ONC3/NFS™ Administrator's Guide

Document Number 007-0850-070

CONTRIBUTORS

Written by Kim Simmons, Pam Sogard, Susan Ellis, and Susan Thomas
Illustrated by Dan Young
Production by Julia Lin
Engineering contributions by Andrew Cherenson, Dana Treadwell, James Yarbrough, and Jon Livesey

ONC3 / NFS™ Administrator's Guide
Document Number 007-0850-070

Contents

List of Figures

List of Tables

About This Guide

The *ONC3/NFS Administrator's Guide* documents the Silicon Graphics® Open Network Computing / Network File System (ONC3 / NFS). ONC3 / NFS is adapted from Sun Microsystems, Inc.'s ONC+ version 1.2, and was previously referred to as the Network File System (NFS). The purpose of this guide is to provide the information needed to set up and maintain the ONC3 / NFS services. It explains ONC3 / NFS software fundamentals and provides procedures to help you install, test, and troubleshoot ONC3 / NFS on your network. It also contains planning information and recommendations for administering the service.

ONC+ has been optimized for use on Silicon Graphics systems, and has been integrated with the IRIS Indigo Magic™ environment and system toolchest. The Silicon Graphics implementation of ONC+ can run only on a Silicon Graphics system.

ONC3 / NFS is made up of distributed services that allow users to access file systems and directories on remote systems and treat them as if they were local. Networks with heterogeneous architectures and operating systems can participate in the same ONC3 / NFS service. The service can also include systems connected to different types of networks.

The components of ONC3 / NFS are described below. Further information is provided in the following chapters.

NFS	The Network File System (NFS) is the distributed file system in ONC3 / NFS.
CacheFS	The Cache File System (CacheFS) is a new file system type in IRIX 5.3 that provides client-side caching for NFS and other file system types.
NIS	The Network Information Service (NIS) is a collection of databases of network entity location information that can be used by applications, including NFS.

Summary of Contents

Table i contains a summary of each chapter in this guide and suggests how to use the chapter.

Table i *ONC3/NFS Administrator's Guide* Chapter Summaries

Chapter	Summary	When to Read
Chapter 1, "Understanding ONC3/NFS"	Introduces the vocabulary of ONC3/NFS, and the fundamentals of ONC3/NFS operation.	Read this chapter if you are new to ONC3/NFS. If you already have ONC3/NFS experience, you can skip Chapter 1.
Chapter 2, "Planning ONC3/NFS Service"	Explains ONC3/NFS processes and their options in detail.	You should be thoroughly familiar with the information in this chapter before continuing with Chapter 4, "Setting Up and Testing ONC3/NFS."
Chapter 3, "Using Automount Map Options"	Describes special features of the automounter.	Read this chapter if you plan to customize your automount environment.
Chapter 4, "Setting Up and Testing ONC3/NFS"	Contains procedures for implementing ONC3/NFS on server and client systems and verifying their operation.	Use this chapter as a guide to implementing the ONC3/NFS service on your network.
Chapter 5, "Maintaining ONC3/NFS"	Contains procedures for changing the parameters in ONC3/NFS after it is in service.	Use these procedures for routine upkeep of ONC3/NFS.

Table i (continued) *ONC3/NFS Administrator's Guide* Chapter Summaries

Chapter	Summary	When to Read
Chapter 6, "Troubleshooting ONC3/NFS"	Provides general problem-solving information and check-out procedures. Also describes specific problems that can occur with ONC3/NFS and suggests what you can do to correct them.	Use this chapter to diagnose and correct ONC3/NFS problems. Some of the suggestions in this chapter require an understanding of other network software, such as NIS.
Appendix A, "ONC3/NFS Error Messages"	Explains the interactions between the ONC3/NFS lock manager and IRIX kernel. It also gives a detailed description of ONC3/NFS daemons.	Read this appendix if you need a detailed understanding of how the lock manager or ONC3/NFS daemon processes work.

What You Should Know

To use the setup and maintenance information in this guide, you should have experience in the following areas:

- Setting up network services
- Assessing the needs of network users
- Maintaining hosts databases
- Understanding the UNIX® file system structure
- Using UNIX editors

To troubleshoot ONC3/NFS, you should be familiar with these concepts:

- Theory of network services
- Silicon Graphics network implementation

Supplementary Documentation

You can find supplementary information in these documents:

- *IRIX Advanced Site and Server Administration Guide* (Silicon Graphics publication) explains the fundamentals of system and network administration for Silicon Graphics systems on a local area network.
- *NIS Administration Guide* (Silicon Graphics publication) explains how to set up and maintain Silicon Graphics implementation of the network information service.
- *IRIX Network Programming Guide* (Silicon Graphics publication) explains the programmatic interfaces to ONC3/NFS.
- *Diskless Workstation Administration Guide* (Silicon Graphics publication) describes the setup and maintenance of diskless workstations.
- *Defense Data Network Protocol Handbook*, available from the Network Information Center, 14200 Park Meadow Dr., Suite 200, Chantilly, VA 22021. This three-volume set contains information on TCP/IP and UDP/IP.
- Stern, Hal *Managing NFS and NIS* O'Reilly & Associates, Inc. 1991. This book contains detailed, but not Silicon Graphics-specific, information about NFS and how to administer and use it.

Typographical Conventions

These type conventions and symbols are used in this guide:

Italics	Filenames, variables, IRIX command arguments, command flags, host names, user IDs, map names, the first use of new terms, titles of publications, icon names
`Screen type`	Code examples, file excerpts, and screen displays (including error messages), and */etc/export* and */etc/fstab* options
`Bold Screen type`	User input

()	(Parentheses) Following IRIX commands, they surround the reference page (man page) section where the command is described
[]	(Brackets) Surround optional syntax statement arguments
#	IRIX shell prompt for the superuser (*root*)

Product Support

Silicon Graphics offers a comprehensive product support and maintenance program for its products. For information about using support services for this product, refer to the *Release Notes* that accompany it.

Chapter 1

Understanding ONC3/NFS

This chapter introduces the Silicon Graphics implementation of the Sun Microsystems Open Network Computing Plus (ONC+) distributed services, which was previously referred to as Network File System (NFS). In this guide, NFS refers to the distributed file system in ONC3/NFS.

The information in this chapter is prerequisite to successful ONC3/NFS administration. It defines ONC3/NFS and its relationship to other network software, introduces the ONC3/NFS vocabulary, and identifies the software elements that support ONC3/NFS operation. It also explains special utilities and implementation features of ONC3/NFS. You should be familiar with the information in this chapter before setting up or modifying the ONC3/NFS environment.

The components of ONC3/NFS are described below.

NFS	The distributed file system in ONC3/NFS. It contains the automounter and lock manager. ONC3/NFS includes the new level of the NFS protocol, NFS3, an optimized version of NFS, designed to be transparent to users. NFS is multithreaded to take advantage of multiprocessor performance.
CacheFS	The Cache File System (CacheFS) is a new file system type in IRIX 5.3 that provides client-side caching for NFS and other file system types. Using CacheFS on NFS clients with local disk space can significantly increase the number of clients a server can support and reduce the data access time for clients using read-only file systems.
NIS	The network information service (NIS) is a database of network entity location information that can be used by NFS. Information about NIS is published in a separate volume called the *NIS Administration Guide*.

This chapter contains these sections:

What Is NFS?

NFS is a network service that allows users to access file hierarchies across a network and treat them as if they were local. File hierarchies can be entire file systems or individual directories. Systems participating in the NFS service can be heterogeneous. They may be manufactured by different vendors, use different operating systems, and be connected to networks with different architectures. These differences are transparent to the NFS application.

NFS is an application layer service that can be used on any network running the Transmission Control Protocol (TCP) or User Datagram Protocol (UDP). It relies on *remote procedure call*s (RPC) for session layer services and *external data representation* (XDR) for presentation layer services.

XDR is a library of routines that translate data formats between processes.

Figure 1-1 illustrates the NFS software implementation in the context of the Open Systems Interconnect (OSI) model.

application	NFS
presentation	XDR
session	RPC
transport	UDP/TCP
network	IP
data link	network interface
physical	

Figure 1-1 NFS Software Implementation

NFS and Diskless Workstations

It is possible to set up a system so that all the required software, including the operating system, is supplied from remote systems by means of the NFS service. Workstations operating in this manner are considered *diskless workstations*, even though they may be equipped with a local disk.

Instructions for implementing diskless workstations are given in the *Diskless Workstation Administration Guide*. However, it is important to acquire a working knowledge of NFS before setting up a diskless system.

The Cache File System

A *cache* is a temporary storage area for data. The Cache File System (CacheFS) enables you to use local disk drives on workstations to store frequently used data from a remote file system or CD-ROM. The data stored on the local disk is the cache.

When a file system is cached, the data is read from the original file system and stored on the local disk. The reduction in network traffic improves performance. If the remote file system is on a storage medium with slower response time than the local disk (such as a CD-ROM), caching provides an additional performance gain.

CacheFS can use all or part of a local disk to store data from one or more remote file systems. A user accessing a file does not need to know whether the file is stored in a cache or is being read from the original file system. The user opens, reads, and writes files as usual.

NFS and the Network Information Service

The Network Information Service (NIS) is a database service that provides location information about network entities to other network servers and applications, such as NFS. NFS and NIS are independent services that may or may not be operating together on a given network. On networks running NIS, NFS may use the NIS databases to locate systems when NIS queries are specified.

Client-Server Fundamentals

In an NFS transaction, the workstation requesting access to remote directories is known as the *client*. The workstation providing access to its local directories is known as the *server*. A workstation can function as a client and a server simultaneously. It can allow remote access to its local file systems while accessing remote directories with NFS. The client-server relationship is established by two complementary processes, *exporting* and *mounting*.

Exporting

Exporting is the process by which an NFS server provides access to its file resources to remote clients. Individual directories, as well as file systems, can be exported, but exported entities are usually referred to as file systems. Exporting is done either during the server's boot sequence or from a command line as superuser while the server is running.

Once a file system is exported, any authorized client can use it. A list of exported file systems, client authorizations, and other export options are specified in the */etc/exports* file (see "/etc/exports and Other Export Files" in Chapter 2 for details). Exported file systems are removed from NFS service by a process known as *unexporting*.

A server can export any file system or directory that is local. However, it cannot export both a parent and child directory within the same file system; to do so is redundant.

For example, assume that the file system */usr* contains the directory */usr/demos*. As the child of */usr*, */usr/demos* is automatically exported with */usr*. For this reason, attempting to export both */usr* and */usr/demos* generates an error message that the parent directory is already exported. If */usr* and */usr/demos* were separate file systems, this example would be valid.

Mounting

Mounting is the process by which file systems, including NFS file systems, are made available to the IRIX operating system and consequently, the user. When NFS file systems or directories are mounted, they are made available to the client over the network by a series of remote procedure calls that enable the client to access the file system transparently from the server's disk. Mounted NFS directories or file systems are not physically present on the client system, but the mount looks like a local mount and users enter commands as if the file systems were local.

NFS clients can have directories mounted from several servers simultaneously. Mounting can be done as part of the client's boot sequence, automatically, at file system access, with the help of a user-level daemon, or with a superuser command after the client is running. When mounted directories are no longer needed, they can be relinquished in a process known as *unmounting*.

Like locally mounted file systems, NFS mounted file systems and directories can be specified in the */etc/fstab* file (see "/etc/fstab and Other Mount Files" in Chapter 2 for details). Since NFS file systems are located on remote systems, specifications for NFS mounted resources must include the name of the system where they reside.

Mount Points

The access point in the client file system where an NFS directory is attached is known as a *mount point*. A mount point is specified by a conventional IRIX pathname.

Figure 1-2 illustrates the effect of mounting directories onto mount points on an NFS client.

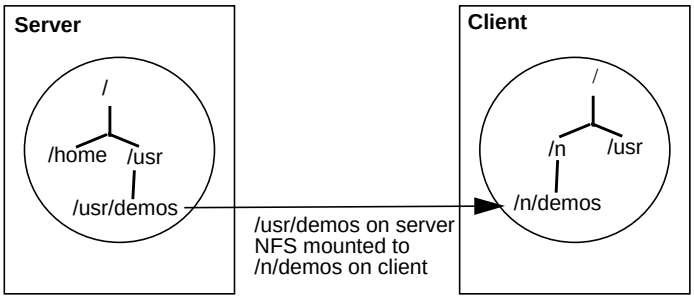

Figure 1-2 Sample Mounted Directory

The pathname of a file system on a server can be different from its mount point on the client. For example, in Figure 1-2 the file system */usr/demos* is mounted in the client's file system at mount point */n/demos*. Users on the client gain access to the mounted directory with a conventional *cd*(1) command to */n/demos*, as if the directory were local.

Mount Restrictions

NFS does not permit *multihopping*, mounting a directory that is itself NFS mounted on the server. For example, if *host1* mounts */usr/demos* from *host2*, *host3* cannot mount */usr/demos* from *host1*. This would constitute a *multihop*.

NFS also does not permit *loopback mounting*, mounting a directory that is local to the client via NFS. For example, the local file system */usr* on *host1* cannot be NFS mounted to *host1*, this would constitute a loopback mount.

Automatic Mounting

As an alternative to standard mounting via */etc/fstab* or the *mount* command, NFS provides an automatic mounting feature called the automounter, or *automount*. The automounter dynamically mounts file systems when they are referenced by any user on the client system, then unmounts them after a specified time interval. Unlike standard mounting, *automount*(1M), once set up, does not require superuser privileges to mount a remote directory. It also creates the mount points needed to access the mounted resource. NFS servers cannot distinguish between directories mounted by the automounter from those mounted by conventional mount procedures.

Unlike the standard mount process, *automount* does not read the */etc/fstab* file for mount specifications. Instead, it reads alternative files (either local or through NIS) known as *maps* for mounting information (see "automount Files and Maps" in Chapter 2 for details). It also provides special maps for accessing remote systems and automatically reflecting changes in the */etc/hosts* file and any changes to the remote server's */etc/exports* file.

Default configuration information for automounting is contained in the file */etc/config/automount.options*. This file can be modified to use different options and more sophisticated maps.

automount Restrictions

CacheFS file systems cannot be automatically mounted with *automount*.

Stateless Protocol

NFS implements a *stateless protocol* in which the server maintains almost no information on NFS processes. This stateless protocol insulates clients and servers from the effects of failures. If a server fails, the only effect to clients is that NFS data on the server is unavailable to clients. If a client fails, server performance is not affected.

Clients are independently responsible for completing NFS transactions if the server or network fails. By default, when a failure occurs, NFS clients continue attempting to complete the NFS operation until the server or network recovers. To the client, the failure can appear to be slow

performance on the part of the server. Client applications continue retransmitting until service is restored and their NFS operations can be completed. If a client fails, no action is needed by the server or its administrator in order for the server to continue operation.

The major advantage of a stateless server is robustness in the face of client, server, or network failures. This robustness is especially important in a complex network of heterogeneous systems, many of which are not under the control of a centralized operations staff, and some of which are systems that are often rebooted without warning.

Input/Output Management

In NFS transactions, data input and output is asynchronous read-ahead and write-behind, unless otherwise specified. As the server receives data, it notifies the client that the data was successfully written. The client responds by freeing the blocks of NFS data successfully transmitted to the server. In reality, however, the server might not write the data to disk before notifying the client, a technique called *delayed writes*. Writes are done when they are convenient for the server, but at least every 30 seconds.

Although delayed write is the default method of operation for NFS, synchronous writes are also an option (see "/etc/exports Options" in Chapter 2 for more details about NFS options). With synchronous writes, the server writes the data to disk before notifying the client that it has been written. Synchronous writes may slow NFS performance due to the time required for disk access, but increase data integrity in the event of system or network failure.

NFS File Locking Service

To help manage file access conflicts and protect NFS sessions during failures, NFS offers a file and record locking service called the *network lock manager*. The network lock manager is not an integral part of NFS. It is a separate service NFS makes available to user applications with the facility to use it. To use the locking service, applications must make calls to standard IRIX lock routines (*fcntl*(2), *flock*(3B), and *lockf*(3C)). For NFS files, these calls are received by the network lock manager process (*lockd*(1M)).

The network lock manager processes must run on both the client and the server to function properly. Communication between the two processes is by means of RPC. Calls for service issued to the client process are handed to the server process, which uses its local IRIX locking utilities to handle the call. If the file is in use, the lock manager issues an advisory to the calling application, but it does not prevent the application from accessing a busy file. The application must determine how to respond to the advisory, using its own facilities.

There are four basic kernel-to-lock manager requests:

KLM_LOCK Lock the specified record.

KLM_UNLOCK
Unlock the specified record.

KLM_TEST Test if the specified record is locked.

KLM_CANCEL
Cancel an outstanding lock request.

Despite the fact that the network lock manager adheres to *lockf*/*fcntl* semantics, its operating characteristics are influenced by the nature of the network, particularly during crashes.

Locking and Crash Recovery

As part of the file locking service, the network lock manager assists with crash recovery by maintaining state information on locked files. It uses this information to reconstruct locks in the event of a server or client failure.

When an NFS client goes down, the lock managers on all of its servers are notified by their status monitors, and they simply release their locks, on the assumption that the client will request them again when it wants them. When a server crashes, however, matters are different. When the server comes back up, its lock manager gives the client lock managers a grace period to submit lock reclaim requests. During this period, the lock manager accepts only reclaim requests. The client status monitors notify their respective lock managers when the server recovers. The default grace period is 45 seconds.

After a server crash, a client may not be able to recover a lock that it had on a file on that server, because another process may have beaten the recovering application process to the lock. In this case the SIGLOST signal is sent to the process (the default action for this signal is to kill the application).

The local lock manager does not reply to the kernel lock request until the server lock manager has responded to it. Further, if the lock request is on a server new to the local lock manager, the lock manager registers its interest in that server with the local status monitor and waits for its reply. Thus, if either the status monitor or the server's lock manager is unavailable, the reply to a lock request for remote data is delayed until it becomes available.

Locking and the Network Status Monitor

To handle crash recoveries, the network lock manager relies on information provided by the *network status monitor*. The network status monitor is a general service that provides information about network systems to network services and applications. The network status monitor notifies the network lock manager when a network system recovers from a failure, and by implication, that the system failed. This notification alerts the network lock manager to retransmit lock recovery information to the server.

To use the network status monitor, the network lock manager registers with the status monitor process (*statd*(1M)) the names of clients and servers for which it needs information. The network status monitor then tracks the status of those systems and notifies the network lock manager when one of them recovers from a failure.

Chapter 2

Planning ONC3/NFS Service

To plan the ONC3/NFS service for your environment, it is important to understand how ONC3/NFS processes work and how they can be configured. This chapter provides prerequisite information on ONC3/NFS processes and their configuration options. It also explains the conditions under which certain options are recommended.

This chapter contains these sections:

The Export Process

Access to files on an NFS server is provided by means of the *exportfs*(1M) command. The *exportfs* command reads the file */etc/exports*(4) for a list of file systems and directories to be exported from the server. Normally, *exportfs* is executed at system startup by the */etc/init.d/network* script. It can also be executed by the superuser from a command line while the server is running. Exported file systems must be local to the server. A file system that is NFS-mounted from another server cannot be exported (see "Mount Restrictions" in Chapter 1 regarding *multihop*).

exportfs Command Options

The *exportfs* command has several options used to configure its operation. Four of these options are briefly described below. For more complete information on *exportfs* options, see the *exportfs*(1M) manual page.

-a (all) Export all resources listed in */etc/exports*.

-i (ignore) Do not use the options set in the */etc/exports* file.

-u (unexport) Terminate exporting designated resources.

-v (verbose) Display any output messages during execution.

Invoking *exportfs* without options reports the file systems that are currently exported.

/etc/exports and Other Export Files

Exporting starts when *exportfs* reads the file */etc/exports*(4) for a list of file systems and directories to be exported from the server. As it executes, *exportfs* writes a list of file systems it successfully exported, and information on how they were exported, in the */etc/xtab*(4) file. Anytime the */etc/exports* file is changed, *exportfs* must be executed to update the */etc/xtab* file. If an entry is not listed in */etc/xtab*, it has not been exported, even if it is listed in */etc/exports*.

In addition to the */etc/xtab* file, the server maintains a record of the exported resources that are currently mounted and the names of clients that have mounted them. The record is maintained in a file called */etc/rmtab*. Each time a client mounts a directory, an entry is added to the server's */etc/rmtab* file. The entry is removed when the directory is unmounted. The information contained in the */etc/rmtab* file can be viewed using the *showmount*(1M) command.

Note: The information in */etc/rmtab* may not be current, since clients can unmount file systems without informing the server.

/etc/exports Options

There are a number of export options for managing the export process. Some commonly used export options are briefly described below. For a complete explanation of options, see the *exports*(4) manual page.

`ro`	(read only) Export this file system with read-only privileges.
`rw`	(read, write) Export this file system with read and write privileges. `rw` is the default.
`rw=`	(read mostly) Export this file system read-only to all clients except those listed. **Note:** Directories are exported either `ro` or `rw`, not both ways. The option specified first is used. ♦
`anon=`	(anonymous UID) If a request comes from the user *root* (UID = 0), use the specified UID as the effective UID instead. By default, the effective UID is `nobody` (UID = –2). Specifying a UID of –1 disables access by unknown users or by *root* on a host not specified by the `root` option. Use the `root` option to permit accesses by *root*.
`root=`	Give superuser privileges to *root* users of NFS-mounted directories on systems specified in root access list. By default, `root` is set to `none`.
`access=`	Grant mount privileges to a specified list of clients only. Clients can be listed individually or as an NIS netgroup (see *netgroup*(4)).
`nohide`	(IRIX enhancement) By default, the contents of a child file system are hidden when only the parent file system is mounted. Allow access to this file system if its parent file system is mounted.
`wsync`	(IRIX enhancement) Perform all write operations to disk before sending an acknowledgment to the client. Overrides delayed writes. (See "Input/Output Management" in Chapter 1 for details.)

When a file system or directory is exported without specifying options, the default options are `rw` and `anon=nobody`.

Sample */etc/exports* File

A default version of the */etc/exports* file is shipped with NFS software and stored in */etc/exports* when NFS is installed. You must add your own entries to the default version as part of the NFS setup procedure (given in "Setting Up the NFS Server" in Chapter 4). This sample */etc/exports* illustrates entries and how to structure them with various options:

```
/                       -ro
/reports                -access=finance,rw=susan
/usr                    -nohide
/usr/demos              -ro,access=client1:client2:client3
/usr/catman             -nohide
```

In this sample */etc/exports*, the first entry exports the root directory (/) with read-only privileges. The second entry exports a separate file system, */reports* read-only to the netgroup *finance*, with write permission specified for *susan*. Users who mount */usr* can access the */usr/demos* file system because `nohide` is specified.

The fourth entry uses the access list option. It specifies that *client1*, *client2*, and *client3* are authorized to access */usr/demos* with read-only privileges. To avoid possible problems, *client1*, *client2*, and *client3* should be fully qualified domain names (as returned by *hostname*(1)).

Note: If you are using an access list to export to a client with multiple network interfaces, the */etc/exports* file must contain all names associated with the client's interfaces. For example, a client named *octopus* with two interfaces needs two entries in the */etc/exports* file, typically *octopus* and *gate-octopus*. ♦

The fifth entry is an example of an open file system. It exports */usr/catman* to the entire world with read-write access (the default when neither `ro` or `rw` is specified) to its contents. Activities performed as superuser on */usr/catman* files have no effect, since `anon` is not specified.

Export Restrictions

CacheFS file systems cannot be exported.

Recommendations for Exporting

Consider these suggestions for setting up exports on your NFS service:

1. Use the `ro` option unless clients must write to files. This reduces accidental removal or changes to data.
2. In secure installations, set `anon` to **–1** to disable *root* on any client, except those specified in the `root` option, from accessing the designated directory as *root*.
3. Be cautious with your use of the `root` option.
4. If you are using NIS, consider using netgroups for long `access` lists.
5. Use `nohide` to export related but separate file systems to minimize the number of mounts clients must perform.
6. Use `wsync` when minimizing risk to data is more important than optimizing performance.

The */etc/fstab* Mount Process

An NFS client mounts directories at startup via */etc/fstab* entries, or by executing the *mount*(1M) command. The *mount* command can be executed during the client's boot sequence, from a command line entry, or graphically, using the System Manager tool. The *mount* command supports the NFS3 protocol if that protocol is also running on the server.

Mounts must reference directories that are exported by a network server and mount points that exist on the client. Directories that serve as mount points may or may not be empty. If using the System Manager for NFS mounting, the mount points must be empty. If the directory is not empty, the original contents are hidden and inaccessible while the NFS resources remain mounted.

mount and *umount* Command Options

The *mount* and *umount*(1M) commands have many options for customizing mounting and unmounting that can apply to either EFS or NFS file systems.

Several commonly-used options are briefly described below in their NFS context (see *mount*(1M) for full details).

-t *type* (type) Set the type of directories to be mounted or unmounted. *type* is `nfs` for NFS mounting, `nfs3` for the new NFS3 protocol, and `nfs3pref` for mounts that attempt NFS3 protocol, but fall back to `nfs` if the attempt fails. To mount NFS3, the server must support NFS3.

-a (all) Attempt to mount all directories listed in */etc/fstab*, or unmount all directories listed in */etc/mtab*.

-h *hostname* (host) Attempt to mount all directories listed in */etc/fstab* that are remote-mounted from the server *hostname*, or unmount directories listed in */etc/mtab* that are remote-mounted from server *hostname*.

-b *list* (all but) Attempt to mount or unmount all file systems listed in */etc/fstab* except those associated with the directories in *list*. *list* contains one or more comma-seperated directory names.

-o *options* (options) Use the options *options*, instead of the options in */etc/fstab*.

/etc/fstab and Other Mount Files

Mounting typically occurs when the *mount* command reads the */etc/fstab* file. Each NFS entry in */etc/fstab* contains up to six fields. An NFS entry has this format:

file_system directory type options frequency pass

where:

file_system is the remote server directory to be mounted.

directory is the mount point on the client where the directory is attached.

type is the file system type. This can be `nfs` for NFS resources, `nfs3` for the NFS3 protocol, and `nfs3pref` for mounts that try `nfs3` but fall back to `nfs` if the mount fails.

options	is mount options (see "/etc/fstab Options" in this chapter).
frequency	is always set to zero (0) for NFS and CacheFS entries.
pass	is always set to zero (0) for NFS and CacheFS entries.

The *mount* command maintains a list of successfully mounted directories in the file */etc/mtab*. When *mount* successfully completes a task, it automatically updates the */etc/mtab* file. It removes the */etc/mtab* entry when the directory is unmounted. The contents of the */etc/mtab* file can be viewed using the *mount* command without any options. See the *mount*(1M) manual page for more details.

/etc/fstab Options

There are several options for configuring mounts. When you use these options, it is important to understand that export options (specified on a server) override mount options. NFS */etc/fstab* options are briefly described below (see the *fstab*(4) manual page for complete information):

`ro`	Read-only permissions are set for files in this directory.
`rw`	Read write permissions are set for files in this directory (default).
`hard`	Specifies how the client should handle access attempts if the server fails. If the NFS server fails while a directory is hard-mounted, the client keeps trying to complete the current NFS operation until the server responds (default).
`soft`	Alternative to hard mounting. If the NFS server fails while a directory is soft-mounted, the client attempts a limited number of tries to complete the current NFS operation before returning an error.
`intr`	(interrupt) Allows NFS operations to be interrupted by users. The default setting is `off`.
`bg`	(background) Mounting is performed as a background task if the first attempt fails. The default setting is `off`.
`fg`	(foreground) Mounting is performed as a foreground task. The default setting is on.

`private`	(IRIX enhancement) Uses local file and record locking instead of a remote lock manager and minimizes delayed write flushing. Diskless clients are the primary users of this option.
`rsize`	(read size) Changes the read buffer to the size specified (default is 8K).
`wsize`	(write size) Changes the write buffer to the size specified (default is 8K).
`timeo`	(NFS timeout) Sets a new timeout limit (default is .11 seconds.)
`retrans`	(retransmit) Specifies an alternative to the number of times NFS operations are retried (default is 5).
`port`	Specifies an alternative UDP port number for NFS on the server (default port number is 2049).
`noauto`	Tells *mount* **–a** to ignore this */etc/fstab* entry.
`grpid`	Allows files created in a file system to have the parent directory's group ID, not the process' group ID.
`nosuid`	Turns setuid execution off for nonsuperusers (default is `off`).
`nodev`	Disallows access to character and block special files (default is `off`).

In addition to these options, */etc/fstab* also offers several options dedicated to attribute caching. Using these options, you can direct NFS to cache file attributes, such as size and ownership, to avoid unnecessary network activity. See the *fstab*(4) manual page for more details.

Sample */etc/fstab* File

NFS entries in */etc/fstab* are designated by the `nfs` identifier, while EFS (local file systems) entries are designated by `efs`. This sample */etc/fstab* file includes a typical NFS entry:

```
/dev/root             /          efs rw,raw=/dev/rroot 0 0
/dev/usr              /usr       efs rw,raw=/dev/rusr 0 0
redwood:/usr/demos    /n/demos   nfs ro,hard,intr,bg 0 0
```

In this example, the NFS directory */usr/demos* on server *redwood* is mounted at mount point */n/demos* on the client system with read-only (`ro`) permissions (see Figure 1-2). If the server fails after the mount has taken place, the client attempts to complete any current NFS transactions indefinitely (`hard`) or until it receives an interrupt (`intr`). Mounting executes as a background task (`bg`) if it didn't succeed the first time.

Recommendations for */etc/fstab* Mounting

Some recommendations for */etc/fstab* mounting are:

1. Use conventional mounting for clients that are inoperable without NFS directories (such as diskless workstations) and for directories that need to be mounted most of the time.
2. If directories are mounted with the `rw` (read-write) option or if they contain executable files, they should be mounted with the `hard` option. Hard mounting offers more certainty that processing will complete if the server temporarily fails.
3. The `intr` option is recommended when using a hard mount. It allows the user to break retransmission attempts if the server becomes unavailable for an extended period of time.
4. The `bg` option should always be specified to expedite the boot process if a server is unavailable when the client is booting. In other words, a client hangs until the server comes back up unless you specify `bg`.
5. If you use `nohide` when exporting file systems on the server, the client can mount the top-most directory in the exported file system hierarchy. This gives access to all related file systems while reducing individual mount calls and the complexity of the */etc/fstab* file.

   Note: A severe performance problem occurs if the `nohide` option is used when exporting an NFS back file system for a CacheFS mount. The `nohide` option creates duplicate node IDs with different file handles, causing CacheFS to remove files from the cache sooner than normal. Either avoid using the `nohide` option for NFS file systems that are used as the back file system or map CacheFS mounts to the back file system on the server one-to-one.
6. Use `private` when the NFS directory on the server is not shared between multiple NFS clients.

7. Do not put NFS mount points in the root (*/*) directory of a client. Mount points in the root directory can slow the performance of the client and can cause the client to be unusable when the server is unavailable.

The Automounter

The automount utility dynamically mounts NFS directories on a client when a user references the directory. This function is provided by the *automount* command. It can be set up to execute when a client is booted, or it can be executed by the superuser from a command line while the client is running. The automounter supports the NFS3 file system type.

To start the automounter at boot time, the *automount* flag must be set to on (see the *chkconfig*(1M) manual page for details). If the flag is on, the automounter is invoked by the */etc/init.d/network* script and started with any *automount* options specified in the */etc/config/automount.options* file.

automount Command Options

The *automount* command offers many options that allow you to configure its operation (for a complete description, see the *automount*(1M) manual page). Some commonly used options are:

-D	Assign a value to an environment variable.
-f	Read the specified local master file before the NIS master map.
-m	Do not read the NIS master map.
-M	Use the specified directory as the *automount* mount point.
-n	Disable dynamic mounts.
-T	Trace and display each NFS call.
-tl	Maintain the mount for a specified duration of client inactivity (default duration is 5 minutes).
-tm	Wait a specified interval between mount attempts (default interval is 30 seconds).

-tp	Hold information about server availability in a cache for a specified time (default interval is 5 seconds).
-tw	Wait a specified interval between attempts to unmount file systems that have exceeded cache time (default interval is 60 seconds).
-v	Display any output messages during execution.

automount Files and Maps

Just as the conventional mount process reads */etc/fstab* and writes to */etc/mtab*, *automount* can be set up to read input files for mounting information. *automount* also records its mounts in the */etc/mtab* file and removes */etc/mtab* entries when it unmounts directories.

By default, when *automount* executes at boot time, it reads the */etc/config/automount.options* file for initial operating parameters. The information contained in the */etc/config/automount.options* file can contain the complete information needed by the automounter or the information can direct *automount* to a set of files that contain customized automounting instructions. */etc/config/automount.options* cannot have comments in it.

The default version of */etc/config/automount.options* is:

```
-v /hosts -hosts -intr,nosuid,nodev
```

This */etc/config/automount.options* directs *automount* to execute with the verbose (*–v*) option. It also specifies that *automount* should use */hosts* as its daemon mount point. When a user accesses a file or directory under */hosts*, the `–hosts` argument directs automount to use the pathname component that follows `/hosts` as the name of the NFS server. All accessible file systems exported by the server are mounted to the default mount point */tmp_mnt/hosts* with the `intr`, `nosuid`, and `nodev` options.

For example, if the system *redwood* has the following entry in */etc/exports*:

```
/usr/share/catman      -ro,nohide
```

If a client system is using the default */etc/config/automount.options* file, as above, then executing the following command on the client lists the contents of the directory */usr/share/catman* on *redwood*:

```
ls -l /hosts/redwood/usr/share/catman/*
```

automount Mount Points

Mount points for *automount* serve the same function as mount points in conventional NFS mounting. They are the access point in the client's file system where a remote NFS directory is attached. There are two major differences between *automount* mount points and conventional NFS mount points.

With *automount*, mount points are automatically created and removed as needed by the *automount* program. When the *automount* program is started, it reads configuration information from */etc/config/automount.options*, additional *automount* maps, or both, and creates all mount points needed to support the specified configuration.

By default, *automount* mounts everything in the directory */tmp_mnt* and creates a link between the mounted directory in */tmp_mnt* and the accessed directory. For example, in the default configuration, mounts take place under */tmp_mnt/hosts/hostname*. The automounter creates a link from the access point */hosts/hostname* to the actual mount point under */tmp_mnt/hosts/hostname*. This command **`ls /hosts/redwood/tmp`** displays the contents of server redwood's */tmp* directory. You can change the default root mount point with the *automount –M* option.

automount Map Types

The *automount* feature uses three kinds of maps:

- master maps
- direct maps
- indirect maps

Master Maps

The master map is the first file read by the *automount* program. There is only one master map on a client. It specifies the types of supported maps, the name of each map to be used, and options that apply to the entire map (if any). By convention, the master map is called */etc/auto.master*, but the name can be changed.

For complex automount configurations, a master map can be specified in the */etc/config/automount* options file.

The master map can be a local file or an NIS database file. It contains three fields: *mount point*, *map name* and *map options*. A crosshatch (#) at the beginning of a line indicates a comment line. A sample of master map entries is:

```
#Mount Point    Map Name              Map Options
/hosts          -hosts                -intr,nosuid,nodev
/net            /etc/auto.irix.misc   -intr,nosuid
/home           /etc/auto.home        -intr,timeo=10
/-              /etc/auto.direct      -ro,intr
/net            /etc/indirect3        -ro,nfs3
```

The mount point field serves two purposes. It determines whether a map is a direct or indirect map, and it provides mount point information. A dash (/–) in the mount point field designates a direct map. It signals *automount* to use the mount points specified in the direct map for mounting this map. For example, to mount the fourth entry in the sample above, *automount* gets a mount point specification from the direct map */etc/auto.direct*. In the fifth entry, an entire indirect map, which includes all its entries, is declared to use the NFS3 protocol. If NFS3 is not available on the server, the mount fails.

A directory name in the mount point field designates an indirect map. It specifies the mount point *automount* should use when mounting this map. For example, the second entry in the sample above tells *automount* to mount the indirect map */etc/auto.irix.misc* at mount point */net*. A mount point for direct and indirect maps can be several directory levels deep.

The map name field in a master map specifies the full name and location of the map. Notice that *–hosts* is considered an indirect map whose mount point is */hosts*. The *–hosts* map mounts all the exported file systems from a server. If frequent access to just a single file system is required for a server with

many exports, it is more efficient to access that file system with a map entry that mounts just that file system.

The map options field can be used to specify any options that should apply to the entire map. Options set in a master map can be overridden by options set for a particular entry within a map.

Direct Maps

Direct maps allow mounted directories to be distributed throughout a client's local file system. They contain the information *automount* needs to determine when, what, and how to mount a remote NFS directory. You can have as many direct maps as needed.

A direct map is typically called */etc/auto.mapname*, where *mapname* is some logical name that reflects the map's contents. Direct maps can also be grouped based on logical characteristics. For instance, in the above master map example, the direct map */etc/auto.direct*, indicated by the /– mount point, can also include mounting information for software to be mounted as read-only.

All direct maps contain three fields: *directory*, *options*, and *location*. An example of an */etc/auto.direct* direct map is:

```
#Directory          Options    Location
/usr/local/tools    -nodev     ivy:/usr/cooltools
/usr/frame                     redwood:/usr/frame
/usr/games          -nosuid    peach:/usr/games
```

In a direct map, users access the NFS directory with the pathname that is identical to the directory field value in the direct map. For example, a user gives the command **`cd /usr/local/tools`** to mount */usr/cooltools* from server *ivy* as specified in the direct map */etc/auto.direct*. Notice that the directory field in a direct map can include several subdirectory levels.

The options field can be used to set options for an entry in the direct map. Options set within a map for an individual entry override the general option set for the entire map in the master map. The location field contains the NFS server's name and the remote directory to mount.

Note: When direct map mount points are mounted into routinely accessed directories, unexpected mount activity can occur. ♦

Indirect Maps

Indirect maps allow remotely mounted directories to be housed under a specified shared top-level location on the client's file system. They contain the specific information the *automount* program needs to determine when, what, and how to NFS mount a remote directory. You can have as many indirect maps as needed.

An indirect map is typically called */etc/auto.mapname*, where *mapname* is some logical name that reflects the map's contents. Indirect maps can be grouped according to logical characteristics. For example, in the master map above, the indirect map */etc/auto.home*, indicated by the mount point */home*, can include mounting information for all home directories on various servers.

Indirect maps contain three fields: *directory*, *options*, and *location*. Entries might look something like this for the */etc/auto.home* indirect map:

```
#Directory    Options     Location
willow        -intr       willow:/usr/people
pine          -nosuid     pine:/usr/people
ivy           -ro,intr    ivy:/usr/people
jinx          -ro,nfs3    jinx:/usr
```

With an indirect map, user access to an NFS directory is always relative to the mount point specified in the master map entry for the indirect map. That is, the directory is the concatenation of the mount-point field in the master map and the directory field in the indirect map. For example, given our sample */etc/auto.master* and indirect map */etc/auto.home*, a user gives the command **`cd /home/willow`** to access the NFS directory *willow:/usr/people*.

If a user changes the current working directory to the */home* directory and tries to list its contents, the directory appears empty unless a subdirectory of */home*, such as */home/willow*, was previously accessed, thereby mounting */home* subdirectories. Access to the mount point of an indirect map only shows information for mounts currently in effect; it does not trigger mounts, as with direct maps. Users must access a subdirectory to trigger a mount.

The directory field in an indirect map is limited to one subdirectory level. Additional subdirectory levels for indirect maps must be indicated in the mount point field in the master map or on the command line.

The options field can be used to set options for an entry in the indirect map. For example, the fourth entry attempts to mount */usr* using the NFS3 protocol, all other entries in the map are unaffected. Options set within a map for an entry override the general options set for the entire map in the master map. The location field contains the NFS server's name and the remote directory to mount.

Recommendations for Automounting

Some recommendations for automounting are:

1. Use the automounter when the overhead of a mount operation is not important, when a file system is used more often than the *automount* time limit (5 minutes by default, specified by the **–tl** option), or when file systems are used infrequently. Although directories that are used infrequently do not consume local or remote resources, they can slow down applications that report on file systems, such as *df*(1).
2. The default configuration in */etc/config/automount.options* is usually sufficient because it allows access to all systems. It performs the minimal number of mounts necessary when it is used in conjunction with the `nohide` export option on the server.
3. Use indirect maps whenever possible. Direct maps create more */etc/mtab* entries, which means more mounts are performed, so system overhead is increased. With indirect maps, mounts occur when a process references a subdirectory of the daemon or map mount point. With direct maps, when a process reads a directory containing one or more direct mount points, all of the file systems are mounted at the mount points. This can result in a flurry of unintended mounting activity when direct mount points are used in well-traveled directories.
4. Try not to mount direct map mount points into routinely accessed directories. This can cause unexpected mount activity and slow down system performance.
5. Use a direct rather than an indirect map when directories cannot be grouped, but must be distributed throughout the local file system.
6. Plan and test maps on a small group of clients before using them for a larger group. Some changes to the *automount* environment require that systems be rebooted (see Chapter 5, "Maintaining ONC3/NFS" for details on changing the map environment).

The CacheFS File System

CacheFS is optimally used on an NFS client that has sufficient local disk space to reduce network data access time. Once the data has been cached, file read and read-only directory operations are as fast as those on a local disk (EFS file systems). Write performance, however, is closer to an NFS write operation.

The original file system (which is typically NFS) is called the *back file system* and files in it are *back files*. The *cached file system* resides on the local disk and files in it are *cached files*. The *cache directory* is a directory on the local disk where the data for the cached file system is stored. The file system in which the cache directory resides is called the *front file system* and its files are *front files*.

Planning and setting up a CacheFS configuration is similar to that of an NFS client-server configuration.

Command and File Options for CacheFS

CacheFS-specific options have been added to the conventional *mount* command and */etc/fstab* file and are described in this section. For the complete description of these commands and files, refer to “The /etc/fstab Mount Process” on page 15. The *cfsadmin*(1M) and *fsck_cache*(1M) commands are new with CacheFS.

mount and *umount* Commands

When mounting and unmounting a CacheFS file system, the following option is used for CacheFS. For descriptions of the other options, see “mount and umount Command Options” on page 15.

`-t` *type* (type) Set the type of directories to be mounted or unmounted. *type* is `cachefs` for all CacheFS mounting.

fsck_cachefs Command

The CacheFS version of *fsck*(1M) checks the integrity of a cache directory. By default, it corrects any problems it may find. It is automatically invoked when a CacheFS file system is mounted. The syntax for *fsck_cachefs* is:

fsck_cachefs [**-m** | **-o noclean**] *cache_directory*

The two command line options are:

-m Check, but do not repair the file system.

-o noclean Force a check on the cache directory, even if there is no reason to suspect an integrity problem.

/etc/fstab File

The */etc/fstab* file has several new options that are used with CacheFS for mounting, unmounting, and consistency checking.

Any mount options not recognized by CacheFS are passed to the back file system mount if one is performed.

Note: Any mount points which share the same cache directory must have the same set of the following options: `write-around`, `non-shared`, `noconst`, and `purge`.

The options that are new for CacheFS are:

`backfstype=`*file_system_type*
Specifies the back file system type (for example, `nfs`). Any file system type may be used except `proc`, `fd`, and `swap`. The `backfstype` argument *must* be specified.

`backpath=`*path* Specifies the path where the back file system is already mounted. If this argument is not specified, CacheFS determines a mount point for the back file system.

`cachedir=`*directory*
Specifies the name of the cache directory. It must be an existing directory, previously created with *cfsadmin*(1M).

`cacheid=`*ID*	Allows you to assign a string to identify each separate cached file system. If you do not specify a `cacheid`, CacheFS generates one. You need the `cacheid` when you delete a cached file system with *cfsadmin -d*. A `cacheid` you choose is easier to remember than one automatically generated. The *cfsadmin* command with the **-l** option includes the `cacheid` in its display.
`write-around` | `non-shared`	Determines the write modes for CacheFS. In the default `write-around` mode, as writes are made to the back file system, the affected file is purged from the cache. The non-shared mode can be used when only one source is writing to the cached file system. In this mode, all writes are made to both the front and back file systems, and file remains in the cache.
`noconst`	Disables consistency checking between the front and back file systems. Use noconst when the back file system and cache file system are read-only. Otherwise, always allow consistency checking. The default is to enable consistency checking. If none of the files in the back file system will be modified, you can use the `noconst` option to mount when mounting the cached file system. Changes to the back file system may not be reflected in the cached file system.
`local-access`	Improves performance by having CacheFS check the mode bits. By default, the back file system interprets the mode bits used for access checking to ensure data integrity.
`purge`	Remove any cached information for the specified file system.
`suid` | `nosuid`	Allow set-uid (default) or do not allow set-uid.

Consistency Checking *mount* Options for *fstab*

To ensure that the cached directories and files are kept up to date, CacheFS periodically checks consistency of files stored in the cache. To check consistency, CacheFS compares the current modification time to the previous

modification time; if the modification times are different, all data and attributes for the directory or file are purged from the cache and new data and attributes are retrieved from the back file system.

When an operation on a directory or file is requested, CacheFS checks to see if it is time to verify consistency. If so, CacheFS obtains the modification time from the back file system and performs the comparison. If the write mode is `write-around`, CacheFS checks on every operation.

Table 2-1 provides more information on *mount* consistency checking parameters.

Table 2-1 Consistency Checking Arguments for the *-o mount* Option

Parameter	Description
`acdirmin`=*n*	Specifies that cached attributes are held for at least *n* seconds after a directory update. After *n* seconds, if the directory modification time on the back file system has changed, all information about the directory is purged and new data is retrieved from the back file system. The default for *n* is 30 seconds.
`acdirmax`=*n*	Specifies that cached attributes are held for no more than *n* seconds after a directory update. After *n* seconds, the directory is purged from the cache and new data is retrieved from the back file system. The default for *n* is 30 seconds.
`acregmin`=*n*	Specifies that cached attributes are held for at least *n* seconds after file modification. After *n* seconds, if the file modification time on the back file system has changed, all information about the file is purged and new data is retrieved from the back file system. The default for *n* is 30 seconds.
`acregmax`=*n*	Specifies that cached attributes are held for no more than *n* seconds after a file modification. After *n* seconds, all file information is purged from the cache. The default for *n* is 30 seconds.
`actimeo`=*n*	Sets `acregmin`, `acregmax`, `acdirmin`, and `acdirmax` to *n*.

cfsadmin Command

The *cfsadmin*(1M) command is used to administer the cached file system on the local system. It is used to

- create a cached file system
- list the contents and statistics about the cache
- delete the cached file system
- modify the resource parameters when the file system is unmounted

The *cfsadmin* command works on a *cache directory*, which is the directory where the cache is actually stored. A pathname in the front file system identifies the cache directory.

The syntax for the *cfsadmin* command is:

```
cfsadmin -c [ -o cachefs_parameters ] cache_directory
cfsadmin -d [ cache_ID | all ] cache_directory
cfsadmin -l cache_directory
cfsadmin -u [ -o cachefs_parameters ] cache_directory
```

The options and their parameters are:

-c Create a cache under the directory specified by *cache_directory*. This directory must not exist prior to cache creation.

-d Delete the file system and remove the resources of the *cache_ID* that you specify or all file systems in the cache if you specify **all**.

Note: You must run *fsck_cachefs*(1M) after deleting a file system to correct the resource counts for the cache.

-l List the file systems that are stored in the specified cache directory. A listing provides the *cache_ID*, and statistics about resource utilization and cache resource parameters.

-u Update the resource parameters of the specified cache directory. The parameter values (specified with the *-o* option) can only be increased; to decrease the values, you must remove the cache, then re-create it. All file systems in the cache must be unmounted when you use this option.

Changes take effect the next time you mount the file system in the cache directory.

Using the *-u* option with the *-o* option resets all parameters to their default values.

cache_ID — Specifies an identifying name for the file system that is cached. If you do not specify an ID, CacheFS assigns a unique identifier.

-o *options* — Specifies the CacheFS resource parameters. Multiple resource parameters must be separated by commas. The following section describes the cache resource parameters.

Cache Resource Parameters

The default values for the cache parameters are for a cache that uses the entire front file system for caching. To limit the cache to only a portion of the front file system, you should change the parameter values.

Table 2-2 shows the parameters for space and file allocation.

Table 2-2 CacheFS Parameters

Parameters for Space Allocation	Parameters for File Allocation
maxblocks	**maxfiles**
minblocks	**minfiles**
threshblocks	**threshfiles**

Table 2-3 shows the default values for the cache parameters. The default values for parameters devote the full resources of the front file system to caching.

Table 2-3 Default Values of Cache Parameters

Cache Parameters	Default Value
maxblocks	90%
minblocks	0%

Table 2-3 (continued) Default Values of Cache Parameters

Cache Parameters	Default Value
threshblocks	85%
maxfiles	90%
minfiles	0%
threshfiles	85%

The **maxblocks** parameter sets the maximum number of blocks, expressed as a percentage, that CacheFS is allowed to claim within the front file system. The **maxfiles** parameter sets the maximum percentage of available inodes (number of files) CacheFS can claim.

Note: The **maxblocks** and **maxfiles** parameters do not guarantee the resources will be available for CacheFS—they set maximums. If you allow the front file system to be used for purposes other than CacheFS, there may be fewer blocks or files available to CacheFS than you intend.

The **minblocks** parameter does not guarantee availability of a minimum level of resources. The **minblocks** and **threshblocks** parameters work together. CacheFS can claim more than the percentage of blocks specified by **minblocks** only if the percentage of available blocks in the front file system is greater than **threshblocks**. The **minfiles** and **threshfiles** parameters work together in the same fashion.

The **threshfiles** and **threshblocks** values apply to the entire front file system, not file systems you have cached under the front file system. The **threshblocks** and **threshfiles** values are ignored until the **minblocks** and **minfiles** values have been reached.

Note: Using the whole front file system solely for caching eliminates the need to change the **maxblocks**, **maxfiles**, **minblocks**, **minfiles**, **threshblocks**, or *threshfiles* parameter.

When the minimum, maximum, and threshold values are identical, CacheFS allows the cache to grow to the maximum size specified—if you have not reduced available resources by using part of the front file system for other storage purposes.

CacheFS Tunable Parameters

The CacheFS tunable parameters are used to fine tune the performance of CacheFS file opens and reads. The CacheFS tunable parameters are contained in the file */var/sysgen/mtune/cachefs*. They can be modified with the *systune*(1M) command.

There are three tunable parameters for CacheFS. Their descriptions are listed in Table 2-4.

Table 2-4 CacheFS Tunable Parameters

Parameter	Description
cachefs_max_lru	Controls the maximum number of files held open for all mounted CacheFS file systems in anticipation of future use. Holding files open reduces the overhead of opening and closing and is most noticeable for intensive open/close operations. Performance improves as the value is increased, but the system becomes vulnerable to system crashes and the time for unmounting a CacheFS file system increases.
cachefs_readahead	Controls the number of readaheads performed on any given read from a file.
cachefs_max_threads	Controls the maximum number of asynchronous I/O daemons allowed to run for each CacheFS file system.

The parameter's maximum, minimum, and default values are listed in Table 2-5.

Table 2-5 CacheFS Tunable Parameter Values

Parameter	Default Value	Minimum Value	Maximum Value
cachefs_max_lru	1000	0	10000
cachefs_readahead	1	0	10
cachefs_max_threads	5	1	10

Chapter 3

Using Automount Map Options

Automount maps offer a number of options that increase mounting efficiency and make map building easier. This chapter explains each option and provides examples of how to include them in maps. Except as noted, the options described in this chapter can be used in either direct or indirect maps.

This chapter contains these sections:

Including Group Mounts in Maps

Group mounts are a means of organizing entries in a direct map so that a single mount provides several directories that users are likely to need simultaneously. Group mounts work only with direct maps. The map entry for a group mount specifies the parent directory to be mounted. Subentries specify the individual child directories the mount makes available and any mount options that apply to them. The directories in a group mount need not be on the same server.

A sample group mount entry is:

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

This example shows that, when */usr/local* is mounted, users have access to three directories: */export/local/iris_bin*, a read-only directory on server *ivy; /usr/local/share*, a read-write directory on server *willow*; and */home/jones/src*, a read-only directory on server *oak*. The backslash (\) at the end of a line indicates that a continuation line follows. Continuation lines are indented with blank spaces or tabs.

Without the group mount feature, the single entry shown in the previous example would require three separate mounts and three individual map entries, as shown in this example:

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

Group mounts and separate entries differ in that group mounts guarantee that all directories in the group are mounted whenever any one of them is referenced. This is not the case for separate entries. For example, notice the error message that occurs in this sequence when the user specifies a relative pathname to change directories:

```
% cd /usr/local/bin
% cd ../src
UX:csh:ERROR: ../src - No such file or directory
```

The error occurs because the directory */usr/local/src* is not mounted with */usr/local/bin*. A separate *cd*(1) command is required to mount */usr/local/src*.

Using Hierarchical Formats in Group Mounts

When the root of a file hierarchy must be mounted before any other mounts can occur, it must be specified in the map. A *hierarchical* mount is a special case of group mounts in which directories in the group must be mounted in a particular order. For hierarchical mounts, the automounter must have a separate mount point for each mount within the hierarchy.

The sample group mount entry shown in the previous section illustrates nonhierarchical mounts under */usr/local* when */usr/local* is already mounted, or when it is a subdirectory of another mounted system. The concept of *root* here is very important. The symbolic link returned by the automounter to the kernel request is a path to the mount root, the root of the hierarchy mounted under */tmp_mnt*.

An example of a hierarchical mount is:

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

The mount points used here for the hierarchy are */*, */bin*, */share*, and */src*. These mount point paths are relative to the mount root, not to the system's file system root. The first entry in this example has / as its mount point. It is mounted *at* the mount root. The first mount of a hierarchy is not required to be at the mount root. The automounter creates directories to build a path to the first mount point if the mount point is not at the mount root.

A true hierarchical mount can be a disadvantage if the server of the root hierarchy becomes unavailable. When this happens, any attempt to unmount the lower branches fail, since unmounting must proceed through the mount root, and the mount root cannot be unmounted while its server is unavailable.

Specifying Alternative Servers

In an automount map, you can specify alternative servers to be used in the event the specified server is unavailable when mounting is attempted. This example illustrates an indirect map in which alternative servers are used:

```
man     –ro,intr     oak:/usr/man \
                          rose:/usr/man \
                          willow:/usr/man
frame   –ro,intr     redwood:/usr/frame2.0 \
                          balsa:/export/frame
```

The mount point *man* lists three server locations, and *frame* lists two. Mounting can be done from any listed server, as long as it is available.

Alternative locations are recommended for mounting read-only hierarchies. However, they are not advised for read-write files, since alternating versions of writable files causes problems with version control.

In the example above, multiple mount locations are expressed as a list of mount locations in the map entry. They can also be expressed as a comma-separated list of servers, followed by a colon and the pathname, if the pathname is the same for all alternate servers:

```
man      –ro,intr          oak,rose,willow:/usr/man
```

In this example, manual pages are mounted from either *oak*, *rose*, or *willow*, but this list of servers does not imply order. However, the automounter does try to connect to servers on the local network first before soliciting servers on a remote network. The first server to respond to the automounter's RPC requests is selected, and *automount*(1M) attempts to mount the server.

Although this redundancy is very useful in an environment where individual servers may or may not be exporting their file systems, it is beneficial at mount time only. If a server goes down while a mount is in effect, the directory becomes unavailable. An option here is to wait 5 minutes until the auto-unmount takes place and try again. At the next attempt, the automounter chooses one of the available servers. It is also possible to use the *umount*(1M) command to unmount the directory, and then inform the automounter of the change in the mount table with the command **`/etc/killall -HUP automount`**, and retry the mount. See the *automount*, *killall*(1M), and *umount*(1M) manual pages for more details.

Using Metacharacters

The automounter recognizes some characters, *metacharacters*, as having a special meaning. Metacharacters are used to do substitutions and to disable the effects of special characters. Metacharacters recognized by the automounter are described below.

The Ampersand (&)

The automounter recognizes an ampersand as a string substitution character. It replaces ampersands in the location field with the directory field character string specification wherever the ampersand occurs in the location specification. For example, assume you have a map containing many subdirectory specifications, like this:

```
#Directory    Mount Options    Location
john          -nodev           willow:/home/willow:john
mary          -nosuid,intr     willow:/home/willow:mary
joe           -ro,intr         willow:/home/willow:joe
able                           pine:/export/home:able
baker                          peach:/export/home:baker
```

Using the ampersand, the map above looks like this:

```
#Directory    Mount Options    Location
john          -nodev           willow:/home/willow:&
mary          -nosuid,intr     willow:/home/willow:&
joe           -ro,intr         willow:/home/willow:&
able                           pine:/export/home:&
baker                          peach:/export/home:&
```

Let's say the server name and directory name are the same, as in this example:

```
#Directory    Mount Options    Location
willow        –intr            willow:/home/willow
peach         –ro              peach:/home/peach
pine                           pine:/home/pine
oak           –nosuid,intr     oak:/home/oak
poplar        –nosuid,intr     poplar:/home/poplar
```

Using the ampersand results in entries that look like this:

```
#Directory    Mount Options    Location
willow        –intr            &:/home/&
peach         –ro              &:/home/&
pine                           &:/home/&
oak           –nosuid,intr     &:/home/&
poplar        –nosuid,intr     &:/home/&
```

You can also use directory substitutions in a direct map. For example, assume a direct map contains this entry:

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

Using an ampersand, this entry can be shortened to this:

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

Notice that the ampersand substitution uses the whole directory string. Since directory specifications in a direct map begin with a slash (/), it is important to remember that the slash is carried over when you use the ampersand. For example, if a direct map contains this entry,

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

the automounter interprets the map entry in this way:

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

The Asterisk (*)

The automounter recognizes an asterisk as a wild card substitution for a directory specification given in a command line. Asterisks must always be the last entry in a map, since the automounter does not read beyond an asterisk entry.

Consider the map in this example:

```
#Directory    Mount Options    Location
oak           –nosuid,intr     &:/export/&
poplar        –nosuid,intr     &:/export/&
*                              &:/home/&
```

In this example, a command line entry with the directory argument `redwood` is equivalent to this map entry:

```
redwood                         redwood:/home/redwood
```

In the next map, the last two entries are always ignored:

```
#Directory     Mount Options     Location
*                                &:/home/&
oak            –nosuid,intr      &:/export/&
poplar         –nosuid,intr      &:/export/&
```

The Backslash (\)

The automounter recognizes the backslash as a signal to disable the effects of the special character that follows it. It interprets the special character literally. For example, under certain circumstances, you might need to mount directories whose names could confuse the automounter's map parser. An example might be a directory called *rc0:dk1*. This name could result in an entry like:

```
/junk          –ro               vmsserver:rc0:dk1
```

The presence of the two colons in the location field confuses the automounter's parser. To avoid this confusion, use a backslash to escape the second colon and remove its special meaning of separator:

```
/junk          –ro               vmsserver:rc0\:dk1
```

Double quotes (")

Automount recognizes double quotes (") as string delimiters. Blank spaces within double quotes are not interpreted as the start of a new field. This example illustrates double quotes used to hide the blank space in a two-word name:

```
/smile                          dentist:/"front teeth"/smile
```

Using Environment Variables

You can use the value of an environment variable by prefixing a dollar sign to its name. You can also use braces to delimit the name of the variable from appended letters or digits. Environment variables can appear anywhere in an entry line, except as a *directory*.

The environment variables can be inherited from the environment or can be defined explicitly with the **–D** command line option. For instance, if you want each client to mount client–specific files in the network in a replicated format, you could create a specific map for each client according to its name, so that the relevant line for the system *oak* looks like this:

```
/mystuff                acorn,ivy,balsa:/export/hostfiles/oak
```

For *willow*, the entry looks like this:

```
/mystuff              acorn,ivy,balsa:/export/hostfiles/willow
```

This scheme is viable within small networks, but maintaining system-specific maps across a large network is burdensome. An alternative for large networks is to start the automounter with a command like this:

```
# /usr/etc/automount –D HOST=‘hostname‘ ...
```

The entry in the direct map looks like this:

```
/mystuff               acorn,ivy,balsa:/export/hostfiles/$HOST
```

Now each system finds its own files in the *mystuff* directory, and centralized administration and distribution of maps is easier.

Including Supplementary Maps

A line of the form +*mapname* causes the automounter to consult the mentioned map as if it were included in the current map. If *mapname* is a relative pathname (no slashes), the automounter assumes it is an NIS map. If the pathname is an absolute pathname the automounter looks for a local map of that name. If the map name starts with a dash (–), the automounter consults the appropriate built-in map.

For instance, you can have a few entries in your local *auto.home* map for the most commonly accessed home directories and follow them with the included NIS map, as shown in this example:

```
ivy             -rw,intr          &:/home/&
oak             -rw,intr          &:/export/home
+auto.home
```

If the automounter finds no match in the included map, it continues scanning the current map. This allows you to use additional entries after the included map, as shown in this example:

```
ivy             -rw,intr          &:/home/&
oak             -rw,intr          &:/export/home
+auto.home
*               -rw               &:/home/&
```

Finally, the included map can be a local file, or even 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
*             &:/export/&       # wild card
```

Notice that in all cases the wild card should be the last entry, since the automounter does not continue consulting the map after it reads the asterisk. It assumes the wild card has found a match.

Chapter 4

Setting Up and Testing ONC3/NFS

This chapter explains how to set up ONC3/NFS services and verify that they work. It provides procedures for enabling exporting on NFS servers, for setting up mounting and automounting on NFS clients, and for setting up the network lock manager. It also explains how to create a CacheFS file system. Before you begin these procedures, you should be thoroughly familiar with the information provided in Chapter 2, "Planning ONC3/NFS Service."

This chapter contains these sections:

Note: To do the procedures in this chapter, you should have already installed ONC3/NFS software on the server and client systems that will participate in the ONC3/NFS services. The *ONC3/NFS Release Notes* explain where to find instructions for installing ONC3/NFS software.

Setting Up the NFS Server

Setting up an NFS server requires verifying that the required software is running on the server, editing the server's */etc/exports* file, adding the file systems to be exported, exporting the file systems, and verifying that they have been exported. The instructions below explain the set-up procedure. They assume that NFS software is already installed on the server. Do this procedure as the superuser on the server.

1. Check the NFS configuration flag on the server.

   When the */etc/init.d/network* script executes at system startup, it starts NFS running if the *chkconfig*(1M) flag *nfs* is on. To verify that *nfs* is on, type the *chkconfig*(1M) command and check its output, for example:

   ```
   # /etc/chkconfig
           Flag                State
           ====                =====
           ...
           nfs                 on
           ...
   ```

   This example shows that the *nfs* flag is set to on.

2. If your output shows that *nfs* is off, type this command and reboot your system:

   ```
   # /etc/chkconfig nfs on
   ```

3. Verify that NFS daemons are running.

   Four *nfsd* and four *biod* daemons should be running (the default number specified in */etc/config/nfsd.options* and */etc/config/biod.options*). Verify that the appropriate NFS daemons are running using the *ps*(1) command, shown below. The output of your entries looks similar to the output in these examples:

   ```
   # ps -ef | grep nfsd
   root   102      1  0  Jan 30 ?      0:00 /usr/etc/nfsd 4
   root   104    102  0  Jan 30 ?      0:00 /usr/etc/nfsd 4
   root   105    102  0  Jan 30 ?      0:00 /usr/etc/nfsd 4
   root   106    102  0  Jan 30 ?      0:00 /usr/etc/nfsd 4
   root  2289   2287  0 14:04:50 ttyq4 0:00 grep nfsd
   ```

```
# ps -ef | grep biod
root   107      1  0  Jan 30 ?      0:00 /usr/etc/biod 4
root   108      1  0  Jan 30 ?      0:00 /usr/etc/biod 4
root   109      1  0  Jan 30 ?      0:00 /usr/etc/biod 4
root   110      1  0  Jan 30 ?      0:00 /usr/etc/biod 4
root  2291   2287  4 14:04:58 ttyq4 0:00 grep biod
```

If no NFS daemons appear in your output, they were not included in the IRIX kernel during NFS installation. To check the kernel, type this command:

```
# strings /unix | grep nfs
```

If there is no output, rebuild the kernel with this command, then reboot the system:

```
# /etc/autoconfig -f
```

4. Verify that mount daemons are registered with the portmapper.

   Mount daemons must be registered with the server's portmapper so the portmapper can provide port numbers to incoming NFS requests. Verify that the mount daemons are registered with the portmapper by typing this command:

   ```
   # /usr/etc/rpcinfo –p | grep mountd
   ```

   After your entry, you should see output similar to this:

   ```
   100005  1   tcp  1230  mountd
   100005  1   udp  1097  mountd
   391004  1   tcp  1231  sgi_mountd
   391004  1   udp  1098  sgi_mountd
   ```

   The *sgi_mountd* in this example is an enhanced mount daemon that reports on SGI-specific export options.

5. Edit the */etc/exports* file.

   Edit the */etc/exports* file to include the file systems you want to export and their export options (*/etc/exports* and export options are explained in "/etc/exports and Other Export Files" in Chapter 2). This example shows one possible entry for the */etc/exports* file:

   ```
   /usr/demos -ro,access=client1:client2:client3
   ```

   In this example, the file system */usr/demos* are exported with read-only access to three clients: *client1*, *client2*, and *client3*. Domain information can be included in the client names, for example *client1.eng.sgi.com*.

6. Run the *exportfs*(1M) command.

   Once the */etc/exports* file is complete, you must run the *exportfs* command to make the file systems accessible to clients. You should run *exportfs* anytime you change the */etc/exports* file. Type this command:

   ```
   # /usr/etc/exportfs -av
   ```

   In this example, the **–a** exports all file systems listed in the */etc/exports* file, and the **–v** causes *exportfs* to report its progress. Error messages reported by *exportfs* usually indicate a problem with the */etc/exports* file.

7. Use *exportfs* to verify your exports.

   Type the *exportfs* command with no parameters to display a list of the exported file system(s) and their export options, as shown in this example:

   ```
   # /usr/etc/exportfs
   /usr/demos -ro,access=client1:client2:client3
   ```

   In this example, */usr/demos* is accessible as a read-only file system to systems *client1*, *client2*, and *client3*. This matches what is listed in the */etc/exports* file for this server (see instruction 5 of this procedure). If you see a mismatch between the */etc/exports* file and the output of the *exportfs* command, check the */etc/exports* file for syntax errors.

The NFS software for this server is now running and its resources are available for mounting by clients. Repeat these instructions to set up additional NFS servers.

Setting Up an NFS Client

Setting up an NFS client for conventional mounting requires verifying that NFS software is running on the client, editing the */etc/fstab* file, adding the names of directories to be mounted, and mounting the directories in */etc/fstab* by giving the *mount*(1M) command or by rebooting your system. These directories remain mounted until you explicitly unmount them.

Note: For instructions on mounting directories not listed in */etc/fstab*, see "Temporary NFS Mounting" in Chapter 5.

The procedure below explains how to set up NFS software on a client and mount its NFS resources using the *mount* command. You must do this procedure as the superuser.

1. Use *chkconfig* to check the client's NFS configuration flag.

   To verify that *nfs* is on, give the *chkconfig* command and check its output (see "Setting Up the NFS Server" in this chapter for details on *chkconfig*).

2. If your output shows that *nfs* is `off`, type this command and reboot your system:

   # **`/etc/chkconfig nfs on`**

3. Verify that NFS daemons are running.

   Four *nfsd* and four *biod* daemons should be running (the default number specified in */etc/config/nfsd.options* and */etc/config/biod.options*). Verify that the appropriate NFS daemons are running using the *ps*(1) command, shown below. The output of your entries looks similar to the output in these examples:

```
# ps -ef | grep nfsd
root   102      1  0  Jan 30 ?      0:00 /usr/etc/nfsd 4
root   104    102  0  Jan 30 ?      0:00 /usr/etc/nfsd 4
root   105    102  0  Jan 30 ?      0:00 /usr/etc/nfsd 4
root   106    102  0  Jan 30 ?      0:00 /usr/etc/nfsd 4
root  2289   2287  0 14:04:50 ttyq4 0:00 grep nfsd
```

```
# ps -ef | grep biod
root   107      1  0  Jan 30 ?      0:00 /usr/etc/biod 4
root   108      1  0  Jan 30 ?      0:00 /usr/etc/biod 4
root   109      1  0  Jan 30 ?      0:00 /usr/etc/biod 4
root   110      1  0  Jan 30 ?      0:00 /usr/etc/biod 4
root  2291   2287  4 14:04:58 ttyq4 0:00 grep biod
```

If no NFS daemons appear in your output, they were not included in the IRIX kernel during NFS installation. To check the kernel, type this command:

```
# strings /unix | grep nfs
```

If there is no output, rebuild the kernel with this command, then reboot the system:

```
# /etc/autoconfig -f
```

4. Edit the */etc/fstab* file.

   Add an entry to the */etc/fstab* file for each NFS directory you want mounted when the client is booted. The example below illustrates an */etc/fstab* with an NFS entry to mount */usr/demos* from the server *redwood* at mount point */n/demos*:

   ```
   /dev/root          /           efs rw,raw=/dev/rroot 0 0
   /dev/usr           /usr        efs rw,raw=/dev/rusr 0 0
   redwood:/usr/demos /n/demos    nfs ro,intr,bg 0 0
   ```

   Note: The background (bg) option in this example allows the client to proceed with the boot sequence without waiting for the mount to complete. If the bg option is not used, the client hangs if the server is unavailable.

5. Create the mount points for each NFS directory.

   After you edit the */etc/fstab* file, create a directory to serve as the mount point for each NFS entry in */etc/fstab* file. If you specified an existing directory as a mount point for any of your */etc/fstab* entries, remember that the contents of the directory are inaccessible while the NFS mount is in effect.

   For example, to create the mount point */n/demos* for mounting the directory */usr/demos* from server *redwood*, give this command:

   ```
   # mkdir -p /n/demos
   ```

6. Mount each NFS resource.

   You can use the *mount* command in several ways to mount the entries in this client's */etc/fstab*. See the *mount*(1M) manual page for a description of the options. The examples below show two methods: mounting each entry individually and mounting all *fstab* entries that specify a particular server. The first example is:

   ```
   # mount /n/demos
   ```

   In this example, only the mount point is specified. All other information needed to perform the mount, the server name *redwood* and its resource */usr/demos*, is provided by the */etc/fstab* file.

   The second example is:

   ```
   # mount -h redwood
   ```

   In this example, all NFS entries in */etc/fstab* that specify server *redwood* are mounted.

   Note: If you reboot the client instead of using the *mount* command, all NFS entries in */etc/fstab* will be mounted.

The NFS software for this client is now ready to support user requests for NFS directories. Repeat these instructions to set up additional NFS clients.

Setting Up the Automounter

Since the automounter runs only on NFS clients, all setup for the automounter is done on the client system. This section provides two procedures for setting up the automounter: one for setting up a default automount environment and one for setting up a more complex environment.

Setting Up a Default Automount Environment

If you set up the default automount environment on a client, at system startup *automount*(1M) reads the */etc/config/automount.options* file for mount information. By default, */etc/config/automount.options* contains an entry for a special map called *–hosts*. The *–hosts* map tells the automounter to read the hosts database (*/etc/hosts*, NIS and/or DNS (BIND), see the *resolver*(4) manual page) and use the server specified on the command line if the hosts database has a valid entry for that server. When using the *–hosts* map, when a client accesses a server, *automount* gets the exports list from the server and mounts all directories exported by that server. *automount* uses */tmp_mnt/hosts* as the mount point.

A sample *–hosts* entry in */etc/config/automount.options* is:

```
-v    /hosts    -hosts    -intr,nosuid,nodev
```

Use this procedure to set up the default automount environment on an NFS client. You must do this procedure as the superuser.

1. Verify that NFS flags are on.

   By default, the *nfs* and *automount* flags are set to on. To verify that they are on, give the *chkconfig* command and check its output (see instruction 1 of "Setting Up an NFS Client" in this chapter for sample *chkconfig* output). If either flag is set to `off`, use one of these commands to reset it, then reboot:

   # **`/etc/chkconfig nfs on`**
   # **`/etc/chkconfig automount on`**

2. Verify that the default configuration is working:

   # **`cd /hosts/`***servername*

In place of *servername*, substitute the host name of any system whose name can be resolved by the host name resolution method you are using (see the *resolver*(4) manual page). If the system specified is running NFS and has file systems that can be accessed by this client, *automount* mounts all available file systems to */tmp_mnt/hosts/servername*. If the system is not running NFS or has nothing exported that you have access to, you get an error message when you try to access its file systems.

3. Verify that directories have been mounted, for example (the \ character shows that the output is wrapped):

   ```
   # mount
   servername:/ on /tmp_mnt/hosts/servername type nfs
   (rw,dev=c0005)
   ```

   The automounter has serviced this request. It dynamically mounted */hosts/servername* using the default *automount* environment.

Setting Up a Custom Automount Environment

A customized automount environment allows you to select the NFS directories that are dynamically mounted on a particular client, and allows you to customize the options in effect for particular mounts. You must complete four general steps to set up a customized automount environment:

1. Creating the maps
2. Starting the *automount* program
3. Verifying the *automount* process
4. Testing *automount*

Step 1: Creating the Maps

A customized automount environment contains a master map and any combination of direct and indirect maps. Although a master map is required, *automount* does not require both direct and indirect maps. You can use either direct or indirect maps exclusively.

Instructions for creating each type of map are given below. Notice from these instructions that a crosshatch (#) at the beginning of a line indicates a

comment line in all types of maps. Include comment lines in your maps to illustrate map formats until you become familiar with each map type.

1. Create the master map on the client.

   The master map points *automount* to other files that have more detailed information needed to complete NFS mounts. To create the master map, become superuser and create a file called */etc/auto.master* with any text editor. Specify the mount point, map name, and any options that apply to the direct and indirect maps in your entries, for example:

   ```
   #Mount Point   Map Name            Map Options
   /food/dinner   /etc/auto.food      -ro
   /-             /etc/auto.exercise  -ro,soft,intr
   /hosts         -hosts              -intr,nosuid,nodev
   ```

2. Create the indirect map.

   Create your indirect map and insert the entries it needs. This example is the indirect map */etc/auto.food*, listed in */etc/auto.master* in instruction 1:

   ```
   #Directory    Options      Location
   ravioli                   venice:/food/pasta
   crepe         -rw         paris:/food/desserts
   chowmein                  hongkong:/food/noodles
   ```

3. Create the direct map.

   Create your direct map and insert the entries it needs. This example is the direct map */etc/auto.exercise*, listed in */etc/auto.master* in instruction 1:

   ```
   #Directory        Options   Location
   /leisure/swim              spitz:/sports/water/swim
   /leisure/tennis            becker:/sports/racquet/tennis
   /leisure/golf     -hard    palmer:/sports/golf
   ```

Step 2: Starting the *automount* Program

You can set up the software on a client so that *automount* starts when the client is booted, and you can also start *automount* from the command line. The procedures in this section explain how to set up the automounter to start during the boot sequence.

If *automount* is configured on at system startup, the */etc/init.d/network* script reads the contents of the */etc/config/automount.options* file to determine how to start the *automount* program and what to mount and how to mount it.

Depending on the site configuration specified in the */etc/automount.options* file, *automount* either finds all necessary information in the */etc/automount.options* file, or it is directed to local or NIS maps (or both) for additional mounting information.

Note: If you plan to use NIS database maps other than the *–hosts* built-in map, you need to create the NIS maps. See the *NIS Administration Guide* for information on building custom NIS maps.

Follow this procedure to set *automount* to start automatically at system startup:

1. Configure *automount* on with the *chkconfig* command (if needed):

   # **/etc/chkconfig automount on**

2. Modify the */etc/config/automount.options* file.

   Using any standard editor, modify */etc/config/automount.options* to reflect the *automount* site environment. See *automount*(1M) for details on the */etc/config/automount.options* file. Based on the previous examples, the */etc/config/automount.options* file contains this entry:

   ```
   -v -m -f /etc/auto.master
   ```

   The **–v** option directs error messages to the screen during start up and into the */var/adm/SYSLOG* file once *automount* is up and running. The **–m** option tells *automount* not to check the NIS database for a master map. Use this option to isolate map problems to the local system by inhibiting *automount* from reading the NIS database maps, if any exist. The **–f** option tells *automount* that the argument that follows it is the full path name of the master file.

   Note: In general, it is recommended that you start the automounter with the verbose option (**–v**), since this option provides messages that can help with problem solving.

3. Reboot the system.

Step 3: Verifying the *automount* Process

Verify that the *automount* process is functioning by performing the following two steps.

1. Validate that the *automount* daemon is running with the *ps* command:

   # **ps -ef | grep automount**

   You should see output similar to this:

```
root    455     1  0   Jan 30 ?        0:02 automount -v -m -f /etc/auto.master
root   4675  4673  0 12:45:05 ttyq5    0:00 grep automount
```

2. Check the */etc/mtab* entries.

   When the *automount* program starts, it creates entries in the client's */etc/mtab* for each *automount* mount point. Entries in */etc/mtab* include the process number and port number assigned to *automount*, the mount point for each direct map entry, and each indirect map. The */etc/mtab* entries also include the map name, map type (direct or indirect), and any mount options.

   Look at the */etc/mtab* file. A typical */etc/mtab* table with *automount* running looks similar to this example (wrapped lines end with the \ character):

```
/dev/root / efs rw,raw=/dev/rroot 0 0
/dev/usr /usr efs rw,raw=/dev/rusr 0 0
/debug /debug dbg rw 0 0
/dev/diskless /diskless efs rw,raw=/dev/rdiskless 0 0
/dev/d /d efs rw,raw=/dev/rd 0 0
flight:(pid12155) /src/sgi ignore \
    ro,intr,port=885,map=/etc/auto.source,direct 0 0
flight:(pid12155) /pam/framedocs/nfs ignore \
    ro,intr,port=885,map=/etc/auto.source,direct 0 0
flight:(pid12155) /hosts ignore ro,intr,port=885,\
    map=-hosts,indirect,dev=1203 0 0
```

   The entries corresponding to *automount* mount points have the file system type `ignore` to direct programs to ignore this */etc/mtab* entry. For instance, *df*(1) and *mount* do not report on file systems with the type `ignore`. When a directory is NFS mounted by the *automount* program, the */etc/mtab* entry for the directory has *nfs* as the file system type. *df* and *mount* report on file systems with the type `nfs`.

Step 4: Testing *automount*

When the *automount* program is set up and running on a client, any regular account can use it to mount remote directories transparently. You can test your automount set-up by changing to a directory specified in your map configuration.

The instructions below explain how to verify that *automount* is working.

1. As a regular user, *cd*(1) to an automounted directory.

   For example, test whether *automount* mounts */food/pasta*:

   ```
   % cd /food/dinner/ravioli
   ```

   This command causes *automount* to look in the indirect map */etc/auto.food* to execute a mount request to server *venice* and apply any specified options to the mount. *automount* then mounts the directory */food/pasta* to the default mount point */tmp_mnt/food/dinner/ravioli*. The directory */food/dinner/ravioli* is a symbolic link to */tmp_mnt/food/dinner/ravioli*.

   Note: The */food/dinner* directory appears empty unless one of its subdirectories has been accessed (and therefore mounted).

2. Double-check your setup using a different directory.

   To have *automount* NFS mount */sports/water/swim* automatically, give this command:

   ```
   % cd /leisure/swim
   ```

   This command causes *automount* to look in the direct map */etc/auto.exercise* to execute a mount request to server *spitz* and apply specified options to the mount. It then mounts the directory */sports/water/swim* to the default mount point */tmp_mnt/leisure/swim*. The directory */leisure/swim* is a symbolic link to */tmp_mnt/leisure/swim*.

3. Verify that the individual mount has taken place.

   Use the *pwd*(1) command to verify that the mount has taken place, as shown in this example:

   ```
   % pwd
   /leisure/swim
   ```

4. Verify that both directories have been mounted with the automounter.

   You can also verify automounted directories by checking the output of a *mount* command:

   ```
   % mount
   ```

   mount reads the current contents of the */etc/mtab* file and includes conventional and automount mounted directories in its output.

The custom configuration of automount is set up and ready to work for users on this client.

Setting Up the Lock Manager

The NFS lock manager provides file and record locking between a client and server for NFS-mounted directories. As an NFS utility, the lock manager is in effect when NFS software in installed and operating properly on both the server and client systems. It is implemented by two daemons, *lockd*(1M) and *statd*(1M). These daemons must be running on an NFS server and its clients for the lock manager to function.

The NFS lock manager program must be running on both the NFS client and the NFS server to function properly. Use this procedure to check the lock manager setup:

1. Use *chkconfig* on the client to check the lock manager flag.

   To verify that the *lockd* flag is on, give the *chkconfig* command and check its output (see instruction 1 of "Setting Up an NFS Client" in this chapter for sample *chkconfig* output). If your output shows that *lockd* is `off`, give this command and reboot your system:

   ```
   # /etc/chkconfig lockd on
   ```

2. Verify that both lock manager daemons are running.

   Give these *ps* commands and check their output to verify that the lock manager daemons, *rpc.lockd*(1M) and *rpc.statd*(1M), are running:

```
# ps -ef | grep statd
root   131     1  0   Aug  6 ?        0:51 /usr/etc/rpc.statd
root  2044   427  2 16:13:24 ttyq1    0:00 grep statd
# ps -ef | grep lockd
root   129     1  0   Aug  6 ?        0:51 /usr/etc/rpc.lockd
root  2045   427  2 16:13:24 ttyq1    0:00 grep lockd
```

   If either *rpc.lockd* or *rpc.statd* is not running, start them manually by giving these commands in this order:

```
# /usr/etc/rpc.statd
# /usr/etc/rpc.lockd
```

3. Repeat instructions 1 and 2, above, on the NFS server.

Setting Up the CacheFS File System

When you set up a cache, you can use all or part of an existing file system. You can also set up a new slice to be used by CacheFS. In addition, when you create a cache, you can specify the percentage of resources, such as number of files or blocks, that CacheFS can use in the front file system. The configurable cache parameters are discussed in the section “Cache Resource Parameters” on page 32.

Before starting to set up CacheFS, check that it is configured to start on both the server and client.

1. Check the CacheFS configuration flag.

   When the */etc/init.d/network* script executes at system startup, it starts CacheFS running if the *chkconfig*(1M) flag *cachefs* is on. To verify that *cachefs* is on, type the *chkconfig*(1M) command and check its output, for example:

   ```
   # /etc/chkconfig
           Flag                 State
           ====                 =====
           ...
          cachefs              on
           ...
   ```

   This example shows that the *cachefs* flag is set to on.

2. If your output shows that *cachefs* is `off`, type this command and reboot your system:

   ```
   # /etc/chkconfig cachefs on
   ```

Front File System Requirements

CacheFS typically uses a local EFS file system for the front file system.You can use an existing EFS file system for the front file system or you can create a new one. Using an existing file system is the quickest way to set up a cache. Dedicating a file system exclusively to CacheFS gives you the greatest control over the file system space available for caching.

Caution: Do not make the front file system read-only and do not set quotas on it. A read-only front file system prevents caching, and file system quotas interfere with control mechanisms built into CacheFS.

Setting Up a Cached File System

There are two steps to setting up a cached file system:

1. You must create the cache with the *cfsadmin* command. See “Creating a Cache” on page 61.
2. You must mount the file system you want cached using the **-t cachefs** option to the *mount* command. See “Mounting a Cached File System” on page 62.

Creating a Cache

The following example is the command to create a cache:

`cfsadmin -c` *directory_name*

The following example creates a cache and creates the cache directory */local/mycache*. Make sure the cache directory does not already exist.

```
# cfsadmin -c /local/mycache
```

This example uses the default cache parameter values. The CacheFS parameters are described in the section “Cache Resource Parameters” on page 32. See the *cfsadmin*(1M) manual page and “cfsadmin Command” on page 31 for more information on *cfsadmin* options.

Setting Cache Parameters

The following example shows how to set parameters for a cache.

`cfsadmin -c -o` *parameter_list cache_directory*

The *parameter_list* has the following form:

parameter_name1=value,parameter_name2=value,...

The parameter names are listed in Table 2-2 on page 32. You must separate multiple arguments to the *-o* option with commas.

Note: The maximum size of the cache is by default 90% of the front file system resources. Performance deteriorates significantly if an EFS file system exceeds 90% capacity.

The following example creates a cache named */local/cache1* that can use up to 80% of the disk blocks in the front file system and can grow to use 55% of the front file system blocks without restriction unless 60% (or more) of the front file system blocks are already in use.

```
# cfsadmin -c -o maxblocks=80,minblocks=55,threshblocks=60 \
/local/cache1
```

The following example creates a cache named */local/cache2* that can use up to 75% of the files available in the front file system.

```
# cfsadmin -c -o maxfiles=75 /local/cache2
```

The following example creates a cache named */local/cache3* that can use 75% of the blocks in the front file system, that can use 50% of the files in the front file system without restriction unless total file usage already exceeds 60%, and that has 70% of the files in the front file system as an absolute limit.

```
# cfsadmin -c -o \
maxblocks=75,minfiles=50,threshfiles=60,maxfiles=70 \
/local/cache3
```

Mounting a Cached File System

There are two ways to mount a file system in a cache:

- Using the *mount* command
- Creating an entry for the file system in the */etc/fstab* file

Using *mount* to Mount a Cached File System

The following command mounts a file system in a cache.

```
mount -t cachefs -o backfstype=type,cachedir=cache_directory \
back_file system mount_point
```

The arguments used with the *-o* option are described in "/etc/fstab File" on page 28. See the *mount*(1M) manual page for more information about the arguments used when mounting a cached file system.

For example, the following command makes the file system *merlin:/docs* available as a cached file system named */docs*:

```
# mount -t cachefs -o backfstype=nfs,cachedir=/local\
/cache1 merlin:/docs /docs
```

Mounting a Cached File System That Is Already Mounted

Use the **backpath** argument when the file system you want to cache has already been mounted. **backpath** specifies the mount point of the mounted file system. When the **backpath** argument is used, the back file system must be read-only. If you want to write to the back file system, you must unmount it before mounting it as a cached file system.

For example, if the file system *merlin:/doc* is already NFS-mounted on */nfsdocs*, you can cache that file system by giving that path name as the argument to *backpath*, as shown in the following example:

```
# mount -t cachefs -o \
backfstype=nfs,cachedir=/local/cache1,backpath=/nfsdocs \
merlin:/doc /doc
```

Note: There is no performance gain in caching a local EFS disk file system.

Mounting a CD-ROM as a Cached File System

So far, examples have illustrated back file systems that are NFS-mounted, and the device argument to the *mount* command has taken the form *server:file_system*. If the back file system is an ISO9660 file system, the device argument is the CD-ROM device in the */CDROM* directory. The file system type is **iso9660**.

The following example illustrates caching an ISO9660 back file system on the device */CDROM* as */doc* in the cache */local/cache1*:

```
# mount -t cachefs -o
backfstype=iso9660,cachedir=/local/cache1,\
ro,backpath=/CDROM /CDROM /doc
```

Because you cannot write to the CD-ROM, the `ro` argument is specified to make the cached file system read-only. The arguments to the *-o* option are explained in “/etc/fstab and Other Mount Files” on page 16.

You must specify the **backpath** argument because the CD-ROM is automatically mounted when it is inserted. The mount point is in the */CDROM* directory and is determined by the name of the CD-ROM. The special device to mount is the same as the value for the **backpath** argument.

Note: When a CD-ROM is changed, the CacheFS file system must be unmounted and remounted.

Creating an *fstab* Entry for Cached File Systems

As with other file system types, you can put an entry in the */etc/fstab* file for a cached file system to mount the cached file system automatically every time the system boots. The */etc/fstab* file has the following fields:

- device to mount
- mount point
- file system type
- mount options
- dump frequency
- *fsck* pass

Enter the special device name of the back file system as the device to mount. For NFS file systems, the entry takes the form *server:path*. The device to *fsck* is the cache directory path. The mount point is the mount point of the cached file system. The dump frequency and *fsck* pass should always be 0. The following example shows an entry for a cached file system (the lines beginning with hash marks (#) are comments):

```
#device        mount  FS       mount                              dump      fsck
#to mount      point  type     options                            frequency pass

svr1:/usr/abc  /docs  cachefs  rw,backfstype=nfs,cachedir=/cache1  0         0
```

Checking a Cached File System

The *fsck_cachefs*(1M) command checks the integrity of cached file systems. The CacheFS version of *fsck* automatically corrects problems without requiring user interaction.

To check a cached file system, type:

`fsck_cachefs -o noclean` *cache_directory*

The following example forces a check of the cache directory */local/cache1*:

```
# fsck_cachefs -o noclean /local/cache1
```

You should not need to run *fsck* manually for cached file systems; *fsck* is run automatically when the file system is mounted.

Two options are available for the CacheFS version of *fsck*: *-m* and *-o* **noclean**. The *-m* option causes *fsck* to check the specified file system without making any repairs. The *-o* **noclean** option forces a check of the file system. See the *fsck_cachefs*(1M) man page for more information.

Chapter 5

Maintaining ONC3/NFS

This chapter provides information about maintaining ONC3/NFS. It explains how to change the default number of NFS daemons and modify automount maps. It also gives suggestions for using alternative mounting techniques and avoiding mount point conflicts. It also describes how to modify and delete CacheFS file systems.

This chapter contains these sections:

Changing the Number of NFS Server Daemons

Systems set up for NFS normally run four *nfsd*(1M) daemons. *nfsd* daemons, called NFS server daemons, accept RPC calls from clients. Four NFS server daemons might be inadequate for the amount of NFS traffic on your server. Degraded NFS performance on clients is usually an indication that their server is overloaded.

To change the number of NFS server daemons, create the file */etc/config/nfsd.options* on the server if it doesn't already exist and specify the number of daemons to start at system start up. For example, to have the */etc/init.d/network* script start eight *nfsd* daemons, the */etc/config/nfsd.options* file needs to look like this:

```
# cat /etc/config/nfsd.options
8
```

Modify this number only if a server is overloaded with NFS traffic. In addition to increasing NFS daemons, consider adding another server to your NFS setup. The maximum recommended number of NFS daemons is 24 on a large server. If you increase the number of NFS server daemons, confirm your choice by giving this command:

```
# /usr/etc/nfsstat -s

Server RPC:

calls    badcalls  nullrecv   badlen xdrcall  duphits  dupage
21669881 0         118760787  0      0        12246    7.56
```

If the output shows many null receives, such as in this example, you should consider lowering the number of NFS server daemons. There is no exact formula for choosing the number of NFS daemons, but here are several rules of thumb you can consider:

- One *nfsd* for each CPU plus one to three *nfsd*s as a general resource
- One *nfsd* for each disk controller plus one to three *nfsd*s as a general resource (a logical volume counts as one controller, no matter how many real controllers it is spread over)
- One *nfsd* for each CPU, one *nfsd* for each controller, and one to three *nfsd*s as a general resource

Temporary NFS Mounting

In cases where an NFS client requires directories not listed in its */etc/fstab* file, you can use manual mounting to temporarily make the NFS resource available. With temporary mounting, you need to supply all the necessary information to the *mount*(1M) program through the command line. As with any mount, a temporarily mounted directory requires that a mount point be created before mounting can occur.

For example, to mount */usr/demos* from the server *redwood* to a local mount point */n/demos* with read-only, hard, interrupt, and background options give this command:

```
# mkdir -p /n/demos
# mount –o ro,intr,bg redwood:/usr/demos /n/demos
```

A temporarily mounted directory remains in effect until the system is rebooted or until the superuser manually unmounts it. Use this method for one-time mounts.

Modifying the Automounter Maps

You can modify the automounter maps at any time. Some of your modifications take effect the next time the automounter accesses the map, and others take effect when the system is rebooted. Whether or not booting is required depends on the type of map you modify and the kind of modification you introduce.

Rebooting is generally the most effective way to restart the automounter. You can also kill and restart the automounter using an *automount*(1M) command line. Use this method sparingly, however. (See the *automount*(1M) manual page.)

Modifying the Master Map

The automounter consults the master map only at startup time. A modification to the master map, */etc/auto.master*, takes effect only after the system has been rebooted or *automount* is restarted (see “Modifying Direct Maps”).

Modifying Indirect Maps

You can modify, delete, or add to indirect maps (the files listed in */etc/auto.master*) at any time. Any change takes effect the next time the map is used, which is the next time a mount is requested.

Modifying Direct Maps

Each entry in a direct map is an *automount* mount point, and the daemon mounts itself at these mount points at startup. Therefore, adding or deleting an entry in a direct map takes effect only after you have gracefully killed and restarted the *automount* daemon or rebooted. However, except for the name of the mount point, direct map entries can be modified while the automounter is running. The modifications take effect when the entry is next mounted, because the automounter consults the direct maps whenever a mount must be done.

For instance, suppose you modify the file */etc/auto.indirect* so that the directory */usr/src* is mounted from a different server. The new entry takes effect immediately (if */usr/src* is not mounted at this time) when you try to access it. If it is mounted now, you can wait until auto-unmounting takes place to access it. If this is not satisfactory, unmount with the *umount*(1M) command, notify *automount* that the mount table has changed with the command **`/etc/killall -HUP automount`**, and then access it. The mounting should be done from the new server. However, if you want to delete the entry, you must gracefully kill and restart the *automount* daemon. *automount* must be killed with the SIGTERM signal:

```
# /etc/killall -TERM automount
```

You can then manually restart *automount* or reboot the system.

Note: If gracefully killing and manually restarting *automount* does not work, rebooting the system should always work.

Mount Point Conflicts

You can cause a mount conflict by mounting one directory on top of another. For example, say you have a local home partition mounted on */home*, and you want the automounter to mount other home directories. If the automounter maps specify */home* as a mount point, the automounter hides the local home partition whenever it mounts.

The solution is to mount the server's */home* partition somewhere else, such as */export/home*, for example. You need an entry in */etc/fstab* like this:

```
/net/home     /export/home     efs rw,raw=/dev/rhome 0 0
```

This example assumes that the master file contains a line similar to this:

```
/home         /etc/auto.home
```

It also assumes an entry in */etc/auto.home* like this:

```
terra         terra:/export/home
```

where *terra* is the name of the system.

Modifying CacheFS File System Parameters

Note: Before changing parameters for a cache, you must unmount all file systems in the cache directory with the *umount* command.

The following command changes the value of one or more parameters:

`cfsadmin -u -o` *parameter_list cache_directory*

Note: You can only increase the size of a cache, either by number of blocks or number of inodes. If you want to make a cache smaller, you must remove it and re-create it with new values.

The following commands unmount */local/cache3* and change the **threshfiles** parameter to 65%:

```
# umount /local/cache3
# cfsadmin -u -o threshfiles=65 /local/cache3
```

Displaying Information About Cached File Systems

The following command returns information about all file systems cached under the specified cache directory.

cfsadmin -l *cache_directory*

The following command shows information about the cache directory named */usr/cache/lolita*:

```
# cfsadmin -l /usr/cache/lolita
cfsadmin: list cache FS information
     maxblocks        90%           (122628 blocks)
     minblocks        0%            (0 blocks)
     threshblocks     85%           (115815 blocks)
     hiblocks         85%           (104234 blocks)
     lowblocks        75%           (91971 blocks)
     maxfiles         90%           (206480 files)
     minfiles         0%            (0 files)
     threshfiles      85%           (195009 files)
     hifiles          85%           (175508 files)
     lowfiles         75%           (154860 files)
     maxfilesize      3MB
     lolita:_usr_people_jmy_work:_usr_people_jmy_work
     flags            CFS_DUAL_WRITE CFS_ACCESS_BACKFS
     popsize          65536
     fgsize           256

Current Usage:
     blksused         757
     filesused        124
     flags
```

If there are multiple mount points for a single cache, cfsadmin returns information similar to the following:

```
# cfsadmin -l /usr/cache/bonnie
cfsadmin: list cache FS information
     maxblocks        90%           (122628 blocks)
     minblocks        0%            (0 blocks)
     threshblocks     85%           (115815 blocks)
     hiblocks         85%           (104234 blocks)
     lowblocks        75%           (91971 blocks)
     maxfiles         90%           (206480 files)
     minfiles         0%            (0 files)
     threshfiles      85%           (195009 files)
     hifiles          85%           (175508 files)
     lowfiles         75%           (154860 files)
     maxfilesize      3MB
     bonnie:_jake:_hosts_bonnie_jake
     flags            CFS_DUAL_WRITE CFS_ACCESS_BACKFS
     popsize          65536
     fgsize           256
     bonnie:_depot:_hosts_bonnie_depot
     flags            CFS_DUAL_WRITE CFS_ACCESS_BACKFS
     popsize          65536
     fgsize           256

bonnie:_proj_sherwood_isms:_hosts_bonnie_proj_sherwood_isms
     flags            CFS_DUAL_WRITE CFS_ACCESS_BACKFS
     popsize          65536
     fgsize           256

bonnie:_proj_irix5.3_isms:_hosts_bonnie_proj_irix5.3_isms
     flags            CFS_DUAL_WRITE CFS_ACCESS_BACKFS
     popsize          65536
     fgsize           256

Current Usage:
     blksused         759
     filesused        279
     flags
```

Deleting a CacheFS File System

The following command deletes a file system in a cache:

cfsadmin -d *cache_id cache_directory*

Note: Before deleting a cached file system, you must unmount all the cached files systems for that cache directory.

The cache ID is part of the information returned by *cfsadmin -l*. After deleting one or more of the cached file systems, you must run the *fsck_cachefs* command to correct the resource counts for the cache.

The following commands unmount a cached file system, delete it from the cache, and run *fsck_cachefs:*

```
# umount /usr/work
# cfsadmin -d _dev_dsk_c0t1d0s7 /local/cache1
# fsck_cachefs -t cachefs /local/cache1
```

You can delete all file systems in a particular cache by using **all** as an argument to the *-d* option. The following command deletes all file systems cached under */local/cache1*:

```
# cfsadmin -d all /local/cache1
```

The **all** argument to *-d* also deletes the specified cache directory.

Chapter 6

Troubleshooting ONC3/NFS

This chapter suggests strategies for troubleshooting the ONC3/NFS environment, including automounting. This chapter contains these sections:

General Recommendations

If you experience difficulties with ONC3/NFS, review the ONC3/NFS documentation before trying to debug the problem. In addition to this guide, the *ONC3/NFS Release Notes* and the manual pages for *mount*(1M), *nfsd*(1M), *showmount*(1M), *exportfs*(1M), *rpcinfo*(1M), *mountd*(1M), *inetd*(1M), *fstab*(4), *mtab*(4), *lockd*(1M), *statd*(1M), *automount*(1M), and *exports*(4) contain information you should review. You do not have to understand them fully, but be familiar with the names and functions of relevant daemons and database files.

Be sure to check the console and */var/adm/SYSLOG* for messages about ONC3/NFS or other activity that affects ONC3/NFS performance. Logged messages frequently provide information that helps explain problems and assists with troubleshooting.

Understanding the Mount Process

This section explains the interaction of the various players in the *mount* request. If you understand this interaction, the problem descriptions in this chapter will make more sense. Here is an sample *mount* request:

```
# mount krypton:/usr/src /n/krypton.src
```

These are the steps *mount* goes through to mount a remote file system:

1. *mount* parses */etc/fstab*.
2. *mount* checks to see if the caller is the superuser and if */n/krypton.src* is a directory.
3. *mount* opens */etc/mtab* and checks that this mount has not already been done.
4. *mount* parses the first argument into the system *krypton* and remote directory */usr/src*.
5. *mount* calls library routines to translate the host name (*krypton*) to its Internet Protocol (IP) address. Depending on the host resolution order in */etc/resolv.conf*, *mount* uses */etc/resolv.conf*, the NIS databases, or the DNS databases to determine the NFS server. See *resolver*(4).
6. *mount* calls *krypton*'s *portmap* daemon to get the port number of *mountd*. See *portmap*(1M).
7. *mount* calls *krypton*'s *mountd* and passes it */usr/src*.
8. *krypton*'s *mountd* reads */etc/exports* and looks for the exported file system that contains */usr/src*.
9. *krypton's mountd* calls library routines to expand the host names and network groups in the export list for */usr*.
10. *krypton*'s *mountd* performs a system call on */usr/src* to get the file handle.
11. *krypton*'s *mountd* returns the file handle.
12. *mount* does a *mount* system call with the file handle and */n/krypton.src*.
13. *mount* does a *statfs*(2) call to *krypton*'s NFS server (*nfsd*).
14. *mount* opens */etc/mtab* and adds an entry to the end.

Any of these steps can fail, some of them in more than one way.

Identifying the Point of Failure

When analyzing an NFS problem, keep in mind that NFS, like all network services, has three main points of failure: the server, the client, and the network itself. The debugging strategy outlined below isolates each individual component to find the one that is not working.

Checking Out a Server

If a client is having NFS trouble, check first to make sure the server is up and running. From a client, give this command:

/usr/etc/rpcinfo –p *server_name* **| grep mountd**

This checks whether the server is running. If the server is running, this command displays a list of programs, versions, protocols, and port numbers similar to this:

```
100005    1   tcp   1035  mountd
100005    1   udp   1033  mountd
391004    1   tcp   1037  sgi_mountd
391004    1   udp   1034  sgi_mountd
```

If the *mountd* server is running, use *rpcinfo* to check if the *mountd* server is ready and waiting for mount requests by using the program number and version for *sgi_mountd* returned above. Give this command:

/usr/etc/rpcinfo –u *server_name* **391004 1**

The system responds:

```
program 391004 version 1 ready and waiting
```

If these fail, log in to the server and check its */var/adm/SYSLOG* for messages.

Checking Out a Client

If the server and the network are working, give the command **ps -de** to check your client daemons. *inetd*(1M), *routed*(1M), *portmap*, and four *biod*(1M) and *nfsd* daemons should be running. For example, the command **ps -de** produces output similar to this:

```
PID TTY      TIME COMD
103 ?        0:46 routed
108 ?        0:01 portmap
136 ?        0:00 nfsd
137 ?        0:00 nfsd
138 ?        0:00 nfsd
139 ?        0:00 nfsd
142 ?        0:00 biod
143 ?        0:00 biod
144 ?        0:00 biod
145 ?        0:00 biod
159 ?        0:03 inetd
```

If the daemons are not running on the client, check */var/adm/SYSLOG*, and ensure that *network* and *nfs chkconfig*(1M) flags are on. Rebooting the client almost always clears the problem.

Checking Out the Network

If the server is operative but your system cannot reach it, check the network connections between your system and the server and check */var/adm/SYSLOG*. Visually inspect your network connection. You can also test the logical network connection with various network tools like *ping*(1M). You can also check other systems on your network to see if they can reach the server.

Troubleshooting NFS Common Failures

The sections below describe the most common types of NFS failures. They suggest what to do if your remote mount fails, and what to do when servers do not respond to valid mount requests.

Remote Mount Failed

When network or server problems occur, programs that access hard-mounted remote files fail differently from those that access soft-mounted remote files. Hard-mounted remote file systems cause programs to continue to try until the server responds again. Soft-mounted remote file systems return an error message after trying for a specified number of intervals. See *fstab*(4) for more information.

Programs that access hard-mounted file systems do not respond until the server responds. In this case, NFS displays this message both to the console window and to the system log file */var/adm/SYSLOG*:

```
server not responding
```

On a soft-mounted file system, programs that access a file whose server is inactive get the message:

```
Connection timed out
```

Unfortunately, many IRIX programs do not check return conditions on file system operations, so this error message may not be displayed when accessing soft-mounted files. Nevertheless, an NFS error message is displayed on the console.

Programs Do Not Respond

If programs stop responding while doing file-related work, your NFS server may be inactive. You may see the message:

NFS server *host_name* not responding, still trying

The message includes the host name of the NFS server that is down. This is probably a problem either with one of your NFS servers or with the network

hardware. Attempt to *ping* and *rlogin*(1C) to the server to determine whether the server is down. If you can successfully *rlogin* to the server, its NFS server function is probably disabled.

Programs can also hang if an NIS server becomes inactive.

If your system hangs completely, check the servers from which you have file systems mounted. If one or more of them is down, it is not cause for concern. If you are using hard mounts, your programs will continue automatically when the server comes back up, as if the server had not become inactive. No files are destroyed in such an event.

If a soft-mounted server is inactive, other work should not be affected. Programs that timeout trying to access soft-mounted remote files fail, but you should still be able to use your other file systems.

If all of the servers are running, ask some other users of the same NFS server or servers if they are having trouble. If more than one client is having difficulty getting service, then the problem is likely with the server's NFS daemon *nfsd*. Log in to the server and give the command **ps –de** to see if *nfsd* is running and accumulating CPU time. If not, you may be able to kill and then restart *nfsd*. If this does not work, reboot the server.

If other people seem to be able to use the server, check your network connection and the connection of the server.

Hangs Partway through Boot

If your workstation mounts local file systems after a boot but hangs when it normally would be doing remote mounts, one or more servers may be down or your network connection may be bad. This problem can be avoided entirely by using the background(**bg**) option to *mount* in */etc/fstab* (see *fstab*(4)).

Everything Works Slowly

If access to remote files seems unusually slow, give this command on the server:

ps –de

Check whether the server is being slowed by a runaway daemon. If the server seems to be working and other people are getting good response, make sure your block I/O daemons are running. To check block I/O daemons, give this command on the client:

ps –de | grep biod

This command helps you determine whether processes are hung. Note the current accumulated CPU time, then copy a large remote file and again give this command:

ps –de | grep biod

If there are no *biod*s running, restart the processes by giving this command:

/usr/etc/biod 4

If *biod* is running, check your network connection. The *netstat*(1) command **netstat -i** tells you if packets are being dropped. A packet is a unit of transmission sent across the network. Also, you can use **nfsstat -c** and **nfsstat -s** to tell if the client or server is retransmitting a lot. A retransmission rate of 5% is considered high. Excessive retransmission usually indicates a bad network controller board, a bad network transceiver, a mismatch between board and transceiver, a mismatch between your network controller board and the server's board, or any problem or congestion on the network that causes packet loss.

Cannot Access Remote Devices

You can not use NFS to mount a remote character or block device (that is, a remote tape drive or similar peripheral).

Understanding the Automount Process

This section presents a detailed explanation of how the automounter works that can help you with troubleshooting automounter operation.

There are two distinct stages in the automounter's actions: the initial stage; system start up, when */etc/init.d/network* starts the automounter; and the mounting stage, when a user tries to access a file or directory on a remote system. These two stages, and the effect of map type (direct or indirect) on automounting behavior are described below.

System Startup

At the initial stage, when */etc/init.d/network* invokes *automount*, it opens a user datagram protocol (UDP) socket and registers it with the portmapper service as an NFS server port. It then starts a server daemon that listens for NFS requests on the socket. The parent process proceeds to mount the daemon at its mount points within the file system (as specified by the maps). Through the *mount* system call, it passes the server daemon's port number and an NFS *file handle* that is unique to each mount point. The arguments to the *mount* system call vary according to the kind of fleshiest. For NFS file systems, the call is:

```
mount ("nfs", "/usr", &nfs_args);
```

where *nfs_args* contains the network address for the NFS server. By having the network address in *nfs_args* refer to the local process (the automount daemon), *automount* causes the kernel to treat it as if it were an NFS server. Once the parent process completes its calls to *mount*, it exits, leaving the automount daemon to serve its mount points.

Mounting

In the second stage, when the user actually requests access to a remote file hierarchy, the daemon intercepts the kernel NFS request and looks up the name in the map associated with the directory.

Taking the location (*server:pathname*) of the remote file system from the map, the daemon then mounts the remote file system under the directory */tmp_mnt*. It answers the kernel, saying it is a symbolic link. The kernel sends an NFS READLINK request, and the automounter returns a symbolic link to the real mount point under */tmp_mnt*.

The Effect of Map Types

The behavior of the automounter is affected by whether the name is found in a direct or an indirect map. If the name is found in a direct map, the automounter emulates a symbolic link, as stated above. It responds as if a symbolic link exists at its mount point. In response to a GETATTR, it describes itself as a symbolic link. When the kernel follows up with a READLINK, it returns a path to the *real* mount point for the remote hierarchy in */tmp_mnt*.

If, on the other hand, the name is found in an indirect map, the automounter emulates a directory of symbolic links. It describes itself as a directory. In response to a READLINK, it returns a path to the mount point in */tmp_mnt*, and a *readdir*(3) of the automounter's mount point returns a list of the entries that are currently mounted.

Whether the map is direct or indirect, if the file hierarchy is already mounted and the symbolic link has been read recently, the cached symbolic link is returned immediately. Since the automounter is on the same system, the response is much faster than a READLINK to a remote NFS server. On the other hand, if the file hierarchy is not mounted, a small delay occurs while the mounting takes place.

Troubleshooting CacheFS

A common error message that can occur during a mount is `No space left on device`. The most likely cause of this error is inappropriate allocation of parameters for the cache (see “Cache Resource Parameters” on page 32 for explanations about these parameters).

The following example shows this error for a CacheFS client machine named *sluggo*, caching data from a server *neteng*. One mount has been performed successfully for the cache */cache*. A second mount was attempted and returned the error message `No space left on device`. The *cfsadmin -l* command returned the following:

```
cfsadmin: list cache FS information
     maxblocks        90%           (109109 blocks)
     minblocks        0%            (0 blocks)
     threshblocks     85%           (103047 blocks)
     hiblocks         85%           (92743 blocks)
     lowblocks        75%           (81832 blocks)
     maxfiles         90%           (188570 files)
     minfiles         0%            (0 files)
     threshfiles      85%           (178094 files)
     hifiles          85%           (160285 files)
     lowfiles         75%           (141428 files)
     maxfilesize      3MB
neteng:_home:_home
     flags            CFS_DUAL_WRITE
     popsize          65536
     fgsize           256

Current Usage:
     blksused         406
     filesused        30
     flags            CUSAGE_ACTIVE
```

The *df* command reported the usage statistics for */cache* on *sluggo*. The following shows the *df* command and its returned information:

```
#df -i /cache
Filesystem  Type   blocks   use     avail  %use    iuse   ifree   %iuse Mounted
/dev/root   efs    1939714  1651288 288426 85%     18120  191402  9%    /
```

By default, *minfiles* and *minblocks* are both 0. This means if any files or blocks are allocated, CacheFS uses *threshfiles* and *threshblocks* to determine whether to perform an allocation or fail with the error `ENOSPC`. CacheFS fails an allocation if the usage on the front file system is higher than *threshblocks* or *threshfiles*, whichever is appropriate fro the allocation being done. In this example, the *threshfiles* value is 178094, but only 18120 files are in use. The *threshblocks* value is 103047 (8K blocks) or 1648752 512-byte blocks. The *df* output shows the total usage on the front file system is 1651288 512-byte blocks. This is larger than the threshold, so further block allocations fail.

The possible resolutions for the error are:

- Use *cfsadmin* to increase *minblocks* or *threshblocks* or both. Increasing *threshblocks* should be more effective since */dev/root* is already 85% allocated.
- Remove unnecessary files from */dev/root*. At least 2536 512-byte blocks of data need to be removed; removing more makes the cache more useful. At the current level of utilization, CacheFS needs to continually throw away files to allow room for the new ones.
- Use a separate disk partition for */cache*.

Appendix A

ONC3 / NFS Error Messages

This chapter explains error messages generated during NFS mounting and by the automounter. It contains these sections:

mount Error Messages

This section gives detailed descriptions of the NFS mounting failures that generate error messages.

`/etc/mtab: No such file or directory`
: The mounted file system table is kept in the file */etc/mtab*. This file must exist before *mount*(1M) can succeed.

`mount: ... already mounted`
: The file system that you are trying to mount is already mounted or there is an incorrect entry for it in */etc/mtab*.

`mount: ... Block device required`
: You probably left off the host name (*krypton:*) portion of your entry:

  ```
  # mount krypton:/usr/src /krypton.src
  ```

  The *mount* command assumes you are doing a local mount unless it sees a colon in the file system name or the file system type is `nfs` in */etc/fstab*. See *fstab*(4).

`mount: ... not found in /etc/fstab`

If you use *mount* with only a directory or file system name, but not both, it looks in */etc/fstab* for an entry with file system or directory field matching the argument. For example,

```
# mount /krypton.src
```

searches */etc/fstab* for a line that has a directory name field of */krypton.src*. If it finds an entry, such as this,

```
krypton:/usr/src /krypton.src nfs rw,hard 0 0
```

it mounts as if you had given this command:

```
# mount krypton:/usr/src /krypton.src
```

If you see this message, it means the argument you gave *mount* is not in any of the entries in */etc/fstab*.

`/etc/fstab: No such file or directory`

mount tried to look up the name in */etc/fstab* but there was no */etc/fstab*.

`mount: ... not in hosts database`

The host name you gave is not in the */etc/hosts* database. Check the spelling and the placement of the colon in your *mount* call. Try to *rlogin*(1C) or *rcp*(1C) to the other system.

`mount: directory path must begin with a slash (/).`

The second argument to *mount* is the path of the directory to be mounted. This must be an absolute path starting at /.

`mount: ... server not responding: RPC: Port mapper failure`

Either the server from which you are trying to mount is inactive, or its *portmap*(1M) daemon is inactive or hung. Try logging in to that system. If you can log in, give this command:

```
# /usr/etc/rpcinfo –p hostname
```

This should produce a list of registered program numbers. If it does not, start the *portmap* daemon again. Note that starting the *portmap* daemon again requires that you kill and restart *inetd*(1M), *ypbind*(1M), and *ypserv*(1M). *ypbind* is

active only if you are using the NIS service. The *ypserv* daemon only runs on NIS servers. See *network*(1M) for information about how to stop and restart daemons.

There are two methods for dealing with a server that is inactive or whose *portmap* daemon is not responding. You could reboot the server or you perform these commands:

1. On the server, become the superuser and kill the daemons. Give this command:

   # **/etc/killall portmap inetd ypbind**

2. Start new daemons:

   # **/usr/etc/portmap**
   # **/usr/etc/ypbind**
   # **/usr/etc/inetd**

If you cannot *rlogin* to the server, but the server is operational, check your network connection by trying *rlogin* to some other system. Also check the server's network connection.

`mount: ... server not responding: RPC: Program not registered`

This means *mount* reached the *portmap* daemon but the NFS mount daemon (*rpc.mountd*(1M)) was not registered.

Go to the server and be sure that */usr/etc/rpc.mountd* exists and that an entry appears in */etc/inetd.conf* exactly like this (shown wrapped):

```
mountd/1 dgram rpc/udp wait root
/usr/etc/rpc.mountd mountd
```

Give the command **ps -de** to be sure that the internet daemon (*inetd*) is running. If you had to change */etc/inetd.conf*, give this command:

/etc/killall 1 inetd

This command informs *inetd* that you have changed */etc/inetd.conf*.

`mount: ... No such file or directory`
: Either the remote directory or the local directory does not exist. Check your spelling. Use the *ls*(1) command for the local and remote directories. For SGI systems, check to see if you are attempting to access a hidden file or directory:

  # **showmount -x** *servername*

  and check for file systems exported without the `nohide` option.

`mount: not in export list for ...`
: Your host name is not in the export list for the file system you want to mount from the server. You can get a list of the server's exported file systems with this command:

  # **showmount –e** *servername*

  If the file system you want is not in the list, or your host name or network group name is not in the user list for the file system, log in to the server and check the */etc/exports* file for the correct file system entry. A file system name that appears in the */etc/exports* file but not in the output from *showmount*(1M) indicates that you need to run *exportfs*(1M).

`mount: ... Permission denied`
: This message is a generic indication that some authentication failed on the server. It could simply be that you are not in the export list (see above), the server could not figure out who you are, or the server does not recognize that you are who you say you are. Check the server's */etc/exports*. In the last case, check the consistency of the NIS and local host name and user ID information.

`mount: ... Not a directory`
: Either the remote path or the local path is not a directory. Check your spelling and use the *ls* command for both the local and remote directories.

`mount: ... You must be root to use mount`
: You must do the mount as *root* on your system because it affects the file system for the whole system, not just your directories.

Verbose *automount* Error Messages

The following error messages are likely to be displayed if the automounter fails and the verbose option is on (*automount*(1M) **–v** option). Below each error message is a description of the probable cause of the problem.

`no mount maps specified`
: The automounter was invoked with no maps to serve, and it cannot find the NIS *auto.master* map. Recheck the command, and check for the existence of an NIS *auto.master* map:

```
# ypwhich -m | grep auto.master
```

mapname`: Not found`
: The required map cannot be located. This message is produced only when the **–v** option is given. Check the spelling and pathname of the map name.

`leading space in map entry` *entry* `text in` *mapname*
: The automounter 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:

```
foo
bar geez:/usr/geez
```

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

`bad directory` *directory* `in indirect map` *mapname*
: While scanning an indirect map, the automounter has found an entry directory containing a "`/`". Indirect map directories must be simple names, not pathnames.

`bad directory` *directory* `in direct map` *mapname*
: While scanning a direct map, the automounter has found an entry directory without a leading "`/`". Directories in direct maps must be full pathnames.

`NIS bind failed`
: The automounter was unable to communicate with the *ypbind* daemon. This is information only — the automounter continues to function correctly provided it requires no explicit NIS support. If you need NIS, check to see if there is a *ypbind* daemon running.

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

`WARNING:` *mountpoint* `already mounted on`
: The automounter is attempting to mount over an existing mount point. This is indicative of an automounter internal error (bug).

`server:`*pathname* `already mounted on` *mountpoint*
: The automounter is attempting to mount over a previous mount of the same file system. This could happen if an entry appears both in */etc/fstab* and in an automounter map (either by accident or because the output of **`mount -p`** was redirected to */etc/fstab*). Delete one of the redundant entries.

`can't mount server:`*pathname*`:` *reason*
: The mount daemon on the server refuses to provide a file handle for *server:pathname*. Check the server's export list.

`remount` *server*`:`*pathname* `on` *mountpoint*`: server not responding`
: The automounter has failed to remount a file system it previously unmounted. This message may appear at intervals until the file system is successfully remounted.

`WARNING:` *mountpoint* `not empty`
: The mount point is not an empty directory. The directory contains entries that are hidden while the automounter is mounted there. This is advisory only.

General *automount* Error Messages

This section lists error messages generated by the automounter that can occur at any time.

`WARNING: default option "`*option*`" ignored for map` *mapname*
: Where *option* is an unrecognized default mount option for the map *mapname*.

`option ignored for directory in` *mapname*
: The automounter has detected an unknown mount option. This is advisory only. Correct the entry in the appropriate map.

`bad entry in map` *mapname* `"`*directory*`" map` *mapname*`, directory` *directory*`: bad`
: The map entry is malformed, and the automounter cannot interpret it. Recheck the entry; perhaps there are characters in it that need a special escape sequence.

`Can't get my address`
: The automounter cannot find an entry for the local system in the host database.

`Cannot create UDP service`
: Automounter cannot establish a UDP connection.

`svc_register failed`
: Automounter cannot register itself as an NFS server. Check the kernel configuration file.

`couldn't create` *pathname*`:` *reason*
: Where *pathname* is */tmp_mnt* or the argument to the **–M** command line option.

`Can't mount` *mountpoint*`:` *reason*
: The automounter couldn't mount its daemon at *mountpoint*.

`Can't update` *pathname*
: Where *pathname* is */etc/mtab* it means that the automounter was not able to update the mount table. Check the permissions of the file.

`exiting`
: This is an advisory message only. The automounter has received a SIGTERM (has been killed) and is exiting.

`WARNING:` *pathname*`: line` *line_number*`: bad entry`
Where *pathname* is */etc/mtab* it means that the automounter has detected a malformed entry in the */etc/mtab* file.

server`:`*pathname* `no longer mounted`
The automounter is acknowledging that *server:pathname* which it mounted earlier has been unmounted by the *umount*(1M) command. The automounter notices this within 1 minute of the unmount or immediately if it receives a SIGHUP.

`trymany: servers not responding:` *reason*
No server in a replicated list is responding. This may indicate a network problem.

server`:`*pathname* `–` *linkname* `: dangerous symbolic link`
The automounter is trying to use *server:pathname* as a mount point but it is a symbolic link that resolves to a pathname referencing a mount point outside of */tmp_mnt* (or the mount point set with the **–M** option). The automounter refuses to do this mount because it could cause problems in the system's file system, e.g. mounting on */usr* rather than in */tmp_mnt*.

`host server not responding`
The automounter attempted to contact *server* but received no response.

`Mount of server:`*pathname* `on` *mountpoint*`:` *reason*
The automounter failed to do a mount. This may indicate a server or network problem.

`pathconf: server: server not responding`
The automounter is unable to contact the mount daemon on the server that provides portable operating systems based on UNIX (POSIX) *pathconf*(2) information.

`pathconf: no info for` *server*`:`*pathname*
The automounter failed to get *pathconf* information for *pathname*.

`hierarchical mountpoints:` *pathname1* `and` *pathname2*
: The automounter does not allow its mount points to have a hierarchical relationship. An automounter mount point must not be contained within another automounted file system.

mountpoint`: Not a directory`
: The automounter cannot mount itself on *mountpoint* because *mountpoint* is not a directory. Check the spelling and pathname of the mount point.

`dir` *mountpoint* `must start with '/'`
: Automounter mount point must be given as full pathname. Check the spelling and pathname of the mount point.

mapname`: yp_err`
: Error in looking up an entry in an NIS map. May indicate NIS problems.

hostname`: exports: rpc_err`
: Error getting export list from *hostname*. This indicates a server or network problem.

`nfscast: cannot send packet:` *reason*
: The automounter cannot send a query packet to a server in a list of replicated file system locations.

`nfscast: cannot receive reply:` *reason*
: The automounter cannot receive replies from any of the servers in a list of replicated file system locations.

`nfscast:select:` *reason* `Cannot create socket for nfs: rpc_err`
: These error messages indicate problems attempting to contact servers for a replicated file system. This may indicate a network problem.

`NFS server (`*pid*`@`*mountpoint*`) not responding; still trying`
: An NFS request made to the automount daemon with process identifier *pid* serving *mountpoint* has timed out. The automounter may be temporarily overloaded or dead. Wait a few minutes. If the condition persists, reboot the client or use *fuser*(1M) to find and kill all processes that use automounted directories (or change to a non-automounted directory in the case of a shell), kill the current automount process, and restart it again from the command line.

General CacheFS Errors

This section describes the error messages that may be generated from commands used to administer the CacheFS file system.

cfsadmin Error Messages

This section gives detailed descriptions of the CacheFS *cfsadmin* command failures that generate error messages.

`cfsadmin: must be run by root`
: You must be logged in as root to run *cfsadmin*(1M).

`cfsadmin: Cache name is in use and cannot be modified.`
: This error occurs when you attempt to remove a cache ID from the cache name, if the cache is active (has mounted file systems).

`cfsadmin:` *cachepath* `already exists.`
: The cache directory path cachepath specified with the cachedir option already exists. The last component of the path *cachepath* must not exist when creating a cache. All other path components must exist.

`cfsadmin: creating` *labelpath* `failed.`
: The cache label file *labelpath* could not be created. This message always appears with one of the following:

```
Could not remove labelpath: errmsg
Error creating labelpath: errmsg
Writing labelpath failed: errmsg
Writing labelpath failed on sync: errmsg
```

In each case, the system error is given in *errmsg*.

`cfsadmin: create` *resource* `failed:` *errmsg*
: The cache resource file *resource* could not be created. The system error is given in *errmsg*.

cfsadmin: Cache *cachedir* is in use and cannot be modified.
: The cache *cachedir* was in use when an attempt was made to modify the contents of the cache label. This operation may only be performed when the cache has no mounted file systems.

`cfsadmin: Cache size cannot be reduced, maxblocks current` *p*`%, requested` *n*`%`
An attempt was made to reduce the maximum file system block allocation percentage from *p*% to *n*%. The allocation can only be increased.

`cfsadmin: Cache size cannot be reduced, maxfiles current` *p*`% requested` *n*`%`
An attempt was made to reduce the maximum number of files allocated from *p*% to *n*%. The allocation can only be increased.

`cfsadmin:` *cacheid* `is not a valid cache id.`
The cache identifier given by *cacheid* is not valid. You may have specified an invalid cache identifier on the command line for *cfsadmin*. This can occur when deleting a cache.

`cfsadmin:` *lowblocks* `can't be >=` *hiblocks*`.`
The block allocation specified by *minblocks* is greater than or equal to that specified by *maxblocks*. *minblocks* must be less than *maxblocks*.

`cfsadmin:` *lowfiles* `can't be >=` *hifiles*.
The file allocation specified by *minfiles* is greater than or equal to that specified by *maxfiles*. *minfiles* must be less than *maxfiles*.

`cfsadmin: Could not open` *resource*`:` *errmsg*`, run` *fsck*
The cache resource file *resource* could not be opened. The error message is given by *errmsg*. *cachefs_fsck*(1M) should be run.

`cfsadmin: Could not read cache_usage,` *val*`, run` *fsck*
The cache usage structure could not be read from the resource file. *val* is the return value from *read*(2).

`cfsadmin: Could not open option file` *optpath*
The cache option file *optpath* could not be opened. The entire cache should be removed and reconstructed.

`cfsadmin: Could not read option file` *optpath*
The cache option file *optpath* could not be read. The entire cache should be removed and reconstructed.

`cfsadmin: Reading` *cachelabel* `failed.`
The cache label file *cachelabel* could not be read. This message appears with one of the following messages:

```
Cannot stat file cachelabel: errmsg
File cachelabel does not exist.
Cache label file cachelabel corrupted
Cache label file cachelabel wrong size
Error opening cachelabel: errmsg
Reading cachelabel failed: errmsg
```

The above messages occur when the cache label file is not a regular file or the label contains the incorrect cache version. The system error is given in *errmsg*.

`cfsadmin: Could not open` *resource*: *errmsg*`, run` *fsck*
The resource file *resource* could not be opened in order to enlarge it or mark it as dirty. The error is given in *errmsg*. *cachefs_fsck*(1M) should be run.

`cfsadmin: Resource file has wrong size` *cursize expected*`, run` *fsck*
The size of the resource file is incorrect. Its size is *cursize* when it should be *expected*.

`cfsadmin: Could not write cache_usage,` *val*`, run` *fsck*
The cache usage structure could not be written to the resource file. *val* is the return value from *write*(2).

`cfsadmin: Could not write file,` *val*`, run` *fsck*
The expanded resource file could not be initialized. *val* is the return value from *write*(2).

mount_cachefs Error Messages

This section gives detailed descriptions of the CacheFS mounting failures that generate error messages.

mount_cachefs(1M) is normally executed from *mount*(1M).

`mount_cachefs: must be run by root`
: You must be logged in as root to run *mount*(1M).

`mount_cachefs: mount failed, options do not match.`
: The mount options supplied on the *mount* command are not compatible with the mount options currently set for the cache. This occurs when there are multiple mount points for one cache. All mount points must have the same options.

`mount_cachefs: suid and nosuid are mutually exclusive`
: Both of the options `suid` and `nosuid` have been specified. Only one is allowed.

`mount_cachefs: rw and ro are mutually exclusive`
: Both of the options `rw` and `ro` have been specified. Only one is allowed.

`mount_cachefs: acregmin cannot be greater than acregmax`
: The specified `acregmin` option has a value greater than that for `acregmax`.

`mount_cachefs: acdirmin cannot be greater than acdirmax`
: The specified `acdirmin` option has a value greater than that for `acdirmax`.

`mount_cachefs: only one of non-shared or write-around may be specified`
: Both of the options `non-shared` and `write-around` have been specified. Only one is allowed.

umount_cachefs Error Messages

This section gives detailed descriptions of the CacheFS unmounting failures that generate error messages.

umount_cachefs(1M) is normally executed from *umount*(1M).

`umount_cachefs: must be run by root`
: You must be logged in as root to run *umount*(1M).

`umount_cachefs: warning:` *dir* `not in mtab`
: The mount point directory *dir* has no entry in the mount table. This means that the mount table has been corrupted. The *umount* command can still be successful; however, the back file system can not be unmounted.

`umount_cachefs: could not exec /sbin/umount on back file system` *errmsg*
: An attempt was made to run *umount* on the back file system and failed. The system error is given in *errmsg*.

Index

S

T

U

V

W

Tell Us About This Manual

As a user of Silicon Graphics products, you can help us to better understand your needs and to improve the quality of our documentation.

Any information that you provide will be useful. Here is a list of suggested topics:

- General impression of the document
- Omission of material that you expected to find
- Technical errors
- Relevance of the material to the job you had to do
- Quality of the printing and binding

Please send the title and part number of the document with your comments. The part number for this document is 007-0850-070.

Thank you!

Three Ways to Reach Us

- To send your comments by **electronic mail**, use either of these addresses:
  - On the Internet: techpubs@sgi.com
  - For UUCP mail (through any backbone site): *[your_site]*!sgi!techpubs
- To **fax** your comments (or annotated copies of manual pages), use this fax number: 650-965-0964
- To send your comments by **traditional mail**, use this address:

  Technical Publications
  Silicon Graphics, Inc.
  2011 North Shoreline Boulevard, M/S 535
  Mountain View, California 94043-1389