NFS failover depends on the file systems being identical.

`replicated mounts must not be soft`

Replicated mounts require that you wait for a timeout before failover occurs. The `soft` option requires that the mount fail immediately when a timeout starts, so you cannot include the `–soft` option with a replicated mount.

`share_nfs: Cannot share more than one filesystem with 'public' option`

Check the `/etc/dfs/dfstab` file to make sure that only one file system is selected to be shared with the `–public` option. Only one public file handle can be established per server, so only one file system per server can be shared with this option.

`WARNING: No network locking on` *hostname*`:`*path*`: contact admin to install server change`

An NFS client has unsuccessfully attempted to establish a connection with the network lock manager on an NFS server. Rather than fail the mount, this warning is generated to warn you that locking is not going to work.

CHAPTER **3**

NFS Reference

This chapter provides an introduction to the NFS commands. This chapter also provides information about all of the pieces of the NFS environment and how these pieces work together.

- "NFS Files" on page 39
- "NFS Daemons" on page 40
- "NFS Commands" on page 43
- "Other Useful Commands" on page 56
- "How It All Works Together" on page 61

NFS Files

You need several ASCII files to support NFS activities on any computer. Table 3–1 lists these files and their functions.

TABLE 3–1 NFS ASCII Files

File Name	Function
`/etc/mnttab`	Lists file systems that are currently mounted, including automounted directories (see the **`mnttab`**(4) man page); do not edit this file.
`/etc/netconfig`	Lists the transport protocols; do not edit this file.
`/etc/nfssec.conf`	Lists NFS security services; do not edit this file.

TABLE 3–1 NFS ASCII Files *(continued)*

File Name	Function
`/etc/rmtab`	Lists file systems remotely mounted by NFS clients (see the **`rmtab`**(4) man page); do not edit this file.
`/etc/vfstab`	Defines file systems to be mounted locally (see the **`vfstab`**(4) man page).
`/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.

Only one entry is in `/etc/default/fs`: the default file-system type for local disks. You can determine the file system types that are supported on a client or server 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 sends locking requests from the client to the NFS server. On the NFS server, it starts local locking.

The daemon is normally started without any options. You can use three options with this command (see the **`lockd`**(1M) man page).

The `–g` *graceperiod* option selects the number of seconds that the clients have to reclaim locks after a server reboot. During this time, the NFS server only processes reclaims of old locks. All other requests for service must wait until the grace period is over. This option affects the NFS server-side response, so can be changed only on an NFS 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 a reduction increases the chances that a client might not be able to recover all its locks.

The `–t` *timeout* option selects the number of seconds to wait before retransmitting a lock request to the remote server. This option affects the NFS client-side service. 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.

The *nthreads* option specifies the maximum number of concurrent threads that the server handles per connection. Base the value for *nthreads* on the load expected on the NFS server. The default value is 20. Since each NFS client using TCP uses a single connection with the NFS server, each TCP client is granted the ability to use up to 20 concurrent threads on the server. All NFS clients using UDP share a single connection with the NFS server. Under these conditions it might be necessary to increase the number of threads available for the UDP connection. A minimum calculation would be to allow two threads for each UDP client, but this is specific to the workload on the client, so two threads per client might not be sufficient. The disadvantage to using more threads is that when the threads are used, more memory is used on the NFS server, but if the threads are never used, increasing *nthreads* will have no effect.

mountd

This is a remote procedure call (RPC) server that handles file system mount requests from remote systems and provides access control. 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. You can use two options with this command (see the **`mountd`**(1M) man page): `–v` and `–r`.

The `–v` option runs the command in verbose mode. Each time an NFS server determines the access a client should get, a message is printed on the console. The information generated can be useful when trying to determine why a client can not access a file system.

The `–r` option rejects all future mount requests from clients. This does not affect clients that already have a file system mounted.

nfsd

This daemon handles other client file-system requests. You can use several options with this command. See the **nfsd**(1M) man page for a complete listing.

The –l option sets the connection queue length for the NFS/TCP over connection-oriented transports. The default value is 32 entries.

The –c *#_conn* option selects the maximum number of connections per connection-oriented transport. The default value for *#_conn* is unlimited.

The *nservers* option is the maximum number of concurrent requests that a server can handle. 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 only shows 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 an NFS server. If a server crashes, upon rebooting statd on the server contacts statd on the client. The client statd can then attempt to reclaim any locks on the server. The client statd also informs 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. For more information see the **statd**(1M) man page.

In the Solaris 7 release, the way that statd keeps track of the clients has been improved. In all earlier Solaris releases, statd created files in /var/statmon/sm for each client using the client's unqualified hostname. This caused problems if you had two clients in different domains that shared a hostname, or if there were clients that were not resident in the same domain as the NFS server. Since the unqualified hostname only lists the hostname, without any domain or IP-address information, the older version of statd had no way to differentiate between these types of clients. To fix this, the Solaris 7 statd creates a symbolic link in /var/statmon/sm to the unqualified hostname using the IP address of the client. The new link will look like:

```
# ls -l /var/statmon/sm
lrwxrwxrwx   1 root         11 Apr 29 16:32 ipv4.192.9.200.1 -> myhost
--w-------   1 root         11 Apr 29 16:32 myhost
```

In this example, the client hostname is myhost and the clients' IP address is 192.9.200.1. If another host with the name myhost were mounting a file system, there would be two symbolic links to the hostname.

NFS Commands

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

`clear_locks`

This command enables you to remove all file, record, and share locks for an NFS client. You must be `root` to run this command. From an NFS server you can clear the locks for a specific client and from an NFS client you can clear locks for that client on a specific server. The following example would clear the locks for the NFS client named `tulip` on the current system.

```
# clear_locks tulip
```

Using the `-s` option enables you to specify which NFS host to clear the locks from. It must be run from the NFS client, which created the locks. In this case, the locks from the client would be removed from the NFS server named `bee`.

```
# clear_locks -s bee
```

Caution - This command should only be run when a client crashes and cannot clear its locks. To avoid data corruption problems, do not clear locks for an active client.

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.

Many types of file systems are included in the standard Solaris installation. For a complete description of all file system types, see “Types of File Systems” in *System Administration Guide, Volume I.* Each file system type has a specific man page that lists the options to mount that are appropriate for that file system type. The man page for NFS file systems is **mount_nfs**(1M); for UFS file systems it is **mount_ufs**(1M); and so forth.

The Solaris 7 release includes the ability to select a pathname to mount from an NFS server using an NFS URL instead of the standard server:/pathname syntax. See “How to Mount an NFS File System Using an NFS URL” on page 20 for further information.

Caution - The version of the mount command included in any Solaris release from 2.6 to the current release, will not warn about options that are not valid. The command silently ignores any options that cannot be interpreted. Make sure to verify all of the options that were used to prevent unexpected behavior.

---

mount Options for NFS File Systems

The subsequent text lists some of the options that can follow the –o flag when mounting an NFS file system.

bg|fg

These options can be used to select the retry behavior if a mount fails. The –bg option causes the mount attempts to be run in the background. The –fg option causes the mount attempt to be run in the foreground. 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 can do other processing while waiting for the mount request to complete.

largefiles

This option makes it possible to access files larger than 2 Gbytes on a server running the Solaris 2.6 release. Whether a large file can be accessed can only be controlled on the server, so this option is silently ignored on NFS version 3 mounts. Starting with release 2.6, by default, all UFS file systems are mounted with –largefiles. For mounts using the NFS version 2 protocol, the `–largefiles` option causes the mount to fail with an error.

nolargefiles

This option for UFS mounts guarantees that there are and will be no large files on the file system (see the **`mount_ufs`**(1M) man page). Because the existence of large files can only be controlled on the NFS server, there is no option for `–nolargefiles` using NFS mounts. Attempts to NFS mount a file system using this option are rejected with an error.

public

This option forces the use of the public file handle when contacting the NFS server. If the public file handle is supported by the server, the mounting operation is faster because the MOUNT protocol is not used. Also, because the MOUNT protocol is not used, the public option allows mounting to occur through a firewall.

rw|ro

The `–rw` and `–ro` options indicate whether a 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 need to be changed by users. The read-only option is appropriate for directories that should not be changed by users; for example, shared copies of the man pages should not be writable by users.

sec=*mode*

You can use this option to specify the authentication mechanism to be used during the mount transaction. The value for *mode* can be one of the values shown in Table 3–2. The modes are also defined in `/etc/nfssec.conf`.

TABLE 3–2 NFS Security Modes

Mode	Authentication Service Selected
`krb4`	Kerberos Version 4
`none`	No authentication

TABLE 3–2 NFS Security Modes *(continued)*

Mode	Authentication Service Selected
dh	Diffie-Hellman (DH) authentication
sys	Standard UNIX authentication

soft | hard

An NFS file system mounted with the soft option returns an error if the server does not respond. The hard option causes 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 mounted using the hard option and becomes unavailable, an application using this file system will hang until the file system becomes available.

Using the mount Command

Both of these commands 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 uses the –O option to 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
```

This command uses client failover:

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

Note - When used from the command line, the listed servers must support the same version of the NFS protocol. Do not mix version 2 and version 3 servers when running mount from the command line. You can use mixed servers with autofs, in which case the best subset of version 2 or version 3 servers is used.

Here is an example of using an NFS URL with the mount command:

```
# mount -F nfs nfs://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 enables you to remove a remote file system that is currently mounted. The umount command supports the –V option to allow for testing. You might also use the –a option to umount several file systems at one time. 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.

Because the file system is already mounted and should have an entry in /etc/mnttab, you do not need to include a flag for the file system type.

The command cannot 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 is busy until the working directory is changed. The umount command can 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
```

Notice that this command does not actually unmount the file systems.

mountall

Use this command to mount all file systems or a specific group of file systems listed in a file system table. The command provides a way to select the file system type to be accessed with the `–F` *FSType* option, to select all the remote file systems listed in a file system table with the `–r` option, and to select all the local file systems with the `–l` option. Because all file systems labeled as NFS file system type are remote file systems, some of these options are redundant. For more information, see the **`mountall`**(1M) man page.

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 `–k` option runs the `fuser –k` *mount_point* command to kill any processes associated with the *mount_point*. The `–s` option indicates that unmount is not to be performed in parallel. `–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. You cannot combine the `–h` option with `–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 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 does 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 cannot be shared that is part of a larger file system already being 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` must to be moved to a separate disk slice.

Note - You can gain access to a file system that 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

Some of the options that you can include with the `–o` flag are:

rw|ro

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

rw=*accesslist*

The file system is shared read-write to the listed clients only. All other requests are denied. Starting with the Solaris 2.6 release, the list of clients defined in *accesslist* has been expanded. See “Setting Access Lists With the `share` Command” on page 52 for more information. You can use this option to override an `–ro` option.

The options that you can use with NFS file systems only include:

aclok

This option enables an NFS server supporting the NFS version 2 protocol to be configured to do access control for NFS version 2 clients. Without this option all clients are given minimal access. With this option the clients have maximal access. For instance, on file systems shared with the `–aclok` option, if anyone has read permissions, then everyone does. However, without this option, it is possible to deny access to a client who should have access permissions. Whether it is preferred to permit too much access or to permit too little, depends on the security systems already in place. See "Securing Files (Tasks)" in *System Administration Guide, Volume II* for more information about access control lists (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 get correct access, but cannot manipulate the ACLs. If the software supports the NFS_ACL protocol, then the clients get correct access and can manipulate the ACLs. Starting with release 2.5, the Solaris system supports both protocols.

---

anon=*uid*

You use *uid* to select the user ID of unauthenticated users. If you set *uid* to `-1`, the server denies access to unauthenticated users. You can grant root access by setting `anon=0`, but this will allow unauthenticated users to have root access, so use the `root` option instead.

index=*filename*

You can use the `–index=`*filename* option to force the loading of a HyperText Markup Language (HTML) file instead of displaying a listing of the directory when a user accesses an NFS URL. This option mimics the action of current browsers if an `index.html` file is found in the directory that the HTTP URL is accessing. This is the equivalent of setting the `DirectoryIndex` option for `httpd`. For instance, if the `dfstab` file entry looks like:

```
share -F nfs -o ro,public,index=index.html /export/web
```

these URLs will display the same information:

```
nfs://<server>/<dir>
nfs://<server>/<dir>/index.html
nfs://<server>//export/web/<dir>
nfs://<server>//export/web/<dir>/index.html
```

(continued)

```
http://<server>/<dir>
http://<server>/<dir>/index.html
```

nosuid

This option signals that all attempts to enable the `setuid` or `setgid` mode should be ignored. NFS clients cannot be able to create files with the `setuid` or `setgid` bits on.

public

The –`public` option has been added to the `share` command to enable WebNFS browsing. Only one file system on a server can be shared with this option.

root=*accesslist*

The server gives root access to the hosts in the list. By default, the server does not give root access to any remote hosts. If the selected security mode is anything other than –`sec=sys`, then you can only include client host names in the *accesslist*. Starting with the Solaris 2.6 release, the list of clients defined in *accesslist* is expanded. See "Setting Access Lists With the `share` Command" on page 52 for more information.

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

sec=*mode*[:*mode*]

mode selects the security modes that are needed to get access to the file system. By default, the security mode is UNIX authentication. You can specify multiple modes, but use each security mode only once per command line. Each –`mode` option applies to any subsequent –`rw`, –`ro`, –`rw=`, –`ro=`, –`root=`, and –`window=` options, until another –`mode` is encountered. Using –`sec=none` maps all users to user `nobody`.

window=*value*

value selects the maximum life time in seconds of a credential on the NFS server. The default value is 30000 seconds or 8.3 hours.

Setting Access Lists With the `share` Command

In Solaris releases prior to 2.6, the *accesslist* included with either the `–ro=`, `–rw=`, or `–root=` option of the `share` command were restricted to a list of host names or netgroup names. Starting with the Solaris 2.6 release, the access list can also include a domain name, a subnet number, or an entry to deny access. These extensions should make it easier to control file access control on a single server, without having to change the name space or maintain long lists of clients.

This command 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
```

Note - You cannot specify both `rw` and `ro` without arguments. If no read-write option is specified, the default is read-write for all clients.

To share one file system with multiple clients, you must enter all options on the same line, because multiple invocations of the `share` command on the same object "remember" only the last command run. This command enables 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
```

When sharing a file system using multiple authentication mechanisms, make sure to include the `–ro`, `–ro=`, `–rw`, `–rw=`, `–root`, and `–window` options after the correct security modes. In this example, UNIX authentication is selected for all hosts in the netgroup named `eng`. These hosts can only mount the file system in read-only mode. The hosts `tulip` and `lilac` will be able to mount the file system read-write if they use Diffie-Hellman authentication. With these options, `tulip` and `lilac` will be able to mount the file system read-only even if they are not using DH authentication, if the host names are listed in the `eng` netgroup.

```
# share -F nfs -o sec=dh,rw=tulip:lilac,sec=sys,ro=eng /usr/src
```

Even though UNIX authentication is the default security mode, it is not included if the `–sec` option is used, so it is important to include a `–sec=sys` option if UNIX authentication is to be used with any other authentication mechanism.

You can use a DNS domain name in the access list by preceding the actual domain name with a dot. The dot indicates that the string following it is a domain name, not a fully qualified host name. The following entry allows mount access to all hosts in the `eng.sun.com` domain:

```
# share -F nfs -o ro=.:.eng.sun.com /export/share/man
```

In this example, the single "`.`" matches all hosts that are matched through the NIS or NIS+ name spaces. The results returned from these name services do not include the domain name. The "`.eng.sun.com`" entry matches all hosts that use DNS for name space resolution. DNS always returns a fully qualified host name, so the longer entry is required if you use a combination of DNS and the other name spaces.

You can use a subnet number in an access list by preceding the actual network number or the network name with "`@`". This differentiates the network name from a netgroup or a fully qualified host name. You must identify the subnet in either `/etc/networks` or in a NIS or NIS+ name space. The following entries have the same effect if the `129.144` subnet has been identified as the `eng` network:

```
# share -F nfs -o ro=@eng /export/share/man
# share -F nfs -o ro=@129.144 /export/share/man
# share -F nfs -o ro=@129.144.0.0 /export/share/man
```

The last two entries show that it is not necessary to include the full network address.

If the network prefix is not byte aligned, as with Classless Inter-Domain Routing (CIDR), the mask length can be explicitly specified on the command line. The mask length is defined by following either the network name or the network number with a slash and the number of significant bits in the prefix of the address. For example:

```
# share -f nfs -o ro=@eng/17 /export/share/man
# share -F nfs -o ro=@129.144.132/17 /export/share/man
```

In these examples, the "`/17`" indicates that the first 17 bits in the address are to be used as the mask. For additional information on CIDR, look up RFC 1519.

You can also select negative access by placing a "`-`" before the entry. Notice that because the entries are read from left to right, you must place the negative access entries before the entry they apply to:

```
# share -F nfs -o ro=-rose:.eng.sun.com /export/share/man
```

This example would allow access to any hosts in the `eng.sun.com` domain except the host named `rose`.

`unshare`

This command allows you to make a previously available file system unavailable for mounting by clients. You can use the `unshare` command 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. You must remove the entry for this file system 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 might 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 shares all entries in `/etc/dfs/dfstab`. You can include a file name to specify the name of a file that lists `share` command lines. If you do not include a file name, `/etc/dfs/dfstab` is checked. If you use a “`-`” to replace the file name, then you can type `share` commands 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 makes all currently shared resources unavailable. The `–F` *FSType* option selects a list of file system types defined in `/etc/dfs/fstypes`. This flag enables 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 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 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 local directories 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 mount and umount commands consult the table. Generally, there is no reason to run this command manually; it runs automatically when a system is booted.

Other Useful Commands

These commands can be useful when troubleshooting NFS problems.

- "`nfsstat`" on page 56
- "`pstack`" on page 57
- "`rpcinfo`" on page 58
- "`snoop`" on page 60
- "`truss`" on page 60

`nfsstat`

You can use this command 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 important for debugging problems when new software or hardware is added to the computing environment. Running this command at least once a week, and storing the numbers, provides 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
```

(continued)

```
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    readdirplus 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 remaining lines report NFS activities. In both sets of statistics, knowing the average number of `badcalls` or `calls` and the number of calls per 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 note 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 that the NFS clients have enough memory. The number of `commit` operations on the server go up when clients do not have available resources.

pstack

This command displays a stack trace for each process. It must be run by `root`. You can use it 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).

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, efffff14, 0, 0, ffffffff, 400) + 3c8
 00010fb0 _start   (0, 0, 0, 0, 0, 0) + 5c
```

It shows that the process is waiting for a new connection request. This is a normal response. If the stack shows that the process is still in poll after a request is made, it is possible that the process is hung. Follow the instructions in "How to Restart NFS Services" on page 32 to fix this problem. Review the instructions in "NFS Troubleshooting Procedures" on page 27 to fully verify that your problem is a hung program.

rpcinfo

This command generates information about the RPC service running on a system. You can also use it to change the RPC service. Many options are available with this command (see the **rpcinfo**(1M) man page). This is a shortened synopsis for some of the options that you can use 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 host name 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 host name is used. You can substitute the RPC program number for *progname*, but many users will remember the name and not the number. You can use the `–p` option in place of the `–s` option on those systems that 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. Many options are 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* stores all the captured packets into the named file, and *hostname* indicates to display only packets going to and from a specific host.

The `–d` *device* option is useful on those servers that have multiple network interfaces. You can use many other expressions 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 proper host. Also, look for error messages. Saving the packets to a file can make it much easier to review the data.

truss

You can use this command to see if a process is hung. It must be run by `root`. You can use many options 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 listed system calls from the trace.

This example shows that the process is waiting for another connection request from a new client.

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

This is a normal response. If the response does not change after a new connection request has been made, the process could be hung. Follow the instructions in “How to Restart NFS Services” on page 32 to fix the hung program. Review the instructions in “NFS Troubleshooting Procedures” on page 27 to fully verify that your problem is a hung program.

How It All Works Together

The following sections describe some of the complex functions of the NFS software.

Version 2 and Version 3 Negotiation

Because NFS servers might be supporting clients that are not using the NFS version 3 software, part of the initiation procedure includes negotiation of the protocol level. If both the client and the server can support version 3, then that version will be used. If either the client or the server can only support version 2, then that version will be selected.

You can override the values determined by the negotiation by using the `–vers` option to the `mount` command (see the **`mount_nfs`**(1M) man page). Under most circumstances, you should not have to specify the version level, as the best one is selected by default.

UDP and TCP Negotiation

During initiation, the transport protocol is also negotiated. By default, the first connection-oriented transport supported on both the client and the server is selected. If this does not succeed, then the first available connectionless transport protocol is used. The transport protocols supported on a system are listed in `/etc/netconfig`. TCP is the connection-oriented transport protocol supported by the release. UDP is the connectionless transport protocol.

When both the NFS protocol version and the transport protocol are determined by negotiation, the NFS protocol version is given precedence over the transport protocol. The NFS version 3 protocol using UDP is given higher precedence than the NFS version 2 protocol using TCP. You can manually select both the NFS protocol version and the transport protocol with the `mount` command (see the **`mount_nfs`**(1M) man page). Under most conditions, it is better to allow the negotiation to select the best options.

File Transfer Size Negotiation

The file transfer size establishes the size of the buffers that are used when transferring data between the client and the server. In general, larger transfer sizes are better. The NFS version 3 protocol has an unlimited transfer size, but starting with the Solaris 2.6 release, the software bids a default buffer size of 32 Kbytes. The client can bid a smaller transfer size at mount time if needed, but under most conditions this is not necessary.

The transfer size is not negotiated with systems using the NFS version 2 protocol. Under this condition the maximum transfer size is set to 8 Kbytes.

You can use the `–rsize` and `–wsize` options to set the transfer size manually with the `mount` command. You might need to reduce the transfer size for some PC clients. Also, you can increase the transfer size if the NFS server is configured to use larger transfer sizes.

How File Systems Are Mounted

When a client needs to mount a file system from a server, it must obtain a file handle from the server that corresponds to the file system. This process requires that several transactions occur between the client and the server. In this example, the client is attempting to mount `/home/terry` from the server. A snoop trace for this transaction is shown below.

```
client -> server PORTMAP C GETPORT prog=100005 (MOUNT) vers=3 proto=UDP
server -> client PORTMAP R GETPORT port=33492
client -> server MOUNT3 C Null
server -> client MOUNT3 R Null
client -> server MOUNT3 C Mount /export/home9/terry
server -> client MOUNT3 R Mount OK FH=9000 Auth=unix
client -> server PORTMAP C GETPORT prog=100003 (NFS) vers=3 proto=TCP
server -> client PORTMAP R GETPORT port=2049
client -> server NFS C NULL3
server -> client NFS R NULL3
client -> server NFS C FSINFO3 FH=9000
server -> client NFS R FSINFO3 OK
client -> server NFS C GETATTR3 FH=9000
server -> client NFS R GETATTR3 OK
```

In this trace, the client first requests the mount port number from the portmap service on the NFS server. After the client received the mount port number (`33492`), that number is used to ping the service on the server. After the client has determined that a service is running on that port number, the client then makes a mount request. When the server responds to this request, it includes the file handle for the file system (`9000`) that is being mounted. The client then sends a request for the NFS port number. When the client receives the number from the server, it pings the NFS service (`nfsd`), and requests NFS information about the file system using the file handle.

In the following trace, the client is mounting the file system with the `–public` option.

```
client -> server NFS C LOOKUP3 FH=0000 /export/home9/terry
server -> client NFS R LOOKUP3 OK FH=9000
client -> server NFS C FSINFO3 FH=9000
server -> client NFS R FSINFO3 OK
client -> server NFS C GETATTR3 FH=9000
server -> client NFS R GETATTR3 OK
```

Notice that by using the default public file handle (which is `0000`), all of the transactions to get information from the portmap service and to determine the NFS port number are skipped.

Effects of the –`public` Option and NFS URLs When Mounting

Using the –`public` option can create conditions that cause a mount to fail. Adding an NFS URL can also confuse the situation. The specifics of how a file system is mounted when using these options are described below.

Public option with NFS URL — Forces the use of the public file handle. The mount fails if the public file handle is not supported.

Public option with regular path — Forces the use of the public file handle. The mount fails if the public file handle is not supported.

NFS URL only — Use the public file handle if enabled on the NFS server. If the mount fails using the public file handle, then try the mount using the MOUNT protocol.

Regular path only — Do not use the public file handle. The MOUNT protocol is used.

Client-Side Failover

Using client-side failover, an NFS client can switch to another server if the server supporting a replicated file system becomes unavailable. The file system can become unavailable if the server it is connected to crashes, if the server is overloaded, or if there is a network fault. The failover, under these conditions, is normally transparent to the user. Once established, the failover can occur at any time without disrupting the processes running on the client.

Failover requires that the file system be mounted read-only. The file systems must be identical for the failover to occur successfully. See “What Is a Replicated File System?” on page 64 for a description of what makes a file system identical. A static file system or one that is not changed often is the best candidate for failover.

You cannot use file systems that are mounted using CacheFS with failover. Extra information is stored for each CacheFS file system. This information cannot be updated during failover, so only one of these two features can be used when mounting a file system.

The number of replicas that need to be established for each file system depends on many factors. In general, it is better to have a couple of servers, each supporting multiple subnets rather than have a unique server on each subnet. The process requires checking of each server in the list, so the more servers that are listed, the slower each mount will be.

Failover Terminology

To fully comprehend the process, two terms need to be understood.

- *failover* – Selecting a server from a list of servers supporting a replicated file system. Normally, the next server in the sorted list is used, unless it fails to respond.
- *remap* – Making use of a new server. Through normal use, the clients store the path name for each active file on the remote file system. During the remap, these path names are evaluated to locate the files on the new server.

What Is a Replicated File System?

For the purposes of failover, a file system can be called a *replica* when each file is the same size and has the same vnode type as the original file system. Permissions, creation dates, and other file attributes are not considered. If the file size or vnode types are different, then the remap fails and the process hangs until the old server becomes available.

You can maintain a replicated file system using `rdist`, `cpio`, or an other file transfer mechanism. Because updating the replicated file systems causes inconsistency, follow these suggestions for best results:

- Rename the old version of the file before installing a new one.
- Run the updates at night when client usage is low.
- Keep the updates small.
- Minimize the number of copies.

Failover and NFS Locking

Some software packages require read locks on files. To prevent these products from breaking, read locks on read-only file systems are allowed, but are visible to the client side only. The locks persist through a remap because the server doesn't “know” about them. Because the files should not be changing, you do not need to lock the file on the server side.

Large Files

Starting with 2.6, the Solaris release supports files that are over 2 Gbytes. By default, UFS file systems are mounted with the `–largefiles` option to support the new functionality. Previous releases are not able to handle files of this size. See “How to Disable Large Files on an NFS Server” on page 17 for instructions.

No changes need to occur on a Solaris 2.6 NFS client to be able to access a large file, if the file system on the server is mounted with the `–largefiles` option. However, not all 2.6 commands will be able to handle these large files. See `largefile(5)` for a list of the commands that can handle the large files. Clients that cannot support the NFS version 3 protocol with the large file extensions will be unable to access any large files. Although clients running the Solaris 2.5 release can use the NFS version 3 protocol, large file support was not included in that release.

How the WebNFS Service Works

The WebNFS service makes files in a directory available to clients using a public file handle. A file handle is an address generated by the kernel that identifies a file for NFS clients. The *public file handle* has a predefined value, so there is no need for the server to generate a file handle for the client. The ability to use this predefined file handle reduces network traffic by eliminating the `MOUNT` protocol and should increase response time for the clients.

By default the public file handle on an NFS server is established on the root file system. This default provides WebNFS access to any clients that already have mount privileges on the server. You can change the public file handle to point to any file system by using the `share` command.

When the client has the file handle for the file system, a `LOOKUP` is run to determine the file handle for the file to be accessed. The NFS protocol allows the evaluation of only one path name component at a time. Each additional level of directory hierarchy requires another `LOOKUP`. A WebNFS server can evaluate an entire path name with a single transaction, called multicomponent lookup, when the `LOOKUP` is relative to the public file handle. With multicomponent lookup, the WebNFS server is able to deliver the file handle to the desired file without having to exchange the file handles for each directory level in the path name.

In addition, an NFS client can initiate concurrent downloads over a single TCP connection, which provides quick access without the additional load on the server caused by setting up multiple connections. Although Web browser applications support concurrent downloading of multiple files, each file has its own connection. By using one connection, the WebNFS software reduces the overhead on the server.

If the final component in the path name is a symbolic link to another file system, the client can access the file if the client already has access through normal NFS activities.

Normally, an NFS URL is evaluated relative to the public file handle. The evaluation can be changed to be relative to the server's root file system by adding an additional slash to the beginning of the path. In this example, these two NFS URLs are equivalent if the public file handle has been established on the `/export/ftp` file system.

```
nfs://server/junk
nfs://server//export/ftp/junk
```

WebNFS Limitations With Web Browser Use

Several functions that a Web site using HTTP can provide are not supported by the WebNFS software. These differences stem from the fact that the NFS server only sends the file, so any special processing must be done on the client. If you need to have one Web site configured for both WebNFS and HTTP access, then consider the following issues:

- NFS browsing does not run CGI scripts, so a file system with an active Web site that uses many CGI scripts might not be appropriate for NFS browsing.
- The browser might start different viewers, to handle files in different file formats. Accessing these files through an NFS URL will start an external viewer as long as the file type can be determined by the file name. The browser should recognize any file name extension for a standard MIME type when an NFS URL is used. Because the WebNFS software does not check inside the file to determine the file type—unlike some Web browser applications—the only way to determine a file type is by the file name extension.
- NFS browsing cannot utilize server-side image maps (clickable images). However, it can utilize client-side image maps (clickable images) because the URLs are defined with the location. No additional response is required from the document server.

The Secure NFS System

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. Historically, most NFS implementations have used UNIX (or AUTH_SYS) authentication, but stronger authentication methods such as AUTH_DH have also been available. When using UNIX authentication, an NFS server authenticates a file request by authenticating the computer making the request, but not the user, so a client user can run `su` and impersonate the owner of a file. If DH authentication is used, the NFS server authenticates the user, making this sort of impersonation much harder.

With root access and knowledge of network programming, anyone can introduce arbitrary data into the network and extract 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, as 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 environment 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 a Secure NFS system.

Secure RPC

Secure RPC is fundamental to the Secure NFS system. 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 can be plugged into it. Currently, there are three systems: UNIX, DH, and KERB (for Kerberos Version 4).

When UNIX authentication is used by a network service, the credentials contain the client's host name, UID, GID, and group-access list, but the verifier contains nothing. Because there is no verifier, a superuser could falsify 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 either DH authentication or KERB authentication.

DH Authentication

DH 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 name space. 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 the *Solaris Naming Administration Guide* for more information on how to set up the maps and tables.

The security of DH 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. The requirements for this scheme to work are:

- 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 synchronization to the point where the server begins to reject the client's requests, the DH 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 by 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 can 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 the "Overview of Secure RPC" in *System Administration Guide, Volume II* for more details.

Using Secure RPC With NFS

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 deleted. Now no process can access secure network services or mount an NFS file system. The important processes during a reboot are usually run as root, so these processes 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. The Secure NFS system provides protection only after the kernel and the key server 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 leave evidence. If you polled the network for boot servers, you would discover the devious boot server's location.
- Most setuid programs are owned by `root`; if the secret key for `root` is stored in `/etc/.rootkey`, these programs behave as they always have. If a setuid program is owned by a user, however, it might 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 access to your account. This is because your secret key gets passed to that computer's key server, which then stores it. This is only a concern if you do not trust the remote computer. If you have doubts, however, do not 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 key server.

- If a home directory is shared with the `–o sec=dh` or `–o sec=krb4` options, then remote logins can be a problem. If the `/etc/hosts.equiv` or `~/.rhosts` files are not set to prompt for a password, the login will succeed, but the users cannot access their home directories because 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.

PART III All About Autofs

This part of the manual discusses the autofs service and the procedures to use autofs.

CHAPTER **4**

Autofs Administration

This chapter provides information on how to perform autofs administration tasks, such as modifying automounter maps, using the automounter to reach non-NFS type devices, and configuring maps.

Setting Up Autofs

This section includes procedures to start and stop the autofs service.

▼ How to Start the Automounter

♦ **To enable the daemon without rebooting, become superuser and type the following command.**

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

This starts the daemon.

▼ How to Stop the Automounter

♦ **To disable the daemon without rebooting, become superuser and type the following command.**

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

Common Tasks and Procedures

This section describes some of the most common tasks you might 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 the *Solaris Naming Administration Guide* to perform the tasks discussed in this section.

Administrative Tasks Involving Maps

The following list shows the different administrative tasks you might need to perform involving maps to change your autofs environment.

- "How to Modify the Master Map" on page 76
- "How to Modify Indirect Maps" on page 76
- "How to Modify Direct Maps" on page 76
- "Avoiding Mount-Point Conflicts " on page 77

Table 4–1 describes the types of maps and their uses.

TABLE 4–1 Types of autofs 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 4–2 describes how to make changes to your autofs environment based on your name service.

TABLE 4–2 Map Maintenance

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

Table 4–3 tells you when to run the `automount` command, depending on the modification you have made to the type of map. For example, if you have made an addition or a deletion to a direct map, you need to run the `automount` command on the local system to allow the change take effect; however, if you've modified an existing entry, you do not need to run the `automount` command for the change to take effect.

TABLE 4–3 When to Run the `automount` Command

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

Modifying the Maps

The following procedures require that you use 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 the *Solaris Naming Administration Guide.*

2. **For each client, become superuser by typing `su` at a prompt and then the 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 the *Solaris Naming 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 the *Solaris Naming 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 only 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, then access it.

Note - Because of the additional steps, and because they do not take up as much space 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 the autofs service to mount other source directories, you might encounter a problem. If you specify the mount point `/src`, the service 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 then 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.

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 following examples show how this mounting could be done through autofs. The Volume Manager and autofs do not work together, so 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.

▼ How to Access CD-ROM Applications With Autofs

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 want to mount must appear as a name following a colon.

▼ How to Access PC-DOS Data Diskettes With Autofs

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 Access NFS File Systems Using CacheFS

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

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

2. **Add the cachefs entry to the appropriate automounter map.**

   For example, adding this entry to the master map caches all home directories:

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

   Adding this entry to the `auto_home` map only caches the home directory for the user named `rich`:

```
rich -fstype=cachefs,cachedir=/var/cache,backfstype=nfs dragon:/export/home1/rich
```

Note - Options that are included in maps that are searched later override options set in maps that are searched earlier. The last options found are the ones that are used. In the previous example, a specific entry added to the `auto_home` map only needs to include the options listed in the master maps if some of the options needed to be changed.

Customizing the Automounter

You can set up the automounter maps in several ways. The following tasks give detailed instructions on how to customize the automounter maps to provide an easy-to-use directory structure.

▼ How to Set Up a Common View of `/home`

The ideal is for all network users 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 master map: `/etc/auto_master`.

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

(continued)

```
/xfn      -xfn
```

A map for auto_home is also installed 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+.

Note - Users should not be permitted to run setuid executables from their home directories; without this restriction, any user could have superuser privileges on any computer.

▼ How to Set Up /home With Multiple Home Directory File Systems

1. **Install home directory partitions under /export/home.**

   If there are several partitions, install 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/&
```

   Notice 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 and the key in the map. This common view of all home directories is valuable when logging in to another user's computer. Autofs 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 in to 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, you need only change rusty's entry in the auto_home map to reflect the new location. Everyone else can continue to use the /home/rusty path.

▼ How to Consolidate Project-Related Files Under /ws

Assume 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 auto_ws map determines the contents of the /ws directory.

2. **Add the -nosuid option as a precaution.**

   This option prevents users from running setuid programs that might exist in any workspaces.

3. **Add entries to the auto_ws map.**

   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/&
```

(continued)

(Continuation)

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

The ampersand (&) at the end of each entry is 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. You must assign each of these subdirectories 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 larger, 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 substantial amounts of disk space. Through the life of the project you might 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, as the directory hierarchy under `/ws` is not changed.

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

▼ 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 *Solaris Naming 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
   ```

   Notice 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 auto_local map.**

   Your directory structure looks like this:

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

To satisfy the need to serve clients of different architectures, references to the `bin` directory need 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 SPARCstation™ 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 5.6 of the operating system, you need to export `/export/local/bin/sparc5.6` from the server and create similar entries for other releases. Because 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

The best way to share replicated file systems that are read-only is to use failover. See "Client-Side Failover" on page 63 for a discussion of failover.

♦ **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.

▼ How to Apply Security Restrictions

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

```
/home     auto_home     -nosuid
```

The `nosuid` option prevents users from creating files with the `setuid` or `setgid` bit set.

This entry overrides the entry for `/home` in a generic 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. If the entries in the `auto_home` map include mount options, then the `nosuid` option is overwritten, so either no options should be used in the `auto_home` map or the `nosuid` option must be included with each entry.

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

▼ How to Use a Public File Handle With Autofs

♦ **Create an entry like the following:**

```
/usr/local     -ro,public    bee:/export/share/local
```

The public option forces the public handle to be used. If the NFS server does not support a public file handle, the mount will fail.

▼ How to Use NFS URLs With Autofs

♦ **Create an entry like the following:**

```
/usr/local     -ro    nfs://bee/export/share/local
```

The service tries to use the public file handle on the NFS server, but if the server does not support a public file handle then the MOUNT protocol is used.

Disabling Autofs Browsability

Starting with the Solaris 2.6 release, the default version of /etc/auto_master that is installed has the –nobrowse option added to the entries for /home and /net. In addition, the upgrade procedure adds the –nobrowse option to the /home and /net entries in /etc/auto_master if these entries have not been modified. However, it might be necessary to make these changes manually or to turn off browsability for site-specific autofs mount points after the installation.

You can turn off the browsability feature in several ways. Disable it using a command-line option to the automountd daemon, which completely disables autofs browsability for the client. Or disable it for each map entry on all clients using the autofs maps in either a NIS or NIS+ name space, or for each map entry on each client, using local autofs maps if no network-wide name space is being used.

▼ How to Completely Disable Autofs Browsability on a Single NFS Client

1. **Add the –n option to the startup script.**

   As root, edit the /etc/init.d/autofs script and add the –n option to the line that starts the automountd daemon:

```
/usr/lib/autofs/automountd -n \
 < /dev/null > /dev/console 2>&1 # start daemon
```

2. **Restart the autofs service.**

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

▼ How to Disable Autofs Browsability for All Clients

To disable browsability for all clients, you must employ a name service such as NIS or NIS+. Otherwise, you need to manually edit the automounter maps on each client. In this example, the browsability of the `/home` directory is disabled. You must follow this procedure for each indirect autofs node that needs to be disabled.

1. **Add the** `–nobrowse` **option to the** `/home` **entry in the name service** `auto_master` **file.**

   ```
   /home      auto_home      -nobrowse
   ```

2. **On all clients: run the** `automount` **command.**

   The new behavior takes effect after running the `automount` command on the client systems or after a reboot.

   ```
   # /usr/sbin/automount
   ```

▼ How to Disable Autofs Browsability on an NFS Client

In this example, browsability of the `/net` directory is disabled. The same procedure can be used for `/home` or any other autofs mount points.

1. **Check the** `automount` **entry in** `/etc/nsswitch.conf`**.**

   For local file entries to take precedence, the entry in the name service switch file should list `files` before the name service. For example:

   ```
   automount:  files nisplus
   ```

   This is the default configuration in a standard Solaris installation.

2. **Check the position of the** `+auto_master` **entry in** `/etc/auto_master`**.**

   For additions to the local files to take precedence over the entries in the name space, the `+auto_master` entry must be moved below `/net`:

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

   A standard configuration places the `+auto_master` entry at the top of the file. This prevents any local changes from being used.

3. **Add the** `–nobrowse` **option to the** `/net` **entry in the** `/etc/auto_master` **file.**

```
/net     -hosts     -nosuid,nobrowse
```

4. **On all clients: run the** `automount` **command.**

   The new behavior takes effect after running the `automount` command on the client systems or after a reboot.

```
# /usr/sbin/automount
```

Troubleshooting Autofs

Occasionally, you might 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 that autofs generates. The list is divided into two parts:

- Error messages generated by the verbose (`–v`) option of `automount`
- Error messages that might 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 might 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 the possible problem.

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 can exist only in a file system that cannot be mounted (it cannot 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 *server*, 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 a NIS map. This can indicate NIS problems.

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

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

mountpoint`: Not a directory`

Autofs cannot mount itself on *mountpoint* because it is 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 can 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 tries to execute the maps, which creates 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
```

CHAPTER **5**

About Autofs

This chapter describes the parts of the autofs service and gives more detailed information about the autofs maps.

Autofs Programs

Two programs support the autofs service. Both are run when a system is booted, but only `automountd` is persistent.

`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 that have many automounted file systems, you might need 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 become very large. The output from `df` can be filtered by using the `–F` option (see the **`df`**(1M) man page) or by using `egrep` to help fix this problem.

Another factor to consider is that adjusting the duration also changes how quickly changes to the automounter maps will be reflected. Changes will not be seen until the file system is unmounted. Refer to "Modifying the Maps" on page 76 for instructions on how to modify automounter maps.

automountd

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

`automountd` [`–Tnv`] [`-D` *name*=*value*]

where `–T` selects to display each RPC call to standard output, `–n` disables browsing on all autofs nodes, `–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 default value for the automount map is `/etc/auto_master`. Use the `–T` option for troubleshooting.

Autofs Maps

Autofs uses three types of maps:

- Master map
- Direct maps
- Indirect maps

Master Map

The `auto_master` map associates a directory with a map. It is a master list specifying all the maps that autofs should check. The example below shows what an `auto_master` file could contain.

CODE EXAMPLE 5–1 Sample `/etc/auto_master` File

```
# Master map for automounter
#
+auto_master
/net            -hosts           -nosuid,nobrowse
/home           auto_home        -nobrowse
/xfn            -xfn
/-              auto_direct     -ro
```

This example shows the generic `auto_master` file with one addition for the `auto_direct` map. 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. 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.
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 (`/etc/nsswitch.conf`). Special maps are also used for `/net` and `/xfn` (see "Mount Point `/net`" on page 96 and "Mount Point `/xfn`" on page 97).
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. Options for each specific type of file system are listed in the mount man page for that file system (for example, see the **`mount_nfs`**(1M) man page for NFS specific mount options). For NFS specific mount points,

the bg (background) and fg (foreground) options do not apply.

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 maximum number of characters of an entry is 1024.

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.

Note - Autofs runs on all computers and supports /net and /home (automounted home directories) by default. These defaults can 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.

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. Notice 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 might not be available through /net/gumbo.

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 might no longer work. Instead, you should set up an entry in a map specifically for the file system you want rather than use /net.

Note - Autofs checks the server's export list only at mount time. After a server's file systems are mounted, autofs does not check with the server again until the server's file systems are automatically unmounted. Therefore, newly exported file systems are not "seen" until the file systems on the client are unmounted and then remounted.

Mount Point `/xfn`

This mount point provides the autofs directory structure for the resources that are shared through the FNS name space (see the *Solaris Naming Setup and Configuration Guide* for more information about FNS).

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. 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
```

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. Options for each specific type of file system are listed in the mount man page for that file system (for example, see the **`mount_cachefs`**(1M) man page for CacheFS specific mount options).

location

location is the location of the file system, specified (one or more) as *server:pathname* for NFS file systems or :*devicename* for High Sierra file systems (HSFS).

Note - The *pathname* should not include an automounted mount point; it should be the actual absolute path to the file 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
```

Note - There is no concatenation of options between the automounter maps. Any options added to an automounter map 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.

See "How Autofs Selects the Nearest Read-Only Files for Clients (Multiple Locations)" on page 105 for other important features associated with this type of map.

Mount Point `/–`

In Code Example 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 defined in the `auto_master` file. 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.)

An 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 entries are not duplicated.

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. Options for each specific type of file system are listed in the mount man page for that file system (for example, see the **`mount_nfs`**(1M) man page for NFS specific mount options).

location

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

Note - The *pathname* should not include an automounted mount point; it should be the actual absolute path to the file 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. Code Example 5–1 shows an `auto_master` map that contains the entry:

```
/home       auto_home         -nobrowse
```

`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
```

(continued)

(Continuation)

```
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 in to 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 override all options listed in maps that are searched earlier. For instance, options included in the auto_master map are 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 Works

Autofs is a client-side service that automatically mounts the appropriate file system. 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, until the file system is automatically unmounted by autofs after a period of inactivity.

The 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 are mounted in the future. These points are also known as trigger nodes.

After 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.

Starting with the Solaris 2.5 release, the automountd daemon is completely independent from the automount command. Because of this separation, it is possible to add, delete, or change map information without first having to stop and start the automountd daemon process.

After initially mounting autofs mounts, the automount command is used to update 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 daemon. After the file system is mounted, further access does not require any action from automountd until the file system is automatically unmounted.

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 name space 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 (See Figure 5–1). This script also runs the automount command, which reads the master map (see "How Autofs Starts the Navigation Process (Master Map)" on page 103) and installs autofs mount points.

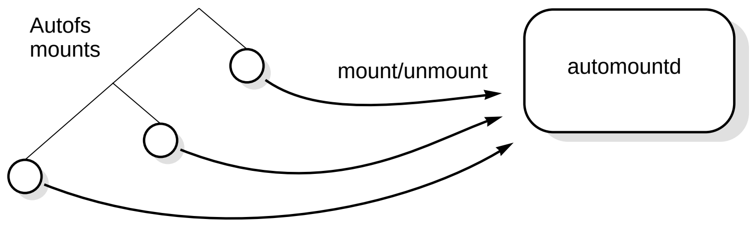

Figure 5–1 `/etc/init.d/autofs` Script Starts `automount`

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, creates the trigger nodes, and performs the mount.
4. Autofs allows the intercepted request to proceed.
5. Autofs unmounts the file system after a period 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 does not check that the object has been unmounted, resulting in possible inconsistency. A reboot clears all of the autofs mount points.

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 110.

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.

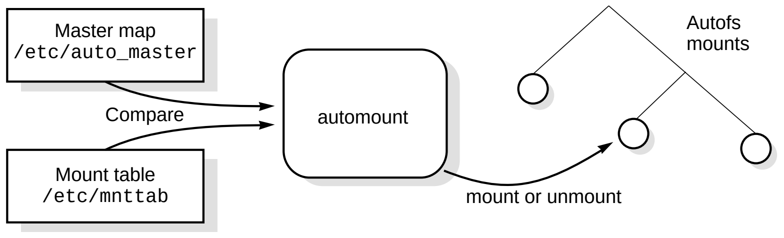

Figure 5–2 Navigation Through the Master Map

Autofs Mount Process

What the autofs service does when a mount request is triggered depends on how the automounter maps are configured. The mount process is generally the same for all mounts, but the final result changes with the mount point specified and the complexity of the maps. Starting with the Solaris 2.6 release, the mount process has also been changed to include the creation of the trigger nodes.

A Simple Autofs Mount

To help explain the autofs mount process, assume that the following files are installed.

```
$ cat /etc/auto_master
# Master map for automounter
#
+auto_master
/net        -hosts        -nosuid,nobrowse
/home       auto_home     -nobrowse
/xfn        -xfn
/share      auto_share
$ cat /etc/auto_share
# share directory map for automounter
#
ws          gumbo:/export/share/ws
```

When the /share directory is accessed, the autofs service creates a trigger node for /share/ws, which can be seen in /etc/mnttab as an entry that resembles the following entry:

```
-hosts  /share/ws     autofs  nosuid,nobrowse,ignore,nest,dev=###
```

When the /share/ws directory is accessed, the autofs service completes the process with these steps:

1. Pings the server's mount service to see if it's alive
2. Mounts the requested file system under /share. Now /etc/mnttab file contains the following entries:

```
-hosts  /share/ws     autofs  nosuid,nobrowse,ignore,nest,dev=###
gumbo:/export/share/ws /share/ws   nfs   nosuid,dev=####    #####
```

Hierarchical Mounting

When multiple layers are defined in the automounter files, the mount process becomes more complex. If the /etc/auto_shared file from the previous example is expanded to contain:

```
# share directory map for automounter
#
ws       /       gumbo:/export/share/ws
         /usr    gumbo:/export/share/ws/usr
```

The mount process is basically the same as the previous example when the /share/ws mount point is accessed. In addition, a trigger node to the next level (/usr) is created in the /share/ws file system so that the next level can be mounted if it is accessed. In this example, /export/share/ws/usr must exist on the NFS server for the trigger node to be created.

Caution - Do not use the –soft option when specifying hierarchical layers. Refer to "Autofs Unmounting" on page 104 for an explanation of this limitation.

Autofs Unmounting

The unmounting that occurs after a certain amount of idle time is from the bottom up (reverse order of mounting). If one of the directories at a higher level in the hierarchy is busy, only file systems below that directory are unmounted. During the unmounting process, any trigger nodes are removed and then the file system is unmounted. If the file system is busy, the unmount fails and the trigger nodes are reinstalled.

Caution - Do not use the `–soft` option when specifying hierarchical layers. If the `–soft` option is used, requests to reinstall the trigger nodes can timeout. The failure to reinstall the trigger notes leaves no access to the next level of mounts. The only way to clear this problem is to have the automounter unmount all of the components in the hierarchy, either by waiting for the file systems to be automatically unmounted or by rebooting the system.

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 any of the replicated locations can provide the same service to any user. This procedure makes sense only when you mount a file system that is read-only, as 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 is used automatically without any effort required by the user.

If the file systems are configured as replicas (see “What Is a Replicated File System?” on page 64), then the clients have the advantage of using failover. Not only is the best server automatically determined, but if that server becomes unavailable, the client automatically uses the next-best server. Failover is a new feature implemented in the Solaris 2.6 release.

An example of a good file system to configure as a replica is man pages. In a large network, more than one server can 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
```

Here you can mount the man pages from the servers `oak`, `rose`, or `willow`. Which server is best depends on a number of factors including: the number of servers supporting a particular NFS protocol level, the proximity of the server, and weighting.

During the sorting process, a count of the number of servers supporting the NFS version 2 and NFS version 3 protocols is made. Whichever protocol is supported on the most servers becomes the protocol supported by default. This provides the client with the maximum number of servers to depend on.

Once the largest subset of servers with the same protocol version is found, that server list is sorted by proximity. Servers on the local subnet are given preference over servers on a remote subnet. The closest server is given preference, which reduces latency and network traffic. Figure 5–3 illustrates server proximity.

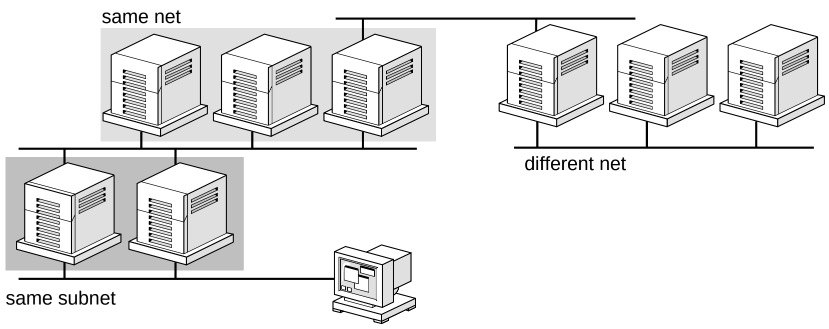

Figure 5–3 Server Proximity

If several servers supporting the same protocol are on the local subnet, the time to connect to each server is determined and the fastest is used. The sorting can also be influenced by using weighting (see “Autofs and Weighting” on page 107).

If version 3 servers are more abundant, the sorting process becomes more complex. Normally, servers on the local subnet are given preference over servers on a remote subnet. A version 2 server can complicate matters, as it might be closer than the nearest version 3 server. If there is a version 2 server on the local subnet and the closest version 3 server is on a remote subnet, the version 2 server is given preference. This preference is only checked if there are more version 3 servers than version 2 servers. If there are more version 2 servers, then only a version 2 server is selected.

With failover, the sorting is checked once at mount time to select one server from which to mount, and again anytime the mounted server becomes unavailable. Multiple locations are useful in an environment where individual servers might not export 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.

Autofs and Weighting

You can influence the selection of servers at the same proximity level by adding a weighting value to the autofs map. For example:

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

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 - All other server selection factors are more important than weighting. Weighting is only considered when selecting between servers with the same network proximity.

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 5–1 shows the predefined map variables.

TABLE 5–1 Predefined Map Variables

Variable	Meaning	Derived From	Example
`ARCH`	Architecture type	`uname -m`	`sun4`
`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, the clients can mount through a map entry like the following:

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

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

Note - Most applications written for any of the sun4 architectures can run on all sun4 platforms, so the –`ARCH` variable is hardcoded to `sun4` instead of `sun4m` or `sun4c`.

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, such as `xfn` or `hosts`.

This name service switch file contains an entry for autofs labeled as `automount`, which contains the order in which the name services are searched. The following file 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
```

(continued)

```
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
```

In this example, the local maps are searched before the NIS maps, so 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/bill
bonny             cs.csc.edu:/export/home/bonny
```

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

```
bill              cs.csc.edu:/export/home/bill
bonny             cs.csc.edu:/export/home/bonny
+auto_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
```

Note - You cannot use + entries in NIS+ or NIS maps.

Executable Autofs Maps

You can create an autofs map that will execute some commands to generate the autofs mount points. You could benefit from using an executable autofs map if you need to be able to create the autofs structure from a database or a flat file. The disadvantage to using an executable map is that the map will need to be installed on each host. An executable map cannot be included in either the NIS or the NIS+ name service.

The executable map must have an entry in the `auto_master` file.

```
/execute    auto_execute
```

Here is an example of an executable map:

```
#!/bin/ksh
#
# executable map for autofs
#

case $1 in
          src)  echo '-nosuid,hard bee:/export1' ;;
esac
```

For this example to work, the file must be installed as `/etc/auto_execute` and must have the executable bit set (set permissions to `744`). Under these circumstances running the following command:

```
% ls /execute/src
```

causes the `/export1` file system from `bee` to be mounted.

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.

Default Autofs Behavior With Name Services

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

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 name that contains one or more underscores, the underscores are changed to dots, which allows the old NIS file names to work. Then autofs looks up the map again, as shown in Figure 5–4.

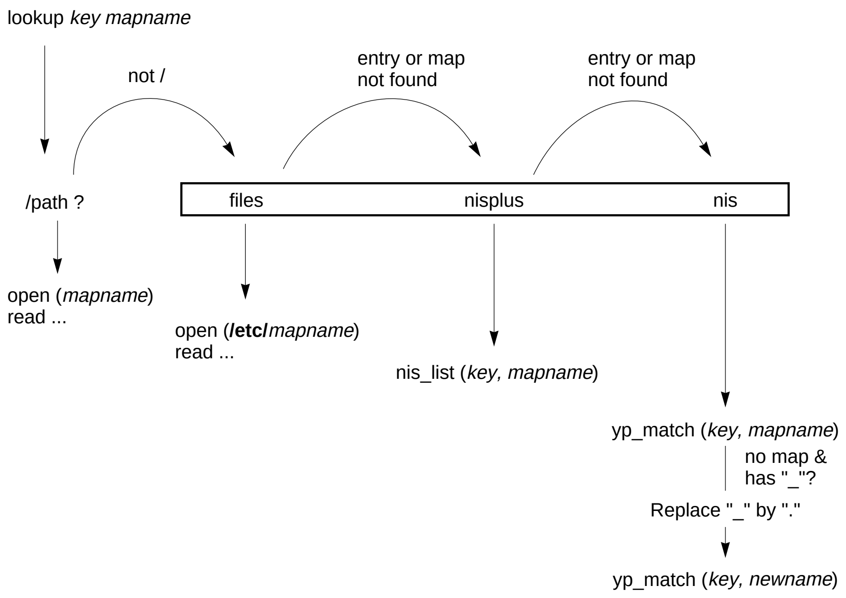
lookup *key mapname*
not /
entry or map not found
entry or map not found
/path ?
files
nisplus
nis
open (*mapname*) read ...
open (**/etc/***mapname*) read ...
nis_list (*key, mapname*)
yp_match (*key, mapname*)
no map & has "_"?
Replace "_" by "."
yp_match (*key, newname*)

Figure 5–4 How Autofs Uses the Name Service

The screen activity for this session would look like the following example.

```
$ grep /home /etc/auto_master
/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 previous map 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 (*)

You can use the universal substitute character, the asterisk (*), to match any key. You could mount the `/export` file system from all hosts 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 might 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 - "
```

APPENDIX **A**

NFS Tunables

You can set several parameters that can improve the functioning of the NFS service. You can define these parameters 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 that identifies it.

Note - 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 the SunOS release that you are running, before making changes or applying values from previous releases.

Table A–1 lists the parameters that are part of the nfs module. Table A–2 lists the parameters that are part of the nfssrv module. Table A–3 lists the parameters that are part of the rpcmod module. "How to Set the Value of a Kernel Parameter" on page 119 shows how to change these parameters. See the **`system`**(4) man page for information about the `/etc/system` file.

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.

TABLE A–1 NFS Parameters for the nfs Module *(continued)*

Symbol Name	Description	Default Setting
`nfs_acl_cache`	This symbol controls whether ACLs are cached on clients that are using the NFS_ACL protocol.	Defaults to off (0). You probably can safely enable this symbol (1), which might be in future Solaris releases.
`nfs_cots_timeo`	This symbol controls the default timeout value of NFS version 2 client operations over connection-oriented transports.	Defaults to 600 tenths of a second.
`nfs3_cots_timeo`	This symbol controls the default timeout value of NFS version 3 client operations over connection-oriented transports.	Defaults to 600 tenths of a second.
`nfs_do_symlink_cache`	This symbol controls whether symbolic links are cached for file systems mounted using NFS version 2 software.	Defaults to on (1). You can disable this symbol (0) if something like amd is to be used on the system. Client system performance might be reduced if this symbol is disabled.
`nfs3_do_symlink_cache`	This symbol controls whether symbolic links are cached for file systems mounted using NFS version 3 software.	Defaults to on (1). You can disable this symbol (0) but client system performance might be reduced.
`nfs_dynamic`	This symbol controls whether dynamic retransmission support is used for file systems mounted using NFS version 2 software.	Defaults to on (1). You can safely turn off this symbol (0), with possible interoperability problems with servers that are slow or cannot support full 8 KB read or write transfers.
`nfs3_dynamic`	This symbol controls whether dynamic retransmission support is used for file systems mounted using NFS version 3 software.	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 software.	Defaults to off (0). You can probably safely enable this symbol (1) but it might negatively impact normal directory name caching.

TABLE A–1 NFS Parameters for the nfs Module *(continued)*

Symbol Name	Description	Default Setting
`nfs3_lookup_neg_cache`	This symbol controls whether failed lookup requests are cached for file systems mounted using NFS version 3 software.	Defaults to off (0). You can probably safely enable this symbol (1) but it might negatively impact normal directory name caching.
`nfs_max_threads`	This symbol controls the maximum number of async threads started per file system mounted using NFS version 2 software.	Defaults to 8. Because this number affects the number of threads per file system, on a client 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 software.	Defaults to 8. Because this number affects the number of threads per file system, on a client with many file systems a large change could severely degrade performance.
`nfs3_max_transfer_size`	This symbol controls the NFS version 3 client file block size.	Defaults to 32 KB. Strongly recommend that it not be changed.
`nfs_nra`	This symbol controls the number of read-ahead blocks that are read for file systems mounted using NFS version 2 software.	Defaults to 4. A higher value might not increase performance, but will cause increased memory utilization on the client.
`nfs3_nra`	This symbol controls the number of read-ahead blocks that are read for file systems mounted using NFS version 3 software.	Defaults to 4. A higher value might not increase performance, but will cause 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. You can set this symbol to 1 to disable caching.
`nfs_shrinkreaddir`	This symbol controls whether over-the-wire NFS Version 2 READDIR requests are shrunk to 1024 bytes. Some 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. You can safely enable this symbol (1) but it might negatively impact system performance while reading directories.

TABLE A–1 NFS Parameters for the nfs Module *(continued)*

Symbol Name	Description	Default Setting
`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 be 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). You can enable this symbol (1), but problems with interoperability might 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 to safely increase write throughput.	Defaults to on (1). You can disable this symbol (0), but performance might be reduced.

TABLE A–3 NFS Parameters for the rpcmod Module

Symbol Name	Description	Default Setting
`authdes_cachesz`	This symbol controls the size of the authdes reply cache. It is a performance enhancement feature, to avoid verifying client credentials on every secure RPC request.	Defaults to 128. System performance might be reduced if this value is set too high.
`authkerb_cachesz`	This symbol controls the size of the authkerb reply cache. It is a performance enhancement feature, to avoid verifying client credentials on every secure RPC request.	Defaults to 128. System performance might be reduced if this value is set too high.
`svc_ordrel_timeout`	This symbol lists the number of milliseconds after which the kernel will force a connection tear–down to complete. It is used in rare cases when a TCP connection that is used for kernel RPC gets hung while being torn down. The connection can get hung when a NFS server initiates a graceful close (FIN) of a connection, and a client fails to complete the close (FIN ackowledgement) handshake.	Defaults to 600000 ms (10 minutes). If the value is too small, then the server will not allow clients enough time to properly tear down TCP connections. If the value is too large, then a buggy or malicious client could tie up TCP connections on the server.

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 that 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 read-ahead blocks that are read for file systems mounted using NFS version 2 software.

3. **Reboot the system.**

Index