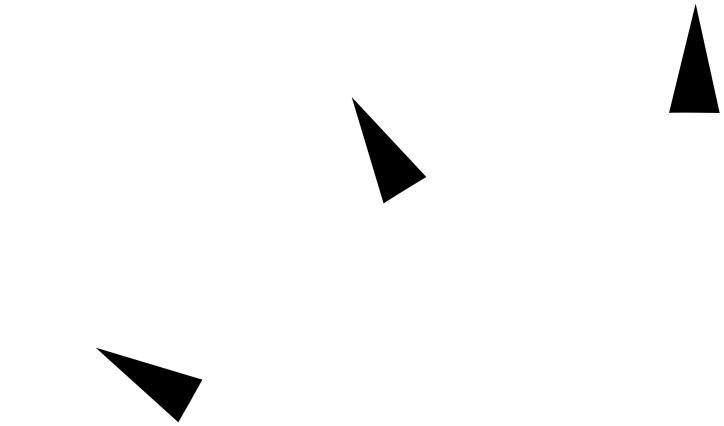
NFS Administration Guide



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

Contents

Figures

Tables

Preface

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

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

This book is organized into explanatory background material, task-oriented instructions, and statistical reference information.

Who Should Use This Book

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

How This Book Is Organized

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

Chapter 2, "How To Set Up NFS Servers," provides information on how to setup NFS servers. It assumes you are using NIS or NIS+ as your name service.

Chapter 3, "How To Use the NFS Environment," describes how to use NFS file systems.

Chapter 4, "Setting Up and Maintaining NFS Security," presents background information on the security features of NFS, as well as fundamental procedures for setting up and maintaining NFS security.

Chapter 5, "NFS Troubleshooting," describes problems that may occur on machines using NFS services. It contains a summary of NFS sequence of events and procedures for tracking NFS problems. Background and reference sections are also included.

Chapter 6, "Using Autofs," provides procedures for setting up and using autofs. It also includes background, reference, and troubleshooting sections.

Typographic Conventions

Table P-1 describes the typographic conventions used in this book.

Table P-1 Typeface and Symbol Meanings

Typeface or Symbol	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `system% You have mail`
`AaBbCc123`	What you type, contrasted with on-screen computer output	`system%` **`su`** `password`:
AaBbCc123	Command-line placeholder: replace with a real name or value	To delete a file, type `rm` *filename.*
AaBbCc123	Book titles, new words or terms, or words to be emphasized	Read Chapter 6 in *User's Guide.* These are called *class* options. You *must* be root to do this.
%	UNIX® C shell prompt	`system%`
$	UNIX Bourne shell prompt	`system$`
#	Superuser prompt, either shell	`system#`

Solaris NFS Environment 1

This chapter provides an overview of the Solaris network file system (NFS) environment.

The NFS system allows you to share files and directories among many computers on a network. For example, applications running simultaneously on several computers can read from and write to a single file system. To participating users, the file system appears to be located on their local systems. The possible uses for shared file systems are endless.

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

1

NFS System

The NFS system is a SunSoft distributed computing file system that can be used to link computers that are running different operating systems. For example, computers running DOS can share files with computers running UNIX.

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

A computer becomes an NFS server if it has file systems to *export* over the network. A server keeps a list of currently exported file systems and their access restrictions (read / write, read-only, and so on).

Autofs

File systems shared through NFS software can be mounted using automatic mounting. Autofs, a client-side service, is a file system that provides advanced automatic mounting. The `automount` program runs in the background mounting and unmounting remote directories on an as-needed basis.

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

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

Mounting some file hierarchies with `automount` does not exclude the possibility of mounting others with `mount`. A diskless computer *must* mount `/` (root), `/usr`, and `/usr/kvm` through the `mount` and the `/etc/vfstab` file. Do *not* use autofs to mount `/usr/share`. `Automountd` depends on some files in that directory.

Autofs Features

Autofs works with file systems specified in NIS *maps* or NIS+ *tables.* These maps or tables can be maintained as NIS, NIS+, or local files.

Autofs maps or tables can specify several remote locations for a particular file. This way, if one of the servers is down, autofs can try to mount from another computer. To specify which servers are preferred for each file system in the maps, you can assign each server a weighting factor.

You can invoke autofs from a shell command line, or it will start automatically when the computer enters run level 2.

1

How To Set Up NFS Servers 2

This chapter provides information on how to set up NFS servers. It assumes you are using NIS or NIS+ as your name service. The chapter is organized into overview, how-to, and reference sections.

If you would rather review background information first, read the following section, “About the NFS Environment.”

About the NFS Environment

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

The NFS environment makes it possible for a computer to share local files and directories, and permits remote users to access those files and directories as though they were local to the user’s computer.

2

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

The benefits of NFS software are as follows:

- Allows multiple computers to use the same files, so the same data can be accessed by everyone on the network
- Reduces storage costs by having computers share applications
- provides data consistency and reliability as all users can read the same set of files
- Mounting of file systems transparent to users
- Accessing remote files is transparent to users
- Supports heterogeneous environments
- Reduces system administration overhead

NFS File Systems

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

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

NFS Servers and Clients

A computer that makes a local file system available for mounting by remote computers is called a server. A computer that mounts a file system shared by a remote computer is a client of that computer. Any computer with a disk can be server, a client, or both at the same time.

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

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

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

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

Each line in the file consists of a `share` command—the same command you enter at the `share(1M)`, in the `dfstab` file command line to share a file system explicitly. The `share` command is located in `/usr/sbin`.

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

Once a file system has been shared on a server through NFS operation, it can be accessed from a client. NFS file systems are mounted automatically with autofs and name service maps (NIS and NIS+).

NFS Administration Tasks

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

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

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

- Sharing and unsharing file systems as necessary
- Modifying administrative files to update the lists of file systems your computer shares and/or mounts automatically
- Checking the status of the network. (Refer to Chapter 5, "NFS Troubleshooting")
- Diagnosing and fixing NFS related problems as they arise
- Setting up maps to use the automatic mounting facility called autofs (See Chapter 6, "Using Autofs")

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

Setting Up Automatic Sharing

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

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

   where `-F nfs` indicates that the file system is to be shared through NFS software (this is also the default); *specific-options* is a comma-separated list of options that regulates how the file system is shared; *description* is a comment that describes the file system to be shared; and *pathname* is the full name of the file system to be shared, starting at root (/).

```
share -F nfs -o ro,rw=homedog:chester /usr/src
```

In the previous example, read-only access is assigned to any client except homedog and chester, who have read/write access.

```
share -F nfs -o rw=engineering,ro=homedog /usr/src
```

In the previous example, read/write is assigned to any client in the engineering netgroup. The client homedog has read-only access. For more information about netgroups, see *User Accounts, Printers, and Mail Administration.*

Specific options that can follow the -o flag include:

rw which shares *pathname* read/write to all clients (by default) except those that are specified under ro=.

ro which shares *pathname* read-only to all clients, except those that
b are specified under rw=.

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

---

ro=*client*[:*client*] which shares *pathname* read-only to the listed client computers or netgroup names (overriding rw).

rw=*client*[:*client*] which shares *pathname* read/write to the listed client computers (overriding ro).

anon=*uid* which allows you to specify a different *uid* for "anonymous" users—users whose *uid* is 0—when accessing *pathname*. By default, anonymous users are mapped to username nobody, which has the UID 60001. User nobody has ordinary user privileges, not superuser privileges.

root=*host*[:*host*] which allows a user from host *host* whose *uid* is 0 to access *pathname* as root; root users from all other hosts become anon. If this option is not specified, no user from any host is granted access to *pathname* as root.

Caution – Granting root access to other hosts has far-reaching security implications; use the root= option with extreme caution. See the following discussion for more information.

In the NFS environment, a server shares file systems it owns so clients can mount them using autofs. However, a user who becomes the superuser at a client is denied access as the superuser to NFS file systems. When a user logged in as `root` on one host requests access to a remote file shared through NFS software, the user's ID is changed from 0 to the user ID of the username `nobody`. The access rights of user `nobody` are the same as those given to the public for a particular file. For example, if the public has only execute permission for a file, then user `nobody` can execute only that file.

secure Allows you to share a file system with additional user authentication required

kerberos Allows you to share a file system with kerberos authentication (see *Security, Performance, and Accounting Administration*).

Examples of Automatic Sharing Entries in `/etc/dfs/dfstab`

You want to permit the root user on `samba` to always have root access to the `/usr/src` on the server computer. Make the following entry to the server's `dfstab` file.

```
share -F nfs -o root=samba /usr/src
```

You want to permit the root users on `samba`, `homedog`, and `chester` to always have root access to the `/usr/src` on the server computer. Make the following entry to the server's `dfstab` file.

```
share -F nfs -o root=samba:homedog:chester /usr/src
```

You want all client processes with UID 0 to have superuser access to `/usr/src`. You should make the following entry in the server's `dfstab` file.

```
share -F nfs -o anon=0 /usr/src
```

`anon` is short for "anonymous." Anonymous requests, by default, get their user ID changed from its previous value (whatever it may be) to the user ID of username nobody. NFS servers label as anonymous any request from a root user (someone whose current effective user ID is 0) who is not in the list

following the `root=` option in the `share` command. The previous command tells the kernel to use the value 0 for anonymous requests. The result is that all root users retain their user ID of 0.

You need to make sure that NFS software is running on the server, if this is the first share command or set of share commands that you have initiated.

1. **Run the server script** `/etc/init.d/nfs.server stop`**.**
2. **Run the server script** `/etc/init.d/nfs.server start.`
   This runs the necessary daemons `mountd` and `nfsd`.

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

At this point, set up your autofs maps so clients can access the file systems you've shared on the server.

Sharing Objects

If you need to share an object multiple times each share command replaces all previous shares of the specified file system(s). If you try to share the root file system to more than one machine using

```
share -F nfs  -o rw=<client>,root=<client>,anon=0 /
```

and then, wishing to add an additional client, you enter

```
share -F nfs  -o rw=<anotherclient>,root=<anotherclient>,anon=0 /
```

The second share will overrule the first share. Therefore you must include all the file systems you wish to share each time you add a file system

```
share -F nfs -o
      rw=<client>:<anotherclient>,root=<client>:<anotherclient>
      ,anon=0 /
```

Setting Up at Boot Time

If you want to mount file systems at boot time instead of using autofs maps, follow this procedure. This method is not recommended because it is very time consuming for a system administrator.

1. Edit the `/etc/vfstab` file.

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

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

Example of a `vfstab` *entry*

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

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

How To Use the NFS Environment 3

This chapter provides information on how to perform such NFS administration tasks as adding new file systems to share, unsharing file systems, displaying shared local file systems, and displaying mounted file systems. The NFS administration utilities package installs five commands: `share`, `unshare`, `mount`, `unmount`, and `showmount`.

NFS Commands

`share(1M)`

Allows you to add to a server a new file system to share. The `share` command is also used to set up your NFS servers. See Chapter 2, “How To Set Up NFS Servers.” You can also use the `share` command to display a list of the file systems on your system that are currently shared.

`unshare(1M)`

Allows you to make a previously available file system unavailable for mounting by clients.

`mount(1M)`

Used without arguments, this command allows you to mount a remote file system on your computer, or to display a list of file systems, both local and remote, that are currently mounted on your computer. You can also use this command for troubleshooting purposes. See Chapter 5, "NFS Troubleshooting," for more information. Autofs will automatically mount file systems for clients (users) as they request access to them.

`umount(1M)`

Allows you to remove a remote file system you previously mounted. Use this command for troubleshooting purposes only

`showmount(1M)`

Shows you which file systems are shared from an NFS server.

NFS Administrative Tasks

This section includes instructions for performing tasks related to managing file systems services with NFS software. After each set of instructions, there are examples of the screen input and output associated with the task, with all relevant assumptions defined and described.

Adding a New File System to Share

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

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

If you type the `share` command without an argument, the command displays all file systems shared on the server you are currently logged into that are currently shared. If you specify a file system type, the `share` command displays all file systems of the specified type that are currently shared.

The `/etc/dfs/dfstab` file allows you to share file systems automatically whenever your system enters run level 3. For example, if you want a directory to be available to clients on a regular basis, and you can anticipate few occasions when you would need to make it unavailable, type a `share` command for that directory into the `dfstab` file. Then, whenever you take the system to run level 3, the directory becomes available to clients automatically.

Each line of the file consists of the `share` command line needed to share a particular file system; the `share` command you type in the file has the same syntax as the `share` command you type at the command line.

Unsharing File Systems

The `unshare` command can be used to unshare any file system—whether the file system was shared explicitly with the `share` command or automatically through the `dfstab` file. If you use the `unshare` command to unshare a file system that you shared through the `dfstab` file, remember that it will be shared again when you exit and re-enter run level 3.

When you unshare an NFS file system, access from clients with existing mounts is inhibited.

Displaying Shared Local File Systems

- **Use the `share` command with or without specifying nfs.**

If you enter the command without arguments, it displays all NFS file systems on your system that are currently shared. If you specify a file system type and no other options, the command displays all file systems of the specified type that are currently shared.

3

Displaying Mounted File Systems

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

Code Example 3-1

```
corey(/home/bermudez/Encrypt): mount
/ on /dev/dsk/c0t3d0s0 read/write/setuid on Mon Nov 29 13:53:47
1993
/usr on /dev/dsk/c0t3d0s6 read/write/setuid on Mon Nov 29
13:53:47 1993
/proc on /proc read/write/setuid on Mon Nov 29 13:53:47 1993
/dev/fd on fd read/write/setuid on Mon Nov 29 13:53:47 1993
/tmp on swap read/write on Mon Nov 29 13:53:52 1993
/opt on /dev/dsk/c0t3d0s5 setuid/read/write on Mon Nov 29
13:53:54 1993
/usr/openwin on /dev/dsk/c0t3d0s7 setuid/read/write on Mon Nov 29
13:53:55 1993
/net/hostess/install on hostess:/install
corey(/home/bermudez/Encrypt):
```

`-o` *specific options*
which is a list of file system type specific options that can be specified after the `-o` flag. There are different options for NFS specifications (described later in this section).

The options that can follow the `-o` flag when mounting either NFS file system are:

`rw | ro`
where `rw` indicates that the file system is to be mounted read/write and `ro` indicates it is to be mounted read-only. (If no option is specified, `rw` is the default.)

`suid | nosuid`
where `suid` indicates that set-uid bits are to be obeyed on execution and `nosuid` indicates that they are to be ignored. (If no option is specified, set-uid is the default.)

Some NFS specific options that can follow the `-o` flag are:

`bg | fg`
where `bg` indicates that a mount retry should be initiated in the background when the server does not respond, and `fg` indicates it should be initiated in the foreground. If no option is specified, `fg` is the default. (This option does not apply to autofs.)

`nointr`
which disallows keyboard interruptions of NFS operations.

`retry=`*n*
which is the number of times to retry the mount operation. The default for *n* is 10,000 times.

`timeo=`*n*
which sets the timeout to *n* tenths of a second. If no option is specified, 1.1 seconds is the default.

3

Setting Up and Maintaining NFS Security 4

The NFS environment is a powerful and convenient way to share file systems on a network of different computer architectures and operating systems. However, the same features that make sharing file systems through NFS operation convenient also pose some security problems. An NFS server authenticates a file request by authenticating the computer making the request, but not the user. If superuser privilege is not restricted when a file system is shared, a client user can run `su` and impersonate the owner of a file.

Given root access and knowledge of network programming, anyone is capable of introducing arbitrary data into the network, and picking up any data from the network. The most dangerous attacks are those involving the introduction of data, such as impersonating a user by generating the right packets, or recording conversations and replaying them later. These attacks affect data integrity. Attacks involving passive eavesdropping—merely listening to network traffic without impersonating anybody—are not as dangerous, since data integrity is not compromised. Users can protect the privacy of sensitive information by encrypting data that goes over the network.

4

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

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

Secure RPC

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

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

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

When UNIX authentication is used by a network service, the credentials contain the client's computer-name, UID, `gid`, and group-access-list, but the verifier contains nothing. Because there is no verifier, a root user could deduce appropriate credentials, using commands such as `su`. Another problem with

UNIX authentication is that it assumes all computers on a network are UNIX computers. UNIX authentication breaks down when applied to other operating systems in a heterogeneous network.

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

DES Authentication

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

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

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

If a network runs a time synchronization program, then the time on the client and the server is synchronized automatically. If a time synchronization program is not available, timestamps can be computed using the server's time instead of the network time. The client asks the server for the time before starting the RPC session, then computes the time difference between its own clock and the server's. This difference is used to offset the client's clock when computing timestamps. If the client and server clocks get out of sync to the point where the server begins to reject the client's requests, the DES authentication system resynchronizes with the server.

The client and server arrive at the same encryption key by generating a random *conversation key*, also known as the session key, and then using public key cryptography (an encryption scheme involving public and secret keys) to

deduce a *common key.* The common key is a key that only the client and server are capable of deducing. The conversation key is used to encrypt and decrypt the client's timestamp; the common key is used to encrypt and decrypt the conversation key.

KERB Authentication

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

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

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

- Routines used by the client to create, acquire, and verify tickets
- An authentication option to Secure RPC
- A client-side daemon, `kerbd` (1M)

See *Security, Performance, and Accounting Administration* for more details.

AUTH_DES Client/Server Session

This section describes the series of transactions in a client/server session using AUTH_DES.

Step 1

Sometime prior to a transaction, the administrator runs a program, either `newkey` (`1M`) or `nisaddcred` (`1`) that generates a *public key* and a *secret key.* (Each user has a unique public key and secret key.) The public key is stored in a public database; the secret key is stored in encrypted form, in the same database. To change the key pair, use the `chkey` (`1`) command.

Step 2

The user logs in and runs the `keylogin` program (or the `keylogin` program may be included in the user's environment configuration file, such as ~/.login, ~/.cshrc, or ~/.profile, so that it runs automatically whenever the user logs in). The `keylogin` program prompts the user for a secure RPC, or network, password and uses the password to decrypt the secret key. The `keylogin` program then passes the decrypted secret key to a program called the *Keyserver*. (The Keyserver is an RPC service with a local instance on every computer.) The Keyserver saves the decrypted secret key, and waits for the user to initiate a secure RPC transaction with a server.

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

Step 3

When the user initiates a transaction with a server:

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

Step 4

The transmission including the encrypted timestamp and the encrypted conversation key is then sent to the server. The transmission includes a credential and a verifier. The credential contains threecomponents:

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

The window is the difference the client says should be allowed between the server's clock and the client's timestamp. If the difference between the server's clock and the timestamp is greater than the window, the server would reject the client's request.

The client's verifier contains:

- The encrypted timestamp
- An encrypted verifier of the specified window, incremented by 1

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

Step 5

When the server receives the transmission from the client:

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

Step 6

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

- The client's computer name
- The conversation key
- The window

- The client's timestamp

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

Note – Implicit in these procedures is the name of caller, who must be authenticated in some manner. The Keyserver cannot use DES authentication to do this because it would create a deadlock. To solve this problem, the Keyserver stores the secret keys by UID, and grants requests only to local root processes. The client process then executes a set-UID process, owned by root, which makes the request on the part of the client, telling the Keyserver the real UID of the client.

Step 7

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

- The index ID, which the server records in its credential cache.
- The client's timestamp minus one, encrypted by conversation key

The reason for subtracting one from the timestamp is to ensure that it is invalid and cannot be reused as a client verifier.

Step 8

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

Step 9

The client returns the index ID to the server in its second transaction and sends another encrypted timestamp.

Step 10

The server sends back the client's timestamp minus one, encrypted by the conversation key.

With every transaction after the first, the client sends its index ID and another encrypted timestamp, and the server returns the timestamp minus one.

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

- If a server crashes when no one is around (after a power failure for example), all of the secret keys that are stored on the system are wiped out. Now no process is able to access secure network services, or mount an NFS file system. The important processes at this time are usually root processes, so things would work if root's secret key were stored away, but nobody is around to type the password that decrypts it. `keylogin -r` allows `root` to store the clear secret key in `/etc/.rootkey` which `keyserve(1M)` reads.
- Some systems boot in single-user mode, with a `root` login shell on the console and no password prompt. Physical security is imperative in such cases.
- Diskless computer booting is not totally secure. Somebody could impersonate the boot server, and boot a devious kernel that, for example, makes a record of your secret key on a remote computer. Secure NFS provides protection only after the kernel and the Keyserver are running. Before that, there is no way to authenticate the replies given by the boot server. This is not a serious problem, because somebody would probably not be able to write this compromised kernel without source code. Also, the crime would have evidence. If you polled the network for boot-servers, you would discover the devious boot-server's location.
- Most set-UID programs are owned by root; because root's secret key is always stored at boot time, these programs behave as they always have. If a set-UID program is owned by a user, however, it may not always work. For example, if a set-UID program is owned by `dave`, and `dave` has not logged into the computer since it booted, then the program would not be able to access secure network services.
- If you log in to a remote computer (using `login`, `rlogin`, or `telnet`) and use `keylogin` to gain access, you give away access to your account. This is because your secret key gets passed to that computer's Keyserver, which then stores it. This is only a concern if you don't trust the remote computer. If you have doubts, however, don't log in to a remote computer if it requires a password. Instead, use the NFS environment to mount file systems shared by the remote computer. As an alternative, you can use `keylogout(1)` to delete the secret key from the Keyserver.

Administering Secure NFS

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

With UNIX authentication, the name of a domain is the UID. UIDs are assigned per domain. A problem with this scheme is that UIDs clash when domains are linked across the network. Another problem with UNIX authentication has to do with superusers; with UNIX authentication, the superuser ID (UID 0) is assigned one per computer, not one per domain. Therefore, a domain can have multiple superusers—all with the same UID.

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

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

Instructions for Administering Secure NFS

This section includes step-by-step instructions for performing tasks related to managing Secure NFS. After each set of instructions for many tasks, there is an example of the screen input and output associated with the task, with all relevant assumptions defined and described.

Setting Up Secure NFS

1. **Assign your domain a domain name, and make the domain name known to each computer in the domain. See the** *Name Services Administration Guide* **if you are using NIS+ as your name service.**
2. **Establish public keys and secret keys for your clients' users using the** `newkey(1M)` **command, and have each user establish his or her own secure RPC password using the** `chkey` **command.**

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

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

3. Usually, the login password is identical to the network password. In this case, keylogin is not required. If the passwords are different, the users have to log in, and then do a keylogin. You still need to use the `keylogin -r` command as root to store the decrypted secret key in `/etc/.rootkey`.
4. **If you are running NIS, verify that the** `ypbind` **daemon is running and that there is a** `ypserv` **running in the domain.**

```
ps -ef | grep ypbind
```

5. **To verify that the** `keyserv` **daemon (the Keyserver) is running, type the following:**

```
ps -ef | grep keyserv
```

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

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

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

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

(For KERB authentication, add the kerberos option.)

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

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

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

(For KERB authentication, add the kerberos option.)

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

Note – If a client does not mount as `secure` a file system that is shared as secure, users have access as user `nobody`, rather than as themselves.

When you reinstall, move, or upgrade a computer, remember to save `/etc/.rootkey` if you don't establish new keys or change them for root. If you do delete `/etc/.rootkey`, it's not fatal. You can always type `keylogin -r (1)`.

4

NFS Troubleshooting 5

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

Before trying to clear NFS problems, you should have some understanding of the issues involved. The information in this chapter contains enough technical details to give experienced network administrators a thorough picture of what is happening with their computers. If you do not yet have this level of expertise, you should be able to at least recognize the names and functions of the various daemons, system calls, and files. Before you read this chapter, familiarize yourself with the following manual pages: `mount(1M)`, `share(1M)`, `mountd(1M)`, and `nfsd(1M)`.

5

General Information on NFS Troubleshooting

When tracking down an NFS problem, keep in mind that there are three main points of possible failure: the server, the client, and the network itself. The strategy outlined in this section tries to isolate each individual component to find the one that is not working. The `mountd` daemon must be present in the server for a remote mount to succeed. Remote mounts also need an `nfsd` daemon to execute on NFS servers.

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

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

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

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

```
NFS server <hostname> not responding, still trying
```

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

```
NFS server <hostname> ok
```

A program accessing a soft-mounted file system whose server is not responding may not check the return conditions. In any case, the kernel prints the following message:

```
. . . <hostname> server not responding:RPC:Timed out
```

Note – Do not soft-mount read file systems with read-write data or file systems from which executables will be run because of possible errors. Writeable data could be corrupted if the application ignores soft errors Mounted executables could be misinterpreted..

NFS Troubleshooting Instructions

Determining Where NFS Service Has Failed

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

```
ps -ef | grep mountd
```

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

```
ps -ef | grep nfsd
```

3. **To enable daemons without rebooting, become root and type the following:**

```
/usr/lib/nfs/mountd
```

to enable `mountd` **type the following; or**

```
/usr/lib/nfs/nfsd -a 8
```

type the following to enable `nfsd`**.**

Clearing Server Problems

1. **Make sure that the server's kernel responds. From the client, type the following:**

```
ping <servername>
```

2. **Check that the server's `nfsd` processes are responding. From the client, type the following:**

```
/usr/sbin/rpcinfo -u <servername> nfs
```

   If the server is running, it prints a list of program, version, protocol, and port numbers.

3. **Check that the server's `mountd` is responding.**

```
/usr/sbin/rpcinfo -u <servername> mountd
```

   If the server is running normally, but your computer cannot communicate with it, check the network connections between your computer and the server.

Clearing Remote Mounting Problems

Example 1

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

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

1. **Sign on to the server and check its run level with the `who` command.**

```
who -r
```

2. **If the server is at run level 2, try going to another run level and back, or try rebooting the server to restart `rpcbind`.**
3. **Try to log in to the server from the client computer, using the `rlogin` command.**
4. **If you can't log in, but the server is up, try to log in to another remote computer to check your network connection. If that connection is working, check the server's network connection.**

Example 2

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

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

♦ **Determine whether or not the mount daemon is running:**

```
rpcinfo -u <server> mountd
```

Example 3

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

Either the remote directory or the local directory does not exist.

♦ **Check the spelling of the directory names. Use `ls` on both directories.**

Example 4

```
mount: ...: Permission denied
```

Your computer name may not be in the list of clients allowed access to the file system you want to mount.

1. To display the server's share list, type the following.

```
dfshares  <server>
```

or

```
showmount -e  <server>
```

2. If the file system you want is not in the list, log in to the server and run the `share` command without options.

Fixing Hung Programs

If programs hang while doing file-related work, your NFS server may be dead. You may see the following message on your console.:

```
NFS server <hostname> not responding, still trying
```

This message indicates that NFS server *hostname* is down, or that there is a problem with the server or with the network.

1. Check the server(s) from which you mounted the file system.

If one or more are down, do not be concerned. When the server comes back up, programs resume automatically. No files are destroyed.

If all servers are running, ask someone else using these same servers if they are having trouble. If more than one client computer is having problems getting service, there is a problem with the server.

If no other client computers are having trouble with the server, check your network connection and the connection of the server.

2. **Log in to the server and run `ps` to see if `nfsd` is running and accumulating CPU time.**

   (Run `ps -ef` a few times, letting some time pass between each call.)

3. **If `nfsd` is not running, you may be able to kill and then restart `nfsd`. If this does not work, reboot the server.**

   If `nfsd` is not running, it may be that the server has been taken to a run level that does not support file sharing. Use `who -r` to obtain the server's current run level.

For more information about NFS status, use the `snoop` command. See the man pages.

5

Using Autofs 6

This chapter tells you how to use autofs, a new implementation of automatic mounting. Autofs is a file system that mounts file systems as needed and unmounts file systems when they are not being used. It enables access to remote data when needed.

Autofs optimizes network applications' performance, and streamlines administrative tasks.

The first section of this chapter, "How Autofs Works" on page 40, tells you how to design and maintain map files.

The second section of this chapter, "Autofs Reference" on page 65, provides advanced information on how to configure maps to meet your specific environmental needs.

The third section, "Common Problems and Recommended Solutions" on page 70, describes some common scenarios for use of autofs and how to design autofs maps to best meet your needs for accessing file systems.

6

How Autofs Works

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

Three components that work together to accomplish automatic mounting are:

1. The `automount` command
2. The autofs file system
3. The `automountd` daemon

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

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

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

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

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

THis is a simplified overview of how autofs works:

The automount daemon `automountd` starts at boot time from the `/etc/init.d/autofs` script. This script also runs the `automount` command, which reads the master map (see "How Autofs Navigates Through the Network (Maps)" on page 45) and installs autofs mount points.

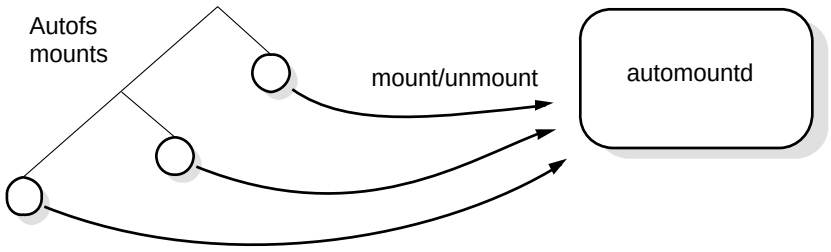


Figure 6-1 `/etc/init.d/autofs` Script Starts `automount`

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

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

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

6

Setting Up Autofs Maps

Autofs uses four types of maps:

- `auto_home`
- `auto_master`
- Direct maps
- Indirect maps

`auto_home` *Map*

The `auto_home` map associates user login names with their directory locations. It tells autofs to mount a home directory by pointing to the `auto_home` map. The map is a listing of all the users in the system, followed by the path to their home directory. To view the `auto_home` map, use Administration Tool. To bring up Administration Tool, type `admintool` at the command prompt.

`auto_master` *Map*

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

Direct Maps

A direct map is an automount point. With a direct map, there is a direct association between a mount point on the client and a directory on the server.

Indirect Maps

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

`autofs` *Command Syntax*

To invoke autofs, use the following syntax.

```
/usr/sbin/automount  [ –t mount-timeout ] [ -v ]
```

The `automount(1M)` man page contains a complete description of all options.

`automountd` *Command Syntax*

To invoke the `automountd(1M)` daemon, use the following syntax**.**

```
/usr/lib/autofs/automountd [ –Tv ] [ –D name=value ]
```

The `automountd(1M)` man page contains a complete description of all options.

Master Map Syntax

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

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

mount-point

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

map-name

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

[*mount-options*]

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

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

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

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

Direct and Indirect Map Syntax

Lines in direct and indirect maps have the following general syntax.

```
key  [ mount-options]    location
```

key

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

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

[*mount-options*]

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

location

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

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

How Autofs Navigates Through the Network (Maps)

Autofs searches a series of maps to navigate its way through the network. Maps are files that contain information such as the password entries of all users on a network, or the names of all host computers on a network; that is, network-wide equivalents of UNIX administration files. Maps are available locally or through a network name service like NIS or NIS+. You create maps to meet the needs of your environment using Administration Tool. See "Modifying How Autofs Navigates the Network (Modifying Maps)" on page 60.

There are four kinds of autofs maps:

- Auto_home—associates user login names with their directory locations
- Master—associates directories with maps
- Indirect—a directory containing automount points (used for most maps)
- Direct—an automount point (rarely used)

The following sections describe these maps.

How Autofs Starts the Navigation Process (Master Map)

Autofs reads the master map at system startup. Each entry in the master map is a direct or indirect map name, its path, and its mount options, as shown in Figure 6-2. The specific order of the entries is not important. Autofs compares entries in the master map with entries in the mount table.

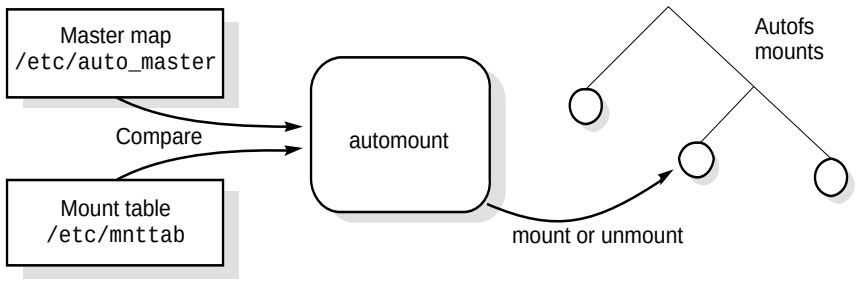


Figure 6-2 Master Map

For example, Table 6-1 shows what a typical `auto_master` file would contain:

Table 6-1 `auto_master` File Contents

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

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

Mount Point /–

In Table 6-1, the mount point `/-` tells autofs not to associate the entries in `auto_direct` with any directory. Indirect maps use mount points. Direct maps use mount points specified in the `auto_direct` map. (Remember, in a direct map the key, or mount point, is a full path name.)

A NIS or NIS+ `auto_master` can have only one direct map entry. An `auto_master` that is a local file can have any number of entries.

Mount Point /`home`

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

Mount point `/net`

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

```
cd /net/gumbo
```

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

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

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

1. `ping`s the null procedure of the server's mount service to see if it's alive
2. Requests the list of exported file systems from the server
3. Sorts the exported list according to length of path name

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

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

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

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

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

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

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

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

Replace it with the following entry:

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

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

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

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

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

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

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

Direct Maps

With a direct map, there is a direct association between a mount point on the client and a directory on the server. Direct maps have a full path name and indicate the relationship explicitly. Of all the maps, the entries in a direct map most closely resemble, in their simplest form, the corresponding entries in `/etc/vfstab` (`vfstab` contains a list of all file systems to be mounted). An entry that appears in `/etc/vfstab` as:

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

appears in a direct map as:

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

A typical `/etc/auto_direct` map is:

```
/usr/local              -ro\
    /bin                                ivy:/export/local/sun3 \
    /share                              ivy:/export/local/share\
    /src                                ivy:/export/local/src
/usr/man                -ro             oak,rose,willow:/usr/man
/usr/games              -ro             peach:/usr/games
/usr/frame              -ro             redwood:/usr/frame2.0 \
                                        balsa:/export/frame
```

There are a couple of important but previously unmentioned features in this map: *multiple locations* and *multiple mounts*, which are discussed in the next two sections.

How Autofs Finds Specific File Systems (Indirect Maps)

An indirect map is a directory containing automount points. Indirect maps are useful for accessing specific file systems, like the home directories. Figure 6-3 contrasts the indirect map structure with the direct map structure.

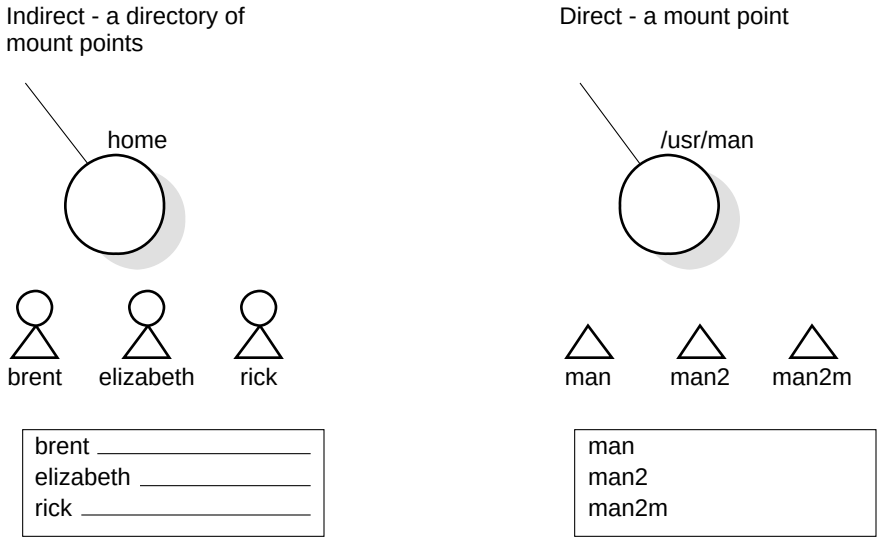


Figure 6-3 Indirect Map Structure Versus Direct Map Structure

Table 6-1 showed an `auto_master` map that contained the entry:

```
/home       auto_home
```

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

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

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

Note – Any option in the indirect map entry overrides all options in the master map.

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

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

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

```
cd ~david
```

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

On a network with NIS, you have to change all the relevant databases (such as `/etc/passwd`) on all systems on the network to accomplish this. Make the changes on the NIS master server and propagate the relevant databases to the slave servers. The `auto_home` map can be updated using Administration Tool. On a network running NIS+, propagating the relevant databases to the slave servers is done automatically.

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

In the example of a direct map, which was:

```
/usr/local              -ro \
    /bin                            ivy:/export/local/sun4\
    /share                          ivy:/export/local/share\
    /src                            ivy:/export/local/src
/usr/man                -ro         oak:/usr/man \
                                    rose:/usr/man \
                                    willow:/usr/man
/usr/games              -ro         peach:/usr/games
/usr/spool/news         -ro         pine:/usr/spool/news
/usr/frame              -ro         redwood:/usr/frame2.0 \
                                    balsa:/export/frame
```

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

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

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

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

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

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

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

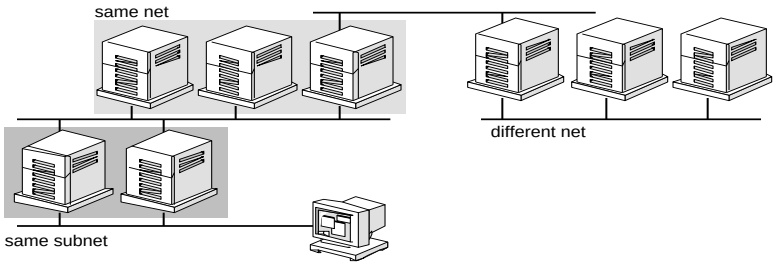


Figure 6-4 Server Proximity

This redundancy is used once at mount time to select one server from which to mount. Autofs does not check the status of the mounted-from server by autofs once the mount occurs. Multiple locations are very useful in an environment where individual servers may not be exporting their file systems. If the server goes down while the mount is in effect, the file system becomes unavailable. You have the option to wait five minutes until the auto-unmount takes place and try again. (It takes autofs about 5 minutes to do an auto-unmount.) Next time autofs will choose one of the other, available servers. You can do the unmount yourself using the `umount` command (you must be superuser). For example:

```
# umount /shared/local/bin
```

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

Multiple Mounts

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

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

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

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

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

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

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

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

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

But in practice, the mount point is not specified because in the case of a single mount as in the previous example, the location of the mount point is *at* the mount root or "/." So instead of the previous example, you type the following example.

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

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

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

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

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

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

In these mount option examples:

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

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

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

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

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

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

Variables in a Map Entry

You can create a client-specific variable by prefixing a dollar sign (**$**) to its name. This helps you to accommodate different architecture types accessing the same file system location. You can also use brackets to delimit the name of the variable from appended letters or digits. Table 6-2 shows the predefined map variables.

Table 6-2 Predefined Map Variables

Variable	Meaning	Derived From	Example
ARCH	Architecture type	`/usr/kvm/arch`	sun4
CPU	Processor type	`uname -p`	sparc
HOST	Host name	`uname -n`	dinky
OSNAME	Operating system name	`uname -s`	SunOS
OSREL	Operating system release	`uname -r`	5.1
OSVERS	Operating system version (version of the release)	`uname -v`	FCS1.0

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

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

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

Maps That Refer to Other Maps

A map entry, `+`*`mapname`*, used in a file map causes `automount` to read the specified map as if it were included in the current map. The "+" means that the entry is referring to the master map. If `mapname` is not preceded by a slash, then autofs treats the map name as a string of characters and uses the

name service switch policy to find it. If the path name is an absolute path name, then `automount` looks for a local map of that name. If the map name starts with a dash (-), `automount` consults the appropriate built-in map.

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

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

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

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

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

If a file's map has its execute bit set, then `autofs` tries to execute it to obtain a map entry, instead of reading it.

Note – If your `/etc/auto_home` or other local maps have the execute bit set, autofs logs errors to the console when the map is accessed.

To fix the problem, reset the execute bit:

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

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

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

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

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

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

Note – "+" entries cannot be used in NIS+ or NIS maps because NIS+ and NIS search the entry from right to left, not left to right.

Modifying How Autofs Navigates the Network (Modifying Maps)

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

Administrative Tasks Involving Maps

Table 6-3 lists the different administrative tasks you may need to perform involving maps in order to change your autofs environment.

Table 6-3 Map Administration Tasks

Task
Creating maps
Modifying indirect maps
Modifying direct maps
Modifying the `auto_home` map
Modifying the master map
Adding changes to a direct map
Deleting changes from a direct map
Creating an entry in the `auto_master` map

Table 6-4 describes the types of maps and their uses.

Table 6-4 Types of Maps and Their Uses

Type of Map	Use
`auto_home`	Associates names with locations
`auto_master`	Associates a directory with a map
direct	Directs autofs to reference-oriented file systems
indirect	Directs autofs to specific file systems

Table 6-5 describes how to make changes to your autofs environment based on your name service**.**

Table 6-5 Map Maintenance

Name Service	Use
Local files	An editor
NIS	`make` files
NIS+	nistbladm

Table 6-6 tells you when to run the `automount` command depending on the modification you have made to the type of map. For example, if you've made an addition or a deletion to a direct map, you need to run the `automount` command to make the change take effect; however, if you've modified an existing entry, you do not need to run autofs to make the change take effect.

Table 6-6 When to Run the `automount` Command

Type of Map	Run autofs?	
	Add/Delete	Modify existing entry
auto_home	N	N
auto_master	Y	Y
direct	Y	N
indirect	N	N

Modifying the Maps

The following procedures assume that you are using NIS+ as your name service. For NIS or local file name services, see the appropriate documentation.

To modify the master map:

1. **Using the `nistbladm` command, make the changes you want to the master map.**
   See the *Name Services Administration Guide.*
2. **For each client, become superuser by typing `su` at a prompt and then your superuser password.**
3. **For each client, run the `automount` command to ensure the changes you made take effect.**
4. **Notify your users of the changes.**
   Notification is required so that the users can also run the `automount` command as superuser on their own workstations.

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

To modify indirect maps:

- **Using the `nistbladm` command, make the changes you want to the indirect map.**
  See the *Name Services Administration Guide.*

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

To modify direct maps:

1. **Using the `nistbladm` command, add or delete the changes you want to the direct map.**
   See the *Name Services Administration Guide.*
2. **If you added or deleted a mount point entry in step 1, run the `automount` command.**
3. **Notify your users of the changes.**
   Notification is required so that the users can also run the `automount` command as superuser on their own workstations.

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

For instance, suppose you modify the map `auto_direct` so that the directory `/usr/src` is now mounted from a different server. If `/usr/src` is not mounted at this time, the new entry takes effect immediately when you try to access `/usr/src`. If `/usr/src` is mounted now, you can wait until the auto unmounting takes place, and then access it. If this is not satisfactory, you can unmount with the `umount` command and then access `/usr/src`. The mounting will now be done from the new server. If you deleted the entry, you would have to run the `automount` command for the deletion to take effect.

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

Avoiding Mount Point Conflicts

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

You need to mount the partition somewhere else, say on `/export/src`. You would then need, for example, an entry in `/etc/vfstab` that says:

```
/dsk/d0t3d0s5 - export/src ufs 1 yes -
```

and an entry in `auto_src` that says:

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

where `terra` is the name of the computer.

Default Autofs Behavior

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

Autofs uses the name service specified in the automount entry of the `/etc/nsswitch.conf` file. In the example below, the + indicates NIS+. If NIS+ is specified, as opposed to local or NIS, all map names are used as is. If NIS is selected (indicated by an absence of the + sign) and autofs cannot find a map that it needs, but finds a map that contains one or more underscores, the underscores are changed to dots. Then autofs looks up the map again, as shown in Figure 6-5.

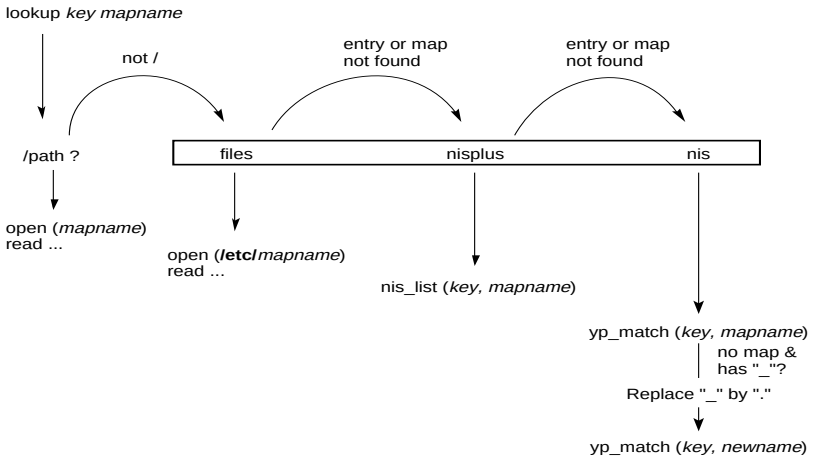


Figure 6-5 How Autofs Uses the Name Service

The screen activity would look like the following example. Notice the + sign, indicating NIS+.

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

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

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

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

Autofs Reference

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

Metacharacters

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

Ampersand (&)

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

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

You can use the ampersand character (**&**) to substitute the key wherever it appears. If you use the ampersand, the previous map now looks like the following:

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

Asterisk (*)

Notice that all the previous entries have the same format. This enables you to use the catchall substitute character, the asterisk (*). The asterisk reduces the whole thing to:

```
*                    &:/export
```

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

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

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

which you can also write as:

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

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

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

because autofs would interpret it as:

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

Special Characters

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

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

Accessing Non-NFS File Systems

Autofs can also mount files other than NFS files. Autofs mounts files on removable media, such as diskettes or CD-ROM. You can also mount files on removable media using Volume Manager, but Volume Manager and autofs do not work together.

Instead of mounting a file system from a server, you put the media in the drive and reference it from the map. Examples of CD-ROM applications are packages such as unbundled products and features. An example of a floppy drive application is DOS. If you want to access non-NFS file systems and you are using autofs, see the following procedures. For more information about Volume Manager, see *File System Administration*.

To access CD-ROM applications:

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

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

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

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

To access floppy disks containing data in PC-DOS files:

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

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

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

Accessing NFS File Systems Using CacheFS

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

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

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

```
cfsadmin -c /usr/cache
```

The following example shows in bold the string you would add to the master map to cache home directories.

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

6

Common Problems and Recommended Solutions

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

Note – Use Administration Tool or see the *Name Services Administration Guide* to perform the tasks discussed in this section. If you are using local files, or NIS, see the appropriate documentation.

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

Problem:

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

Solution:

1. **Create the** `auto_local` **map using the** `nistbladm` **command.**
   See the *Name Services Administration Guide.*
2. **Choose a single, site-specific name for the shared name space so that files and directories that belong to this space are easily identifiable.**

   For example, if you choose `/usr/local` as the name, then the path `/usr/local/bin/scrag` is obviously a part of this name space.
3. **For ease of user community recognition, create an autofs indirect map and mount it at** `/usr/local`**. Set up the following entry in the NIS+ (or NIS)** `auto_master` **map:**

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

   Note that the `ro` mount option implies that clients will not be able to write to any files or directories.
4. **Export the appropriate directory on the server.**

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

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

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

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

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

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

To support incompatible client operating system versions:

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

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

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

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

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

To replicate shared files across several servers:

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

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

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

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

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

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

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

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

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

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

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

How To Set Up a Common View of the /home Directory Structure

Problem:

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

Solution:

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

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

A map for auto_home is also preinstalled under /etc.

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

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

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

To apply security restrictions:

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

```
/home     auto_home     -nosuid
```

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

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

To set up the home directory servers:

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

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

2. **Use the Database Manager of the Administration Tool on-line facility to create and maintain the `auto_home` map.**
   Whenever you create a new user account, type the location of the user's home directory in the `auto_home` map. Map entries can be simple, for example:

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

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

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

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

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

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

How to Consolidate Project-Related files/`ws` Directory Structure)

Problem:

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

Solution:

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

```
/ws      auto_ws      -nosuid
```

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

2. Add the `-nosuid` option as a precaution. (This option prevents users from running `set uid` programs that may exist in any workspaces.)

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

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

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

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

This first attempt provides a map that looks quite simple, but perhaps too simple. It turns out that it is not adequate. The project organizer decides that the documentation in the `man` entry should be provided as a subdirectory under each subproject. Also, each subproject requires subdirectories to describe several versions of the software. Each of these subdirectories must be assigned to an entire disk partition on the server.

Modify the entries in the map as follows:

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

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

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

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

6

Troubleshooting Autofs

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

This section is written especially for advanced system administrators and programmers, and you can skip it without your ability to administer autofs being affected. However, you may want to read it to have an idea of the issues involved.

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

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

Start autofs with the verbose option, otherwise you may experience problems without knowing why.

Reference for Autofs Troubleshooting

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

Error Messages Generated by the Verbose Option

mapname`: Not found`

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

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

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

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

In the previous example, the warning is generated when autofs 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 autofs has found an entry key containing a /. Indirect map keys must be simple names— not path names.

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

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

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

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

`WARNING:` *mount point* `already mounted on`

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

`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` *mount point*`: server not responding`

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

General Error Messages

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

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

`host` *server* not responding

Autofs attempted to contact but received no response.

`Mount of` *`server`*`:`*pathname* on *mount point: reason*

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

`pathconf: server: server not responding`

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

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

Autofs failed to get path conf information for *pathname*.

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

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

`mountpoint``: Not a directory`

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

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

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

`mapname`: *nis_err*

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

`hostname``: exports:` *`rpc_err`*

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

`nfscast: cannot send packet:` *reason*

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

`nfscast: cannot receive reply:` *reason*

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

`nfscast:select:` *reason*

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

6

Index