

SCO® NFS®

Administrator's Guide



SCO[®] NFS[®]

Administrator's Guide

© 1983-1992 The Santa Cruz Operation, Inc.
© 1980-1992 Microsoft Corporation.
© 1989-1992 UNIX System Laboratories, Inc.
All Rights Reserved.

No part of this publication may be reproduced, transmitted, stored in a retrieval system, nor translated into any human or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of the copyright owner, The Santa Cruz Operation, Inc., 400 Encinal, Santa Cruz, California, 95060, U.S.A. Copyright infringement is a serious matter under the United States and foreign Copyright Laws.

The copyrighted software that accompanies this manual is licensed to the End User only for use in strict accordance with the End User License Agreement, which should be read carefully before commencing use of the software. Information in this document is subject to change without notice and does not represent a commitment on the part of The Santa Cruz Operation, Inc.

SCO OPEN DESKTOP Software is commercial computer software and, together with any related documentation, is subject to the restrictions on U.S. Government use as set forth below.

If this procurement is for a DOD agency, the following DFAR Restricted Rights Legend applies:

RESTRICTED RIGHTS LEGEND: Use, duplication or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of rights in Technical Data and Computer Software Clause at DFARS 252.227-7013. Contractor/Manufacturer is The Santa Cruz Operation, Inc., 400 Encinal Street, Santa Cruz, CA 95060.

If this procurement is for a civilian government agency, the following FAR Restricted Rights Legend applies:

RESTRICTED RIGHTS LEGEND: This computer software is submitted with restricted rights under Government Contract No. _____ (and Subcontract No. _____, if appropriate). It may not be used, reproduced, or disclosed by the Government except as provided in Paragraph (g)(3)(i) of FAR Clause 52.227-14 or as otherwise expressly stated in the contract. Contractor/Manufacturer is The Santa Cruz Operation, Inc., 400 Encinal Street, Santa Cruz, CA 95060.

SCO, SCO Open Desktop, SCO and The Santa Cruz Operation, the SCO Open Desktop logo, and the SCO logo are registered trademarks of The Santa Cruz Operation, Inc. in the USA and other countries.

All other brand and product names are or may be trademarks of, and are used to identify products or services of, their respective owners.

SCO NFS is derived from Interactive Systems Corporation
SYSTEM V STREAMS NFS, a joint development of ISC and Sun Microsystems.

Date: 24 June 1992

Document version: 1.2.0A

Preface ***1***

About this guide	1
Organization of this guide	2
Using this guide	2
Conventions used in this guide	2
Reference pages	3
Related reading	4

Chapter 1

Introduction ***5***

Network background	5
Network operation	5
Network architecture and layered protocols	6
TCP/IP	7
How NFS fits into the network	7
NFS features	8
Planning a distributed computing environment	9
Gauging load distribution	9
Naming filesystems	10
Centralizing administration	11

Chapter 2

NFS: Network File System ***13***

Basic concepts	13
How files interact	13
How daemons interact	14
Managing user accounts	14
Configuring NFS	15
Setting up an NFS server	15
Enabling export	15
Checking the /etc/hosts file	16
Setting up an NFS client	17
Mounting the remote filesystem	17
Enabling default mounting	17
Enabling user mounting	19

Starting and stopping NFS	20
Administering NFS	20
Debugging NFS	20
General hints	21
Remote mount failed	23
Programs are hung	26
Everything works slowly	26
Serial number validation errors	27
Incompatibilities with remote filesystems	27
File operations not supported	27
Cannot access remote devices	27
Cannot access different filesystem types	28
Cannot access indirect filesystems	28
Handling clock skew in user programs	28

Chapter 3

NIS: Network Information Service **31**

Understanding NIS	31
NIS maps	32
NIS domain	33
NIS servers	33
NIS daemons	34
NIS utilities	35
NIS user accounts	35
Understanding YP_MAP_X_LATE	36
Security limitations on configuration	36
Planning an NIS configuration	37
Selecting domains	37
Selecting servers	38
Security	38
Configuring NIS	39
Setting up an NIS server	39
Initializing NIS	40
Starting NIS	47
Managing NIS maps	48
Map manipulation utilities	48
Modifying existing maps	49
Propagating an NIS map	50
Using cron	50

Using the interactive approach	51
Using yppush on the master server	51
Making new NIS maps	51
Creating a new map from an ASCII file	52
Creating a new map from standard input	53
Propagating a new NIS map	53
Managing user accounts	54
NIS account types	54
Administration utilities	54
Creating NIS accounts	55
Creating a distributed account	55
Creating a local account	56
Modifying NIS accounts	56
Changing a distributed account to a local account	56
Changing a local account to a distributed account	57
Modifying password file entries	57
Removing password file entries	58
Administering an NIS network	58
Changing server configuration	58
Adding NIS servers	58
Removing servers	59
Changing the master server	60
Debugging NIS	62
Checking TCP/IP	62
Checking the protected password file	62
Different versions of an NIS map	62
ypserv crashes	63
Commands hang	64
NIS unavailable	66
ypbind crashes	66
NIS interoperability	67
Security under NIS	68
Map integration restrictions in Secure Mode	68
Special NIS password change	68
Manual pages covering security	69
Directory of NIS services	69

Understanding Automounter	73
How Automounter works	73
Automounter maps	77
Planning an Automounter configuration	78
Configuring Automounter	79
Preparing the maps	79
Writing a master map	79
Writing an indirect map	82
Writing a direct map	84
Specifying subdirectories	87
Metacharacters	88
Environment variables	90
Including other maps	91
Starting Automounter	91
Administering Automounter	93
Restarting Automounter	93
Modifying the maps	93
Modifying the master map	93
Modifying indirect maps	93
Modifying direct maps	94
Mount point conflicts	94
Unexpected filesystem behavior	95
Increasing mount table entries	95
Debugging Automounter	96
General error messages	96
Error messages generated by the verbose option	98
Index	101

Preface

Network File System (NFS) is a software product that enables you to mount directories across the network and treat remote files as if they were local. NFS was developed by Sun Microsystems® as a standard for the exchange of data between different machines and operating systems.

As a system administrator, you are responsible for configuring an NFS network, solving network problems, adding new machines, and adding new users. To accomplish these tasks, SCO NFS provides the following services:

- NFS itself, which allows clients to mount and unmount directories from remote servers
- Network Information Service (NIS), which allows management of configuration files across the network
- Automounter, which allows remote file hierarchies to be mounted automatically and transparently, as needed

About this guide

Before reading this guide, you should be familiar with basic networking concepts, and with NFS from a user's perspective. Chapter 1 of this guide provides this background information. You should also be familiar with SCO TCP/IP, because NFS uses TCP/IP as the network transport. For information on installation, see the *SCO NFS Release and Installation Notes*.

Organization of this guide

This guide is organized into these chapters:

- This preface describes the conventions and formats used in this guide.
- Chapter 1, “Introduction,” describes NFS in general and suggests guidelines for planning an NFS configuration.
- Chapter 2, “NFS: Network File System,” describes NFS from the perspective of a system administrator. Information on how NFS works, information to help you configure NFS servers and clients, and pointers to help you debug NFS are included.
- Chapter 3, “NIS: Network Information Service,” describes how NIS works, how to configure NIS servers, and how to manage NIS to centralize administrative tasks.
- Chapter 4, “Automounter,” describes how Automounter works and how to configure an NFS network for automatic filesystem mounting and unmounting.

Using this guide

You should first read background and planning information in Chapter 1, particularly if you are new to distributed computing services. After you have completed planning your NFS network, you will be ready to configure the individual components. If you are setting up a simple network with a small number of servers and you wish to configure only NFS, proceed to Chapter 2. If you wish to use Automounter, you must configure NFS according to the instructions in Chapter 2 before configuring Automounter as described in Chapter 4. If you wish to use NIS for centralized administration, it is recommended that you first configure NIS as instructed in Chapter 3, then configure NFS and Automounter.

Conventions used in this guide

This guide uses the following notational conventions:

bold	represents commands, command options, parameters in files, data structures, and daemons
BOLD CAPS	represent parameters contained in files
<i>italics</i>	represent files and directories
<i>bold italics</i>	represent variables that you supply; for example, in the command argument <code>path:pathname</code> , the variable <i>pathname</i> is replaced with an actual pathname when you type the command

<>	represent special keys that you press; for example, <Ctrl>x means to hold down the Control key and press the x key simultaneously, then release them
Courier	represents system responses, excerpts from files, and programming examples

Reference pages

Reference pages, also called manual pages or man pages, describe commands, daemons, and files, and other items related to a given product. Reference pages can be viewed online using the **man** command. For example, to get information about the **showmount** command, you enter **man showmount** at the UNIX system prompt.

Commands that have reference pages have one or more letters associated with them, such as **tar(C)**. The letters in parentheses tell you which reference page section to look in to find information on that command. The letters also tell you which product the command belongs to. For example, commands with the (ADM) suffix are UNIX system administration commands. Commands with the (NADM) suffix are NFS administration commands. The following letters are relevant to NFS:

C	UNIX system user-level commands
ADM	UNIX system administration commands
NADM	NFS network administration commands
NC	NFS network commands
NF	NFS network file formats
NS	NFS network system calls and library routines

For information on anything with the (NADM), (NC), (NF), or (NS) suffixes, for example **portmap(NADM)**, refer to the *SCO NFS Administrator's Reference*. For information on manual pages that have the (ADMN), (ADMP), (SFF), or (TC) suffixes, for example **rlogind(ADMN)**, refer to the *SCO TCP/IP Command Reference*. For all other commands, check your *SCO UNIX Operating System User's Reference* for a list of manual page sections and their abbreviations.

Related reading

Refer to the other manuals in this set for more information on the various aspects of NFS:

- The *SCO NFS Release and Installation Notes* describe how to install NFS and provide the latest information on the product.
- The *SCO NFS Programmer's Guide and Command Reference* describes the SCO NFS Development System, as well as all administrative commands, daemons, and files associated with NFS.

For more information about NFS functionality and administration, refer to *Managing NFS and NIS* by Hal Stern, published by O'Reilly & Associates, 1991 (O'Reilly & Associates, Inc., 632 Petaluma Ave., Sebastopol, CA 95472; ISBN 0-937175-75-7).

For information about the NFS protocol, refer to "NFS: Network File System Protocol Specification" (RFC 1094), by Bill Nowicki. You can get this document from Jon Postel, RFC Editor, USC Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90292-6695.

RFC's (Requests For Comment) are a series of working papers devoted to networking topics. They are available on the Internet Anonymous FTP site — `NIC@NIC.ddn.mil` — or through e-mail — `SERVICE@NIC.ddn.mil` with the subject line "RFC `rfcnumber`." (These FTP sites are subject to change without notice.)

Chapter 1

Introduction

The SCO Network File System (NFS) software product allows administrators to mount directories located on remote hosts across the network and then treat those directories as if they were local. NFS was originally developed by Sun Microsystems as a standard for the exchange of data between different machines and operating systems. This chapter presents background information on NFS, describes its features, and suggests guidelines for planning a network configuration. Readers who are new to concepts and terms of network operation should read this entire chapter. Experienced network administrators can skip the section “Network background,” glance at the section “NFS features,” and read the section “Planning a network configuration.”

Network background

Because NFS operates on a network, you need some knowledge of how a network operates and how data is transmitted across a network as a background for understanding NFS. This section contains that information.

Network operation

A distributed network of machines, such as the sample network in Figure 1-1, can provide more aggregate computing power than a mainframe computer, with far less variation in response time over the course of the day. Thus, a network is generally more cost-effective than a central mainframe. However, even in a network environment, sharing programs and data can sometimes be difficult. Either files have to be copied to each machine where they are needed, or users have to log into the remote machine where the required files are located. Network logins are time-consuming, and having multiple copies of a file becomes confusing as uncoordinated changes are made to separate copies.

To solve this problem, a distributed filesystem was designed to permit client systems to access shared files on a remote system. Client machines, also called *clients*, request resources provided by other machines, called *servers*. A server machine makes particular filesystems available, and client machines can mount these as local filesystems. Thus, users can access remote files as if they were on their local machine.

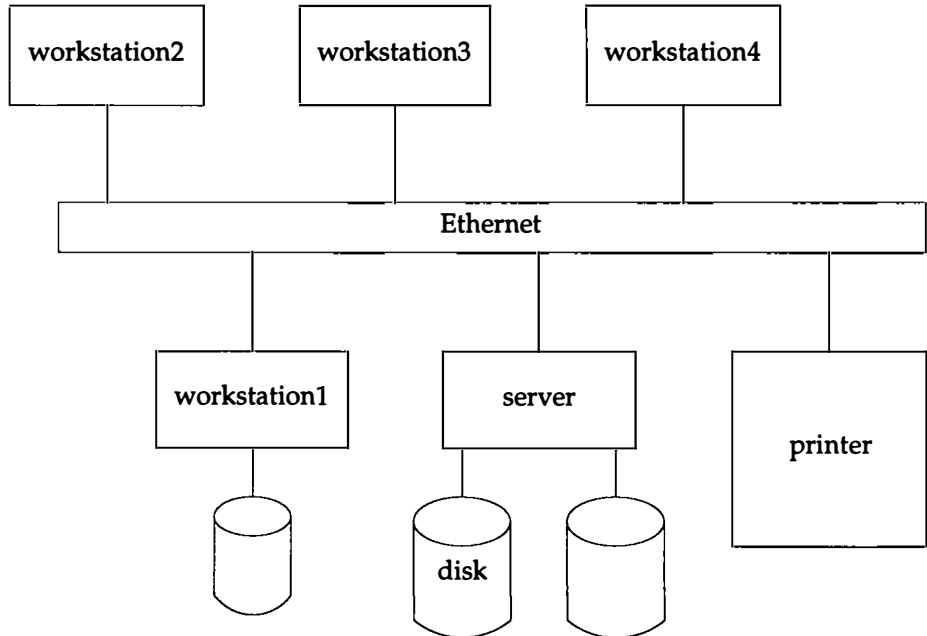


Figure 1-1 Sample network

Network architecture and layered protocols

When machines communicate over a network, a number of problems can arise:

- Hardware may fail.
- The network may become congested.
- Data may be lost, corrupted, duplicated, or scrambled.

Due to the number and complexity of problems, protocols to solve them are layered, with lower layers closer to the hardware and higher layers closer to the user. Each layer in the protocol addresses a specific task, and assumes the support of underlying layers. The number of layers and tasks that the layers perform depends on who defines these layers. The following section describes TCP/IP, the set of programs and protocols used by SCO NFS.

TCP/IP

Transmission Control Protocol with Internet Protocol (TCP/IP) has four software layers built on an underlying hardware layer. Its model is shown in Figure 1-2.

Layer	Functionality
4	Application
3	Transport
2	Internet
1	Network Interface

Figure 1-2 TCP/IP model

The layers operate as follows:

- Network Interface accepts and transmits data over the network
- Internet Layer takes care of communication among machines
- Transport Layer provides communication between application programs
- Application Layer accesses the Internet, and sends and receives data

For a more detailed explanation of the TCP/IP protocols, refer to the *SCO TCP/IP Administrator's Guide*.

How NFS fits into the network

NFS fits into the application layer of the TCP/IP model. NFS was designed to fit into the network services architecture, and not to extend the operating system onto the network. Thus, NFS is an interface to allow a variety of machines and operating systems to play the roles of client and server, but it is not a distributed operating system.

The goal with NFS is to make all disks available as needed. Individual workstations have access to all information residing anywhere on the network. Printers and supercomputers may also be available on the network.

NFS features

NFS provides a number of features:

- Data Exchange Standard

NFS is a standard for the exchange of data between different machines and operating systems.

- Transparent Information Access

You are able to get directly to the files you want without knowing the network address of the data. To a user or an application, there appears to be no difference between reading or writing a file located on a private disk, and reading or writing a file on a disk attached to a remote computer.

- Extensibility

A distributed system must have an architecture that allows integration of new software technologies without disturbing the existing software environment. To allow this, NFS provides network services rather than a new network operating system. This means that NFS does not depend on extending the underlying operating system onto the network, but instead offers a set of protocols for data exchange. These protocols can be extended easily.

- Easy Network Administration

The UNIX systems have a convenient set of maintenance commands that have been developed over the years. NFS provides some new utilities for network administration, and has retained most of the existing utilities.

- Reliability

The file server protocol is designed so that client workstations can continue to operate even when the server crashes and reboots. Continuation after reboot is achieved without making assumptions about the fail-stop nature of the underlying server hardware.

Another factor in reliability is a stateless server (that is, the server treats each transaction independently), which provides robust performance in the face of client, server, or network failures. Should a client fail, it is not necessary for a server or human administrator to take any action to continue normal operation. Should a server or the network fail, it is necessary only that clients continue to attempt to complete NFS operations until the server or the network become operational. This robustness is especially important in a complex network of heterogeneous systems.

- High Performance

The flexibility of NFS allows configuration for a variety of cost and performance trade-offs. For example, configuring servers with large, high-performance disks and clients with no disks may yield better performance at lower cost than having many machines with small, inexpensive disks. Furthermore, it is possible to distribute the filesystem data across many servers and achieve the added benefit of multiprocessing without losing transparency. In the case of read-only files, copies can be kept on several servers to avoid bottlenecks.

Several performance enhancements have been made to NFS, such as *fast paths* for frequent operations, asynchronous service of multiple requests, caching of disk blocks, and asynchronous read-ahead and write-behind. The fact that caching and read-ahead occur on both client and server effectively increases the cache size and read-ahead distance. Caching and read-ahead do not add state to the server. Aside from improved performance, nothing is lost if cached information is discarded. In the case of write-behind, both the client and the server attempt to flush critical information to disk whenever necessary, to reduce the impact of an unanticipated failure. The client does not free write-behind blocks until the server has verified that the data has been written.

Planning a distributed computing environment

SCO NFS Release 1.2.0, with its NIS and Automounter utilities, provide a powerful tool for managing a distributed computing environment. A carefully planned implementation of such a network can streamline administrative procedures and improve access to resources. However, faulty organization can lead to increased administrative complexity and poor network performance. While you will certainly discover the best solution for your system by trial and error, the following sections suggest ways to optimize your NFS configuration.

Gauging load distribution

NFS enables you to create a work environment in which all applications and files appear local; in fact, NFS and its related utilities disguise the fact that you are on a network at all. This is accomplished by distributing applications and data files in such a way that resources are accessed transparently and quickly.

Before configuring an NFS network or when adding new management facilities to an existing one, you must assess patterns of use in your network and weigh them against your resources. For example, in a network with a small number of machines, it may be most efficient to have each host perform some

client and some server tasks. A larger network, on the other hand, will almost certainly benefit from designating some hosts exclusively as servers. Each arrangement can use the remote filesystem mounting options offered by NFS and Automounter in different ways.

From an administrator's point of view, Automounter should be used with the objective of reducing the number of active mounts. Manual pages (conventionally residing in */usr/man*) are a good candidate for Automounter management. Because they are used relatively rarely, they can be removed from machines with home directories and mounted only as needed. However, a frequently used service, such as the news spool, would be a poor candidate for Automounter management because it is in constant use and frequent mounting and unmounting operations would impair access to the service. It would be better managed through a conventional NFS mount.

In an environment where a large number of servers export filesystems that are available for mounting by Automounter, network efficiency may be improved by duplicating commonly used files. Besides providing fault tolerance should one of the servers crash, it allows clients to find the closest or least occupied server, improving response time for remote file requests.

Naming filesystems

Filesystems should be named logically and consistently, in ways that both conform to users' expectations and which simplify administrator's duties. Whenever possible, it is desirable to first construct a naming scheme based on the types of services available to a client, then assign filesystem names and fill them with applications or users.

Here are some suggestions for assigning names in an NFS environment:

- When creating file hierarchies, try to maintain conventional UNIX-system naming schemes, such as */bin* and */lib*.
- Whenever possible, use the same pathnames on the client as on the server. Parallel pathnames will aid in troubleshooting on the server.
- Choose a naming scheme that allows for growth; you will not want to rename filesystems when you increase server capability on your system.

Centralizing administration

SCO NFS includes the Network Information Service (NIS), a distributed database system that allows you to manage commonly used configuration files (such as */etc/hosts* and */etc/passwd*) with a single utility. When changes are to be made in these files or when a new host is added to the system, you can modify the files on a single server, rather than manually editing or adding files on each host in the system.

NIS is most useful on a large network; it is also useful in networks making use of Automounter, because Automounter configuration files can also be managed with NIS. However, a smaller, relatively static network may not warrant the administrative overhead of maintaining NIS maps on a system of servers. Also, if you must maintain a system with a high degree of security, it may not be prudent for NIS to manage sensitive files (that is, the */etc/passwd* and */etc/group* files). See the “NIS: Network Information Service” chapter of this guide for more information on NIS and security.

In planning a complex NFS network, it is recommended that the administrative aspect be considered early in the development phase, and that the distributed filesystem be designed with NIS management in mind.

Chapter 2

NFS: Network File System

This chapter describes NFS from the viewpoint of the system administrator. Included is a discussion of how NFS works, information to help you configure NFS servers and clients, and pointers to help you debug NFS.

In NFS, a server exports directories to be shared and clients mount the directories to access the files in them. The following subsections describe the general interaction of files and daemons in these activities. For details of what happens when a client mounts a remote filesystem, see the section “Remote mount failed” later in this chapter.

Basic concepts

The following sections explain the basic functionality of NFS and the daemons that provide it.

How files interact

To export a filesystem, the server must list it along with access rights in its */etc/exports* file. In addition, the server must have a name and address for each client to receive services in its */etc/hosts* file.

You can use SCO NFS to export the following filesystem types:

- EAFS
- S51K
- UNIX

All exported filesystems appear to clients as “SCO NFS” filesystems, regardless of their original filesystem types.

To mount a directory, the client simply issues a **mount** command, specifying the filesystem to be mounted, the filesystem's server, and the local mount point. The **mount(NADM)** command automatically adds an entry to the client's */etc/mnttab(F)* file; the **umount** command automatically deletes the entry in this file. Consequently, this file maintains a current list of mounted filesystems.

The client can also mount remote filesystems by default when the system starts by placing an entry in */etc/default/filesys*. The client's */etc/default/filesys* file must list all remote filesystems for which access is desired and their locations.

NOTE NFS supports resource sharing by exporting filesystems from servers. However, the applications that access these shared filesystems reside on the client host and use the client's CPU and RAM. Therefore, filesystems are distributed in NFS, not applications.

How daemons interact

Two remote programs provide the NFS service — **mountd(NADM)** and **nfsd(NADM)**. A client's **mount** request calls **mountd**, which checks the server's */etc/exports* file for access permission of the client and returns a pointer to a filesystem. After **mount** completes, access to that mount point directory and below goes through the **nfsd** daemon on the server system. Client-kernel file access requests (delayed-write and read-ahead) are handled by the **biod** (see **nfsd(NADM)**) daemons on the client.

NFS client requests are fulfilled through Remote Procedure Calls (RPC). Using RPC, the client process can cause the server process to execute a procedure call, as if the client process had executed the procedure call in its own address space. The **portmap(NADM)** daemon maps RPC procedure calls to the appropriate TCP/IP port numbers (that is, to these calling processes) and thus is also an essential part of the NFS service.

pcnfsd is used by machines running the DOS version of NFS. For further information, refer to the **pcnfsd(NADM)** manual page.

Managing user accounts

Within a network environment, each user and group must have a unique identification number across the entire network, rather than just on the home machine. The user ID should be a number unique to the user. A system knows the user by the ID number associated with the login name; therefore, a login name must have the same user ID number in all password files of machines that are networked in a local domain.

CAUTION Failure to keep IDs unique prevents users from moving files between directories on different machines, because the system responds as if the directories were owned by two different users. In addition, file ownership may become confused when an NFS server exports a directory to an NFS client whose password file contains users with UIDs matching those of different users on the NFS server.

If you wish to maintain user equivalence, each new user must be added to the proper password files for all machines on the system. For general information on how to do this, see the *SCO TCP/IP User's and Administrator's Guide*. If you are using NIS, see also the “NIS: Network Information Service” chapter of this guide.

Configuring NFS

Preparing NFS for use across a network entails setting up NFS servers and clients, mounting remote filesystems, and starting the NFS daemons. The following sections explain these operations.

Setting up an NFS server

Setting up an NFS server involves enabling a machine to export a filesystem and checking the */etc/hosts* file. All examples in this section use the server named *jupiter*.

Enabling export

Follow these steps to enable a machine to export a filesystem.

1. Log in as *root*.
2. Change (*cd*) to the */etc* directory. Edit the */exports* file using your preferred editor to add the mount-point pathname of the filesystem to be exported. For example:

```
/usr/man                # export to any machine
/usr2 -access=venus:mars # export to only these machines
```

See the *exports(NF)* manual page for syntax and contents of this file.

NOTE An NFS server can only export filesystems of its own. For example, if a client *mars* remote-mounted the filesystem */tools* from the server *jupiter*, *mars* cannot in turn export */tools* to a third client. Thus, NFS does not support “daisy-chaining” (third-party mounting).

3. Make sure **/etc/mountd** is running. This daemon is started from the NFS startup script, usually **/etc/nfs**, and must be running for a remote mount to succeed. Check **/etc/nfs** for lines such as the following to enable **mountd**:

```
if [ -r /etc/exports]
then
    echo " mountd\c"
    mountd
fi
```

4. Make sure some number of **nfsd** NFS daemon processes are running on the NFS server. The actual number depends on the number of client NFS requests that the server should be able to handle concurrently, and thus depends on the speed and capacity of the server machine.

Check the NFS startup script, **/etc/nfs**, for lines like these to enable **nfsd**:

```
if [ -r /etc/exports]
then
    echo " nfsd(x4)\c"
    nfsd 4
fi
```

This example invokes four **nfsd** daemons.

The super user can enable **nfsd** at any time by typing:

/etc/nfsd 4

5. Export the files in **/etc/exports** by entering the command:

/etc/exportfs -a

Once the server is enabled, **exportfs** is run as part of the startup script whenever NFS is started. See the **exportfs(NADM)** manual page for more information,

After these steps, the NFS server should be able to export the named file-system.

Checking the /etc/hosts file

The name, network address, and aliases of each NFS client or server host must appear in the **/etc/hosts** file. If this information is not present, log in as the super user and edit the file.

Here is a sample file:

```
102.6.3.76      jupiter
192.9.1.20      earth
176.2.5.42      venus
142.9.7.36      mars
```

See the **hosts(SFF)** manual page for the syntax and contents of the file.

Setting up an NFS client

To enable an NFS client, you need only mount a remote filesystem after NFS is started. You can do so by issuing the **mount** command or by specifying remote filesystems in */etc/default/filesys* to be mounted by default whenever NFS is started.

Mounting the remote filesystem

You can remote-mount any exported filesystem onto your client machine, as long as the filesystem's server can be reached over the network and the client machine is included in the server's */etc/exports* list for that filesystem. On the machine where the filesystem is to be mounted, the super user executes the **mount** command. For example, to mount the manual pages from remote machine *jupiter* on the directory */usr/jupiter.man*, enter:

```
mount -r -f NFS,soft,bg jupiter:/usr/man /usr/jupiter.man
```

This example illustrates the use of the **soft** option to specify a soft mount, which means that the client does not retry the mount operation if the server does not respond to the first request. The default is a hard mount, which means that the client continues to try to mount the requested directory even if the server does not respond. A soft mount is recommended for read-only directories, such as online manual pages. The underlying reasoning is that with a soft mount, halting the server does not halt the client.

This example also illustrates the use of the **bg** to specify that the mount take place in the background. The default specifies foreground mounting; however, if the server is not responding, the client hangs while waiting for the server to respond. Background mounting is recommended for the same reason as soft mounting: to protect the client if the server is not available.

To make sure the filesystem is mounted where it is expected to be, use the **mount** command without any arguments. This displays the currently mounted filesystems.

Enabling default mounting

If you mount certain remote filesystems regularly, you can specify them for default mounting in */etc/default/filesys*. If a filesystem has an entry in */etc/default/filesys*, it is mounted whenever the client is rebooted. A client uses its */etc/default/filesys* file to keep track of specific remote filesystems and directory locations, called *mount points*, through which the client can access directories.

Make sure that all filesystem information, including access rights, appears in this file. If any information is not present, edit the file to add it. In addition, use **mkdir** to create any mount points that do not already exist in the filesystem.

Here is a sample entry:

```
bdev=jupiter:/usr/man mountdir=/usr/man \
    fsck=no rcfscck=no rcmount=yes \
    mount=no fstyp=NFS \
    nfsopts="soft,bg,intr,rsiz=1024,wsiz=1024"
```

Table 2-1 explains the entry.

Table 2-1 /etc/default/filesys format

Entry	Value	Indicates
bdev	jupiter:/usr/man	the remote filesystem to mount
mountdir	/usr/man	where to mount the filesystem
fsck	no	whether to check the filesystem with fsck (ADM) when a user mounts the filesystem with mnt
rcfsck	no	whether to check the filesystem with fsck (ADM) when the system enters multiuser mode
rcmount	yes	whether the filesystem is automatically mounted when the system enters multiuser mode
mount	no	whether users can mount and unmount this filesystem
fstyp	NFS	the filesystem type; this must always be set to NFS for SCO NFS to function correctly
nfsopts	soft	for recovery after server failure; recommended only for read-only filesystems
	bg	so the mount takes place in the background; recommended to prevent the client from hanging if the server is not responding
	intr	so interrupts work across the network

If you have a 3Com EtherLink 3C501 card on the client or server, you must set the **rsiz** and **wsiz** parameters in **nfsopts** as shown in the sample entry to prevent excessive NFS retransmissions.

See the *filesys(F)* manual page for a complete description of the syntax and contents of */etc/default/filesys*.

NOTE The **mkdev fs** command cannot be used to add NFS entries to */etc/default/filesys*.

Enabling user mounting

Under normal circumstances, only the super-user can mount filesystems using NFS. However, you can enable users to mount and unmount selected filesystems using the **mnt(C)** and **umnt(C)** commands. Enabling user mounting can reduce the amount of time it takes to bring your system into multiuser mode and reduce the amount of network traffic by ensuring that filesystems are mounted only when needed.

To enable user mounting, you must create an entry in */etc/default/filesys* for each filesystem to be mounted. The syntax that enables user mounting is the same as for default mounting (listed in the previous section, “Enabling default mounting”) except for two parameters:

mount=yes enables users to mount and unmount the filesystem
rcmount=no prevents the filesystem from being mounted automatically when the system enters multiuser mode

The following sample entry allows a user to mount the */usr/man* directory from the server *jupiter* (if */usr/man* is exported by *jupiter*):

```
bdev=jupiter:/usr/man mountdir=/usr/man \
fsck=no rfsck=no rcmount=no \
mount=yes fstyp=NFS \
nfsopts="soft,bg,intr,rsiz=1024,wsiz=1024"
```

The user can mount this directory by issuing the command **mnt** with the mountpoint listed in the **mountdir** parameter, for example:

```
mnt /usr/man
```

The filesystem can be unmounted using the **umnt** command with the mount point as argument.

You should instruct your users in proper mounting procedures:

- Filesystems should be unmounted when not in regular use.
- The **mnt** and **umnt** commands cannot be issued if the user's current working directory is the mount point listed in */etc/default/filesys*. Otherwise, an error message `Mount point busy` will be returned.
- Users can display a list of mounted filesystems by issuing the **mnt** command without an argument. The command **mnt -t** will list the contents of */etc/default/filesys*; users are unable to mount any file system if the value given for the line “Mount if requested” is “No.”

Refer to the **mnt(C)** manual page for more information.

NOTE You can also enable user mounting transparently by using Automounter. If you create an Automounter map for an exported filesystem rather than an entry in */etc/default/filesys*, the filesystem will be mounted automatically whenever the filesystem or one of its files are accessed. Such a mount is transparent to the user, and the filesystem unmounts automatically after a certain period of inactivity. For more information, see the “Automounter” chapter of this guide.

Starting and stopping NFS

NFS must be started by a shell script invoked by `init(M)` when the system is brought up to multiuser mode (run level 2). This is normally configured automatically during installation by linking the script */etc/nfs* to */etc/rc2.d/S89nfs*. You must start NFS on at least one server and at least one client to have a functional NFS network.

To stop NFS on a machine, log in as *root* and enter the following command:

```
/etc/nfs stop
```

When you stop NFS on a node, its resources cannot be accessed from any other node. NFS is shut down automatically when the system is taken to run levels 0, 1, or 6.

Administering NFS

The following sections explain procedures for NFS debugging and maintenance.

Debugging NFS

Most problems involving System V NFS network services lie in one of the following four areas, which are listed in order of probability:

- The network-access control policies do not allow the operation, or architectural constraints prevent it.
- The client software or environment is not functioning correctly.
- The server software or environment is not functioning correctly.
- The network is not functioning correctly.

To debug NFS, you should be familiar with how NFS works and the names and functions of the various daemons and database files:

Name	Function
biod (see nfsd (NADM))	NFS client I/O daemon
mount (NADM)	mounts a filesystem
mountd (NADM)	NFS mount request server
nfsd (NADM)	NFS server I/O daemon
rpcinfo (NADM)	reports RPC information
showmount (NADM)	shows all clients with remote mounts
<i>filesystem</i> (F)	format of filesystem mount commands table
<i>mnttab</i> (NF)	format of mounted filesystem table
<i>exports</i> (NF)	NFS filesystems being exported

For example, consider a sample mount request as made from an NFS client machine:

```
mount -f NFS krypton:/usr/src /krypton.src
```

The example asks the server machine *krypton* to return a file handle (**fhandle**) for the directory */usr/src*. This **fhandle** is then passed to the kernel in the **mount**(NS) system call. The kernel looks up the directory */krypton.src* and, if everything is correct, ties the **fhandle** to the directory in a mount record. From now on, all filesystem requests to that directory and below go through the **fhandle** to the server *krypton*.

This describes the way the system should work. We now look at the things that can go wrong: first, some general pointers for debugging NFS; then, a list of specific errors and their probable causes.

General hints

When there are network or server problems, programs that access hard-mounted remote files fail in different ways from those that access soft-mounted remote files. Hard-mounted remote filesystems cause programs to retry until the server responds again, while soft-mounted remote filesystems return errors after a certain number of retries.

Once a hard mount succeeds, programs that access hard-mounted files hang as long as the server fails to respond. In this case, NFS displays the following message on the console:

```
server_name not responding
```

On a soft-mounted filesystem, a program receives an error message when it tries to access a file whose server is not responding. You may also wish to specify the **intr** option when hard-mounting filesystems from an unreliable server; this allows you to interrupt the **mount** command and returns an error message from the server.

If a client is having NFS trouble, first check that the server is up and running. From a client, type the following command:

```
rpcinfo -p server_name
```

It should display a list of program, version, protocol, and port numbers similar to the following:

program	vers	proto	port	
100000	2	tcp	111	portmapper
100000	2	udp	111	portmapper
100017	1	tcp	1024	rexed
100005	1	udp	1027	mountd
100003	2	udp	2049	nfs
100024	1	udp	1039	status
100024	1	tcp	1025	status
100021	1	tcp	1026	nlockmgr
100021	1	udp	1051	nlockmgr
100020	1	udp	1054	llockmgr
100020	1	tcp	1028	llockmgr
100021	2	tcp	1032	nlockmgr

You can also use **rpcinfo** to check if the **mountd** server is running:

```
rpcinfo -u server_name mountd
```

This should return:

```
program 100005 version 1 ready and waiting
```

If these steps fail, run the **rpcinfo -p** command on the server's console to verify that the programs listed above are registered.

If the server is functional but a client machine cannot reach it, check the Ethernet connections between the machines.

If the server and the network are functional, use **ps** to check the client daemons. A **nfsclnt** and several **biod** daemons should be running. For example, the command **ps -ef** should display lines for **nfsclnt** and **biod**.

The following sections deal with the most common types of failure. "Remote mount failed" covers the steps to take if a remote mount fails. The sections "Programs are hung" and "Everything works slowly" discuss servers that do not respond when filesystems are mounted.

Remote mount failed

If **mount** fails for any reason, check the sections below for specific details about what to do. They are arranged according to where they occur in the mounting sequence and labeled with the error message likely to be displayed.

The **mount** command can get its parameters either from the command line or from the file */etc/default/filesys* (see the **mount(NADM)** manual page for further information). The following example assumes command-line arguments, but the same debugging techniques would apply if */etc/default/filesys* were the source of the options.

```
mount -f NFS krypton:/usr/src /krypton.src
```

The interaction of the various parts of the mount request should be considered. In the preceding example, **mount** goes through the following steps to mount a remote filesystem:

1. The **mount** command opens */etc/mnttab* and checks that this mount was not already done.
2. The **mount** command parses the first argument into host *krypton* and remote directory */usr/src*.
3. The **mount** command then resolves the host *krypton* into an Internet protocol address.
4. The **mount** command calls *krypton*'s portmapper to get the port number of **mountd**.
5. The **mount** command calls *krypton*'s **mountd** and passes it the */usr/src* pathname.
6. *krypton*'s **mountd** reads its */etc/exports* and looks for the exported filesystem that contains */usr/src*.
7. *krypton*'s **mountd** does a **nfs_getfh(NS)** system call on */usr/src* to get the **fhandle**.
8. *krypton*'s **mountd** returns the **fhandle** to the client.
9. On the client machine, **mount** does a **mount(NS)** system call with the **fhandle** and */krypton.src*.
10. The **mount** command checks whether or not the caller is a super user and */krypton.src* is a directory.
11. The **mount** command does a **statfs(S)** call to *krypton*'s NFS server (**nfstd**).
12. The **mount** command opens */etc/mnttab* and adds an entry.

Any one of these steps can fail, and some of them can fail in more than one way. The following sections give detailed descriptions of the failures associated with specific error messages.

mount: can't open /etc/mnttab

The table of mounted filesystems is kept in the file */etc/mnttab*. This file must exist before **mount** can succeed. The file */etc/mnttab* is created when the system is booted, and it is maintained automatically after that by the **mount** and **umount** commands.

mount: /dev/nfsd is already mounted, *sys_name* is busy,

or allowable number of mount points exceeded

This message reveals an attempt to mount a filesystem that is already mounted. All NFS mount requests that fail with this message display the name */dev/nfsd* (a byproduct of the implementation) regardless of the actual mount request.

mount: *name* or *name*, no such file or directory

The **-f NFS** or **krypton**: part of the sample command was probably omitted. The **mount** command assumes that a local mount is being done unless the **-f** flag is used on the command line, or the requested directory as listed in */etc/default/filesys* specifies filesystem type NFS.

More simply, this message also appears when the specified local mount point is not an existing directory.

mount: can't open /etc/default/filesys

The **mount** command tried to look up the information needed to complete a mount request in */etc/default/filesys*, but there was no such file. As the system administrator, you need to create this file as part of initial system setup.

sys_name not in hosts database

The system name specified on the **mount** command suffixed by the ":" is not listed in the file */etc/hosts*. Check the spelling of the host name and the placement of the colon in the **mount** command.

mount: directory argument *name* must be a full pathname

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

mount: ... server not responding(1):

RPC_PMAP_FAILURE - RPC_TIMED_OUT

Either the server to which the mount is being attempted is down, or its portmapper is dead or hung. Attempt to log in to that machine; if you succeed, then the problem may be in the portmapper. Run the following command from your system as the super user to test the portmapper on the server system:

rpcinfo -p host_name

The result should be a list of registered programs. If this is not the case, kill the remote portmapper and restart it. Restarting the portmapper is a complicated process because all registered services are lost, and their associated daemons must be restarted, too. To find the process IDs of **portmap** and other service daemons, enter this command as the super user:

```
ps -ef
```

Kill the daemons with:

```
kill -9 portmap_pid daemon_id1 daemon_id2
```

Start new ones with commands such as:

```
/etc/portmap
/etc/mountd
/etc/nfsd
```

Another alternative to all this is simply to reboot the server when it is convenient. Because of the stateless nature of the NFS server implementation, there should be no adverse effect on the clients of the system, other than the time that they suspend awaiting the return of the server.

If the server is up but it is not possible to reach it using **ping**, check the client's Ethernet connection by trying to **ping** another machine. Also, check the server's Ethernet connection.

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

```
RPC_PROG_NOT_REGISTERED
```

This means that **mount** got through to the portmapper, but the NFS mount daemon **mountd** was not registered. The server should be checked to ensure that */etc/mountd* exists and is running.

```
mount: /dev/nfsd or directory_name, no such file or directory
```

Either the remote directory does not exist on the server or the local directory does not exist. Again, note that */dev/nfsd* is always printed to represent the remote directory.

```
mount: access denied for sys_name:name
```

The client machine on which the mount attempt is being made is not in the server's export list for the filesystem to be mounted. A list of the server's exported filesystems can be obtained by running:

```
showmount -e host_name
```

If the desired filesystem is not in the list, or the machine name is not in the user list for the filesystem, then check the */etc/exports* file on the server for the correct filesystem entry. A filesystem name that appears in the */etc/exports* file but not in the output from **showmount** indicates a

failure in **mountd**. Perhaps it could not read that line in the file, or it could not find the filesystem, or else the filesystem name was not a locally mounted filesystem. See *exports(NF)* for more information.

This message can also be an indication that authentication failed on the server. It may be displayed because the machine that is attempting the mount is not in the server's export list; check the server's */etc/exports* file.

mount: **name**: no such file or directory
The remote path on the server is not a directory.

mount: not super user
The **mount** command can be used only by the super user, because it affects the filesystem for the entire machine.

Programs are hung

If programs hang doing file-related work, the NFS server may have crashed. The following message may appear on the machine's console:

NFS server **sys_name** not responding, still trying

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

If a client machine completely stops responding, check the servers from which the filesystems were mounted. If one of them is down, client machines may wait indefinitely. When the server comes back up, programs continue automatically and are unaffected.

If a soft-mounted server dies, other work should not be affected. Programs that time-out trying to access soft-mounted remote files will fail, but it should still be possible to get work done on other filesystems.

Everything works slowly

If access to remote files seems unusually slow, the server should be checked by entering (on the server):

ps -ef | more

If the server is functioning and other users are getting good response, use the **ps -ef** command on the client to see if block I/O (**biod**) daemons are running. If the daemons are not running or are hung, they should be killed. First, find the process IDs by entering:

ps -ef | grep biod

Then, kill them with:

kill -9 pid1 pid2 pid3 pid4

Restart the daemons with:

```
/etc/biod 4
```

To determine whether the daemons are hung, use **ps** as above, then copy a large file. Another **ps** shows whether the **biods** are accumulating CPU time; if not, they are probably hung.

If **biod** appears to be functioning correctly, check the Ethernet connection. Use the **nfsstat -c** and **nfsstat -s** commands to discover whether a client is doing a lot of retransmitting. A retransmission rate of 5% is considered high. Excessive retransmission usually indicates a bad Ethernet board, a bad Ethernet tap, a mismatch between board and tap, or a mismatch between the client machine's Ethernet board and the server's board.

Serial number validation errors

When you see a message of the following nature, a validation error has occurred:

```
Copy Protection Violation!
Duplicate Serial Number(s) detected on 123.4.567.007
Product(s) att00001 shutting down.
```

This means that there are two copies of the same program running at the same time. Contact your SCO representative to obtain a valid activation key for the second copy.

Incompatibilities with remote filesystems

A few things work in unexpected ways, or not at all, on remote NFS filesystems. The next section discusses the incompatibilities and offers suggestions for working around them.

File operations not supported

File locking operations on directories is not supported on remote filesystems.

Append mode and atomic writes are not guaranteed to work on remote files accessed by more than one client simultaneously.

Applications that use mandatory locking are not supported over NFS, nor are applications that use **read()**. An application that uses the **read()** function returns an error message similar to the following:

```
25 Apr 1990 rfs_read: attempt to read from non-file
```

Cannot access remote devices

With NFS, it is not possible to access a remote-mounted device or any other character, block-special file, or named pipes.

Cannot access different filesystem types

Under NFS, it is not possible to access a filesystem of a type different from that of your native system. For example, if you are running NFS under the UNIX system, it is not possible to access a DOS filesystem from your UNIX client machine. SCO NFS only supports Acer, S51K, and UNIX filesystems.

Cannot access indirect filesystems

Under NFS, it is not possible to access a filesystem indirectly. For example, you may not use a server machine to access a filesystem on a third server machine. All NFS access must be direct from server to client.

Handling clock skew in user programs

Because the NFS architecture differs in some minor ways from earlier versions of the XENIX and UNIX systems, users should be aware of those places where their own programs could run up against these incompatibilities.

The clocks on the NFS server and client may be out of sync because each machine keeps its own time. This might cause problems. Consider the following example.

Many programs assume that an existing file could not be created in the future. For example, the command `ls -l` has two basic forms of output, depending upon how old the file is:

```
$ date
Sat Apr 12 15:27:48 GMT 1986
$ touch file2
$ ls -l file*
-rw-r--r-- 1 jsbach      0 Dec 27  1984 file
-rw-r--r-- 1 jsbach      0 Apr 12 15:27 file2
```

The first type of output from `ls` prints the year, month, and day of last file modification if the file is more than six months old. The second form prints the month, day, hour, and minute of last file modification if the file is less than six months old.

The `ls` command calculates the age of a file simply by subtracting the modification time of the file from the current time. If the result is greater than six months, the file is "old".

Assume that the time on the server is Apr 12 15:30:31, which is three minutes ahead of the local machine's time:

```
$ date
Apr 12 15:27:31 GMT 1986
$ touch file3
$ ls -l file*
-rw-r--r-- 1 jsbach      0 Dec 27  1983 file
-rw-r--r-- 1 jsbach      0 Apr 12 15:26 file2
-rw-r--r-- 1 jsbach      0 Apr 12  1986 file3
```

The difference between the current time and the library's modify time makes the `ls` command think that the new file was created long ago.

In general, users should remember that applications that depend upon local time and/or the filesystem timestamps have to deal with clock skew problems if remote files are used.

Chapter 3

NIS: Network Information Service

Network Information Service (NIS) is a networked system administration service that supplements NFS (Network File System). This chapter presents background information on NIS, suggests guidelines for planning an NIS configuration, and explains configuration and administration procedures. Novice and experienced network administrators alike should read the entire chapter; although you may be familiar with NIS functionality, the section “Understanding NIS” includes details on features unique to SCO’s implementation.

The term “Network Information Service (NIS)” now refers to the service formerly known as Sun “Yellow Pages” or “YP.” The functionality remains the same; only the name has changed. The name “Yellow Pages” is a registered trademark in the United Kingdom of British Telecommunications PLC and may not be used without permission from British Telecom. Nonetheless, many NIS utilities retain the “yp-” prefix, for example, **ypmake** and **ypserv**.

Understanding NIS

The primary purpose of NIS is to ensure the consistency of system administration information in a heterogeneous network environment. It does so by designating a single host as the *master* of all the files and databases containing this information and providing a method to distribute the information to all other hosts on the network.

NIS distributes information from the following files on the master server by default:

- */etc/passwd*
- */etc/ethers*
- */etc/networks*
- */etc/services*
- */etc/hosts*
- */etc/group*
- */etc/protocols*

/usr/lib/mail/aliases can also be managed by NIS if your system uses **sendmail**. In addition, other frequently used files (such as telephone lists and Auto-mounter maps) may be selected for NIS administration. When NIS is initialized on the master server, information from these files is compiled into a series of databases called *maps*. The maps are then propagated to other servers (*slave* and *copy-only* servers) on the network, where they are translated back into ASCII files. When it is necessary to update information in these files, the ASCII files are edited on the master server, and the maps are modified and repropagated using NIS commands.

In the SCO implementation of NIS, all look-ups of NIS-managed information are performed locally on servers. When a new map is received, it is converted to the corresponding ASCII file(s) that are then available locally to requesting processes. Hence, there are no clients in the SCO NIS implementation; that is, requests are not satisfied remotely, across the network. However, SCO NIS servers can satisfy look-up requests from non-SCO clients.

The following sections describe the basic concepts and functionality of NIS.

NIS maps

NIS serves information stored in NIS *maps*, each having its own name. Maps are a series of databases containing a set of keys and associated values. For example, in a map called *hosts*, all of the host names within a network are the keys, and the Internet addresses of these host names are the values. Programs may use the *mapname* to access the map and must know the format of the data in the map.

Many of the current maps are derived from ASCII files traditionally found in */etc*, such as *hosts*, *group*, *passwd*, and others. In most cases, the data within the NIS map is identical to the format within the ASCII file. Maps are implemented as files located in the subdirectories of the directory */etc/yp* on NIS server machines.

Each map consists of a pair of **dbm**(S) database files, one containing a directory (that is, a bitmap of indices accessed by a hash table) and the other data. **dbm** files are created when NIS is initialized or as needed using NIS commands. Because of the non-ASCII file structure imposed by **dbm**, you must use NIS tools (such as **ypxfr** and **yppush**) to transfer maps from one machine to another.

NIS domain

An NIS *domain* consists of a group of computers sharing configuration information on a network, although NIS domains differ from Internet, MMDF, and **sendmail** domains. Each NIS domain has a corresponding set of maps that are contained in a subdirectory of */etc/yp*; the name of the subdirectory is the same as the domain. For example, maps for the literature domain are in */etc/yp/literature*.

To retrieve data from an NIS database, you must set an NIS domain name with the **domainname(NC)** command. It is easier to manage your network if you use a single domain name on all machines on a network.

NIS servers

Three kinds of machines have NIS databases in an SCO system: a *master* server, a *slave* server, and a *copy-only* server. The master server updates the databases of the other two servers. Both slave and copy-only servers handle maps, setup, and administration in a similar way. The only difference is that a slave server may satisfy requests to transfer maps if the master server becomes unavailable. In a heterogeneous environment, a slave server may also accept NIS requests from any non-SCO client, but the copy-only server does not accept NIS requests.

NOTE The copy-only server is not a client in the conventional sense of the term. That is, an NIS request entered from a copy-only server machine is not fulfilled over the network but rather on the local host. SCO NFS does not provide NIS client functionality, but non-SCO NIS clients may make requests of SCO NIS master and slave servers.

Database changes in an NIS domain are propagated from the master server to other servers. If you create or change NIS databases on slave server or copy-only server machines instead of master server machines, all modifications you make are only temporary and are overwritten when the master server updates them. Therefore, make all database creations and modifications on the master server machine only.

A server can be a master with regard to one NIS map and a slave or copy-only with regard to another. To avoid the confusion caused by randomly assigning maps to server and copy-only machines, we recommend that you make a single server the master for all the maps created by **ypinit(NADM)** within a single domain. This document assumes that one server is the master for all maps in the database.

NIS daemons

NIS services are implemented by the following daemons:

- | | |
|-----------------|---|
| ypserv | satisfies requests for NIS services. Requests are usually to transfer a map, display its contents, or to display information about the NIS domain. |
| ypbind | listens for NIS requests and directs them to a ypserv process that can satisfy the request |
| ypxfrd | satisfies requests from slave or copy-only servers to transfer maps. In SCO NIS, maps are transferred only when the identity of the master has been verified by the slave or copy-only server and a ypxfr command is issued. |
| yppasswd | satisfies requests to change passwords of NIS-managed accounts and communicates with the master server to update password information. This service is not available to servers in Secure Mode (see the “Security limitations on configuration” section of this chapter for more information). yppasswd must only be run on the master server. |

The action of associating a server with a particular host is called *binding* that host in the NIS domain. Hosts must be *bound* in order for NIS services to be provided. Binding is accomplished at system start-up when the **ypbind** daemon broadcasts for a server to satisfy a process request. Unless the host system crashes or the network administrator intervenes, bindings remain constant in an NIS domain.

A heterogeneous environment differs from an SCO environment with regard to binding:

- SCO environment
 - all hosts are servers that contain both NIS maps and ASCII files derived from them
 - look-ups are resolved locally
 - all hosts are bound to themselves by default. That is, the local **ypbind** binds to the local **ypserv**, which in turn satisfies all NIS requests originating on that machine.
- heterogeneous environment
 - client hosts possess neither NIS maps nor ASCII files derived from them
 - look-ups are resolved across the network through NIS requests
 - the client’s **ypbind** broadcasts a request for service and binds to the first responding **ypserv** process

NIS utilities

A number of utilities are available to perform the following types of activities necessary for NIS administration:

- examine and manipulate NIS maps
- propagate NIS maps
- manage user accounts
- display information about the NIS domain and the servers it contains

A host must be bound in an NIS domain (that is, **ypbind** must be running) before an NIS command can be executed from that host.

NIS user accounts

Two types of accounts exist on an NIS network: local accounts and distributed accounts. Local accounts allow users access to the local host machine only, while distributed accounts allow users to access any machine on the NIS network. Each distributed account contains the same name and UID throughout the NIS network.

NIS password information is maintained by the following files in the */etc* directory:

<i>passwd.yyp</i>	contains entries for distributed accounts. After entries are added using the yppassmgt command, the <i>passwd</i> map is created from this file. Each NIS domain has a unique <i>passwd.yyp</i> file existing only on the master server.
<i>passwd.local</i>	contains entries for local accounts. Entries are created by default when NIS is initialized on a system or added later using the passmgt command.
<i>passwd</i>	contains entries for all active accounts, local and distributed. On the master server, <i>passwd</i> contains all entries from <i>passwd.yyp</i> and <i>passwd.local</i> . On slave and copy-only servers, it contains entries from <i>passwd.local</i> and ASCII translations of the <i>passwd</i> map received from the master.

Entries in *passwd.local* take precedence over entries in *passwd.yyp*. That is, if the two files include a user with the same name, the information in *passwd.local* is incorporated into *passwd* and the corresponding information in *passwd.yyp* (and the *passwd* map) is overridden.

If you want to create local user accounts on an NIS server, do so before you configure NIS. Create these accounts using **sysadmsh**(ADM). Refer to the *SCO UNIX Operating System System Administrator's Guide* for more information on **sysadmsh**. For information regarding account creation after initializing NIS, refer to the section "Managing user accounts" later in this chapter.

Understanding YP_MAP_X_LATE

The `/etc/yp/YP_MAP_X_LATE` file contains a complete listing of active NIS maps. The most common entries are *hosts*, *group*, *passwd*, and *ypservers*, although all standard and non-standard maps (that is, created by you) must be listed here if you wish NIS to serve them.

ypinit(NADM) consults `YP_MAP_X_LATE` when NIS is initialized and creates the maps listed therein. All desired maps must therefore be listed in `YP_MAP_X_LATE` before your system is initialized. If you add new maps after initialization, they need not be listed in `YP_MAP_X_LATE` to be made functional; however, because NIS consults `YP_MAP_X_LATE` when propagating maps, new maps must be listed here for them to be propagated using **yppush**(NADM), **ypxfr**(NADM), or other NIS utilities. For this reason, we recommend that you keep `YP_MAP_X_LATE` up-to-date on your master server.

Although slave and copy-only servers do not propagate maps and hence do not consult their `YP_MAP_X_LATE` files, we recommend that you keep these files current on non-master servers to simplify NIS administration.

The `YP_MAP_X_LATE` file also contains NIS aliases for NIS maps; aliases may be used by all NIS servers. If you have a system on your network that cannot handle long map names (that is, longer than 14 characters), you can add an alias to `YP_MAP_X_LATE`, showing a shortened alias for the map. If you have additional aliases to include in `YP_MAP_X_LATE`, add them before configuring NIS. The format of this file is described in the `ypmapxlate`(NF) manual page.

Security limitations on configuration

To preserve the integrity of trusted databases, NIS may not modify certain sensitive files within a C2 Secure system. Although SCO NIS operates on a system with any of the security defaults (High, Improved, Traditional, and Low), the password and group databases are updated by NIS only in the security defaults that use `/etc/passwd` and `/etc/group` as the master databases. This condition is true if Traditional or Low security (Unsecure Mode) is chosen at installation time. If High or Improved security (Secure Mode) is chosen, `/tcb` manages `/etc/passwd` and `/etc/group` and does not allow NIS to update these files.

Only copy-only NIS servers may be initialized in Secure Mode settings on SCO systems. These servers may receive maps, but they may not propagate them. In addition, if a copy-only server in Secure Mode receives a password or group map, it cannot translate these maps into `/etc/passwd` or `/etc/group` ASCII files. Table 3-1 lists the security configurations permitted for each type of SCO NIS server.

Table 3-1 Security configurations for SCO NIS servers

SCO NIS server type	Permitted security settings
master	Unsecure Mode only (Traditional or Low)
slave	Unsecure Mode only (Traditional or Low)
copy only	Secure Mode (High or Improved) or Unsecure Mode (Traditional or Low)

These security precautions help prevent unauthorized NIS password and group information from reaching SCO hosts. However, the same security precautions may not be available for non-SCO client implementations. Consult your documentation for non-SCO NIS client implementations if you are concerned with security.

CAUTION The installation of the NIS Runtime System compromises C2 security and reduces your ability to maintain a secure system. With NIS installed, it is relatively easy to change the master machine and change all the passwords on all systems in the network. Also, passwords (in encrypted form) are stored in text files and can be read by anyone with the correct privileges. If you are concerned with security, NIS may not be an appropriate service for your system.

Planning an NIS configuration

When planning an NIS configuration, you must gauge your network administration needs with regard to domain structure, server disposition, security concerns, and the physical structure of the network. Your configuration will also be affected if the network is heterogeneous, that is, if it includes non-SCO hosts. The following sections explain these considerations in more detail.

Selecting domains

For most NIS installations, a single domain is sufficient to satisfy users' needs. However, if your site has multiple networks, you may wish to create separate domains for each network. A possible network division might be into accounting, marketing, and manufacturing domains.

In such an environment, it is recommended that there be one master and at least one slave server for each domain.

Selecting servers

Two factors are important in selecting master and slave servers: uptime and synchronization. The master server should be a host that is stable and available for NIS requests. This suggests that a machine be dedicated as a master server and that there is at least one slave server for each master in a domain should the master become unavailable.

The master and slave servers should have CPU's comparable to those of the machines to which they will propagate maps. This is particularly important in heterogeneous environments where a number of non-SCO NIS clients may seek to bind with the most available server. Because it is essential that current map information be uniformly available across the network, availability and equivalent speed of servers ensure that the propagation of map information is synchronized.

In an SCO NIS environment, slave servers can be used as a backup to the master server and to improve the ability to service map update requests from copy-only servers. In a heterogeneous environment, slave servers should be added as needed to keep up with requests from non-SCO clients.

The physical structure of the network may influence the distribution of slave servers. For example, if you have a network that includes a bridge, you will probably want at least two slave servers on either side. This is because NIS requests are delayed when crossing bridges, such that a server on the same side of a bridge as the requesting host will satisfy a request sooner than one on the far side, even if the closer server is heavily loaded.

Security

Although NIS is designed to administer password and group authorization files, such administration is incompatible with C2 security. SCO NIS maintains C2 security in an NIS environment by enforcing the following restrictions:

- only copy-only servers may be initialized on Secure Mode machines
- copy-only servers in Secure Mode may not use password and group maps for authentication. This means that master and slave servers are not afforded C2 protection.

In a heterogeneous environment, SCO copy-only servers under C2 protection continue to refuse password and group maps from non-SCO master or slave servers. However, an SCO master or slave server (by definition in Unsecure Mode) may be able to propagate maps to C2-protected non-SCO clients. Consult your non-SCO client's documentation to address your security concerns. Refer also to the "Security under NIS" section later in this chapter.

Configuring NIS

SCO NIS is enabled by setting up NIS servers, initializing NIS and starting the NIS daemons. The following sections explain these operations.

NOTE Please read this entire section before configuring NIS on your system.

Setting up an NIS server

The following is an overview of the steps involved in setting up NIS servers:

1. Your network must be configured and operating properly before initializing NIS. Make sure the fully-qualified host name is set by running the **hostname** command. You should see the hostname and the default network domain name, for example, `jupiter.sco.com`. Usually, the network domain name and host name are set up by the system startup script.

CAUTION If your hostname and network domain name are not correct, you may have configuration problems with your network. However, changing the hostname and the default network name at this point may also force you to reconfigure other network services, such as **sendmail**, **MMDF** and **SCO LAN Manager**. Consult your TCP/IP documentation for more information on network configuration.

2. Create local user accounts. If you do not create local user accounts before you run **ypinit** or **mkdev nis**, you must run **passmgmt** after initialization to make distributed accounts local.
3. Edit **YP_MAP_X_LATE** to remove names of files you do not wish NIS to service and add those you do. You should also modify NIS aliases for NIS maps as required. You must use NIS aliases if a map name exceeds the limit allowed by your operating system (usually 14 characters).
4. Ensure that the NIS domain name is set to **unknown** by entering **domainname** at your prompt. If the screen returns anything other than **unknown**, enter **domainname unknown**. **ypinit** and **mkdev nis** do not enter the appropriate command in the NFS startup script unless your domain name is set to **unknown**. **domainname(NC)** sets or displays the name of the current NIS domain.
5. Make sure the following files in **/etc** are complete and current on your master server: *passwd*, *hosts*, *ethers*, *group*, *networks*, *protocols*, and *services* (as well as */usr/lib/mail/aliases* if you use **sendmail**). All files listed in **YP_MAP_X_LATE** should be available. If any of these files were removed from **YP_MAP_X_LATE**, do not be concerned.

6. Add the following entry to the **PATH** variable of your */.profile* file (if you are running the Bourne or Korn shell) or */.cshrc* file (if you are running the C shell):

```
/etc/yp
```

You are now ready to initialize your NIS server.

Initializing NIS

The following steps explain how to initialize an NIS server. The procedure varies according to the type of server.

1. Log in as *root* and enter the following command to shut down the system:

```
/etc/shutdown -g0 -y
```

You see messages as the system services stop. Press **<Return>** at the **Safe to Power Off** message.

2. Type **y** and press **<Return>** when prompted to reset the location of the console. The following prompt appears:

```
Type <Ctrl>D to proceed with normal startup,  
(or give root password for system maintenance)
```

3. Type the *root* password, then press **<Return>**.
4. Enter either of the following commands at your operating system prompt:

```
mkdev nis
```

```
or
```

```
/etc/yp/ypinit
```

The functionality and the options for **ypinit** and **mkdev nis** are identical.

5. NIS prompts you to choose the host type:

```
You must specify whether this host is to be a master, server,  
or copy-only server. Please enter the server type:[mscq]
```

- Enter **m** to indicate that the local host is to be the NIS master server.
- Enter **s** to indicate that you are setting up a slave server.
- Enter **c** to indicate that you are setting up a copy-only server.
- Enter **q** to discontinue initialization.

If NIS has been previously initialized on your system, you see the following:

```
NIS has been initialized previously on this machine
Please verify that you would like to re-initialize NIS
and then remove the file /etc/yp/ypinit_done
```

Remove the file `/etc/yp/ypinit_done`, reenter `mkdev nis` or `/etc/yp/ypinit`, and repeat this step.

If you attempt to configure a master or slave server on a host in Secure Mode (that is, at High or Improved security defaults), you see the following message:

```
ypinit: ERROR: Invalid configuration.
```

```
The security settings of this system are incompatible with the
NIS configuration requested. Please refer to your Release and
Installation Notes for further details. Initialization Aborted.
```

Refer to the `ypinit(NADM)` manual page for caveats concerning security restrictions.

6. If you are initializing a master server, proceed to step 7.

If you are initializing a copy-only server in Secure Mode, you see:

```
Due to the restrictions of running NIS under the current security
setting of this system, this server will accept, but not integrate
sensitive maps received through NIS. Please refer to your Release
and Installation Notes for further details.
```

```
Hit return to continue
```

If you are initializing a slave or copy-only server, NIS prompts you to enter a host name. You see:

```
You must specify a host name to act as the NIS master
during map transfer [q to quit]:
```

Enter the name of a master server with an up-to-date and stable database.

7. You then see:

```
The local host's domain name hasn't been set. Please set it.
Please enter the domain name or q to quit:
```

Enter the NIS domain name; you need an NIS domain name to retrieve data from an NIS database.

You see this prompt:

```
Installing the NIS database will require that you answer a few
questions. Questions will all be asked at the beginning of
the procedure.
```

8. If this is the first installation and initialization of NIS or if you have previously removed NIS, proceed to step 9.

If an NIS database already exists on your machine, you see:

Can we destroy the existing domainname and its contents? [y/n/q: n]

Enter **y** to continue, which tells NIS to rebuild the previous NIS database with current information.

If you enter the default **n** or **q**, NIS initialization aborts, and you must manually remove the old NIS databases.

9. If you are initializing a copy-only server in Secure Mode, proceed to step 11.

NIS maintains local host accounts in */etc/passwd.local* and */etc/group.local*. For slave servers and copy-only servers, the host retrieves current NIS maps from the master server and merges the NIS maps with the local account information kept in */etc/passwd.local* and */etc/group.local*. The merged account information is stored in */etc/passwd* and */etc/group*. The network account information is then accessible on the local host. However, local account information overrides common entries, so NIS maps do not affect local maps.

If the */etc/passwd.local* file does not exist on the machine, you see the following message:

NIS stores local logins in */etc/passwd.local*, which currently doesn't exist on this machine. The */etc/passwd.local* file must be created before continuing.

Is it okay to copy */etc/passwd* to */etc/passwd.local*? [y/n/q: n]

Enter **y** to continue.

If you enter the default **n**, you must manually create the */etc/passwd.local* file, then start the initialization procedure again.

10. If the */etc/group.local* file does not exist on your machine, you see:

NIS stores local groups in */etc/group.local*, which currently doesn't exist on this machine. The */etc/group.local* file must be created before continuing.

Is it okay to copy */etc/group* to */etc/group.local*? [y/n/q: n]

Enter **y** to continue.

If you enter the default **n**, you must manually create the */etc/group.local* file, then start the initialization procedure again.

11. Next, verify that the `YP_MAP_X_LATE` file contains correct information.

You see:

The `YP_MAP_X_LATE` file contains the following:

map_names

Is this correct? [y/n/q: n]

If the list is not correct, enter the default **n**. Edit `YP_MAP_X_LATE` to contain the correct entries, and start the initialization procedure again. Refer to the `ypmapxlate(NF)` manual page for more information.

If the list is correct, enter **y** to continue.

12. If you are initializing a slave or copy-only server, proceed to step 14.

You see the following prompt:

The NIS network password file resides in `/etc/passwd.yp` on master servers only. Since this node is being set up as a master server, `/etc/passwd.yp` must be created.

Is it okay to create `/etc/passwd.yp`? [y/n/q: n]

Enter **y** to continue.

If you enter **n**, you must manually create the `/etc/passwd.yp` file, then start the initialization procedure again.

13. You see the following prompt:

The NIS network password file resides in `/etc/group.yp` on master servers only. Since this node is being set up as a master server, `/etc/group.yp` must be created.

Is it okay to copy `/etc/group` to `/etc/group.yp`? [y/n/q: n]

Enter **y** to continue.

If you enter **n**, you must manually create the `/etc/group.yp` file, then start the initialization procedure again.

14. You then see:

You must specify whether ypbind is to be run with the secure option (-s), the ypset option (-ypset), allowing anyone to change your binding, or the ypsetme option (-ypsetme) which only allows requests with *host-name's* IP address to change *host-name's* binding, or with no options.

Please see ypserv(NADM) for more information

- 1) ypbind -s
- 2) ypbind -ypset
- 3) ypbind -ypsetme
- 4) ypbind

Please enter your choice (1-4,q):

Enter the response appropriate to your security concerns. In an SCO NIS environment, the **-ypsetme** option (3) is recommended. In a heterogeneous environment, the **-s** option (1) is appropriate in most cases. For further information, consult the ypserv(NADM) manual page, which includes information about ypbind.

15. If you are initializing a master server, proceed to step 16.

When the initialization procedure is complete, you see a series of messages resembling the following display when initializing a copy-only server:

There will be no further questions. The remainder of the procedure should take a few minutes to set up the NIS server.

The NIS maps will be transferred from *master_server* when *server_name* goes to run level 2.

If NIS has been previously initialized on your system, you see:

The backup directory /usr/lib/nisrt already exists

Do you wish to continue with the SCO NIS Runtime System Initialization?

Enter **y** to continue; NIS then backs up system *passwd* files. If you enter **n** or **q**, NIS installation aborts.

You then see (for both slave and copy-only servers):

Backing up system passwd files ...

server_name's NIS data base has been set up without any errors.

Your slave or copy-only server is now initialized. Proceed to step 18.

16. You then see:

At this point, we have to construct a list of the hosts which will be running NIS (NIS server hosts). The local host is in this list of NIS server hosts. Please continue to add the names for the other hosts, one per line. When you are done with the list, type a `(Ctrl)D`.

Enter all of the server names — master, slave, and copy-only — including all non-SCO server machines. The machine name and the Internet Protocol (IP) domain name are acceptable host names; the NIS domain name is not. Press `(Return)` after each host name. After entering the last host name, press `(Ctrl)D` to continue, and enter `y` to verify the host names are correct. NIS then creates and updates the maps.

When you initialize NIS, all of the UNIX system files that have the same names as NIS files are backed up in parallel subdirectories under `/usr/lib/nisrt`. When you initialize the NIS server, NIS changes some system files. For example, NIS saves the system file `/bin/passwd` to `/usr/lib/nisrt/bin/passwd` and then replaces `/bin/passwd` with a link to the file `/etc/yppasswd`.

When the initialization procedure is complete, you see a series of messages resembling the following display:

There will be no further questions.

The remainder of the procedure should take 5 to 10 minutes.

If NIS has been previously initialized on your system, you see:

The backup directory `/usr/lib/nisrt` already exists

Do you wish to continue with the SCO NIS Runtime System
Initialization?

Enter `y` to continue; NIS then backs up system *passwd* files. If you enter `n` or `q`, NIS installation aborts.

You then see:

```
Backing up system passwd files ...
```

```
Building /etc/yp/domain_name/ypservers...
```

```
Running /etc/yp/ypmake...
```

```
updated passwd
updated group
updated ethers
updated hosts
updated rpc
updated services
updated protocols
updated netgroup
updated aliases
updated ypbind
updated ypserv
updated yppasswd
```

host_name has been set up as an NIS master server without any errors.

This display of updated maps should correspond to the maps listed in */etc/yp/YP_MAP_X_LATE*.

17. The **ypinit** script then checks your tunable kernel parameters to see if adequate Streams buffers have been allocated for NIS. If not, it updates the Streams parameter and displays the following message:

```
Streams buffer sizes have been updated
```

```
Do you wish to relink the kernel now? (y/n/q)
```

Answer **y**. If you do not relink, the network may be slow or erratic because of insufficient resources. It takes several minutes to relink the kernel.

Your master server is now initialized.

18. Enter the following command to re-boot the system:

```
/etc/shutdown -g0 -y
```

Enter **<Ctrl>D** when prompted to return to multiuser mode.

The first time an NIS slave or copy-only server enters multiuser mode, its startup script calls **ypxfr** to transfer maps from the master server. Subsequent system startups do not transfer maps automatically.

NOTE If you wish to add distributed accounts, you must run **yppassmgt** on the master server. Refer to the section “Creating a distributed account” of this chapter for more information.

Starting NIS

After you complete the installation and initialization procedures, you must activate NIS.

When you initialize NIS with **ypinit**, a copy of */etc/nfs* is saved to */usr/lib/nisrt/etc/nfs* and the */etc/nfs* script is modified to enable automatic startup and shutdown of NIS daemons. By default, installation of NFS places the startup script in */etc/rc2.d/S89nfs* and the shutdown script in */etc/rc0.d/K79nfs*. Both files are linked to */etc/nfs*, such that NIS daemons are started and stopped whenever NFS is started or stopped.

After you have configured your machine as an NIS server and start NFS, it displays a series of messages similar to the following:

```
NFS Startup...
ONC daemons: portmap pcnfsd rwalld rusersd
NFS daemons: exportfs mountd nfsd(4) nfscld(4) biod(4) automount
Lock Manager: statd lckclnt(4) lockd
NIS services: ypserv ypbind yppasswd ypxfrd
NFS Startup complete.
The system is ready.
```

Other messages may also be displayed if you have configured your machine as a slave or copy-only server. For example, the following message indicates that the *protocols.byname* map was successfully converted from **dbm** to ASCII format:

```
Transferring proto.nam ...
map2ascii conversion for proto.nam map succeeded
```

Some map conversions, specifically for the *passwd* and *group* maps, will fail in Secure Mode.

If the startup of NFS does not display an NIS service message, either an error occurred during the initialization of NIS or **ypinit** has not modified */etc/nfs* to enable the automatic starting of NIS daemons. Go back to single-user mode, determine the source of the problem, and run **ypinit** again.

Managing NIS maps

The following sections explain how to modify existing NIS maps, propagate maps in an NIS domain, and create new NIS maps.

Map manipulation utilities

SCO provides the following utilities to administer NIS maps:

- ypmake** modifies the default maps listed in the `YP_MAP_X_LATE` file. **ypmake** also runs **yppush** by default to propagate newly updated maps.
- makedbm** modifies only the *ybservers* map and any non-default maps created by the administrator. It can also be used to disassemble maps for examination in ASCII form.
- ypinit** manipulates maps when NIS is being initialized but should not be used when NIS is already running

For more information, refer to the **ypmake(NADM)**, **makedbm(NADM)**, and **ypinit(NADM)** manual pages.

The SCO implementation of NIS provides **ypmake**, which searches for **make(CP)**. **make** is conventionally used to maintain groups of programs and is used by NIS to build and propagate maps. However, **make** is part of the SCO Development System and may not be installed on your system. If **make** is not available on your system, use **ypmake**, and modify the `/etc/yp/ypmake` file as you are instructed throughout this document.

If **make** is available on your system, you can issue either the **make** or **ypmake** command; however, **ypmake** finds and invokes **make** if available. **make** also requires a separate list of dependencies called the *Makefile*. Because this list is not included in **make** as it is in **ypmake**, you must edit the `/etc/yp/Makefile` file every time you are instructed to edit `/etc/yp/ypmake`.

NOTE We recommend that you use **ypmake** for NIS use even if **make** is already on your system. Because **ypmake** invokes **make**, you should keep the *ypmake* or *Makefile* files concurrent, particularly if both commands are available on your system.

Modifying existing maps

To modify existing maps, you must change the databases that are served by NIS on the master server. Update the databases by first editing the ASCII file from which they are derived, then running **ypmake**. For example, you can modify the *hosts* map by following these steps:

1. Log into the master server.
2. Edit the */etc/hosts* file.
3. Issue the following commands to rebuild and propagate the map:

```
cd /etc/yp
./ypmake hosts
```

Use **ypmake** without arguments to modify and propagate all maps automatically.

WARNING Run **ypmake** or **make** only on master servers; unpredictable behavior in the NIS map set may result if they are run on slave or copy-only servers.

However, certain maps cannot be modified using **ypmake**. These are maps for which no ASCII file exists, such as the *ypservers* map. To modify these maps, use the **makedbm** command with the **-u** switch to unassemble them into a form that can be modified using standard tools (such as **awk**, **sed**, or **vi**). Build a new version using **makedbm** manually by either:

- redirecting the output of **makedbm** to a temporary file, modifying the file, and then using the file as input to **makedbm**. The following example could be used to add an entry to *ypservers*:

```
cd /etc/yp
makedbm -u ypservers > /tmp/ypservers
[Add an entry in /tmp/ypservers]
cat /tmp/ypservers | makedbm - /etc/yp/domain_name/ypservers
```

- piping the output of **makedbm** directly into **makedbm**. This is appropriate if the disassembled map can be updated by modifying it with, for example, **awk**, **sed**, or a **cat** append. The following example could be used to remove an entry from *ypservers*:

```
cd /etc/yp
makedbm -u ypservers | grep -v server_name \
| makedbm - /etc/yp/domain_name/ypservers
```

You must run the **yppush** command after modifying a map using **makedbm**; refer to the section “Adding servers” of this chapter for more information.

Propagating an NIS map

Propagating a map means moving it from place to place, generally from the NIS master server to an NIS slave or copy-only server. Initially, the map is moved at system startup. After you initialize an NIS nonmaster server, **ypxfr**(NADM) transfers updated maps from the master server. The **ypxfr** command can be run in the following ways:

- periodically by **cron**(C)
- by **ypserv**(NADM)
- interactively by a user

In all cases, **ypxfr** checks for a file in */etc/yp/map2ascii* with the same name as the transferred map. If this file exists, it is executed as the last step of the **ypxfr** process. These *map2ascii* files translate the new map into its ASCII counterpart and install it in its proper place. **map2ascii** routines are supplied for the most commonly used maps. You can write other routines and place them in the directory for use on other maps.

ypxfr's transfer attempts and results can be captured in a log file. If */usr/adm/nislog* exists, results are appended to it. The user is responsible for limiting the log file. To turn off logging, remove the log file. Refer to the **ypxfr**(NADM) manual page for more information about the log file.

Methods for transferring updated maps are described in the following sections.

Using cron

Maps have differing rates of change; for instance, *protocols.byname* might not change for a long time, but *passwd.byname* might change several times a day in a large organization. You can set up **root**'s **crontab**(C) entries to run **ypxfr** periodically at a rate appropriate for any map in an NIS database. The **ypxfr** program contacts the master server and transfers the map only if the master's copy is more recent than the local copy.

To avoid a **crontab** entry for each map, group several maps into a shell script that can be run from a single **crontab** script file. Each shell script should contain maps with approximately the same characteristics. The following suggested groupings, mnemonically named, can be found in */etc/yp*: *ypxfr1phr*

(hourly), *ypxfr1pdy* (once daily), and *ypxfr2pd* (twice daily). If the rates of change are inappropriate for a particular environment, you can easily modify or replace these shell scripts. The following **crontab** entries will call the scripts:

```
0 * * * * * /etc/yp/ypxfr1pdy
0 0 * * * * /etc/yp/ypxfr1pdy
0 0,12 * * * * /etc/yp/ypxfr2pdy
```

Run the same shell scripts at each NIS slave and copy-only server in the domain; do not run them from the master server. Alter the time of execution from one server to another to prevent the checking from impeding the master server. To transfer the map from a particular server, rather than the master server, use the **-h** option to **ypxfr** from within the shell script. You can check and transfer maps having unique change characteristics by explicit invocations of **ypxfr** within **crontab**.

Using the interactive approach

To use the interactive approach, run **ypxfr** as a command. Typically, this is done only in exceptional situations: for example, when setting up a temporary NIS server to create a test environment, when transferring a new NIS map, or when quickly updating an NIS server that was out of service.

Using yppush on the master server

The **ypxfr** command is also invoked by **ypserv** when the **yppush(NADM)** command is run on the NIS master server. **yppush** enumerates the NIS map **ypservers** to generate a list of NIS servers in the domain and sends a “Transfer Map” request to each named NIS server. The **ypserv** command spawns a copy of **ypxfr**, invoking it with the **-C** flag, and passes it the information needed to identify the map and call back the initiating **yppush** process with a summary status.

NOTE **yppush** requires that a version of a given NIS map already be present on the non-master systems before it can be transferred. Therefore, **yppush** cannot be used to propagate new maps; rather, **ypxfr** must be run on each non-master machine to request the new map.

Making new NIS maps

This section explains how to create a sample non-standard NIS map. The sample map, which we will call *mymap*, consists of key-value pairs in which the keys are strings such as **key1**, **key2**, and **key3**, and the values are **value1**, **value2**, **value3**, and so on.

You can follow either of two procedures when creating new maps:

- Use an existing ASCII file as input.
- Use standard input.

After the map is created, you must also complete these steps:

1. Edit `YP_MAP_X_LATE` on all servers to include the new map.
2. Create a new `map2ascii` routine if you wish the new map to be converted to ASCII format after it is transferred.
3. Run `ypxfr` on each server to transfer the new map.

CAUTION Creating a new NIS map is an advanced network administration procedure. You should have a thorough understanding of NIS and be familiar with administrative tools such as `awk`, `sed`, and `grep` before proceeding.

Creating a new map from an ASCII file

This section explains how to create a new NIS map from an ASCII file. An existing ASCII file named `/etc/yp/mymap.asc`, created with an editor or a shell script on the master server, is used as an example.

The ASCII file must be located in the subdirectory `home_domain` and contain a number of key-value pair strings. Create the NIS map for this file by entering the following commands:

```
cd /etc/yp
makedbm mymap.asc home_domain/mymap
```

If you want to add another pair (`key1,value1`) to the map, you can easily make the modification.

Change the map by first modifying the ASCII file. Modifications that you make to the map, rather than in the ASCII file, are lost. Make the modifications as follows:

```
cd /etc/yp
[make editorial change to mymap.asc]
makedbm mymap.asc home_domain/mymap
```

You may also wish to modify `/etc/yp/ypmake` or `/etc/yp/Makefile` on the master server so that the map can be conveniently rebuilt. Typically, an ASCII file is filtered through `awk`, `sed`, or `grep` to make it suitable for input to `makedbm`. Consult the existing `ypmake` file as a source for programming examples.

If `make(C)` is available on your system, use the mechanisms already in place in `/etc/yp/Makefile` when deciding how to create dependencies that `make` recognizes. Specifically, use `.time` files to see when the `Makefile` was last run for the map.

NOTE If you plan to use **ypmake** to maintain your new map, use the **NOPUSH** option the first time **ypmake** is run. Otherwise, **yppush** may hang when trying to access a server that does not have a copy of the new map.

Creating a new map from standard input

If there is no original ASCII file, create the NIS map on the master server with the default domain *home_domain* as shown:

```
cd /etc/yp
makedbm - home_domain/mymap
key1 value1
key2 value2
key3 value3
<Ctrl>D
```

To modify the map, use **makedbm** to create a temporary ASCII intermediate file as shown:

```
cd /etc/yp
makedbm -u home_domain/mymap > mymap.temp
```

At this point, edit *mymap.temp* so that it contains the correct information. Create a new version of the database with the commands:

```
makedbm mymap.temp home_domain/mymap
rm mymap.temp
```

Propagating a new NIS map

To propagate a new NIS map, you must include copies of the map's **dbm** files in the domain directory on each NIS server in the domain. See the "Propagating an NIS map" section of this chapter for more information.

The */etc/yp/YP_MAP_X_LATE* file contains a listing of all current maps in an NIS domain. Edit this file on all servers to include the new map before transferring the map.

If you translate the new map into an ASCII file and install it upon transfer, you need to write a **map2ascii** routine and place it in */etc/yp/map2ascii*. Consult an existing **map2ascii** routine as a source for programming examples.

To get an initial copy of the map, run the following command on each non-master server:

```
ypxfr -h new_master mymap
```

The **-h** forces **ypxfr** to get *mymap* from the master server. For subsequent transfers, **ypxfr** will read the map itself to find the master. Remember to add **crontab** entries if you want the new map transferred regularly.

NOTE The map must be globally available before non-SCO clients begin to access it. If the map is available from some NIS servers, but not all, you may observe unpredictable behavior from client programs.

Managing user accounts

The following sections describe basic procedures for NIS user account administration.

NIS account types

An NIS network has two kinds of user accounts on each computer: those that are *distributed* (and must be administered from the master machine) and those that are *local* (and must be administered from the local computer):

- | | |
|-------------|---|
| distributed | provides account information that is uniform on each machine in an NIS domain. Changes made to distributed accounts are automatically propagated through NIS maps to other NIS nodes. You can easily manage distributed accounts from a central node, the master server. |
| local | provides account information that can be unique to a particular machine. Because they are not affected by maps that are propagated between servers, local accounts allow you to exempt certain users from NIS management. In addition, changes that you make to local accounts do not affect NIS. |

NOTE When NIS is installed, pre-existing accounts become local accounts by default, on both master and slave machines.

Administration utilities

You need the following utilities to administer user accounts in an NIS network:

- | | |
|-------------------|--|
| sysadmsh | to create distributed accounts (on the master server only) |
| passmgmt | to manage local accounts (on all hosts) |
| yppassmgmt | to manage distributed accounts (on the master server only) |

The **passmgmt** and **yppassmgmt** commands also modify, retire, lock, and unlock local information in the network password file. Refer to the *SCO UNIX Operating System System Administrator's Guide* for more information on **sysadmsh**, and the **passmgmt(NADM)** and **yppassmgmt(NADM)** manual pages for more information on these commands.

NOTE You must be in multiuser mode (init state 2) with NIS daemons running before creating or modifying user accounts.

You should always run **ypmake** on the master server after making any change to user accounts.

Creating NIS accounts

The following sections explain how to create distributed and local NIS accounts.

WARNING Do not make administrative accounts (for example, *root*, *MMDF*, or *UUCP*) distributed. Doing so may affect NIS functionality in unpredictable ways and seriously compromise the security of your network.

Creating a distributed account

To create a new distributed account, follow these steps:

1. Log in to the master server machine.
2. Create a normal user account and home directory using **sysadmsh**.
3. This account is by default a distributed account. To propagate the new account to other NIS servers, enter:

```
cd /etc/yp
./ypmake passwd
./ypmake group
```

A distributed account requires only one home directory. However, a home directory must be available on all NIS-managed hosts through NFS or Automounter mounting. Refer to the corresponding chapters of this guide for more information.

NOTE A distributed account must be created on the master server only.

We recommend that you use **sysadmsh** to create distributed accounts under NIS. However, you can create distributed accounts directly using **ypassmgmt** by following these steps:

1. Log in to the master server machine.
2. Determine the shell, account name, UID, GID, comment, and home directory. (Default UID and GID values will be assigned by **ypassmgmt** if you do not include these options.) Refer to the **ypassmgmt(NADM)** manual page for more information.

3. Make sure there are no conflicts with existing accounts, then enter the account information using the following command options:

```
yppassmgt -a -u uid -g gid -c comment \  
-h homedir -s shell account_name
```

4. Assign an initial password using the `yppasswd` command.
5. Create a home directory using `sysadmsh`.
6. Run the following commands to distribute the account:

```
cd /etc/yp  
./ypmake passwd  
./ypmake group
```

Creating a local account

To create a local account, follow these steps:

1. Log in to the machine where you want the local account.
2. Determine the shell, account name, UID, GID, comment, and home directory. (Default UID and GID values will be assigned by `passmgt` if you do not include these options.) Refer to the `passmgt(NADM)` manual page for more information.
3. Make sure there are no conflicts with existing accounts, then enter the account information using the following command options:

```
passmgt -a -u uid -g gid -c comment \  
-h homedir -s shell account_name
```

4. Assign an initial password using the `passwd` command.
5. Create a home directory using `sysadmsh`.

Modifying NIS accounts

The following sections explain how to modify NIS account status and password file entries.

Changing a distributed account to a local account

To change a distributed account to a local account, use the `passmgt -n` option to add an entry to the `passwd.local` file on the local host. Enter:

```
passmgt -n account_name
```

Until you or the user change the password for the local account, the local account's attributes will still be the same as the distributed account's. This is because `passmgt` modifies the local `passwd.local` file only; it does not alter the distributed map.

If you wish to remove the user account from all machines except the local host, you must run **yppassmgt -d *account-name*** on the master server. However, it may be useful for the user to have both a distributed account and an account local to one machine, each with its own password.

Changing a local account to a distributed account

To change a local account to a distributed account, you must first create a new distributed account on the master server, then remove the local account on the local machine. Follow these steps:

1. Log in to the master server.
2. Create the new account using **sysadmsh** and distribute it using **ypmake** (see the section “Creating a distributed account” earlier in this chapter).
3. Log in to the local host.
4. Run the following command to remove the account from *passwd.local*:

```
passmgt -d account_name
```

5. When **ypmake** was run from the master server while there was still an existing local account, the entry in *passwd.local* filtered the incoming *passwd* map. This prevented the distributed account information from being incorporated into the local */etc/passwd*. You must therefore run the following command to request a new version of the distributed account:

```
/etc/yp/ypxfr passwd.nam
```

Modifying password file entries

Depending on the type of account, you can modify individual password file entries using the following command options:

- for distributed accounts (modifies */etc/password.yp* on the master server):

```
yppassmgt -m options account_name
```

- for local accounts (modifies */etc/password.local* on the local server):

```
passmgt -m options account_name
```

Except for the encrypted password field, you can modify all fields in the appropriate password files from the command line.

Removing password file entries

You can remove individual password file entries using the following command options:

- for distributed accounts (removes entries from */etc/passwd.yp* on the master server):

```
yppassmgt -d account_name
```

- for local accounts (removes entries from */etc/passwd.local* on the local server):

```
passmgt -d account_name
```

NOTE Do not use the commands **rmuser**, **rmgroup**, **rmpasswd**, or **unretire** in an NIS environment. Doing so may have unpredictable consequences for account administration.

Administering an NIS network

The following sections discuss changing NIS server configuration troubleshooting procedures, and interoperability and security issues relevant to NIS network administration.

Changing server configuration

The following sections explain how to add and remove servers and change the master server.

Adding NIS servers

After you initialize NIS, you can add servers to your NIS network at any time by:

1. editing the *ypservers* map on the master server and propagating it to other servers, and
2. installing and initializing NIS on the new server

Because the *ypservers* map is not built from an ASCII file, you must dump its contents and recreate it using NIS tools. The procedure is similar to that described in the “Modifying existing maps” section of this chapter.

The following steps explain how to add a new server:

1. Log into the master server in multiuser mode.
2. Enter the following command to dump the *ypservers* map into an ASCII file:

```
ypcat -k ypservers > /tmp/ypservers
```

3. Add the new server's name to the list of servers in */tmp/ypservers*.
4. Change directories (**cd**) to the */etc/yp* and issue the following command to rebuild the *ypservers* map:

```
cat /tmp/ypservers | makedbm - /etc/yp/domainname/ypservers
```
5. Run the following command to force the propagation of the revised *ypservers* map:

```
yppush ypservers
```
6. Log into the new server in system maintenance mode. If SCO NFS or the SCO NIS packages are not yet installed, you must first install them using **custom**; refer to the *SCO NFS Release and Installation Notes* for more information.
7. Initialize NIS on the new server following the instructions in the "Configuring NIS" section of this chapter. When you bring the new server to multiuser mode, it automatically requests that all current maps be transferred.

Removing servers

To remove a slave or copy-only server from an NIS domain you must:

1. remove the NIS package from the server using **custom**, and
2. edit the *ypservers* map on the master server and propagate it to other servers

CAUTION Do not use the following procedure to remove a master server. If you do, you must reconfigure your NIS network. Rather, refer to the "Changing a master server" section later in this chapter.

When SCO NIS is removed from a system, the */etc/passwd.local* file is removed. The */etc/passwd* file, which under NIS included both distributed and local accounts, becomes the authoritative password file. This means that distributed NIS accounts become regular user accounts when NIS is removed. If you want to restrict users with distributed NIS accounts from using the old server, you must remove their accounts before returning the old server to multiuser mode.

NOTE Although you can prevent NIS maps from reaching a server by killing its NIS daemon processes, the master still attempts to reach the server during map propagation, adding unnecessary burden on the system. For this reason, you should complete the following steps to remove NIS from a server.

Follow these steps to remove a server:

1. Log into the old server in system maintenance mode.
2. Run **custom** to remove the following NIS packages:
 - NISRT
 - NISMAN
3. If necessary, run **sysadmsh** to modify or retire previously distributed accounts.
4. If necessary, remove *root's crontab* entries for NIS map transfers.
5. Reboot the system and return to multiuser mode. If kernel parameters were adjusted when NIS was removed, you are also prompted to relink the kernel.
6. Log into the master server in multiuser mode.
7. Edit the *ypservers* map as described in the section "Adding servers" to remove the old server's name.
8. Run the following command to force the propagation of the revised *ypservers* map:

yppush ypservers

Changing the master server

You may need to change your master server, for example, if you rearrange server configuration or the master host is removed from service. Any server in the NIS domain can become the new master server, because each one has all active maps. However, because the old NIS master's name is contained in the existing map, you can neither use an existing copy at the new master nor send a copy there with **ypxfr**. Rather, you must recreate these maps by running **ypmake** (or **make**) on the new master server to associate the new server listing with each map.

The old master server must be running NIS to complete this procedure. This is because non-master servers consult their own copies of maps, which include the name of the old master server, when they receive a map transfer request originating from **yppush** on the master. New maps with the new master server listing must come from the old master server for the maps to be incorporated on slave and copy-only servers. Therefore, if the old master is not available when you change master servers, you must rebuild your entire NIS system from scratch.

CAUTION Changing master servers is a major reconfiguration of your network and should not be undertaken unless absolutely necessary. Make sure you understand the procedure thoroughly and back up your network configuration files before you begin.

The procedures described here assume that the new master is running SCO NIS. Other implementations of NIS configure servers in a different fashion; consult the documentation for your version of NIS before changing master servers.

To change NIS master servers, follow these steps:

1. Log into the new master server in multiuser mode.
2. Copy the `/etc/passwd.yp` file from the old master. If this file is not available, you must recreate distributed accounts manually. If you have modified the `/etc/yp/ypmake` or `/etc/yp/Makefile` files, you must also copy them from the old master.
3. Run the following commands to rebuild the maps:

```
cd /etc/yp
rm *.time
ypmake -NOPUSH=1
```

By removing the timestamp files, you force the maps to be rebuilt regardless of prior modifications. The `NOPUSH` option to `ypmake` prevents maps from being pushed to other servers until you have completed this entire procedure.

4. If the map only exists as a `dbm` database, you can remake it on the new master by unassembling an existing copy from any NIS server and running the unassembled version back through `makedbm`. For example, enter:

```
cd /etc/yp
ypcat -k ypservers | makedbm - domain_name/ypservers
```

You must complete this step for the `ypservers` map and any other map you have created.

5. Log into the old master server.
6. Copy the new maps from the new master server with the following command:

```
/etc/yp/ypxfr -h new_master -f map_name
```

This command must be run on the old master for every current NIS map. The `-h` option forces `ypxfr` to transfer the map from the new master; the `-f` option forces the transfer even if the contents of the old and new maps are the same.

7. Propagate the new maps from the old master server with the following command:

```
/etc/yp/yppush map_name
```

This command must be run on the old master for every current NIS map. When this is completed, all non-master servers will have received maps including the new master listing.

8. If you are removing the old master server from NIS service, edit the *ypservers* map on the new master server following the instructions in the “Removing NIS servers” section of this chapter.

You can automate the map transfer procedure on the old master server using shell scripts and other tools. For example, the following shell script automatically completes steps 6 and 7:

```
:
MAPS='oawk '{print $1 ; }' /etc/yp/YP_MAP_X_LATE'
for map in $MAPS; do
    echo moving $map
    ypxfr -h new_master -f $map
    yppush $map
done
```

Make sure your */etc/yp/YP_MAP_X_LATE* file is current before running this command; refer to the “Understanding */etc/yp/YP_MAP_X_LATE*” section of this guide for more information.

Debugging NIS

This section describes how to solve common NIS problems.

Checking TCP/IP

Before troubleshooting what appears to be an NIS problem, verify that TCP/IP is up, configured, and running properly.

Checking the protected password file

After you initialize and activate NIS at any time during system operation, you can detect errors and verify the integrity of the protected password database with the */tcbin/authck* and the */tcbin/integrity* commands. Refer to the *authck(ADM)* and the *integrity(ADM)* manual pages for more information.

Different versions of an NIS map

Because NIS works by propagating maps among servers, different versions of a map may be found at servers on the network. This variation of map version occurs only during propagation and is normally transient.

Typically, normal update is prevented when an NIS server or gateway machine between NIS servers is down during a map transfer attempt. When all of the NIS servers and all of the gateways between them are running, **ypxfr** should succeed.

If a particular slave or copy-only server has problems updating, run **ypxfr** interactively on that server. If **ypxfr** fails, the screen displays a reason. This helps in understanding and fixing the problem. If **ypxfr** fails intermittently, create a log file to enable logging of messages. The following program saves all output from **ypxfr**:

```
ypslave$ cd /usr/adm
ypslave$ touch nislog
```

The output resembles what **ypxfr** generates when it is run interactively, but each line in the log file is timestamped. The timestamp shows when **ypxfr** began its work. If copies of **ypxfr** ran simultaneously but their work took differing amounts of time, they can write their summary status line to the log files in an order different from that in which they were invoked. Any pattern of intermittent failure shows up in the log. When the problem is fixed, turn off logging by removing the log file. If you do not do this, the log file grows without limit.

On the misbehaving NIS slave or copy-only server, also inspect the */usr/lib/crontab* file and the **ypxfr** shell scripts that it invokes. Typing mistakes in these files cause propagation problems, as do failures to refer to a shell script within **crontab**, or failures to refer to a map within any shell script.

Make sure the NIS slave or copy-only server is in the map **ypservers** within the domain. If not, the nonmaster server functions correctly as a server but is not notified by **yppush** when a new copy of a map exists.

***ypserv** crashes*

If the **ypserv** process crashes almost immediately and does not stay up even with repeated activations, the method for debugging is virtually identical to that described in the section “**ypbind** crashes” later in this chapter. Check the **portmap** daemon by entering:

```
ypserver$ ps -ef | grep portmap
```

If there is no **portmap**, reboot the local server by issuing the **shutdown** command. If the **portmap** daemon is present, enter the following:

```
ypserver$ rpcinfo -p
```

Output should be similar to the following example:

```
[program, version, protocol, port, service]:

[100000, 2, tcp, 111, portmapper]
[100000, 2, udp, 111, portmapper]
[100024, 1, udp, 1030, status]
[100024, 1, tcp, 1040, status]
[100020, 1, udp, 1038, llockmgr]
[100020, 1, tcp, 1041, llockmgr]
[100021, 2, tcp, 1042, nlockmgr]
[100021, 1, tcp, 1043, nlockmgr]
[100021, 1, udp, 1048, nlockmgr]
[100021, 3, tcp, 1044, nlockmgr]
[100021, 3, udp, 1053, nlockmgr]
```

The four entries representing the **ypserv** process are:

```
[100004, 1, udp, 1037, ypserv]
[100004, 1, tcp, 1046, ypserv]
[100004, 2, udp, 1051, ypserv]
[100004, 2, tcp, 1055, ypserv]
```

If they are not there, **ypserv** was unable to register its services. Reboot the machine. If they are there and change each time an attempt is made to restart */etc/ypserv*, reboot the machine.

Commands hang

The most common problem at an NIS client node is that a command hangs and generates a console message:

```
NIS: server not responding for domain <odette>..
Still trying
```

Occasionally, commands hang even though the overall system appears to be working correctly and new commands can be run.

The preceding display indicates that **ypbind** on the local machine cannot communicate with **ypserv** in the domain *odette*. This often happens when machines that run **ypserv** crash. It can also occur if the network or the NIS server machine is so overloaded that **ypserv** cannot respond to the local machine's **ypbind** within the timeout period.

Under these circumstances, all of the other NIS client nodes on the net show the same or similar problems. The condition is temporary in most cases; the messages usually go away when the NIS server machine reboots and **ypserv** returns, or when the load on the NIS server nodes or the Ethernet decreases.

However, in the following circumstances, the condition does not change until the steps described are followed:

- The NIS client has not set, or has incorrectly set, **domainname** on the machine. Clients must use a domain name that NIS servers know. Use **domainname(NC)** to see the client domain name, and compare it with the domain name set on the NIS servers. The domain name should be set in */etc/rc2.d/S89nfs*. When */etc/rc2.d/S89nfs* fails to set or incorrectly sets **domainname**, enter the following commands as *root*:
 - Edit */etc/rc2.d/S89nfs* to correct the **domainname** line; the correct domain name enables the machine to boot properly.
 - Enter **domainname good_domain_name** to set the **domainname**.
- If the domain name is correct, check the local network to make sure that it has at least one NIS server machine. Machines can only bind to a **ypserv** process on the local network, not on another accessible network. At least one NIS server for each machine's domain must be running on the local network. Having two or more NIS servers improves availability and response characteristics for NIS.
- If the local network has an NIS server, check it to make sure that it is running. Also check other machines on the local network. If several client machines have problems simultaneously, a server problem is likely. Try the **ypwhich** command on a client machine that is running; if **ypwhich** never returns an answer, kill the **ypwhich** process. To do so, find the process number using the **ps** command, then issue a **kill** command from another screen or terminal:

```
ps -ef | grep yp
kill -9 ypwhich_PID_#
```

(The second column of the **ps** display is the PID number.) The previous **ps** command also shows if there are **ypserv** and **ypbind** processes running. If the server's **ypbind** daemon is not running, start it by entering:

```
/etc/ypbind
```

If a **ypserv** process is running, enter a **ypwhich** on the NIS server machine. If **ypwhich** returns no answer, **ypserv** has probably hung and should be restarted. As super user, kill the existing **ypserv** process as described above and restart **ypserv** by issuing the command:

```
/etc/ypserv
```

If **ps** shows no **ypserv** process running, start one.

NOTE You must be logged in as *root* to start or kill **yp** processes. You can determine their status from multiuser mode using the **ps -ef** command, but only in Unsecure Mode.

NIS unavailable

If NIS is unavailable on one machine even though other machines on the network appear to be functioning, many different symptoms may be displayed, including:

- Some commands appear to operate correctly, while others terminate and print an error message about the unavailability of NIS.
- Some commands enter a recovery mode particular to the program involved.
- Some commands or daemons crash with obscure messages or no message at all.

For example, messages such as the following might be displayed:

```
my_machine$ ypcat myfile
ypcat: can't bind to NIS server for domain <odette>.
        Reason: can't communicate with ypbind.
```

```
my_machine$ /etc/yp/ypoll myfile
Sorry, I can't make use of NIS. I give up.
```

In such cases, run the following command on a directory containing files owned by many users (including users not in the local machine's */etc/passwd* file, for example */usr*):

```
my_machine$ ls -l
```

If the `ls -l` command reports file owners not in the local machine's */etc/passwd* file as numbers, rather than names, this and similar symptoms usually indicate that NIS is not working and the `ypbind` process is not running. Use the command `ps -ef` to check for the process. If it is not found, enter the following command to start the process:

```
my_machine$ /etc/ypbind
```

NIS problems should disappear.

ypbind crashes

If `ypbind` crashes almost immediately each time it is started, problems probably lie in some other part of the system. Determine the presence of the `portmap` daemon by typing:

```
my_machine$ ps -ef | grep portmap
```

If it is not running, reboot the machine using */etc/shutdown*.

If `portmap` itself does not stay up or behaves strangely, problems are probably more fundamental. In this case, check the state of the network.

You may be able to talk to the **portmap** on the local machine from a machine operating normally. From such a machine, enter the following:

```
other_machine$ rpcinfo -p your_machine_name
```

If **portmap** is functioning correctly, the output should be similar to the following:

```
[program, version, protocol, port]:

[100005, 1, tcp, 1046 llockmgr]
[100001, 2, tcp, 1055 llockmgr]
[100001, 1, tcp, 1055 nlockmgr]
[100002, 1, tcp, 1052 nlockmgr]
[100008, 1, tcp, 1049 nlockmgr]
[100007, 1, tcp, 1027 nlockmgr]
[100007, 1, udp, 1026 nlockmgr]
[100007, 2, tcp, 1031 nlockmgr]
[100007, 2, udp, 1030 nlockmgr]
```

Note that port numbers are different on different machines. The four entries that represent the **ypbind** process are:

```
[100007, 1, tcp, port_# ypbind]
[100007, 1, udp, port_# ypbind]
[100007, 2, tcp, port_# ypbind]
[100007, 2, udp, port_# ypbind]
```

If these entries are not there, **ypbind** was unable to register its services. Reboot the machine. If the entries are there and change each time you attempt to restart */etc/ypbind*, reboot the system even if **portmap** is up.

NIS interoperability

SCO systems resolve all lookup requests locally. The group and password entries "+" and "-" that allow and restrict network access are not supported by SCO systems. If your SCO system is acting as the master server, it should be able to resolve distributed lookup requests from non-SCO systems.

Regardless of whether your master server is on an SCO or a non-SCO system, all of the maps received by SCO systems are filtered before they are integrated (that is, translated into ASCII format). For example, */etc/yp/map2ascii/group.nam* filters incoming maps that are derived from */etc/group*. The filtering process eliminates:

- conflicts due to two or more users with the same UID
- entries found in *map.local* that supersede incoming map information
- the "+" entry that allows network access from incoming maps
- UIDs and GIDs outside the valid range. Look for the **MIN_ADMIN_UID** variable in the */etc/default/authsh* file for a list of valid ranges.

In addition, the */etc/yp/map2ascii/passwd.nam* and */etc/yp/map2ascii/group.nam* files check for security compatibility.

NOTE If you wish to use NIS on a network that includes hosts configured with SCO UNIX System V C2 security specifications, verify that all of the maps provided by the master server are compatible with the more restrictive machines on the network. If the filtering process does not interpolate the maps compatibly with restrictive machines, you may have to edit the maps transferred by the non-SCO machine.

Security under NIS

This section describes the files and entries involved in security issues.

Map integration restrictions in Secure Mode

If your copy-only server is configured with one of the Secure Mode defaults, its *map2ascii* files will prevent any *passwd* or *group* maps from being integrated, regardless of their source.

WARNING Do not attempt to circumvent C2 security restrictions by removing security checks from the *map2ascii* files. Doing so may cause unpredictable behavior and is therefore an unsupported modification.

Special NIS password change

When a password is changed with the *passwd(C)* command without NIS, the entry given in the system password database file is changed. To change a password in NIS, use the *yppasswd(NC)* command. To enable this service, you must start up the server daemon, *yppasswdd(NADM)*, on the machine serving as the master for the NIS password file. For convenience, the original */bin/passwd* file is moved to */usr/lib/nisrt/bin/passwd* and a link to */bin/yppasswd* is made. This means that when you invoke *passwd*, you are actually invoking *yppasswd*.

Manual pages covering security

More detailed information on security issues can be found on the following manual pages:

- **passwd(C)**
- **export(NF)**
- **group(NF)**
- **netgroup(NF)**
- **yppasswd(NC)**
- **yppasswdd(NADM)**

Directory of NIS services

The following list describes all of the NIS services and their corresponding manual pages.

domainname(NC)	sets or displays the name of the current NIS domain
makedbm (NADM)	a low-level tool for building an NIS map. It is normally called from <i>/etc/yp/ypmake</i> (or the <i>Makefile</i> when using make), although it can also be used to modify maps for which no ASCII files exist or to create new maps. makedbm can also “disassemble” a map so you can see the key-value pairs it contains. The disassembled form can be modified with standard tools (such as editors, awk , grep , and cat) and input back into makedbm .
map2ascii(NADM)	describes the NIS commands that translate certain maps into their ASCII counterparts
passmgmt(NADM)	an NIS local account management utility that can create local accounts and change local accounts to distributed accounts
ypbind(NADM)	the NIS binder daemon. ypbind is a client agent that passes NIS requests to the server daemon. It must run on all machines that use NIS services, both servers and clients. Information on ypbind is included in the ypserv(NADM) manual page.
ypcat(NC)	dumps the contents of an NIS map. Use ypcat instead of makedbm -u in cases where it does not matter which server's version is seen.
ypclnt(NS)	a group of function calls that provides an interface to NIS

ypfiles(NF)	describes the database structure of NIS
ypinit(NADM)	the NIS database initialization tool. Many maps must be constructed from files located in <i>/etc</i> such as <i>/etc/hosts</i> , <i>/etc/passwd</i> , and others. ypinit does all such construction automatically. In addition, it constructs initial versions of maps required by the system but not built from files in <i>/etc</i> , such as the map <i>ypservers</i> . Use this tool to set up all NIS servers for the first time. Do not use it as a general administrative tool for running systems.
ypmake(NADM)	a script that builds several commonly changed components of the NIS database from ASCII files on the master and propagates maps to other servers. ypmake looks to see if make(CP) is available on your system; if it is available, make is called along with the <i>Makefile</i> script. If not, ypmake runs independently.
ypmapxlate(NF)	describes the <i>YP_MAP_X_LATE</i> file, which contains a complete listing of active NIS maps. It is used by NIS utilities when creating, modifying and propagating maps. It is also a translation table that handles NIS map names.
ypmatch(NC)	prints the value for one or more specified keys in an NIS map. There is no control over which server's version of the map is seen.
yppassgmt(NADM)	an NIS distributed account management utility. It can create and change distributed accounts, and change distributed accounts to local accounts.
yppasswd(NC)	an NIS password administration utility that changes appropriate password databases. It uses the network to contact the master server for password changes on distributed accounts. Because yppasswd is linked to passwd(C) , NIS passwords are updated transparently using the conventional passwd command.
yppasswdd(NADM)	the daemon that handles password change requests from yppasswd(NC) to modify the NIS password file. This server is not run by default. If you want to enable remote password updating for the network information service, put an entry for yppasswdd in the <i>/etc/nfs</i> file on the master server. It cannot be run on slave and copy-only servers.
yppoll(NADM)	asks any ypserv for information about a map that it holds internally

yppush(NADM)	requests each of the ypserv processes within a domain to transfer a particular map, waits for a summary response from the transfer agent, then prints out the results for each server. Run yppush on the NIS master server only.
ypserv(NADM)	the NIS database server daemon. ypserv satisfies Remote Procedure Calls (RPC's) for NIS services. It must run on each NIS server machine. The ypserv(NADM) manual page also contains information about the ypbind and ypxfrd daemons.
ypset(NADM)	tells a ypbind process (by default, the local one) to get NIS services for a domain from a named NIS server
ypwhich(NADM)	tells you which NIS server a node is currently using for NIS, or which NIS server is master of a named map
ypxfr(NADM)	transfers an NIS map from the master to non-master servers, using NIS itself as the transport medium. You can run ypxfr interactively, or periodically from the <i>crontab</i> file. In addition, ypserv uses ypxfr as its transfer agent when propagating a map from the master server.
ypxfrd(NADM)	the NIS map transfer daemon. ypxfrd satisfies requests from non-master servers to transfer maps. It must run on each NIS server machine. Information on ypxfrd is included in the ypserv(NADM) manual page.

Chapter 4

Automounter

Automounter allows remote NFS filesystems to be mounted automatically and transparently. This chapter explains how Automounter works and how to configure and administer Automounter in your network.

Understanding Automounter

The chapter “NFS: Network File System” of this guide explains how to export and mount hierarchies through NFS with the **mount** command. You can also NFS mount file hierarchies using a different method — *automounting*.

The **automount** program enables users to mount and unmount remote directories on an as-needed basis. Whenever a user on a client machine running Automounter invokes a command that accesses a remote file or directory, such as opening a file with an editor, the hierarchy to which that file or directory belongs is mounted and remains mounted for as long as it is needed. No mounting is done at boot-time, reducing significantly the time needed to boot up. It is all done automatically and transparently.

How Automounter works

Automounter actually mounts all file hierarchies under the directory */tmp_mnt*. It then uses NFS’s symbolic link support to associate the actual mount point with the one in */tmp_mnt*. This symbolic link is conceptually similar to a standard UNIX System V **link(ADM)** but it can be established between pathnames residing on different filesystems. The result is that the file hierarchies, actually mounted under */tmp_mnt*, appear to be mounted on the correct mount points.

To illustrate a direct mount, let's assume that Automounter was configured to mount the remote file hierarchy *bigmo:/usr/man* on the local directory */usr/man*. The first user to issue the command `cd /usr/man` causes Automounter to perform the equivalent of the following commands:

1. `mkdir /tmp_mnt/usr/man`

If the */tmp_mnt* or */tmp_mnt/usr* directories do not exist at the time, Automounter creates them as well.

2. `mount -f NFS bigmo:/usr/man /tmp_mnt/usr/man`

Automounter performs a conventional NFS mount.

3. `ln -s /tmp_mnt/usr/man /usr/man`

The `ln -s` command shows a symbolic link operation associating */usr/man* and */tmp_mnt/usr/man*. However, Automounter does not really create symbolic links. Rather, it acts as a symbolic link server, in this case redirecting access requests destined for */usr/man* to */tmp_mnt/usr/man*.

As a result, *bigmo:/usr/man* appears to be directly NFS-mounted on the local */usr/man* directory.

Figure 4-1 illustrates direct mounting, showing how Automounter can mount automatically the online UNIX manual pages from a specified server.

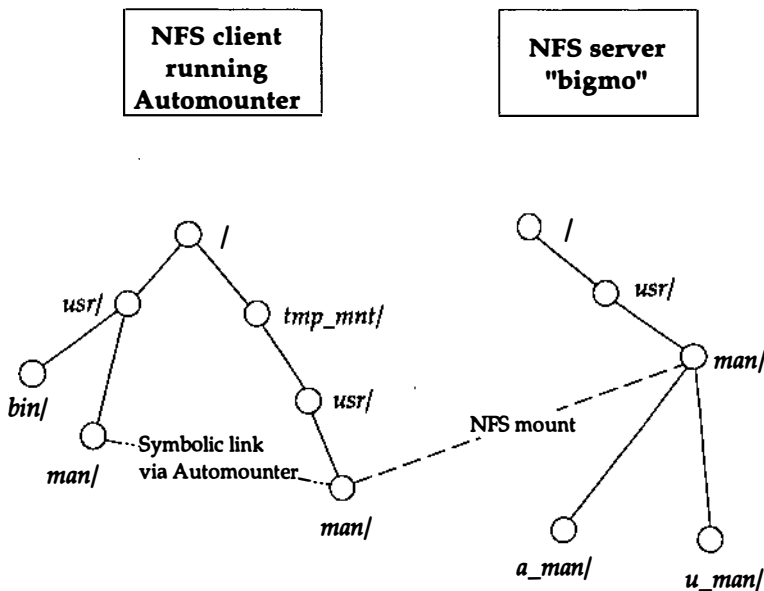


Figure 4-1 An example of direct mounting using Automounter

To illustrate an indirect mount, let's assume that Automounter is configured such that the remote hierarchies *lucy:/doc.stuff/spellers* and *alice:/doc/bin/roff* appear to be mounted on the local directory */doc_tools*. When a user issues the commands */doc_tools/spellers/mspell* and */doc_tools/roff/nroff*, it causes Automounter to perform the equivalent of the following commands:

1. **mkdir /tmp_mnt/doc_tools/spellers**
mkdir /tmp_mnt/doc_tools/roff

If the */tmp_mnt* or */tmp_mnt/doc_tools* directories do not exist at the time, Automounter creates them as well.

2. **mount -f NFS lucy:/doc.stuff/spellers /tmp_mnt/doc_tools/spellers**
mount -f NFS alice:/doc/bin/roff /tmp_mnt/doc_tools/roff

Automounter performs conventional NFS mounts.

3. **ln -s /tmp_mnt/doc_tools/spellers /doc_tools/spellers**
ln -s /tmp_mnt/doc_tools/roff /doc_tools/roff

Again, Automounter does not really create symbolic links. Rather, it acts as a symbolic link server, in this case redirecting access requests destined for */doc_tools* to */tmp_mnt/doc_tools*.

NOTE The directory */doc_tools* in this example is created and managed by Automounter; it does not exist on any disk.

Figure 4-2 illustrates indirect mounting, showing how Automounter can assemble and automatically mount a directory of tools from specified servers.

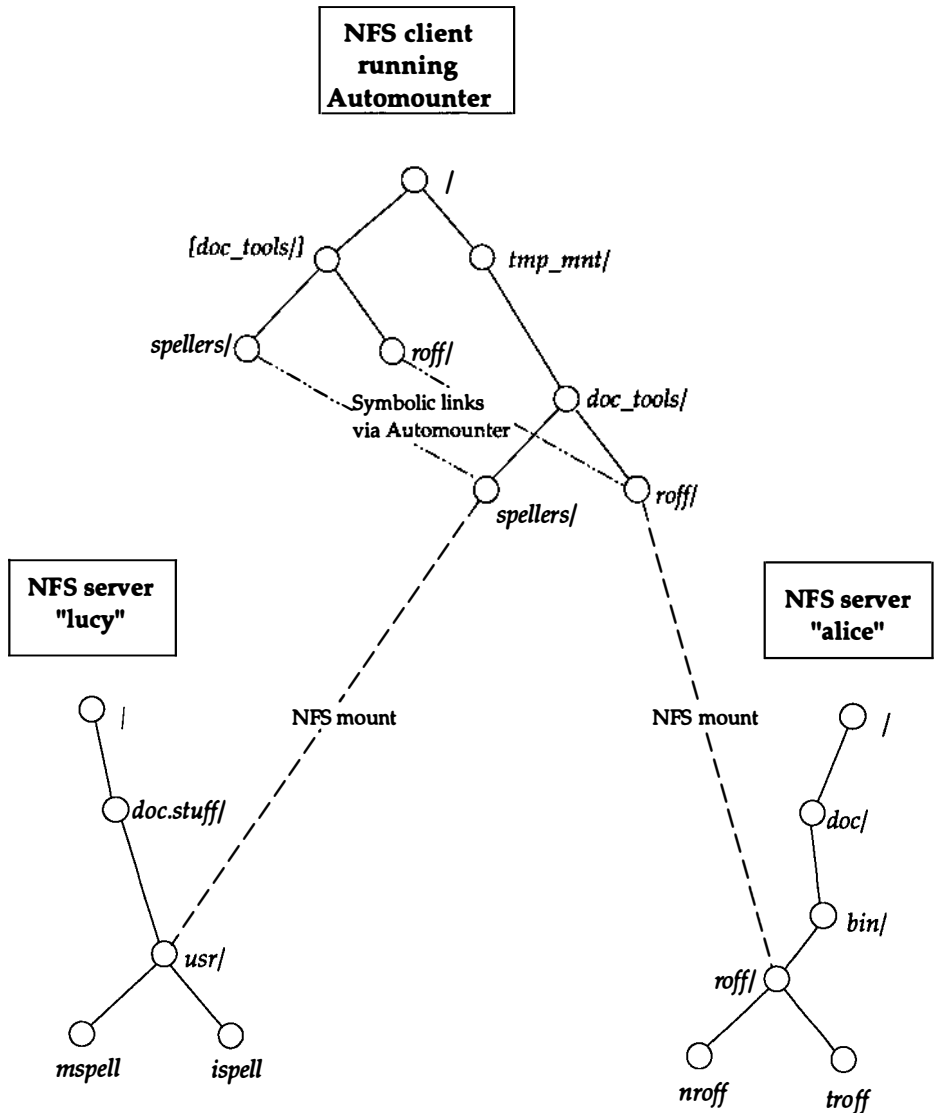


Figure 4-2 An example of indirect mounting using Automounter

A comparison of these examples illustrates the major differences between direct and indirect mounting:

- direct mounting
 - mounts complete directories
 - is symbolically linked to a subdirectory in a local filesystem without hiding other existing subdirectories
- indirect mounting
 - assembles discontinuous subdirectories under a virtual parent directory
 - creates a parent directory that cannot contain files other than those managed by Automounter

Thus, in our examples, a directly automounted */usr/man* directory can coexist with a local */usr/lib* directory. However, a local */doc_tools* directory (if it existed previous to Automounter configuration) is hidden by an indirectly automounted */doc_tools/spellers* directory.

Automounter maps

Automounter determines which mount points to monitor and file hierarchies to mount from a special set of files called *maps*. Maps are configuration files that describe the possible configurations for NFS mounts on the system. These maps can reside on the local machine or be managed via NIS.

There are three types of Automounter maps:

- | | |
|----------|---|
| direct | lists complete directories and their locations. Whenever any node in the directory hierarchy is accessed, Automounter mounts the entire directory. |
| indirect | lists individual directory entries and their locations. Automounter creates a virtual parent directory from subdirectories that may reside on different devices or hosts. Whenever any node in the subdirectory hierarchy is accessed, Automounter mounts the subdirectory. |
| master | lists all indirect and direct maps, applicable options and mount points. If a master map <i>auto.master</i> is present when the system boots up, Automounter reads the master map and starts a daemon for the specified mount point. |

Direct automounting is similar to conventional NFS mounting in that an entire filesystem is exported. For example, if the filesystem */usr/man* on the server *bigmo* is listed in a direct map on the client, it is mounted automatically whenever a user on the client accesses the filesystem. This may be done, for instance, by issuing the command **man rpcinfo**, which accesses the file *./cat.NADM/rpcinfo.NADM* in the */usr/man* directory. The mount point in this

example appears to the user to be */usr/man*, and as in any other **mount** procedure, the existing directory entries are hidden by the directory mounted on top of it.

Indirect mounting allows you to distribute resources to different devices or hosts while maintaining the appearance to the user that these resources belong to the same filesystem. For example, a user may want access to a number of documentation tools, including layout tools, fonts, and spell checkers. The system administrator can use Automounter to make it appear that all these resources are available in the */doc_tools* directory of the user's home machine, even though each resource could be on a different device or host.

When **automount** is started, either from the command line or */etc/nfs*, it forks a daemon to serve the mount points specified in the maps. It does this by establishing itself as the NFS server for the specified mount points. When one of these mount points is accessed or crossed, Automounter fields the NFS protocol request as would the real NFS server daemon, **nfsd**(NADM). It then mounts the appropriate remote hierarchy, as specified in Automounter maps.

When a predetermined amount of time has elapsed without the hierarchy being accessed, Automounter automatically unmounts it.

Planning an Automounter configuration

A typical use for Automounter is an installation of several large server systems and 10 or more workstations where NIS is being used. NFS mounts for each user on each workstation are managed centrally from one of the servers using NIS, which also allows distributed access to a shared database of user accounts. Automounter ensures that users can access essential system files regardless of the workstation from which they log in to the system. In a larger system, you might use NIS to accomplish such transparency without additional administration on each workstation or the performance burden represented by dozens of active NFS mounts.

Automounter is also useful on a smaller NFS network where hosts must double as clients and servers. NIS management might not be necessary in such a configuration, but Automounter can still reduce the number of active mounts and improve network access and efficiency.

Automounter also allows a client to configure redundant servers. That is, it allows a client to configure a set of servers from which a particular file hierarchy can be mounted. Therefore, in the direct mount example (Figure 4-1), instead of just specifying *bigmo* as the remote server for */usr/man*, it is possible to specify remote servers *bigjoe* and *bigben*, as well. If configured in this fashion, Automounter locates the first available server (using an RPC query), then mounts */usr/man* from that system.

Configuring Automounter

To configure Automounter into an NFS network, you must:

- ensure that the remote directories you wish to mount are listed in the */etc/exports* files on your servers (see the “NFS: Network File System” chapter of this guide for more information)
- create at least one Automounter map in the */etc* directory on the client
- start Automounter by issuing the **automount** command, either on the command line or through an entry in the */etc/nfs* start-up script

The following sections explain how to prepare Automounter maps and start Automounter.

NOTE We recommend that you use indirect maps whenever possible, because they can be modified more easily and they create fewer entries in the mount table (*/etc/mnttab*).

We also recommend that you create a master map. Doing so simplifies Automounter administration and enables Automounter to start automatically when your system goes to multiuser mode.

Preparing the maps

Unlike the **mount** command, **automount** does not consult the */etc/default/filesys* file for information on which file hierarchies to mount. Rather, it consults the map file(s) specified on the command line at startup time. If no maps are specified, it looks for an NIS map called *auto.master*. If the NIS *auto.master* is not present, **automount** exits silently.

By convention, all Automounter maps are located in the */etc* directory and have filenames prefixed with *auto*.

Writing a master map

Each line in the master map, by convention called */etc/auto.master*, has the syntax:

mount-point map-name [mount-options]

Here:

mount-point is the full pathname of a directory. If the directory does not exist, Automounter creates it if possible. If the directory exists and is not empty, mounting on it hides its contents. Automounter issues a warning message in this case.

- map-name** is the map Automounter uses to find the mount points and locations.
- mount-options** is an optional comma-separated list of options that regulate the mounting of the entries mentioned in *map-name*, unless the entries in *map-name* list other options

A line whose first character is “#” is treated as a comment, and everything that follows until the end of the line is ignored. A backslash “\” at the end of the line permits splitting long lines into shorter ones. The notation “/-” as a mount point indicates that the map in question is a direct map and no particular mount point is associated with the map as a whole.

A typical *auto.master* file contains:

```

/-          /etc/auto.direct      -ro
/home       /etc/auto.home       -rw
/net        -hosts
```

Automounter recognizes the following special mount points and maps:

Mount point “/-”

In the earlier example, the mount point /- is a filler that Automounter recognizes as a directive not to associate the entries in */etc/auto.direct* with any directory. Rather, the mount points are the ones mentioned in the map. (Remember, in a direct map the key is a full pathname.)

Mount point /home

The mount point */home* is the directory under which the entries listed in */etc/auto.home* (an indirect map) are to be mounted. That is, they are mounted under */tmp_mnt/home*, and the NFS symbolic link machinery is used to associate */home/directory* and */tmp_mnt/home/directory*.

/home is the preferred convention for automounting home directories. Doing so allows you to provide users with their familiar home environment regardless of the host from which they log in to the system.

Mount point /net

Automounter mounts under the directory */net* all the entries under the special map, *-hosts*. This is a built-in map that does not use any external files except the hosts database */etc/hosts* or the NIS map, *hosts.byname*, if NIS is running. The */net* convention affords the user transparent access to any NFS server on the network (where the client's access permissions match those listed in */etc/exports*).

NOTE Because Automounter does not mount the entries until needed, the specific order is not important. Once the `automount` daemon is in place, a user entering the following command changes directory to the top of the file hierarchy (that is, the *root* filesystem) on the machine *gumbo*,

```
cd /net/gumbo
```

This is true as long as the machine is in the hosts database and it exports any of its filesystems. However, the user may not see all the files and directories under */net/gumbo*. This is because Automounter can mount only the exported filesystems of host *gumbo*, in accordance with the restrictions placed on the exporting in */etc/exports*.

Automounter performs the following tasks in response to the previous command.

1. Query the server's mount service using the RPC null procedure to see if it is responding.
2. Request the list of exported hierarchies from the server.
3. Sort the exported list according to length of the pathname.

```
/usr/src
/export/home
/usr/src/scs
/export/root/blah
```

This sorting ensures that the mounting is done in the proper order, that is, */usr/src* is done before */usr/src/scs*.

4. Proceed down the list, mounting all the filesystems at mount points in */tmp_mnt* (creating the mount points as needed).

NOTE Automounter has to mount all of the filesystems that the server in question exports. Therefore, if the user issues the following `ls` command:

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

Automounter mounts all of *gumbo's* exported systems, not just */usr*.

In addition, unmounting that occurs after a certain default period is from the bottom up. This means if one of the directories at the top is busy, Automounter has to remount the hierarchy and try again later.

The *-net* special map thus provides a very convenient way for users to access directories in many different hosts without having to use `rlogin` or `rsh`. Furthermore, they no longer have to modify their */etc/default/filesys* files or mount the directories by hand as superuser.

Notice that both */net* and */home* are arbitrary names dictated by convention. Automounter creates them if they do not exist already.

Writing an indirect map

The syntax for an indirect map is:

key [*mount-options*] *location*

Here:

key is the pathname of the mount point.

mount-options are the options you want to apply to this particular mount.

location is the location of the resource, specified as *server:pathname[:subdirectory]*.

key is the basename (not the full pathname) of the directory that is used as mount point. Once the key is obtained by Automounter, it is suffixed to the mount point associated with it either by the command line or by the master map that invokes the indirect map in question.

As in the master map, a line whose first character is “#” is treated as a comment and everything that follows until the end of the line is ignored. A backslash at the end of line permits splitting long lines into shorter ones.

The only formal difference between a direct and an indirect map is that the key in a direct map is a full pathname, whereas in an indirect pathname it is a simple name (no slashes). For instance, the following is an entry in a direct map:

```
/usr/man                      -ro                      goofy : /usr/man
```

The following is an entry in an indirect map:

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

Clearly, the **key** in the indirect map requires more information, specifically the actual location of the mount point, *parsley*. This information must be provided at the command line or through another map. For instance, if the above line is part of a map called */etc/auto.veggies*, the appropriate level of qualification can be provided through invoking **automount** as follows:

```
automount /veggies /etc/auto.veggies
```

The same qualification can be achieved by adding the following specification to the master map:

```
/veggies                      /etc/auto.veggies                      -ro, soft, nosuid
```

In either case, the mount directory */veggies* qualifies the entries in the indirect map */etc/auto.veggies*. The end result is that the hierarchy */usr/greens* from the machine *veggies* is mounted on */veggies/parsley*, when needed.

For instance, one of the entries in the master map presented above as an example, reads:

```
/home      /etc/auto.home      -rw
```

Here */etc/auto.home* is the name of the indirect map that contains the entries to be mounted under */home*.

A typical *auto.home* map might contain:

willow		willow : /home/willow
cypress		cypress : /home/cypress
poplar		poplar : /home/poplar
pine		pine : /export/pine
apple		apple : /export/home
ivy		ivy : /home/ivy
peach	-rw, nosuid	peach : /export/home

As an example, assume that the map above is on the host *oak*. If the user, *laura*, has an entry in the password database specifying her home directory as */home/willow/laura*, then whenever she logs into machine *oak*, Automounter mounts (as */tmp_mnt/home/willow*) the directory */home/willow* residing in machine *willow*. If one of the directories is indeed *laura*, she is in her home directory, which is mounted read/write, as specified by the options field in the master map entry.

Suppose, however, that *laura*'s home directory is specified as */home/peach/laura*. Whenever she logs into *oak*, Automounter mounts the directory */export/home* from *peach* under */tmp_mnt/home/peach*. Her home directory is mounted read/write, *nosuid*. Any option in the file entry overrides all options in the master map or the command line.

Now, assume the following conditions occur:

- User *laura*'s home directory is listed in the password database as */home/willow/laura*.
- The machine *willow* exports its *home* hierarchy to the machines mentioned in *auto.home*.
- All those machines have a copy of the same *auto.home* and the same password database.

Under these conditions, the user *laura* can run **login** or **rlogin** on any of these machines and have her home directory mounted in place for her.

Furthermore, now *laura* can also enter the command:

```
cd ~bigmo
```

Automounter mounts *bigmo*'s home directory for her (if all permissions apply).

On a network without NIS, you have to change all the relevant databases (such as */etc/passwd*) on all systems on the network to accomplish this. On a network running NIS, make the changes on the NIS master server and propagate the relevant databases to the slave servers.

Writing a direct map

The syntax for a direct map (like that for an indirect map) is:

key [*mount-options*] *location*

Here:

key is the full pathname of the mount point. (Remember that in an indirect map, this is not a full pathname.)

mount-options are optional but, if present, override the options of the calling line or the defaults for the entry in question. (See the section "Starting Automounter" later in this chapter.)

location is the location of the resource, specified as *server:pathname[:subdirectory]*.

The following is a sample */etc/auto.direct* map:

```
/usr/local \
        /bin    -ro, soft    ivy : /usr/local/bin \
        /src    -ro, soft    ivy : /usr/local/src
/usr/man
        -ro, soft    oak : /usr/man \
                    rose : /usr/man \
                    willow : /usr/man
/usr/games
        -ro, soft    peach : /usr/games
/usr/spool/news
        -ro, soft    pine : /usr/spool/news
```

Two unusual features in this map, multiple mounts and multiple locations, are discussed in the next two subsections.

Multiple mounts

A map entry can describe a multiplicity of mounts, where the mounts can be from different locations and with different mount options. Consider the first entry in the previous example:

```
/usr/local \
        /bin    -ro, soft    ivy : /usr/local/bin \
        /src    -ro, soft    ivy : /usr/local/src
```

This is, in fact, one long entry whose readability has been improved by splitting it into three lines by using the backslash and indenting the continuation lines with blank spaces or tabs. This entry mounts */usr/local/bin* and */usr/local/src* from the server *ivy*, with the options read-only and soft. The entry could also read:

```
/usr/local \
        /bin    -ro, soft    ivy : /usr/local/bin \
        /src    -ro         oak : /usr/local/src
```

Here, the options are different and more than one server is used.

The preceding example might also be represented with two separate entries, for example:

```
/usr/local/bin      -ro, soft      ivy: /usr/local/bin
/usr/local/src      -ro              oak: /usr/local/src
```

The difference is that the first case, a multiple mount, guarantees that both directories will be mounted when you reference one of them. In the case of the separate entries, if you reference one directory you cannot change (cd) to the other using a relative path, because it is not mounted yet.

```
cd /usr/local/bin
cd ../src
../src: No such file or directory
```

A multiple mount obviates this problem. In multiple mounts, each file hierarchy is mounted on a subdirectory within another file hierarchy. When the root of the hierarchy is referenced, Automounter mounts the whole hierarchy.

A special case of multiple mounts is when the root of a hierarchy has to be mounted as well. This is called a “hierarchical” mount. The following illustration shows a true hierarchical mounting:

```
/usr/local \
/          -rw      peach: /usr/local \
/bin      -ro, soft ivy: /usr/local/bin \
/src      -ro       oak: /usr/local/src
```

NOTE A true hierarchical mount can be problematic if the server for the root of the hierarchy goes down. Any attempt to unmount the lower branches fails, because the unmounting has to proceed through the mount root, which also cannot be unmounted while its server is down.

If multiple mounts share the same options, it is not necessary to duplicate these options for each server. Rather, the options precede the different mounts, as follows:

```
/usr/local      -ro, soft \
                /bin      ivy: /usr/local/bin \
                /src      oak: /usr/local/src
```

If one of the mount points needs different specification, modify only its options:

```
/usr/local      -ro, soft \
                /bin      ivy: /usr/local/bin \
                /src      -rw      oak: /usr/local/src
```

Multiple locations

The mount point `/usr/man` in the following direct map lists more than one location.

```

/usr/local \
    /bin    -ro, soft    ivy: /usr/local/bin \
    /src    -ro, soft    ivy: /usr/local/src
/usr/man    -ro, soft    oak: /usr/man \
                                rose: /usr/man \
                                willow: /usr/man
/usr/games    -ro, soft    peach: /usr/games
/usr/spool/news -ro, soft    pine: /usr/spool/news

```

This means that the mounting can be done from any of the replicated locations, in this case, *oak*, *rose*, or *willow*. This list can also be expressed as a comma-separated list of servers, followed by the colon and the pathname (as long as the pathname is the same for all of the replicated servers), for example:

```

/usr/man    -ro, soft    oak, rose, willow: /usr/man

```

The first server to respond to the RPC ping issued by Automounter is selected, and an attempt is made to mount from it. The list does not imply an ordering, although servers on the local network will be contacted first.

This redundancy, very useful in an environment where individual servers may or may not be exporting their file systems, exists only at mount time. There is no status checking of the mounted-from server by Automounter once the mount occurs. If the server goes down while the mount is in effect, the filesystem becomes unavailable. An option here is to wait five minutes until the auto-unmount takes place and try again. Next time around, Automounter chooses one of the other, available servers. Another option is to use the **umount** command, inform Automounter of the change in the mount table (as specified in the later section, “Refreshing the mount table,” and retry the mount.

CAUTION Care should be taken when using multiple locations for read/write file hierarchies, because the actual server may change from access to access.

Specifying subdirectories

The earlier subsection “Writing an indirect map” showed the following typical *auto.home* file:

```
willow                willow:/home/willow
cypress               cypress:/home/cypress
poplar                poplar:/home/poplar
pine                  pine:/export/pine
apple                  apple:/export/home
ivy                    ivy:/home/ivy
peach                 peach:/export/home
- rw, nosuid
```

Given this *auto.home* indirect file, every time a user wants to access a home directory in, for example, */home/willow*, all the directories under it are mounted. Another way to organize an *auto.home* file is by user name:

```
john                  willow:/home/willow/john
mary                  willow:/home/willow/mary
joe                   willow:/home/willow/joe
```

The above example assumes that home directories are of the form */home/user* rather than */home/server/user*. If a user now enters the following command:

```
ls ~john ~mary
```

Automounter has to perform the equivalent of the following actions:

```
mkdir /tmp_mnt/home/john
mount -f NFS willow:/home/willow/john /tmp_mnt/home/john
ln -s /tmp_mnt/home/john /home/john

mkdir /tmp_mnt/home/mary
mount -f NFS willow:/home/willow/mary /tmp_mnt/home/mary
ln -s /tmp_mnt/home/mary /home/mary
```

It is possible to optimize the work done by Automounter by modifying the entries in the *auto.home* map to use the optional “subdirectory” field of the *location*. The new *auto.home* map looks like:

```
john                  willow:/home/willow:john
mary                  willow:/home/willow:mary
joe                   willow:/home/willow:joe
```

Here “john,” “mary,” and “joe” are entries in the “subdirectory” field. Now, when a user refers to *john*’s home directory, Automounter mounts *willow:/home/willow* and links */tmp_mnt/home/willow/john* and */home/john*.

If the user then requests access to *mary*'s home directory, Automounter sees that *willow:/home/willow* is already mounted, and simply links */tmp_mnt/home/willow/mary* and */home/mary*. In other words, Automounter now only performs the equivalent of the following:

```
mkdir /tmp_mnt/home/john
mount -f NFS willow:/home/willow /tmp_mnt/home
ln -s /tmp_mnt/home/john /home/john
ln -s /tmp_mnt/home/mary /home/mary
```

In general, it is a good idea to provide a *subdirectory* entry in the *location* when different map entries refer to the same mounted filesystem from the same server.

Metacharacters

Automounter recognizes some characters as having a special meaning. Some are used for substitutions, some to escape other characters.

Ampersand (&)

Suppose you have a map with a lot of subdirectories specified, for example:

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

[. . .]

Consider using string substitutions. You can use the ampersand character (&) to substitute the key wherever it appears.

Using the ampersand, the above map now looks as follows:

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

[. . .]

Suppose the name of the server is the same as the key itself, for instance:

willow	willow:/home/willow
peach	peach:/home/peach
pine	pine:/home/pine
oak	oak:/home/oak
poplar	poplar:/home/poplar

[. . .]

The use of the ampersand results in:

```
willow          & : /home/&
peach           & : /home/&
pine            & : /home/&
oak             & : /home/&
poplar          & : /home/&
                [. . .]
```

Asterisk (*)

Notice that all the above entries have the same format. This permits you to use the catch-all substitute character, the asterisk (*). The asterisk reduces the whole thing to:

```
*              & : /home/&
```

Here each ampersand is substituted by the value of any given key. Notice that once Automounter reads the catch-all key it does not continue reading the map, so that the following map is viable:

```
oak            & : /export/&
poplar         & : /export/&
*              & : /home/&
```

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

```
*              & : /home/&
oak            & : /export/&
poplar         & : /export/&
```

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

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

This is a good candidate to be written as:

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

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

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

This is because Automounter interprets it as follows:

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

Backslash (\)

Under certain circumstances, you may have to mount directories whose names may confuse Automounter's map parser. This is a concern only with non-UNIX system servers. An example might be a directory called *rc0:dk1*; this could result in an entry like:

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

The presence of the two colons in the location field confuses 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 (")

You can also use double quotes, as in the following example, where they hide the blank space in the name:

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

Environment variables

Environment variables can be used by prefixing a dollar sign (\$) to the name of the variable. Braces are used to delimit the name of the environment variable from appended letters or digits. These variables can be used anywhere in an entry line, except as a *key*.

Environment variables can either 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, create a specific map for each client according to its name, so that the relevant line for host *oak* is:

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

For *willow* it is:

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

This scheme is viable within a small network, but maintaining this kind of host-specific map across a large network would soon become unfeasible. The solution in this case is to start Automounter with a command line similar to the following:

```
automount -D HOST='hostname' ...
```

The entry in the direct map should read:

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

Now each host finds its own files in the *mystuff* directory, and the task of centrally administering and distributing the maps becomes easier.

Including other maps

A line of the form *+mapname* causes Automounter to consult the mentioned map as if it were included in the current map. If *mapname* is a relative pathname (no slashes), Automounter assumes it is an NIS map. If the pathname is an absolute pathname, Automounter looks for a local map of that name. If the mapname starts with a dash "-", Automounter consults the appropriate built-in map (a special map that NIS creates by default; see the "Mount point net" section in this chapter).

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:

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

After consulting the included map, Automounter continues scanning the current map if no match is found, so you can add more entries, for instance:

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

Finally, as mentioned before, the map included 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
```

Starting Automounter

Before starting Automounter, make sure that:

- there are no equivalent entries in */etc/default/filesys*
- all the entries in the maps refer to NFS exported-files

The syntax to invoke Automounter is:

```
automount
    [ -mnTv ]
    [ -D name = value ]
    [ -f master-file ]
    [ -M mount-directory ]
    [ -tl duration ]
    [ -tm interval ]
    [ -tw interval ]
    [ directory map [ -mount-options ] ] ...
```

The **automount(NADM)** manual page contains a complete description of all options. The *mount-options* that you can specify on the command line or in the maps are the same as those for a standard NFS **mount**, excluding **bg** (background) and **fg** (foreground), which do not apply.

By default, the NFS startup script, */etc/nfs*, starts Automounter at boot time through the lines:

```
echo "automount\c"
automount -m -f /etc/auto.master
```

The **-m** option instructs Automounter not to look for the NIS map; the **-f** option instructs it to use the local file, */etc/auto.master*, as the master map.

To use the NIS *auto.master* map, start **automount** without the **-m** option:

```
automount
```

If the map is found, Automounter follows the directives contained within. If NIS is not running or the map is not found, Automounter exits silently.

It is also possible to specify master map information, that is, mount-points, map-names, and mount-options, on the **automount** command line itself:

```
automount /net -hosts /home /etc/auto.home -rw /etc/auto.direct -ro
```

This is equivalent to starting **automount** with no options and the following */etc/auto.master* map:

```
/net          -hosts
/home         /etc/auto.home          -rw
/-           /etc/auto.direct         -ro
```

Other combinations of **automount** arguments can change the name of the master map, and add, nullify, or override master map entries. Again, see the **automount(NADM)** manual page for more details.

Administering Automounter

The following sections describe procedures for Automounter maintenance and debugging.

Restarting Automounter

Every time Automounter mounts or unmounts a file hierarchy, it modifies */etc/mnttab* to reflect the current situation. Automounter keeps an image in memory of */etc/mnttab*, and refreshes this image every time it performs a mounting or an automatic unmounting. If you use the **umount** command to unmount one of the automounted hierarchies (a directory under */tmp/mnt*), you must force Automounter to re-read the */etc/mnttab* file. To do so, enter the command:

```
ps -ef | grep automount | egrep -v grep
```

This gives you the process ID of Automounter. When Automounter receives a SIGHUP signal, it re-reads */etc/mnttab*. To send it that signal, enter:

```
kill -1 pid
```

Here, *pid* stands for the process ID you obtained from the previous **ps** command.

Modifying the maps

You can modify Automounter maps at any time. However, that does not guarantee that all your modifications will take effect the next time Automounter mounts a hierarchy. It depends on what map you modify and what kind of modification you introduce.

NOTE Rebooting your machine is generally the simplest way of restarting Automounter.

Modifying the master map

Automounter consults the master map only at startup time. A modification to the master map will take effect only next time you reboot the machine.

Modifying indirect maps

Entries can be modified, deleted, or added to indirect maps and the change will take effect next time the map is used, which is next time a mount occurs.

Modifying direct maps

Each entry in a direct map is an automount mount point and it only mounts itself at these mount points at startup. Therefore, adding or deleting an entry in a direct map will only take effect next time you reboot the machine. However, existing entries can be modified (for example, changing mount options or server names, but not names of mount points) while Automounter is running, and will take effect when the entry is next mounted, because Automounter consults the direct maps whenever a mount has to be done.

For instance, suppose you modify the file, */etc/auto.direct*, so that the directory, */usr/src*, is now 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 the default unmounting takes place before you access it. If this is not satisfactory, you can unmount with the **umount** command; if you do, you must notify **automount** that the mount table has changed (see the “Restarting Automounter” section earlier in this chapter) before accessing the directory.

You can now mount the directory from the new server. However, if you wanted to delete the entry, you would have to reboot the machine for the deletion to take effect.

NOTE For this reason, and because they do not clutter the mount table like direct maps do, indirect maps are preferable and should be used whenever possible.

Mount point conflicts

If you have a home partition on a local disk that is mounted on */home* and you want also to use Automounter to mount other home directories, you may encounter a mount point conflict. That is, if you give it the mount point */home*, then Automounter will hide the local home partition whenever you try to reach it.

The solution is to mount the partition somewhere else, for example, on */export/home*. You would then need an entry in */etc/default/filesys* that says:

```
/dev/xx#z      ...      /export/home      ...
```

Here *xx#z* stands for the name of the partition, assuming that the master file contains a line similar to:

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

An entry is also needed in the *auto.home* map similar to the following (where *terra* is the name of the machine):

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

If the partition is set up such that home directories are located on */home/machine/user*, move all the directories at the *user* level one level up, so as to eliminate the *machine* level:

```
cd /home
mv machine/* .
rmdir machine
```

You need not change the */etc/passwd* entry for users. Their home directories are still accessible through */home/machine/user*, as before. Instead of doing a mount, Automounter recognizes that the filesystem is on the same machine and establishes a symbolic link from */home/machine* to */export/home*.

Unexpected filesystem behavior

Automounter's use of symbolic links may produce unexpected effects. For example, Automounter's method of mounting a filesystem onto the pseudo-staging area */tmp_mnt/...* causes undesired results when changing directories using *".."* in a pathname. If the user issues the command *cd ..* from the mount point root, they end up in */tmp_mnt*, not in the desired parent directory. Similarly, *pwd* also shows the staging area.

This does not affect the Korn shell, which keeps its own copy of the absolute pathname.

Increasing mount table entries

In a system with a high degree of Automounter activity, you must ensure that your filesystem kernel parameters are configured adequately for the volume of mount requests. If you are managing more than eight mounts (including default, NFS and Automounter mounts), you should probably increase the number of entries allocated in the mount table, */etc/mnttab*. When the maximum number of entries is exceeded, Automounter fails and the desired filesystems are not mounted as expected.

You can increase the number of mount table entries by tuning the kernel parameter *NMOUNT*. To do so, follow these steps:

1. After informing users to log off, reboot your system and enter system maintenance mode.
2. Change directories (*cd*) to */etc/conf/cf.d*.

3. Enter the following command to determine the current value of the NMOUNT kernel parameter (the default is 8):

```
./configure -y NMOUNT
```

4. Change the value of NMOUNT with the following command:

```
./configure NMOUNT=new_value
```

The value should be large enough to accommodate your largest volume of mount requests; however, you should keep the value as low as possible to avoid unnecessary search time. You will probably have to experiment to find an optimal value.

5. You must link the new parameter into the kernel. To do so, enter the following command:

```
link_unix
```

Once the kernel has been relinked, you must reboot your system.

For more information, consult the **configure(ADM)** manual page and the “Tuning system performance” chapter of the *SCO UNIX Operating System System Administrator’s Guide*.

Debugging Automounter

The following sections describe general error messages and messages generated when running the verbose option (**automount -v**). You should run the verbose option if Automounter fails to start properly.

All Automounter error messages are prefixed by the string “automount: ”.

General error messages

pathok: couldn’t find devid *device id*

This is an internal Automounter error.

WARNING: default option '*option*' ignored for map *mapname*

Here *option* is an unrecognized default mount option for the map *mapname*.

WARNING: *option* ignored for *key* in *mapname*

Automounter detected an unknown mount option. This is advisory only. Correct the entry in the appropriate map.

bad entry in map *mapname* "*key*"

map *mapname*, key *key*: bad

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

can't get my address

Automounter cannot find an entry for its host in */etc/hosts*(or *hosts.byname*).

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*

Here *pathname* is */tmp_mnt* or the argument to the -M command line option.

can't mount *mountpoint: reason*

Automounter could not mount its daemon at *mountpoint*.

exiting

This is an advisory message only. Automounter has received a SIGTERM (has been killed) and is exiting.

server:pathname no longer mounted

Automounter is acknowledging that *server:pathname*, which it mounted earlier, has been unmounted by the *umount* command. 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.

host *server* not responding

Automounter attempted to contact *server* but received no response.

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

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

hierarchical mountpoints: *pathname1* and *pathname2*

Automounter does not allow its mount points to have a hierarchical relationship; that is, an Automounter mount point must not be contained within another automounted filesystem.

mountpoint: Not a directory

Automounter cannot mount itself on *mountpoint* because it is not a directory. Check the spelling and pathname of the mount point.

dir *mountpoint* must start with "/"

The Automounter mount point must be given as the full pathname. Check the spelling and pathname of the mount point.

mapname: yp_err

There is an error in looking up an entry in an NIS map. It may indicate NIS problems.

hostname : exports: rpc_err

There is an error getting export list from *hostname*. This indicates a server or network problem.

nfscast: Cannot send packet: **reason**

Automounter cannot send a query packet to a server in a list of replicated filesystem locations.

nfscast: cannot receive reply: **reason**

Automounter cannot receive replies from any of the servers in a list of replicated filesystem locations.

nfscast:select: **reason**

Cannot create socket for nfs: rpc_err

These error messages indicate problems attempting to reach servers for a replicated filesystem using the **ping** command. This may indicate a network problem.

Error messages generated by the verbose option

no mount maps specified

Automounter was invoked with no maps to serve, and it cannot find the NIS *auto.master* map. It exits. Recheck the command, or restart NIS if that was the intention.

mapname:not found

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

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

Automounter discovered an entry in a map that contains leading spaces. This is usually an indication of an improperly continued map entry. For example:

```
foo
                                /bar                frobz:/usr/frotz
```

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

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

While scanning an indirect map, Automounter found an entry key containing a **" / "**. Indirect map keys must be simple names, not pathnames.

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

While scanning a direct map, Automounter found an entry key without a prepended `"/"`. Keys in direct maps must be full pathnames.

NIS bind failed

Automounter cannot communicate with the ypbind daemon. This is information only. 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 mntpnt **mountpoint: reason**

Automounter cannot create a mount point required for a mount. This usually occurs when attempting to mount hierarchically all of a server's exported filesystems. A required mount point may exist only in a filesystem that cannot be mounted (it may not be exported) and it cannot be created because the exported parent filesystem is exported read-only.

WARNING: **mountpoint** already mounted on

Automounter is attempting to mount over an existing mount point. This indicates an Automounter internal error.

server:pathname already mounted on **mountpoint**

Automounter is attempting to mount over a previous mount of the same filesystem. This could happen if an entry appears both in `/etc/default/filesys` and in an Automounter map (either by accident or because the output of `mount -p` was redirected to `default/filesys`). 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 export table on **server**.

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

Automounter failed to remount a filesystem it previously unmounted. This message may appear at intervals until the filesystem is successfully remounted

WARNING: **mountpoint** not empty!

The mount point is not an empty directory. The directory **mountpoint** contains entries that are hidden while Automounter is mounted there. This is advisory only.

Index

Special characters

/-, 80
 & (ampersand), 88
 * (asterisk), 89
 + (plus) symbol, Automounter maps, 91
 " (double quotes), 90
 \ (backslash), 90

A

Account, managing, 14
 Adding
 NIS servers, 58
 user, 15, 35, 54-56
 Administration, centralizing, 11
 Append mode, 27
 Architecture, 6
 Atomic write, 27
 authck(ADM), 62
 auto.master. *See* Maps, Automounter, master
 Automounter
 restarting, 93
 configuring, 78-79
 debugging, 96
 defined, 1, 73
 multiple servers, 86
 planning, 10, 78
 starting, 78, 91
 using with NFS, 10, 73, 78-79
 using with NIS, 11, 77-78, 83, 92, 99
 automount(NADM), 73, 79, 92

B

bdev, filesystems(F) entry, 18
 bind, NIS, 99
 Binding, NIS. *See* ypbind(NADM).
 biod(NADM), 14, 21-22, 26

C

Changing, NIS master server, 60
 Client
 definition, 6
 NFS, 17, 19, 26
 Clock, NFS problems, 28
 Command line, starting Automounter, 78-79, 92
 Configuring
 Automounter, 79
 NFS, 15
 NIS, 39
 Conventions
 documentation, 2
 notational, 2
 Copy-only server, NIS, 33
 cron(C), using with NIS, 50

D

Daemons
 Automounter, 78, 99
 NFS, 14, 21, 25
 NIS, 34
 Debugging
 Automounter, 96
 NFS, 20, 23, 26
 NIS, 62
 Default, mounting using /etc/default/filesys, 17
 /dev/nfsd, 24-25
 Distributed computing environment,
 planning, 9
 Distributed network, 5
 Distributed user account, NIS, 35, 54-57
 Documentation conventions, 2
 Domain, NIS, 33, 37, 39
 domainname(NC), 33, 39, 65, 69

E

Enabling, export, 15
Environment variable, automount(NADM), 90
Environment variables, 40
Error messages
 Automounter, 98
 NFS, 24-26
 NIS, 64, 66
/etc/auto.master. *See* Maps, Automounter, master
/etc/default/filesys, 14, 17, 19, 23, 91, 99
/etc/exports, 13, 15
/etc/group, 36
/etc/hosts, 13, 16, 24-25, 39, 80, 97
/etc/mnttab, 14, 23-24, 93
/etc/nfs, 15, 47
/etc/passwd, 35-36
/etc/passwd.local, 35, 57-58
/etc/passwd.yp, 35, 57-58
/etc/yp, 32
/etc/yp directory, 40
/etc/yp/Makefile, 48-49
/etc/ypservers, 39
/etc/yp/ypmake, 48, 60
/etc/yp/YP_MAP_X_LATE, 36, 39, 53
exportfs(NADM), 16
Exporting, filesystem, 13, 15
exports(NF), 21

F

fhandle, 21, 23
File
 handle, 21, 23
 locking, 27
 NFS, 27
fileys(F), 21
Filesystem
 default mounting, 17
 exporting, 15
 mounting, 22-23
 remote, 27-28
 user mounting, 19
Filesystems, naming conventions, 10
fsck, fileys(F) entry, 18
fstyp, fileys(F) entry, 18

H

Hard mount, 17, 21
/home, 80, 82
hosts.byname map, 80, 97
hosts(NF), 16

I

Initializing, NIS, 40
integrity(ADM), 62
Interoperability, NIS, 67

K

Kernel, mounting filesystems, 21
Kernel parameters, modifying, 46
key, 82, 84, 96, 98

L

Local, user account, NIS, 35, 54, 56-57

M

make(CP), 48-49, 60
makedbm(NADM), 48, 51-53, 58-60, 69
Managing, user accounts, 14
map2ascii(NADM), 51, 67
Maps, Automounter
 direct, 77, 84, 94
 indirect, 77, 82, 93
 master, 77, 79, 93
Maps, NIS
 Automounter, 79
 creating, 51
 defined, 32
 different versions, 62
 hosts.byname, 80, 97
 managing, 48
 modifying, 49
 propagating, 33, 50, 53
Master server, NIS, 33
Metacharacters, Automounter, 88
mkdev nis, 39-40
mnt(C), 19
mnttab(NF), 21
mount

mount (*continued*)
 error message, 24
 filesys(F) entry, 18
 table, 21
 Mount table, 14, 23-24, 93
 mount(ADM)
 directory, 14
 remote filesystem, 17
 mountdir, filesys(F) entry, 18
 mountd(NADM), 14, 21
 Mounting, Automounter
 direct, 77
 hierarchical mounts, 85, 97, 99
 included maps, 91
 indirect, 77-78
 mount options, 79, 82, 84, 92, 96
 mount points, 79-80, 82, 84, 92, 94, 97, 99
 multiple mounts, 84
 Mounting, NFS
 directory, 14
 hard, 17, 21
 mount points, 17, 19
 remote filesystem, 17, 21-23
 soft, 17, 21
 mount(NADM), 14, 17, 21, 23, 73, 79
 mount(NS), 21
 Multiple locations, 86

N

Naming conventions, filesystems, 10
 /net, 80
 Network
 architecture, 6
 background, 5
 distributed, 5
 operation, 5
 protocol, 6
 NFS
 adding user, 15
 background, 5
 configuring, 15
 daemons, 14
 debugging, 20, 26
 definition, 1, 5
 exporting a filesystem, 15
 features, 8
 files, 13
 handling remote filesystems, 27-28

NFS (*continued*)
 planning, 9
 programs hung, 26
 protocol, 4
 setting up clients, 17
 setting up servers, 15
 starting, 20
 stopping, 20
 using with Automounter, 10, 73, 78-79
 using with NIS, 11, 31
 nfscast, error message, 98
 nfscInt(NADM), 22
 nfsd(NADM), 14, 21, 78
 nfs_ghet, system call, 23
 nfsops, filesys(F) entry, 18
 nfsstat(NADM), 27
 NIS
 administering, 58
 bind, 99
 configuring, 39
 daemons, 34
 debugging, 62
 defined, 1, 31
 domains, 33, 37, 39
 initializing, 40
 interoperability, 67
 planning, 11, 37
 security, 36, 38, 62, 68
 setting up servers, 39
 starting, 47
 user account, 35, 54-58
 using with Automounter, 11, 77-78, 83, 92, 99
 using with NFS, 11, 31
 utilities, 35, 48, 54, 69
 Notational conventions, 2

P

passmgmt(NADM), 35, 54, 56, 69
 pathok, error message, 96
 pcnfsd(NADM), 14
 Planning, configuration, 9, 37, 78
 portmap(NADM), 63, 66
 Portmapper, 23-25
 Propagating NIS maps, 33, 50, 53
 Protocol, network, 6
 Protocol, NFS, 4

R

rcfsck, `filesystems(F)` entry, 18
rmount, `filesystems(F)` entry, 18
Reference works, 4
Remote
 filesystem mounting, 17
 mount failed, 23
remount, error message, 99
Removing
 NIS servers, 59
 user accounts, 58
rmtab(NF), 21
rpcinfo(NADM), 21-22, 25

S

Security, NIS, 36, 38, 62, 68
Server
 definition, 6
 NFS, 15-16, 26, 97
 NIS, 33, 38-39, 58-60
showmount(NADM), 21, 25
Slave server, NIS, 33
Soft mount, 17, 21
Starting
 NFS, 20
 NIS, 47
statfs(S), system call, 23
Stopping, NFS, 20
Symbolic links, 73, 77, 95
sysadmsh(ADM), creating accounts, 35, 54-55

T

TCP/IP, 7
/tmp_mnt, 73, 81, 95
trymany, error message, 97

U

umnt(C), 19
umount(NADM), 14, 93
User
 accounts, managing, 14
 adding, 15, 35, 54-56
 modifying accounts, 57
 removing accounts, 58
User ID, selecting, 14

Utilities, NIS, 35, 48, 54, 69

Y

Yellow Pages. *See* NIS, defined
ypbind(NADM), 34, 64, 66, 69
ypcat(NC), 58-59, 69
ypclnt(NS), 69
ypinit(NADM), 33, 39-40, 48-49, 70
ypmake(NADM), 48-49, 55, 60, 70
ypmapxlate(NF), 36, 39, 70
ypmatch(NC), 70
yppassmgmt(NADM), 35, 54-55, 57, 70
yppasswdd(NADM), 34, 68, 70
yppasswd(NC), 68, 70
yppoll(NADM), 70
yppush(NADM), 51, 71
ypservers, NIS map, 51
ypserv(NADM), 34, 51, 63-64, 71
ypset(NADM), 71
ypwhich(NADM), 65, 71
ypxfrd(NADM), 34, 71
ypxfr(NADM), 50-51, 53, 62, 71



Please help us to write computer manuals that meet your needs by completing this form. Please post the completed form to the Publications Manager nearest you: The Santa Cruz Operation, Ltd., Croxley Centre, Hatters Lane, Watford WD1 8YN, United Kingdom; The Santa Cruz Operation, Inc., 400 Encinal Street, P.O. Box 1900, Santa Cruz, California 95061, USA or SCO Canada, Inc., 130 Bloor Street West, 10th Floor, Toronto, Ontario, Canada M5S 1N5.

Volume title: _____
(Copy this from the title page of the manual, for example, SCO UNIX Operating System User's Guide)

Product: _____
(for example, SCO UNIX System V Release 3.2 Operating System Version 4.0)

How long have you used this product?

- ☐ Less than one month ☐ Less than six months ☐ Less than one year
☐ 1 to 2 years ☐ More than 2 years

How much have you read of this manual?

- ☐ Entire manual ☐ Specific chapters ☐ Used only for reference

	Agree		Disagree		
The software was fully and accurately described	☐	☐	☐	☐	☐
The manual was well organized	☐	☐	☐	☐	☐
The writing was at an appropriate technical level (neither too complicated nor too simple)	☐	☐	☐	☐	☐
It was easy to find the information I was looking for	☐	☐	☐	☐	☐
Examples were clear and easy to follow	☐	☐	☐	☐	☐
Illustrations added to my understanding of the software	☐	☐	☐	☐	☐
I liked the page design of the manual	☐	☐	☐	☐	☐

If you have specific comments or if you have found specific inaccuracies, please report these on the back of this form or on a separate sheet of paper. In the case of inaccuracies, please list the relevant page number.

May we contact you further about how to improve SCO UNIX documentation?
If so, please supply the following details:

Name _____ Position _____

Company _____

Address _____

City & Post/Zip Code _____

Country _____

Telephone _____ Facsimile _____

24 June 1992



BH02601P000

69614



AU02600P001