FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

CURSOS A DISTANTANCIA

SERVIDORES DE CORREO Y LISTAS DE CORREO

DEL 21 DE NOVIEMBRE AL 19 DE DICIEMBRE DE 1998

CLAVE (CA 152)

PROFESOR:

ING. JAMES PATRICK KELLEGHAN MONGE

complemento

Section 9
sendmail Administration

Introduction

The **sendmail**(1M) program implements a general-purpose internetwork mail routing facility under the UNIX operating system. It features aliasing, forwarding, automatic routing to network Gateways, and flexible configuration features. The **sendmail** program came into being because mailing through heterogeneous internets became problematic (address mapping, for instance, was one of the worst problems to arise). This was historically handled in an *ad hoc* manner, but internets have grown to sizes that make programs such as **sendmail** necessary. It acts as a unified *post office* to which all mail can be submitted.

Not all users need to install **sendmail**; see "Who Needs the **sendmail** Facility," to determine if you should use it.

sendmail is not tied to any one transport protocol—its function can be likened to a crossbar switch, relaying messages from one domain into another. In the process, **sendmail** can perform a limited amount of message header editing to put the message into a format that is appropriate for the receiving domain.

All **sendmail** processing is performed under the control of a configuration file. Due to the flexibility requirements for **sendmail**, the configuration file can seem somewhat unapproachable. However, a standard configuration file is supplied, which should suit most sites; most other configurations can be built by incrementally adjusting an existing configuration file.

The section "sendmail Configuration File," describes the configuration file's syntax and semantics, and gives tips about writing your own **sendmail** configuration file.

Although **sendmail** is intended to run without the need for monitoring, it has a number of features that can be used to monitor or adjust the operation under unusual circumstances. The monitoring features are described later in this section.

The following sections describe

- Who needs the **sendmail** facility
- **sendmail** start-up
- Important **sendmail** files and directories
- Purpose of the **sendmail.cf** configuration file
- Some important arguments to **sendmail**
- Day-to-day maintenance of your mail system
- Recommendations for adjusting configuration parameters
- Use of **sendmail** command line arguments and options, configuration options, mailer flags, and other configuration parameters

Who Needs the sendmail Facility

Not everyone needs to use **sendmail**. If your system uses only UUCP as a mail transport mechanism, you might not want to install **sendmail**. However, if you have a heterogenous system, mixing mail transport protocols such as SMTP and UUCP, or if you have a very large number of sites on your network, you will want to install **sendmail**. **sendmail** also has features such as automatic Gateway routing and an extremely flexible configuration, which you may want on your system.

sendmail Startup

The **sendmail** program is part of the supplemental BSD-compatibility software package. Before invoking **sendmail** with the options appropriate for your site configuration, refer to the *BSD/XENIX Compatibility Guide* for important information about installing and configuring the BSD package.

When **sendmail** starts up, it reads the **/usr/ucblib/sendmail.cf** file (the **sendmail** configuration file) to determine the names of mailers, how to parse addresses, how to rewrite the message header, and the settings of various options.

The configuration file is described briefly in the next few paragraphs, along with other files that are important to the system administrator. The file is described in more detail later in this section.

Important sendmail Directories and Files

As **sendmail** administrator, you should get familiar with the following files and directories:

/usr/ucblib/sendmail	The binary of **sendmail**.
/usr/ucblib/sendmail.cf	The **sendmail** configuration file, in textual form.
/usr/ucblib/sendmail.fc	The configuration file represented as a memory image.
/usr/ucblib/sendmail.hf	The **sendmail** program help file.
/usr/ucblib/aliases	The textual version of the alias file.
/usr/ucblib/aliases.pag	The alias file in **dbm**(3) format.
/usr/ucblib/aliases.dir	The alias file in **dbm**(3) format.
/var/spool/smtpq	The directory in which the mail queue and temporary files reside.

sendmail.cf Configuration File

The **sendmail** configuration table has three major purposes:

- To set up the environment for **sendmail**.
- To rewrite addresses in the message.
- To map addresses into the actual set of instructions necessary to get the message delivered.

sendmail Arguments

The complete list of arguments to **sendmail** is described in detail later in this section. Some important arguments are described in the following paragraphs.

Queue Interval

The amount of time from when a process forks to when it exits the queue is defined by the **–q** argument. Running **sendmail** in mode **f** or **–q** argument defines the maximum amount of time that a message can sit in the queue.

Daemon Mode

If you allow incoming mail over an IPC connection, you should have a daemon running. Daemon mode is set by using the **–bd** arguments. The **–bd** arguments and the **–q** argument can be combined in one call:

```
/usr/ucblib/sendmail –bd –q30m
```

Forcing the Queue

Sometimes the queue gets clogged. You can force a queue run by using the **–q** argument (with no value) to **sendmail**. Use the **–v** (verbose) argument with the **–q** argument, as shown below, to watch what happens during the queue run:

```
/usr/ucblib/sendmail –q –v
```

Debugging

sendmail provides several debug flags. Each debug flag has a number and a level, where higher levels print out more information. Debug flags with levels greater than nine print out so much information that you would not normally want to see them except for debugging that particular piece of code. Debug flags are set by using the **–d** option, with the following format:

```
debug-flag:      -d debug-list
debug-list:      debug-option [ , debug-option ]
debug-option:    debug-range [ . debug-level ]
debug-range:     integer | integer – integer
debug-level:     integer
```

where spaces are shown for reading ease only. For example,

```
-d12        Set flag 12 to level 1
-d12.3      Set flag 12 to level 3
-d3-17      Set flags 3 through 17 to level 1
-d3-17.4    Set flags 3 through 17 to level 4
```

Trying a Different Configuration File

You can specify an alternative **sendmail** configuration file by using the **–C** argument. For example, the command:

```
/usr/ucblib/sendmail -Ctest.cf
```

uses the configuration file **test.cf** instead of the default **/usr/ucblib/sendmail.cf**. If the **–C** argument has no value, it defaults to **sendmail.cf** in the current directory.

Changing the Values of Options

You can override an option by using the **–o** argument. For example, the command

```
/usr/ucblib/sendmail -oT2m
```

sets the **T** (timeout) option to 2 minutes for this run only.

Normal sendmail Operations

The following paragraphs describe day-to-day **sendmail** processes.

Mail Queue

Normally, the mail queue is processed transparently. However, sometimes manual intervention is necessary. For example, if a major host is down for a period of time, the queue can get clogged. Although **sendmail** should recover gracefully when the host comes up, performance can get unacceptably bad in the meantime. When this happens, force the queue, as described later in this section, under "Forcing the Queue."

Printing the Queue

To print the queue, use the **sendmail** command with the **–bp** options.

The printed queue shows a listing of the queue IDs, the size of the message, the date the message entered the queue, and the sender and recipients.

Format of Queue Files

All queue files have the form:

xfAA99999

where *x* is the type (translated below) and *AA99999* is the file ID.

The types are translated as follows:

d The data file. The message body (excluding the header) is kept in this file.

l The lock file. If this file exists, the job is currently being processed, and a queue run does not process the file. For that reason, an extraneous **lf** file can cause a job to apparently disappear, without even timing out.

n This file is created when an ID is being created. It is a separate file to ensure that no mail can ever be destroyed due to a race condition. It should exist for no more than a few milliseconds at any given time.

q The queue control file. This file contains the information necessary to process the job.

t A temporary file. These files are images of the **qf** file when it is being rebuilt. It should be renamed to a **qf** file very quickly.

x A transcript file, existing during the life of a session, showing everything that happens during that session.

The **qf** file is structured as a series of lines each beginning with a code letter. The lines are as follows:

D The name of the data file. Only one of these lines can be present.

H A header definition. Any number of these lines may be present. The order of header definitions is important: they represent the order in the final message. These files use the same syntax as header definitions in the configuration file.

R A recipient address. This address is normally completely aliased, but is actually realiased when the job is processed. One line is present for each recipient.

S The sender address. Only one of these lines can be present.

E An error address. If any such lines exist, they represent the addresses that should receive error messages.

T The job creation time. This value is used to compute when to time out the job.

P The current message priority. This value is used to order the queue. Higher numbers specify lower priorities. The priority changes as the message sits in the queue. The initial priority depends on the message class and the size of the message.

Forcing the Queue

Normally, **sendmail** runs the queue automatically at intervals. The algorithm is to read and sort the queue, and then to attempt to process all jobs in order. When it attempts to run the job, **sendmail** first checks to see if the job is locked. If so, it ignores the job.

No attempt is made to ensure that only one queue processor exists at any time, since there is no guarantee that a job cannot take forever to process. Due to the locking algorithm, it is impossible for one job to freeze the queue. However, an uncooperative recipient host or a program recipient that never returns can accumulate many processes in your system. Unfortunately, there is no way to resolve this situation without violating the protocol.

In some cases, a major host goes down for a couple of days, creating a prohibitively large queue. As a result, **sendmail** may spend an inordinate amount of time sorting the queue. This situation can be fixed by moving the queue to a temporary place and creating a new queue. The old queue can be run later when the offending host returns to service.

First move the entire queue directory:

```
cd /var/spool
mv smtpq osmtpq; mkdir smtpq; chmod 777 smtpq
```

Then kill the existing daemon (since it continues processing in the old queue directory) and create a new daemon.

To run the old mail queue, use the following command:

```
/usr/ucblib/sendmail -oQ/var/spool/osmtpq -q
```

where:

-oQ Specifies an alternate queue directory.

-q Runs every job in the queue.

-v Gives detailed display of information.

When the queue is finally empty, you can remove the directory:

```
rmdir /var/spool/osmtpq
```

Alias Database

The alias database exists in two forms. One is a text form, maintained in the file **/usr/ucblib/aliases**. Alias entries are of the following form:

name: *name1*, *name2*, ...

Continue an entry by starting any continuation lines with a space or a tab. Blank lines and lines beginning with a pound sign (#) are comments.

The second form is processed by the **dbm**(3) library. This form is in the files **/usr/ucblib/aliases.dir** and **/usr/ucblib/aliases.pag**. **sendmail** uses this second form to resolve aliases to improve performance.

Rebuilding the Alias Database

You can explicitly rebuild the **dbm** version of the database by executing the following command:

```
/usr/ucblib/sendmail -bi
```

If the **D** option is specified in the configuration, **sendmail** rebuilds the alias database automatically, if possible, when it is out of date. **sendmail** rebuilds the database under the following conditions:

- The **dbm** version of the database is mode 666.
- **sendmail** is running **setuid** to root.

> **Caution**
>
> Auto-rebuild can be dangerous on heavily loaded machines with large alias files; if it takes more than 5 minutes to rebuild the database, there is a chance that several processes may start the rebuild process simultaneously.

Potential Problems in the Alias Database

The following problems can occur with the alias database as a result of **sendmail** accessing the **dbm** version of the alias database while it is only partially built. This can happen under the following circumstances:

- One process accesses the database while another process is rebuilding it.
- The process rebuilding the database dies (due to being killed or a system crash) before completing the rebuild.

sendmail uses two techniques to try to relieve the problems mentioned above. First, it ignores interrupts while rebuilding the database, avoiding the problem of someone aborting the process and leaving a partially rebuilt database. Second, at the end of the rebuild, it adds an alias of the form @: @ (which is not normally legal). Before **sendmail** accesses the database, it checks to ensure that this entry exists.

Note: *The* **a** *option is required in the configuration for this action to occur.*

sendmail waits for the @: @ entry to appear, at which point it forces a rebuild itself.

Note: *The* **D** *option must be specified in the configuration file for the rebuild to occur. If the* **D** *option is not specified, a warning message is generated and* **sendmail** *continues.*

List Owners

If an error occurs on sending to a certain address, for example *x*, **sendmail** looks for an alias of the form **owner-** *x* to receive the errors. This feature is typically useful for a mailing list where the submitter of the list has no control over the maintenance of the list itself; in this case, the list maintainer would be the owner of the list. For example:

```
unix-wizards: jc@ramona, jc@oxford, nosuchuser, jim@matisse
owner-unix-wizards: jc@ramona
```

would cause **jc@ramona** to get the error that occurs when someone sends to **unix-wizards** due to the inclusion of **nosuchuser** on the list.

Per-User Forwarding (.forward Files)

As an alternative to the alias database, any user can put a file with the name **.forward** in his or her home directory. If this file exists, **sendmail** redirects mail for that user to the list of addresses listed in the **.forward** file. For example, if the home directory for user **jcgravy** has a **.forward** file with contents:

```
jcgravy@kaiser
jpc@ramona
```

then any mail arriving for **jcgravy** is redirected to the specified accounts.

Special Header Lines

Several header lines have special interpretations defined by the **sendmail** configuration file. Others have interpretations, built into **sendmail**, which cannot be changed without changing the code. The following paragraphs describe the built-in header lines.

Return-Receipt-To:	If this header is sent, a message is sent to any specified addresses when the final delivery is complete, that is, when successfully delivered to a mailer with the l flag (local delivery) set in the mailer descriptor.

Errors-To:	If errors occur anywhere during processing, this header directs the error messages to listed addresses rather than to the sender. This function is intended for mailing lists.
Apparently-To:	If a message comes in with no recipients listed in the message (in a To:, Cc:, or Bcc: line) sendmail adds an Apparently-To: header line for any recipients it is aware of. At least one recipient line is required under the ARPANET protocol RFC822.

Tuning Configuration Parameters

You may want to change the value of one or more configuration parameters, depending on the requirements of your site. Most parameters are set by an option in the **sendmail.cf** configuration file. For example, the line **OT3d** sets option **T** to the value **3d** (three days).

Most parameter defaults are set appropriately for most sites. Sites with high mail loads might need to tune some parameter values as appropriate for their mail load. In particular, sites experiencing a lot of small messages, many of which are delivered to many recipients, may need to adjust the parameters that determine queue priorities.

The following paragraphs describe the configurable parameters.

Timeout Parameters

All time intervals are set using a scaled syntax. For example, **10m** represents ten minutes, whereas **2h30m** represents two and a half hours. The full set of scales is as follows:

- **s** seconds
- **m** minutes
- **h** hours
- **d** days
- **w** weeks

Queue Interval

The argument to the **-q** flag specifies how often a subdaemon runs the queue. This interval is typically set to between 15 minutes and one hour.

Read Timeouts

It is possible to time out when reading the standard input or when reading from a remote **sendmail** server. Technically, this timeout is not acceptable within the published protocols. However, it might be appropriate to set a large timeout (such as an hour) in certain environments. A large timeout reduces the chance of large numbers of idle daemons piling up on your system.

This timeout is set using the **r** option in the configuration file.

Message Timeouts

After sitting in the queue for a few days, a message times out, ensuring that at least the sender is aware that the message could not be sent. The timeout is typically set to three days. This timeout is set using the **T** option in the configuration file.

The time of submission is set in the queue, rather than the amount of time left until timeout. As a result, you can flush messages that have been hanging for a short period by running the queue with a short message timeout. For example, the command

```
/usr/ucblib/sendmail -oT1d -q
```

runs the queue and flushes anything that is one day old.

Forking During Queue Runs

When you use the **Y** option, **sendmail** forks before each individual message while running the queue, preventing **sendmail** from consuming large amounts of memory. This option is especially useful in memory-poor environments.

Note that if the **Y** option is not set, **sendmail** keeps track of hosts that are down during a queue run.

Queue Priorities

Every message is assigned a priority, consisting of the message size (in bytes) offset by the sum of the message class times the *work class factor* and the number of recipients times the *work recipient factor*.

The queue is ordered according to the priority and the creation time of the message (in seconds, since January 1 1970). Higher numbers for the priority specify that the message is processed later when running the queue.

The message size is also included so that large messages are penalized relative to small messages. The message class allows users to send *high priority* messages by including a *Precedence:* field in their message; the value of this field is looked up in the **P** lines of the configuration file. Since the number of recipients affects the amount of load a message presents to the system, this factor is also included into the priority.

The recipient and class factors can be set in the configuration file by using the **y** and **z** options, respectively. The **y** and **z** options default to 1000 (for the recipient factor) and 1800 (for the class factor). The default priority is set as follows:

$$pri = size - (class * z) + (nrcpt * y)$$

Note: *Higher values for this parameter actually mean that the job is treated with lower priority.*

The priority of a job can also be adjusted each time it is processed (that is, each time an attempt is made to deliver it) using the *work time factor,* set by the **Z** option. This factor is added to the priority, so it normally decreases the precedence of the job, on the grounds that jobs that have failed many times tend to fail again in the future.

Load Limiting

You can use the **x** option to instruct **sendmail** to queue, but not deliver mail if the system load average gets too high.

When the load average exceeds the value of the **x** option, the delivery mode is set to **q** (queue only) if the *queue factor* (**q** option) divided by the difference in the current load average and the **x** option plus one exceeds the priority of the message; that is, the message is queued if

$$pri > \frac{QF}{LA - X + 1}$$

The **q** option defaults to 10000, so each point of load average is worth 10000 priority points (as described above, that is, bytes + seconds + offsets).

For drastic cases, the **x** option defines a load average at which **sendmail** refuses to accept network connections. Locally generated mail (including incoming **uucp** mail) is still accepted.

Delivery Mode

The **d** configuration option determines the delivery mode in which **sendmail** operates. Delivery modes specify how quickly mail is delivered.

Legal modes are

- **i** Deliver interactively (synchronously)
- **b** Deliver in background (asynchronously)
- **q** Queue only (do not deliver)

The tradeoffs for each mode are as follows:

Mode i.	Passes the maximum amount of information to the sender, but is hardly ever necessary
Mode q	Puts the minimum load on your machine, but means that delivery may be delayed for up to the queue interval.
Mode b	Is a good compromise between modes **i** and **q**. However, mode **b** can cause large numbers of processes if you have a mailer that takes a long time to deliver a message.

Log Level

The level of logging can be set for **sendmail**. Using a standard configuration table, the default level is 9.

The levels are as follows:

0 No logging

1 Major problems only

2 Message collections and failed deliveries

3 Successful deliveries

4 Messages being deferred (due to a host being down, and so forth)

5 Normal message queue-ups

6 Unusual but benign incidents, for example, trying to process a locked queue file

9 Log internal queue ID to external message ID mappings; this information is useful for tracing a message as it travels between several hosts

12 Several messages that are basically only of interest when debugging

16 Verbose information regarding the queue

File Modes

A number of files can have a number of modes. The modes depend on what you want to do with the file, and on the level of security you require.

setuid for sendmail

sendmail can safely be made setuid to **root**. At the point where it is about to **exec**(2) a mailer, **sendmail** checks to see if the userid is zero; if so, it resets the userid and groupid to a default (set by the **u** and **g** options). However, this causes mail processing to be accounted [using **sa**(7)] to **root** rather than to the user sending the mail.

Note: *This resetting can be overridden by setting the* **S** *flag to the mailer for mailers that are trusted and must be called as root.*

Temporary File Modes

The **F** option specifies the mode of all temporary files that **sendmail** creates. Reasonable values for the **F** option are 0600 and 0644. If the more permissive mode is selected, it is not necessary to run **sendmail** as **root** at all (even when running the queue).

Protecting the Alias Database

Some sites prefer to set the alias database (**/usr/ucblib/aliases**) permission to mode 666. Some dangers are inherent in this approach; any user can add himself or herself to any list, or can steal any other user's mail. However, in a relaxed-security environment, the cost of having a read-only database can be greater than the expense of finding and eradicating such security breaches.

The database that **sendmail** actually uses is represented by the two files **aliases.dir** and **aliases.pag** (both in **/usr/ucblib**). The mode on these files should match the mode on **/usr/ucblib/aliases**. If **aliases** is writable and the **dbm** files (**aliases.dir** and **aliases.pag**) are not, users cannot reflect their desired changes through to the actual database. However, if **aliases** is read-only and the **dbm** files are writable, a slightly sophisticated user can arrange to steal mail anyway.

If your **dbm** files are not writable by the world or you do not have auto-rebuild enabled (with the **D** option), you must be careful to reconstruct the alias database each time you change the text version. To reconstruct the database, use the **newaliases** command, or the following form of the **sendmail** command:

```
/usr/ucblib/sendmail -bi
```

If auto-rebuild is not enabled, and the **newaliases** command is not executed, any intended changes to the alias database are ignored.

sendmail Command Line Flags

sendmail arguments must be presented with flags before addresses.

The flags are as follows:

–f *addr*	The sender's machine address is *addr*. This flag is ignored unless the real user is listed as a *trusted user* or if *addr* contains an exclamation point (because of **uucp** restrictions).
–r *addr*	An obsolete form of **–f**.
–h *cnt*	Sets the *hop count* to *cnt*. This count represents the number of times this message has been processed by **sendmail** (to the extent that it is supported by the underlying networks). *cnt* is incremented during processing, and if it reaches MAXHOP (currently 30), **sendmail** throws away the message with an error.
-F*name*	Sets the full name of this user to *name*.
–n	Doesn't do aliasing or forwarding.
–t	Reads the header for **To:**, **Cc:**, and **Bcc:** lines, and sends to everyone listed in those lists. The **Bcc:** line is deleted before sending. Any addresses in the argument vector are deleted from the send list.

-b*x* Sets operation mode to *x*. Operation modes are as follows:

- **m** Deliver mail (default)
- **a** Run in ARPANET mode
- **s** Speak SMTP on input side
- **d** Run as a daemon
- **t** Run in test mode
- **v** Just verify addresses, do not collect or deliver
- **i** Initialize the alias database
- **p** Print the mail queue
- **z** Freeze the configuration file

The special processing for the ARPANET includes reading the **From:** line from the header to find the sender, printing ARPANET style messages (preceded by three digit reply codes for compatibility with the FTP protocol), and ending lines of error messages with <CRLF>.

-q*tim* Tries to process the queued mail. If the time is given, a **sendmail** runs through the queue at the specified interval to deliver queued mail; otherwise, it only runs once.

-C*file* Uses a different configuration file. **sendmail** runs as the invoking user (rather than as **root**).

-d*level* Sets debugging level.

o*xvalue* Sets option *x* to the specified *value*. These options are described later in this section.

The **f** option may be specified as the **-s** flag.

Configuration Options

The following options may be set using the **–o** flag on the command line or the**O** line in the configuration file. Many should be specified only if the invoking user is trusted.

A*file* — Use the named *file* as the alias file. If no file is specified, use **aliases** in the current directory.

a*N* — If set, wait up to *N* minutes for an @: @ entry to exist in the alias database before starting up. If it does not appear in *N* minutes, rebuild the database (if the **D** option is also set), or issue a warning.

B*c* — Set the blank substitution character to *c*. Unquoted spaces in addresses are replaced by this character.

c — If an outgoing mailer is marked as being expensive, do not connect immediately. This option requires that queueing be compiled in, since it depends on a queue run process to actually send the mail.

d*x* — Deliver in mode *x*. Legal modes are

- i Deliver interactively (synchronously)
- b Deliver in background (asynchronously)
- q Just queue the message (deliver during queue run)

D — If set, rebuild the alias database if necessary and possible. If this option is not set, **sendmail** never rebuilds the alias database unless explicitly requested using **–bi**.

e*x* — Dispose of errors using mode *x*. The values for *x* are

- p Print error messages (default)
- q No messages, just give exit status
- m Mail back errors
- w Write back errors (mail if user not logged in)
- e Mail back errors and give zero exit status always

F*n* — The temporary file mode, in octal. 644 and 600 are good choices.

f	Save UNIX-style **From:** lines at the front of headers. Normally they are assumed redundant and discarded.
g*n*	Set the default group ID for mailers to run in to *n*.
H*file*	Specify the help file for **sendmail** program.
i	Ignore dots in incoming messages.
L*n*	Set the default log level to *n*.
M*xvalue*	Set the macro *x* to *value*. This option is intended only for use from the command line.
m	Send to me too, even if I am in an alias expansion.
N*netname*	The name of the home network; **ARPA** by default. The argument of a **HELO** command is checked against *hostname.netname* where *hostname* is requested from the kernel for the current connection. If they do not match, **Received:** lines are augmented by the name that is determined in this manner so that messages can be traced accurately.
o	Assume that the headers may be in old format, that is to say, spaces delimit names. This option actually turns on an adaptive algorithm: If any recipient address contains a comma, parenthesis, or angle bracket, it is assumed that commas already exist. If this flag is not on, only commas delimit names. Headers are always output with commas between the names.
Q*dir*	Use the named *dir* as the queue directory.
q*factor*	Use *factor* as the multiplier in the map function to decide when to just queue up jobs rather than run them. This value is divided by the difference between the current load average and the load average limit (**x** flag) to determine the maximum message priority to send. Defaults to 10000.
r*time*	Timeout reads after *time* interval.
S*file*.	Log statistics in the named *file*.

s	Be safe when running things, that is to say, always run the queue file, even if you are going to attempt immediate delivery. **sendmail** always runs the queue file before returning control to the client under any circumstances.
T*time*	Set the queue timeout to *time*. After this interval, messages that have not been successfully sent are returned to the sender.
t*S,D*	Set the local time zone name to *S* for standard time and *D* for daylight time; this setting is only used under version six.
u*n*	Set the default userid for mailers to *n*. Mailers without the *S* flag in the mailer definition run as this user.
v	Run in verbose mode.
x*LA*	When the system load average exceeds *LA*, just queue messages (that is to say, do not try to send them). When the system load average exceeds *LA*, refuse incoming **sendmail** connections.
y*fact*	The indicated *fact*or is added to the priority (*lowering* the priority of the job) for each recipient, that is to say, this value penalizes jobs with large numbers of recipients.
Y	If set, deliver each job that is run from the queue in a separate process. Use this option if you are short of memory, since the default tends to consume considerable amounts of memory while the queue is being processed.
z*fact*	The indicated *fact*or is multiplied by the message class (determined by the *Precedence:* field in the user header and the **P** lines in the configuration file) and subtracted from the priority. Thus, messages with a higher *Priority:* are favored
Z*fact*	The *fact*or is added to the priority every time a job is processed. Thus, each time a job is processed, its priority is decreased by the indicated value. In most environments, this value should be positive, since hosts that are down are all too often down for a long time.

Mailer Flags

The following flags may be set in the mailer description.

f The mailer wants a **–f** *from* flag, but only if this operation is a network forward. (That is to say, the mailer gives an error if the executing user does not have special permissions.)

r Same as **f**, but sends a **–r** flag.

S The userid is not reset before calling the mailer. This option would be used in a secure environment where **sendmail** ran as **root** and could be used to avoid forged addresses. This flag is suppressed if given from an unsafe environment (for example, a user's **mail.cf** file).

n A UNIX-style **From:** line is not inserted on the front of the message.

l This mailer is local: that is, final delivery is performed.

s The quote characters are stripped off the address before calling the mailer.

m The mailer can send to multiple users on the same host in one transaction. When a **$u** macro occurs in the *argv* part of the mailer definition, that field is repeated as necessary for all qualifying users.

F This mailer wants a **From:** header line.

D This mailer wants a **Date:** header line.

M This mailer wants a **Message-Id:** header line.

x This mailer wants a **Full-Name:** header line.

P This mailer wants a **Return-Path:** line.

u Uppercase should be preserved in user names for this mailer.

h Uppercase should be preserved in host names for this mailer.

A ARPANET-compatible mailer, and all appropriate modes should be set.

U This mailer wants UNIX-style **From:** lines with the **uucp**-style *remote from <host>* on the end.

e This mailer should not be connected normally, as this mailer is expensive to connect to; any necessary connection occurs during a queue run.

X This mailer has a hidden dot algorithm as specified in RFC821; any line beginning with a dot has an extra dot prepended (to be stripped at the other end). This extra dot ensures that lines in the message do not terminate the message prematurely.

L This mailer limits the line lengths as specified in RFC821.

P This mailer uses the return-path in the **sendmail** program **MAIL FROM:** command rather than just the return address; although this path is required in RFC821, many hosts do not process return paths properly.

I This mailer speaks SMTP to another **sendmail**—as such it can use special protocol features. **-I** is optional.

C If mail is *received* from a mailer with this flag set, any addresses in the header that do not have an at sign (@) after being rewritten by ruleset three, have the **@domain** clause from the sender tacked on. This option allows mail with headers of the form:

```
From: usera@hosta
To: userb@hostb, userc
```

to be rewritten automatically as:

```
From: usera@hosta
To: userb@hostb, userc@hosta.
```

E Escape lines beginning with **From:** in the message with a ">" sign.

Monitoring the Internetwork

The following paragraphs describe interface programs and commands that check communications and monitor status on the network. For details about the use or format of a specific command listed below, refer to the reference manual entry for that command.

loopback The **loopback** interface lets you use the network protocols to communicate with your own host. Use loopback to check the networking software without making any assumptions about the state of network media and other systems.

An example of the use of the loopback interface is the command:

```
rlogin loopback
```

Note: *The loopback interface is assigned the Internet address 127.0.0.1, and the Internet name* **loopback**.

ping(1M) Tests a host's ability to send data to another host (or to your local host) and receive a response. To use **ping** to check your local host, specify your local host name or the **loopback** name as the host name.

ruptime(1) Displays the status of hosts on the local network. Information includes the length of time the host has been up, the number of active users logged in, and the system load.

rwho(1) Displays the status of users on the local network.

netstat(1) Displays a variety of network status information by symbolically displaying the contents of various network data structures. The **-r** option to **netstat** is described earlier in this section, under "Routing Table Status."

sendmail Configuration

This section describes the **sendmail** configuration file in detail, and includes hints on how to write one of your own. The syntax of the configuration file is designed to be reasonably easy to parse, since parsing is done every time **sendmail** starts up. An overview of the configuration file is given first, followed by details of the semantics.

Configuration File Line Syntax

The configuration file is organized as a series of lines, each beginning with a single character defining the semantics for the rest of the line. Lines beginning with a space or a tab are continuation lines (although the semantics are not well defined in many places). Blank lines and lines beginning with a sharp symbol (#) are comments.

R and S — Rewriting Rules

The core of address parsing is the set of rewriting rules. These rules are an ordered production system. **sendmail** scans through the rewriting rules looking for a match on the left-hand side (LHS) of the rule. When a rule matches, the address is replaced by the right-hand side (RHS) of the rule.

Some rewriting rules are used internally and must have specific semantics. Other rewriting rules do not have specifically assigned semantics, and can be referenced by the mailer definitions or by other rewriting sets.

Format for the **R** and **S** lines is shown below:

S*n*	Sets the current ruleset being collected to *n*. If you begin a ruleset more than once, it deletes the old definition.
R*lhs rhs comments*	The fields must be separated by at least one tab character; there may be embedded spaces in the fields. The *lhs* pattern is applied to the input. If it matches, the input is rewritten to the *rhs*. The *comments* are ignored.

D—Define Macro

Macros are named with a single character, selected from the entire ASCII set. Note that user-defined macro names should be selected from the set of uppercase letters, as lowercase letters and special symbols are used internally.

Format for macro definition is as follows:

```
Dx val
```

where *x* is the name of the macro and *val* is the value it should have. Macros are expressed in most places by using $ *x*.

C and F—Define Classes

Classes of words can be defined to match on the left-hand side of rewriting rules. For example, a class of all local names for this site might be created so that attempts to send to oneself can be eliminated. These classes can either be defined directly in the configuration file or read in from another file. Classes use names from the set of uppercase letters. Lowercase letters and special characters are reserved for system use. Format for **C** and **F** lines is shown below:

```
C word1 word2 ...
Fc file
```

The first form defines the class *c* to match any of the named words. You can put the names on different lines; for example, the two forms:

```
CHmonet ucbmonet
```

and

```
CHmonet
CHucbmonet
```

are equivalent.

The second form reads the elements of the class *c* from the named *file*.

M—Define Mailer

The **M** line defines programs and interfaces to mailers. The format of the **M** line is as follows:

```
Mname, { field = value }*
```

where *name* is the name of the mailer (used internally only) and the *field=name* pairs define attributes of the mailer. Fields are translated as follows:

Path	The pathname of the mailer
Flags	Special flags for this mailer
Sender	A rewriting set for sender addresses
Recipient	A rewriting set for recipient addresses
Argv	An argument vector to pass to this mailer
Eol	The end-of-line string for this mailer
Maxsize	The maximum message length to this mailer

Only the first character of the field name is checked.

H—Define Header

The **H** line defines the format of the header lines that **sendmail** inserts into the message.

The format of the **H** line is as follows:

```
H [ ? mflags ? ] .hnamea : htemplate
```

Continuation lines are reflected directly into the outgoing message. The *htemplate* is macro expanded before insertion into the message. If the *mflags* (surrounded by question marks) are specified, at least one of the specified flags must be stated in the mailer definition for this header to be automatically output. If one of these headers is in the input, it is reflected to the output regardless of these flags.

Some headers have special semantics, which are described in the following paragraphs.

O—Set Option

The **O** line specifies options to set. Options are represented by single characters. The syntax of the **O** line is as follows:

```
O o value
```

This line sets option *o* to be *value*. Depending on the option, *value* may be a string, an integer, a Boolean (with legal values **t**, **T**, **f**, or **F**; the default is TRUE), or a time interval.

T—Define Trusted Users

Trusted users are those users that can override the sender address by using the **–f** flag. Trusted users are typically **root**, **uucp**, and **network**, but it can be convenient to extend this list to include other users, perhaps to support a separate **uucp** login for each host.

The syntax of the **T** line is as follows:

```
Tuser1 user2 ...
```

More than one of these lines may be present.

P—Precedence Definitions

The **P** control line defines values for the *Precedence:* field. The format of the **P** line is as follows:

```
Pname = num
```

When the *name* is found in a *Precedence:* field, the message class is set to *num*. Higher numbers specify higher precedence. Numbers less than zero have the special property that error messages are not returned. The default precedence is zero.

Configuration File Semantics

The following sections describe the semantics of the configuration file.

Special Macros and Conditionals

Macros are expressed in the construct $*x*, where *x* is the name of the macro. In particular, lowercase letters are reserved to have special semantics, used to pass information into or out of **sendmail**, and some special characters are reserved to provide conditionals, and so forth. Conditionals can be specified using the following format:

```
$?x text1 $| text2 $.
```

The conditional shown above interpolates *text1* if the macro **$x** is set, and *text2* otherwise. The **else** (**.b**) clause is optional. The following macros *must* be defined to transmit information into **sendmail**:.

e The entry message

j The *official* domain name for this site

l The format of the UNIX -style **From:** line

n The name of the daemon (for error messages)

o The set of *operators* in addresses

q Default format of sender address

The **$e** macro is printed when **sendmail** starts up. The **$j** macro must be the first word. The **$j** macro should be in RFC821 format. The **$l** and **$n** macros can be considered constants except under very unusual circumstances. The **$o** macro consists of a list of characters, considered to be tokens, which separate tokens when doing parsing. For example, if @ is in the **$o** macro, the input **a@b** is scanned as three tokens: **a**, @, and **b**. Finally, the **$q** macro specifies how a default address should appear in a message.

The following macros are defined by **sendmail**:

a Origination date in ARPNET format

b Current date in ARPNET format

c Hop count

d Date in UNIX (ctime) format

f Sender (from) address

g Sender address relative to the recipient

h Recipient host

i Queue ID

p sendmail's pid

r Protocol used

s Sender's host name

t Numeric representation of the current time

u Recipient user

v Version number of sendmail

w Hostname of this site

x Full name of the sender

z Home directory of the recipient

Three types of dates can be used. The $**a** and $**b** macros are in ARPANET format; $**a** is the time as extracted from the **Date:** line of the message (if there was one), and $**b** is the current date and time (used for postmarks). If no **Date:** line is found in the incoming message, $**a** is set to the current time also. The $**d** macro is equivalent to the $**a** macro in UNIX-style (ctime) format.

The $**f** macro is the ID of the sender as originally determined; when mailing to a specific host, the $**g** macro is set to the address of the sender *relative to the recipient.* For example, if I send to **jim@matisse** from the machine **ramona**, the $**f** macro is **jim** and the $**g** macro is **jim@ramona**.

The $**x** macro is set to the full name of the sender. This value can be determined in several ways:

- It can be passed as a flag to **sendmail**.
- It can be passed as the value of the **Full-name:** line in the header if it exists.
- It can be passed as a value in the comment field of a **From:** line.

If the full name of the sender cannot be found in one of the above ways, and if the message was originated locally, the full name is looked up in the **/etc/passwd** file.

When sending, the $**h**, $**u**, and $**z** macros get set to the host, user, and home directory (if local) of the recipient. The first two are set from the $@ and $: part of the rewriting rules, respectively.

The $**p** and $**t** macros are used to create unique strings (for example, for the *Message-Id:* field). The $**i** macro is set to the queue ID on this host; if put into the timestamp line, it can be extremely useful for tracking messages. The $**v** macro is set to be the version number of **sendmail**; this number is normally put in timestamps and has been proven extremely useful for debugging. The $**w** macro is set to the name of this host if it can be determined. The $**c** field is set to the *hop count,* that is to say, the number of times this message has been processed. This number can be determined by the **–h** flag on the command line or by counting the timestamps in the message.

The $**r** and $**s** fields are set to the protocol used to communicate with **sendmail** and the sending host name; these fields are not supported in the current version.

Special Classes

The class **$=w** is set to be the set of all names by which this host is known. This class can be used to delete local host names.

Left-Hand Side

The LHS of rewriting rules contains a pattern. Normal words are simply matched directly. Metasyntax is introduced using a dollar sign. The metasymbols are

$*	Match zero or more tokens
$+	Match one or more tokens
$-	Match exactly one token
$=*x*	Match any token in class *x*
$~*x*	Match any token not in class *x*

If any metasymbols match, they are assigned to the symbol $*n* for replacement on the RHS, where *n* is the index in the LHS. For example, if the LHS:

```
$-:$+
```

is applied to the input:

```
BANANA:jim
```

the rule matches, and the values passed to the RHS are

```
$1  BANANA
$2  jim
```

Right-Hand Side

When the LHS of a rewriting rule matches, the input is deleted and replaced by the RHS. Tokens are copied directly from the RHS unless they begin with a dollar sign. Metasymbols are

$*n*	Substitute indefinite token *n* from LHS
$[*name*$]	Canonicalize *name*
$>*n*	*Call* ruleset *n*
$#*mailer*	Resolve to *mailer*
$@*host*	Specify *host*
$:*user*	Specify *user*

The $*n* syntax substitutes the corresponding value from a $+, $–, $*, $=, or $~ match on the LHS. It can appear anywhere.

A host name enclosed between $[and $] is replaced by the canonical name. For example, $[**csam**$] becomes **lbl-csam.arpa**.

The $>*n* syntax causes the remainder of the line to be substituted as usual and then passed as the argument to ruleset *n*. The final value of ruleset *n* then becomes the substitution for this rule.

The $# syntax should be used *only* in ruleset zero. It causes evaluation of the ruleset to terminate immediately, and signals to **sendmail** that the address is resolved. The complete syntax is

$#*mailer*$@*host*$:*user*

This syntax specifies the mailer, host, and user information necessary to direct the mailer. If the mailer is local, the *host* portion is optional. *Mailer* and *host* must be single words, but *user* can consist of multiple words.

An RHS can also be preceded by a $@ or a $: to control evaluation. A $@ prefix causes the ruleset to return with the remainder of the RHS as the value. A $: prefix causes the rule to terminate immediately, as the ruleset continues; this prevents continued application of a rule. The prefix is stripped before continuing.

The $@ and $: prefixes can precede a $> format; for example,

```
R$+$:$>7$1
```

matches anything, passes that match to ruleset seven, and continues; the $: is necessary to avoid an infinite loop. Substitution occurs in the order described: that is, parameters from the LHS are substituted, host names are converted into canonical form, **subroutines** are called, and finally $#, $@, and $: are processed.

Semantics of Rewriting Rule Sets

The five rewriting sets have specific semantics. These semantics are related as depicted by Figure 9–1.

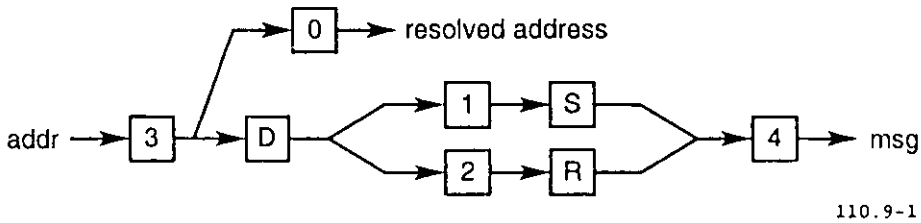

Figure 9–1. Rewriting Set Semantics

In Figure 9–1, **D** is the sender domain addition, **S** is the mailer-specific sender rewriting, and **R** is the mailer-specific recipient rewriting.

Rule set three should turn the address into *canonical form*. This form should have the following basic syntax:

```
local-part@host-domain-spec
```

If no @ sign is specified, the host-domain-spec *may* be appended from the sender address (if the **C** flag is set in the mailer definition corresponding to the *sending* mailer). **sendmail** applies ruleset three before doing anything with any address.

After ruleset three, ruleset zero is applied to addresses that actually specify recipients. It must resolve to a (*mailer, host, user*) triple. The *mailer* must be defined in the mailer definitions from the configuration file. The *host* is defined into the **$h** macro for use in the argv expansion of the specified mailer.

Rulesets one and two are applied to all sender and recipient addresses, respectively. They are applied before any specification in the mailer definition. They must never resolve.

Ruleset four is applied to all addresses in the message. It is typically used to translate internal to external form.

Mailer Flags

A number of flags can be associated with each mailer, each identified by a letter. Many are assigned semantics internally. These flags are detailed later in this section. Any other flags can conditionally assign headers to messages destined for particular mailers.

error Mailer

The mailer with the special name **error** can be used to generate a user error. The (optional) host field is a numeric exit status to be returned, and the user field is a message to be printed. For example, the entry:

```
$#error$:Host unknown in this domain
```

on the RHS of a rule causes the specified error to be generated if the LHS matches. This mailer is functional only in ruleset zero.

Building a Configuration File from Scratch

Building a configuration table from scratch is an extremely difficult job. Fortunately, it is almost never necessary to do so; nearly every situation that may come up can be resolved by changing an existing table. If you must write your own configuration file for **sendmail**, the following paragraphs offer some suggestions.

Relevant Issues

The canonical form you use should almost certainly be as specified in the ARPANET protocols RFC819 and RFC822. RFC822 describes the format of the mail message itself. **sendmail** follows this RFC closely, to the extent that many of the standards described therein cannot be changed without changing the code. In particular, the following characters have special interpretations:

```
< > ( ) " \
```

Attempting to use these characters for any use other than their RFC822 purpose in addresses is likely to fail.

RFC819 describes the specifics of the domain-based addressing. This addressing is touched on in RFC822 as well.

Beware of errors in RFC819.

How to Proceed

Before you begin, it is a good idea to examine existing configuration tables to use as models.

Once you have a model, start by building ruleset three. Beware of changing the address in this ruleset. In particular, stripping local domains is best deferred, since stripping can leave addresses with no domain spec at all. **sendmail** appends the sending domain to addresses with no domain; stripping domains can change the semantics of addresses. Also try to avoid fully qualifying domains in this ruleset.

Testing the Rewriting Rules—the -bt Flag

When you build a configuration table, you can use the *test mode* of **sendmail** to check the configuration. For example, the command:

```
sendmail -bt -Ctest.cf
```

reads the configuration file **test.cf** and enters test mode. In test mode, you enter lines of the form:

```
rwset address
```

where *rwset* is the rewriting set you want to use and *address* is an address to apply the set to. Test mode shows the steps it takes as it proceeds, finally showing the address it ends with. Use a comma-separated list of *rwset*s for sequential application of rules to an input; ruleset three is always applied first. For example:

```
1,21,4 monet:patrick
```

first applies ruleset three to the input **monet:patrick**. Ruleset one is then applied to the output of ruleset three, followed similarly by rulesets 21 and four. For more detail, use the **-d21** flag to turn on more debugging. For example,

```
sendmail  -bt  -d21.99
```

displays a lot of information: a single word address can print out pages of information.

Building Mailer Descriptions

To add an outgoing mailer to your mail system, you must define the characteristics of the mailer. Each mailer must have an internal name. This name can be arbitrary, except that the names **local** and **prog** must be defined.

The pathname of the mailer must be given in the **P** field. If this mailer should be accessed through an IPC connection, use the string **[IPC]** instead.

The **F** field defines the mailer flags. You should specify an **f** or **r** flag to pass the name of the sender as a **–f** or **–r** flag, respectively. These flags are passed only if they were passed to *sendmail*, to placate sensitive mailers that give errors under some circumstances. If the mailer is not extremely sensitive, you can specify **–f $g** in the argv template. If the mailer must be called as **root**, specify the **S** flag; this flag does not reset the userid before calling the mailer.

Note: *If you use the* **S** *option,* **sendmail** *must be running setuid to* **root**.

If this mailer is local (if it performs final delivery rather than performing another network hop), give the **l** flag. Quote characters, backslashes (\), and quotation marks (" ") can be stripped from addresses if the **s** flag is specified; if this flag is not given, they are passed through. If the mailer is capable of sending to more than one user on the same host in a single transaction, use the **m** flag. If this flag is on, the argv template containing **$u** is repeated for each unique user on a given host. The **e** flag marks the mailer as being expensive, which causes **sendmail** to defer connection until a queue can be run.

Note: *The* **c** *configuration option must be given for the* **e** *flag to be effective.*

The **C** flag is unusual, as it applies to the mailer from which the message is received, rather than to the mailer to which the message is sent. If the **C** flag is set, the domain portion of the sender address (that is to say, the **@host.domain** part) is saved and appended to any addresses in the message that do not already contain a domain portion. For example, a message of the form:

```
From: jim@ramona,
To: jmc@oxford, patrick
```

is modified to

```
From: jim@ramona
To: jmc@oxford, patrick@ramona
```

if and only if the **C** flag is defined in the mailer corresponding to **jim@ramona**. Other flags are described in "TCP/IP Administration," under "Sendmail Command Line Flags."

The **S** and **R** fields in the mailer description are per-mailer rewriting sets to be applied to sender and recipient addresses, respectively. These fields are applied after the sending domain is appended and the general rewriting sets (numbers one and two) are applied, but before the output rewrite (ruleset four) is applied. A typical use is to append the current domain to addresses that do not already have a domain.

For example, a header of the form:

```
From: jim
```

might be changed to

```
From: jim@ramona
```

or

```
From: ramonsrvr!eric
```

depending on the domain it is being shipped into. The **E** field defines the string to use as an end-of-line indication. The default is a string containing a new-line only. The usual backslash escapes (**\r, \n, \f, \b**) can also be used.

Finally, an argv template is given as the **E** field. It can have embedded spaces. If there is no argv with a **$u** macro in it, **sendmail** speaks to the mailer. If the pathname for this mailer is **[IPC]**, the argv should be

```
IPC $h [ port ]
```

where *port* is the optional port number.

About This Part

Part 2— Distributed File System Administration

UNIX System V Release 4.0 supports the NFS software included in the UNIX System V Release 4.0.

The sections in this part describe the tasks you can perform using Distributed File System (DFS) administration—a set of commands and files that allows you to administer NFS and other distributed file system types that may be available in the future. The files and commands that support common administration of distributed file systems are provided in the DFS Administration Utilities package, which is a part of the System V Release 4.0 software release package.

Note: *The sections in Part 2 are not intended to be a complete reference fc administering NFS or another distributed file system. Although there are several tasks common to the administration of such packages, there are many differences as well. Refer to "Part 3, Network File System Administration" for a discussion of NFS administration, or the appropriate distributed file system administrator's guide.*

Audience

"Part 2, DFS Administration" is intended only for administrators who are administering more than one distributed file system package simultaneously. If you are running only one distributed file system package, all the information you need to administer that package is in the package-specific administration documentation. The information in this part is directed to people who are experienced administrators of stand-alone systems. It is assumed that they understand such basic concepts as the following:

- Changing run levels (init states)
- Assigning user ID (uid) numbers
- Mounting and unmounting local resources
- Assigning access rights to local resources

It is assumed, too, that they are familiar with the basic concepts of distributed file systems including

- Server/client model
- Domains

and that they know enough about the benefits of each package to make the decision to install.

For information about basic system administration, see the *UNIX System V Release 4.0 System Administrator's Guide*.

Organization

This part is organized by the tasks you can perform using DFS Administration commands and files.

In addition to the command line interface, a menu interface is provided through which you can enter DFS commands. The menus are included with System Administration Menus (**sysadm**), a menu-based administration interface that is standard with System V Release 4.0. Once you access a **sysadm** menu, help screens provide you with all the information you need to complete a task. Therefore, the DFS menu section does not lead you step-by-step through the menu-based procedures. It provides an overview of the features provided by the DFS menus and directs you to the *System Administrator's Guide* for instruction in using the menu interface. The DFS commands that you enter at the command line are described here extensively.

The organization of "Part 2, DFS Administration" is as follows:

Section 10. Introduction to DFS Administration

Presents an overview of DFS administration, and describes all the commands and files that the DFS Administration Utilities package either installs or utilizes.

Section 11. Setting Up DFS Administration

Describes the software that must be installed before you can use DFS administration and directs you to installation instructions.

Section 12. DFS Management Menus

Describes the menu interface to DFS administration.

Section 13. Using DFS Administration Commands and Files

Tells you how to share and unshare resources using DFS commands and files; how to mount and unmount remote resources; and to display information about resources that are shared and mounted on your local system and on network clients.

Basic Terminology

Terminology is used that should be known to administrators familiar with basic system administration and basic concepts, including:

client	A system that uses the resources of another system; a system can be both a client, utilizing resources that reside on other systems, and a server, making local resources available to other systems.
file system type	An implementation of a file system to support a particular file type. (The characteristics of a file are determined by its file type.) NFS is implemented as a file system that supports files of a type whose characteristics make them suitable for sharing across a network.
mount point	A location within a directory tree through which your computer accesses mounted file systems. You need to create and/or specify a mount point when you mount a remote resource. Any empty directory can serve as a mount point. (A resource can be mounted on a directory that is not empty, but the mount then obscures the directory's contents.)

resource	An object, such as a directory or a file system, which is made available to users of a computer system. An NFS resource can be a whole or partial file system and includes all the ordinary files and directories below the root of the shared resource that are part of the same file system.
server	A system that provides resources to another system; a system can be both a server, making local resources available to other systems, and a client, utilizing resources that reside on other systems.
sharing	Making a local resource available to remote systems.
unsharing	Making a shared local resource unavailable to remote systems.

Section 10
Introduction to DFS Administration

Some tasks involved in the administration of a distributed file system package are the same, regardless of the package you use. The Virtual File System architecture introduced in System V Release 4.0 provides a mechanism for administering file system types in a generic way. What this means is that a set of commands is provided with which you can administer different file system types in a uniform manner.

DFS administration provides generic commands that call package-specific commands, freeing you of the need to learn two sets of corresponding commands.

DFS administration commands and files allow you to share and unshare local resources, mount and unmount remote resources, and display information about shared and mounted resources. Resources can be shared or mounted *automatically*, by adding commands to a file so that the sharing or mounting is done whenever you enter run level 3; and resources can be shared and mounted *explicitly*, by typing commands at the command line.

DFS Commands and Files

The DFS Administration Utilities package installs six commands: **share**, **unshare**, **shareall**, **unshareall**, **dfshares**, and **dfmounts**. DFS administration also utilizes a number of files and commands installed by the Release 4.0 Essential Utilities or created by scripts or commands.

All the files and commands relevant to DFS administration are described in this section. Files and commands that operate on all file system types are described in this section only as they relate to administering distributed file systems.

The DFS administration files are

- **/etc/dfs/fstypes**, which registers the distributed file system packages you have installed on your system and sets one as the default. The **fstypes** file is created by the first distributed file system package you install.
- **/etc/dfs/dfstab**, which allows you to share a resource or a set of resources automatically when you enter run level 3.
- **/etc/vfstab**, which allows you to mount resources automatically when you enter run level 3.
- **/etc/dfs/sharetab**, which logs the resources currently shared on your system. The **sharetab** file is created by the **share** command and does not require handling by an administrator.
- **/etc/mnttab**, which logs the file systems currently mounted by your system, including remote file systems and directories. The **mnttab** file is created by the **mount** command and does not require handling by an administrator.

DFS administration commands are

- **share**(1M), which allows you to make a resource available for mounting by clients, or to display a list of the resources on your system that are currently shared.
- **unshare**(1M), which allows you to make a previously available resource unavailable for mounting by clients.
- **shareall**(1M), which executes a script that shares a predetermined set of resources.
- **unshareall**(1M), which executes a script that unshares all currently shared resources.
- **mount**(1M), which allows you to mount a remote resource on your system, or to display a list of resources, both local and remote, that are currently mounted on your system.
- **umount**(1M), which allows you to remove a remote resource you previously mounted.
- **mountall**(1M), which executes a script that mounts a predetermined set of resources.

- **umountall**(1M), which executes a script that unmounts all currently mounted resources.
- **dfshares**(1M), which displays a list of remote resources that are available to you, as well as a list of local resources that are currently shared.
- **dfmounts**(1M), which shows you which local resources are mounted by which clients.

Command Syntax

All DFS administration commands and the generic commands used by DFS administration have the following syntax:

```
command [-F fstype] [-o specific_options] [additional_options] [operands]
```

The options and operands are described below:

-F *fstype*	Indicates the specific file system type on which the command is to operate.
-o *specific options*	Indicates options specific to the file system type on which the command is to operate.
operands	Indicates nonoption arguments to the command.