CONVEX NFS
System Manager's Guide

Order No. DSW-113

Third Edition
November 1990

CONVEX Computer Corporation
Richardson, Texas USA

CONVEX NFS
System Manager's Guide

Order No. DSW-113

Printed in the United States of America

Revision Information for

CONVEX NFS
System Manager's Guide

Edition	Document No.	Description
Third	710-001630-205	Released with ConvexOS V9.0, November 1990. Includes these changes and additions: Added Chapter 3 - The Automounter Added Chapter 5 - Secure Networking
Second, Rev.2	710-001630-203	Released with CONVEX UNIX V7.0, November 1988.
Second, Rev. 1	710-001630-202	Released with CONVEX UNIX V6.2, April 1988.
2.0	710-001630-201	Released with CONVEX UNIX V6.1, October 1987.
1.0	710-000730-000	Initial release with CONVEX UNIX V6.0, April 1987.

Contents

Installing and Debugging NETdisk

Using the Automounter

The Yellow Pages Service

Secure Networking

Reporting Problems

List of Figures

Using This Guide

Purpose and Audience

This document explains tasks you must perform as administrator of the CONVEX Network File System. These tasks include setting up and changing network configuration, troubleshooting NFS-related problems, and administering some of the services/facilities that work with NFS.

This guide addresses installation and maintenance of an NFS configuration. If you are not familiar with the CONVEX NFS product, see *CONVEX NFS Concepts* for an introductory overview.

Organization

Specifically, this guide is organized into the following chapters and appendixes. Chapter 1,Installing and Debugging NFS, explains how to setup and maintain a basic NFS configuration. Chapter 2, Installing and Debugging NETdisk, explains the procedures for adding diskless workstations to your NFS configuration.

Chapter 3, Using The Automounter, explains the automounter facility, a utility that mounts and unmounts remote file systems on an as-needed basis. Chapter 4, The Yellow Pages Service, explains configuration procedures for Yellow Pages (YP), a distributed network lookup service provided with CONVEX NFS. Chapter 5, Secure Networking, explains the installation and administration of Secure NFS.

Technical Assistance

If you have questions that are not answered in this book, contact the CONVEX Technical Assistance Center (TAC).

- Within the continental U.S., call 1(800)952-0379.
- From Canada, call 1(800)345-2384.
- All other locations, contact the local CONVEX office.

Associated Documents

Using this software may require information not specific to the tasks described in this document.

For more information on CONVEX NFS, you can order these books from CONVEX Computer Corporation:

- *CONVEX NFS Concepts*, DSW-109. This book introduces CONVEX NFS to system managers who have no previous experience with NFS.
- *CONVEX NFS Reference Set*, DSW-111 This book provides a reference to the network programming facilities included with CONVEX NFS.
- *Convex NFS Programmer's Reference* DSW-114. This book is the standard reference for CONVEX NFS.

Ordering Documentation

To order the current edition of this or any other CONVEX document, send requests to:

CONVEX Computer Corporation
Customer Service
P.O. Box 833851
Richardson, TX 75083-3851 USA

Include the order number or the exact title, as listed above.

Notational Conventions

This section discusses notational conventions used in this book.

General Conventions

In general, the following conventions are used in this guide:

- **`Bold constant-width font`** identifies user input in examples.
- *Italics*
  - Designate user-supplied variables in a command-line example.
  - Introduce new and important terms.
  - Identify variables in mathematical equations.
  - Indicate titles of documents.
- `Constant-width font` is used to designate input and output, including:
  - Command names and options.
  - System calls.
  - Data structures and types.
  - Directives, program statements, display examples, printout examples, and error messages returned.
- Horizontal ellipsis (...) shows repetition of the preceding item(s).
- Vertical ellipses show that lines of code have been left out of an example.
- Words and abbreviations that indicate keyboard keys you press are identified in a distinctive bold type. For example, **RETURN** refers to the carriage return key. Words separated by a hyphen indicate two keys that you must press simultaneously. For example, **CTRL-X** indicates that you must press and hold down the **CTRL** key and then press the **X** key.
- The word "enter" in a phrase such as "enter **`ls`**" means that you type the command and then press **RETURN**.
- References to the *ConvexOS Programmer's Reference* appear in the form adb(1), where the name of the man page is followed by its section number enclosed in parentheses.

Note

A Note highlights supplemental information.

Caution

A Caution highlights procedures or information necessary to avoid damage to equipment, software, or data.

Reader Response

If you have comments or questions about the contents of this book, please notify the CONVEX documentation department by using the Reader's Forum at the end of this document

Installing and Debugging NFS

1

This chapter discusses the basics of NFS, describes debugging procedures, and explains how to tune the system.

Early sections of this chapter define NFS, describe how it works, and explain how to install NFS on servers and clients and how to mount a file system remotely. Various aspects of system debugging are discussed next. The chapter closes with a discussion of tuning issues (such as how to enable asynchronous operation and various security measures).

What Is NFS?

NFS enables users to share file systems over a network. A client may mount or unmount file systems from an NFS server machine. The client always starts the binding to a server's file system by using the `mount` command or the `mount` system call. Typically, a client mounts one or more remote file systems at start-up by placing lines like these in the /etc/fstab file that mount reads when the system comes up:

```
titan:/usr2 /usr2 nfs rw,hard 0 0
venus:/usr/man /usr/man nfs rw,soft 0 0
cyclops: /usr/docs /mnt nfs rw,intr,bg 0 0
```

See the fstab(5) man page for a full description of the format.

Because clients start all remote mounts, NFS servers control who may mount a file system by limiting named file systems to approved clients listed in the /etc/exports file. For example, this /etc/exports entry:

```
/usr/local              # export to the world
/usr2      -access=mickey:minnie:goofy
           # export to only these machines
```

means that /usr2 may only be accessed by machines mickey, minnie, and goofy. Note that pathnames given in /etc/exports must be the mount point of a local file system, and cannot be subdirectories of another exported file system. See exports(5) for a full description of the format.

How NFS Works

Two remote programs implement NFS service—`mountd` and `nfsd`. A client's mount request talks to `mountd`. `mountd` checks the access permission of the client and returns a pointer to a file system. After the mount completes, access to that mount point and below goes through the pointer to the server's `nfsd` daemon using RPC. Client kernel file access requests (delayed-write and read-ahead) are handled by the `biod` daemons on the client.

You can install and administer NFS using only the information given here. For further information, consult the *CONVEX Network File System Reference Set.*

How to Become an NFS Server

An NFS server is simply a machine that exports a file system or systems. To set up an NFS server, use the following procedure:

1. Install NFS software using the installation procedure distributed with CONVEX NFS.
2. Become superuser.
3. Place the mount-point pathname of the file system you want to export in the /etc/exports file. (See exports(5) for file format details.) For example, to export /usr/src/mybin to every machine, place the following line in /etc/exports.

   ```
   /usr/src/mybin
   ```

   An NFS server may export only file systems of its own.
4. Check the /etc/rc.local file for lines that start `/etc/portmap`. (The NFS installation script adds these lines automatically.) Look for lines like these:

   ```
   if [ -f /etc/portmap ]; then
           /etc/portmap & echo -n 'portmap'>/dev/console
   fi
   ```

   If you do not find these lines in the file, add them immediately after you configure the network with `ifconfig`.

→ Note

You must edit the /etc/rc.local file to become an NFS or yellow pages (YP) server. To verify the daemons that must be started and the order in which they must be started, refer to Starting and Killing NFS Daemons.

5. As we saw above, `mountd` must be present for a remote mount to succeed. Make sure `mountd` is available for an RPC call by checking the /usr/etc/inetd.conf file on the NFS server for this line:

   ```
   mount      dgram udp   waitroot1 /usr/etc/rpc.mountd mountd
   ```

   If it isn't there, add it. For details, see the inetd.conf(5) man page. The configuration file is described in more detail in Chapter 4 of the *RPC Programmer's Manual.*
6. Make sure that /etc/`portmap` is running. To do this, use `ps` and `grep` as follows:
   `# ps ax | grep portmap`
7. If /etc/portmap is not running, use the following command sequence to start `portmap` and reconfigure /usr/etc/inetd:
   `# /etc/portmap`
   `# ps ax | grep inetd`
   `# kill -1 `*`process number`*
   where *`process number`* is the number found by the `ps` and `grep` pipe.

8. Remote mount also needs some number (typically 4) of `nfsd`'s to execute on NFS servers. Add the following lines to /etc/rc.local:

```
if [ -f /usr/etc/nfsd -a -f /etc/exports -a -f /usr/etc/exportfs ] ; then
            /etc/nfsd 4 & echo -n ' nfsd'>/dev/console \
            >/etc/xtab; /usr/etc/exportfs -a &
fi
```

   Add these lines, or your own version of them, if the new NFS server's /etc/rc.local does not enable `nfsd`s. You can enable nfsds without rebooting, by typing, as superuser:

```
# /usr/etc/nfsd 4
# /usr/etc/exportfs -a
```

After these steps, the NFS server should be able to export the named file system. Read the next section, then try a remote mount.

How to Become an NFS Client

An NFS client is a machine that accesses networked file systems. To set up an NFS client, use the following procedure:

1. Install NFS software according to the installation procedure distributed with CONVEX NFS.
2. Add lines to /etc/rc.local to invoke /usr/etc/`biod`. The necessary lines are:

```
if [ -f /usr/etc/biod ]; then
            /etc/biod 4 & echo -n 'biod'>/dev/console
fi
```

   If you cannot find these lines in /etc/rc.local, add them. To enable `biod` without rebooting, enter:

```
# /usr/etc/biod 4
```

➜ Note

You must edit the /etc/rc.local startup file to become an NFS or yellow pages (YP) client. To verify the daemons that must be started and the order in which they must be started, refer to the section, Starting and Killing NFS Daemons.

3. Add the following mount lines to /etc/rc.local:

```
/etc/umount -at nfs
/etc/mount -vat nfs>/dev/console
```

Now you should be able to mount networked file systems on the server. See the next section for an explanation of how to mount file systems remotely.

How to Mount a File System Remotely

You can mount any exported file system onto your machine if you can reach its server over the network and you are included in its /etc/exports list for that file system. Before you mount directories, however, enable read and execute permission for "others" (i.e., drwxrwxr-x) for the mount point (the client directory where you will mount the exported file system). You must do this for both NFS and 4.2, or local partitions. If you do not, attempts to determine the current working directory (with pwd) in the mounted file system may fail.

After you have set correct access permissions, log in as superuser on the machine where you want to mount the file system and enter the following:

```
# mount mount_options server_name:/file_system /mount_point
```

You can use the mount_options flag to create hard, interruptible hard, and soft mounted partitions. Differences in mount options cause programs to react in different ways when they encounter problems with networks or servers.

Hard mounts are the default and cause all operations (system calls) to succeed regardless of the load or a crash by the server. Hard mounts do not acknowledge interrupts, nor do they time out. Instead, they retransmit until they receive an acknowledgment from the server. Because hard mounts do not time out and because no data is lost even when a server crashes they are the option of choice for critical tasks and for writing to remote file systems. There are, however, some disadvantages to hard mounts—operations may take a long time to complete if hard mounts are used on an unreliable remote server. If the server fails to respond, you receive the message "NFS Server Not Responding" on the console and at your terminal.

Suppose, for example, you run ls on an NFS-mounted file system when a server for some remote file system crashes. The process hangs, waiting for a response to a getattr request from the NFS server (and ignoring interrupt signals), until the server comes back up. Then one of the previously ignored interrupts kills the process.

Interruptible hard mounts solve many of the problems associated with hard mounts. They operate identically to hard mounts, but enable you to interrupt operations from the keyboard if a server does not respond. If you are running an `ls` that gets hung, for example, you can break out of it within a short time by typing an interrupt from your keyboard. Certain types of processes—emacs sessions, for example—ignore all types of signals, and cannot be interrupted, no matter what mount option is used. You can, however, interrupt these processes by sending a kill signal from another keyboard.

Soft-mounted partitions return the ETIMEDOUT error to a user's application after three or four attempts to contact a downed server. This allows the application to recover and retry the request or print an error and continue. Soft mounts work best with file systems mounted read-only with the -ro option and for file systems intended primarily for perusal (sources, for example, or directories of manual pages).

For more information on mount options, refer to mount(8) and fstab(5). To make sure you have successfully mounted a file system (and that you have mounted it where you expected to) use either `df` or `mount`, without an argument. These display currently mounted file systems. Using the -t option with `df` enables you to specify the type of file system—NFS or "4.2"—to be displayed. To display only the NFS partitions mounted, for example, enter **df -t nfs**.

Starting and Killing NFS Daemons

Like many system-level programs, both NFS and YP rely heavily on daemons. Under most circumstances, you do not notice the operation of these daemons. However if they die or are killed accidentally, you need to know how to restart them. The listing below contains instructions for restarting these daemons. Always restart daemons in the order in which they appear in the /etc/rc.local file. (A sample file showing the correct line order appears in /etc/rc.local/V9.0.)

Often you must kill daemons to start others; instructions for killing YP daemons are also included here. (YP daemons /usr/etc/ypbind and /usr/etc/ypserv are discussed in the chapter on YP. Ignore instructions for use of these daemons if you are not running YP.)

- /etc/portmap—Always start this daemon first in any sequence. There is no reason to ever kill it. If this daemon should die, kill and then restart /usr/etc/inetd, /etc/ypbind, /usr/etc/ypserv, and /usr/etc/nfsd according to the instructions below.

- /usr/etc/ypserv—Start this daemon only if your machine is a yellow pages slave or master server machine. Start it after the domain name is set and after /etc/portmap is started. If it dies, restart it with the command sequence:

```
# ps ax | grep ypserv
# kill -9  pid1, pid2, ...
# /usr/etc/ypserv
```

  where *pid1, pid2, ...* are the numbers found by the previous instruction.

- /etc/ypbind—Start this daemon only if your machine is running YP, (i.e., if your machine is a yellow pages server or client). When this daemon is running, system programs use YP services rather than reading the local files in the /etc directory. If this daemon dies, restart it with the command:

```
# ps ax | grep ypbind
# kill -9  pid1, pid2, ...
# /usr/etc/ypbind
```

  where *pid1, pid2, ...* are the numbers found by the previous instruction.

- /etc/biod—Start this daemon after /etc/portmap but before you mount file systems with NFS. Typically, you will start four biod daemons. To kill and restart them, use the following command sequence:

```
# ps ax | grep biod
# kill -9  pid1, pid2, ...
# /usr/etc/biod 4
# /usr/etc/exportfs -a
```

  where *pid1, pid2, ...* are the numbers found by the previous instruction.

- /etc/nfsd—Start this daemon after the other daemons in /etc/rc.local. Typically, you will start four nfsd daemons. To kill and restart them, use the following command sequence:

```
# ps ax | grep nfsd
# kill -9 pid1, pid2, ...
# /usr/etc/nfsd 4
```

where *pid1, pid2, ...* are the numbers found by the previous instruction.

- /etc/inetd—Start this daemon from the /etc/rc file. It should be sent a SIGHUP (kill -1) signal; this signal causes inetd to read its configuration file, /etc/inetd.conf. inetd reconfigures itself each time you kill it, sets up new sockets, and restarts itself. To kill inetd, use the following command sequence:

```
# ps ax | grep inetd
# kill -1 process number
```

where *process number* is the number found by the previous instruction.

Record Locking

CONVEX NFS supports advisory record locking. This capability ensures System V compatibility and provides a mechanism for locking records in a network environment, making distributed databases using NFS more feasible and useful. Utilities used to support record locking include:

- `fcntl` - executes file and record locking requests
- `lockf` - a user-friendly front end to fcntl
- `lockd` - the network lock daemon
- `statd` - a status monitor.

Subsequent sections explain these programs.

How Record Locking Works

Record-locking mechanisms enable users to control access to particular file regions, or records. Because ConvexOS has no concept of records, records are defined as arbitrary sequences of bytes located at the current file pointer. (Note that record-locking schemes differ from file-locking programs, which prevent processes from reading or writing entire files.) Advisory record locking enables cooperating processes to implement record locking, but the kernel does not force other processes to respect the advisory locks. Therefore, errant processes may access previously locked records without using advisory locks, possibly resulting in database inconsistencies. However, in practice, application programs running on multiple NFS clients can cooperate to lock and update records of a master database effectively.

How `lockf` Works

The `lockf` library routine is a user interface to the `fcntl` system call, which performs file and record locking. `lockf` places advisory locks on records. In this respect, it is an improvement over, and should not be confused with, `flock`, which provides only a file-locking mechanism, and only works for local files. `lockf` works by enabling you to specify the number of contiguous bytes within a particular file to be locked or unlocked. This data is passed in the size argument and may be positive (to indicate bytes extending forward from the current file offset) or negative (to indicate bytes preceding but not including the current offset). Logically, these regions of contiguous bytes represent records. If you expand these regions of contiguous bytes far enough, you have, in effect, implemented file locking; `lockf` can be used for file locking using this method.

With `lockf` you can lock and unlock file regions; test a region for other locks; and test and then lock a region. Test-and-lock operations that fail with a -1 set the No more processes (EACCESS) diagnostic to indicate that the section is already locked by another process. Lock operations sleep until locks are granted or a signal is caught.

`lockf` does not offer all of the functionality of `fcntl`. In particular, `lockf` restricts you to:

- Use of exclusive locks (`lockf` does F_WRLCK fcntl() requests only)
- Starting at the current position in the file

Note that `lockf` returns the EDEADLK diagnostic, not the ENOLCK returned by `fcntl`. (`flock` returns EDEADLK to ensure compatibility with /usr/group standards.) Typically, complex tasks like specifying shared record locks or using a file offset different from the current position are best handled with `fcntl`.

How `lockd` Works

`lockd` is a user daemon process that supports local and remote record locking. `lockd` processes and arbitrates lock requests from local processes and remote NFS client processes. When the user issues an `lockf` or `fcntl` call, the kernel contacts the local /usr/etc/`rpc.lockd` process. (This happens whether the file is local or remotely mounted via NFS.) If the `lockd` process has not been registered with the port mapper (/etc/`portmap`), `fcntl` exits with the ENOLCK diagnostic. If /etc/`portmap` has not been started, `fcntl` waits (theoretically, forever) either for `portmap` to be started or until a signal is delivered to the process (user presses the interrupt key, for example).

After the local `lockd` process has been contacted, the kernel sends the particular locking request to the local `lockd` process (set lock, test lock, unlock, ...), and the local `lockd` processes the request as follows:

- `lockd` determines whether the request is for a file on the local machine or for a partition remotely mounted with NFS.
- If the request is for a lock on a remote file, `lockd` contacts the local `statd` process to register for monitor server-crash- recovery services. Crash recovery is explained more completely in the section, How to Use the Lock Manager.
- If the request is for a remote file, `lockd` contacts the remote server's `lockd` process to register the request and return the result of the operation; if the request is for a local file, `lockd` registers the request and returns the results to the kernel.

The kernel forwards all record-locking requests to the local `lockd` process. The local `lockd` process contacts the local `statd` process for remote files. Then the local `lockd` process either contacts the remote `lockd` (if the request is for a remote file) or processes the request (if the request is for a local file). The results are returned to the kernel, and the kernel delivers the results to the user.

How to Install the Lock Manager System

To install the lock manager system, you must edit your /etc/rc.local file to start up the two daemon programs—/etc/`rpc.statd` and /etc/`rpc.lockd`— that make up the lock manager. Modify /etc/rc.local as shown below.

1. Find the following lines in /etc/rc.local:

```
#
/usr/etc/quotaon -a
#
echo -n 'local daemons:'>/dev/console
if [ -f /usr/etc/nfsd ]; then
    /usr/etc/nfsd 4 & echo -n ' nfsd'>/dev/console
fi
```

2. Add the following lines to /etc/rc.local. Place these lines immediately below the lines listed in step 1.

```
if [ -f /usr/etc/rpc.statd ]; then
        /usr/etc/rpc.statd & echo -n ' statd'>/dev/console
fi
if [ -f /usr/etc/rpc.lockd ]; then
    /usr/etc/rpc.lockd & echo -n ' lockd'>/dev/console
fi
```

Use this procedure whether you plan to use record locking only on local files or in the network environment with NFS.

The /etc/`rpc.statd` program is a status monitor. It is informed of the recovery of NFS servers after they crash and communicates with /etc/`rpc.lockd` to provide crash-and-recovery functions.

/usr/etc/`rpc.lockd` processes record-locking requests sent either locally by the kernel or remotely by another lock daemon. Lock requests are forwarded to the server's lock daemon using standard RPC and XDR routines. The lock daemon then requests the `statd` daemon (status monitor) for monitor services.

How to Use the Lock Manager

The lock manager system enables you to develop applications that perform transaction-type record locking on ConvexOS and System V implementations. Primary features of the system include:

- Compatibility with System V record locking
- Extension of the record-locking concept onto the NFS network environment by use of the user processes `rpc.statd` and `rpc.lockd`.
- File location transparency. In general, applications do not need to know whether files are local (locked by local `lockd` processes) or remote (mounted via NFS and locked by remote `lockd` processes).

Your application may be sent a SIGLOST signal if a file lock on an NFS file cannot be reclaimed after a server crash. This signal is not standard to System V—it is an extension developed when record locking was extended into the network environment using NFS. If you want to use SIGLOST in your application, you can use the `#ifdef` C preprocessor feature as shown below. Using `#ifdef` enables you to use SIGLOST while maintaining System V source code compatibility.

```
#include <signal.h>
        ...
#ifdef SIGLOST
int lostlock();
#endif
        ...
#ifdef SIGLOST
        signal(SIGLOST, lostlock)
#endif
        ...
#ifdef SIGLOST
lostlock()
```

```
{
 /*
  * Code here to handle lost record lock;  would perhaps
  * try to regain its lock again or exit with a message
  */
}
#endif SIGLOST
```

To use the lock manager system, you must make appropriate calls either to the `lockf` library routine or, with appropriate arguments, to the actual `fcntl` system call. (`lockf` is simply a user-friendly interface to `fcntl`.) `lockf` and `fcntl` are compatible with the System V versions. You can use the system to lock an entire file (even if the file grows or shrinks); therefore, your application can lock either pieces of the file or the entire file itself.

Crash Recovery

The `statd` process maintains state information about which machines have outstanding record lock requests of this server. This information is contained in the /etc/sm directory in the form of one directory entry for each host that has requested monitor services. When statd starts, it notifies the statd process of each host listed in the /etc/sm directory that it has recovered. When the remote statd receives this notification, it notifies its local `lockd` of the recovery. The local `lockd` tries to reclaim the lock, if possible. If it cannot, the process receives a SIGLOST signal to note the lost lock.

Errors Returned

Basically, the errors returned are similar to errors returned for most of the other system calls (EBADF, EFAULT, EINVAL, EAGAIN, EINTR). The new error code for fcntl is ENOLCK, which is returned only by the record-locking code. ENOLCK basically means that no resources are available for this lock, or that the local `lockd` process cannot be contacted.

The `lockf` library returns the same errors as the `fcntl` system call does, except that it changes the ENOLCK error to be EDEADLCK for /usr/group compatibility.

Remote Execution Utilities: `rex` and `rexd`

`rex` and its accompanying daemon, `rexd`, enable you to execute commands remotely, providing you with the opportunity to off-load `nroff`, `make`, and other jobs onto lightly loaded machines. Unlike `rsh` and `rlogin`, `rex` neither starts a shell on the remote host nor performs remote logins. Instead, `rex` accomplishes the remote execution task by mounting local file partitions on remote machines via NFS. In an open environment, `rex` can also be used to improve load balancing between machines. `rex` does not require installation and uses a simple syntactical structure. Subsequent sections describe the use and administration of `rex` and `rexd`.

Like `rlogin` and `rsh`, `rex` operates quickly, without discernible start-up time. In fact, because `rex` doesn't perform remote logins or start remote shells, it is faster than either `rsh` or `rlogin`. Note, however, that while `rex`'s quick start-up time is certainly an advantage, `rex` users are limited to relatively simple command sequences. Because `rex` does not start a remote shell, it cannot interpret the complex command sequences interpreted by `rsh`.

rex and rexd are included by default on standard NFS release distribution tapes. Because no installation is required, you should be able to use these utilities immediately. If you have trouble locating or using rex or rexd, contact the CONVEX Technical Assistance Center.

Using rex

The standard syntax for using rex is:

```
% rex -i -n -d host command arguments
```

where:

-i
Is used to invoke interactive mode (described subsequently).

-n
Is used to specify no input. This option is used to close standard input so that jobs can be run in the background with job control. (Ordinarily, remote standard input is passed from rex's standard input.)

-d
Is used to invoke debugging mode. In debugging mode, rex prints diagnostic messages as work is being done.

host
Is the name of the remote host jobs are to be run on.

command
Is the command to be run.

arguments
Are standard command-line arguments (for example, the -la arguments commonly used with ls).

rex is commonly used to off-load high-overhead jobs (nroff and make jobs are prime examples). If, for example, you want to run an nroff job on a processor named lotsacycles, use rex as follows:

```
% rex lotsacycles nroff *.s
```

Although rex is convenient, you might have tried to use rsh to perform the same task. Unfortunately, rsh would not have worked because the two utilities work differently. rsh executes commands in your home directory on the remote host, while rex executes commands after mounting your current local working directory on the remote host. In the example above, rsh would have looked for .s files in your home directory on the remote machine, and never would have found the local files you wanted to process. In fact, unless your remote home directory contained some .s files, rsh probably would have failed to do anything except return an error message. Table 2-1 illustrates a list of the differences between rex and rsh:

rex	rsh
Executes command with argument on host after mounting current local working directory and file system (if not already mounted).	Executes command with arguments in home directory on host.
cannot execute shell built-ins (e.g., alias).	Can execute shell built-ins because default shell is started on host.
Inherits current directory and ENVIRONMENT variables.	Does not inherit current directory or ENVIRONMENT variables.
Runs interactive commands.	Cannot run interactive commands.
Propagates terminal mode and window size to remote host via interactive commands.	Cannot perform these functions.

Table 2-2 Differences Between rex and rsh

The important thing to remember is that while rsh has no concept of networked file systems, rex always uses them. (If a local file system is not mounted on the remote host, rex always tries to mount it.) rex always works from local files, on a remote machine. rsh always works from remote files, on a remote machine.

Because rex works from remotely mounted versions of the local file system, you should use relative pathnames that exist within the current file system. Note that rex mounts entire file systems, whether or not you are at a mount point. For example, the following command sequence:

```
% pwd
% /mnt/moe/src
% rex make all
```

mounts /mnt, not /mnt/moe/src. Once the file system is mounted, rex changes directories to the current working directory.

You can also use rex to run interactive jobs (like text editors) remotely. Command sequences similar to the following enable you to edit local files on a remote host interactively:

```
% rex -i convex vi *
```

To edit remote files, use a command sequence similar to:

```
% rex -i convex vi /etc/hosts
```

This command sequence enables you to interactively edit convex's /etc/hosts file. Using this approach, you can more easily complete tedious tasks such as editing messages of the day around the network.

Examples of Advanced rex Usage

The example below shows how rex can be used in a shell script to simplify day-to-day programming tasks. In this example, rex is used to run a job on the most lightly loaded machine available from the set including fred, barney, pebbles, and bambam. To use the script, add it to your /bin directory as qrex, and invoke it via:

```
% qrex job_name

#
# rex Application Example
#
#!/bin/sh
#
# qrex -- run a rex job on the lowest loaded machine that is listed as
#             a good candidate.
#
# The HOSTS variable contains an egrep format string of hosts that
# run the rex daemon and can compile sources to produce
# Convex binaries.
#
HOSTS="^fred |^barney |^wilma |^pebbles |^bambam "
REXHOST=`ruptime -l | egrep "$HOSTS" | grep " up " | tail -1 | awk
'{print $1}'`
if [ -z "$REXHOST" ]; then
echo $0: No host available 1>&2
exit 1
fi
rex $REXHOST "$@"
```

rex can also be used with symbolic links to shorten command sequences that you use frequently. The following command sequence, for example:

```
%  ln -s /usr/bin/rex ~/bin/convex
```

establishes a symbolic link between rex and a command name (convex) in your local /bin directory.

Having created the link, you can shorten command sequences referencing the host, convex. For example, once you've created the symbolic link, you can type:

```
% convex make
```

instead of:

```
% rex convex make
```

If you decide to use symbolic links, you'll obviously need symbolic links for every host you intend to use.

Administrative Issues

To administer the remote-execution system, you must understand not only `rex`, but also `rexd`, its accompanying daemon. The following sections describe how to use and administer `rexd`, and discuss how to avoid potential security problems.

Using `rexd`

`rexd` is the server-side daemon for remote program execution. It is started by `inetd` and so is included in the `inetd` configuration file, /etc/inetd.conf. If you encounter problems with `rex`, check inetd.conf for the following line:

```
rexd        streamtcp    waitroot1 /usr/etc/rexd    rexd
```

If you do not find this line, add it and restart `inetd` according to the instructions included in this chapter.

`rexd` is started when a user invokes `rex`—it does not run continuously. When pending rex requests have been completed, `rexd` times itself out, and `inetd` takes control of the socket `rexd` was using.

There is no obvious way for `rexd` to break. If `rexd` does not seem to be working, however, kill it using the following command sequence:

```
# ps ax | grep inetd
# kill -1 [process ID found above]
```

Once `inetd` receives the signal, it reconfigures inetd.conf and restarts itself. The reconfiguration should eliminate problems with `rexd`.

Security Issues

Unlike other networking utilities, `rexd` enables users to mount directories. In theory, this flexible approach can lead to security problems. If, for example, a user with a home directory in mnt invokes the following command sequence:

```
% rex convex4 sleep 10000
```

/mnt is mounted on a temporary partition on the remote host, convex4. The mount point stays active for 10000 seconds, giving users on convex4 access to /mnt.

You can avoid this type of problem, of course, by protecting sensitive partitions in /etc/exports. If, in the previous example, /mnt had not been included in the exports file, users would be unable to mount the partition. Instead, they would receive the error message:

```
cannot mount file system
```

Note that permission checking under `rex` is as strong as the permission checking used with `rsh` and `rlogin`. In all cases, the system checks not only for legitimate passwords, but also checks against the listings in /etc/host.equiv. All things considered, `rex` is a reasonable security risk in open environments, but if you operate a secure environment, you may want to reconsider distributing `rex` to your users. If you decide not to provide `rex`, you can turn it off by removing its entry from /etc/inetd.conf, then reconfiguring `inetd` by sending a SIGHUP (kill -1) signal.

Using Named Pipes

Software supporting named pipes is distributed to CONVEX licensees. Named pipes are ordinary pipes that exist permanently in the file system with directory entries and pathnames. Because these pipes can be accessed by name, they can be used for applications for which unnamed pipes are not suited. Typically, named pipes are used to allow a number of processes to communicate with a daemon process.

Note that CONVEX named pipes can be used only locally. Even if the named pipe exists as an inode on a remote machine, data written to the pipe is only accessible to processes on the same client. Therefore, even though multiple clients mount the same file system, they will have separate streams associated with any named pipe in that file system.

For more information on the use of named pipes, see the mknod(2), mknod(8), and open(2) man pages.

Debugging the Network File System

Before trying to debug NFS, read the section on how NFS works plus the man pages mount(8) nfsd(8), biod(8), showmount(8), rpcinfo(8), mountd(8c), inetd(8c), fstab(5), mtab(5), and exports(5). You do not have to understand them fully, but you should be familiar with the names and functions of the various daemons and database files.

When tracking down an NFS problem keep in mind that, like all network services, the three main points of failure are: the server, the client, or the network itself. The debugging strategy outlined below tries to isolate each individual component to find the one that isn't working.

For example, let's look at a sample mount request made from an NFS client machine:

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

and try to understand how it works and how it can fail. The example asks the server machine krypton to return a file handle (fhandle) for the directory /usr/src. This fhandle is then passed to the kernel in the mount system call. The kernel looks up the directory /krypton.src and, if conditions are right, it ties the fhandle to the directory in a mount record. From now on, all file system requests to that directory and below go through the fhandle to the server krypton.

Now, obviously, if you are experiencing problems, things are not working as described in this example. So how do you determine where the problem is? First, look at the next section, which contains some general pointers. Then, read the subsequent sections that describe problems and their likely causes.

General Hints

If a client is having NFS trouble, make sure the server is up and running. From a client, you can type:

```
% /usr/etc/ping server_name
```

to see if the server is up at all. If the server responds, type:

```
% /usr/etc/rpcinfo -p server_name
```

The server should print a list of program, version, protocol, and port numbers that resembles:

```
program vers proto   port
100000    2   tcp    111  portmapper
100000    2   udp    111  portmapper
100004    2   udp   1026  ypserv
100004    2   tcp   1027  ypserv
100004    1   udp   1026  ypserv
100004    1   tcp   1027  ypserv
100007    2   tcp   1028  ypbind
100007    2   udp   1034  ypbind
100007    1   tcp   1028  ypbind
100007    1   udp   1034  ypbind
100009    1   udp   1023  yppasswdd
100003    2   udp   2049  nfs
100002    1   udp   1120  rusersd
100002    2   udp   1120  rusersd
100001    1   udp   1126  rstatd
100001    2   udp   1126  rstatd
100001    3   udp   1126  rstatd
100008    1   udp   1131  walld
100005    1   udp   1134  mountd
100011    1   udp   1137  rquotad
100012    1   udp   1140  sprayd
100017    1   tcp   1055  rexd
100026    1   udp   1145  bootparam
```

If that works, you can also use `rpcinfo` to determine whether the `mountd` server is running:

```
% /usr/etc/rpcinfo -u server_name 100005 1
```

(The **100005 1** comes from the `mountd` line in the `rpcinfo` output above.) This should come back with the response:

```
proc 100005 version 1 ready and waiting
```

If these fail, log in to the server's console and see if the server is running.

If the server is alive but your machine cannot reach it, check the Ethernet connections between your machine and the server.

If the server is fine and the network is fine, use `ps` to check your client daemons. You should have a `portmap` and several `biod` daemons running. (If you are running YP, `ypbind` must also be running.) For example, running `ps` should result in output something like this for a client using YP:

```
% ps ax
32 ?  I     1:07 /etc/portmap
38 ?  I     0:42 /usr/etc/ypbind
61 ?  S     0:45  (biod)
62 ?  S     0:36  (biod)
63 ?  S     0:30  (biod)
64 ?  S     0:27  (biod)
```

The four sections below deal with the most common types of failure. The first tells what to do if your remote mount fails; the next three describe servers not responding after you have file systems mounted.

When Remote Mount Operations Fail

This section deals with problems related to mounting. If `mount` fails for any reason, check the sections below for specific details about what to do. They are arranged according to where they occur in the mounting sequence and are labeled with the error message you are likely to see.

`mount` can get its parameters either from the command line or from the file /etc/fstab (see the mount(8) man page). The example below assumes command line arguments, but the same debugging techniques work if /etc/fstab is used in the mount -a command.

Keep in mind the interaction of the various players in the mount request. If you understand this, the problem descriptions below make more sense.

Let's look again at the previous mount request example:

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

and see what steps `mount` goes through to mount a remote file system.

1. `mount` opens /etc/mtab and checks that this mount has not already been done.
2. `mount` parses the first argument into host krypton and remote directory /usr/src.
3. If you are running YP, mount calls YP binder daemon `ypbind` to determine which server machine to find YP server on. It then calls the `ypserv` daemon on that machine to get the Internet protocol (IP) address of krypton from the "hosts" map. (If you are not running YP, `mount` looks in the local version of /etc/hosts.)
4. `mount` calls krypton's portmapper (/etc/portmap) to get the port number of `mountd`.
5. `mount` calls krypton's `mountd` and passes it /usr/src.
6. krypton's `mountd` reads /etc/exports and looks for the file system containing /usr/src.
7. krypton's `mountd` calls YP server `ypserv` to expand the host names and netgroups in the export list for /usr/src. `mountd` determines access permissions from the expanded files, then proceeds if all the requested mount is allowed. (This happens only if you are running YP—otherwise, `mountd` uses the local versions of /etc/hosts and /etc/netgroups.)
8. krypton's `mountd` does a `getfh` system call on /usr/src to get the fhandle.
9. krypton's `mountd` returns the fhandle.
10. `mount` supplies the fhandle and /krypton.src via the `mount` system call.
11. `mount` checks if the caller is superuser and if /krypton.src is a directory, then associates the fhandle with /krypton.src.
12. The `mount` system call does a `statfs` call to krypton's NFS server (`nfsd`).
13. `mount` opens /etc/mtab and adds an entry to the end.

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

- mount: ... already mounted

  The file system that you are trying to mount is already mounted or it has a bogus entry in /etc/mtab.

- mount: ... Block device required

You probably left off the krypton: part of

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

The `mount`" command assumes you are doing a local mount unless it sees a colon in the file system name or the file system type is NFS in /etc/fstab.

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

  If you see this message, it means the argument you gave `mount` was not in any of the entries in /etc/fstab. If `mount` is called with a directory or file-system name but not both, it looks in /etc/fstab for an entry whose file system or directory field matched the argument. For example:

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

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

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

  it does the mount as if you had typed:

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

- /etc/fstab: No such file or directory

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

- ... not in hosts database

  This means YP could not find the hostname you gave it in the /etc/hosts database or that the YP daemon (`ypbind`) is dead on your machine. (Note that if you are not running YP, no hostname entry is in the local /etc/hosts file.) First check the spelling and the placement of the colon in your mount call. If those looks right, make sure that `ypbind` is running by typing:

  ```
  # ps ax | grep ypbind
  ```

  Try `rlogin` or `rpc` to some other machine. If this also fails, your `ypbind` is probably dead or hung. If you get this message only for one host name, it means that the /etc/hosts entry on YP server needs to be checked. See the section on debugging YP later in this chapter.

- mount: directory path must begin with '/'

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

- mount: ... server not responding: RPC: Port mapper failure - RPC: Timed out

  Either the server you are trying to mount from is down, or its portmapper is dead or hung. Try logging in to that machine. If you can log in, try running:

  ```
  # rpcinfo -p
  ```

  You should get a list of registered program numbers. If you do not, restart the portmapper on the server according to the instructions in the "Starting and Killing NFS Daemons" section. Note that restarting the portmapper requires that you then kill and restart both `inetd` and `ypbind`.

  If you cannot `rlogin` to the server but the server is up, check your Ethernet connection by trying to `rlogin` to some other machine. Also check the server's Ethernet connection.

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

  This means that `mount` got through to the portmapper but the NFS mount daemon (`rpc.mountd`) was not registered. Go to the server and be sure that /usr/etc/`rpc.mountd` exists and that there is an entry in /etc/inetd.conf exactly like:

```
mount      dgram udp   waitrootl /usr/etc/rpc.mountd  mountd
```

Finally, use `ps` to be sure that the Internet daemon (`inetd`) is running. If you had to change /etc/inetd.conf, you must kill `inetd` and restart it. Again, refer to the instructions in the "Starting and Killing NFS Daemons" section. (A complete description of the inetd.conf file is contained in Chapter 4 of the *RPC Programmer's Guide.*)

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

  Either the remote directory or the local directory does not exist. Check spelling. Try to `ls` both directories.

- mount: not in export list for ...

  Your machine name is not in the export list for the file system you want to mount from the server. You can get a list of the server's exported file systems by running:

  # showmount -e *hostname*

  If the file system you want is not in the list, or your machine name or netgroup name is not in the user list for the file system, log in to the server and check the /etc/exports file for the correct file system entry. A file system name that appears in the /etc/exports file, but not in the output from `showmount`, indicates a failure in `mountd`. Either it could not parse that line in the file, or it could not find the file system, or the file system name was not a locally-mounted file system. See the exports(5) man page for more information. If exports seems okay, check the server's `ypbind` daemon: it may be dead or hung.

- mount: ...: Permission denied

  This message is a generic indication that some authentication failed on the server. It could simply be that you are not in the export list (see above), or the server couldn't figure out who you are (`ypbind` dead), or it could be that the server doesn't believe you are who you say you are. Check the server's /etc/exports and `ypbind`. Here, you can change your hostname with hostname and retry the mount.

- mount: ...: Not a directory

  Either the remote path or the local path is not a directory. Check spelling and try to `ls` both directories.

- mount: ...: Not owner

  You have to do the mount as root on your machine because it affects the file system for the whole machine, not just you.

- mount: ...: Cannot mount a directory on top of itself

  Self-explanatory.

When Programs Hang

If programs hang doing file-related work, your NFS server may be dead. If, for example, you are using machine krypton as a server, you may see the message "NFS server krypton not responding, still trying" on your terminal. If you see a message like this, there is probably a problem either with one of your NFS servers or with the Ethernet. You can determine which NFS server it is by looking in /etc/hosts. Programs can also hang if a YP server dies (refer to YP below).

If your machine hangs completely, check the server(s) from which you have mounted. If one of them (or more) is down, do not worry; when the server comes back up, your programs continue automatically and they won't even know the server died. No files are destroyed.

If a soft-mounted server dies, other work should not be affected. Programs that time-out trying to access soft-mounted remote files fail with errno ETIMEDOUT, but you should still be able to get work done on your other file systems.

If all the servers are running, ask someone else using the server or servers that you are using if they are having trouble. If more than one machine is having problems getting service, it is probably a problem with the server's NFS daemon (`nfsd`). Log in to the server and do a `ps` to see if `nfsd` is running and accumulating CPU time. If not, you may be able to kill and then restart `nfsd`. If this does not work, you must reboot the server.

If other people seem to be okay, you should check your Ethernet connection and the connection of the server.

When the System Hangs at Startup

If your machine comes part way up after a boot, but hangs where it would normally be doing remote mounts, probably one or more servers is down or your network connection is bad. Refer to the "Program Hung" and "Remote Mount Failed" sections above.

To avoid a hung system, add the bg flag to the NFS partitions listed in /etc/fstab. For example:

```
# krypton:/usr/man /usr/man nfs rw,soft,bg 0 0
```

When Remote File Access Seems Slow

If access to remote files seems unusually slow, type:

```
# ps aux
```

on the server to be sure it is not being clobbered by a runaway daemon, bad tty line, etc. If the server seems okay and other people are getting good response, make sure your block I/O (`biod`) daemons are running; type **`ps ax`** and look for `biod`. If they are not running or are hung, you can find the process ID's by typing:

```
# ps ax | grep biod
```

and kill them with:

```
# kill -9 pid1 pid2 pid3 pid4
```

Restart them with:

```
# /usr/etc/biod 4
```

To determine whether they are hung, do a ps as above, copy a large remote file, then do another `ps`. If the biods do not accumulate CPU time, they are probably hung.

If `biod` is okay, check your Ethernet connection. The command `netstat -i` tells you if you are dropping packets. Also, you can use `nfsstat -c` and `nfsstat -s` to lookup retransmission rates for the client or server. A retransmission rate of 5% is considered high. Excessive retransmission usually means a bad Ethernet board, a bad Ethernet tap, a mismatch between board and tap, or a mismatch between your Ethernet board and the server's board.

Tuning NFS

The following topics are discussed in this section:

- Superuser access to networked files
- How to enable asynchronous NFS operation
- How to check privileged ports
- How to check IP source addresses
- How to patch SUN clients for compatibility with file systems with large block sizes
- Correcting clock skew in user programs.

Superuser Access to Remote Files

Under NFS, a server exports file systems it owns so clients may remote mount them. When a client becomes superuser, it is by default denied permission on remote-mounted file systems. Let's look at the following example:

```
% cd
% touch test1 test2
% chmod 777 test1
% chmod 700 test2
% ls -l test*
-rwxrwxrwx  1 jsbach       0 Mar 24 16:12 test1
-rwx------  1 jsbach       0 Mar 24 16:12 test2
```

Now, retry it as root.

```
% su
Password:
# touch test1
# touch test2
touch: test2: Permission denied
# ls -l test*
-rwxrwxrwx  1 jsbach       0 Mar 21 16:16 test1
-rwx------  1 jsbach       0 Mar 21 16:12 test2
```

The problem usually appears during the execution of a set-uid root program. Programs that run as root cannot access files or directories unless the permission for other allows it.

Also, you cannot change ownership of remote-mounted files. Since users cannot do a `chown` command and root is treated as a normal user on remote access, only root on the server can change the ownership of remote files. For example, as yourself, you attempt to `chown` a new program, a.out, that must be set-uid root. It does not work, as demonstrated here:

```
% chmod 4777 a.out
% su
Password:
# chown root a.out
a.out: Not owner
```

To change ownership, you must log in to the server as root, then make the change. Or, you can move the file to a file system owned by your machine (for example /usr/tmp is always owned by the local machine) and make the change there.

If you are prepared to accept the security risks, you may also solve this problem by enabling over-the-net root access. Enabling over-the-net root access allows client root accounts superuser privileges on remotely-mounted partitions. Over-the-net root access can be enabled on a file-system-by-file-system basis (via changes to the /etc/exports file). Before you consider using this method, however, note carefully:

Note

CONVEX does not recommend enabling over-the-net root access as discussed in the following paragraphs.

To enable over-the-net root access for particular file systems, add one of the flags (-anon=0 or -root=) to the listings in /etc/exports. Although the arguments discussed here are used most frequently, other arguments are available; see the exports(5) man page for more information. Modify the server's /etc/exports file as follows:

- To enable over-the-net root access to particular hosts on the access list, use the -root=hostname command. The command syntax is `-root=`*hostname* `[:`*hostname*`]`. In the following example, root access is given to convex2, but not to convex3.

```
/tmp
/sw
/mnt1
/usr        -root=convex2,access=convex2:convex3
```

- To enable over-the-net root access for all hosts, add the -anon=0 flag after the listing for each file system to be accessed by root. In the following example, the /usr file system has been modified to accept over-the-net root access by convex2 and convex3:

```
/tmp
/sw
/mnt1
/usr        -anon=0,access=convex2:convex3
/fonts            -access=convex4
/usr/spool -access=convex2:convex3:convex4
```

- To enable read-only access: For each file system to be accessed read-only, add the -ro flag after the listing. In the following example, over-the-net root access is mapped to UID -2 for every file system listed (because the default value is -2). No user, including root, can write the files in /usr/src. Note that although users can mount /usr/src read/write, attempted writes by any user (including root) fail with the EROFS (Read-Only File System) error:

```
/tmp
/sw
/mnt1
/usr
/fonts
/usr/src   -ro,access=convex2:convex3
```

- To enable read-write access only on selected machines, use the -rw=hostname [:hostname] option. For example, the following lines specify that convex3 has read-write privileges, but not convex2.

```
/tmp
/sw
/mnt1
/usr
/fonts
/usr/src  -rw=convex3,access=convex2:convex3
```

Asynchronous NFS Operation

NFS servers normally operate in a synchronous manner—data is written to permanent storage before an NFS transaction is acknowledged. Synchronous operation enables the server to maintain file integrity despite crashes. With synchronous operation, the client cannot differentiate between a server crash and slow server response. The client retries until the transaction is completed successfully.

The penalty paid for reliability is extra I/O performed on the server. Each time the client makes a write request, the server writes not only the data buffer, but "in-core" copies of the inode and indirect blocks as well. CONVEX systems software staff estimate that writing large files could proceed several times faster if the server operated asynchronously, making full use of the ConvexOS buffer cache.

Note

Do not attempt the following procedures until you carefully consider the NFS environment to be changed and its ability to recover from inconsistent data if a server fails. Weigh carefully your need for the extra performance against the inconvenience and administrative effort required when the server crashes. Note that client machines may not be aware of the server's crash and may not be able to detect that data has been lost.

To enable asynchronous operation on an export file system basis, add the async option in the /etc/exports file. For example, adding the following lines to /etc/exports enables asynchronous operation for /tmp and /usr/spool:

```
/tmp       -async
/usr/spool -access=convex2:convex3:convex4,async
```

Note that file systems served by asynchronous servers should be mounted "soft," so that NFS returns an error when the server malfunctions. You can set the time-out value and retry count values so that the operation times out and returns an error before the server can be rebooted. (The default values ensure this.)

Checking Privileged Ports

The Berkeley UNIX operating system includes some Internet domain source ports to which only privileged users can attach (these are known as "privileged ports"). NFS does not check to see if a client is bound to one of these.

That is, an NFS server has no way of knowing whether a client's file request originated from the real client's kernel or from an intruder's user program. You may want to enforce privileged port checking for extra security. To turn on NFS server port checking, use the following procedure:

1. Add the following line to bootcmd on the SPU:
   `tune cpu nfs_portmon = 1`

2. Be sure that the /etc/inetd.conf file includes the following line:
   `mount dgram udp wait root 1 /usr/etc/rpc.mountd mountd`

   This line ensures that -n option is not set in the rpc.mountd daemon.

3. Restart `inetd` according to the instructions in the "Starting and Killing NFS Daemons" section, that is, send `inetd` a SIGHUP (kill -1) signal. (If you do not restart `inetd`, privileged port checking does not start until the next reboot.)

Note

Some non-UNIX systems do not enforce the privileged port convention (in particular, PCs with 3Com boards).

Client Bugs With Large File System Block Sizes

If you are using a Sun 3 workstation as a client to a CONVEX NFS server, note the following. Releases prior to and including V3.0 of the Sun operating system panic when they access remote file systems with block sizes larger than 8 Kbytes. (Panics are due to an NFS bug in the Sun kernel.) This bug is relevant to you because CONVEX servers can export file systems with block sizes as large as 64 Kbytes.

It is possible to fix this bug by patching the Sun 3 kernel. To do this, use `adb` to change nfs_mount+2b6. The previous instruction was bles, hex 6f06 (halfword). Replace it with a bra instruction, hex value 6006 (halfword). (Note that all these changes are to be made on the Sun machine.)

```
sun# adb -w /vmunix
nfs_mount+2b6?x        (check for 6f06)
nfs_mount+2b6?w 6006(replace 6f06 with 6006)
$q
sun#
```

Versions of NFS ported by other vendors may not be designed to support file systems with block sizes larger than 8 Kbytes. You may have difficulty using other types of hardware as a client for these file systems. Naturally, CONVEX NFS fully supports file systems block sizes from 4 Kbytes through 64 Kbytes.

Correcting Clock Skew in User Programs

Because each machine on the network keeps its own time, the NFS server and client clocks may not be synchronized. This might cause a problem in certain situations. All known clock skew problems have been corrected. Here are examples of two problems and how they were fixed.

Many programs make the assumption that an existing file could not have been created in the future. For example, `ls` makes this assumption. The command `ls -l` has two basic forms of output, depending on how old the file is:

```
# date
Jan 22 15:27:01 PST 1985
# touch file2
# ls -l file*
-rw-r--r--  1 root          0 Dec 27  1983 file
-rw-r--r--  1 root          0 Jan 22 15:27 file2
```

The first form of `ls` prints the year, month, and day of the last change to the file—if the file is more than six months old. The second form prints the month, day, and minute of the last change to the file is fewer than six months old.

`ls` calculates the age of a file by simply subtracting the time of the last change to the file from the current time. If the results are greater than six months, in seconds, the file is "old."

Now assume that the time on the server is Jan 22 15:30:31 (three minutes ahead of our local machine's time):

```
# date
Jan 22 15:27:31 PST 1985
# touch file3
# ls -l file*
-rw-r--r--  1 root          0 Dec 27  1983 file
-rw-r--r--  1 root          0 Jan 22 15:26 file2
-rw-r--r--  1 root          0 Jan 22  1985 file3
```

The problem is that the difference between the two times is huge:

(now) - (time_of_last_change) =
(now) - (now + 180 seconds) =
-180 seconds = huge unsigned number,
that is greater than six months.

Thus, `ls` believes the new file was created long ago in the past. `ls` has been modified to deal with files that are created a short time in the future.

`ranlib` was also modified to deal with clock skew. `ranlib` timestamps a library when it is produced, and `ls` compares that time stamp with the last modified date of the library. If the last modified date occurred after the time stamp, `ls` instructs the user to run `ranlib` again and reload the library.

If the library lives on a server whose clock is ahead of the client that runs `ranlib`, `ls` always issues an error message. `ranlib` has been altered and now sets the time stamp to the maximum value of the current time and the library's modify time.

In general, if your application depends on either local time or the file system time stamps, it must deal with clock-skew problems when it uses remote files.

Incompatibilities With Non-NFS File Systems

A few things work differently, or do not work at all, on remote NFS file systems. Here we discuss the incompatibilities and suggest how to work around them.

File Operations Not Supported

The `flock` call is not defined across the network. Neither does it recognize locks placed by `lockf` or `fcntl`. This means that server processes using `flock` may mistakenly believe they have exclusive access to a file that has already been locked using `lockf`.

To prevent this problem, use `lockf` exclusively and avoid `flock`. For more information, see the sections on record locking in this chapter.

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

Access to Remote Devices

While using NFS, attempts to access with NFS a remote-mounted device or any other character or block special file (like named pipes) are treated as if a local device is being accessed.

Installing and Debugging NETdisk

2

The NETdisk product is included with all standard shipments of CONVEX NFS. NETdisk allows a CONVEX server machine to support booting diskless workstations using NFS.

This chapter discusses NETdisk at a conceptual level and provides sample installation procedures. Specifically, this chapter describes:

- disk space requirements
- how to use the INSTALL program to install the client software
- how to boot a diskless client workstation
- how to add and remove clients
- how to install optional software after running INSTALL
- important files used by the NETdisk system
- the steps taken by a diskless client workstation when booting from a CONVEX NFS server.

For further information on NETdisk, see the INSTALL(8), setup_client(8), and sunboot(8s) man pages.

➜ Note

The only clients currently supported are Sun-3 workstations running SunOS 4.0 and 4.0.3. Other workstations that use the same boot procedure as the Sun-3 may be used (see the "Theory of Operation" later in this chapter for further information).

Though some of the install scripts described in this chapter offer Sun-4 as an option, a Sun format change prevents Sun-4 workstations from installing correctly with these scripts. If you wish to install Sun-4 workstations, please contact the Convex Technical Assistance Center. Contact your CONVEX sales representative if you wish to boot a Sun-2 system running SunOS 4.0 or later from your CONVEX server.

What Is NETdisk?

NETdisk is a collection of administrative programs loaded into the /usr/etc/install directory when NFS is installed. NETdisk is used to boot diskless workstations (called clients) from a CONVEX NFS server.

With NETdisk, you can use relatively inexpensive diskless workstations without purchasing expensive boot or file servers. With NETdisk, you can use your CONVEX NFS server to boot a diskless client and to serve exclusively the files it needs.

To boot a diskless workstation you must load the workstation manufacturer's software distribution tape on your CONVEX server. These tapes must be ordered from the workstation vendor. CONVEX recommends purchasing software distributions on 1600-bpi or 6250-bpi 1/2 inch magnetic tape. Using software distribution tapes lets you load the software without having to use the network to access a remote tape drive.

For example, the following software distribution formats are available from Sun Microsystems:

- 1/2 inch 1600-bpi magnetic tape
- 1/4 inch QIC-24 format cartridge tape

Files on these tapes can be extracted either remotely or locally; a remote Sun computer must be used to read the 1/4 inch cartridge tape format.

NETdisk Disk Space Requirements

The amount of disk space consumed per workstation depends on the following factors:

- The number of optional software packages loaded per architecture. Each loaded architecture may be loaded with optional software packages; the more packages loaded, the larger the disk space consumption.
- The number of clients supported. Each client workstation requires its own root directory.
- The size of the clients' swap files. Each client workstation requires one or more files that it uses exclusively as a swap device. Disk space must be statically allocated for this swap file on the CONVEX server to be certain that a swap write request does not fail because a disk partition filled up. The size of the swap file varies depending on the amount of memory and type of application the client workstation intends to run.

Following are approximate disk space values for a SunOS 4.0 release; use these values to decide how much disk space to allocate for NETdisk:

- architecture-shared executables (/usr): 58 megabytes
- root directory per client workstation: 2 megabytes
- swap file per client workstation: 12 megabytes (minimum)

Thus, to support one Sun-3 workstation with all standard optional software loaded requires about 72 megabytes of disk space. Each additional client, however, only requires about 14 Megabytes. Supporting 10 diskless Sun-3 workstations would therefore require 198 Megabytes of disk storage (58 megabytes for /usr and 140 megabytes for the root directories and swap files of the 10 diskless clients).

➜ Note

CONVEX recommends using a separate file system to hold the NETdisk files.

CONVEX suggests that you create a new file system called /export to hold the NETdisk files. Then, use the following full pathnames:

/export/root/client-name	root directory for client client-name
/export/swap/client-name	swap file for client client-name
/export/exec/arch-name	architecture (/usr) directory for architecture arch-name

For example, the root directory for a Sun-3 architecture workstation named jogger would be /export/root/jogger, the swap file would be /export/swap/jogger, and the architecture-shared Sun-3 binaries would be in the /export/exec/sun3 directory.

These disk space values are estimates. Your disk space requirements may differ depending on the size of swap files used and the amount of optional software installed.

Before Loading Software

Follow these steps before loading the software and booting a diskless workstation.

1. Add the host name and internet protocol (IP) address of the new client workstation (the one you are going to boot) to the /etc/hosts file. For information on assigning IP addresses, see the *CONVEX Internet Services System Manager's Guide.*
2. Identify the Ethernet address of the client workstation by powering up the monitor.
3. Login to the NETdisk server.
4. Use your favorite editor to add the Ethernet address to the /etc/ethers file.
5. If your installation uses Yellow Pages (YP), remake the YP maps.
6. Execute the following arp command:

```
/etc/arp -e /etc/ethers
```

This command takes the client Ethernet address in the /etc/ethers file and makes a permanent address entry in the kernel.

7. Check to see that the /etc/rc.local file contains the following lines which are inserted automatically by the NFS installation procedure:

```
if [ -d /tftpboot ] ; then
      if [ -f /etc/ethers -a -f /etc/arp ] ; then
         /etc/arp -e /etc/ethers & echo -n ' rarp'          >/
dev/console
      fi
fi
```

In subsequent boots of the diskless client workstation, these lines in the /etc/rc.local file will automatically map the client's Ethernet address to a permanent address entry in the kernel.

8. Set up the exports file system. See the newfs(8) man page for details on setting up a new file system.

Using INSTALL to Load Software

This section guides the superuser through an example installation of the client architecture software. This example installation assumes the following:

- You are using the local 1/2 inch magnetic tape drive to install a Sun-3 architecture.
- A diskless Sun-3 workstation named jogger is also configured and set up at the same time.
- The CONVEX machine is running in multiuser mode and the NFS system is fully functional.
- The SunOS 4.0 software distribution tape for the Sun-3 (68020 SUNBIN) has been loaded on the local CONVEX machine's tape unit 0.
- The CONVEX server machine is running the yellow pages (YP) system.
- The /export file system has been set up and is large enough to hold the executables, root, and swap files.
- The hosts database lists the client as a known host.
- The /etc/ethers file (or the ethers map) has the client Ethernet address.
- The current directory "." is in the PATH environment variable.
- You have superuser (root) privileges.

This procedure takes approximately two hours to complete. The program, however, does not have to be constantly monitored after you have selected the optional software you want installed. You may need to change tapes. The INSTALL program prompts you to do so, if necessary.

1. Log in as superuser.
2. Change your working directory to the /usr/etc/install directory, as shown below.
   ```
   # cd /usr/etc/install
   ```

This directory contains all the administrative scripts needed to extract software distributions from tapes and to add and remove clients.

3. Enter **INSTALL** as shown below to invoke the INSTALL program.
   ```
   # INSTALL
   ```

4. The INSTALL program prompts you for the type of installation. Answer `local` as shown below.
   ```
   Enter tape drive type ? [local | remote]: local
   ```

5. The INSTALL program then asks which tape drive to use. You must specify a non-rewind device. For example, for a local installation on 1/2 inch magnetic tape, use /dev/rmt12. For tape unit 1, use /dev/rmt13.
   ```
   Enter tape type ? [ar[08] | st[08] | mt[08] | xt[08] | /dev/???]: /
   dev/rmt12
   ```

When using a local magnetic tape, you must specify the full pathname to a non-rewind tape drive. Do not use the shortcut **mt8** specification because the INSTALL program assumes it to be a Sun cartridge tape drive, which is unavailable on CONVEX machines.

If you had selected **remote** in the previous step, INSTALL would have prompted for the host name to which the remote tape is connected. Although not the case in this example, note that remote installations require that the remote host's /.rhosts file contain the local CONVEX host name on a line by itself.

Also, when doing a remote installation using a 1/4 inch QIC-24 cartridge tape from an already configured Sun system, the correct response at this point is **st08** to select the Sun QIC-24 cartridge tape drive.

6. Next enter the architecture type to load. Since this example procedure assumes a Sun-3 architecture, the appropriate response is **sun3**, as illustrated below.
   ```
   Enter next architecture type to load

   [ sun3 | sun4 | ... | continue | done ]: sun3
   ```

7. The INSTALL program asks where to load the executables. CONVEX strongly recommends loading these in the /export/exec directory, as shown below.
   ```
   Enter pathname for sun3 executables ? /export/exec
   ```

The directory /export/exec/sun3 is automatically created to hold the executables.

8. The INSTALL program now tries to access the tape. Be sure the tape is mounted and online. Press **RETURN** when ready.
   ```
   Reading the table of contents for sun3 architecture...

   Mount a sun3 release tape and hit RETURN, or enter "exit" :
   ```

At this point, the INSTALL program reads a standard format-header file on the tape that describes the entire contents of the tapes comprising the release. The INSTALL program then prompts for confirmation to load each file on the tapes. No files are loaded at this time. Rather, an extraction list is created for automatic extraction at a later time.

In this example installation, all optional software is extracted except for the `Games` file, which will be used in a later exercise to add optional software after an initial installation.

```
Select optional software for the sun3 architecture:
Install "User File System" [20971520 bytes; required] ? [y/n]: y
Install "Sys" [2721792 bytes; desirable] ? [y/n]: y
Install "Networking tools and programs" [953344 bytes; desirable] ? [y/n]: y
Install "Debugging tools" [3383296 bytes; desirable] ? [y/n]: y
Install "SunView_Users Programs" [1453056 bytes; common] ? [y/n]: y
Install "SunView_Programmers Programs" [2064384 bytes; optional] ? [y/n]: y
Install "SunView_Demo Program source" [564224 bytes; optional] ? [y/n]: y
Install "Text Processing tools" [690176 bytes; optional] ? [y/n]: y
Install "Install tools" [1004544 bytes; optional] ? [y/n]: y
Install "User Level Diagnostics tools" [1491968 bytes; optional] ? [y/n]: y
Install "SunCore Libraries" [2991104 bytes; optional] ? [y/n]: y
Install "uucp programs" [270336 bytes; optional] ? [y/n]: y
Install "System V programs and libraries" [4481024 bytes; optional] ? [y/n]: y
Install "Manual Pages" [6072320 bytes; optional] ? [y/n]: y
Install "Demonstration Programs" [2790400 bytes; optional] ? [y/n]: y
Install "Games" [2468864 bytes; optional] ? [y/n]: n
Install "Versatec" [6117376 bytes; optional] ? [y/n]: y
Install "SunOs Security Features" [146432 bytes; optional] ? [y/n]: y
```

1. The INSTALL program asks you if you want to load shared objects in shared locations. This is useful only when loading more than one architecture type, such as loading both Sun-3 and Sun-4 architectures on the same CONVEX server. In this case, answer **n**, as shown below.

   ```
   Load sharable objects in shared location ? [y/n] : n
   ```

If you answer **y** to the previous prompt, the INSTALL program automatically loads shared object files (e.g., man pages, nroff macro sets, etc.) into a directory called /export/exec/share. Then the directories /export/exec/sun3 and /export/exec/sun4 (for Sun-3 and Sun-4 architectures, respectively) are created automatically as links to /export/exec/share.

By answering **n** to the prompt, you direct the INSTALL program to load the object files directly into the /export/exec/sun3/share or /export/exec/sun4/share directories.

2. This completes the first phase of the procedure—loading the Sun-3 architecture. The INSTALL program supports loading more than one architecture at a time;

therefore, it returns to ask for the next architecture type to load. To proceed, respond with **done**, as shown below.

```
Enter next architecture type to load

[ sun3 | sun4 | ... | continue | done ]: done
```

3. The next phase of the INSTALL procedure is to set up the root and swap files for any clients that are ready to be installed. You can skip this phase by specifying **done**. For the sake of illustration, however, specify the host name **jogger**, as shown below.

```
Enter a sun3 client name ? [ name | done ]: jogger
Verifying ip address...   192.18.44.130 jogger
Verifying ethernet address...   8:0:20:0:f:ad jogger
```

The INSTALL program checks the /etc/hosts file (or the hosts database if YP is running) to verify that jogger is a known host. INSTALL also checks the /etc/ethers file (or the ethers map) to determine the Ethernet address of jogger.

➜ Note

The /etc/hosts and /etc/ethers files must be updated prior to invoking the INSTALL program. If they are not, the program fails to find the needed information and prompts for another client host name. You can suspend the procedure or login from another terminal to fix the files and try again. Or you can enter done and add clients using the setup_client command after the INSTALL program has finished running.

4. The INSTALL program asks for the yellow pages type for the client. This example procedure assumes that yellow pages is running on the CONVEX server, so the appropriate response is **client**.

```
Enter yp type of jogger ? [ master | slave | client | none ]:
client
```

5. The INSTALL program next requests information about jogger. You are prompted for the swap size, root directory, swap and dump file, and home directory. CONVEX suggests using the values listed below as initial default values. The swap size of 12 Mbytes can be easily increased later if necessary.

```
Enter swap size of jogger ? 12m
Enter root pathname of jogger ? /export/root
Enter swap pathname of jogger ? /export/swap
Enter dump pathname of jogger (or "none") ? none
Enter home pathname of jogger (or "none") ? none
```

In the previous example, the "dump pathname" and "home pathname" queries are answered **none** because this sample installation does not use the /home partition and the dump pathname defaults to the swap partition /export/swap. The pathnames **/export/dump** and **/export/home** may also be used.

6. Next, you are asked to confirm that the information you've specified is correct.

```
Information for jogger ok ? [y/n] : y
```

Then, you are asked to enter a new client name, if you want to add another client. More clients may be added at this time by specifying another client host name and repeating steps 4-6. In this sample procedure, no other clients are added.

```
Enter a sun3 client name ? [ name | done ]: done
```

If you answer **n** to the "Information for jogger ok" request, the program prompts you again to specify the file swap size, root pathname, swap pathname, and so on.

1. The INSTALL program prompts you to start installing the software. Enter **y** to begin the installation, as shown in the next screen. Installing all of the optional software takes approximately two hours—about one hour for each tape. The next prompt you see will be to load the second release tape.

```
Are you ready to start the installation ? [y/n] : y
Beginning Installation for the sun3 architecture.
Installation of sun3 executable files begins :
[Loading version 4.0 of sun3 architecture.]
Loading prototype root tree...
Extracting "usr" files from "/dev/rmt12" release tape.
Extracting "Sys" files from "/dev/rmt12" release tape.
Extracting "Networking" files from "/dev/rmt12" release tape.
Extracting "Debugging" files from "/dev/rmt12" release tape.
Extracting "SunView_Users" files from "/dev/rmt12" release tape.
```

After approximately an hour, the INSTALL procedure has extracted the files from the first tape.

2. You are then prompted to load the second tape and press **RETURN**.

```
Tape loaded is #1
Load release tape #2 for architecture sun3 and hit RETURN:
```

The output appears as follows.

```
Extracting "SunView_Programmers" files from "/dev/rmt12" release tape.
Extracting "SunView_Demo" files from "/dev/rmt12" release tape.
Extracting "Text" files from "/dev/rmt12" release tape.
Extracting "Install" files from "/dev/rmt12" release tape.
Extracting "User_Diag" files from "/dev/rmt12" release tape.
Extracting "SunCore" files from "/dev/rmt12" release tape.
Extracting "uucp" files from "/dev/rmt12" release tape.
Extracting "System_V" files from "/dev/rmt12" release tape.
Extracting "Manual" files from "/dev/rmt12" release tape.
Extracting "Demo" files from "/dev/rmt12" release tape.
Extracting "Versatec" files from "/dev/rmt12" release tape.
Extracting "Security" files from "/dev/rmt12" release tape.
Installation of sun3 executable files completed.
```

The INSTALL program sets up the clients and updates the associated administrative files. The output appears as follows.

```
Starting installation of sun3 clients...
Start creating sun3 client "jogger" :
Updating bootparams ...
Creating root for client "jogger".
Creating 12m bytes of swap for client "jogger".
Setting up /tftpboot directory.
Completed creating sun3 client "jogger".
Updating bootparams YP map...
updated bootparams
pushed bootparams
Diskless Client Installation Completed.
```

The system prompt returns.

```
#
```

Booting a Sun

This section describes the steps required to boot jogger, the Sun workstation configured in the previous section. In general, booting is very simple and can be achieved by either cycling the power to the Sun workstation or by simply typing **b** to the Sun PROM monitor.

This sample boot procedure assumes the following:

- The Sun-3 has been connected to the Ethernet and is functional.
- The Sun-3 is turned on and the Sun PROM monitor prompt (>) is displayed.
- The Sun-3 architectural support programs and root directory for jogger have been loaded as described in the previous section.
- The Sun-3 is on the same subnet as the CONVEX server where the Sun-3 architectural support programs are loaded.
- The name of the CONVEX server is conserve and it is in the yellow pages domain convex.
- NFS on the CONVEX machine is functional, and the rpc.bootparamd server is running or installed in the /etc/inetd.conf file.

The booting procedure is simple and requires only minimal input. Once the Sun has been instructed to boot, booting multiuser mode is completely automatic and quick. It should take less than five minutes to boot diskless Sun workstations.

1. Ensure that the Sun-3 has power and the following Sun PROM monitor prompt is displayed.
   >

2. Enter **b** to boot the Sun directly to multi-user operation, as shown in the next screen.
   > **b**

The boot process continues automatically. The Sun determines its Internet protocol (IP) address and downloads a boot program from the CONVEX server conserve using tftp. The output appears as follows.

```
Using IP Address 192.18.44.130 = C0122C82
Boot: le(0,0,0)
Booting from tftp server at 192.18.44.122 = C0122C7A
Downloaded 126056 bytes from tftp server.
```

In the previous screen, the tftp server address 192.18.44.122 is the IP address of the Convex NETdisk server, conserve, responding to the tftp request to download the boot program. The address of 192.18.44.130 is the booting Sun's IP address determined from the server using Reverse ARP (RARP) protocols.

The C0122C82 address is the uppercase hex value of this IP address, where 192=C0, 18=12, 44=2C, and 130=82: thus, 192.18.44.130=C0122C82. The tftp protocol requests this uppercase hex string, which serves as the filename for the boot program, to download from the server.

The boot program is then executed. It again determines the correct IP address to use, connects to the rpc.bootparamd server on conserve, and uses the NFS protocols to download the vmunix kernel. The output appears as follows.

```
Using IP Address 192.18.44.130 = C0122C82
hostname: jogger
domainname: convex
server name 'conserve'
root pathname '/export/root/jogger'
root on conserve:/export/root/jogger fstype NFS
Boot: vmunix
Size: 632768+115120+16076 bytes
```

The lines below show the output as the Sun kernel is executed and the system comes up in multiuser mode.

```
SunOS Release 4.0 (GENERIC) #5: Thu Apr 14 19:24:56 PDT 1988
Copyright (c) 1988 by Sun Microsystems, Inc.
mem = 4096K (0x400000)
avail mem = 2949120
Ethernet address = 8:0:20:0:f:ad
si0 at obio 0x140000 pri 2
sd0 at si0 slave 0
sd1 at si0 slave 1
sd2 at si0 slave 8
sd3 at si0 slave 9
st0 at si0 slave 32
st1 at si0 slave 40
zs0 at obio 0x20000 pri 3
zs1 at obio 0x0 pri 3
le0 at obio 0x120000 pri 3
bwtwo0 at obmem 0x100000 pri 4
bwtwo0: resolution 1152 x 900
hostname: jogger
domainname: convex
root on conserve:/export/root/jogger fstype NFS
swap on conserve:/export/swap/jogger fstype NFS size 10240K
dump on conserve:/export/swap/jogger fstype NFS
checking filesystems
Automatic reboot in progress...
Thu Sep  1 07:40:26 PDT 1988
checking quotas: done
starting rpc and net services: portmap ypbind keyserv routed.
mount: conserve:/export/exec/sun3 already mounted
rdate conserve
starting additional services: biod
starting system logger
starting local daemons: auditd sendmail statd lockd.
link-editor directory cache
preserving editor files
clearing /tmp
standard daemons: update cron uucp.
starting network daemons: inetd printer.
Thu Sep  1 07:41:22 PDT 1988
```

Finally, the system displays the login prompt for jogger.

```
jogger login:
```

Adding and Removing NETdisk Clients

This section describes using the setup_client program to add and remove NETdisk clients.

The INSTALL program may be used interactively to load client architecture support onto a CONVEX server. Once the client architecture support has been loaded, however, it may be more convenient to use the `setup_client` program to add new clients or remove existing ones. The `setup_client` program is also found in the /usr/etc/install directory. For more information on the `setup_client` program, refer to the setup_client(8) man page.

The `setup_client` program is not an interactive one like INSTALL. Instead all information is specified on the command line and the process of adding and removing the client happens without confirmation. You can display the expected options to `setup_client` by entering **`setup_client`** with no arguments.

To remove the client, jogger, that was added in the previous section, use the following procedure:

1. Log in as superuser.
2. Change your current working directory to /usr/etc/install, as shown in the next screen.

   ```
   # cd /usr/etc/install
   ```

3. Enter **`setup_client`** to determine the arguments it expects. When you enter this command with no arguments, the program responds with a list of options it expects to see specified on the command line.

```
# setup_client
setup_client: incorrect number of arguments.
usage:
setup_client op name yp size rootpath swappath dumppath
homepath execpath sharepath arch
where:
op          = "add" or "remove"
name        = name of the client machine
yp          = "master" or "slave" or "client" or "none"
size        = size for swap
(e.g. 16M or 16m ==> 16 * 1048576 bytes
16K or 16k ==> 16 * 1024 bytes
16B or 16b ==> 16 * 512 bytes
16         ==> 16 bytes )
rootpath    = pathname of root (e.g. /export/root )
swappath    = pathname of swap (e.g. /export/swap )
dumppath    = pathname of dump (e.g. /export/dump ) or "none"
homepath    = pathname of home (e.g. /home or homeserver:/home) or "none"
execpath    = full pathname of exec directory (e.g. /export/exec/sun3 )
sharepath   = full pathname of shared files (e.g. /export/exec/share or "none"
arch        = "sun3" or "sun4" etc
```

To execute `setup_client`, include the appropriate arguments in the order specified in the previous output.

To remove the client, jogger, which was added in the section "Using INSTALL to Load Software," use the `setup_client` program as shown below.

```
# setup_client remove jogger client 10m /export/root /export/swap none none \
 /export/exec/sun3 none sun3
```

The output appears as follows.

```
Start removing sun3 client "jogger" :
Updating bootparams ...
updated bootparams
pushed bootparams
Removing root for client "jogger".
Removing swap for client "jogger".
Completed removing sun3 client "jogger".
```

To add the client, jogger, use the `setup_client` program as shown below.

```
# setup_client add jogger client 10m /export/root /export/swap none none \
 /export/exec/sun3 none sun3
```

The following lines illustrate sample output from the previous command.

```
Start creating sun3 client "jogger" :
Updating bootparams ... updated bootparams pushed bootparams
Creating root for client "jogger".
Creating 10m bytes of swap for client "jogger".
Setting up /tftpboot directory.
Updating /etc/exports to export "jogger" info.
Completed creating sun3 client "jogger".
```

Making Mount Points

Mount points are locations within a directory tree through which your computer accesses directories. These directories can be mounted locally from a disk or tape, or remotely from another computer on the network. For example, when you use the restore command, you mount the files on the backup tape through /mnt, an already existing mount point in the root file system. Furthermore, when the installation script creates a client's root swap and home directories in a server's /export file system, it also creates the mount points /, /swap, and /home for them in the client's directory tree. If the client must mount additional directories, you need to create mount points for them.

Mount points are essentially empty directories, which you create through the mkdir command. Simply enter

```
% mkdir mount_point
```

where mount_point is the full pathname of the empty directory you want to create. You need to create mount points for any nondefault directories mounted through the /etc/fstab file or through the `mount` command.

An NFS Client's fstab File

When a server is first set up, the installation script creates default fstab files for each of the server's clients. The default client /etc/fstab file resembles the following example lines, set up by server dancer for client raks:

```
dancer:/export/root/raks / nfs rw 0 0
dancer:/export/swap/raks / nfs rw 0 0
dancer:/usr /usr nfs ro 0 0
dancer:/home/dancer /home/dancer nfs rw 0 0
```

Entries in the file have the following syntax:

```
server:server_dir client_mountpoint type options freq
pass
```

Here is how these parameters apply to the sample fstab file above.

- server
  This is the name of the server exporting the directory the client wants to mount. In the sample file, dancer is the only server listed.
- server_dir
  This is the name of the directory to be mounted from the indicated server. In the first entry, the directory is /export/root/raks, client raks' individual root directory. Note that in the default fstab file for the client, the only mounted directories are those that the client requires to operate: its individual root, swap, and home, and /usr which, along with the client root contains all essential programs.
- client_mountpoint
  This is the mount point on the client through which it accesses directories mounted from the server. In the first entry in the example /etc/fstab file above, the mount point is the client's root directory.
- type
  This is the type of mount taking place. In the example above, this is an nfs mount. If the client has its own disk, you can also set up its fstab file so that it can perform BSD 4.3 mounts of file systems on its local disk.
- options
  This can be any one of a number of options provided for nfs or BSD 4.3 mounts. Refer to the fstab(5) man page for a complete list. For example, if you are running secure, you may want to select the secure option for nfs mounts.

  The example file shows the client performing nfs mounts with the common options -rw or -ro. The client can mount its own root swap and home directories with read-write access. However, it accesses the server's /usr directory with read-only permission; thus a user on the client cannot add or modify a file in /usr.
- freq
  This is the interval in days between dumps of the directory.
- pass
  This indicates the fsck pass in which the listed file system is checked. If a zero is indicated, the file system is not checked during `fsck`.

The contents of /etc/fstab remain the same until you change them.

Adding New Users

When you first add a new user, you must create an entry for them in their machine's /etc/passwd file, then create a home directory for them. If their machine is running YP, go to Chapter 4 and follow the instructions under "How to Set Up a YP Client." If their machine is not running YP, do the following:

1. Obtain basic information from the user, such as preferred user name (which should be unique within your network domain), full name to be displayed in mail headers, and preferred login shell.
2. Become superuser and access the /etc directory of the person's machine, for example, raks.

   ```
   #cd /export/root/raks/etc
   ```

3. Edit the local passwd file, using a text editor.

   a. Create an entry for the new user on a separate line.

   b. Type the user's requested login name, for instance,

   ```
   shamira
   ```

The colon after the user name is the delimiter used in the passwd file to indicate the end of a field.

4. Leave the password field blank by typing another colon, as in:

   ```
   shamira::
   ```

5. Add a user ID for shamira, according to your company's policies. Do not use a user ID that already exists in the local password file. Especially do not use 0 or 1, the user IDs for root and daemon. If your company does not have a policy for assigning user IDs, you have to create one. As one suggestion, some companies use a person's employee number in the user ID field of /etc/passwd. Follow the user ID with a colon.
6. Add a group ID number for the group you want the person to be in. If the user will not belong to a group, just type a colon to leave the field blank.
7. Type the person's name as he or she requested for the mail header. Follow it with a colon.
8. Type the full pathname of the person's home directory, which should be /home/server_name/user_name, and follow it with the delimiting colon.
9. Finally, type the user's preferred login shell. Here is an example of a complete entry for new user shamira:

   ```
   shamira::235:12:Marsha Wong:/home/dancer/shamira:/bin/csh
   ```

10. Close the /etc/passwd file.
11. Create a home directory for the new user.

    ```
    #cd /home/dancer
    #mkdir shamira
    #chown userid# shamira
    ```

12. Have the user log in to the client by supplying the new user name and then pressing **RETURN** when requested for a password.
13. Have the user run the passwd command to add a password to the login name.

Installing Optional Software After Running INSTALL

Adding and Removing NETdisk Clients illustrates installation of a Sun-3 client architectural support. All optional software packages except the **Games** file are loaded at that time. This section shows how to invoke the INSTALL program once again to extract optional software after the basic architectural support has been loaded.

All the assumptions stated in the "Adding and Removing NETdisk Clients" section apply to this installation session.

1. Log in as superuser.
2. Change your working directory to /usr/etc/install, as shown below.

```
# cd /usr/etc/install
```

3. Enter **INSTALL** as shown below to invoke the INSTALL program.

```
# INSTALL
```

4. The INSTALL program prompts you for the type of installation. Answer **local** as shown below.

```
Enter tape drive type ? [local | remote]: local
```

5. The INSTALL program then asks which tape to use. For a local installation on 1/2 inch magnetic tape, use **/dev/rmt12** since that is a non-rewind tape device. For tape unit 1, you would use /dev/rmt13.

```
Enter tape type ? [ar[08] | st[08] | mt[08] | xt[08] | /dev/???]: /
dev/rmt12
```

6. Next, enter the architecture type to load. This exercise assumes a Sun-3 architecture; therefore the appropriate response is **sun3**, as shown below.

```
Enter next architecture type to load

[ sun3 | sun4 | ... | continue | done ]: sun3
```

7. The INSTALL program asks where to load the executables. Supply the appropriate pathname. The previous sample installation loads the executables in /export/exec; therefore, this pathname is used in the following example.

```
Enter pathname for sun3 executables ? /export/exec
```

8. The INSTALL program notices that the /export/exec directory already exists and prompts you to confirm what is about to be done. This exercise is to install optional software; therefore, answer **use** to the program prompt, as shown below.

```
INSTALL: exec tree /export/exec/sun3 appears to already exist.
You may select one of the following options:
ignore  continue as if this architecture had not been specified
remove  remove existing exec tree, then continue
use     continue, loading any new optional software specified
upgrade continue, loading all new files unconditionally
clients continue with sun3 clients, but use existing exec tree? use
```

9. The INSTALL procedure tries to access the tape. (Be sure the tape is mounted and online.) Press **RETURN** to proceed.

```
Reading the table of contents for sun3 architecture...
Mount a sun3 release tape and hit RETURN, or enter "exit" :
```

10. The following lines show input to extract only the **Games** file. Follow this pattern of answers as appropriate to extract a different file.

```
Select optional software for the sun3 architecture:
Install "User File System" [20971520 bytes; required] ? [y/n]: n
Install "Sys" [2721792 bytes; desirable] ? [y/n]: n
Install "Networking tools and programs" [953344 bytes; desirable] ? [y/n]: n
Install "Debugging tools" [3383296 bytes; desirable] ? [y/n]: n
Install "SunView_Users Programs" [1453056 bytes; common] ? [y/n]: n
Install "SunView_Programmers Programs" [2064384 bytes; optional] ? [y/n]: n
Install "SunView_Demo Program source" [564224 bytes; optional] ? [y/n]: n
Install "Text Processing tools" [690176 bytes; optional] ? [y/n]: n
Install "Install tools" [1004544 bytes; optional] ? [y/n]: n
Install "User Level Diagnostics tools" [1491968 bytes; optional] ? [y/n]: n
Install "SunCore Libraries" [2991104 bytes; optional] ? [y/n]: n
Install "uucp programs" [270336 bytes; optional] ? [y/n]: n
Install "System V programs and libraries" [4481024 bytes; optional] ? [y/n]: n
Install "Manual Pages" [6072320 bytes; optional] ? [y/n]: n
Install "Demonstration Programs" [2790400 bytes; optional]? [y/n]: n
Install "Games" [2468864 bytes; optional] ? [y/n]: y
Install "Versatec" [6117376 bytes; optional] ? [y/n]: n
Install "SunOs Security Features" [146432 bytes; optional] ? [y/n]: n
Load sharable objects in shared location ? [y/n] : n
```

11. This procedure completes the loading of the **Games** file. Respond with **done** to the next two prompts to indicate that the no other architecture support is to be loaded and that no clients are to be loaded.

```
Enter next architecture type to load
[ sun3 | sun4 | ... | continue | done ]: done
Enter a sun3 client name ? [ name | done ]: done
```

12. The INSTALL program asks you if you're ready to start the installation. If so, answer **y**, as shown below.

```
Are you ready to start the installation ? [y/n] : y
```

13. The INSTALL program asks for confirmation once again before proceeding. In this case, choose **ignore** to load the optional software directly on top of the existing files.

```
Beginning Installation for the sun3 architecture.
/usr/etc/install/setup_exec: /export/exec/sun3 already exists.
You may select one of the following options:
abort    exit with error status
ok       exit with ok status
remove   remove existing tree, then continue
ignore   continue (load on top of existing tree)? ignore
```

14. The INSTALL program realizes that the **Games** file is not on the first tape and prompts for the second tape to be loaded. Load the tape, then press **RETURN**

```
Installation of sun3 executable files begins :
[Loading version 4.0 of sun3 architecture.]
Tape loaded is #1
Load release tape #2 for architecture sun3 and hit RETURN:
```

The output from the previous command appears as follows.

```
Extracting "Games" files from "/dev/rmt12" release tape.
Installation of sun3 executable files completed.
```

Because no new clients are to be loaded, the procedure to install **Games** is complete, and the system prompt returns.

```
Starting installation of sun3 clients...
Diskless Client Installation Completed.

#
```

Important NETdisk Files and Directories

The following files and directories are significant to the NETdisk release.

- /etc/exports
  Automatically updated by the INSTALL program, this file specifies the export options used by the client root directory, swap files, and executables.
- /etc/bootparams
  Automatically updated by the diskless client installation scripts INSTALL and `setup_client`. This file is used by the rpc.bootparamd program, which supplies information about a client's host name and location of its root, swap, and dump files. A new yellow pages map is also produced for this file.
- /usr/etc/install
  Directory that holds the INSTALL and `setup_client` programs. When executing these programs, be sure that your current working directory is /usr/etc/install and that "." is in your PATH environment variable.
- /tftpboot
  Directory containing the boot programs that are downloaded using the tftp protocol when a diskless client is booting. For example, the boot program for a Sun-3 machine is named /tftpboot/boot.sun3. The diskless clients download the file that is their Internet address converted into uppercase hex numbers. This file is typically a symbolic link to the boot.sun? file (where sun? is the name of the sun client architecture). For example, the client named jogger has an Internet address of 192.18.44.130. This number converted to hex is C0122C82 (i.e., 192=C0, 18=12, 44=2C, and 130=82). The client downloads the file /tftpboot/C0122C82, which is a symbolic link to tftpboot/boot.sun3, and executes it.
- /export/exec
  This directory contains the architecture specific programs and libraries (essentially, the /usr file system) that are shared among similar clients. For example, the programs for a Sun-3 machine are stored in the /export/exec/sun3 directory.
- /export/root
  This directory contains the root file system for each client that is served from this server. The client, jogger, finds its root file system in the /export/root/jogger directory.
- /export/swap
  This directory contains the files that clients use for swapping. The swap file for the client, jogger, is /export/swap/jogger.
- /export/dump
  This directory is typically unused, but can be specified to hold the crashdump image of a crashed client. Normally, the image is written to the swap file.

Theory of Operation

This section explains, in general terms, the mechanics of booting a diskless workstation. The general booting procedure is further discussed in the boot man page sunboot(8s), which supplements this description.

Booting consists of several interacting procedures:

- Reverse ARP
- tftp "get" of the boot program
- bootparams query to discover important identity information
- NFS mount and access for loading operating system.

The server daemons handling these requests in NETdisk servers are, respectively: kernel Reverse ARP handler, `in.tftpd`, `rpc.bootparamd`, `rpc.mountd`, and `nfsd`.

These operations work as follows: The client, when booting, knows only its Ethernet address. This address is used in a broadcast Reverse ARP request to discover its Internet address. The NETdisk server responds to this request with the client internet address, found in the server's arp cache. The arp cache is a kernel table of Internet/ethernet address mappings.

Once the client has its IP address, it sends a tftp "get" request of a file named the same as its IP address in uppercase hex characters. Some clients also append a dot and uppercase architecture name (such as .SUN4) to the IP address in this request, but the in.tftpd daemon handles this peculiarity.

This initial tftp request is first directed to the server that answers the Reverse ARP query, but the request is broadcast on the local network if no response is received from that server.

The file to be downloaded via tftp is the actual boot program. So that there is no need for information passed between booting phases, the boot program also does a Reverse ARP request as above. Boot then sends a bootparams "whoami" request (directed first to the responding server, then broadcast, as with tftp. The result of this remote procedure call tells the client its host name, domain name, and IP address of an acceptable IP router. A second bootparam procedure, "getfile," is then used to determine the NFS directory of the operating system image (i.e., the client's root pathname). This pathname is used in an NFS mount request to get the directory file handle, which is then used in normal NFS lookup and read procedures to load the kernel (vmunix by default).

Once the kernel is loaded and running, it does Reverse ARP and bootparam requests. In addition to getting the pathname for the root, the kernel also gets pathnames for swap and dump. Default keys used in the "getfile" requests are root, swap, and dump. To override these keys, use the -a option with the Sun PROM monitor boot command as follows.

```
>b le() -a
```

You will then be prompted for the name of the root device.

Using the Automounter 3

Introduction

The `automount` program enables users to mount and unmount remote directories on an as-needed basis. When a user on a client machine running the automounter invokes a command that accesses a remote file or directory, such as opening a file with an editor, the automounter mounts the hierarchy to which that file or directory belongs. The automounter leaves the file or directory mounted as long as it is needed, but automatically unmounts it if it is not accessed for a certain amount of time. No mounting occurs at boot-time, and users no longer must know the superuser password to mount a directory. It is all done automatically and transparently.

Mounting some file hierarchies with `automount` does not exclude the possibility of mounting others with mount. A diskless machine must mount /export/root, /export/swap, /usr and /export/exec/kvm through the mount program and /etc/fstab file.

The next section explains how the automounter works. It also contains instructions for setting it up.

How the automounter works

Unlike `mount`, `automount` does not consult /etc/fstab for a list of hierarchies to mount. Rather, it consults a series of maps, which can be either direct or indirect. The names of the maps can be passed to `automount` from the command line, or from another (master) map.

➜ Note

The following explanation is intended for advanced system administrators and programmers. If you do not have this level of experience, read the summary at the end of this subsection for an overview of automounter tasks.

The automounter mounts all files under /tmp_mnt and provides a symbolic link from the requested mount point to the actual mount point under /tmp_mnt. For example, if a user wants to mount a remote directory src under /usr/src, the actual mount point is /tmp_mnt/usr/src and /usr/src is a symbolic link to that location. Note that, as with any other kind of mount, a mount affected through the automounter on a nonempty mount point hides the original contents of the mount point for as long as the mount is in effect.

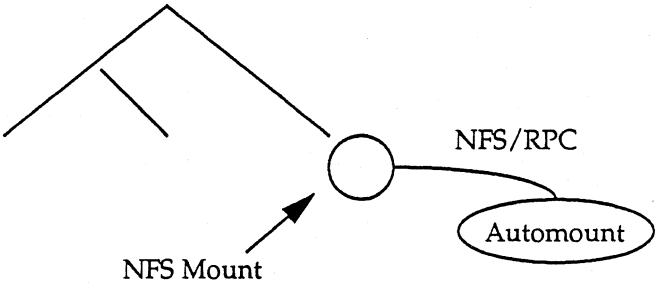
There are two distinct stages in the automounter's actions:

- The initial stage, boot time, when rc.local starts the automounter.
- The mounting stage, when a user tries to access a file or directory in a remote machine.

When rc.local invokes `automount`, `automount` opens a UDP socket and registers it with the rpcbind service as an NFS server port. Next, `automount` forks off a server daemon to listen for NFS requests on the socket. The parent process then mounts the daemon at its mount points within the file system (as specified by the maps). Through the mount system call automount passes the server daemon's port number and an NFS file handle that is unique to each mount point. The arguments to the mount system call vary according to the kind of file system; for NFS file systems, the call is

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

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

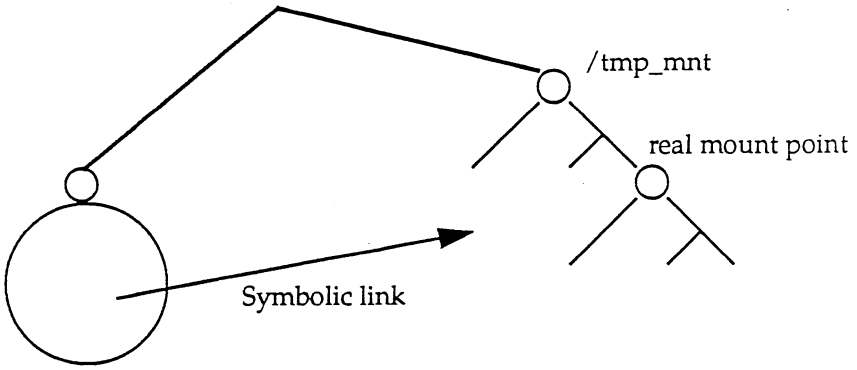


In the second stage, when the user actually requests access to a remote file hierarchy, the daemon intercepts the kernel NFS request and looks up the name in the map associated with the directory. Taking the location (server:pathname) of the remote file system from the map, the daemon then mounts the remote file system under the directory /tmp_mnt. It answers the kernel, telling it it is a symbolic link. The kernel sends an NFS READLINK request, and the automounter returns a symbolic link to the real mount point under /tmp_mnt.

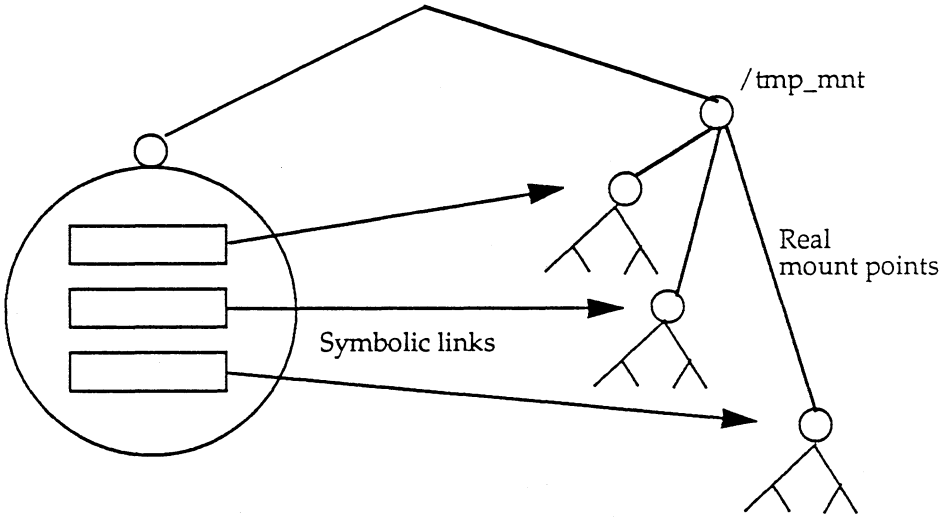
The behavior of the automounter is affected by whether the mount-point name is found in a direct or an indirect map.

- If the name is found in a direct map, the automounter emulates a symbolic link, as stated above. It responds as if a symbolic link exists at its mount point. In response to a GETATTR, it describes itself as a symbolic link. When the kernel follows up with a READLINK, the automounter returns a path to the real mount point for the

remote hierarchy in /tmp_mnt.



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



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

Summary

When `automount` is called from the command line or rc.local, `automount` forks a daemon to serve each mount point in the maps, making the kernel believe that the mount has taken place. The daemon sleeps until a request is made to access the corresponding file hierarchy. At that time the daemon does the following:

1. Intercepts the request
2. Mounts the remote file hierarchy
3. Creates a symbolic link between the requested mount point and the actual mount point under /tmp_mnt

4. Passes the symbolic link to the kernel
5. Unmounts the file hierarchy after a predetermined amount of time has passed with the link not being touched (usually, five minutes).

Preparing the Maps

A server never knows, nor cares, whether the files it shares are accessed through `mount` or `automount`. Therefore, you need not do anything different on the server for `automount` than for `mount`.

A client, however, needs special files for the automounter. As mentioned previously, `automount` does not consult /etc/fstab. Rather, it consults the map file(s) specified at the command line (refer to the "Invoking `automount`" section). All automounter maps are located in /etc. All their names contain the prefix auto.

There are three kinds of `automount` maps:

- master
- indirect
- direct

They are described below.

The Master Map

The master map contains references, direct or indirect, to all of the file systems the automounter mounts. Each line in a master map, by convention called /etc/auto.master, has the syntax:

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

where:

- mount-point is the full pathname of a directory.
- map is the name of the map the automounter should use to find the mount points and locations.
- mount-options is an optional, comma separated list of options that regulate the mounting of the entries mentioned in the map unless the map entries list other options. (Options listed in the map entries override options specified in this mount-options list.)

A simple, one-line master map for map auto.home would look like:

```
/home      /etc/auto.home   -rw,intr,secure
```

A line whose first character is # is treated as a comment; everything that follows until the end of line is ignored. A backslash (\) at the end of line permits splitting long lines into shorter ones.

Direct and Indirect Maps

Lines in direct and indirect maps have the syntax:

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

where

- key is the pathname of the mount point.
- The mount-options are the options you want to apply to this particular mount.
- location is the location of the resource, specified as server:pathname.

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

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

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

and the following would be an entry in an indirect map:

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

Use indirect maps to organize portions of a remote file system that you may want to move later. This allows you to edit the mount point for a single indirect map, rather than all of the files in that map, should you wish to reorganize your mount tree.

As you can see, the key in the indirect map is begging for more information: where is the mount point parsley really located? That is why you must either provide that information at the command line or through another map. For example, if the above line is part of a map called /etc/auto.veggies you would have to call it up either as:

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

or specify, in the master map:

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

In either case, you are associating a mount directory veggies with the entries (parsley in this case) mentioned in the indirect map /etc/auto.veggies. The end result is that the hierarchy /usr/greens from the machine veggies is mounted on /veggies/parsley when needed.

Writing a Master Map

As stated above, the syntax for each line in the master map is

```
Mount-point      Map    [ Mount-options ]
```

A typical auto.master file would contain

```
#Mount-point    Map              Mount-options

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

Note that /etc/auto.home is the name of an indirect map and /etc/auto.direct is the name of a direct map. The automounter recognizes some special mount points and maps, which are explained below.

Mount point /net

In this example, the automounter mounts under /net all the entries under the special map -hosts. This is a built-in map that does not use any external files except the hosts database /etc/hosts (or hosts.byname YP map). Notice that because the automounter does not mount the entries until needed, the specific order is not important.

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. Those entries are mounted under /tmp_mnt/home and a symbolic link is provided between /home/directory and /tmp_mnt/home/directory.

Mount point /-

The mount point /- is a filler that tells the automounter to reference /etc/auto.direct, but not to associate the entries in /etc/auto.direct with any directory. The automounter then uses the mount points listed in the map. (Remember, in a direct map the key is a full pathname.)

How the Automounter Works

After the `automount` daemon is in place, a user entering the command

```
example $ cd /net/gumbo
```

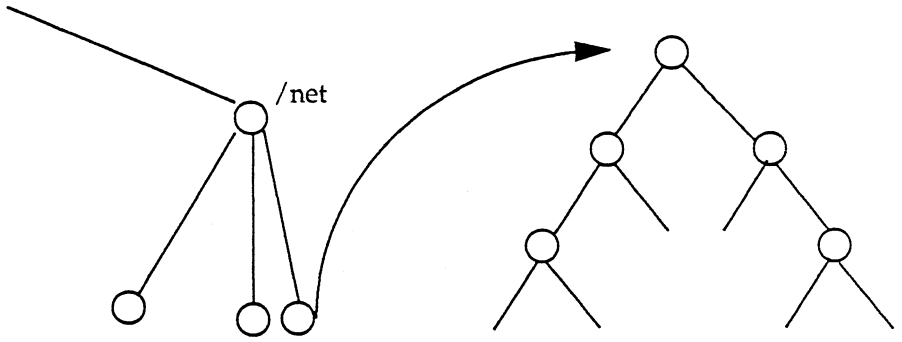
changes directory to the top of the hierarchy of files (i.e., the root file system) of the machine gumbo as long as the machine is in the hosts database. However, the user may not see all files and directories under /net/gumbo. This is because the automounter can mount only the shared file systems of host gumbo in accordance with the restrictions placed on the sharing.

When you issue the example command above, the automounter does the following:

1. ping the null procedure of the server's mount service to see if it's alive
2. Request the list of shared hierarchies from the server
3. Sort the shared list according to length of pathname (to ensure proper mounting order):
   ```
   /usr/src
   /export/home
   /usr/src/sccs
   /export/root/blah
   ```
4. Proceed down the list, mounting all the file systems at mount points in /tmp_mnt

(creating the mount points as needed).

5. Return a symbolic link that points to the top of the recently mounted hierarchy.



Note that for the -hosts map the automounter must mount all the file systems that the server in question advertises for sharing. Even if the request is as follows:

```
example $ ls /net/gumbo/usr/include
```

the automounter mounts all gumbo shared systems, not just /usr.

In addition, unmounting that occurs after a certain amount of time has passed works from the bottom up. This means if one of the directories at the top is busy, the automounter must remount the hierarchy and try again later.

Nevertheless, the -hosts special map provides a convenient way for users to access directories in many different hosts without having to use `rlogin` or `rsh`. (These remote commands have to establish communication through the network every time they are invoked.)

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

Writing a Direct Map

The syntax for a direct map is:

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

where:

- key is the full pathname of the mount point. (Remember that in an indirect map the key is not a full pathname.)
- mount-options are optional but, if present, override — for the entry in question — the options of the calling line, if any, or the defaults. (Refer to the "Invoking `automount`" section later in this chapter.)
- location is the location of the resource, specified as server:pathname.

Of all the maps, the entries in a direct map most closely resemble, in their simplest form, corresponding entries in /etc/fstab. An entry that appears in /etc/fstab as:

```
dancer:/usr/local /usr/local/tmp nfs ro 0 0
```

appears in a direct map as:

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

The following is a typical /etc/auto.direct map:

```
/usr/local \
            /bin        -ro,soft      ivy:/export/local/sun3 \
            /share      -ro,soft      ivy:/export/local/share \
            /src        -ro,soft      ivy:/export/local/src \
/usr/man                '-ro,soft     oak:/usr/man \
                                      rose:/usr/man \
                                      willow:/usr/man
/usr/games              -ro,soft      peach:/usr/games
/usr/spool/news         -ro,soft      pine:/usr/spool/news
/usr/frame              -ro,soft      redwood:/usr/frame1.3 \
                                      balsa:/export/frame
```

Notice a couple of unusual features in this map. These are the subject of the next two subsections.

Multiple Mounts

A map entry can describe a multiplicity of mounts, where the mounts can be from different locations and with different mount options. Consider the first entry in the previous example. It is, in fact, one long entry whose readability has been improved by splitting it into three lines by using the backslash and indenting the continuation lines. This entry mounts /usr/local/bin, /usr/local/share, and /usr/local/src from the server ivy with the options read-only and soft. The entry could also read:

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

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

Multiple mounts can be hierarchical. When file systems are mounted hierarchically, each file system is mounted on a subdirectory within another file system. When the root of the hierarchy is referenced, the automounter mounts the whole hierarchy. The concept of root here is very important. The symbolic link returned by the automounter to the kernel request is a path to the mount root. This is the root of the hierarchy that is mounted under /tmp_mnt. This mount point should theoretically be specified:

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

In practice, it is not specified because in a trivial case of a single mount as above, it is assumed that the location of the mount point is at the mount root or '.'. Therefore, instead of the above, it is acceptable, indeed, preferable, to enter

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

However, the mount point specification becomes important when mounting a hierarchy: here the automounter must have a mount point for each mount within the hierarchy. The example above is a good illustration of multiple, nonhierarchical mounts under /usr/local when the latter is already mounted.

The following illustration shows a true hierarchical mounting:

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

→ Note

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

The mount points used here for the hierarchy are /, /bin, /share, and src. Note that these mount-point paths are relative to the mount root, not the host's file system root. The first entry in the example above has / as its mount point. It is mounted at the mount root. There is no requirement that the first mount of a hierarchy be at the mount root. The automounter issues mkdir to build a path to the first mount point if it is not at the mount root.

Multiple Locations

In the example for a direct map, which was:

```
/usr/local \
            /bin        -ro,soft    ivy:/export/local/sun3 \
            /share      -ro,soft    ivy:/export/local/share \
            /src        -ro,soft    ivy:/export/local/src
/usr/man                -ro,soft    oak:/usr/man \
                                    rose:/usr/man \
                                    willow:/usr/man
/usr/games              -ro,soft    peach:/usr/games
/usr/spool/news         -ro,soft    pine:/usr/spool/news
/usr/frame              -ro,soft    redwood:/usr/frame1.3 \
                                    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 that the mounting can be done from any of the replicated locations. This procedure makes sense only when you are mounting a hierarchy read-only, since theoretically you would like to have some control over the locations of files you write or modify.

A good example is the man pages. In a large network, more than one server may export the current set of manual pages. It does not matter which server you mount them from, just so long as the server is up and running and sharing its file systems. In the example above, multiple mount locations are expressed as a list of mount locations in the map entry:

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

This could also be expressed as a list of servers, separated by commas, and followed by a colon and the pathname (as long as the pathname is the same for all the replicated servers):

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

Here you can mount the man pages from the servers oak, rose, or willow. From this list the automounter first selects servers on the local network and pings these servers. This launches a series of RPC requests to the null procedure of the mount service in each server. (Note that the list does not imply any ordering.) The first server to respond is selected, and an attempt is made to mount from it.

This redundancy, which is useful in an environment where individual servers may or may not be sharing their file systems, exists only at mount time. There is no status checking of the mounted-from server by the automounter once the mount occurs. If the server goes down while the mount is in effect, the file system becomes unavailable. An option here is to wait five minutes until the autounmount takes place and try again. The next time, the automounter will choose one of the available servers. Another option is to use the umount command, inform the automounter of the change in the mount table (as specified in the section, "The Mount Table"), and retry the mount.

Writing an Indirect Map

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

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

where key is the name (not the full pathname) of the directory to be used as mount point. When the automounter obtains the key, it appends the key to its associated mount point. If the mount point is not provided on the command line, the automounter takes the mount point from the master map that invoked the indirect map.

For example, one entry in the master map presented above as an example, reads:

```
/home            /etc/auto.home  -rw,intr,secure
```

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

A typical auto.home map might contain:

#key	mount-options	location
willow		willow:/home/willow
cypress		cypress:/home/cypress
poplar		poplar:/home/poplar
pine		pine:/export/pine
apple		apple:/export/home
ivy		ivy:/home/ivy
peach	-rw,nosuid	peach:/export/home

As an example, assume that the map above is on host oak. If user laura has an entry in the password database specifying her home directory as /home/willow/laura, then, whenever she logs into machine oak, the automounter mounts (as /tmp_mnt/home/willow) the directory /home/willow residing in machine willow. If one of the subdirectories is indeed laura, user laura will be in her home directory, which is mounted read/write, interruptible, and secure.

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

Now, assume the following conditions occur:

- User laura's home directory is listed in the password database as /home/willow/laura
- Machine willow shares its home hierarchy with the machines mentioned in auto.home
- All those machines have a copy of the same auto.home and the same password database.

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

Furthermore, now laura can also enter the command

```
% cd ~brent
```

and the automounter will mount brent's home directory for her (if permissions allow it).

On a network without YP, you must change all relevant databases (such as /etc/passwd) on all systems on the network in order to accomplish this. On a network running YP, propagate all the relevant databases throughout the network to ensure this.

Specifying Subdirectories

The previous subsection, Writing an Indirect Map, showed the following typical auto.home file:

```
#key        mount-options    location
willow                       willow:/home/willow
cypress                      cypress:/home/cypress
poplar                       poplar:/home/poplar
pine                         pine:/export/pine
apple                        apple:/export/home
ivy                          ivy:/home/ivy
peach       -rw,nosuid       peach:/export/home
```

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

```
#key        mount-options    location
john                         willow:/home/willow/john
mary                         willow:/home/willow/mary
joe                          willow:/home/willow/joe
```

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

```
% ls ~john ~mary
```

the automounter performs the equivalent of the following actions:

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

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

However, the complete syntax of a line in a direct or indirect map is actually:

key [mount-option] server:pathname[:subdirectory]

Until now, you used the form server:pathname to indicate the location. This is an ideal place for you to also indicate the subdirectory, like this:

#key	mount-options	location
john		willow:/home/willow:john
mary		willow:/home/willow:mary
joe		willow:/home/willow:joe

Here, john, mary, and joe are entries in the subdirectory field. Now, when a user refers to john's home directory, the automounter mounts willow:/home/willow. It then places a symbolic link between /tmp_mnt/home/willow/john and /home/john.

If the user then requests access to mary's home directory, the automounter sees that willow:/home/willow is already mounted, so it only returns the link between /tmp_mnt/home/willow/mary and /home/mary. In other words, the automounter now only does:

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

ln -s /tmp_mnt/home/mary /home/mary
```

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

Substitutions

If you have a map with many subdirectories specified, such as:

#key	mount-options	location
john		willow:/home/willow:john
mary		willow:/home/willow:mary
joe		willow:/home/willow:joe
able		pine:/export/home:able
baker		peach:/export/home:baker

consider using string substitutions. You can use the ampersand character (&) to substitute the key wherever it appears. Using the ampersand, the above map now appears as follows:

```
#key            mount-options    location
john                             willow:/home/willow:&
mary                             willow:/home/willow:&
joe                              willow:/home/willow:&
able                             pine:/export/home:&
baker                            peach:/export/home:&
```

If the name of the server is the same as the key itself, for example:

```
#key            mount-options    location
willow                           willow:/home/willow
peach                            peach:/home/peach
pine                             pine:/home/pine
oak                              oak:/home/oak
poplar                           poplar:/home/poplar
```

the use of the ampersand results in:

```
#key            mount-options    location
willow                           &:/home/&
peach                            &:/home/&
pine                             &:/home/&
oak                              &:/home/&
poplar                           &:/home/&
```

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

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

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

```
#key            mount-options    location
oak                              &:/export/&
poplar                           &:/export/&
*                                &:/home/&
```

whereas, in the next map the last two entries are always ignored:

```
#key            mount-options    location
*                                &:/home/&
oak                              &:/export/&
poplar                           &:/export/&
```

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

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

which is a good candidate to be written as:

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

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

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

because the automounter would interpret it as:

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

Environment Variables

You can use the value of an environment variable by prefixing a dollar sign ($) to its name. You can also use braces to delimit the name of the variable from appended letters or digits.

Environmental variables can be inherited from the environment or can be defined explicitly with the -D command-line option. For example, if you want each client to mount client-specific files in the network in a replicated format, create a specific map for each client according to its name. The relevant line for host oak would be:

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

and in willow it would be:

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

This scheme is viable within a small network, but maintaining this kind of host-specific maps across a large network soon becomes unmanageable. The solution for large networks is to invoke the automounter with a command line similar to the following:

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

and have the entry in the direct map read:

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

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

Invoking `automount`

After you complete the maps, make sure that there are no equivalent entries in /etc/fstab and that all entries in the maps refer to NFS shared files.

The syntax to invoke the automounter is:

```
automount [-mnTv] [-Dname=vvalue] [-fmaster-file] \
[-Mmount-directory] [-tsub-options] \
[directory map[-mount-options]] ...
```

The automount(8) man page contains a complete description of all options. The suboptions are the same as those for a standard NFS mount, except for bg (background) and fg (foreground), which do not apply.

Given the following set of three maps:

- auto.master

```
#Mount-point    Map              Mount-options
/net            -hosts
/home           /etc/auto.home   -rw,intr,secure
/-              /etc/auto.direct -ro,intr
```

- auto.home

```
#key            mount-options    location
willow                           willow:/home/willow
cypress                          cypress:/home/cypress
poplar                           poplar:/home/poplar
pine                             pine:/export/pine
apple                            apple:/export/home
ivy                              ivy:/home/ivy
peach           -rw,nosuid       peach:/export/home
```

- auto.direct \

```
/usr/local
          /bin      -ro,soft    ivy:/export/local/sun3 \
          /share    -ro,soft    ivy:/export/local/share \
          /src      -ro,soft    ivy:/export/local/src
/usr/man            -ro,soft    oak:/usr/man \
                                rose:/usr/man \
                                willow:/usr/man
/usr/games          -ro,soft    peach:/usr/games
/usr/spool/news     -ro,soft    pine:/usr/spool/news
/usr/frame          -ro,soft    redwood:/usr/frame1.3 \
                                balsa:/export/frame
```

you can invoke the automounter from the command line or, preferably, from rc.local, in one of the following ways:

- Specify all arguments to the automounter without reference to the master map, as in:
```
automount /net -hosts /home /etc/auto.home -rw,intr,secure \
/- /etc/auto.direct -ro,intr
```

- Specify the same in the *auto.master* file and instruct the automounter to look there for instructions:
```
automount -f /etc/auto.master
```

- Specify mount points and maps in addition to those mentioned in the master map as follows:
```
automount -f /etc/auto.master /src /etc/auto.src -ro,soft
```

- Nullify one of the entries in the master map. (This is particularly useful if you use a map you cannot modify that does not meet the needs of your machine):
```
automount -f /usr/lib/auto.master /home -null
```

- Replace one of the entries with your own:
```
automount -f /usr/lib/auto.master /home /myown/auto.home \
-rw,intr
```

In the example above, the automounter first mounts all items in the map /myown/auto.home under the directory /home. Then, when it consults the master file /usr/lib/auto.master and reaches the line corresponding to /home, it ignores it because it has already mounted on /home.

Given the auto.master file of the previous example, the first two commands are equivalent as long as your network does not have a distributed auto.master file. This file is available only on networks running YP. If your network includes a distributed auto.master file, the second example must be modified in the following way to be equivalent to the first example:

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

The -m option instructs the automounter not to consult the master file distributed by the YP. If you do not run YP, you do not have to specify the -m option. The automounter is completely silent when it does not find a distributed master file.

You can log in as superuser and type any of the above commands at shell level to start the automounter. Ideally, edit rc.local and include your preferred command there.

The Temporary Mount Point

The default name for all mounts is /tmp_mnt. As with other names, this one is arbitrary. It can be changed at invocation by using the -M option. For example:

```
automount -M /auto ...
```

causes all mounts to occur under the directory /auto, which the automounter creates if it does not exist. Do not designate a directory in a read-only file system, as the automounter would then be unable to modify anything.

The Mount Table

Every time the automounter mounts or unmounts a file hierarchy, it modifies /etc/mtab to reflect the current situation. The automounter keeps an image of /etc/mtab in memory and refreshes this image every time it performs a mounting or an automatic unmounting. If you use the umount command to unmount one of the automounted hierarchies (a directory under /tmp_mnt), force the automounter to reread the /etc/mtab file. To do so, enter the command:

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

This gives you the process ID of the automounter. The automounter is designed to reread /etc/mtab when it receives a SIGHUP signal. To send it that signal, enter:

```
% kill -1 PID
```

where PID stands for the process ID you obtained from the previous ps command.

Modifying the Maps

You can modify the automounter maps at any time, but remember that the automounter looks at the master and indirect maps only when it is invoked. If you want a map modification to take effect immediately, you must reboot the machine.

On the other hand, changes to a direct map take effect the next time the automounter has to mount the modified entry. For example, suppose you modify the file /etc/auto.direct so that the directory /usr/src is now mounted from a different server. The new entry takes effect immediately (if /usr/src is not mounted at this time) when you try to access it. If it is mounted now, wait until the auto unmounting takes place, and then access it. If this is not satisfactory, unmount with the umount command, notify `automount` that the mount table has changed (see The Mount Table above), and then access it. Now the mounting is done from the new server.

Error Messages Related to `automount`

The following paragraphs list the error message you are most likely to see if the automounter fails, and explain what each message indicates.

- no mount maps specified

  The automounter was invoked with no maps to serve, and it cannot find the YP auto.master map. This message is produced only when the *-v* option is given. Recheck the command, or restart YP if that was the intention.

- mapname: Not found

  The required map cannot be located. This message is produced only when the *-v* option is given. Check the spelling and pathname of the map name.

- dir mountpoint must start with '/'

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

- mountpoint: Not a directory

  The mount point exists but it is not a directory. Check the spelling and pathname of the mount point.

- hierarchical mountpoint: mountpoint

  The automounter does not allow itself to be mounted within an automounted directory. You will have to use another strategy.

- WARNING: mountpoint not empty!

  The mount point is not an empty directory. This message is produced only when the -v option is given and it is only a warning. It means that the previous contents of the mount point will not be accessible.

- Can't mount mountpoint: reason

  Automounter cannot mount itself at mount point. The reason given with the message should be self-explanatory.

- hostname:file system already mounted on mountpoint

  Automounter has been mounted on an already mounted-on mount point and is attempting to mount the same file system there. This will happen if an entry in /etc/fstab also appears in an automounter map (either by accident or because the output of mount -p was redirected to fstab). Delete one of the redundant entries.

- WARNING: hostname:file system already mounted on mountpoint

  The automounter is mounting itself on top of an existing mount point (warning only).

- couldn't create directory: reason

  Could not create a directory. The reason should be self-explanatory.

- bad entry in map mapname "map entry"

- map mapname, key map key: bad

  The map entry is malformed, and the automounter cannot interpret it. Re-check the entry; perhaps there are characters in it that need escaping.

- mapname: yp_err

  Error looking up an entry in a YP map.

- hostname: exports: rpc_err

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

- host hostname not responding

- hostname:filesystem server not responding

- Mount of hostname:filesystem on mountpoint: reason

  You will see these error messages after the automounter attempted to mount from hostname but either got no response or failed. This may indicate a server or network problem.

- mountpoint - pathname from hostname: absolute symbolic link

  When mounting a hierarchy, the automounter has detected that mountpoint is an absolute symbolic link (begins with "/"). The content of the link is pathname. This may have undesired consequences on the client.

- Cannot create socket for broadcast rpc: rpc_err

- Many_cast select problem: rpc_err

- Cannot send broadcast packet: rpc_err

- Cannot receive reply to many_cast: rpc_err

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

- trymany: servers not responding: reason

  No server in a replicated list is responding. This may indicate a network problem.

- Remount hostname:filesystem on mountpoint: server not responding

  Attempted remount after unmount failed. Indicates a server problem.

- NFS server (pidn@mountpoint) not responding still trying

  An NFS request made to the `automount` daemon with PID n serving mountpoint has timed out. The automounter may be temporarily overloaded or dead. Wait a few minutes; if the condition persists, the easiest solution (for a diskless client) is to reboot the client. If this is not possible, exit all processes that use automounted directories (or, change to a non-automounted directory in the case of a shell), kill the current `automount` process and restart it again from the command line. If this does not work, you must reboot.

The Yellow Pages Service 4

Yellow Pages (YP) is a distributed network lookup service. It maintains a set of files that machines query as they would a database. These files are known as YP maps. All YP maps may be fully replicated on several systems known as YP servers, each of which runs a server process for the maps. It does not matter which server process answers a client request. The response is always the same because the YP map set is identical for each server. This allows you to have multiple servers per network and makes YP service highly reliable and available.

Basic YP Concepts

This section gives an overview of YP, its maps, and its associated commands.

The YP Map

Each YP map contains a set of keys and associated values. For example, in a map called hosts.byname, all the host names within a domain are the keys, and the Internet addresses of these host names are the values. Each YP map has a map name used by programs to access it. Many of the current YP maps are derived from ASCII files traditionally found in /etc such as:

/etc/hosts
/etc/group
/etc/passwd

and a few others. The format of the data within the YP map is identical (in most cases) to the format within the ASCII file. dbm files, which are located in the subdirectories of the directory /usr/etc/yp on YP servers, implement the YP maps.

A YP domain is a named set of YP maps, located in a subdirectory of /usr/etc/yp, the directory where a YP server holds all its maps. The name of this subdirectory is the name of the YP domain. For example, maps for the literature domain are located in /usr/etc/yp/literature.

Each machine belongs to a default domain that is set at boot time by the /etc/rc.local file with the `domainname` command. You must set the domain name on all machines (both servers and clients) that will access the maps of a particular domain. To display or change your YP domain name, use the `domainname` command.

In the YP environment, only YP servers have a set of YP maps, which they make available to clients over the network. There are two kinds of YP servers: YP slave servers and the YP master server. The master server updates the maps of the slave servers. You should always modify maps on the YP master server. The changes propagate from the master server to the YP slave servers. If you create or change YP maps on a slave server instead of a master server, this potentially creates two different versions of the YP map. Furthermore, the YP master server is the only machine with all the necessary ASCII files.

The YP clients run processes that request data from maps on server machines. YP clients do not care which server is the master, since all YP servers should have the same information. The distinction between master and slave server is only important when you make updates.

A server may be a master with regard to one map, and a slave with regard to another. Randomly assigning maps to YP servers can cause a great deal of confusion. You are strongly urged to make a single server the master for all the maps you create by the `ypinit` command within a single domain. The examples in this chapter assume that one server is the master for all maps in the domain.

Commands Used for Maintaining YP

YP offers a series of commands that you can use for setting up and editing maps and performing other YP-related functions. This subsection briefly describes them. These commands are described in detail later in this section and in the man pages.

ypfiles and ypmake are man pages only. They appear here to provide you with a pointer to the useful information they contain.

- `ypserv`
  The YP map server daemon. The ypserv man pages describes `ypserv` and `ypbind`, the YP binder daemon. `ypserv` must run on each YP server. `ypbind` must run on all clients.
- ypfiles(5)
  Describes the file structure of the YP service.
- `ypinit`
  Automatically constructs maps from files located in /etc such as /etc/hosts, /etc/passwd, and others. `ypinit` also constructs initial versions of required maps that are not built from files in /etc for example, ypservers. Use `ypinit` to set up the master YP server for the first time and propagate the setup to the slave YP servers. You typically do not use it as an administrative tool for running systems.
- ypmake(8)
  Describes use of /usr/etc/yp/Makefile, which builds several commonly-changed components of YP maps. These are the maps built from the files in /etc on the master YP server: passwd, hosts, group, netgroup, networks, protocols, and services.
- `makedbm`
  Takes an input file and converts it into a pair of dbm files, which then become valid YP maps. For example, ypmaps.dir and ypmaps.pag are both dbm files. Use `makedbm` to build or rebuild maps not built from /usr/etc/yp/Makefile. You can also use `makedbm` to disassemble a map, so that you can see the key-value pairs that comprise it. You may edit the disassembled form. The disassembled form is in the format required for input back into `makedbm`.

- `ypxfr`
  Moves a YP map from one YP server to another, using YP itself as the transport medium. Run `ypxfr` interactively, or periodically from a crontab file.
- `yppush`
  Requests each of the `ypserv` processes within a domain to transfer a particular map, waits for a summary response from the transfer agent, and prints out the results for each server. Run `yppush` on the master YP server.
- `ypset`
  Tells a `ypbind` process (the local one, by default) to get YP services for a domain from a named YP server. This is not for casual use; only use this command if you are an experienced system manager working on a sizeable problem.
- `yppoll`
  Asks any `ypserv` for the information it holds internally about a single map.
- `ypcat`
  Displays the contents of a YP map. Use it when you do not care which server's version you view. If you need to see a particular server's map, rlogin to that server (or use `rsh` and use `makedbm`).
- `ypmatch`
  Prints the value for one or more specified keys in a YP map. Again, you have no control over which YP server's version of the map you view.
- `ypwhich`
  Use this command to see which YP server a host is using at the moment for YP services, or which YP server is master of a particular map.
- `ypupdated`
  Daemon used for changing YP information. This daemon is normally started up by `inetd`. `ypupdated` consults the file updaters in the /usr/etc/yp directory to determine which maps should be updated and how to change them. `ypupdated` only works if the network is running secure RPC.

How Administrative Files Are Consulted on a YP Network

YP can serve any number of maps. Typically these include some files in /etc. YP services make updating these files much simpler, since you do not have to make the same change to every machine on the network. For example, on networks that do not run YP, programs read the /etc/hosts file to find an Internet address. When you add a new machine, you have to add an entry for this machine to the /etc/hosts files on every machine on the network. On networks running YP, programs that need to consult /etc/hosts do a remote procedure call to the YP servers for the same information.

Programs do not consult the same system administrative files on a network with YP that they would on a network without YP; they consult YP maps instead. The following lists describe how programs on a network running YP consult the administrative files.

Files always consulted

- /etc/passwd
  Always consulted. If there are + or - entries, the YP password map is consulted, otherwise YP is not used. Refer to the passwd(1) man page.
- /etc/group
  Always consulted. If there are + or - entries, the YP group map is consulted, otherwise YP is not used. Refer to the group(5) man page.
- /etc/hosts.equiv and .rhosts
  Always consulted, though neither of these files is in the YP domain. (Refer to the section, "How Security Is Changed with YP", for a fuller explanation of these two files.) If there are + or - entries whose arguments are netgroups, the YP netgroup

map is consulted, otherwise YP is not used. Refer to the hosts.equiv(5) man page.

- /usr/lib/aliases
  Always consulted. Local aliases take precedence over those in the YP database. Refer to aliases(5).

Files never consulted

- /etc/hosts
  Never consulted by disked systems. Consulted by diskless clients only when booting (by the `ifconfig` command in the /etc/rc.local file). After booting the YP map is used instead.
- /etc/services
  Never consulted. The data that was formerly read from this file now comes from the YP services map.
- /etc/protocols
  Never consulted. The data that was formerly read from this file now comes from the YP protocols map.
- /etc/networks
  Never consulted. Data is taken from this file to create the YP networks map.
- /etc/netgroup
  Never consulted. The data that was formerly read from this file now comes from the YP netgroup map.
- /etc/bootparams
  Never consulted. The data that was formerly read from this file now comes from the YP bootparams map.
- /etc/ethers
  Never consulted. The data read from this file comes from the YP ethers map.

YP Installation and Administration

This section covers nine installation and administration topics:

- Setting up a master YP server
- Altering a YP client's database to use YP services
- Setting up a slave YP server
- Setting up a YP client
- Modifying individual YP maps after installing YP
- Propagating a YP map
- Making new YP maps after installing YP
- Adding an additional YP server
- Changing the master server to a different machine

How to Set up a Master YP Server

Before setting up the master YP server, you must take the following steps.

1. Become root.
2. Set up the YP domain name, if it is to differ from the name selected for your network domain during installation.
3. Set up the hostname. If domainname and hostname are set up from /etc/rc.local (this is the default), edit rc.local to reflect the proper hostname.

4. Check that the following files in /etc are complete and reflect an up-to-date picture of your system:
   — passwd
   — hosts
   — ethers
   — group
   — networks
   — protocols
   — services

5. If you know how /etc/netgroup is going to be set up, do that.
   If you do not set up /etc/netgroup, `ypinit` makes an empty netgroup map.

6. Edit /usr/lib/aliases if necessary.

7. For security reasons, you may restrict access to the master YP machine to a smaller set of users than that defined by the complete /etc/passwd file. To do this, copy the complete file to some place other than /etc/passwd. Edit out undesired users from the remaining /etc/passwd using `vipw`. For a security-conscious site, this smaller file should not include the YP escape entry discussed in the next section.

After performing these steps, you are ready to create a new master server.

1. Login as superuser and change directory to /usr/etc/yp.
2. Run `/usr/etc/yp/ypinit`. (An example run of `ypinit` on a master server follows this procedure. Refer to Figure 4-1.)
   You are asked whether you want the procedure to exit at the first non-fatal error. (If `ypinit` exits, you can fix the problem and restart `ypinit`. This option is recommended if you haven't done the procedure before.) You may elect to continue despite non-fatal errors. In this second case you can try to fix all the problems by hand, or fix some, then restart `ypinit`. `ypinit` prompts you for a list of other hosts that will also be YP servers. (Initially, this is the set of YP slave servers, but in the future any of them might become the YP master server.) You need not add any other hosts at this time, but if you know that you will be setting up more YP servers, add them now. You will save yourself some work later, and there is little runtime penalty for doing it. (However, do not name every host in the network.)

3. Invoke `/usr/etc/ypserv` and `/usr/etc/ypbind`.
4. Check /etc/rc.local for the following lines:

```
if [ -f /bin/domainname ] ; then
  if [ "`/bin/domainname`" != "" ]; then
    if [ -f /usr/etc/ypserv -a -d /etc/yp/`/bin/domainname` ]; then
        /usr/etc/ypserv & echo -n ` ypserv'
    fi
    if [ -f /usr/etc/ypbind ]; then
        /usr/etc/ypbind & echo -n ` ypbind'
    fi
    if [ -f /usr/etc/rpc.yppasswdd ]; then
        /usr/etc/rpc.yppasswdd /etc/passwd /etc/pwrestrict \
        -m passwd pwrestrict DIR=/etc & echo -n ` yppasswdd'
    fi
  fi
fi
```

   Add these lines to /etc/rc.local if they do not already exist. `ypserv` then starts up automatically from /etc/rc.local every time the server boots.

Figure 4-1 - ypinit on master:

```
host1# cd /etc/yp
host1# ./ypinit
usage:
          ypinit -m
          ypinit -s master_server
where -m is used to build a master yp server data base, and -s is
used for a slave data base.  master_server must be an existing reach-
able yp server.
host1# ./ypinit -m
The local host's domain name hasn't been set.  Please set it.
host1# domainname host1
host1# ./ypinit -m
Installing the yp data base will require that you answer a few ques-
tions. Questions will all be asked at the beginning of the procedure.
Do you want this procedure to quit on non-fatal errors? [y/n: n]  y
Can we destroy the existing /usr/etc/yp/host1 and its contents? [y/
n: n]  y
At this point, we have to construct a list of the hosts which will
run yp servers.  host1 is in the list of yp server hosts.  Please
continue to add the names for the other hosts, one per line.  When
you are done with the list, type a <ctl D>.
          next host to add:  host2
          next host to add:  ^D
The current list of yp servers looks like this:
host1
host2
Is this correct?  [y/n: y]  y
There will be no further questions. The remainder of the procedure
should take 5 to 10 minutes.
Building /etc/yp/host1/ypservers...
Running /etc/yp/Makefile...
updated ypservers
updated passwd
updated group
updated hosts
updated ethers
updated networks
updated rpc
updated services
updated protocols
updated pwrestrict
updated netgroup
updated bootparams
10 aliases, longest 51 bytes, 310 bytes total
updated aliases
updated publickey
updated netid
host1 has been set up as a yp master server without any errors.
If there are running slave yp servers, run yppush now for any data
bases which have been changed.  If there are no running slaves, run
ypinit on those hosts which are to be slave servers.
```

Using a Non-CONVEX Master Server

If you use a non-CONVEX machine as the master Yellow Pages server, you must make a change to the YP configuration on that machine. CONVEX YP expects the YP services map and RPC map to be built so that they may be searched quickly with a single `yp_match` access. Sun Microsystems and some other vendors do not build these maps, and a change to the master YP Makefile must be made to build the maps so as to allow the CONVEX system to find the information it needs. This change does not interfere with YP accesses by machines from other vendors, as it only supplies additional information for the CONVEX system.

To make this change, copy lines from the /usr/etc/yp/Makefile file on the CONVEX system. Find the string "services.time", copy all lines down to the string "protocols.time" to a temporary file and move the temporary file onto the non-CONVEX machine. On a Sun running SunOS 4.0 or greater, look for the Makefile in /var/yp; on older Suns look for it in /usr/etc/yp. Go to that directory and replace the appropriate lines in the Makefile with those from the temporary file.

The lines copied from the CONVEX Makefile contain several occurrences of `DOMDIR`. Edit these lines, replacing `DOMDIR` with the macro name used by the non-CONVEX vendor. On most Suns, this macro is `$(YPDBDIR/$(DOM)`.

Once the Makefile is ready, remove the files services.time and rpc.time, and build the new maps with `make`. The CONVEX system should now be able to access the services and RPC maps correctly.

Altering a YP Client's Files To Use YP Services

Once you decide to run YP at your site, you should have all hosts on the network access the YP maps, rather than potentially out-of-date information in their local administrative files. To enforce that policy, run a `ypbind` process on the client machine (including machines that may be running YP servers), and abbreviate or eliminate the files that traditionally implemented the YP maps. The files in question are:

/etc/passwd
/etc/hosts
/etc/ethers
/etc/group
/etc/networks
/etc/protocols
/etc/services
/etc/netgroup
/etc/aliases
/etc/netmasks
/.rhosts

The treatment of each file is discussed in this section.

- /etc/networks, /etc/protocols, /etc/ethers, /etc/services, and /etc/netgroup need not exist on any YP clients. These files are never consulted.

- /etc/hosts.equiv is never served by YP. However, you can add escape sequences to reference YP. This reduces problems with `rlogin` that are sometimes caused by different /etc/hosts.equiv files on the two machines.

  To let anyone log in to a machine, edit /etc/hosts.equiv to contain a single line, with only the character, + (plus) on it. A line containing only a + means that all further entries are retrieved from YP rather than the local file.

Alternatively, you can exercise more control over logins by using lines of the form:

```
+@trusted_group1
+@trusted_group2
-@distrusted_group
```

Each of the names to the right of the at sign @ is assumed to be a netgroup name, defined in the global netgroup database. The netgroup database is served by YP.

If neither the + or - is used, only the entries in /etc/hosts.equiv are used; YP is not used.

- /.rhosts also is never served by YP. As shown in Chapter 1, its format is identical to that of /etc/hosts.equiv. However, because this file controls remote root access to the local machine, allowing unrestricted access to it is not recommended. Make the list of trusted hosts explicit, or use netgroup names for the same purpose. You can not use secondary hostnames in your .rhosts hosts.equiv or netgroup files. You can, however, use secondary hostnames in /etc/hosts. All of the above files are related in that they enable local machines to access remote machines in some fashion.

- /etc/hosts must contain entries for the local host's name, and the local loopback name. These are accessed at boot time when the YP service is not yet available. After the system is running, and after the ypbind process is up, the /etc/hosts file is not accessed at all. An example of the hosts file for YP client raks' is:

```
127.0.0.1               localhost
192.9.1.87      raks    # Stefania
```

- /etc/passwd should contain entries for the root user name and the primary users of the machine, and the + escape entry to force the use of the YP service. A few additional entries are recommended: daemon, to allow file-transfer utilities to work; and operator, to let a dump operator login. A sample YP client's /etc/passwd file looks like:

```
root:9wxntql2tHT.k:0:1:Operator:/:/bin/csh
nobody:*:-2:-2::/:
daemon:*:1:1::/:
sys:*:2:2::/:/bin/csh
bin:*:3:3::/bin:
uucp:*:4:4::/var/spool/uucppublic:
news:*:6:6::/var/spool/news:/bin/csh
sync::1:1::/:/bin/sync
raks:7kjDXZD/Hug2s:624:20:Stefania:/home/dancer/raks:/bin/
csh
+::0:0:::
```

The last line informs the library routines to use the YP service. If you remove the last line in the passwd file, you will disable YP password access.

A program that uses getpwent to access /etc/passwd first looks in the password file on your machine; it then looks in the YP password file (only if your machine's password file contains + (plus sign) entries), as shown in the above example. Earlier entries in the file take precedence over, or mask, later ones with the same user name, or the same user ID. Therefore, please note the order of the entries for daemon and for sync (which have the same user ID) and duplicate this order in your own file.

- /etc/group may be reduced to a single line:
  +:

  which forces all translation of group names and group IDs to be made via the YP service. This is the recommended procedure.

How To Set Up a Slave YP Server

The network must be operational before you can set up a slave YP server — in particular, you must be able to copy files from the master YP server to YP slaves. To create a new slave server,

1. Change directory to /usr/etc/yp.
2. Become superuser.
3. Select a host already set up as a YP server as the master.
   Ideally, the named host really is the master server, but it can be any host that has its YP database set up. The host must be reachable.
4. Enter the following to determine whether the host is reachable.
   `rpcinfo -b ypserv 2 | grep` ***hostname***
   The server in question is reachable if it appears in the resulting list.
5. Check that an entry for daemon exists in the /etc/passwd files of both slave and master, and precedes any other entries which have the same user ID.
6. Set up the default domain name on the machine intended to be the YP slave server to be the same domain name as the default domain name on the machine named as the master.
7. Run `ypinit` with the **-s** option. (An example run of `ypinit` on a slave server follows this procedure.)

Figure 4-2 - ypinit for slave:

```
goofy# cd /etc/yp
goofy# ypinit -s mickey
Installing the yp data base will require that you answer a few
questions. Questions will all be asked at the beginning of the
procedure.

Do you want this procedure to quit on non-fatal errors? [y/n:
n]  y

There will be no further questions. The remainder of the pro-
cedure should take a few minutes, to copy the data bases from
mickey.
Transferring ethers.byaddr...
Transferring ethers.byname...
Transferring group.bygid...
Transferring group.byname...
Transferring hosts.byaddr...
Transferring hosts.byname...
Transferring mail.aliases...
Transferring netgroup...
Transferring netgroup.byuser...
Transferring netgroup.byhost...
Transferring networks.byaddr...
Transferring networks.byname...
Transferring passwd.byname...
Transferring passwd.byuid...
Transferring protocols.byname...
Transferring protocols.bynumber...
Transferring services.byname...
Transferring ypservers...
Transferring services...
Transferring rpc.byname...
Transferring rpc.bynumber...
Transferring pwrestrict.byname...
Transferring pwrestrict.byuid...
Transferring bootparams...
Transferring publickey.byname...
Transferring netid.byname

goofy's yellowpages data base has been set up without any er-
rors.

At this point, make sure that /etc/passwd, /etc/hosts, /etc/
networks, /etc/group, /etc/protocols, /etc/services, /etc/
rpc, /etc/pwrestrict, and /etc/netgroup have been edited so
that when the yellow pages is activated, the data bases you
have just created will be used, instead of the /etc ASCII
files.  See the "NFS System Manager's Guide" for more infor-
mation.
```

As mentioned in "How To Set Up a Master YP Server", you are not prompted for a list of other servers, but you do have the opportunity to choose whether or not the procedure gives up at the first non-fatal error.

After running `ypinit` make copies of /etc/passwd, /etc/hosts, /etc/group, /etc/networks, /etc/protocols, /etc/netgroup, and /etc/services. For instance on a machine named ypslave, enter:

```
ypslave# cp /etc/passwd /etc/passwd-
```

Edit the original files in accordance with the preceding section, "Altering a YP Client's Files To Use YP Services", to ensure that processes on the slave YP server actually use the YP services, rather than the local ASCII files. (That is, make sure the YP slave server is also a YP client.) Make backup copies of the edited files, as well. For instance:

```
ypslave# cp /etc/passwd /etc/passwd+
```

Run ypbind

```
ypslave# /usr/etc/ypbind
```

from the slave so that the slave can access domain maps.

After the YP database is set up by `ypinit`, enter **`/usr/etc/ypserv`** to begin supplying YP services. On subsequent reboots, `ypserv` starts automatically from /etc/rc.local.

How To Set Up a YP Client

To set up a YP client, edit the local files as described above in Altering a YP Client's Files to Use YP Services. If `/usr/etc/ypbind` is not running already, start it. Make sure that /etc/rc.local contains the proper line to set the domainname. With the ASCII databases of /etc abbreviated and `/usr/etc/ypbind` running, the processes on the machine will be clients of the YP services.

At this point, there must be a YP server available; processes will hang if no YP server is available while `ypbind` is running. Note the possible alterations to the client's /etc database as discussed above in the section on altering the client. Because some files may not be there, or some may be specially altered, it is not always obvious how the ASCII databases are being used. The escape conventions used within those files to force data to be included or excluded from the YP databases are found in the following man pages: passwd(1), hosts(5), netgroup(5), hosts.equiv(5), and group(5). In particular, notice that changing passwords in /etc/passwd (by editing the file, or by running passwd) only affects the local client's environment. Change the YP password database by running `yppasswd`.

To add a new client diskless workstation to a network already running YP, use the `setup_client` program described in Chapter 2. Use the same syntax described in this chapter. Be sure to specify client for yp_type.

How To Modify Existing YP Maps After YP Installation

Always change databases served by YP on the master server. To change the databases expected to change most frequently, like /etc/passwd, do the following as superuser.

1. Edit the ASCII file.
2. Change directory to /usr/etc/yp.
3. Run make.

Non-standard maps (that is, maps that are specific to the applications of a particular vendor or site), or maps that are expected to change rarely, or maps for which no ASCII form exists (for example, maps created when you installed YP), may be modified manually. Use `makedbm` with the -u option to disassemble them into a form which can be modified using standard tools (such as `awk`, `sed`, or `vi`). Then build a new version using `makedbm`. This may be done by hand in two ways:

- Redirect the output of `makedbm` to a temporary file that can be modified, then fed back into `makedbm` or

- Operate on the output of `makedbm` within a pipeline that feeds into `makedbm` again directly. This is appropriate if the disassembled map can be updated by modifying it with `awk`, `sed`, or a `cat` append, for instance.

Suppose you want to create a non-standard YP map, called mymap. You want it to consist of key-value pairs in which the keys are strings like al, bl, cl, etc. (the l is for left), and the values are ar, br, cr (the "r" is for right). There are two possible procedures to follow when creating new maps. One uses an existing ASCII file as input; the other uses standard input.

For example, suppose there is an existing ASCII file named /usr/etc/yp/mymap.asc created with an editor or a shell script on the machine ypmaster. The map is located in the subdirectory home_domain. Create the YP map for this file by entering:

```
ypmaster#  cd /usr/etc/yp
ypmaster#  makedbm mymap.asc home_domain/mymap
```

At this point you notice the map really should have included another 2-tuple: (dl, dr). You can make the modification simply. In all situations like this, remember to modify the map by first modifying the ASCII file. Modifications made only to the dbm files, and not also in the ASCII file, will be lost. Make the modification like this:

```
ypmaster#  cd /usr/etc/yp
ypmaster#  <make editorial change to mymap.asc>
ypmaster#  makedbm mymap.asc home_domain/mymap
```

When there is no original ASCII file, create the YP map from the keyboard by entering input like this (assume the machine name is ypmaster and the default domain is home_domain):

```
ypmaster# cd /usr/etc/yp
ypmaster# makedbm - home_domain/mymap
al ar
bl br
cl cr
<CTRL-D>
```

When you need to modify that map, use `makedbm` to create a temporary ASCII intermediate file which you can be edit using standard tools. For instance:

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

At this point you can edit mymap.temp to contain the correct information. Create a new version of the database using the commands

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

The preceding paragraphs explained how to use some tools. In reality, almost everything you must do can be done by `ypinit` and /usr/etc/yp/Makefile. If you add non-standard maps to the database, or change the set of YP servers after the system is already up and running, other procedures such as those explained above are required.

Whether you use the Makefile in /usr/etc/yp or some other procedure, the goal is the same: a new pair of well-formed dbm files must end up in the domain directory on the master YP server.

Propagation of a YP Map

To propagate a map means to move it from place to place — in general, to move it from the master YP server to all slave YP servers. Initially, `ypinit` moves it, as described above in "How To Set Up A Slave YP Server". After a slave YP server has been initialized, updated maps are transferred from the master server by `ypxfr`. You can run `ypxfr` in three different ways: via `cron`, via `ypserv`, or interactively. Following are examples of each.

Maps have differing rates of change; for instance protocols.byname may not change for months at a time, but passwd.byname may change several times a day in a large organization. You can set up entries in a crontab file to periodically run `ypxfr` at a rate appropriate for any map in your YP database. `ypxfr` contacts the master server and transfers the map only if the master's copy is more recent than the local copy.

To avoid having to have a crontab entry for each map, several maps with approximately the same change characteristics can be grouped in a shell script, and the shell script can be run from a crontab file.

Suggested groupings, mnemonically named, can be found in /usr/etc/yp, ypxfr_1perhour, ypxfr_1perday, and ypxfr_2perday. If the rates of change are inappropriate for your environment, you can easily modify or replace these shell scripts.

These same shell scripts should be run at each YP slave server in the domain. Alter the exact time of execution from one server to another, so as to prevent the checking from bogging down the master. If you want the map transferred from some particular server, not the master, you can specify that (using `ypxfr -h` option) within the shell script. You can use the -s option to transfer maps from another domain. Finally, maps having unique change characteristics can be checked and transferred by explicit invocations of `ypxfr` within the system crontab file, /.crontab.

You may also send files out from the master server, rather than asking for them on the slave server. `yppush` sends a transfer Map RPC request to `ypserv`, which invokes `ypxfr` to send the map. `yppush` enumerates the YP map ypservers to generate a list of YP servers in your domain, and sends a Transfer Map request to each of the named YP servers. `ypserv` forks off a copy of `ypxfr`, invokes it with the -C flag, and passes it the information it needs to identify the map and call back the initiating `yppush` process with a summary status.

In the cases mentioned above, `ypxfr` transfer attempts and results can be captured in a log file. If /usr/etc/yp/ypxfr.log exists, results are appended to it. No attempt to limit the log file is made. To turn off logging, remove the log file.

Finally, you can run `ypxfr` as a command. Typically, you do this only in exceptional situations — for example when setting up a temporary YP server to create a test environment, or when trying to quickly get a YP server that has been out of service consistent with the other servers.

How to Make New YP Maps After YP Installation

Adding a new YP map entails getting copies of the map's dbm files into the domain directory on each of the YP servers in the domain. The actual mechanism is described above in the section "Propagation of a YP Map". This section describes how to get the proper files in place so the propagation works correctly. Both master and slaves must be set up correctly.

After deciding which YP server is the master of the map, modify /usr/etc/yp/Makefile on the master server so that the map can be conveniently rebuilt. Filter an ASCII file through `awk`, `sed`, and/or `grep` to make it suitable for input to makedbm.

Consult the existing Makefile as a source for programming examples. Use the mechanisms already in place in /usr/etc/yp/Makefile when deciding how to create dependencies that `make` will recognize; specifically, the use of .time files allows you to see when the Makefile was last run for the map.

To get an initial copy of the map, run `yppush` on the YP master server. The map must be globally available before clients begin to access it. If the map is available from some YP servers, but not all, you will see unpredictable behavior from client programs.

How To Add a New YP Server Not in the Original Set

To add a new YP slave server, start by modifying some maps on the master YP server. If the new server is a host that has not been a YP server before, add the host's name to the map ypservers in the default domain. The sequence for adding a server named ypslave to domain_name is:

```
ypmaster# vi /etc/ypservers
ypmaster# cd /usr/etc/yp
ypmaster# make
```

Next, set up the new slave YP server's databases by copying the databases from YP master server ypmaster. To do this, remote log in to the new YP slave, and run the `ypinit` command:

```
ypslave# cd /usr/etc/yp
ypslave# ypinit -s ypmaster
```

→ Note

If a host name is not in ypservers it will not be warned of updates to the YP map files.

Then complete the steps described above in the section, How To Set Up A Slave YP Server.

How To Change to a New Master Server

1. Build the map at the new master. Because the old YP master's name occurs as a key-value pair in the existing map, it is not sufficient to use an existing copy at the new master, or to send a copy there with `ypxfr`. The key must be re-associated with the new master's name. If the map has an ASCII source file, it should be present in its current version at the new master.
   /etc/yp/Makefile must be set up correctly for this to work. If it isn't, do it now. Remake the YP map (called jokes.bypunchline below) locally with the sequence:

   ```
   newmaster# cd /usr/etc/yp
   newmaster# make jokes.bypunchline
   ```

2. Go back to the old master (if it will remain a YP server) and edit /usr/etc/yp/Makefile so that jokes.bypunchline is no longer made there — that is, comment out the section of oldmaster:/etc/yp/Makefile that made jokes.bypunchline.
3. If the map only exists as a dbm database, you can remake it on the new master by disassembling an existing copy (one from any YP server will do) and running the disassembled version back through makedbm. For example:

   ```
   newmaster# cd /usr/etc/yp
   newmaster# ypcat -k jokes.bypunchline |\
   makedbm - mydomain/jokes.bypunchline
   ```

4. After making the map on the new master, send a new copy of the map to the other (slave) YP servers. However, do not use `yppush`; the other slaves will try to get new copies from the old master, rather than the new one. Rather, become superuser on the old master server and type:

   ```
   oldmaster# /usr/etc/yp/ypxfr -h newmaster jokes.bypunchline
   ```

5. Now you have a new copy on the old master, run `yppush`. The remaining slave servers still believe that the old master is the current master, and will attempt to get the current version of the map from the old master. When they do so, they will get the new map, which names the new master as the current master.

If the method above fails, there is a cumbersome but always effective option. On each YP server machine, while superuser, execute the command shown just above. This will certainly work, but should be considered the worst case solution.

Debugging a YP Client

This debugging section covers problems seen on a YP client; the following section covers problems seen on a YP server. Before trying to debug a YP client, read the earlier section in this chapter on how the YP works.

On Client: Commands Hang

The most common problem at a YP client node is for a command to hang and generate console messages that say:

```
yp: server not responding for domain <wigwam>.  Still trying
```

Sometimes many commands begin to hang, even though the system as a whole seems okay and you can run new commands.

The message above indicates that `ypbind` on the local machine is unable to communicate with `ypserv` in the domain wigwam. This often happens when machines that run `ypserv` have crashed. It may also occur if the network or the YP server machine is so overloaded that `ypserv` cannot get a response back to your `ypbind` within the time-out period. Under these circumstances, all the other YP client nodes on your net show the same or similar problems. The condition is temporary in most cases, and the messages usually go away when the YP server machine reboots and `ypserv` gets back in business, or when the load on the YP server nodes or the Ethernet decreases.

However, in the circumstances described below, the situation will never get better.

- The YP client has not set, or has incorrectly set, domainname on the machine. Clients must use a domain name that the YP servers know. Use `domainname` to see the client domain name. Compare that with the domain name set on the YP servers. The domain name should be set in /etc/rc.local. When /etc/rc.local fails to set or incorrectly sets domainname do the following: become superuser on the machine in question, edit /etc/rc.local to fix the domainname line with a proper domain name (this assures domain name will be correct every time the machine boots), and set domainname manually so it is fixed immediately. To do this, enter:

  ```
  # domainname good_domain_name
  ```

- If your domain name is correct, make sure your local network has at least one YP server machine. You can automatically bind only to a `ypserv` process on your local network, not on another accessible network. There must be at least one YP server for your machine's domain running on your local network. Two or more YP servers will improve availability and response characteristics for YP services.

- If your local network has a YP server, make sure it is up and running. Check other machines on your local net. If several client machines have problems simultaneously, suspect a server problem. Find a client machine behaving normally, and try the `ypwhich` command. If `ypwhich` never returns an answer, kill it and go to a terminal on the YP server machine. Enter:

  ```
  # ps ax | grep yp
  ```

  and look for `ypserv` and `ypbind` processes. If the server's `ypbind` daemon is not running, start it up by entering:

  ```
  # /usr/etc/ypbind
  ```

  If there is a `ypserv` process running, do a `ypwhich` on the YP server machine. If `ypwhich` returns no answer, `ypserv` has probably hung and should be restarted.

Kill the existing ypserv process (you must be logged on as root), and start /usr/etc/ypserv:

```
# kill -9 [some pid # from ps]
# /usr/etc/ypserv
```

If ps shows no ypserv process running, start one.

On Client: YP Service Unavailable

When other machines on the network appear to be okay, but YP service becomes unavailable on your machine, many different symptoms may show up. Among them: some commands appear to operate correctly while others terminate printing an error message about the unavailability of YP; some commands limp along in a backup-strategy mode particular to the program involved; and some commands or daemons crash with obscure messages or no message at all. For example, things like the following may show up:

```
my_machine% ypcat myfile
ypcat: can't bind to YP server for domain <wigwam>.
          Reason: can't communicate with ypbind.

my_machine% /usr/etc/yp/yppoll myfile
Sorry, I can't make use of the yellow pages.  I give up.
```

When symptoms like those above occur, try

```
my_machine% ls -l
```

on a directory containing files owned by many users, including users not in the local machine's /etc/passwd file, for example /usr. If the ls -l reports file owners not in the local machine's /etc/passwd file as numbers, rather than names, it is one more symptom that YP service is not working.

These symptoms usually indicate that your ypbind process is not running. Run **ps ax** to check for it. If it you do not find it, type:

```
my_machine# /usr/etc/ypbind
```

to start it. YP problems should disappear.

On Client: ypbind Crashes

If ypbind crashes almost immediately each time it is started, you should look for a problem in some other part of the system. Check for the presence of the portmap daemon by typing:

```
my_machine%  ps ax | grep portmap
```

If you do not find it running, take the CONVEX machine down to single-user mode, then return it to multiuser mode. Because so many other daemons must be started after portmap, this is the least disruptive means of correcting the problem.

If portmap itself will not stay up or behaves strangely, look for more fundamental problems. Check the network software.

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

```
rigel% rpcinfo -p client
```

Where *client* is the name of your machine. If your portmap is okay, the output should look like:

```
program   vers  proto port
100007    2     tcp   1024  ypbind
100007    2     udp   1028  ypbind
100007    1     tcp   1024  ypbind
100007    1     udp   1028  ypbind
100021    1     tcp   1026  nlockmgr
100024    1     udp   1052  status
100021    1     udp   1054  nlockmgr
100024    1     tcp   1027  status
100020    1     udp   1058  llockmgr
100020    1     tcp   1028  llockmgr
100021    2     tcp   1029  nlockmgr
100012    1     udp   1083  sprayd
100011    1     udp   1085  rquotad
100005    1     udp   1087  mountd
100008    1     udp   1089  walld
100002    1     udp   1091  rusersd
100002    2     udp   1091  rusersd
100001    1     udp   109   rstatd
100001    2     udp   1094  rstatd
100001    3     udp   1094  rstatd
100015    6     udp   1365  selection_svc
```

On your machine the port numbers will be different. The four entries that represent the ypbind process are:

```
100007    2     tcp   1024  ypbind
100007    2     udp   1028  ypbind
100007    1     tcp   1024  ypbind
100007    1     udp   1028  ypbind
```

If they are not there, `ypbind` has been unable to register its services. Reboot the machine. If they are there and they change each time you try to restart `/usr/etc/ypbind`, reboot the system, even if the portmap is up. If the situation persists after reboot, call the CONVEX Technical Assistance Center.

On Client: `ypwhich` Inconsistent

When you use `ypwhich` several times at the same client node, the answer you get back varies — the YP server changes. This is normal. The binding of YP client to YP server changes over time on a busy net and when the YP servers are busy. Whenever possible, the system stabilizes at a point where all clients get acceptable response time from the YP servers. As long as your client machine gets YP service, it doesn't matter where the service comes from. Often a YP server machine gets its own YP services from another YP server on the net.

Debugging a YP Server

Before trying to debug a YP server, read the earlier section in this chapter on how YP works.

On Server: Different Versions of a YP Map

Because YP works by propagating maps among servers, you will sometimes find different versions of a map at servers on the network. This version skew is normal if transient, and abnormal otherwise.

Normal update is prevented when some YP server or some router between YP servers is down during a map transfer attempt. (Normal update is described in the earlier section, "Propagation of A YP Map"). When all the YP servers, and all the routers between them, are up and running, `ypxfr` should succeed.

If a particular slave server has problems updating, log in to that server and run `ypxfr` interactively. If `ypxfr` fails, it will tell you why it failed, and you can fix the problem. If `ypxfr` succeeds, but you think it has been failing sometimes, create a log file to enable logging of messages. Enter:

```
ypslave# cd /usr/etc/yp
ypslave# touch ypxfr.log
```

This saves all output from `ypxfr`. The output looks much like what `ypxfr` creates when run interactively, but each line in the log file is timestamped. (You may see unusual orderings in the time stamps. That's okay— the time stamp tells you when `ypxfr` began its work. If copies of `ypxfr` ran simultaneously, but their work took differing amounts of time, they may actually write their summary status line to the log files in an order different from the order in which they were invoked.) Any pattern of intermittent failure shows up in the log. When you have fixed the problem, turn off logging by removing the log file. If you forget to remove it, it will grow without limit.

While still logged in to the problem YP slave server, inspect the system crontab file, /usr/spool/cron/crontabs/root and the ypxfr* shell scripts it invokes. Typos in these files cause propagation problems, as do failures to refer to a map within any shell script.

Also, make sure that the YP slave server is in the map ypservers within the domain. If it's not, it will still work as a server, but `yppush` won't tell it when a new copy of a map exists.

If the problem is not obvious, you can work around it while you debug using `rcp`, `ftp`, or `tftp` to copy a recent version from any healthy YP server. You may not be able to do this as root, but you can probably do it as daemon. For instance, to transfer map "busted", do the following:

```
ypslave# chmod go+w /usr/etc/yp/mydomain
ypslave# su daemon
$ rcp ypmaster:/etc/yp/mydomain/busted.\* /usr/etc/yp/mydomain
$ ^D
ypslave# chown root /usr/etc/yp/mydomain/busted.*
ypslave# chmod go-w /usr/etc/yp/mydomain
```

Notice that the * character has been escaped in the command line, so that it will be expanded on ypmaster, instead of locally on ypslave. Also, notice that the map files should be owned by root, so you must change ownership of them after the transfer. Obviously, if you can do the `rcp` as root, it makes the whole thing easier.

On Server: `ypserv` Crashes

When the `ypserv` process crashes almost immediately, and won't stay up even with repeated activations, the debug process is virtually identical to that described above in the section "On Client: `ypbind` Crashes." Check for the portmap daemon:

```
ypserver# ps ax | grep portmap
```

Reboot the server if you do not find the daemon. If it is there, enter:

```
rigel% rpcinfo -p speed
```

and look for output to the screen like:

```
program   vers  proto port
100004    2     udp   1027  ypserv
100004    2     tcp   1024  ypserv
100004    1     udp   1027  ypserv
100004    1     tcp   1024  ypserv
100007    2     tcp   1025  ypbind
100007    2     udp   1035  ypbind
100007    1     tcp   1025  ypbind
100007    1     udp   1035  ypbind
100009    1     udp   1023  yppasswdd
100003    2     udp   2049  nfs
10002     1     udp   1074  status
100024    1     tcp   1031  status
100021    1     tcp   1032  nlockmgr
100021    1     udp   1079  nlockmgr
100020    1     udp   1082  llockmgr
100020    1     tcp   1033  llockmgr
100021    2     tcp   1034  nlockmgr
100012    1     udp   1104  sprayd
100011    1     udp   1106  rquotad
100005    1     udp   1108  mountd
100008    1     udp   1110  walld
100002    1     udp   1112  rusersd
100002    2     udp   1112  rusersd
100001    1     udp   1115  rstatd
100001    2     udp   1115  rstatd
100001    3     udp   1115  rstatd
```

On your machine, the port numbers will be different. The four entries that represent the `ypserv` process are:

```
100004    2     udp   1027  ypserv
100004    2     tcp   1024  ypserv
100004    1     udp   1027  ypserv
100004    1     tcp   1024  ypserv
```

If they are not there, ypserv has been unable to register its services. Reboot the machine. If they are there, and they change each time you try to restart ypserv reboot the machine. If the situation persists after reboot, call the CONVEX Technical Assistance Center.

Other Possible YP Errors

The following items show how various YP utilities respond if ypserv is killed or aborted. Network problems, host server network problems, or a missing host server portmap also cause these responses.

- ypserver# ypcat passwd
  ypcat: can't bind to yp server for domain <mydomain>.
  Reason: can't bind to a server which serves domain.
- ypserver# ypmatch <username> passwd
  ypmatch: Can't match key <username> in map passwd.byname.
  Reason: can't bind to a server which serves domain.
- ypserver# ypwhich
  ypwhich: Domain <mydomain> not bound on <mydomain>.
- ypserver# ypwhich
  ypwhich: can't call ypbind on <ypclient>: RPC: Timed out

➔ Note

Two ypbind processes have spawned during the ypwhich command, which causes the time out.

- ypserver# /usr/etc/yp/yppoll passwd
  ** never returns **
- ypserver# /usr/etc/yp/ypset <mydomain>
  ** never returns **

The following items show how various YP utilities respond if ypbind is killed or aborted.

- ypclient# ypcat passwd
  ypcat: can't bind to yp server for domain <mydomain>.
  Reason: can't communicate with ypbind.
- ypclient# ypmatch <username> passwd
  ypmatch: Can't match key username in map passwd.byname.
  Reason: can't communicate with ypbind.
- ypclient# ypwhich
  ypwhich: can't call ypbind on <ypclient>: RPC: Timed out
- ypclient# /usr/etc/yp/yppoll passwd
  RPC: Timed out
- ypclient# /usr/etc/yp/ypset <ypserver>
  Sorry, I can't make use of the yellow pages. I give up.

The following item shows how ypset responds if both ypbind and ypserv are not running.

- ypclient# /usr/etc/yp/ypset -d <mydomain> ypserver
  Sorry, I couldn't send my rpc message to ypbind on host <yp-client>.

Changing Security with YP

Security on a system running YP is dependent on how YP consults the administrative files on which its maps are based. Local files on the host are consulted first, then the system consults maps in the YP domain. For example, the system checks the /etc/aliases file for mail aliases, then checks the mail.aliases YP map.

The remaining files on which YP maps are based are global files: /etc/hosts, /etc/networks, /etc/ethers, /etc/services, /etc/netmasks, /etc/protocols, and /etc/netgroup. The information in these files is network wide data, and is accessed only from YP. However, when booting, each machine needs an entry in /etc/hosts for itself. In summary, if YP is running, global files are only checked in the YP maps; a file on your local machine is not consulted.

Special YP Password Change

When you change your password with the `passwd` command, you change the entry explicitly given in your machine's local /etc/passwd file. If your password is not given explicitly, but rather is pulled in from YP with a + entry, then the `passwd` command will print the error message

```
Not in passwd file.
```

To change your password in the YP password file, you must use the `yppasswd` command. To enable this service, you must start up the daemon `yppasswd` server on the machine serving as the master for the YP password file.

/etc/publickey

To enable users to use the secure option to mount a directory, the YP database publickey must exist. publickey is the public key database used for secure networking. Each entry in the database consists of a network user name (which may either refer to a user or a hostname), followed by the user's public key (in hexadecimal notation), a colon, and then the user's secret key encrypted with its login password (also in hexadecimal notation).

This file is altered either by the user through the `chkey` command or by using the `newkey` command. The file /etc/publickey should only contain data on the YP master machine, where it is converted into the YP database publickey.byname. Also refer to the ypupdated(8c), newkey(8), chkey(1), and publickey(3r) man pages.

There are two YP databases that enhance system security called publickey and netid. You use the `newkey` program to put new entries into this map. Give the following command:

```
# cd /usr/etc/yp
# make publickey
```

to create the YP database from the /etc/publickey file. Do not use a text editor to alter the /etc/publickey file because the file contains encryption keys. To alter the /etc/publickey file, you need to use the `newkey` command. There are two options to this command:

```
# newkey -u username
```

for a regular user on a host machine, and

```
# newkey -h hostname
```

for a root user on a host machine. The publickey database contains every user that has a public key. They are identified by a long string.

You should also be aware that there is a YP database called netid; however, you do not need to administer it. The netid database is created from the passwd, host, and group files.

Netgroups: Network Wide Groups of Machines and Users

YP uses the /etc/netgroup file on the master YP server for permission checking during remote mount, login, remote login, and remote shell. It uses /etc/netgroup to generate three YP maps in the /usr/etc/yp/domainname directory: netgroup, netgroup.byuser and netgroup.byhost. The YP map netgroup contains the basic information in /etc/netgroup. The two other YP maps contain a more specific form of the information to speed up the lookup of netgroups given the host or user.

The programs that consult these YP maps are `login`, `mountd`, `rlogin`, and `rsh`. `login` consults them for user classifications if it encounters netgroup names in /etc/passwd. `mountd` consults them for machine classifications if it encounters netgroup names in /etc/exports. `rlogin` and `rsh` consult the netgroup map for both machine and user classifications if they encounter netgroup names in the /etc/hosts.equiv or /.rhosts file.

If You Do Not Use YP

If `ypserv` on the master is disabled, you can no longer update any of the YP maps.

If you choose not to use YP, you can disable it by simply renaming the /usr/etc/ypbind to /usr/etc/ypbind.orig. (This is what the install script does automatically if you tell it you do not want to run YP.)

```
ypclient# ps ax | grep yp
ypclient# kill pid#
ypclient# cd /etc
ypclient# mv /usr/etc/ypbind /usr/etc/ypbind.orig
```

(where *pid#* is the `yp` pid found using the previous command)

To disable YP on a particular YP slave or master, do the following:

```
ypclient# mv /usr/etc/ypserv /usr/etc/ypserv.orig.
```

The install script does this automatically if you do not select `yp`.

Refer to Chapter 2 of this guide for information about the files you need to maintain for a server should you decide to disable YP.

YP User Programs

This section describes how to use four interface programs designed to help you access or modify data stored in yellow pages databases. Specifically, these programs—`ypcat`, `ypmatch`, `yppasswd`, and `ypwhich`—enable you to access selected data from various databases, modify your yellow pages password file, and to determine which machine is acting as a database server. None of these programs demand more than a casual knowledge of the yellow pages. `ypclnt`, also covered in this section, is actually a library of functions that enables you to create your own interface to the yellow

pages. Although `ypclnt` is more complicated to use than the other programs, individuals familiar with C and the ConvexOS operating system should have no problems with it.

ypcat

`ypcat` enables you to print values from the yellow pages maps stored on your machine or on other machines across the network. `ypcat` is typically used with a map name as an argument. The command sequence:

```
% ypcat hosts.byaddr
```

for example, prints the contents of the map hosts.byaddr, which contains a listing of the names and addresses of network hosts. You can also invoke a map by using a "nickname." For example:

```
% ypcat hosts
```

has the same effect as the previous example; hosts.byaddr and hosts are just different ways of referring to the same map. You can list the nicknames for maps in your domain by using the *-x* argument with `ypcat`. For example:

```
% ypcat -x
Use "passwd" for map "passwd.byname"
Use "group" for map "group.byname"
Use "networks" for map "networks.byaddr"
Use "hosts" for map "hosts.byaddr"
Use "protocols" for map "protocols.bynumber"
Use "services" for map "services.byname"
Use "aliases" for map "mail.aliases"
Use "ethers" for map "ethers.byname"
Use "rpc" for map "rpc.bynumber"
Use "pwrestrict" for map "pwrestrict.byname"
```

If you invoke `ypcat -x` on your machine, you get a good look at the maps that are available to you.

To display maps from different domains, use the `-d` switch. If, for example, you wanted to list networks.byname from the venus domain, you would use the following commands:

```
% ypcat -d venus networks
sun-ether     192.9.200          sunether ethernet localnet
loopback      127                loopback-net
sun-oldether  125                sunoldether
mktg-net      105
dragon-net    102
convex-net    100
ucb-ether     46                 ucbether
arpanet       10                 arpa
loopback-net  127
dragon-net    102
hyper-net     101                hyper
convex-net    100                convex
arpanet       10                 # the arpanet
```

ypmatch

ypmatch, like ypcat, enables you to access data stored in databases not only on your machine but on other machines connected to the network. Unlike ypcat, ypmatch enables you to get at *selected* parts of the database by printing information that matches *keys* that you provide. Suppose, for example, you want to check the Internet address of the machine *venus*. You could log in to *venus* remotely, or you could use ypcat to print the contents of the *hosts* database on your machine. A simpler method is to use ypmatch to search the *hosts* database for values associated with the *venus* key. For example:

```
% ypmatch venus hosts
100.0.5.1 venus
```

The keys you supply ypmatch must exactly match the keys in the map. If you supply a key for which no values can be associated, you receive an error message:

```
% ypmatch loopback networks
ypmatch: Can't match loopback. Reason: no such key in map.
```

As with ypcat, you can use the *-x* option to display the map nickname table:

```
% ypmatch -x
  Use "passwd" for map "passwd.byname"
  Use "group" for map "group.byname"
  Use "networks" for map "networks.byaddr"
  Use "hosts" for map "hosts.byname"
  Use "protocols" for map "protocols.bynumber"
  Use "services" for map "services.byname"
  Use "aliases" for map "mail.aliases"
  Use "ethers" for map "ethers.byname"
  Use "rpc" for map "rpc.bynumber"
  Use "pwrestrict" for map "pwrestrict.byname"
```

You can also use the -d flag to access data in a remote database, just as you did with ypcat:

```
% ypmatch -d eclipse docstaff aliases
moe, curly, larry
```

When you specify the *-k* option, the key is printed (with a colon) before the value of the key. This option is useful primarily for improving the clarity of the output you receive when you specify more than one key. Here's an example:

```
% ypmatch -k moe curly larry passwd
moe: moe:VPF75JjbUCrvw:107:51:Moe,88/487,295,3415009:/doc/moe:/bin/csh
curly: curly:WTqWgH2mHeFRc:172:51:Curly,89/488,296,5307495:/doc/curly:/bin/csh
larry: larry:VL.fEiCaWwb/w:133:51:Larry,90/49,297,5273452:/doc/larry:/bin/csh
```

yppasswd

You can use yppasswd to change or install your Yellow Pages password. (As you remember from the previous discussion of maps, user passwords are usually one of the first things networked when YP is installed. The YP password is much like a typical machine password, but it can be used to sign onto many machines across the network.) Your YP password may be different from the one on your own machine.

Adding (or changing) a YP password is just like adding or changing your password on a CONVEX supercomputer. You enter `yppasswd` at a command prompt to invoke the program; then you enter your old password, followed by the new one. As with `passwd` on a CONVEX supercomputer, you must enter the new password twice. Here's an example of a typical `yppasswd` terminal session. (You may receive some messages not listed here, depending on how you use the command):

```
% yppasswd
Old yp passwd: enter_old_password_here
New yp passwd: enter_new_password_here
Retype new passwd: enter_new_password_here
```

Your new password must be at least six characters long. If you are not the password "owner," you must be a superuser to change a password. In either case, you must prove you know the old password. The update protocol passes all the information to the server in one RPC call, without ever looking at it. Thus if you enter your old password incorrectly, you are not notified until after you have entered your new password.

If you try to change your password on a machine that is not running the `yppasswd` daemon (`/usr/etc/rpc.yppasswdd`), you receive a message similar to the following:

```
% yppasswd
convex1 is not running yppasswd daemon
```

If you receive this message, log in to the `passwd` master server and try again.

ypwhich

`ypwhich` provides a general query function that enables you to determine which maps are available to you, which server is supplying yellow pages services, and which machine is acting as master for a particular map.

In the simplest case, you can use `ypwhich` just as you did `ypcat` and `ypmatch` to display a list of maps and their nicknames. You do this by using the `-x` flag, as follows:

```
% ypwhich -x
Use "passwd" for map "passwd.byname"
Use "group" for map "group.byname"
Use "networks" for map "networks.byaddr"
Use "hosts" for map "hosts.byaddr"
Use "protocols" for map "protocols.bynumber"
Use "services" for map "services.byname"
Use "aliases" for map "mail.aliases"
Use "ethers" for map "ethers.byname"
Use "rpc" for map "rpc.bynumber"
Use "pwrestrict" for map "pwrestrict.byname"
```

The `-m` flag enables you to find out which machine is the master server for a particular map. Suppose, for example, that you want to find out which machine is the master server for the RPC map you're using. Simply invoke `ypwhich` with the `-m` switch and the name of the map:

```
% ypwhich -m rpc
convex1
```

In this example, *convex*1 is the master server for the RPC map.

To locate master servers for maps in other domains, use the -d switch in conjunction with the -m switch:

```
% ypwhich -d eclipse -m networks
  convex2
```

The -V1 and -V2 flags allow you to determine, respectively, which server is supplying Version 1.0 and Version 2.0 of the Yellow Pages. (Version 1.0 is provided for backward compatibility with older systems from other vendors. Version 2.0 is used as the default CONVEX version of YP.) For example:

```
% ypwhich -V1
  convex1

% ypwhich -V2
  convex1
```

You can print a list of every map in a domain (with its associated master server) by invoking the -m flag without arguments. For example:

```
% ypwhich -m
bootparams convex1
mail.aliases convex1
pwrestrict.byuid convex1
pwrestrict.byname convex1
netgroup.byhost convex1
netgroup.byuser convex1
netgroup convex1
protocols.byname convex1
services convex1
services.byname convex1
rpc.byname convex1
rpc.bynumber convex1
networks.byaddr convex1
networks.byname convex1
ethers.byname convex1
ethers.byaddr convex1
hosts.byaddr convex1
hosts.byname convex1
group.bygid convex1
group.byname convex1
passwd.byuid convex1
protocols.bynumber convex1
ypservers convex1
passwd.byname convex1
```

To list maps in other domains, use the -d switch in conjunction with the -m switch:

```
% ypwhich -d eclipse -m
bootparams eclipse
mail.aliases eclipse
netgroup.byhost eclipse
netgroup.byuser eclipse
netgroup eclipse
pwrestrict.byuid eclipse
```

```
pwrestrict.byname eclipse
protocols.byname eclipse
services eclipse
services.byname eclipse
rpc.byname eclipse
rpc.bynumber eclipse
networks.byaddr eclipse
networks.byname eclipse
ethers.byname eclipse
ethers.byaddr eclipse
hosts.byaddr eclipse
hosts.byname eclipse
group.bygid eclipse
group.byname eclipse
passwd.byuid eclipse
protocols.bynumber eclipse
ypservers eclipse
passwd.byname eclipse
```

ypclnt

`ypclnt` is a package of library functions that you can use to build your own interfaces to the yellow pages. These functions are designed to be used in *programs;* they are not stand-alone programs, and do not do you much good if you are not a C programmer. This section describes the available functions and provides examples. Refer to the ypclnt(3N) manual page for more information and for a look at the functions themselves.

The functions available to you include the following:

`yp_bind`	Binds client program to a given yp server (domain)
`yp_bind`	Unbinds client programs from given yp domain
`yp_bind`	Returns default system domain name
`yp_bind`	Matches a key in a given yp map
`yp_bind`	Returns first entry in a yp map
`yp_bind`	Returns next entry in a yp map
`yp_bind`	Transfers an entire map in one TCP request
`yp_bind`	Returns map order number (a date stamp in seconds)
`yp_bind`	Returns machine name of master server for a given map
`yp_bind`	Returns string describing a given error
`yp_bind`	Converts YP protocol error to `ypclnt` error code

Figures 4-3, 4-4, and 4-5 illustrate the use of `ypclnt` functions.

Figure 4-3—Using ypclnt, example 1

```
1       #include <stdio.h>
2       #include <rpcsvc/ypclnt.h>
3
4       #define MAP "passwd.byname"
5
6       main()
7       {
8            char *domain;
9            int err;
10           char *key, *old_key, *val;
11           int keylen, old_keylen, vallen;
12
13           if ((err = yp_get_default_domain(&domain)) != 0) {
14                fprintf(stderr, "demo: %s\n", yperr_string(err));
15                exit(1);
16           }
17           if ((err = yp_bind(domain)) != 0) {
18                fprintf(stderr, "demo: %s\n", yperr_string(err));
19                exit(1);
20           }
21           if ((err = yp_first(domain, MAP, &old_key, &old_keylen,
22               &val, &vallen)) != 0) {
23                fprintf(stderr, "demo: %s\n", yperr_string(err));
24                exit(1);
25           }
26           printf("%s", val);
27           while (yp_next(domain, MAP, old_key, old_keylen, &key, &keylen,
28               &val, &vallen) == 0) {
29                printf("%s", val);
30                old_key = key;
31                old_keylen = keylen;
32           }
33           exit(0);
34      }
```

Figure 4- 3 shows the use of `ypclnt` functions `yp_get_default_domain` (at line 13); `yperr_string` (at line 14); `yp_bind` (at line 17); `yp_first` (at line 21); and `yp_next` (at line 27). The program finds the given domain, binds to it, and then reads and prints the entries in the file MAP. In function, this program is essentially a smaller version of `ypcat`.

Figure 4-4—Using `ypclnt`, example 2

```
1       #include <stdio.h>
2       #include <rpc/rpc.h>
3       #include <rpcsvc/ypclnt.h>
4       #include <rpcsvc/yp_prot.h>
5
6       #define MAP "passwd.byname"
7
8       extern int map_foreach();
9
10      main()
11      {
12              char *domain;
13              int err;
14              char *key, *old_key, *val;
15              int keylen, old_keylen, vallen;
16              struct ypall_callback callback;
17
18              if ((err = yp_get_default_domain(&domain)) != 0) {
19                      fprintf(stderr, "demo: %s\n", yperr_string(err));
20                      exit(1);
21              }
22              if ((err = yp_bind(domain)) != 0) {
23                      fprintf(stderr, "demo: %s\n", yperr_string(err));
24                      exit(1);
25              }
26              callback.foreach = map_foreach;
27              if ((err = yp_all(domain, MAP, &callback)) != 0) {
28                      fprintf(stderr, "demo: %s\n", yperr_string(err));
29                      exit(1);
30              }
31              exit(0);
32      }
33
34      map_foreach(instatus, inkey, inkeylen, inval, invallen, indata)
35              int instatus;
36              char *inkey;
37              int inkeylen;
38              char *inval;
39              int invallen;
40              char *indata;
41      {
42              if (instatus != YP_TRUE) {
43                      return(1);
44              }
45              inkey[inkeylen] = inval[invallen] = '\0';
46              printf("%s\n", inval);
47              return(0);
48      }
```

Figure 4-4 is a rewritten version of Figure 4-1, in which `yp_all`, rather than `yp_first` and `yp_next`, is used. `yp_all` typically works much faster than the `yp_first/yp_next` combination, especially when it is likely that you need to read an entire map. When the code in Figures 4-3 and 4-4 were run across a sample *passwd* file, the code in Figure 2 ran 500% faster. (Specifics: passwd file contained 182 entries and 14135 characters; time to read/print with code in Figure 4-3 = 10 seconds, while code in Figure 4-4 read and printed the file in 2 seconds.)

In Figure 4-4 the `inkey` and `inval` entries are not null-terminated. You are given the lengths of each string as `inkeylen` and `invallen`—you should null-terminate *these* strings, as shown at line 45.

Figure 4-5—Using `ypclnt`, example 3

```
1       #include <sys/types.h>
2       #include <sys/time.h>
3       #include <stdio.h>
4       #include <rpcsvc/ypclnt.h>
5
6       #define MAP "passwd.byname"
7       #define KEY "daemon"
8
9       main()
10      {
11              char *domain, *master;
12              int err;
13              char *key, *val;
14              int keylen, vallen;
15              int order;
16
17              if ((err = yp_get_default_domain(&domain)) != 0) {
18                      fprintf(stderr, "demo: %s\n", yperr_string(err));
19                      exit(1);
20              }
21              if ((err = yp_bind(domain)) != 0) {
22                      fprintf(stderr, "demo: %s\n", yperr_string(err));
23                      exit(1);
24              }
25              if ((err = yp_match(domain, MAP, KEY, strlen(KEY), &val, \
&vallen)) != 0) {
26                      fprintf(stderr, "demo: %s\n", yperr_string(err));
27                      exit(1);
28              }
29              printf("%s", val);
30              if ((err = yp_order(domain, MAP, &order)) != 0) {
31                      fprintf(stderr, "demo: %s\n", yperr_string(err));
32                      exit(1);
33              }
34              printf("Map %s has order number %d -- %s", MAP, order, \
ctime(&order));
35              if ((err = yp_master(domain, MAP, &master)) != 0) {
36                      fprintf(stderr, "demo: %s\n", yperr_string(err));
37                      exit(1);
38              }
39              printf("The master server for map %s is %s\n", MAP, master);
40              exit(0);
41      }
42
```

Figure 4-5 shows the use of `yp_match` (at line 25); `yp_order` (at line 30); and `yp_master` (at line 35). The program binds to a specified domain, and then lists values in MAP that match the KEY argument. `yp_match` is used, of course, to search MAP for a particular KEY. `yp_order` and `yp_master` are typically not used for application programs. They return the time that a map was built and the name of the machine that is the master server for MAP, respectively.

The output for Figure 4-5 is shown here:

```
daemon:6c2Lz36Q10Mc.:1:1:Our friend, the daemon:/:/bin/csh
Map passwd.byname has order number 532161701 -- Wed Nov 12 00:41:47
1986
The master server for map passwd.byname is convex1
```

Secure Networking 5

The authentication system employed by CONVEX NFS greatly enhances the security of network environments. The system is general enough to be used by a variety of operating systems. The system uses Data Encryption Standard (DES) encryption and public key cryptography to authenticate both users and machines in the network.

➜ Note

Secure NFS relies on facilities provided by the Yellow Pages (YP). Therefore, you must run YP if you wish to use Secure NFS.

To successfully access files on an NFS file system mounted with the `-secure` option, each user must have a pair of keys in the key database of the server machine. Users and remote hosts are identified by a *netname* unique for that user across the entire network; the netname looks like:

```
unix.1927@convex1.convex.comfor users

unix.p4@convex1.convex.com  for remote hosts
```

The keys are 192-bit binary numbers.

Key Database Files

DES authentication revolves around the key database mentioned above. This database consists of three files:

- /etc/publickey (also available as the publickey.byname YP map) contains keys for each user and machine on the network. The file is a set of ASCII records of the form:

  netname publickey:secretkey
  (48 bytes each)

  The secretkey stored here is encrypted with the user's password to ensure it stays secret. The superuser can place entries in this file and update the YP map by using `newkey`, while users may change their existing keys with `chkey`.

- /etc/keystore is managed exclusively by the keyserver daemon, and is a list of the users on the local machine who are trusted to use Encrypted RPC. It contains binary records of the form:

  uid (2 bytes) *secretkey (24 bytes == 192 bits)*

  Each secretkey stored here is encrypted with the secretkey for the local superuser to keep them secret. Each user needing to use Encrypted RPC must have an entry in this file; the entry is made automatically whenever a user logs in and types a password. Any user who logged in without typing a password does not have an entry and cannot use Secure NFS; users can get around this by running `keylogin` and typing the correct password.

- /etc/.rootkey is also managed exclusively by the keyserver daemon, and contains the secretkey for root, unencrypted. The permissions on this file must disallow access to any user other than root. It is simply an ASCII file terminated with a newline, like this:

  `54918c47567b569644517cad61033675898519f4b1d7c575`*(48 bytes)*

  The root key is written out in this way to permit a machine to reboot after a power failure without a superuser having to be present to type a password. If this was not done, Secure NFS mounts would be inaccessible to everyone until the superuser logged in. This file is not written by the keyserver until root logs in properly. Secure NFS also fails for all users and systems if root has not logged in properly.

These three files should be backed up along with /etc/passwd and other critical files.

Secure NFS Key Management

Getting keys in the key database can be a logistical problem. As system manager you can run `newkey` for each user account for which you know the password, but this is not practical for existing installations. Root keys for other machines can also be a problem, as root passwords should be very well-guarded. Fortunately, there are workarounds.

The `newkey` and `chkey` programs communicate with the YP update daemon to make sure that not only /etc/publickey on the master server but also the YP publickey.byname map are updated correctly. That means that any user in a given YP domain can update existing keys reliably. Superusers of other hosts in the domain may use `newkey` to set their machine up for Secure NFS at any time, and while users may be denied the ability to make their first keys, they may change a set of existing keys at any time. These changes propagate normally via `yppush`, `yppoll`, and `ypxfr`.

To avoid running `newkey` or `passwd` for every user on an existing system, you can permit users to enter their own new keys with `chkey` by starting up the YP update daemon, rpc.ypupdated, with a `-i` flag. `chkey` normally tries to use the Encrypted RPC protocol itself, which won't work without a proper set of keys in place, but it will also try to use normal RPC if that does not work. The `-i` flag on rpc.ypupdated allows it to accept normal RPC, which means that users without keys can get a new pair. This mode of operation is less secure, so only use the `-i` option for a week or so and add something to /etc/motd asking your users to run `chkey` once at some time within that week.

Communicating with hosts in another YP domain is less automatic; you must be using the same key pair as the other domain. The simplest way to accomplish this is as follows. (This procedure must be followed for both domains.)

1. Have the other machine's system manager cut out the /etc/publickey entries for root and all users who need to use your machine and send these entries to you via electronic mail. Use an editor to add these entries to the /etc/publickey file on your machine.
2. For each of the users from the other machine, add a line like

   `unix.3900@snoopy 7100:10,33`

   to /etc/netid. The example shown allows the person with userid 3900 on the other system access to files with userid 7100, and to files owned by groups 10 and 33, on your machine. You may give the user any userid and group list you like, as long as you list at least one group after the colon.
3. After you finish editing /etc/publickey and /etc/netid, run `make` in /usr/etc/yp to update the YP database.

You must perform this procedure any time a key used in both domains changes.

Secure NFS Configuration

To get Secure NFS up and running, set up the key database and start the new daemons as explained in the following steps:

1. Ensure that 'root' has a password.
2. Verify that /etc/inetd.conf contains the following lines:

```
time    stream  tcp     nowait  root    internal
time    dgram   udp     nowait  root    internal
```

3. Verify that /etc/services contains the following lines:

```
time            37/tcp          timserver
time            37/udp          timserver
```

4. Edit /etc/rc.local and ensure that the domainname is set to a valid domain and that the new Secure daemons, `rpc.ypupdated` and `keyserv`, are started up after `ypserv` and `ypbind`.
5. Use `touch` to create empty /etc/netid and /etc/publickey files.
6. Configure the key database for all local and remote users who will need access to Secure file systems and for all machines with which you will be sharing files. This may be done in several ways; refer to "Secure NFS Key Management" above for more details. In any case, enter a root key using:

   ```
   newkey -h <localhostname>
   ```

   Enter the root password at the prompt. To add more hosts, repeat the `newkey` command with the appropriate hostname and password. If all of your hosts are on the same subnet, it may be easier to wait until the YP update daemon is running and let the system managers of the individual machines run `newkey` themselves, since the keys will propagate.

   To add local users' keys, use:

   ```
   newkey -u <username>
   ```

7. Change your directory to /usr/etc/yp to get Yellow Pages set up. If you are not currently running YP, run `ypinit` and answer the prompts; if you are currently running YP, update the maps with `make`.
8. Add the secure option to any /etc/fstab entries you want to mount with Secure NFS, and add the secure option to any /etc/exports entries you want to force client machines to mount with Secure NFS. Note that you must currently have at least one root-access entry on a secure export; use `root=`*localhostname* if there is no other host that should have root access.
9. Start the new `rpc.ypupdated` and `keyserv` daemons; this may be done by hand or by bringing the machine up from single-user mode. If you do start the daemons by hand, you may want to unmount and remount any file systems that should now be using Secure NFS.

At this point, Secure mounts should work. If the mount attempt succeeds but the key database is incorrect, any accesses of the Secure file system will fail, even if performed by root, unless anon=0 is set in the exports entry.

All users (except root) must now use `yppasswd` instead of `passwd` to keep their secret key synchronized with their login password (out of necessity, `passwd` re-encrypts the secret key for root). As a consequence, it is a bad idea to have entries for individual users in local /etc/passwd files. Furthermore, it might be a good idea to overwrite passwd with yppasswd, after saving passwd as opasswd.

10. When you reinstall, move, or upgrade a machine, save /etc/keystore, /etc/publickey and /etc/.rootkey along with everything else you normally save.

If you login, rlogin or telnet to another machine, are asked for your password, and type it correctly, you've given away access to your own account. This is because your secret key is now stored in /etc/keystore on that remote machine. This is only a concern if you don't trust the remote machine. If this is the case, don't ever log in to a remote machine if it asks for your password. Instead, use NFS to remote mount the files you're looking for. At this point there is no keylogout command.

The remainder of this chapter discusses the theory of secure networking, and is useful as a background for both users and managers.

Security in Basic NFS

An NFS server authenticates a file request by authenticating the machine where the request originates, but not the user making the request. On NFS-based file systems, running `su` allows you to impersonate the rightful owner of a file. The familiar command `rlogin` is subject to exactly the same attacks as NFS because it uses the same kind of authentication.

A common solution to network security problems is to leave the solution to each application. Secure NFS puts authentication at the RPC level. The result is a standard authentication system that covers all RPC-based applications, such as NFS and the Yellow Pages. This system allows authentication of users as well as machines, and makes a network environment more like the older time-sharing environment. Users can log in on any machine, just as they could log in on any terminal. Their login password is their passport to network security. No knowledge of the underlying authentication system is required.

Given root access and a good knowledge of network programming, anyone is capable of injecting arbitrary data into the network and picking up any data from the network. However, on a local area network, no machine is capable of packet smashing —capturing packets before they reach their destination, changing the contents, then sending packets back on their original course — because packets reach all machines, including the server, at the same time. Packet smashing is possible on a gateway, though, so make sure you trust all gateways on the network.

The most dangerous attacks involve injection 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 has not been compromised. You can protect the privacy of sensitive information by encrypting data that goes over the network.

RPC Authentication

RPC is at the core of the network security system. To understand the big picture, you must understand how authentication works in RPC. RPC's authentication is open-ended: a variety of authentication systems may be plugged into it and may coexist on the network. DES authentication, the system used by Secure NFS, is discussed in this chapter.

Two terms are important for any RPC authentication system: credentials and verifiers. Using ID badges as an example, the credential is what identifies a person: a name, address, birth date, etc. 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.

RPC uses a similar system. The client process sends both a credential and a verifier to the server with each RPC request. The server sends back only a verifier, since the client already knows the server's credentials.

UNIX Authentication

Before the introduction of Secure NFS, most CONVEX network services used UNIX authentication. The credentials contain the client's machine-name, UID, GID, and group-access-list. The verifier contains nothing. There are two problems with this system. One is the empty verifier, which makes it easy to cook up the right credential using hostname and `su`. If you trust all root users in the network, this is not really a problem, but many networks — especially at universities — are not this secure.

NFS combats deficiencies in this authentication by checking the source Internet address of mount requests as a verifier of the hostname field and accepting requests only from privileged Internet ports. (For details on enabling privileged port checking by NFS, see "Checking Privileged Ports" in Chapter 1 of this guide.) Still, it is not difficult to circumvent these measures, and NFS really has no way to verify the user ID.

With UNIX authentication, it is unrealistic to assume that all machines on a network are UNIX machines. The NFS works with MS-DOS and VMS machines, but UNIX authentication breaks down when applied to them. For instance, MS-DOS doesn't even have a notion of different user IDs.

DES authentication addresses these shortcomings.

DES Authentication

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

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

If a network has time synchronization (`timed`, for example), then client/server time synchronization is performed automatically. However, if this is not available, time stamps can be computed using the server's time instead of network time. To do this, the client asks the server what time it is before starting the RPC session, then computes the time difference between its own clock and the server's. This difference is used to offset the client's clock when computing time stamps. If the client and server clocks get out of sync to the point where the server begins rejecting the client's requests, the DES authentication system just re-synchronizes with the server.

Here's how the client and server arrive at the same encryption key. When a client wishes to talk to a server, it generates at random a key to be used for encrypting the time stamps (among other things). This key is known as the conversation key, CK. The client encrypts the conversation key using a public key scheme and sends it to the server in its first transaction. This key is the only thing that is ever encrypted with public key cryptography. The particular scheme used is described further in the "Public Key Encryption" section. For now, suffice to say that, for any two agents A and B, there is a DES key K_{AB} that only A and B can deduce. This key is known as the common key, K_{AB}.

Figure 5-1 The DES Authentication Protocol

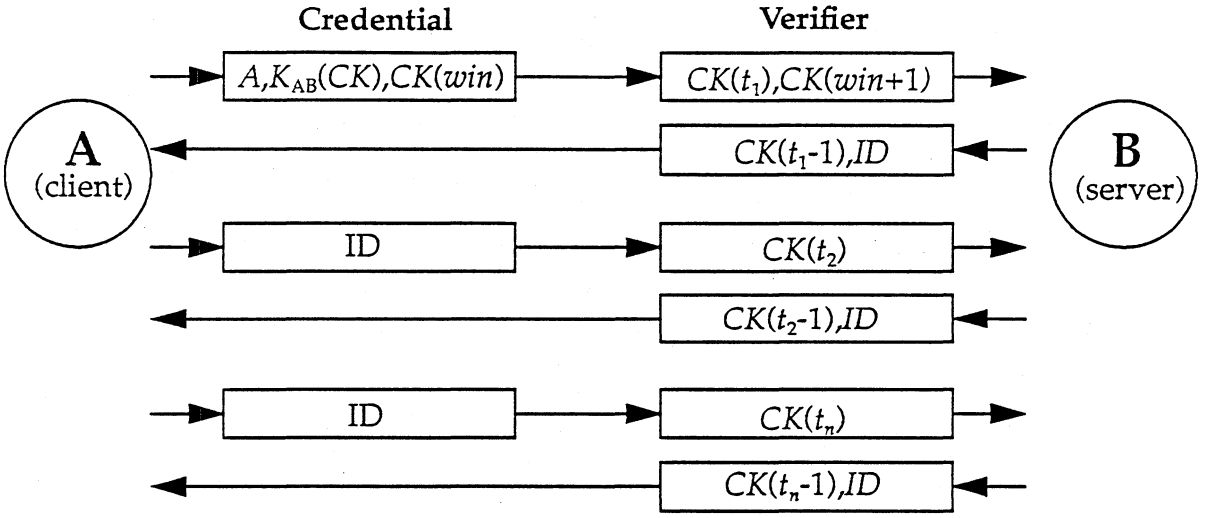


Figure 5-1 above illustrates the authentication protocol in more detail, describing client A talking to server B. A term of the form $K(x)$ means x encrypted with the DES key K. Examining the figure, you can see that, for its first request, the client's credential contains three things: its name *A*, the conversation key *CK* encrypted with the common key K_{AB}, and a thing called *win* (window) encrypted with *CK*. What the window says to the server, in effect, is:

> I will be sending you many credentials in the future, but there may be imposters sending them too, trying to impersonate me with bogus time stamps. When you receive a time stamp, check to see if your current time is somewhere between the time stamp and the time stamp plus the window. If it's not, please reject the credential.

For secure NFS file systems, the window currently defaults to 30 minutes. The client's verifier in the first request contains the encrypted time stamp and an encrypted verifier of the specified window, *win + 1*.

Someone could attempt to impersonate A by writing a program that instead of filling in the encrypted fields of the credential and verifier, just stuffs in random bits. The server decrypts CK into some random DES key, and uses it to decrypt the window and the time stamp. These just end up as random numbers. After a few thousand trials, there is a good chance that the random window/time stamp pair would pass the authentication system. The window verifier makes guessing the right credential much more difficult.

After authenticating the client, the server stores four things into a credential table:

- the client's name (A)
- the conversation key *(CK)*
- the window
- the time stamp

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

The server sends its verifier back to the client; the verifier contains an index ID into the server's credential table, plus the client's time stamp minus one, encrypted by the conversation key, *CK*. The client knows that only the server could have sent such a verifier, since only the server knows what time stamp the client sent and what conversation key is being used. The reason for subtracting one from it is to ensure that it is invalid and cannot be reused as a client verifier.

The first transaction is rather complicated, but after this things are much simpler. The client just sends its ID and an encrypted time stamp to the server, and the server sends back the client's time stamp minus one, encrypted by *CK*.

Public Key Encryption

The particular public key encryption scheme used is the Diffie-Hellman method. This algorithm generates a secret key SK_A at random and computes a public key PK_A using the following formula:

$$PK_A = \alpha^{\{SK_A\}}$$

where *PK* and *SK* are 192 bit numbers and α is a well-known constant. Public key PK_A is stored in a public directory, but secret key SK_A is kept private. Next, PK_B is generated from SK_B in the same manner as above. Now common key K_{AB} can be derived as follows:

$$K_{AB} = PK_B^{\{SK_A\}} =$$
$$(\alpha^{\{SK_B\}})^{\{SK_A\}} =$$
$$\alpha^{\{SK_A SK_B\}}$$

Without knowing the client's secret key, the server can calculate the same common key K_{AB} in a different way, as follows:

$$K_{AB} = PK_A^{\{SK_B\}} =$$
$$(\alpha^{\{SK_A\}})^{SK_B} =$$
$$\alpha^{\{SK_A SK_B\}}$$

No one else but the server and client can calculate K_{AB}, since doing so requires knowing either one secret key or the other. This arithmetic is actually computed modulo *M*, which is another well-known constant. It would seem at first that somebody could guess your secret key by taking the logarithm of your public one, but *M* is so large that this is a computationally infeasible task. To be secure, K_{AB} has too many bits to be used as a DES key, so 56 bits are extracted from it to form the DES key.

Both the public and the secret keys are stored indexed by netid in the Yellow Pages map publickey.byname. The secret key is DES-encrypted with your login password. When you log in to a machine, the `login` program grabs your encrypted secret key, decrypts it with your login password, and gives it to a secure local keyserver to save for use in future RPC transactions. Ordinary users do not have to be aware of their public and secret keys. In addition to changing your login password, the `yppasswd` program randomly generates a new public/secret key pair as well.

The keyserver, `keyserv`, is an RPC service local to each machine that performs all of the public key operations, of which there are only three. They are:

- `setsecretkey` (*secretkey*)
- `encryptsessionkey` (*servername, des_key*)
- `decryptsessionkey` (*clientname, des_key*)

`setsecretkey` tells the keyserver to store away your secret key SK_A for future use; it is normally called by `login`. The client program calls `encryptsessionkey` to generate the encrypted conversation key that is passed in the first RPC transaction to a server.

The keyserver looks up *servername* public key and combines it with the client's secret key (set up by a previous `setsecretkey` call) to generate the key that encrypts *des_key*. The server asks the keyserver to decrypt the conversation key by calling `decryptsessionkey`.

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, since it would create deadlock. The keyserver solves this problem by storing the secret keys by UID and only granting requests to local root processes.

The client process then executes `keyenvoy`, a `setuid` process owned by root, which makes the request on the part of the client, telling the keyserver the real UID of the client. Ideally, the three operations described above would be system calls, and the kernel would talk to the keyserver directly, instead of executing the `setuid` program.

Naming Network Entities

Recall that with UNIX authentication, the name of a network entity is basically the UID. UIDs are assigned per Yellow Pages naming domain, which typically spans several machines.

When domains are linked together, UID clashes become possible. Also, the superuser (with UID of 0) should not be assigned on a per-domain basis, but rather on a per-machine basis. By default, NFS deals with this latter problem by not allowing root access across the network by UID 0 at all.

DES authentication bases naming upon the convention of *netnames*. Simply put, a netname is just a string of printable characters, and fundamentally, it is really these netnames that are authenticated. The public and secret keys are stored on a per-netname, rather than per-username, basis. The Yellow Pages map netid.byname maps the netname into a local UID and group-access-list; some environments may map the netname into something else.

Netnames are globally unique. This is far easier then choosing globally unique UIDs. In the CONVEX environment, user names are unique within each Yellow Page domain. Netnames are assigned by concatenating the operating system type and user ID with the Yellow Pages and ARPA domain names. For example, a ConvexOS system user with a user ID of 508 in the domain eng.convex.COM would be assigned the netname unix.508@eng.convex.COM. A good convention for naming domains is to append the ARPA domain name (COM, EDU, GOV, MIL) to the local domain name. Thus, the Yellow Pages domain eng within the ARPA domain convex.COM becomes eng.convex.COM

Netnames are assigned to machines as well as to users. A machine's netname is formed much like a user's. For example, a UNIX machine named hal in the domain eng.convex.COM has the netname unix.hal@eng.convex.COM. Proper authentication of machines is very important for diskless machines that need full access to their home directories over the net.

Other environments have other ways of generating netnames, but this does not preclude them from accessing the secure network services of the CONVEX environment. To authenticate users from any remote domain, you must make entries for them in two YP databases. One entry is for their public and secret keys, the other is for their local UID and group-access-list mapping. Once these entries exist, users in the remote domain will be able to access all of the local network services, such as the NFS and remote logins.

Applications of DES Authentication

A generalized YP update service is one application of DES authentication. This service allows users to update private fields in YP databases. The YP maps hosts, ethers, bootparams, and publickey employ the DES-based update service.

The more secure Network File System explained in this chapter is another application of DES authentication. There are three security problems with the old NFS using UNIX authentication. The first is that verification of credentials occurs only at mount time when the client gets its key to all further requests, the file handle, from the server. Someone can break security by obtaining a file handle without contacting the server, perhaps by tapping into the net or by guessing. After an NFS file system has been mounted, there is no checking of credentials during file requests, which brings up the second problem.

If a file system has been mounted from a server that serves multiple clients (as is typically the case), there is no protection against someone who has root permission on their machine using su (or some other means of changing uid) gaining unauthorized access to other people's files. The third problem with the NFS is the severe method it uses to circumvent the problem of not being able to authenticate remote client superusers: denying them super-user access altogether.

The new authentication system corrects all of these problems. Guessing file handles is no longer a problem since, in order to gain unauthorized access, would-be trespassers must guess the right encrypted time stamp to place in the credential, which is a virtually impossible task. The problem of authenticating root users is solved, since the new system can authenticate machines.

Actually, the level of security associated with each file system may be altered by the manager. The file /etc/exports contains a list of file systems and the machines that may mount them. By default, file systems are exported with UNIX authentication, but the manager can have them exported with DES authentication by specifying -secure on any line in the /etc/exports file. Associated with DES authentication is a parameter: the maximum window size that the server is willing to accept.

Security Issues Remaining

There are ways to break DES authentication, but most of them are very difficult or can be prevented by a few precautionary measures. Imposters could break the DES key itself, or compute the logarithm of the public key; however, either of these options would require months of compute time on a supercomputer.

Probably the easiest way to circumvent security is to guess someone's password since, unfortunately, many people choose easily guessable passwords. There is no way for software to protect against guessing; it's up to each user to choose a secure password.

The next easiest attack is to attempt replays. For example, suppose someone records all of your NFS transactions with a particular server. As long as the server remains up, there is no point in replaying those transactions since the server always demands time stamps that are greater than the previous ones seen. But suppose that same someone now pulls the plug on your server, causing it to crash. As it reboots, its credential table will be clean, so it will have no track of previously seen time stamps and will accept replayed transactions.

For starters, keep your servers in a secure place so that no one can pull the plug on them. However, all servers occasionally crash without any help. To protect against replayed transactions in this case, specify a window size that is smaller than the time it takes a server to reboot (5 to 10 minutes). The server will reject any replayed transactions because they will have expired.

One security issue DES authentication does not address is tapping of the net. Even with DES authentication in place, there is no protection against someone watching net traffic. (Though trying to make sense of all the bits flying over the net is not a trivial task.) You do not want someone reading your password from the net, so logins pose a bit of a problem. Since a key exchange is a side effect of the authentication system, the network tapping problem can be tackled on a per-application basis.

Anyone attempting to break DES authentication cannot use su to do so. In order to be authenticated, your secret key must be stored by your workstation. This usually occurs when you login; the login program decrypts your secret key with your login password and stores it away for you.

If somebody tries to use su to impersonate you, they won't be able to decrypt your secret key. Editing /etc/passwd isn't going to help them either, because the thing they need to edit, your encrypted secret key, is stored in the Yellow Pages. If you log into somebody else's workstation and type in your password, then your secret key would be stored in their workstation and they could use su to impersonate you. To avoid this problem and others, do not give your password to a machine you do not trust. Someone on that machine could just as easily change login to save all the passwords it sees into a file.

Performance

Public key systems are known to be slow, but secure NFS does not perform much actual public key encryption. Public key encryption only occurs in the first transaction with a service, and even then, caching speeds things up considerably. The first time a client program contacts a server, both it and the server must calculate the common key. The time it takes to compute the common key is basically the time it takes to compute an exponential modulo M. You have to wait only the first time you contact a machine. Since the keyserver caches the results of previous computations, it does not have to recompute the exponential every time.

The extra overhead that DES authentication requires versus UNIX authentication is the encryption. A time stamp is a 64-bit quantity, which also happens to be the DES block size. Four encryption operations take place in an average RPC transaction: the client encrypts the request time stamp, the server decrypts it, the server encrypts the reply time stamp, and the client decrypts it.

Problems with Booting and setuid Programs

Consider the problem of a machine rebooting, say after a power failure at some strange hour when nobody is around. All of the secret keys that were stored get wiped out, and now no process will be able to access secure network services. The important processes at this time are usually root processes, so things would work if root's secret key were stored away; however, nobody is around to type the password that decrypts the key.

The solution to this problem is to store root's decrypted secret key in a file which the keyserver can read. This works well for diskfull machines that can store the secret key on a physically secure local disk, but is not secure for diskless machines, whose secret key must be stored across the network. If you tap the net when a diskless machine is booting, you will find the decrypted key. Such tapping is not easy to accomplish, though.

Another booting problem is the single-user boot. In the single-user booting mode, a root login shell appears on the console. The problem here is that single-user mode does not require a password.

Another problem arises from the fact that diskless machine booting is not totally secure. It is possible for someone to impersonate the boot-server, and boot a devious kernel that, for example, makes a record of your secret key on a remote machine. The present system is set up to provide 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.

Use of this method is unlikely, because anyone creating such a kernel must have access to kernel source code. Also, the crime is not without evidence. If you polled the net for boot-servers, you would discover the devious boot-server's location.

setuid programs not owned by root may pose some problems. For example, if a setuid program is owned by dave, who has not logged into the machine since it booted, then the program will not be able to access any secure network services as dave. Since root's secret key is always stored at boot time, root-owned programs behave as they always have.

Reporting Problems

This appendix introduces the CONVEX Technical Assistance Center (TAC) and the `contact` utility.

The `contact` utility is an online system for reporting problems to the TAC. To use it, enter **`contact`** at the system prompt and answer the questions as they appear on the screen.

This appendix describes:

- Prerequisites for using `contact`
- Tips for using `contact`
- The step-by-step process `contact` takes you through

Technical Assistance Center

The CONVEX Technical Assistance Center (TAC) is staffed by technical specialists who can address diverse questions and problems that arise in a supercomputing environment. If you have a hardware, software, or documentation question, contact the TAC. This group stands ready to solve such problems.

The `contact` Utility

The TAC recommends using the `contact` utility to report a hardware, software, or documentation problem. The `contact` utility is an interactive program that helps the TAC track reports and route them to the CONVEX personnel most qualified to fix a problem.

After you invoke `contact`, it prompts you for information about the problem. When you finish your report, `contact` electronically mails it to the TAC. The TAC notifies you within 48 hours that your report has been received.

To use `contact` requires:

- UNIX-to-UNIX Communication Protocol (UUCP) connection to the TAC
- Full path name of the program or utility in question
- Version number of the program or utility in question

UUCP Connection

Before using `contact`, ask your system administrator if your site has a UUCP connection to the TAC. A UUCP connection allows files to be copied from one UNIX--based system to another. The `uucp` (UNIX-to-UNIX copy) command relies on either a dial-up or hard-wired UUCP communication line.

Finding the Program Path Name

To determine the full path name of the program or utility in question, use the `which` command. Figure A-1 illustrates use of the `which` command to find the full path name of the loader (`ld`) utility.

Figure A-1
Using the `which` command

```
> which ld
/bin/ld
>
```

In this example, the full path name of the loader is /bin/ld.

If you use the C shell (`csh`), you can also use the `whence` command to find the program path name. The `whence` command works like `which`, but faster.

For more information on the `which` command, refer to the which(1) man page. You can also use the `info` online information system by entering **`info which`** at the system prompt.

Finding the Program Version Number

To determine the version number of the program or utility in question, use the `vers` command. Figure A-7 illustrates use of the `vers` command to find the version number of the loader (`ld`) utility. Enter **`vers`**, then the path name of the program or utility.

Figure A-2
Using the `vers` command

```
> vers /bin/ld
/bin/ld: 7.0
>
```

In this example, the loader version number is 7.0.

For more information on the `vers` command, refer to the vers(1) man page. You can also use the `info` online information system by entering **`info vers`** at the system prompt.

Using contact

The contact utility prompts for the following information:

- Your name, title, phone number, and corporate name
- Name and version of the product
- One-line summary of the problem
- Detailed description of the problem
- Priority of the problem
- Instructions on how to reproduce the problem
- Comments about the problem
- Comments about the documentation relating to the problem
- Files to include in the contact report

Following is a step-by-step discussion of these prompts.

Step 1a

To invoke the contact utility, enter **contact** at the system prompt. The system responds with a welcome message and a series of questions regarding your hardware, software, or documentation question. Figure A-3 illustrates use of the contact command and the resulting system response.

Figure A-3
Beginning a contact session

```
> contact
Welcome to contact version 0.11 ()

Enter your name, title, phone number, and corporate name (^D to terminate)
>
```

Step 1b

If there is a .contact file in your home directory, contact skips the first prompt. (Refer to "Using a .contact File" on page 6 for more information.) Figure A-4 illustrates the contact command and the system response when you have a .contact file in your home directory.

Figure A-4
Beginning a contact session
with a .contact file

```
> contact
Welcome to contact version 0.11 ()

Enter the name of the product involved
>
```

Step 2

The `contact` utility prompts for the version number of the product. If you do not know the version number, press **CTRL-Z** to suspend the session.

Use the `which` (or `whence` if you use `csh`) and `vers` commands to find the version number of the product. Use the `fg` command to return to the session, and enter the version number in the form X.X or X.X.X.X.

Step 3

The `contact` utility prompts for a one-line summary of the problem. This summary is the subject header in any further correspondence regarding the problem. Please make this summary as descriptive as possible in one line.

Step 4

The `contact` utility prompts for a detailed description of the problem. Please make this description as complete as possible. Include source code and a stack backtrace when possible. (Refer to the adb(1) or csd(1) man page for information on obtaining a stack backtrace.) The more information you provide, the quicker the TAC can isolate and solve the problem.

Step 5

The `contact` utility prompts for the priority of the problem. Figure A-5 illustrates this prompt and priority levels from which to choose. You must enter a priority number.

Figure A-5
Specifying priority of a problem

```
Enter a problem priority, based on the following:
1) Critical    - work cannot proceed until the problem is resolved.
2) Serious     - work can proceed around the problem, with difficulty.
3) Necessary   - problem has to be fixed.
4) Annoying    - problem is bothersome.
5) Enhancement - requested enhancement.
6) Informative - for informational purposes only.
>
```

Step 6

The `contact` utility prompts for an explanation of how to reproduce the problem. Please include the command syntax and options you used and anything else you did to make the program run.

Step 7

The `contact` utility prompts for any other pertinent comments. Please include all relevant information.

Step 8

The `contact` utility prompts for suggestions regarding documentation supporting the product. Indicate whether the documentation could be revised to address the problem.

Step 9

The `contact` utility prompts for names of files necessary to reproduce the problem. Figure A-6 illustrates this prompt and sample user response.

Figure A-6
Including files in a `contact` report

```
Are there any files that should be included in this report (yes | no)?
> yes
Please enter the names of the files, one to a line (^D to terminate)
> test.f
> ~/subroutines/sub.f
>
```

Note

"Tilde-Escape Sequences" on page 7 are not recognized in responses to this prompt. In `contact`, a tilde in this section indicates your home directory. This convention is based on use of the tilde for expanding file names in `csh`.

If files specified are small text files, they are automatically included in the `contact` report. If the files are too large to be included in this report, `contact` gives further instructions on how to submit these files.

To specify a directory, combine directory files into a single file using the `tar` command (refer to the tar(1) man page for further information) or enter each file name in the directory on a single line in the `contact` report.

Step 10

The `contact` utility prompts you to review, edit, submit, or abort the report. Figure A-7 illustrates this prompt.

Figure A-7
Prompt to review, edit, submit, or abort report

```
Please select one of the following options:
1) Review the problem report.
2) Edit the problem report.
3) Submit the problem report.
4) Abort the problem report.
>
```

Choose the number of the option you want to select. These options let you do the following:

Review	review the text of the `contact` report. You are then prompted again to select an option.
Edit	edit the text of the `contact` report. If you choose to edit the report, `contact` opens your default text editor.
Submit	sends the report to the CONVEX TAC. The TAC notifies you within 48 hours that your report has been received. Choosing this option exits the `contact` utility and returns you to the shell.
Abort	saves the text of the report in a file named ~/dead.report. Choosing this option exits `contact` and returns you to the shell.

Tips for Using contact

The contact utility is interactive and easy to use. This section lists tips to help use it efficiently. In particular, this section tells how to:

- Use a .contact file.
- Abort a contact session.
- Resubmit an aborted report.
- Suspend a contact session.
- Move within contact from one prompt to another.
- Use tilde-escape sequences in the contact utility.

Using a .contact File

When you invoke contact, it first prompts for your name, title, phone number, and company name. You can, however, create a .contact file to skip this first prompt.

Follow these steps to create a .contact file.

1. Create a .contact file in your home directory.
2. Enter your name, job title, phone number, and company name, each on a new line.

When you invoke contact, it automatically includes the .contact file as input for the first prompt and proceeds to the next prompt.

Aborting the Report

To abort a contact report, either press the interrupt key (usually **CTRL-C**) or choose the abort option when prompted by the contact utility. Using **CTRL-C** to abort does not save the contents of the report. Using the abort option saves the contents of the report in a file named ~/dead.report.

Submitting the dead.report File

After you abort a contact session, the contact utility saves the report in a file named ~/dead.report. Using the contact command with the -r option automatically merges the contents of the ~/dead.report file into the new contact session. Enter

```
contact -r
```

and contact finds the ~/dead.report file and merges it into the contact report. You can then edit the report. When you end the editing session, contact resumes at the final prompt, which asks you to review, edit, submit, or abort the report.

Suspending a Report

Sometimes it is necessary to stop in the middle of a `contact` report and return to the shell (for instance, to suspend the `contact` session to find the program path name or version number). To suspend the `contact` session, press **CTRL-Z**.

To return to the `contact` session, type **`fg`**. Using **CTRL-Z** and the `fg` (foreground) command, you can switch between the `contact` utility and the shell. You cannot, however, use **CTRL-Z** and `fg` to switch back and forth in the Bourne shell (`sh`).

Ending a Response

The `contact` utility prompts for information pertinent to your hardware, software, or documentation question. Some prompts require one-line responses; to move to the next prompt, press **RETURN**. Other prompts require more than a one--line response; to move to the next prompt, press **CTRL-D**.

Tilde-Escape Sequences

The `contact` utility treats input beginning with a tilde (~) as a special sequence. The character following the tilde is considered a request for a special function. You can use the following tilde sequences within `contact`:

~e start the text editor (defined in the EDITOR environment variable)

~h display a list of available tilde-escape sequences

~p print the `contact` report to the terminal screen

~r *filename* read the contents of *filename* as a response to the current prompt. Some prompts require only a one-line response. This tilde-escape sequence works only for prompts that allow more than a one-line response.

~~ insert a single tilde as the first character in the line

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

P

R

S

T

U

V

Y