qmail

By John Levine

Start Reading

Publisher: O'Reilly
Pub Date: March 2004
ISBN: 1-56592-628-5
Pages: 248

qmail concentrates on common tasks like moving a sendmail setup to qmail, or setting up a "POP toaster," a system that provides mail service to a large number of users on other computers sending and retrieving mail remotely. The book fills crucial gaps in existing documentation, detailing exactly what the core qmail software does.

qmail

By John Levine

Start Reading

Publisher: O'Reilly
Pub Date: March 2004
ISBN: 1-56592-628-5
Pages: 248

Copyright

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly & Associates books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (http://safari.oreilly.com). For more information, contact our corporate/institutional sales department: (800) 998-9938 or corporate@oreilly.com.

Preface

Since its release in 1998, qmail has quietly become one of the most widely used applications on the Internet. It's powerful enough to handle mail for systems with millions of users, including Yahoo Mail and VSNL (the largest ISP in India), while being compact enough to work on even the smallest PC Unix and Linux systems. Its component design makes it easy to extend and customize while keeping its key functions secure.

Qmail's design is rather different from its best-known predecessor, sendmail. People who are familiar with sendmail often have trouble recasting their problems and solutions in qmail terms. In this book, I try first to help the reader establish a qmail frame of mind, then show how the pieces of qmail work, and finally show how qmail can deal with some more complex mailing tasks such as handling mail for multiple domains, mailing lists, and gateways to other services.

What's Inside?

This book is organized into two sections, consisting of the following chapters.

Part I: Introduction to Qmail

Chapter 1, provides an overview of Internet email and the terminology used to describe it.

Chapter 2, outlines how qmail works, and gives a description of its basic parts and the philosophy behind its design and use.

Chapter 3, covers the basics of downloading, configuring and installing qmail, and other essential packages.

Chapter 4, finishes the job of configuring and starting qmail.

Chapter 5, addresses issues encountered when converting an existing sendmail system and its configuration files to qmail.

Chapter 6, looks at the issues involved in accepting mail from users on the qmail host and other systems, including cleaning up the sloppily formatted mail that most user mail programs send.

Chapter 7, describes the processing of incoming mail, various tricks to let users identify themselves as local users when roaming away from the local network, and adding cryptographic security to mail transfers.

Chapter 8, covers sorting, reading, and otherwise dealing with local mailboxes.

Chapter 9, covers anti-virus and anti-spam techniques, both those that can be built into qmail and ways to call external filters like Spamassassin.

Part II: Advanced Qmail

Chapter 10, defines the way that qmail delivers mail to local addresses.

Chapter 11, defines the way that qmail delivers mail to remote addresses.

Chapter 12, describes qmail's simple but powerful abilities to handle domains with their own sets of addresses, including building mail gateways to other services, and special routing for selected mail destinations.

Chapter 13, covers POP and IMAP, the standard ways that users pick up mail from PC mail programs, as well as "POP toasters," dedicated POP servers with many mailboxes.

Style Conventions

This book uses the following typographical conventions:

Italic

Indicates the names of files, databases, directories, hostnames, domain names, usernames, email addresses, sendmail feature names, Unix utilities, programs, and it is used to emphasize new terms when they are first introduced.

Constant width

Indicates configuration files, commands and variables, m4 macros and built-in commands, and Unix command-line options. It is used to show the contents of files and the output from commands. Keywords are also in constant width.

`Constant width bold`

Used in examples to show commands or text that you would type.

Constant width italic

Used in examples and text to show variables for which a context-specific substitution should be made. (The variable *filename*, for example, would be replaced by some actual filename.)

Examples and Patches

The examples from this book and the author's source code patches for qmail and related packages are freely downloadable from the author's web site at:

http://qmail.gurus.com

Comments and Questions

We have verified the information in this book to the best of our ability, but you may find that features have changed (or even that we have made mistakes!). Please let us know about any errors you find, as well as your suggestions for future editions, by writing to:
O'Reilly Media, Inc.1005 Gravenstein Highway NorthSebastopol, CA 95472(800) 998-9938 (in the United States or Canada)(707) 829-0515 (international or local)(707) 829-0104 (fax)

We have a web page for this book, where we list errata, examples, or any additional information. You can access this page at:
http://www.oreilly.com/catalog/qmail

To comment or ask technical questions about this book, send email to:
bookquestions@oreilly.com

You can sign up for one or more of our mailing lists at:
http://elists.oreilly.com

For more information about our books, conferences, software, Resource Centers, and the O'Reilly Network, see our web site at:
http://www.oreilly.com

You may also write to the author directly at:
qmail@gurus.com

Acknowledgments

I wish to thank my reviewers, Mark Delany and Russell Nelson, for careful reading of the manuscript and many suggestions to improve it. I particularly thank my editor Simon St.Laurent and the staff at O'Reilly for believing my assurances that this book would in fact be finished, despite mounting evidence to the contrary.

Part I: Introduction to Qmail

The first nine chapters provide an introduction to Internet email and qmail. They describe installing and configuring qmail, including advice on setting up a qmail system as a mail hub, converting an existing system from sendmail, and filtering out viruses and spam from incoming mail:

Chapter 1. Internet Email

Despite being one of the oldest applications on the Internet, email remains the Net's "killer application" for most users. For users' email to be sent and delivered, two kinds of programs have to work together, a *mail user agent* (MUA)[1] that a person uses to send and read mail, and a *mail transfer agent* (MTA) that moves the mail from server to server. Qmail is a modern MTA for Unix and Unix-like systems.

[1] Popular MUAs include pine and mutt on Unix systems, and Eudora, Netscape, Outlook, and Outlook Express on PCs and Macs.

Before diving into the details of qmail, it's a good idea to closely examine some of the basics of Internet email that apply to all MUAs and MTAs. Common terms like *envelope* and *mailbox* have special meanings in Internet mail parlance, and both the structure of mail messages and the path that messages take through the mail system are carefully defined. The essential documents are RFC 2821, which defines the Simple Mail Transfer Protocol (SMTP) used to move mail from one place to another, and RFC 2822, which defines the format of mail messages. These RFCs were published in April 2001, updating the original RFCs 821 and 822 published in 1982. (All RFCs are available online at http://www.rfc-editor.org.)

For many years, the only widely used MTA for Unix and Unix-like systems was the venerable sendmail, which has been around in one form or another for 20 years. As a result, many people assume that whatever sendmail does is correct, even when it disagrees with the RFCs or has unfortunate consequences. So even if you're familiar with sendmail (indeed, especially if you're familiar with sendmail), at least skim this chapter so we all can agree on our terminology.

1.1 Mail Basics

The Internet's SMTP mail delivers a message from a sender to one or more recipients. The sender and recipients are usually people, but may also be mailing lists or other software agents. From the point of view of the mail system, the sender and each recipient are addresses. The message is a sequence of lines of text. (RFC 2821 uses the word "mailbox" as a synonym for "address" and "content" for the message.)

1.1.1 Addresses

All email addresses have the simple form local-part@domain. The domain, the part after the at-sign, indirectly identifies a host to which mail should be delivered (although the host rarely has the same name as the domain). The local-part, the part before the at-sign, identifies a mailbox within that domain.

The set of valid domains is maintained by the Internet's Domain Name System (DNS). Every domain is a sequence of names separated by dots, such as example.com. The names in email domains consist of letters, digits, and hyphens. (If current efforts to internationalize domain names ever settle down, the set of valid characters will probably become larger.)

The local-part is interpreted only by the host that handles the address's domain. In principle, the mailbox can contain any characters other than an at-sign and angle brackets, but in practice, it is usually limited to letters, digits, and a small set of punctuation such as dots, hyphens, and underscores. Upper- and lowercase letters are equivalent in domains. It's up to the receiving mail host whether upper- and lowercase are equivalent in local parts, although most mail software including qmail treats them as equivalent.

Addresses appear in two different contexts: "envelope" data that is part of an SMTP transaction defined by RFC 2821, or in the header of a message defined by RFC 2822. In an SMTP envelope, addresses are always enclosed in angle brackets and do not use quoting characters or permit comments. In message headers, the address syntax is considerably more flexible. An address like "Fred.Smith"@example.com (Fred Smith) is valid in message headers but not in SMTP. (The form Fred.Smith@example.com is valid in either.)[2]

[2] Sendmail has often confused the two address contexts and has accepted message header formats in SMTP, both causing and masking a variety of bugs.

1.1.2 Envelopes

Every message handled by SMTP has an *envelope* containing the addresses of the sender and recipients). Often the envelope addresses match the addresses in the To: and From: headers in the message, but they don't have to match. There are plenty of legitimate reasons why they might not.

The envelope sender address is primarily used as the place to send failure reports (usually called bounce messages) if message can't be delivered. If the sender address is null (usually written in angle brackets as <>), any failure reports are discarded. Bounce messages are sent with null envelope senders to avoid mail loops if the bounce message can't be delivered. The sender address doesn't affect normal mail delivery.

The envelope recipient address(es) control where a message is to be delivered. Usually a message starts out with the envelope recipients matching the ones on the To: and Cc: lines, but as a message is routed through the network, the addresses change. If, for example, a message is sent to able@example.com and baker@domain.com, the copy sent

1.2 Mailstore

The *mailstore* is the place where messages live between the time that qmail or another MTA delivers them and the user picks them up. Often, it's also the place where the user saves messages after reading them.

I divide mailstores into two varieties: transparent and opaque. A transparent mailstore is one that an MUA can directly access as files, while an opaque one requires a network protocol to access. (As you might expect, there's considerable overlap between the two, with an MUA running on the system where the mail is stored using a user's mailstore as transparent and one running on a PC elsewhere using the same mailstore as opaque.)

A mailstore has several jobs beyond receiving messages. It must:

- Maintain a little per-message status information, such as whether a message is read, answered, or deleted
- Make it possible to group messages into multiple folders
- Make it possible to delete messages and move them from folder to folder

1.2.1 Transparent Mailstore

Unix systems have had a variety of mailstore file formats over the years. The oldest and still most popular is mbox, a format invented in two minutes in the 1970s for an early Unix mail program, and largely unchanged since then. An mbox is just a text file with the messages one after another. Each message is preceded by a From line and followed by a blank line. The From line looks like this:

```
From fred@example.com Wed Oct 06 19:10:49 1999
```

The address is usually (but not always) the envelope sender of the following message, and the timestamp is the time the message was added to the mailbox. Although it's easy to add a new message to an mbox, it's difficult to manipulate messages in the middle of a mailbox, and sharing a mailbox reliably between two processes is very tricky due to problems with file locking on disks shared over a network. Mboxes have been surprisingly durable considering their nearly accidental origins and their drawbacks, discussed in more detail in Chapter 10.

The MH mail system, developed at the RAND corporation in the 1980s, used a more sophisticated mailstore that made each mailbox a directory, with each message a separate file in the directory. Separate files made it easier to move messages around within mailboxes but still didn't solve the locking problems.

Qmail introduced *Maildir*, a mailbox format that uses three directories per mailbox to avoid any need for locking beyond what the operating system provides. Maildirs are covered in detail in Chapter 10.

1.2.2 Opaque Mailstore

Opaque mailstores became popular when PCs started to gain dial access to the Internet, and users started running mail programs on the PCs rather than using Telnet to connect to shared servers and running mail programs there. The two popular opaque schemes are Post Office Protocol (POP3 for Version 3), and Internet Message Access

1.3 The Structure of Internet Mail

Now that we've seen all the pieces of Internet mail, let's put them together and watch the typical path of a message as it's sent from one person to another.

First the sender runs a MUA, such as Pine or Eudora, and creates the message. Then a click of the Send button (or the equivalent) starts it on its way by passing it to the MTA (most likely qmail if you're reading this book), a process known as submitting the message. If the MUA is running on the same computer as the MTA, the MUA submits the message by running the MTA's injection component with the message as an input file. If the MUA is running on a separate computer, such as a Windows PC, the MUA makes a network connection to the computer running the MTA, and transfers the message using SMTP or a minor variant of SMTP called SUBMIT that's specifically intended for host-to-host message submission.

Either way, the MTA receives the message envelope with the sender and recipient addresses and the message text. Typically the MTA fixes up the header lines in the submitted message so that they comply with RFC 2822, then looks at the domain parts of each recipient address. If the domain is one that the MTA handles locally, the MTA can deliver the message immediately. In the more common case that it's not, the MTA has to send the message over the Net.

To figure out where to send the message, the MTA consults the DNS. Every domain that receives mail has an MX (Mail eXchanger) record in DNS identifying the host that receives mail for the domain.[3] Once the MTA has found the MX host for a domain, it opens an SMTP connection to the MX host and sends the message to it. In some cases, the MX host uses SMTP to forward the message again if, for example, the MX host is a firewall that passes mail between MTAs on a private network and the rest of the Internet.

[3] Well, they're supposed to at least. For backward compatibility with pre-1980 mail systems, if a domain has no MX record but does have an A record containing a numeric IP address, the mail system uses that instead.

Eventually the message arrives at a host where the MTA knows how to deliver mail to the recipient domain. Then the MTA looks at the local part of the recipient address to figure out where to deliver the mail. In the simple case that the address is a user mailbox, the MTA either deposits the message directly into the mailstore or, more likely, calls a local delivery agent program to deliver the mail. (On Unix, a popular local delivery agent is *procmail*, which does mail sorting and filtering as well as delivery.) Depending on the MUA that the recipient user has, the MUA may read the message directly from a transparent mailstore on the mail server, or use POP or IMAP to read the mail on a client PC.

A domain can have more than one MX record in its DNS. Each MX record contains a numeric value known as the preference or distance along with the name of a host. Sending systems try the MX host with the lowest distance first, and if that MX host can't be contacted, successively higher distances until one answers or it runs out of MXes. If there are several MX hosts at the same distance, it tries them all in any order before going on to hosts at a higher distance. If the sending host can't contact any of the MXes, it holds onto the message and retries later.

When the Internet was less reliable, backup MXes with a higher distance than the main MX were useful to receive mail for a domain when the main MX was unavailable, and then send it to the main MX when it came back. Now, backup MXes are only marginally useful, because sending hosts retry mail for at least a few days before giving up. They wait until the main MX is available and then deliver the mail. Multiple MXes at the same distance are still quite useful for busy domains. Large ISPs often have a dozen or more MXes to share the incoming mail load.

Chapter 2. How Qmail Works

People who are familiar with other mail transfer agents (MTAs), notably sendmail, rarely receive satisfactory results from qmail. Qmail was designed and written in a very different way from most other mail programs, so approaches used to solve problems with other programs don't work with qmail and vice versa.

2.1 Small Programs Work Together

Earlier MTAs were written as large monolithic programs. Sendmail, for example, is one large executable program that listens for incoming SMTP connections, accepts locally generated mail, queues mail, attempts outgoing SMTP deliveries, performs local deliveries, interprets *.forward* files, retries mail that for which earlier delivery attempts failed, and about 50 other functions. While this means that all of these functions can share utility routines and it's easy for one function to call on another or pass a message to another, it also means that sendmail is a large program (300 KB of code on my system, not including any libraries it uses) that is slow to start up and expensive to fork, and bugs anywhere in the code can potentially make any of the functions misbehave or fail. Other monolithic MTAs, such as smail and exim, share these problems.

Qmail, on the other hand, is about 10 small programs, none with as much as 30 KB of code, working together. This design approach offers many advantages.

2.1.1 Each Program Does One Thing

Each of qmail's programs performs a single function. For example, *qmail-lspawn* spawns (starts up) local deliveries, and *qmail-clean* deletes the queue files of messages that have been completely processed. The various programs use documented protocols to communicate, which makes it easier both to debug them and to substitute one version of a program for another.

For example, on a local area network (LAN) with several workstations, the most common mail setup is for one server to handle all of the incoming mail and deliveries. All the other workstations use that server as a "smarthost" and immediately forward locally generated mail to the smarthost. In this arrangement, each workstation traditionally has a complete implementation of the MTA, with configuration files set to forward mail to the smarthost. Note that about 90% of the MTA's function is present but not used, and strange bugs often surface when the configuration files on the workstations get out of sync with each other. The optional QMQP package makes it possible to install a tiny "mini-qmail" package on the workstations, with the only configuration being the address of the smarthost. In a regular qmail installation, the program *qmail-queue* takes a message and creates a queue entry so the message can be processed and delivered. Several other programs call *qmail-queue*, including *qmail-smtpd*, which receives incoming mail via SMTP, and *qmail-inject*, which receives locally generated mail. QMQP replaces *qmail-queue* with a small program that immediately forwards incoming mail to the smarthost. There's no need to install the queueing and delivery part of qmail on the workstations, but to the programs that call *qmail-queue*, mail works the same as it always did.

2.1.2 The Principle of Least Privilege

Most monolithic MTAs have to run as the super-user to open the "privileged" port 25 for SMTP service and deliver mail to user mailboxes that are not world-writable. Qmail uses the principle of least privilege, which means it runs only the program that starts local mail deliveries, *qmail-lspawn*, as root. All of the other programs run as nonprivileged user IDs. Different parts of qmail use different IDs—for example, only the parts that change the mail queue run as the user that can write to the queue directories. This offers an extra level of resistance to accidental or deliberate errors.

Qmail also offers the very useful ability to delegate management of a virtual domain to a Unix user in a simple and secure way. The user can manage all the addresses in the domain by adjusting his own files as needed without ever having to bother the system manager or run super-user privileged programs.

2.1.3 Program Wrapping

2.2 What Does a Mail Transfer Agent (MTA) Do?

The Internet model of email delivery divides the process into several separate stages and the software into several parts. The two important kinds of software are the Mail User Agent (MUA) and the Mail Transfer Agent (MTA). The MUA is the program that permits a user to send and receive mail. Familiar mail programs such as Pine, Elm, and Gnus on Unix and Eudora, Pegasus, Outlook, and Netscape or Mozilla on PCs are all MUAs. Each MUA has a rather complex user interface, and has many features, such as composing and reading mail, moving mail among mailboxes, and selecting the order in which to read mail. But an MUA doesn't deliver mail to other users; for that it hands its messages to an MTA.

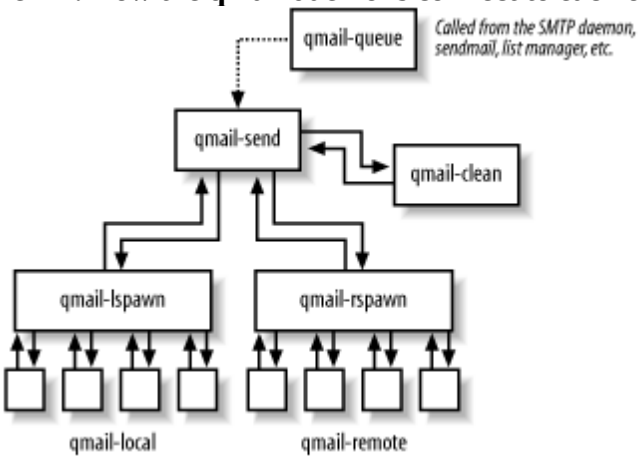
In the first stage of mail delivery, the message is submitted or injected to the MTA. Usually the message comes from an MUA, but it can just as well come from another program, such as a mailing list manager. The MTA examines the address(es) to which each message is sent, and either attempts to deliver the message locally if the address is local to the current host, or attempts to identify a host to which it can relay the message, relaying the message to that host. (If that last sentence sounds a little vague, it's deliberately so, because there are many different ways that mail relaying happens.) Each of these steps could fail—a local address might not exist, it might exist but the MTA might be temporarily or permanently unable to deliver the message to it, the MTA might be temporarily or permanently unable to identify a relay host, or the MTA might be able to identify a relay host, but temporarily or permanently unable to relay messages to it. In case of permanent failure, the MTA sends a failure report back to the message's sender. In case of temporary failure, the MTA hangs on to the message and retries until either the delivery succeeds or eventually the MTA treats the failure as permanent.

Although the basic idea of an MTA is simple, the details can be complex, particularly the details of handling errors. Fortunately, qmail handles most of the details automatically, so administrators and users don't have to.

2.3 The Pieces of Qmail

Qmail consists of five daemons that run continuously, and about ten other programs run either from those daemons or from other commands, as shown in Figure 2-1.

Figure 2-1. How the qmail daemons connect to each other



The primary daemon is qmail-send, which manages the message queue and dispatches messages for delivery. It is connected to two other daemons, qmail-lspawn and qmail-rspawn, which dispatch local and remote deliveries, respectively, using qmail-local and qmail-remote.

Once a message has been completely processed, with all deliveries having either succeeded or permanently failed, *qmail-send* notifies *qmail-clean* to remove the files for the message. The fifth daemon, *tcpserver* is discussed next.

2.3.1 A Message's Path Through Qmail

A message enters qmail either from another program within the system or via incoming SMTP. Regardless of where the mail originates, the originating program runs *qmail-queue*, which copies the message to a file in the queue directory, copies the envelope sender and recipient to a second file, and notifies *qmail-send*. For locally originating mail, *qmail-queue* is generally called from *qmail-inject*, or *newinject*, which adds missing header lines and cleans up address fields. (It's entirely legitimate for programs to call *qmail-queue* directly if they create messages with all needed headers. Mailing list managers such as Majordomo2 do for efficiency.) Most often, *qmail-inject* is run from *sendmail*, a small program that interprets its arguments like the legacy sendmail and calls *qmail-inject*. It's a useful shim to maintain compatibility with the many applications that call sendmail directly to send mail.

For mail arriving from remote systems, *tcpserver* runs as a daemon listening for incoming connections on the SMTP port. Each time a connection arrives, it runs *qmail-smtpd*, which receives a message via SMTP and calls *qmail-queue* to queue the message.

Regardless of where the message originates, *qmail-queue* writes the message to a temporary file in the *queue/todo* directory, putting a new Received: line at the top, and also saves the envelope sender and recipient addresses to files. Then it notifies *qmail-send* by writing a byte to a "trigger" socket file.

qmail-send takes the message out of queue/todo, and analyzes each recipient address to see if it's local, remote, or virtual.

Chapter 3. Installing Qmail

Qmail probably doesn't come preinstalled on your machine. It probably isn't even shipped in source form with your machine. You must go to the FTP server, download it, configure it, compile it, test it, and install it. If this sounds like a huge amount of work, it's not—some of these steps can be a single command.

3.1 Where to Find Qmail

The official place to get qmail is through Dan Bernstein's web and FTP server at http://cr.yp.to. (The .to domain is actually the island nation of Tonga, but they'll sell a "vanity" address to anyone willing to pay, and Dan's professional interests center around cryptography.) An alternate address is http://pobox.com/~djb/qmail.html.

Both URLs are currently redirected to Dan's FTP server, *koobera.math.uic.edu*, at the Math department of University of Illinois at Chicago. For the rest of this book, we'll nickname that site koobera. The actual name of the site is subject to change at any time, which is the whole point behind using cr.yp.to and pobox.com.

If you use a web browser or a graphical FTP program to open an FTP connection to koobera, the list of files you receive may be scrambled. Dan uses an FTP server of his own creation, publicfile, which is good and bad. It's good because it's a typical Dan Bernstein program: small, secure, and fast. It's bad because most web browsers and visual FTP programs don't know how to parse the server's listing format.

Visual FTP programs without special support for anonftpd's file format (EPLF, Easily Parsed Listing Format) cannot give you a listing of files. The standard command-line FTP that comes with BSD, Linux, and most versions of Unix has no such troubles, nor does the FTP distributed with versions of Windows, because neither attempts to parse the listing. The current version of squid, a popular proxy server, has support for EPLF, so if you're accessing the Net through a squid FTP proxy, you should have no troubles.

Once you've made sure you can contact the FTP server, make a directory where you're going to download and build your software such as */var/src* or */usr/local/src*, and FTP a copy of qmail there. Use *gunzip* and *tar* or *pax* to unpack it into a subdirectory.

3.1.1 Copyright

Dan Bernstein reserves most rights when he distributes qmail. Copyright law lets him prohibit anyone from making copies (except within fair use, which includes actually loading the software from disk into memory, memory into cache, cache into processor, and disk onto backup media and back again). Dan has given users several permissions, however. You can redistribute the source to any of qmail 1.00, 1.01, 1.02, and 1.03. This source must be unmodified, in the original *.tar.gz* format, and match a certain checksum provided by Dan.

In addition to redistributing unmodified source, you can also redistribute certain derived works. An executable that is equivalent to that which a user would create through the documented install process is also redistributable. In practice, this means that you can download, compile, patch, install, and use qmail any way you want. The one thing you can't do is to distribute modified versions of qmail. That's why all of the user modifications are distributed as patches relative to the distributed 1.03, rather than as modified versions of qmail itself.[1]

[1] Disclaimer: this description undoubtedly has a different legal import than Dan's permissions. Read Dan's license before you make any decisions about redistributing qmail yourself.

3.2 Creating the Users and Groups

Qmail uses a set of user ids and group ids to control access to various qmail facilities. Because Dan doesn't trust the system libraries (history is on his side), he doesn't make system calls to determine these uids. Instead, the uids are compiled into various programs. That means that the qmail users must exist prior to compiling the programs.

Some versions of Unix and Linux are distributed with the qmail users and groups already defined. If your */etc/passwd* (or equivalent) contains entries for alias, qmaild, qmaill, qmailp, qmailq, qmailr, and qmails, and your */etc/group* contains entries for qmail and nofiles, you're all set and can skip ahead to "Configuring and making the software." Otherwise you must create the users and groups yourself. There are several ways to do this.

3.2.1 The adduser Script

Some Unices have a program called *useradd* or *adduser* to create users and groups. Often, use of this program is mandatory, because the machine uses shadow passwords. To be safe, use the program when it exists. The *INSTALL.ids* file has the necessary commands. Copy that file to */tmp/mu*, locate the right set of commands, delete everything else, delete the pretend root prompt characters in front of the commands, save it to a file, and run that file using *sh /tmp/mu*.

3.2.2 Adding by Hand

Some Unices let you create groups by editing the */etc/group* file and users by editing the */etc/passwd* file, the latter typically through the *vipw* program. Edit */etc/group* and add the following two lines:

```
qmail:*:2107:
nofiles:*:2108:
```

Make sure that 2107 and 2108 are unique group id numbers. If you have to change them, also change them in the user information in the next section.

Always edit */etc/passwd* using the *vipw* program, if it exists. It ensures that your shadow password database (if you're using one) is kept up to date. It also locks the password file against other programs changing it. If you have no *vipw* program, then go ahead and edit with your favorite text editor.

Add the following set of lines to */etc/passwd*:

```
alias:*:7790:2108::/var/qmail/alias:/bin/true
qmaild:*:7791:2108::/var/qmail:/bin/true
qmaill:*:7792:2108::/var/qmail:/bin/true
qmailp:*:7793:2108::/var/qmail:/bin/true
qmailq:*:7794:2107::/var/qmail:/bin/true
qmailr:*:7795:2107::/var/qmail:/bin/true
qmails:*:7796:2107::/var/qmail:/bin/true
```

Verify that 7790 through 7796 are unique user id numbers. If they're already in use, pick some other unused numbers. The exact id numbers don't matter so long as they're all different from each other and different from every other user on the system.

3.2.3 Nofiles Group Really Has No Files

3.3 Configuring and Making the Software

The vast majority of the qmail configuration occurs at runtime. There are, however, a few configuration options that can only be changed at compile time. These options are, as you might expect, not often changed. If you're reading this book front to back, skip this section and come back to it later, because most of the compile-time options won't make any sense to you.

These configuration options are each in a separate file in the qmail source directory, the first line or lines of which are the value. Lines beyond those have an explanation of the meaning of the value.

3.3.1 conf-break

Qmail permits users to have subaddresses, which qmail calls extensions. For example, *nelson-qmail-book@crynwr.com* has an extension of "qmail-book" if the break character is a dash. By default it is a dash character, but some sysadmins may wish to use a plus or equals character for compatibility with other software. (Sendmail uses a plus sign.[2])

[2] One potential cause of confusion is the difference between the break character and the character that separates the parts of extensions. *conf-break* specifies the break between the username and the extension. Extensions are also split into parts; however, they are always split at a dash character. So, if you set your break character to a plus, then *nelson+list-qmail* will be matched by *~nelson/.qmail-list-default* if there's no better match. See Chapter 7 on local delivery for more information.

3.3.2 conf-cc

The compiler is not set in the makefile, as is typical for a Unix program. The makefile actually uses a generic *compile* script. This script is created by the makefile. It combines *conf-cc* with some more information. If your C compiler needs special optimization flags, this is the place to put them.

3.3.3 conf-groups

The first two lines of this file list the names of the groups that qmail uses. They are used in the building process to get the group id (gid) for the install process. The first is the name of the group that several qmail users use to share information through group permissions. The second is the name of the group used by the other qmail users who don't need to use group permissions. Don't change this unless your system already has groups called qmail or nofile that conflict with qmail's use of them.

3.3.4 conf-ld

The first line of this file is the command used to link *.o* files into an executable. The most common change is to replace the --s flag it contains with --g to preserve symbols for debugging. If your linker supports static shared libraries, which start up faster than the more usual dynamic shared libraries, this is where you put the flags or command to use them.

3.3.5 conf-patrn

Qmail refuses to deliver mail to insecure accounts. If a user allows anyone to modify files in his home directory,

3.4 Patching Qmail

Dan's license for qmail forbids the distribution of modified versions of qmail, so many people offer add-ons and patches that you can apply to qmail yourself. Add-ons are distributed as installable packages that you download and install like any other package, but patches are distributed as text files of differences between the original and the patched version of qmail, as created by the *diff* utility. You don't need any patches to get qmail going (other than the errno patch for recent Linux versions), but because so many useful changes are distributed as patches, nearly everyone uses a few of them, so you should be prepared to use them.

The *patch* program, distributed with most Unix-like systems, reads the patch files and applies the changes. If your system doesn't have it, it's available for download from the Free Software Foundation at http://www.fsf.org/software/patch/patch.html. To apply a patch to a package, be sure the source code for the package is stored in a subdirectory of the current directory with the package's usual name (such as *qmail-1.03)*, then feed the patch file to *patch*:

```
$ patch < some-patch.txt
```

patch produces a chatty report of its progress. Patch files invariably contain context diffs, so *patch* warns you if the file you're patching appears not to match the one on which the patch is based. You must look at the rejected patches in the source directory with filenames like filename.*rej* and figure out where the patches should go. Occasionally when you're applying multiple patches to the same set of files, the patches can collide, but for the most part, the useful patches to qmail apply without trouble. Once a patch is applied, rebuild and reinstall the package from the patched source code.

If you're installing the recommended netqmail package, you've already patched the source. Netqmail includes a patch file called *netqmail-1.05.patch* that is automatically applied by *./collate.sh*.If you want to try patching qmail, a good patch to start with is the QMAILQUEUE patch, available at http://www.qmail.org/qmailqueue-patch. (Netqmail users needn't bother, because it's already applied.) It's quite small but very useful. Once you've applied the patch, any qmail component that calls *qmail-queue* to queue a mail message checks the QMAILQUEUE environment variable and if it's set, uses it as the name of a program to run instead. This makes it easy to insert filters of various sorts into qmail's processing without having to add special code to individual programs.

Now that you've built and installed qmail, daemontools, and perhaps other add-on packages, the next chapter tells you how to start it all up.

Building with Recent GLIBC and Fixing the errno Problem

If your system uses the GNU GLIBC Version 2.3.1 or newer, qmail won't compile without some small patches. This problem affects most recent versions of Linux. The qmail source code defines *errno*, the place where system calls put error codes, to be an int variable, but in these libraries it's not, it's a macro.

In the source file *errno.h*, replace the line that declares errno with this:

```
#include <errno.h>
```

In the source files *dns.c* and *qdb_cook.c*, find any lines that declare *errno* or *h_errno* and delete them.

Chapter 4. Getting Comfortable with Qmail

This chapter guides you through the basics of running qmail and delivering mail to users on your qmail host. It's quite possible to run qmail in parallel with your old mail system, which is usually a good idea during a transition, so you can do everything in this chapter while leaving your old mail system in place.

4.1 Mailboxes, Local Delivery, and Logging

Before you start up qmail, you must make a few configuration decisions. None of these are irrevocable, but if you know what you want, it's easier to set them that way at first than to change them later.

4.1.1 Mailbox Format

Qmail supports two mailbox formats: the traditional mbox and Dan's newer Maildir. I won't belabor the difference here (see Chapter 10 for more details) other than to note that mbox stores all its messages in a single file and is supported by all existing Unix mail software, while Maildir stores each message in a separate file in a directory, and is supported by a reasonable set of software (including procmail, the mutt MUA, and several POP and IMAP servers) but not as many as mboxes. If you're converting from an existing mail system that uses mboxes, it's easier to keep using mboxes, but if you're starting from scratch, go with Maildirs.

4.1.2 Local Delivery

If you use mbox files, qmail normally puts the incoming mailboxes in users' home directories. That is, for user fred, the mailbox would be *~fred/Mailbox*. Older mail programs often put all of the mailboxes into */var/mail*. For both security and disk management reasons, it's better to put the mail in the user's home directory with his or her other files, but if you have existing mailboxes in */var/mail*, it's not hard to persuade qmail to continue delivering mail there.

If you're converting from an older MTA, you can either set up qmail to deliver into the same mailboxes as the old MTA or, if you're feeling cautious, set qmail to deliver into Maildirs or home directory mboxes while the old MTA still delivers to */var/mail*. (The disadvantage is that once you're happy with qmail, you have to convert and merge the old mailboxes. See Section 4.7 later in this chapter.)

4.1.3 Logging

The traditional way to make log files is with the system syslog facility. It turns out that syslog is a serious resource hog and on a busy system can lose messages. On a small system this doesn't matter, but on a busy mail host, it sucks up significant resources that otherwise could be devoted to something more useful. Dan Bernstein wrote a logging program called *multilog*, part of the daemontools package, which is far faster and more reliable than syslog, but not particularly compatible with it. If you're sure that syslog won't be a bottleneck, go ahead and use it, but if you might eventually want to use *multilog*, you're better off starting with it because switching a running system is a pain in the neck.

4.2 An Excursion into Daemon Management

A daemon is a program that runs in the background (that is, without interacting with a user) and is useful. On Unix systems, there are two kinds of daemons: the ones that run continuously and the ones that run on demand. Familiar Unix examples of continuous ones include named, the DNS server, and httpd, the Apache web server, while on-demand ones include servers for network services, such as *telnet* and *ftp*, and cleanup scripts run once a day or once a week. The on-demand ones are all started from continuous servers such as *cron* for time-based services, and *inetd* or *tcpserver* (the qmail replacement for *inetd*) for network services.

The daemontools package provides a consistent way to run continuous daemons, optionally (but almost invariably) also arranging to collect log information that the daemons produce. The two key programs are *supervise*, which controls a single daemon, and *svscan*, a "superdaemon," which controls multiple copies of *supervise* and connects each daemon with its logger.

For each daemon to be controlled, *supervise* uses a directory containing information about the daemon. The only file that you must create in that directory is *run*, the program to run. Although it can be a link to the daemon, it's usually a short shell script that sets up the environment and then exec's the daemon. The *supervise* program creates a subdirectory also called *supervise*, where it stores info about what it's doing. Once *supervise* is running, you can use the *svc* program to stop and start the daemon, and send signals to it. (This consistent way to signal daemons is one of *supervise*'s greatest strengths.)

To run *supervise*, follow these two steps: create */service*, which you do with a regular mkdir command as the super-user, and start *svscan*, which I cover in the next section. Once *svscan* is running, it looks at */service* and starts a *supervise* process in each of its subdirectories. Every five seconds it looks again and creates new processes for any new subdirectories. If a subdirectory has a sub-subdirectory called *log*, *svscan* arranges to log the output of the program. In this case, it starts a pair of processes connected by a pipe, equivalent to:

```
supervise subdir | supervise subdir/log
```

The *log* subdirectory contains a *run* file that invariably runs *multilog* to write the output into a rotating set of log files.

4.2.1 Starting a Daemon

One of the least standardized aspects of Unix and Unix-like systems is the way that you start daemons at system boot time. Even if you use *supervise* as I recommend, you still must start the *svscan* daemon to get everything else going. Here are some hints to start *svscan*. If you ignore my advice and run daemons directly, start each of them the way I recommend you start *svscan*.

Versions 0.75 and later of daemontools include a startup script for *svscan* called *svscanboot*, and the daemontools installation process tries, usually successfully, to edit a call to that program into your system startup scripts. It sets up the environment and runs *svscan*, piping its output into a new program called *readproctitle* that copies anything it reads on top of its program arguments, which means that any error messages from *svscan* will show up in ps listings in the arguments to *readproctitle*. This kludge makes it possible to see what's wrong if *svscan* has trouble starting up or starting *supervise* for any of the directories under */service*:

Single rc file

Solaris and some versions of BSD put all of the startup commands in a file called */etc/rc*, usually with local

4.3 Setting Up the Qmail Configuration Files

The final hurdle before starting up qmail is to create a minimal set of configuration files. The qmail distribution includes a script called *config* that makes a set of configuration files that's usually nearly right. I suggest you run the *config* script, then look at the files to see what it did and fix the files up as needed. All of the configuration files are in */var/qmail/control*. The ones you need to create include:

me

The name of this host, e.g., mail.example.com. This provides the default to use for many other configuration files.

defaulthost

The hostname to add to unqualified addresses in submitted mail. If your email addresses are of the form mailbox@example.com, this would contain example.com, so that mail to, say, fred is rewritten to fred@example.com. (Note that this rewriting happens only to locally submitted mail sent via *qmail-inject*, not to mail that arrives via SMTP.)

defaultdomain

The domain to add to unqualified domains in submitted mail addresses, usually your base domain, such as example.com. This would rewrite fred@duluth to fred@duluth.example.com. (This rewriting also happens only in locally submitted mail.)

locals

Domain names to be delivered locally, one per line. Mail to any domain listed in *locals* is delivered by treating the mailbox part as a local address. This usually contains the name of the host and the name of the domain used for user mailboxes, such as example.com and mail.example.com. Do not list virtual domains (domains hosted on this machine but with their own separate sets of mailboxes) in *locals*. I discuss them later.

rcpthosts

Domains for which this host should accept mail via SMTP. This generally contains all of the domains in *locals*, as well as any virtual domains and any domains for which this host is a backup mail server. If *rcpthosts* does not exist, qmail accepts and delivers mail for any domain, a severe misconfiguration known as an "open relay," which will be hijacked by spammers. Be sure your *rcpthosts* file exists before starting qmail. If you haven't defined any virtual domains, just copy *locals* to *rcpthosts*.

There are over 20 more control files, but the rest can be left for later.

4.4 Starting and Stopping Qmail

Starting qmail is easy in principle. You run *qmail-start* and it starts the four communicating daemons that qmail needs. Two details complicate the situation: the default delivery instructions, and connecting the daemons to whatever you want to use for logging.

Because the daemontools package of which *supervise* is a part wasn't written until after qmail 1.03 was released, all of the provided startup files use *splogger* to send the log information to syslog. I find daemontools greatly preferable, so I primarily discuss how to set up qmail using *supervise*.

4.4.1 Choosing a Startup File

Qmail 1.03 comes with a selection of startup files you can use, either directly or as a starting point for a customized startup file of your own. You can find the startup files in */var/qmail/boot*. None of them are usable directly with daemontools, but they're useful as templates. The differences among them only affect what happens when mail is delivered to a user who has no *.qmail* file, because the only difference is the string to use as a default *.qmail*. They include:

binm1

Default delivery using */usr/libexec/mail.local*, the 4.4BSD mail delivery agent, which puts mail in */var/spool/mail*

binm1+df

Same as *binm1*, also providing dot-forward emulation

binm2

Default delivery using */bin/mail* with SVR4 flags, which also puts mail in */var/spool/mail*

binm2+df

Same as *binm2*, also providing dot-forward emulation

binm3

Default delivery using */bin/mail* with flags for older versions of Unix; puts mail in */var/spool/mail*

binm3+df

4.5 Incoming Mail

Next, install the SMTP daemon to receive incoming mail. If you have another mail system already running, set up qmail's SMTP daemon for testing on a different port than the standard port 25.

4.5.1 Configuration Files

The SMTP daemon only needs one configuration file to run: */var/qmail/rcpthosts*. For simple applications, *rcpthosts* can contain the same list of domains as *locals*. It is very important that you set up r*cpthosts* before starting your SMTP daemon. If you don't, your mail system will be an "open relay," which will transmit mail from anywhere to anywhere and be abused by spammers and blacklisted.

A little later we'll also be setting up a control file to tell the daemon what IP addresses are assigned to local users allowed to relay mail.

4.5.2 Setting Up the Daemons

Setting up SMTP involves three layers of daemons. *Supervise* runs *tcpserver*, which waits for incoming network connections. Each time a remote system connects, *tcpserver* starts a copy of *qmail-smtpd*, which collects the incoming message and passes it to *qmail-queue* for delivery. To run it under *supervise*, create a pair of directories, and call them */var/qmail/supervise/qmail-smtpd* and */var/qmail/supervise/qmail-smtpd/log*:

```
# mkdir /var/qmail/supervise/qmail-smtpd
# mkdir /var/qmail/supervise/qmail-smtpd/log
# chown root /var/qmail/supervise/qmail-smtpd  /var/qmail/supervise/qmail-smtpd/log

# mkdir /var/qmail/supervise/qmail-smtpd/log/main
# chown qmaill /var/qmail/supervise/qmail-smtpd/log/main
```

The *run* script eventually becomes rather complex as you add code to handle local versus remote users, spam filters, and the like, but this is adequate to start (see Example 4-3).

Example 4-3. Running the SMTP daemon

```
 1. #!/bin/sh
 2. limit datasize 3m
 3. exec tcpserver \
 4.     -u000 -g000 -v -p -R \
 5.       0 26 \
 6.        /var/qmail/bin/qmail-smtpd 2>&1
```

The limit command on line 2 defends against a denial-of-service attack in which the attacker feeds the SMTP daemon a gargantuan message that fills up all of memory and crashes the machine. Then the tcpserver command on line 3 accepts SMTP connections and runs *qmail-smtpd* for each one. The -u and -g flags on line 4 set the user and group numbers; substitute the values on your system for qmaild. The -v flag does verbose logging (recommended, it's not that verbose) and -p does "paranoid" validation of deduced hostnames of remote systems. The -R flag means to not try to collect ident information from the remote host. (Ident information is rarely useful and a failed ident request can stall the daemon startup for 25 seconds.) On line 5, host number 0 means to accept connections on any IP address assigned to this machine, and 26 means to use port 26 rather than standard SMTP port 25, which allows you to run the daemon for testing without interfering with an existing MTA on port 25. (If there's no other MTA running, you might as well use port 25.) Finally, line 6 has the command for *tcpserver* to run once a connection is open. At the end, 2>&1 combines any output to standard error with the regular output so both appear in the log files.

4.6 Procmail and Qmail

If you do any sorting or filtering of incoming mail, you should install the popular procmail mail filtering package. Although procmail's filter definition language is terse to the point of obscurity, it's very powerful and easy to use once you get the hang of it. In the past, procmail's default mailbox location was in */var/mail*, and it didn't support Maildirs. Recent versions of procmail work well with qmail. Version 3.14 added support for Maildirs, and it's now easy to compile procmail to put the default mailbox in qmail's preferred place.

The source for procmail is available at http://www.procmail.org. Download it to a local work directory and unpack it. To make its default delivery be to *Mailbox*, edit the file *src/authenticate.c*. Around line 47 find the definition of MAILSPOOLHOME, remove the comment characters at the start of the line, and change the file name to *Mailbox*:

```
#define MAILSPOOLHOME "/Mailbox"
```

Or to make the default delivery to a user's Maildir, type:

```
#define MAILSPOOLHOME "/Maildir/"
```

(Note the slash after the directory name, which tells procmail that it's a Maildir rather than an mbox.)

Then make and install procmail as described in its *INSTALL* file. The procmail installation recommends that you install procmail as set-uid to root. When working with qmail, it does not need set-uid to work correctly, and I recommend that you don't do this. When used as the mail delivery agent for sendmail, procmail needs set-uid to run as the id of the delivered-to user. Qmail switches to the correct user ID before running procmail, as it does for any delivery agent, so procmail doesn't need to do so. Installing as set-uid won't cause any immediate problems, but it will pose a possible security problem should there turn out to be lurking bugs in procmail.

To use procmail as your default delivery agent, use this in your qmail *run* file:

```
exec env - PATH="/var/qmail/bin:$PATH" \
        qmail-start '|preline procmail'
```

(The preline command is a qmail component that inserts a From line that procmail needs at the front of the message.) Alternatively, to make procmail the delivery agent for an individual user, put the procmail command into the user's *.qmail* file:

```
|preline procmail
```

Sendmail systems often pass the address extension as an argument to procmail so it can be used as $1 in scripts. That's easy enough to do in *.qmail-default*:

```
|preline procmail -a "$EXT"
```

Procmail makes most environment variables available in its rule files anyway, so if you're not converting from sendmail, just use $EXT in your scripts.

It's frequently advantageous to use different procmail filter definitions for different qmail subaddresses. For example, if you are user fred and use the address fred-lists for your mailing list mail, *.qmail-lists* could contain this:

```
|preline procmail procmaillists
```

to use *procmaillists* to sort list mail.

4.7 Creating Addresses and Mailboxes

With the setup so far, every user in */etc/passwd* automatically has a mailbox with the same name as the login name.[1] If you're using mbox mailboxes, each mailbox is created the first time a message is delivered to it. If you're using Maildirs, you must create the Maildirs yourself using maildirmake. If all of your home directories are stored in */usr/home* or */home*, it's easy enough to give everyone a Maildir. Run a script like this as root to create them:

[1] That's not quite true; for security reasons qmail won't deliver mail to the root user.

```
cd /home
for u in *
do
    maildirmake $u/Maildir
    chown -R $u/Maildir
done
```

The chown is important so that each user owns his own Maildir.

If you have more than two or three mailboxes to create, use the convert-and-create script from http://www.qmail.org/. It creates Maildirs for every user with a mailbox, and copies the mail from */var/mail* mboxes into the new Maildirs.

Once you've created Maildirs for all of your existing users, creating them for new users is considerably easier. Just add a line or two to your system's *adduser* script to create the Maildir as it creates the rest of the new user's files. On Linux systems, use maildirmake to create */etc/skel/Maildir*, a prototype that gives every subsequent new user a Maildir.

4.8 Reading Your Mail

If you use mbox mailboxes, the only additional change you may have to make is to tell your mail program (and your shell if it's one that reports new mail) that the mailbox is in *~/Mailbox* rather than in */var/mail*. Most mail programs check the shell variable $MAIL. For testing, change the MAIL variable at your shell prompt:

```
% setenv MAIL ~/Mailbox    (in csh)
$ export MAIL=~/Mailbox    (ksh and bash)
$ export MAIL=$HOME/Mailbox (in sh)
```

Once you're committed to qmail and your mail is in */var/mail*, you want to copy everyone's mailbox to their home directory, using the convert-and-create script mentioned previously. Then, find the place in */etc/profile* or */etc/cshrc* that sets MAIL and change it to refer to the new mailbox location.

If you use Maildirs, your options are simpler. The only mail program with built-in Maildir support is mutt. On qmail.org there are some patches for pine to handle Maildirs, and a version of movemail for GNU Emacs users. If you use something else, you can use the scripts distributed with qmail such as *elq* or *pinq* that copy mail from a Maildir into an mbox and then run elm or pine. Honestly, if a user normally uses a mail program that expects mbox mailboxes, it's easier to tell qmail to use mboxes than to tell the program to use Maildirs.

An alternative that makes Maildirs available to most mail clients is to use an IMAP server such as Courier that handles Maildirs (see Chapter 13). The IMAP server can retrieve mail from the Maildir and from any number of Maildir-format subfolders. You can set up pine or Mozilla to use IMAP to deal with the Maildir folders, and use its built-in mbox support to handle mboxes directly as files. This has the added advantage that you can check your mail using any IMAP client from other computers if you're away from your usual computer.

4.9 Configuring Qmail's Control Files

Qmail is controlled by a large set of control files stored in */var/qmail/control*. Unlike some other MTAs that group everything into one huge file that they have to parse to figure out what's what, qmail puts each different kind of information into a separate file, so that each file needs little or no parsing. All files are lines of plain text (although a few files are compiled into CDB databases before use). Some, noted below, allow comment lines with a # at the beginning of the line. In files where each line contains multiple fields, the fields are separated by colons.

Most of the control files are optional, and qmail uses a reasonable default in most cases if a file isn't present. The only files that are absolutely essential are *me*, which contains the hostname of the local host, and *rcpthosts*, which lists the names of the domains for which this host accepts mail.

Here's a list of all the control files in alphabetical order, noting which component uses each one. Many of the optional patches introduce new control files, which are discussed during the description of the patch.

Checking Your Configuration with qmail-showctl

Because qmail has a long and somewhat daunting set of configuration files, the package includes a program to tell you what your current configuration is. Run */var/qmail/bin/qmail-showctl* and pipe its output through a pager like *more* to see a detailed narrative of the contents of all of the configuration files. For missing files it tells you what defaulted values are in use.

If you receive the message I have no idea what this file does, it means that the file is not one that *qmail-showctl* recognizes. You can put any extra files you want in the *control* directory and qmail doesn't care. Occasionally, it's useful to leave notes to yourself or to save an old version of a file you're changing. There is an exception: *qmail-showctl* looks at all the files and makes that complaint about the ones not in its list. If you apply patches that use new configuration files, most of the patches don't bother to update *qmail-showctl*, so it'll typically complain about those files too, which is equally harmless.

badmailfrom (qmail-smtpd)

Envelope addresses not allowed to send mail. If the envelope From address on an incoming message matches an entry in badmailfrom, the SMTP daemon will reject every recipient address. Entries may be either email addresses, or *@dom.ain* to reject every address in a domain. This is a primitive form of spam filtering. These days, it's mostly useful to block quickly a mailbomb or flood of rejection messages.

bouncefrom (qmail-send)

4.10 Using ~alias

Although qmail automatically handles deliveries to most users with entries in the Unix password file (or qmail's adjusted version of it; see Chapter 15), any useful mail setup also needs to deliver mail to addresses unrelated to entries in the password file. Qmail handles this in a simple, elegant way with the alias pseudo-user. As part of the installation process, create a user called alias and set its home directory to */var/qmail/alias*. When qmail is running, if mail arrives for a local mailbox that isn't in the normal list of users, qmail prepends alias- to the address and retries the delivery. This makes any address not otherwise handled in effect a subaddress of alias, so you can handle addresses by putting *.qmail* files into *~alias*. For example, if you have a user robert and want mail addressed to bob to be forwarded to him, create *~alias/.qmail-bob* and in it put &robert. Since qmail handles deliveries using the *.qmail* files in *~alias* the same way that it handles any other deliveries, you have all of the same options delivering to nonuser addresses that you do to user addresses.

Because qmail doesn't deliver to root and other users that have a 0 user ID or that don't own their home directories, you should arrange to send root's mail to the system manager by creating *~alias/.qmail-root*. Also create *.qmail-postmaster*, *.qmail-abuse*, *.qmail-webmaster*, and any other role addresses that you want to support.

The final default delivery is, not surprisingly, found in *~alias/.qmail-default*. If that file doesn't exist, unknown addresses bounce, often just what you want. The most common thing to put in that file is a line to run the *fastforward* program (see the next section) to take delivery instructions from a file of addresses, roughly as sendmail does. You can also implement other default delivery rules. For example, if you want to make mail to subaddresses of *~alias* users default to the base address, so mail to fred-foop is delivered to fred if it's not otherwise handled, put a line like this in your default delivery file. (It appears wrapped here, but it has to be on one long line in the file.)

```
| case "$DEFAULT" in *-*) forward "${DEFAULT%%-*}" ;; *) bouncesaying
"Sorry, no mailbox here by that name. (#5.1.1)" ;; esac
```

This says that if an address contains a hyphen, strip off the hyphen and everything after it and redeliver it. Otherwise bounce the message. The bouncesaying command lets you provide your own failure message, but a simple exit 100 would do the trick as well, telling qmail to bounce.

4.11 fastforward and /etc/aliases

Sendmail and other MTAs use configuration files such as */etc/aliases* that contain lists of mailboxes and forwarding instructions. While qmail doesn't have a built-in feature to do that, the add-on *fastforward* package (available at http://cr.yp.to/fastforward.html provides both a mostly compatible way to handle existing */etc/alias* files) and a more general scheme to handle files with forwarding instructions and mailing lists.

4.11.1 Installing fastforward

You can download and install the *fastforward* package the same way you install Dan's other programs, as described in Chapter 3. This section describes *fastforward* Version 0.51.

4.11.2 Using fastforward

The central program in the *fastforward* package is *fastforward* itself. It's designed to be run from a *.qmail* file. When run, it gets the recipient address from $RECIPIENT or optionally $DEFAULT@$HOST, looks up the address in a delivery database, and if it finds the address, follows the delivery instructions for the address.

fastforward takes its instructions from a CDB-format file. There are two ways to create the file: using newaliases to create */etc/aliases.cdb* from */etc/aliases*, which is in sendmail format, or using *setforward* to create a CDB from an arbitrary file, which is in a different, more flexible format. All of *fastforward*'s CDB files have the same format, regardless of which program created them.

The CDB file can refer to mailing list files of addresses; the difference is that the CDB file contains addresses and delivery instructions, while a mailing list file just contains a list of addresses and other mailing list files, for use within a delivery instruction. Mailing list files can be created by *newinclude*, which reads input containing a list of addresses in a format similar to the one sendmail uses for *:include:* files, or by *setmaillist*, which reads input in a more flexible format. Mailing list files created by either program have the same format, so you can use the input format that is more convenient. Compiled mailing list files have the extension *.bin*. In this section, I describe */etc/alias* compatibility and leave the rest for the sections on virtual domains (Chapter 12) and mailing lists (Chapter 14).

The most common way to use *fastforward* is to call it from *~alias/.qmail-default* so it can take a crack at any addresses not handled otherwise:

```
| fastforward /etc/aliases.cdb
```

Or you can also combine it with other default rules. For example, to use *fastforward* and then redeliver mail to subaddresses to the base address of the subaddress:

```
| fastforward -p /etc/aliases.cdb
| case "$DEFAULT" in *-*) forward "${DEFAULT%%-*}" ;; *) bouncesaying "Sorry, no
mailbox here by that
name. (#5.1.1)" ;; esac
```

The -p flag says to "pass through," that is, exit 99 if an address is found or exit 0 if not, so qmail goes on to the next line in the *.qmail* file if *fastforward* didn't deliver it. (In the absence of -p, *fastforward* exits 0 if it forwards the message and 100 otherwise to bounce the mail.)

4.11.3 Alias File Format

Chapter 5. Moving from Sendmail to Qmail

More often than not, a site that plans to run qmail is already running some other mail software on a Unix-ish server, and more often than not, that software is sendmail. This chapter walks through the steps involved in moving a mail system from sendmail to qmail.

5.1 Running Sendmail and Qmail in Parallel

Users tend to be upset when they can't access their email, so it's rarely possible to shut down the old mail system, spend a day getting the new system installed and tested, then turn the mail back on. Fortunately, you don't have to. It's easy to run sendmail and qmail in parallel on the same machine, delivering mail into the same mailboxes, until you're satisfied qmail is working properly, and then shut sendmail down.

Any MTA receives mail through two routes: local and remote. On Unix systems, local mail is injected via the *sendmail* program, and remote mail is injected via SMTP. When you're running qmail and sendmail in parallel, as long as */usr/lib/sendmail* is a link to sendmail, local mail will go to sendmail, and as long as sendmail is listening on port 25, remote mail will also go to sendmail. While you're testing, put qmail's version of sendmail somewhere else, say */var/qmail/bin/sendmail*, and run qmail's SMTP daemon on port 26.

Once you're happy with your qmail installation, move the original */usr/lib/sendmail* to */usr/lib/sendmail.old* (and similarly for any other links to it such as */usr/sbin/sendmail*) and link the qmail version in its place. That will start routing local mail to qmail.

For remote mail, kill the sendmail daemon, and restart the qmail SMTP daemon running on port 25. That will start routing remote mail to qmail. Because sendmail probably still has some mail to flush out, restart the sendmail daemon but without the *-bd* flag that makes it listen on port 25. A typical command would be *sendmail -q30m* to keep retrying failed deliveries every 30 minutes. After a few days, or when the sendmail queue is empty, you can shut sendmail down for good.

5.1.1 Sendmail Switching Systems

Some versions of BSD and Linux have their own schemes to handle multiple mail systems with different versions of sendmail by providing a layer of indirection between the sendmail program that other applications call and the actual program provided by the mail package. These schemes don't do anything that the direct approach can't also do, but they document the setup better and are more likely to survive system upgrades, so you should use them when you can.

NetBSD and FreeBSD use a program called *mailwrapper*, which is installed where sendmail would usually go. It consults a file called */etc/mail/mailer.conf*, which has the names of the actual programs to run when sendmail is called under any of its many aliases. (See Example 5-1.)

Example 5-1. Typical mailer.conf

```
sendmail        /var/qmail/bin/sendmail
send-mail       /var/qmail/bin/sendmail
newaliases      /var/qmail/bin/newaliases
```

Debian and Red Hat Linux have an "alternatives" scheme that uses symlinks. In a typical alternatives setup, */usr/sbin/sendmail* is a symlink to */etc/alternatives/mta*, which is in turn a symlink to the real sendmail program. You can just symlink */etc/alternatives/mta* to */var/qmail/bin/sendmail* or use the *alternatives* (Red Hat) or *update-alternatives* (Debian) command to make the links.

5.2 User Issues

There are two important differences visible to mail users when moving from sendmail to qmail: mailbox format and location, and *.forward* files. The standard qmail distribution includes some examples in */var/qmail/boot* to set up for various degrees of sendmail compatibility, discussed in Chapter 4.

5.2.1 Mailbox Format and Location

Sendmail invariably delivers mail into mbox format mailboxes, which are usually all located in */var/mail* or */var/spool/mail*. Qmail can deliver to either mbox or Maildir files, but normally puts each user's mailbox in the user's home directory. You have several options during a conversion.

The easiest option is to leave all the mailboxes in */var/mail* or a similar shared directory. The disadvantage is that */var/mail* isn't a very good place to put mail, because mail doesn't count toward individual disk quotas, and minor protection errors on mailboxes make it possible to snoop on mail. Because qmail doesn't have a built-in delivery agent that puts mail in */var/mail*, you must tell it to use an external one such as */usr/libexec/mail.local* (4.4 BSD and descendants) or */bin/mail* (older versions of BSD, System V, and Linux).

It is not a good idea to leave mail in */var/mail* other than for testing. A reasonable compromise is to have qmail deliver to mbox files in the home directories, and leave sendmail delivering to */var/mail*. Then when you're happy with qmail, copy all the old mailboxes to the new location using scripts described later in this chapter. You must also adjust the *.profile* and *.login* files so that they set the MAILBOX environment variable to point to the new location.

Although Maildir mailboxes have many operational advantages over mboxes, switching users over on systems with shell users is painful due to the dearth of Maildir mail clients. On systems where most or all of your users pick up mail with POP or IMAP, switching to Maildirs is easier, and I recommend it. Again, set up qmail with Maildirs, then when you stop sendmail, copy the contents of the old mailboxes into the new ones, converting mbox to Maildir at that time.

Qmail comes with a pair of scripts called *pinq* and *elq*, which copy the user's incoming mail from *~/Maildir* to *~/Mailbox* and then run pine or elm. While they work fine, if a user is going to use a mail client that expects an mbox, it makes more sense to deliver to the mbox in the first place. One semiplausible reason to use *pinq* or *elq* is if the filesystem to which the mail is delivered is on a different host than the one uses to read mail, with the files mounted using NFS. Because NFS has locking problems with mboxes, it makes sense to do the deliveries into Maildirs, which work reliably. Assuming the user runs only one copy of *pinq* or *elq* at a time, it can safely copy mail from the Maildir into an mbox on his local disk, and then run pine or elm.

5.2.2 Qmail and .forward Files

Sendmail shell users frequently have *.forward* files to handle their mail deliveries. The most common uses are to forward mail to another address and feed incoming mail to procmail for filtering and sorting, but the *.forward* scheme is quite general, albeit not very well specified.

Qmail offers two migration paths for *.forward*. The format of *.qmail* files is similar enough to *.forward* files that the most common *.forward* files can be turned into *.qmail* files with little or no tweaking. If you have a small number of shell users, turn them into *.qmail* files when you convert, to get rid of *.forward* files once and for all.

5.3 System Issues

The *sendmail.cf* configuration file provides a fantastic amount of configurability to sendmail, some of which is quite useful. Most of sendmail's tricks have straightforward equivalents in qmail. It may be useful to print out *sendmail.cf* so you can check off each configuration option as you deal with it.

5.3.1 Deconstructing sendmail.cf

Much of the configuration information in a typical *sendmail.cf* needs no qmail equivalent. Since sendmail was written in an era when it wasn't clear what mail system would predominate, it can handle a wide variety of long-dead mail addressing formats, and much of *sendmail.cf* defines the syntax of email addresses, something that's built into qmail.

Nonetheless, *sendmail.cf* files usually do have some local customization that you need to translate. Because the configuration language is so arcane, most sites use a set of m4 macros to generate the file. In the following discussion, I mention primarily the m4 macros rather than the generated configuration codes.

5.3.2 Local Deliveries

Sendmail uses several macros starting with LOCAL_MAILER to define the local mail configuration. The qmail equivalent is the default delivery agent set at startup time. The sample boot scripts described in Chapter 4 cover most of the common cases.

If you want to deliver mail into mbox files in */var/spool*, use one of the binm boot scripts, whichever one calls the same mailer that sendmail is calling. If any of your users have *.forward* files, use the +df versions of the boot scripts.

If you want to deliver to mbox files in users' home directories, use the *home* or *home+df* boot script. If you want to deliver into Maildirs, start with the *home* or *home+df* script, but change *./Mailbox* to *./Maildir/*. Don't forget the trailing slash, which tells qmail that it's a Maildir. Qmail will not create Maildirs automatically, so you must create them yourself. If your user directories are all under */home*, running this script as root does the trick:

```
cd /home
for i in *
do
   maildirmake $i/Maildir
   chown -R $i $i/Maildir
done
```

If the sendmail configuration has FEATURE('local_procmail'), it's using procmail to deliver local mail. See Section 4.6 in Chapter 4 for details on setting up procmail.

5.3.3 Hostnames and Masquerading

Sendmail provides an elaborate masquerading system to rewrite addresses on mail. Historically, people used masquerading so that the syntax of mail addresses within an organization could be different from (generally simpler than) the addresses visible outside. While this made some sense when mail systems had different, incompatible, and mutually hostile addressing syntaxes, it's not a very good idea now that mail systems all use Internet-style addresses. Not surprisingly, qmail provides only minimal help for masquerading.

5.4 Converting Your Aliases File

An important but tedious part of a transition from sendmail to qmail is to convert */etc/aliases*. There are two general strategies. The first is to create a *.qmail* file in *~alias* corresponding to each entry, which works fine for small alias files but becomes unwieldy after a few dozen entries. The other is to install the fastforward package (described at the end of Chapter 4) which handles a version of */etc/aliases* pretty close to sendmail's, and then just adjust the alias entries that *fastforward* doesn't handle well.

When sendmail runs a program for a delivery from the aliases file, it uses a variety of heuristics to decide which user runs the program. Qmail's model is much simpler: all programs run from *~alias*, including *fastforward* when it does */etc/aliases* deliveries, are run as the alias user. In most cases that's fine for deliveries that don't store messages or update files. For deliveries that do store messages or update files, you may need to rewrite the delivery rules to be sure that they're run as the appropriate, user as described next.

5.4.1 Address Forwarding

The syntax for addressing forwarding is:

```
address1: address2
address1: address2, address3, address4
```

Alias entries that just forward one address to another can be left in *aliases* as is. To rewrite them as a *.qmail* file instead, create *~alias/.qmail-address1* and put address2 in it. If an address is forwarded to multiple addresses, put each one on a separate line in the *.qmail* file.

5.4.2 Mailing Lists

The syntax for mailing lists is:

```
mylist: :include: /usr/fred/listfile
owner-mylist: fred
```

fastforward's aliases emulation supports sendmail-style lists directly. The only difference is that the included file has to be compiled into a *.bin* file using *newinclude*, as described in Chapter 4.

Although included lists are most easily handled by *fastforward*, it's also possible to convert them to *.qmail* files. Copy *listfile* to *~alias/.qmail-mylist*, stripping out any address comments that aren't permitted in *.qmail* files, and create *~alias/.qmail-owner-mylist* containing the address of the list owner. Qmail provides more facilities for list management, including easy ways for users to handle their own lists. See Chapter 14.

5.4.3 Program Deliveries

The syntax for program deliveries is:

```
progaddr: "|someprogram -flags"
```

Program deliveries are supported by *fastforward*, so long as it's acceptable to run the programs as the alias user. To run programs as any other user, rewrite the delivery instructions to forward to a subaddress of the desired user. If, for example, this program should run as user fred, change the aliases entry to:

```
progaddr: fred-progaddr
```

5.5 Trusted Users

Sendmail has trusted users who can perform certain mail actions not permitted to the hoi polloi. Depending on your point of view, qmail either trusts all users or no users. Each user has full control over his own files and deliveries, but no user has any special ability to masquerade as others, run programs, or anything else. If a sendmail setup depends on trusted users (not many do), the setup must be redesigned to work with qmail.

Chapter 6. Handling Locally Generated Mail

Mail comes from two conceptual places: inside your system and outside it. In this chapter, we look at mail that originates inside your system, mail generated locally on the host where qmail is running. We also take a first look at mail injected by MUAs on computers running on the same LAN, and mail injected by "roaming" local users elsewhere on the Net, which we address in detail in the next chapter.

6.1 qmail-queue

The only way to pass a message into qmail is *qmail-queue*. All of the other relay and injection programs, for both local and remote originated mail, call *qmail-queue* to queue a message and schedule it for delivery. This design has two advantages: it's easy to write new frontends to inject mail because they only need to call *qmail-queue* to pass along the mail, and by replacing *qmail-queue* with another program that offers the same interface, you can create interestingly different systems, such as mini-qmail. (See Chapter 16 for details.) It also offers security advantages, because *qmail-queue* is one of the few set-uid programs (to qmailq, not root) in the qmail package, so it can write files in the queue directories.

qmail-queue is intended to be run from other programs, not from the command line, so it has an interface that only another program could love. It takes no command-line arguments and reads its input from two file descriptors. The first input is read from file descriptor and is the text of the message. *qmail-queue* treats the message as an uninterpreted block of bytes and doesn't change it at all, other than prefixing a Received: line at the front. The received line includes the PID, the message source, and a timestamp:

```
Received: (qmail pid invoked source); 4 Apr 2004 22:35:00 -0000
```

The source is by alias if the invoking user is the alias user; from network if the invoking user is qmaild, the daemon user that means the caller was the SMTP daemon; for bounce if the user is qmails, the qmail-send user; or by uid *NNN* otherwise.

Then *qmail-queue* reads the envelope information from file descriptor 1 in a concise binary format. (In most programs, that's the standard output, but this isn't most programs.)

```
Fsender@sender.com\0 Trcpt1@rcpt.org\0 ... Trcptn@rcpt.net\0 \0
```

First is the letter F, the sender's address, and a null byte. Then there is a list of recipient addresses, each preceded by the letter T and followed by a null byte. Finally there comes an extra null byte.

Once it has the message and the envelope, *qmail-queue* writes them in files in the queue directories and notifies *qmail-send* to process queued messages.

The only output from *qmail-queue* is the return code, which is zero if the message could be queued, and any of a long list of other values if not. (See the manpage for the list.) Because *qmail-queue* only queues a message, its return code says nothing about whether the message could be delivered, only that it could be queued for the rest of qmail to do something with it. If there are delivery problems, qmail reports them by sending bounce messages to the message's sender address.

6.1.1 Passing Input to qmail-queue

qmail-queue reads two input files from two file descriptors, and more often than not both input files are pipes from the calling program, so some care is needed to avoid deadlock. It's important to remember that *qmail-queue* reads the message from fd 0 first, then the envelope from fd 1. This isn't an implementation accident; it's part of the spec.

If you're writing programs that call *qmail-queue* and use pipes, be sure that you write the entire message first, then close the message pipe, and then write the envelope. If the structure of the program doesn't make that convenient, write the envelope information to a file in */tmp*. (You could write the message to a temporary file instead, but the envelope is usually a lot smaller than the message.)

6.2 Cleaning Up Injected Mail

Unlike some other MTAs, qmail distinguishes between injected mail, new messages entered into the mail system, and relayed mail, which is delivered from somewhere else. The difference is that injected mail needs to have its headers cleaned up, while relayed mail doesn't. Configuring qmail to clean up injected mail isn't hard, but depending on your setup, there are several possible ways to handle it.

The *new-inject* package contains two programs: *new-inject*, which is a replacement for *qmail-inject*, and *ofmipd* (Old Fashioned Mail Injection Protocol Daemon), an SMTP daemon that includes the functions of *new-inject*. Although you can survive without *new-inject*, it's easy to install and I encourage you to use it.

6.2.1 Accepting and Cleaning Up Locally Injected Mail

The usual ways to inject local mail are to feed it to qmail-inject or *sendmail*. Both do the cleanup automatically. (The qmail version of *sendmail* is a small wrapper that runs *qmail-inject*.)

Because *new-inject* is almost completely upward compatible with *qmail-inject*, use it in place of *qmail-inject*:

```
# cd /var/qmail/bin
# mv qmail-inject qmail-inject.old
# ln new-inject qmail-inject
```

(I've saved the old *qmail-inject* as *qmail-inject.old* in case there turned out to be some application that needed exactly *qmail-inject*'s features, but after a year, I have yet to need it.)

Some programs inject local mail by opening an SMTP connection to the loopback address 127.0.0.1. If you've installed an SMTP listener following the instructions in Chapter 4, injecting mail via that route already works, but without any cleanup. There are two alternatives to clean up mail injected by SMTP: adjusting the setup of the regular SMTP server to detour locally injected mail through a cleanup program or setting up a separate SMTP daemon running *ofmipd* to receive locally originating mail. I discuss both options later in the chapter.

The standard way to give a freshly created message to qmail for delivery is to use *qmail-inject* or its replacement *new-inject*. Both programs accept a message from the standard input, clean up and complete the headers without modifying the message body, construct the envelope information from the message and command arguments, and pass the result to *qmail-queue* for delivery. A combination of flags on the command line and environment variables give you some control over the header rewriting and control where it gets the envelope addresses. In the following discussion capitalized names refer to environment variables passed to *qmail-inject* or *new-inject*.

The QMAILINJECT environment variable, if it exists, contains a string of letters from the set *cfimrs* that control the header rewriting, as described later. *new-inject* also accepts the uppercase letters FIMRS with the equivalent meanings and also accepts command-line --FIMRS flags.

For testing purposes, the --n flag causes the rewritten message to be copied to standard output rather than queued. To show the envelope addresses, *new-inject* prefixes Envelope-Sender: and Envelope-Recipients: headers, while *qmail-inject* puts the sender address in a Return-Path: line but doesn't do anything with the recipient addresses.

6.2.1.1 Setting the envelope addresses

6.3 Accepting Local Mail from Other Hosts

Most networks have a small number of mail servers that handle the mail for many users who use MUAs on their individual PCs to read and send mail. Outgoing mail from these PCs is sent to the mail server using SMTP, at which point it is the mail server's job to clean up the headers and send the mail on its way.

Locally injected SMTP mail presents two problems. One is to tell which SMTP mail is injected mail from local users rather than the normal incoming mail. This is a crucial distinction, because local users can inject mail addressed anywhere, while incoming mail should be accepted only for the domains that this server handles. (Hosts that promiscuously accept and forward mail from third parties are known as "open relays" and tend to be quickly blacklisted, because the third parties are invariably spammers.) The other, simpler problem is to arrange to clean up the headers in the injected mail the way that *qmail-inject* or *new-inject* clean up locally injected mail.

6.4 Distinguishing Injected from Relayed Mail

All of these techniques involve configuring and patching the SMTP daemons. They're discussed in detail in the next chapter, but here is a short overview.

Hosts on the local network are easily recognized by their IP addresses. Each time *tcpserver* accepts a connection, it consults a rule database indexed by IP address and marks each connection as local or remote. In the common case that a network has a fixed, known set of IP addresses, and users on the network have PCs that use the qmail host to send and receive mail, this is the only setup needed.

Most networks have at least a few "roaming" users who sometimes or always connect from outside the local network. In order for the network to recognize their mail as local, the users have to provide a username and password. The most common way is SMTP AUTH, an extension to SMTP defined in 1999 that adds password authentication to SMTP. Qmail doesn't provide SMTP AUTH, but it's not hard to patch it into the SMTP daemon.

If you have old MUAs that don't handle SMTP AUTH, an older kludge called pop-before-smtp implicitly uses POP logins to authenticate SMTP. Each time a user logs in for POP (or IMAP, for systems that run an IMAP server), the system notes the IP address from which the user logged in. For an hour or so thereafter, SMTP connections from the IP address are treated as local. Users only need to check their mail before sending new mail, so MUAs need no special features to support it. Qmail doesn't support pop-before-smtp either, but add-on packages are available that fit in as "shims" that can be configured to run between the standard parts of the qmail POP and SMTP daemons. These are covered in the next chapter.

Most systems that support SMTP AUTH also support Transport Layer Security (TLS), the same cryptographic security scheme known as SSL on the Web. TLS permits authentication in both directions; the client can check the server's TLS certificate to be sure that the server is who it purports to be, and the client can also present a certificate to the server. In practice, most TLS systems use self-signed certificates that provide no authentication, but like SSL it adds extra security if the traffic passes through networks where it's subject to snooping. Patching qmail to use TLS is also straightforward, but the steps required to set up MUAs with appropriately signed certificates that can be used for authentication are a lot more difficult than setting up SMTP AUTH.

Chapter 7. Accepting Mail from Other Hosts

Unlike some other mail systems, qmail uses separate daemons for incoming and outgoing mail. Incoming mail is handled primarily by *qmail-smtpd*. As discussed at the end of the previous chapter, local mail injected from MUAs on other computers also arrives by SMTP, and it's important to distinguish the local from the incoming mail because they're handled differently.

7.1 Accepting Incoming SMTP Mail

Chapter 4 discussed the basic setup of the SMTP daemon in */service/qmail-smtpd*. The *supervise* daemon runs *tcpserver*, which listens for incoming connections, then runs *qmail-smtpd* to run the SMTP session and queue the received mail. The control file *rcpthosts* lists the domains for which it accepts mail. (If that file doesn't exist, it accepts mail for all domains and can be an open relay, which spammers see as an open invitation to abuse.)

The normal SMTP setup consults a *tcprules* file that lists the IP addresses from which to accept and deny connections. The rules file is */var/qmail/rules/smtprules.txt*, which is compiled into the binary */var/qmail/rules/smtprules.cdb* that *tcpserver* consults.

7.2 Accepting and Cleaning Up Local Mail Using the Regular SMTP Daemon

In the FAQ distributed with qmail 1.03, question 5.5 describes the classic technique for cleaning up remotely injected mail. The *smtprules.cdb* file that *tcpserver* consults contains lines that set the RELAYCLIENT environment variable for hosts allowed to inject and relay mail. If RELAYCLIENT is set, *qmail-smtpd* both skips the usual relay validation and appends the contents of RELAYCLIENT to all envelope destination addresses. If RELAYCLIENT has the value @fixme, mail addressed to fred@example.com is sent to fred@example.com@fixme. If you define fixme as a virtual domain, all mail from these hosts is handled as virtual domain mail.

More concretely, start by creating a fixme virtual domain in *virtualdomains*:

```
fixme:alias-fixup
```

Then create *~alias/.qmail-fixup-default*:

```
| bouncesaying 'Permission denied' [ "@$HOST" != "@fixme" ]
| qmail-inject -f "$SENDER" -- "$DEFAULT"
```

The first line checks that the mail is really sent to the fixme virtual domain, so that sneaky bad guys can't relay mail by sending it to alias-fixup-victim@otherdomain@example.com (assuming example.com is your local domain.) The second line feeds the mail through *qmail-inject*, preserving the original sender and remailing it to $DEFAULT, which was the original destination address before @fixme was added. Finally, add the @fixme strings to the local network entries in *smtprules.txt* and rebuild *smtprules.cdb*:

```
127.:allow,RELAYCLIENT="@fixme"
172.16.42.:allow,RELAYCLIENT="@fixme"
172.16.15.1-127:allow,RELAYCLIENT="@fixme"
:allow
```

Use *svc -h /service/qmail-send* to make qmail notice the new virtual domain.

Although as we see in the next section, this is no longer the best way to handle mail injection, the basic model for treating mail depending on its source IP address remains useful. For example, I find that I receive certain spam from AOL over and over again with very predictable strings in the message. So I route all mail from AOL to a pseudodomain aoltrap in which commands in the *.qmail* file grep each message for the known spammy strings, forward the mail to an abuse reporting script if they find any of the strings, and otherwise forward the mail to $DEFAULT to deliver it normally. While I use a more general spam filter for other incoming mail, the stuff from AOL is different enough that it was worth setting up a special filter, particularly because it only took 10 minutes to set the filter up.

7.2.1 Using Separate Relay and Injection Daemons

Since the *new-inject* package includes *ofmipd*, which combines an SMTP daemon and the same mail cleanup that *new-inject* does, the best way to clean up incoming mail is to arrange for mail clients to inject mail through *ofmipd* rather than *qmail-smtpd*. *ofmipd* doesn't do relay checking, so you have to ensure that only authorized clients can use it.

If you assign more than one IP address to your qmail host, run *qmail-smtpd* on one address and *ofmipd* on another. It's also a good idea to run a copy of *ofmipd* on port 587, the SUBMIT port that is defined (and increasingly used) for mail submission (another name for injection) from MUAs on other hosts.[1] And you must run *ofmipd* on 127.0.0.1 to accept mail from programs that inject mail by setting up a local SMTP session (such as pine and some mailing list packages). You must run separate copies of *tcpserver*, each bound to a separate IP address and port.

7.3 Dealing with Roaming Users

The most difficult part of dealing with injected mail is to recognize mail from "roaming" users not located on the local network. You can recognize them directly by requiring a user/password when they send mail or indirectly by noting their IP when they log into the POP server, then treating mail from the same IP address as local. The former is SMTP authorization, the latter is pop-before-SMTP.

Using an IP Tunnel

A different approach to the roaming user problem is to make the roaming user's computer logically part of the local network by assigning it an IP address on the local network, and arranging to "tunnel" the traffic over the Internet between the PC and the local network. Tunnels have the advantage that once they're set up, they allow access to any local-only service, such as intranet web servers.

The most popular tunnelling systems are the IETF's IP security (IPSEC) and Microsoft's point to point tunnelling protocol (PPTP). IPSEC is available on most recent Unix-like systems and on Windows 2000 and XP. It is quite tedious to set up but is very secure in use, with strong encryption on both the login and all the data that's passed through the tunnel. PPTP is built into all recent versions of Windows, and free Unix servers called poptop and pptpd are available. It's considerably easier to set up than IPSEC but is much less secure, passing data either unencrypted or at best using an encryption scheme that's known to be easy to break.

The widely used *ssh* secure remote login system provides a per-port version of tunnelling called "port forwarding." For example, users can specify that port 2025 on their remote machine is forwarded to port 25 on the mail host on the home network, then set up their mail application to use localhost:2025 for outgoing mail, with the SMTP server seeing the *ssh* host on the local network as the source of the mail. Even though it's possible to log into POP and IMAP servers directly from remote networks, it's also useful to forward remote ports to ports 110 or 143 on the mail server so that the login passwords and retrieved messages are transferred via ssh's encrypted connection rather than in the clear. *ssh* requires a shell login for authentication on the home network, and must be set up (one time) for each port that's to be forwarded. Regardless, *ssh* is often a good compromise, because it is easier to set up than IPSEC while still being reasonably secure.

7.4 SMTP Authorization and TLS Security

To use SMTP authorization with qmail, you must patch *qmail-smtpd* to handle the AUTH command for remote users to log into the server. Although AUTH lets remote users prove who they are, it doesn't provide any security against third parties snooping on the mail as it leaves whatever network the roaming users are on, nor does it provide security against port redirection, where a network connects you to their own SMTP server rather than the one you asked for. (AOL does port redirection, not for malicious purposes, but because it lets their users send out modest amounts of mail as roamers without needing to reconfigure their MUAs, while blocking blasts of spam and viruses.)

The transport-level security (TLS) extension provides an encrypted channel for SMTP sessions similar to that used by SSL secure web servers. TLS is based on certificates that include the host owner's name and address along with the hostname and an email address. Each certificate is in two parts, the private key, which needs to be kept secret, and everything else including the public key, which is not secret. For incoming SMTP sessions, SMTP clients start a secure session, verify the server's certificate and check that the hostname in the certificate matches the name of the host that the client thinks it's talking to. The client can optionally present a certificate to the server for which the server can make the same checks. The server can also use the address in the client certificate to authenticate the user instead of a separate AUTH step, as described later in Authenticating Client Hosts with TLS.

There's a combined patch for *qmail-smtpd* that adds both SMTP AUTH and TLS, and a doubly combined patch that adds SMTP AUTH and TLS, as well as the badrcptto anti-spam patch described in Chapter 9 and some extra logging (the version that I use). The two combined patches both add the same SMTP AUTH and TLS code, so they're the ones I describe here. These are the largest patches described in this book, which makes it more likely that they contain bugs. I've looked at the code and it appears OK to me, but if you're concerned about security, you should read through the patch you use yourself.

For SMTP AUTH, the setup involves setting up a login/password checking program to validate the authorization values that remote hosts present and adjusting the *tcpserver* invocation of *qmail-smtpd*. If you're using the qmail POP server, use the same password validator. Users generally only need to set an option in their MUAs to use AUTH on outgoing mail using the same userid/password pair they use for POP or IMAP.

TLS requires the openssl library (included with many but not all recent Unix-like systems) and a TLS certificate for the SMTP server. If you happen to have an SSL web server with the same name as the mail server, use the same certificate it uses. Otherwise, make a new certificate. All certificates are signed; you can sign yours yourself, but most MUAs expect server certificates to be signed by a certificate authority (CA) that vouches for the authenticity of the certificate. The MUA has a set of validation certificates from well-known CAs built-in (Outlook and Outlook Express share their list with Internet Explorer), and if the signature isn't from one of the authorities in the list, the MUA at least warns the user that the certificate isn't properly signed, and in many cases refuses to transfer any mail. There's generally some way for the MUA's user to tell the MUA to accept the self-signed certificate from the server. If you have very sophisticated users, you can set up your own miniature CA to sign your certificates and try to get your users to install your CA certificate into their MUA's well-known lists. Alternatively, you can pay one of the well-known CAs to sign your certificate, which costs between $50 and $300 depending on the CA. At this point, most TLS users are sophisticated enough to get their MUA to accept one self-signed certificate for the smarthost they use regularly, but if you plan to offer TLS to a less technical user community, your easiest course is to pay a well-known CA for a signature.

If this all sounds like more trouble than it's worth, build your patched qmail with the TLS code turned off, and worry about it later if your users ask for it.

7.5 POP-before-SMTP

An older and more indirect scheme for roaming user authentication is POP-before-SMTP, first used in 1997. It's a very simple idea and has been implemented many times. Whenever a user successfully logs in using POP or IMAP to pick up mail, it notes the IP address where the user logged in. For the next hour or so, that IP address is allowed to use the mail gateway. It has the practical advantage of working with any POP or IMAP MUA, merely by telling users to check their mail before sending. For MUAs that support SMTP AUTH, which is now most of them, AUTH is better than POP-before-SMTP because it doesn't require the extra mail check, and it identifies sent mail with a particular user, not just an IP address. But for the benefit of users who never upgrade their MUA, it's worth keeping POP-before-SMTP around.

I wrote a homebrew POP-before-SMTP system with a daemon that updates the *smtprules* files, but I now prefer Bruce Guenther's *relay-ctrl* package (http://untroubled.org/relay-ctrl/), which has the advantage of not needing any patches to existing software and working reasonably well on clusters of multiple hosts running POP, IMAP, and SMTP servers.

POP-before-SMTP has three parts. The first part observes the POP and IMAP logins and notes the IP addresses. *relay-ctrl* uses the filesystem for its database, so if a user logs in from address 10.1.2.3, it creates a file */var/spool/relay-ctrl/allow/10.1.2.3*. The second part checks the IP address on each incoming SMTP connection, and if the IP has a corresponding file in */var/spool/relay-ctrl/allow*, it sets the environment to allow relay. The third cleans up stale entries by deleting files in */var/spool/relay-ctrl/allow* that are older than the window of time allowed for POP-before-SMTP. The *relay-ctrl* documentation suggests 15 minutes, but I've used times as long as a day without trouble. To keep the relay database reasonably secure, make */var/spool/relay-ctrl* owned by root with mode 0500 so that only root can chdir into it, but make */var/spool/relay-ctrl/allow* mode 777 so that the unprivileged program that notes logins can write there.

For clusters of multiple hosts, whenever a user is authenticated on one host, *relay-ctrl* sends notices to the other hosts about the IP that authenticated, using UDP packets.

To install *relay-ctrl*, download it from http://untroubled.org/relay-ctrl/. (This description is of Version 3.1.1.) Unpack it, adjust the *conf-cc*, *conf-ld* and *conf-man* if you need to reflect your local commands for compiling and linking, and the place to put the *man* files, then make. Become super-user and run *./installer* to install the various programs. The runtime configuration of the *relay-ctrl* package is almost entirely done through environment variables. I suggest creating a directory */etc/relay-ctrl* so you can use *envdir* from the daemontools package to set the environment. (Each file in the directory is the name of a variable, the contents of the file becomes the value of the variable.) Files and environment variables to create include:

RELAY_CTRL_DIR

The directory where the relay data goes, usually */var/spool/relay-ctrl/allow*.

RELAY_CTRL_EXPIRY

The time in seconds to permit relay after a user is validated. Defaults to 900 (15 minutes), but I suggest 3600 (an hour.)

Chapter 8. Delivering and Routing Local Mail

Mail isn't very useful unless it's delivered successfully. This chapter looks at delivering mail addressed to local mailboxes, both for local delivery and for redelivery elsewhere.

8.1 Mail to Local Login Users

Local login users usually receive mail in mbox format in *~/Mailbox* and *~/.mail*.[1] Or they receive mail in Maildir format in *~/Maildir/*.

[1] For historical compatibility, some still use */var/spool/mail/username*, but in this chapter I assume that you have at least moved your users' mailboxes into their home directories where they belong.

8.1.1 Local Delivery .qmail Files and Default Delivery Rules

In the simplest case, a user's *.qmail* file needs to contain only a single line to specify the user's mailbox, either the mbox format mailbox:

```
# deliver into $HOME/Mailbox
./Mailbox
```

or the Maildir:

```
# deliver into a file in $HOME/Maildir/
./Maildir/
```

I suggest that every shell user should have a *.qmail* file (add it to the set of skeleton files that your *adduser* procedure creates), but for users who don't, be sure to set a reasonable default as the argument to *qmail-start* in */service/qmail/run*, as described in Chapter 3.

8.1.2 Maildirs and Mail Clients

Although Maildirs have all sorts of advantages over mboxes, they are not supported in many mail clients. For the popular elm and pine clients, qmail provides small scripts, *elq* and *pinq*, which move mail from the Maildir into an mbox, then run the client. These use the *maildir2mbox* utility, which requires three environment variables to be set. MAIL is the mbox file, usually *$HOME/Mailbox*. MAILTMP is the name of a temporary file used to hold a copy of the updated mbox, which must be on the same filesystem as $MAIL, usually *$HOME/Mailbox.tmp*. MAILDIR is the name of the Maildir, usually $HOME/Maildir.

While these two scripts work adequately, in the long run if you're using Maildirs, you should use a Maildir client. Unix and Linux command-line users can try mutt, a nice freeware client, Courier IMAP (see Chapter 13), and IMAP clients including pine and the KDE mail client.

8.2 Mail Sorting

Unless users receive very little mail, they generally want to sort it before they read it. While Windows mail users tend to pick up all their mail from a single POP mailbox and sort it into local mailboxes in their mail client, Unix users often arrange to sort the mail as it's delivered into mailboxes on the server, and use a client that can handle multiple mailboxes either directly or using IMAP.

There are two general strategies to mail sorting: use multiple incoming addresses or use a filtering program on incoming mail.

8.2.1 Mail Sorting with Subaddresses

The easiest way to sort mailing list mail is to subscribe to each list with a different subaddress. That is, if your address is mary@example.com, you might sign up for three lists as mary-gold@example.com, mary-nade@example.com, and mary-land@example.com.[2] If your system is set up with per-user subdomains as described in Chapter 12, the three addresses could be written as gold@mary.example.com, nade@mary.example.com, and land@mary.example.com. Then create three files *~mary/.qmail-gold*, *~mary/.qmail-nade*, *~mary/.qmail-land*, each with the delivery instructions for the list mail. If you are using a mail client that handles multiple mailboxes, either directly or through the Courier IMAP server (see Chapter 13), deliver each list to its own mailbox.

[2] These are presumably lists about horticulture, cooking, and geography.

This scheme works very well when you only receive mail from a list and you can access the signup through a web site. I use a unique address every time I buy something from a web site that wants an email address. That's useful for both mail sorting and reminding me that a dubious looking piece of mail is in fact from a place to whom I gave the address. It doesn't work so well on discussion lists to which you send as well as receive mail, because it's not easy to put the subaddress on outgoing mail, either to set up the subscription or to send messages to the list. (I've occasionally been reduced to running *qmail-inject* and typing mail headers to it.) It's possible to write a wrapper around qmail's *sendmail* program or *qmail-inject* or, if you're using the QMAILQUEUE patch from Chapter 3, write a wrapper around *qmail-queue* that looks up the destination addresses for a user's outgoing mail in a file and adjusts the return address for mail going to lists. As far as I know, though, nobody's done so. The pragmatic approach is to subscribe both a subaddress and your regular address to a list, and set your regular address to NOMAIL or alias the two together if the list management software permits, so incoming mail from the list goes to the subaddress, while you send outgoing mail from your regular address.

8.2.2 Mail Sorting with Filter Programs

For mail that's sent to a user's regular address, procmail and maildrop provide flexible script-driven mail sorting. They both provide similar sets of features, with the largest difference being one of style. The procmail control language is extremely terse with single-letter commands and options, while maildrop's language is more reminiscent of the Unix shells or Perl. Maildrop includes some extra features to do simple text processing intended mostly for extracting and handling email addresses, and an optional interface to GDBM keyed files. A significant practical difference is that procmail reads an entire message into memory, which means it won't work on very large messages that don't fit. Maildrop falls back to temporary files so it can handle even the largest messages, slowly.

I use procmail, mostly because I've been using it since before maildrop was available. The size limit isn't a problem in practice, because I rarely get mail bigger than 10 MB (certainly not mail that I want), and the filtering I do doesn't need the extra features in maildrop.

Chapter 9. Filtering and Rejecting Spam and Viruses

Filtering spam and viruses out of incoming mail is an unfortunate necessity on today's Internet. It would be easy to write a book on spam filtering techniques, but this chapter is designed to present techniques and examples rather than a complete filtering strategy. (Even if it did have a complete strategy, by the time you read it, the character of spam would have changed enough that you'd have to change your filters anyway.)

9.1 Filtering Criteria

Spam and virus filters can use any of a wide range of message characteristics for filtering. They include:

- The IP address from which the message is received
- The information sent in commands in the SMTP session, including the argument to the HELO or EHLO command, the envelope sender in MAIL FROM, and the envelope recipients in RCPT TO
- The contents of message headers, including From:, To:, Subject:, and Received:
- The contents of the message body

It's also possible and often useful to make filtering decisions based on combinations of messages, such as the number of messages received per minute from a particular IP address, or "bulkiness" scores based on the number of messages seen with similar or identical contents.

9.2 Places to Filter

Filtering can be applied at several places in the receipt and delivery process. The earlier a filter is applied, the more quickly a message is dealt with. Filtering points include:

- At connection time, for IP address and rDNS-based filters
- During the SMTP session, before the message is received, for filters based on envelope information
- During the SMTP session, after the message is received, for filters based on message contents
- As the message is delivered, for user-customizable filters

Most systems use multiple filters applied at different points. The standard qmail SMTP daemon is very lightweight compared to most other MTAs, and does as little work as possible to collect the message and queue it, leaving all of the rest of the work for delivery time. Many of the spam filtering tools, such as Spamassassin, a complex filter that computes a "spamminess" score based on multiple criteria, can run at either SMTP time or delivery time.

If you run it at SMTP time, the disadvantage is that it ties up an incoming SMTP process a lot longer than normal, possibly causing mail to be rejected if *tcpserver* reaches its concurrency limit. Also, the SMTP daemon doesn't know where the mail will be delivered, which makes it hard to apply per-user parameters. The advantages of filtering at SMTP time are that mail can be rejected before it's queued, so the bounce goes back to the actual sending system rather than a probably forged return address; a message addressed to multiple recipients can be processed once rather than separately for each user; and in case of a barrage of spam, the *tcpserver* concurrency limits prevents mail from being accepted faster than it can be delivered.[1]

[1] Hitting the concurrency limit and rejecting mail is good if the rejected mail is spam; it's not so good if the rejected mail isn't spam. But legitimate mail software will retry the delivery, so real mail will only be delayed, not lost.

I used to think that only lightweight filters, such as IP address lookups in DNS blacklists and envelope address lookups in *badmailfrom*, should be run at SMTP time, but as the ratio of spam to real mail has grown, and I see blasts of spam come in that flood the queue and can take hours to filter at delivery time, now I think that it makes sense to run anything at SMTP time that isn't user-specific and doesn't need access to data that the SMTP daemon doesn't have.

9.3 Spam Filtering and Virus Filtering

Although spam and virus filtering have historically been different applications, their implementations are as much similar as different. The most important difference is that while nobody wants to get viruses (except perhaps the abuse desk so they can figure out where they're coming from), users have varying taste in spam filters, and many filters permit some user customization. The only way to detect a virus is to examine the body of a message and see if there's a virus inside, which means it has to be done either at SMTP time after the message is received or as the message is delivered. Virus-filtering vendors have come up with long, frequently updated lists to match all of the viruses that they're aware of. While there are plenty of commercial anti-virus products available that can be plugged into qmail (see qmail-scanner and Amavis), it's possible to catch just about every virus with a simple filter (see Russ Nelson's anti-virus patch).

9.4 Connection-Time Filtering Tools

The ucspi-tcp package contains a set of tools to accept, reject, or conditionally accept mail using rules that key on the IP address or rDNS of the remote site.[2] Some rules are locally defined and used only on a single host, while others are shared among multiple hosts. The standard way to handle shared rules is via DNS blacklists or blocklists (DNSBLs, either way). Local IP and rDNS rules are handled by *tcpserver*, using a rule file created by *tcprules*. This is the same rule file we set up in Chapter 8 to distinguish between local injection hosts and remote relay hosts. DNSBLs are handled by *rblsmtpd*, which runs between *tcpserver* and *qmail-smtpd*.

[2] *tcpserver* can also use info from an IDENT (port 113) server on the remote host. IDENT has almost disappeared from the Net, so I won't say much about it beyond noting that if *tcpserver* receives IDENT data from the remote host, it's put in the TCPREMOTEINFO environment variable. See the *tcprules* documentation for more details.

What Are DNSBLs and DNSWLs?

Many people keep recommended lists of IP addresses to block that they share with others. The standard way to publish these lists is through DNS. The form of a DNSBL is very simple, a DNS zone with a name like any other DNS zone, say badguys.example.com. For each IP address in the DNSBL, there's a pair of records in the zone whose name is the components of the IP address in reverse order. One record is an A record with a value of 127.0.0.2, the other is a TXT record with a string to use as the error message when someone uses the record to block mail. For example, if the address 10.1.2.3 were in the zone, the records would be named 3.2.1.10.badguys.example.com and the text record might contain Blocked due to abuse. Some DNSBLs use the same text record for every entry; others include a URL for each address or range of addresses blocked that provides more information about the entry. Reversing the components of the IP address makes it easier to handle IP address ranges with wild cards, so that *.2.1.10.badguys.example.com would cover all of 10.1.2.0-10.1.2.255. A few DNSBLs only have A records and no TXT records. The standard *rblsmtpd* requires the TXT records, but a patch at http://www.qmail.org/ucspi-rss.diff lets it simulate TXT data for zones without it.

A variant of a DNSBL is a DNS whitelist, or DNSWL, that lists IP addresses from which you should accept mail. The structure of a DNSWL is the same as a DNSBL except that there are only A records, no TXT records. The most common use of a DNSWL is for the operator of a cluster of SMTP servers to override entries in public DNSBLs that the servers use. There are also a few public DNSWLs, such as the HUL from *habeas.com*, that lists hosts that have committed to send mail in a responsible fashion. Earlier versions of the ucspi-package had a separate *antirbl* program to handle DNSWLs, but as of Version 0.88, its function has been folded into *rblsmtpd*, which now handles both DNSBLs and DNSWLs.

9.4.1 Using Local Filtering Rules

Local filtering rules go into the file that is used to build the CDB file used by *tcpserver*. If you set up your qmail system as suggested in Chapter 4, the CDB file is called */var/qmail/rules/smtprules.cdb*, and the source from which it's built is */var/qmail/rules/smtprules.txt*.

9.5 SMTP-Time Filtering Tools

Once *qmail-smtpd* has started, filters can use the message envelope and data to trigger more filter rules. Some of the filters require patching the filter code into *qmail-queue*, while others can use the QMAILQUEUE patch to run the filters on the incoming message before queueing it for delivery.

9.5.1 Filtering in the SMTP Daemon

The three most useful checks in the daemon itself are MAIL FROM rejection, RCPT TO rejection, and Windows EXE virus rejection.

The standard qmail control file *badmailfrom* lists addresses and domains to reject as MAIL FROM arguments. The addresses are listed literally, domains preceded by @, so an address annoying@example.com is rejected if either annoying@example.com or @example.com appears. The rejection actually happens at subsequent RCPT commands because it's clearer to some SMTP clients that the mail can't be delivered.

I wrote a "badrcptto" patch, available at qmail.org, that lets you list recipient addresses to reject by putting them in *badrcptto* or *morebadcptto*, which is compiled into *morebadrcptto.cdb* by the new program *qmail-newbrt*. It only lists addresses; the way to reject recipient domains is to not put them in *rcpthosts*. The rejections happen after the DATA command to deter dictionary validation attacks. (Typical dictionary attacks start by trying a garbage address or two, in order to see whether the recipient MTA rejects them, and if they're not rejected, the attacker goes away.) The main point of badrcptto is one of efficiency. My system has a lot of addresses that get nothing but spam, and it's much faster to reject mail to those addresses at SMTP time than at delivery time. Also, if the message has multiple RCPT TO recipient addresses, it's rejected and not delivered to any of them if any of the addresses appear in *badrcptto*, on the theory that one can presume that any message sent to a known-to-be-bad address is spam even if it's also sent to a valid address. Another minor point is that rejecting at SMTP time sends the rejection to the actual sending host, rather than to the innocent return address, in the usual case that the return address is a fake.

There's a "goodrcptto" version of my patch floating around that flips the sense of the test and accepts mail only to listed addresses. I don't suggest you use it, because it breaks mail sent to subaddresses and -default addresses, some of qmail's most useful features.

The third daemon check deals with viruses. I observed in 2002 that all current viruses are Windows *.exe* files, and it's rare for anyone to send mail with an individually attached *.exe* files that's not a virus. Russ Nelson wrote a simple and extremely effective anti-virus patch, available at www.qmail.org, that recognizes the fixed code pattern present at the beginning of each *.exe* file. I suggest you use it, and tell your users who just have to mail around *.exe* files to put them in ZIP files before sending.

There are some other filtering patches for the SMTP daemon, none of which I recommend. One fairly popular one does a DNS lookup on the domain of each MAIL FROM address and rejects any that don't resolve. Several years back, a lot of spam used nonexistent fake addresses, but once the DNS checks became popular, spammers started forging genuine domains to defeat the DNS check. Nowadays, the DNS check slows mail delivery, because it can require a round-trip DNS lookup to a faraway DNS server, but stops almost no spam.

9.5.2 Separate Filters Called from the SMTP Daemon

Once *qmail-smtpd* has collected the incoming message, it normally runs *qmail-queue* to queue the message for

9.6 Delivery Time Filtering Rules

The most practical way to do delivery time filtering is to call filter programs from procmail or maildrop. (All these examples use procmail, but you can do the same things from maildrop.) Procmail is called in the context of the delivery user, so it's straightforward to use the user's personal preferences for filtering. These procmail rules, for example, call DCC and Spamassassin, both of which add X- message headers to the mail to report what they found. Tagged mail is filed in a separate mailbox, in this case a spam subfolder of Maildir where it's visible as a subfolder in Courier IMAP. The procmail rules can either go in */usr/local/etc/procmailrc*, the global file used by default, or go in an individual user's *procmailrc* for users who want to fiddle with their own rules (see Example 9-9).

Example 9-9. Filtering in procmail

```
 # filter through dcc using the user's whitelist
:0 f
| dccproc -cCMN,40 -ERw .dcc/whiteclnt

:0
* X-DCC-IECC-Metrics: .*bulk
{
         LOG="Reject: tagged by DCC
"
        :0
        ./Maildir/.spam/
}

# filter through spamassassin for messages under 300K
:0 fw
* < 300000
| spamassassin

:0
* X-Spam-Status: Yes
{
         LOG="Reject: tagged by spamassassin
"
        :0
        ./Maildir/.spam/
}
```

9.7 Combination Filtering Schemes

You can mix and match the pieces described previously to construct hybrid filtering schemes. For example, on one of my servers I have some domains that deliver into a POP/IMAP "pop toaster," and other domains that deliver to a variety of shell accounts, mailing lists, and mail forwarders. For the pop toaster domains, I want to do the filtering at SMTP time, because all of the mailboxes are handled the same, while for the other domains I want to do it at delivery time.

To arrange this, I assigned two different IP addresses to the server, and set up the DNS so that the MX records for the pop toaster domains point to the first MX and the rest point to the second MX. Then I set up two separate SMTP server setups under */service*. The one for the pop toaster runs *tcpserver* with QMAILQUEUE set to point to the filtering script, while the other one leaves QMAILQUEUE alone, so mail is queued directly. Hence mail for the pop toaster domains goes to the first MX where it's handled by the first setup, filtered and then queued for delivery, and the *.qmail* files for toaster domains just deliver the mail. The rest of the domains go to the second *tcpserver* setup where mail is not filtered at SMTP time, but the .qmail files for the various recipients run procmail to do the filtering at delivery time.

In theory, a bad guy who knew the details of this setup could deliberately misroute mail for pop toaster accounts to the second MX, thereby avoiding the spam filtering, but that's unlikely because there's no obvious connection between the two sets of domains other than that the two IP addresses are numerically close. If it became a problem, I could set up two completely separate instances of qmail with separate configurations and separate *rcpthosts* files, as described in Chapter 17.

Part II: Advanced Qmail

The last nine chapters build on the foundation in the first part. They start with detailed definitions of qmail's local and remote mail delivery system, and then cover other topics, including virtual domains, mail pickup from remote PCs, running mailing lists, system tuning, and ways to use qmail to solve complex mail handling problems:

Chapter 10. Local Mail Delivery

The way that qmail delivers local mail is fundamentally quite simple but is extremely configurable. This chapter looks in detail at the way that local mail is delivered, then looks at some common problems and applications.

10.1 How Qmail Delivers Local Mail

Every local message is delivered to the local part of its target address, the part of the address to the left of the at-sign. The local part may come directly from an incoming message, or it may be generated internally by qmail, particularly for mail to virtual domains (see Chapter 12), which construct the local part from a combination of the incoming address and information about the virtual domain.

If the local part of an address contains one or more hyphens, the part before the first hyphen is consider the user and the rest is the extension. If the local part doesn't contain a dash (hyphen), the local part is the user and there's no extension.

10.1.1 Identifying the User

The first step in a local delivery is to identify the user corresponding to the local part and retrieve several items about the user. The items are:

- Username, that is, the login name that is usually but not necessarily the same as the qmail user.
- The numeric user ID.
- The numeric group ID.
- The home directory.
- The dash character, if the local part had an extension. This is almost always an actual dash, although for maximum sendmail compatibility some people use a plus sign instead.
- The extension, usually the extension from the local part.

Qmail uses two techniques to retrieve the user information. First it checks the users database, which the qmail manager can and usually should create. (I discuss it in more detail Chapter 15.) If there is no users database or an address doesn't appear in the database, it runs *qmail-getpw* to get the information from the Unix password file. If both of those fail, it prepends alias- to the address and tries again, so that unknown addresses are treated as subaddresses of the alias user.

10.1.2 Locating the .qmail File

All local deliveries are controlled through a *.qmail* file. Once qmail has the user information corresponding to a local part, selecting the qmail file is straightforward. All *.qmail* files are located in the user's home directory.[1] If the local part has no extension, the *.qmail* file is called *.qmail*. If Fred's home directory is */home/fred*, mail for the address fred is handled by */home/fred/.qmail*. If there's an extension, it's .qmail-*extension*; for example, mail to fred-fishing would be handled by */home/fred/.qmail-fishing*. If the *.qmail* file for an address with an extension doesn't exist, qmail also looks for a *.qmail* file, replacing the extension with -default, as in */home/fred/.qmail-default*. If there are

10.2 Mailbox Deliveries

Qmail has two built-in delivery programs: one for mbox mailbox files and one for Maildir directories. In either case, the delivery is attempted under the recipient's user ID and primary group ID, so the mailbox must be writable by the user.

If a line in a *.qmail* file starts with a dot or slash and doesn't end with a slash, it's taken to be the name of an mbox mailbox file. To do the delivery, *qmail-local* opens the file for appending, creating it if it doesn't exist. It then locks the file using the flock( ) or lockf( ) system call.[2] If it can't set the lock within 30 seconds, the delivery fails temporarily. Once the file is opened and locked, *qmail-local* writes a traditional separator line, then the Return-Path: and Delivered-To: lines to provide the message envelope information, then the message, and a newline at the end. Any message line that starts with From, possibly preceded by some number of > angle brackets is quoted by preceding the line with an angle bracket. (This makes it possible to recover the original message by deleting one bracket from any such line.) It then calls fsync( ) to flush the file to disk and closes the file. The delivery fails if *qmail-local* can't create or lock the file, or if any of the writes to the file or the fsync( ) fail.

[2] Some mail systems lock mailboxes in different ways, but qmail doesn't. If flock or lockf isn't adequate for locking your mailboxes, you should switch to Maildirs, which don't need locks to work correctly.

If a line in a *.qmail* starts with a dot or slash and does end with a slash, it's taken to be the name of a Maildir directory. First, *qmail-local* forks, and the child process does the delivery. The child makes the Maildir its current directory, then creates a new file named *tmp/t.p.h* where *t* is the time in seconds since 1970 (the standard internal Unix time format), *p* is the process ID, and *h* is the hostname, so a typical name would be *tmp/1012345678.34567.mail.example.com*. It then writes the Return-Path: and Delivered-To: lines to the file, followed by the message. Unlike mailbox format files, the message is written literally and there is no need to quote lines. It then calls fsync( ) to flush the file to disk, closes the file, links the file from *tmp* to *new*, and unlinks the *tmp* file. The delivery fails if *qmail-local* can't change directory to the Maildir, create or lock the file, or if any of the writes to the file, the fsync, or the link fail. The delivery also fails temporarily if the delivery subprocess doesn't complete in 24 hours, an error I have never seen but might occur with deliveries to an unavailable NFS filesystem. Maildir deliveries do not need explicit locking because the operating system has internal locks that make system calls to create and rename files atomic.

10.3 Program Deliveries

Qmail defines a complex but well-specified environment in which to run the programs specified in *.qmail* command lines. Each command is run under the user's user ID and primary group ID, in the user's home directory, via */bin/sh -c* . The command's standard input is the message file, while the standard output and standard error are a pipe back to *qmail-lspawn*, which logs anything the command writes to its output. If the program fails (exit 100), its output is mailed back to the sender as part of the error report. The message file is guaranteed to be an actual file, so that programs can read the message, seek back to the beginning, and read it again. (This isn't very useful for individual programs, but it's quite useful for programs like *condredirect* that fork off a child program that reads and analyzes the message, then when the child is done, reprocess the message itself.)

The program's environment variables are inherited from the *qmail-start* command that originally started qmail, with quite a few added variables to help manage the delivery:

USER

The delivery username

HOME

The user's home directory

LOCAL

The local part of the recipient address

HOST

The domain part of the recipient address

RECIPIENT

The envelope recipient address, $LOCAL@$HOST

DTLINE

The Delivered-To: line, Delivered-To: $RECIPIENT\n; any newlines within the recipient address are changed to underscores

SENDER

10.4 Subaddresses

Qmail provides each user with an unlimited number of subaddresses, which are the user's address followed by a dash [4] and the address extension. Subaddresses are most useful with virtual domains, where qmail maps each address in the virtual domain to a domain-specific subaddress, but subaddresses are useful for regular users as well. Their primary use is for mail sorting. If you use a different subaddress for every mailing list to which you subscribe, you can use *.qmail* files to sort list mail into separate mailboxes or to reformat incoming mail. I also find it handy to use a unique subaddress every time I register on a web site so in case one of the site owners misuses the address, I know who to blame.

[4] It's possible to use a character other than a dash, but I ignore that option for now.

Remember that subaddressed mail must be handled by a *.qmail* file or it will bounce. Here's a handy one-liner to put in *.qmail-default*:

```
| sed "s/^Subject:/Subject: [$DEFAULT]/" | forward username
```

It puts the address extension in the Subject line of the message to make it easier to see in your mail program. (It will also have a Delivered-To: line showing the subaddress, but most mail programs don't display that.)

10.5 Special Forwarding Features for Mailing Lists

Qmail has some relatively obscure features that make it easier to use *.qmail* files to manage mailing lists. They rewrite the envelope sender on forwarded messages that are remailed to forwarding addresses in *.qmail* files, so that bounces come back to the list owner, who can do something about them, rather than to the original sender, who can't. They can also rewrite the sender address in a special form that tells *qmail-send* to create per-recipient sender addresses, known as Variable Envelope Return Paths (VERPs). The rewritten sender address is used on any forwards, and is also placed in the NEWSENDER variable for command deliveries. Although these features are mostly used by automated list management packages such as ezmlm (see Chapter 14), they can also be useful for small manually maintained lists.

If the local part of the recipient address is user-ext and there is a file *.qmail-ext-owner*, *qmail-local* changes the sender address to user-ext-owner. If there is both *.qmail-ext-owner* and *.qmail-ext-owner-default*, *qmail-local* changes the sender address to *user-ext-owner-@host-*@[ ]. This latter address will be rewritten again by *qmail-send*.

Assume as an example that you're user fred@example.com, and you have a list fred-fishing. You list all of the recipients in *.qmail-fishing*, and set the execute bit on that file to tell qmail that it's a list so all of the entries are forwards. Now any mail sent to fred-fishing@example.com is forwarded to all of the people listed in the qmail file. But what if one of the recipient addresses bounces? The bounce goes back to the original sender. To fix that problem, create a file *.qmail-fishing-owner*, which stores responses in a mailbox or forwards them to someone who can read and act on them. (A simple &fred puts them in your regular mailbox.) Now mail to the list is resent with an envelope sender of fred-fishing-owner@example.com, which will be handled by *.qmail-fishing-owner*. For manually handled lists that's probably adequate, but to finish the example, let's also create *.qmail-fishing-owner-default* and put these lines in it:

```
| echo "$DEFAULT" | sed 's/=/@/' >> badaddrs
./fishingbounces
```

Now mail to the fishing list is queued with an envelope sender of fred-fishing-owner-@example.com-@[ ]. When *qmail-send* processes each recipient address, it further translates the sender address so that a message sent to, say, margaret@domain.com is sent to the recipient host with a sender address of fred-fishing-owner-margaret=domain.com@example.com. If that message bounces, the bounce is handled by *.qmail-fishing-owner-default*. The first line in that file takes $DEFAULT, which in this case is margaret=domain.com, changes the equals sign back to an at-sign, and appends the bouncing address to *badaddrs*. Then it saves the bounce in a mailbox, in case a person wants to look at it. In a more realistic case, the address from the bounce is used to remove the bad address from the list. A complete bounce handler is also needed to analyze mail to fred-fishing-owner-@example.com (that is, with a null $DEFAULT) since mail addresses for which qmail can't even attempt a delivery bounce differently. I cover that in more detail in Section 10.7.

10.6 The Users Database

We now return to the question of how qmail figures out what user handles each local delivery. Each local address is mapped to a set of user data including:

- Username
- Numeric user ID
- Numeric group ID
- Home directory
- Character to separate parts of a subaddress, usually a dash
- Extension, used to find an appropriate *.qmail* file

Qmail provides two schemes to find the user data. The preferred scheme is to use a static lookup table known as the users file. The table is a CDB file (Dan's Constant Data Base, designed for quick lookups) in */var/qmail/users/cdb*, which is created from */var/qmail/users/assign* by *qmail-newu*. For every local delivery, *qmail-lspawn* looks up the local part of the address in that file. If there's no match or the file doesn't exist (which it doesn't unless you create it), as a fallback it calls *qmail-getpw*, which invents user data on the fly from the system password database using the *getpwnam( )* system library routine. Either way, qmail obtains an appropriate users entry for an address, which *qmail-lspawn* uses to perform the delivery.

See Chapter 15 for more details on the users database.

10.7 Bounce Handling

Sometimes a message can't be delivered to the intended address. The process of dealing with an undeliverable message is known as bouncing the message, and a message sent back to report a delivery failure is known, somewhat ambiguously, as a bounce. Sometimes a bounce message can't be delivered, leading to a double bounce and, if a double bounce can't be delivered, occasionally to a triple bounce.

Bounces can originate in two ways. A message sent to a local address can bounce either because the address doesn't exist or because a program run from a qmail file exits with code 100 to tell qmail to bounce it. (There is considerable overlap between these two causes. Many qmail systems have a global default qmail file *~alias/.qmail-default* that runs *fastforward* to look up the address in a sendmail-style */etc/aliases* file. If the address isn't in the file, *fastforward* exits with code 100, which causes a bounce. From the point of view of the sender, the two kinds of local bounces look the same.) A message sent to a remote address may have an invalid domain with no DNS information, or the server(s) that handle that domain aren't available or won't complete an SMTP delivery, or the remote server may explicitly reject the recipient address or the entire delivery using a 4xx or 5xx error code.

In each case, qmail usually generates a bounce message and mails it back to the envelope sender of the original message. If the envelope sender is null, which is the case if the bouncing message is itself a bounce message, qmail handles it as a double bounce and treats it specially, as discussed next.

10.7.1 Single Bounces

If a message delivery attempt bounces, qmail sends a bounce message back to the sender. If a single message is sent to multiple addresses, all of the bounce reports are sent back in a single message.[5] Qmail produces bounce messages in qmail-send Bounce Message Format (QSBMF) that Dan Bernstein designed as a much simpler alternative to the rather baroque Delivery Status Notices (DSNs) defined in RFCs 1892 and 1894. (Qmail does use the three-part error numbers defined in RFC 1893, though.) QSBMF is defined in detail at http://cr.yp.to/proto/qsbmf.txt. Here's a typical QMSBF bounce message:

[5] A message can have multiple addresses if it is injected locally with multiple recipients, if a *.qmail* file remails it to multiple addresses, or if the message arrives via SMTP from a system that, unlike qmail, delivers to multiple recipients in a single SMTP transaction. If a message is sent from qmail system A to multiple invalid recipients on system B, system A sends a separate copy of the message to each recipient, so system B sees all the copies as separate messages. If system B rejects invalid addresses in the SMTP transaction, as sendmail systems usually do, the rejections are all be collected by system A into a single bounce message. But if system B accepts the messages and bounces them later, as qmail does, it sends back its own separate bounce messages for each address that bounces in whatever format B's mail system produces.

```
Return-Path: <>
Received: (qmail 17296 invoked for bounce); 19 Jul 2003 11:30:58 -0400
Date: 19 Jul 2003 11:30:58 -0400
From: MAILER-DAEMON@tom.iecc.com
To: ChrissyFoster52@yahoo.com
Subject: failure notice

Hi. This is the qmail-send program at tom.iecc.com.
I'm afraid I wasn't able to deliver your message to the following addresses.
This is a permanent error; I've given up. Sorry it didn't work out.

<regan@iecc.com>:
Sorry, no mailbox here by that name. (#5.1.1)

<scarlett@iecc.com>:
Sorry, no mailbox here by that name. (#5.1.1)
```

Chapter 11. Remote Mail Delivery

Remote mail delivery is actually somewhat simpler than local mail delivery, because there's really only one way to deliver remote mail: locate a suitable host for the message and deliver the mail to that host.

11.1 Telling Local from Remote Mail

Any domain that is listed in *locals* or *virtualdomains* is local. Anything else is remote. Note in particular that whether a domain appears in *rcpthosts* or *morercpthosts* has no bearing on whether it's local or remote. (If a domain is in *rcpthosts* but isn't local, that makes this host a backup or secondary MX for the domain, which I discuss later in this chapter.)

11.2 qmail-remote

As we saw in Chapter 2, the main *qmail-send* daemon passes remote deliveries to *qmail-rspawn*, which runs *qmail-remote* to attempt each delivery. The arguments to *qmail-remote* are the remote hostname, the envelope sender, and the envelope recipients, with the message to send on its standard input. Within qmail, *qmail-remote* is always run with a single recipient, and the host is the domain part of the recipient address. Other applications can use *qmail-remote* as a simple mail sending application, with as many recipients per message as desired.

Once *qmail-remote* has attempted delivery, it writes recipient report(s) and a message report to its standard output. The format of the reports is documented in the *qmail-remote* manpage.

11.3 Locating the Remote Mail Host

qmail-remote can identify the remote host for a message in two ways. If the *smtproutes* control file has an entry for the recipient domain, that entry determines the remote host, and *qmail-remote* pretends it found a single MX record for that host with distance zero and makes a list of the IP addresses for that host. The list usually has one entry, unless the host has multiple IP addresses.

Failing that, the usual way is through DNS. First, it looks up the hostname and retrieves any MX records, randomizing the order of multiple MX records with the same distance, then finds the IP addresses for each of the MX hosts.

Once it has the list of IP addresses, DNS goes down the list, starting at the lowest distance, trying to contact each host. Once it finds a host that answers, that's the host used for the SMTP delivery. (This description is slightly oversimplified; the omitted details are covered shortly.)

11.3.1 The smtproutes File for Outbound Mail

It's sometimes useful to override MX data with explicit routes for particular domains. The *smtproutes* control file consists of a list of two- or three-field lines, with the fields separated by colons. The first field is the domain to route, the second is the name or IP address of the host to which to deliver mail for that host, and the optional third field is the port to contact on the delivery host, defaulting to port 25.

The three primary uses for *smtproutes* are to override MX data that's known to be wrong, or at least suboptimal, to route mail to private pseudo-domains, and to send outgoing mail to a smarthost. The first situation occurs if a domain has several equal-distance MX hosts, one of which accepts SMTP connections but doesn't actually accept mail on those connections. An *smtproutes* entry forces mail to a host that's working.

Within a local network, it can often be useful to have private pseudo-domains for special applications. For example, I set up a mail-to-news gateway on my news host in the pseudo-domain news so that mail addressed to comp.whatever@news is posted to the appropriate newsgroup. The news gateway isn't accessible to outside users and doesn't appear in the DNS, so I use entries in *smtproutes* on other hosts to route the news pseudo-domain to the gateway machine. (The gateway's *rcpthosts* doesn't list news, so gateway mail from outside is automatically rejected.)

If an entry in *smarthosts* has an empty domain field, that is, it starts with a colon, that entry is taken to be the default route for remote domains. This feature can be useful to send outgoing mail to a gateway host on a local network, or to an ISP's mail server for dialup or consumer broadband users.

When *qmail-remote* looks up domains in *smtproutes*, it looks for successive tails of the recipient domain; if the target domain is bad.example.com, an entry for example.com matches it, unless there's also a more specific entry for bad.example.com.

11.3.2 Secondary MX Servers for Inbound Mail

The DNS makes it possible to list multiple MX hosts for a domain. If the hosts don't have the same distance value, the ones with greater distances are known as secondary servers. The server with the smallest distance is the primary

11.4 Remote Mail Failures

Remote delivery attempts can fail in a myriad of ways. Failures fall into two general categories: temporary, which means that the delivery might work later and should be retried, or permanent, which means that the message can't be delivered at all. On a temporary failure, *qmail-send* retries the delivery later, while on a permanent failure, it immediately sends back a bounce message with whatever error report *qmail-remote* produced. Errors include:

Connected to host but greeting failed

Temporary. The remote server accepted the connection but never sent the initial SMTP greeting.

Connected to host but my name was rejected

Temporary. The remote host rejected the HELO command.

Connected to host but sender was rejected

Temporary or permanent. The remote host rejected the MAIL FROM command. If the remote code was 4xx it's a temporary error, if 5xx a permanent error.

Host does not like recipient

Temporary or permanent. The remote host rejected the RCPT TO command. If the remote code was 4xx it's a temporary error, if 5xx a permanent error.

Failed on DATA command

Temporary or permanent. The remote host rejected the DATA command. If the remote code was 4xx it's a temporary error, if 5xx a permanent error.

Failed after I sent the message

Temporary or permanent. The remote host accepted the DATA command and the text of the mail message, but returned an error code after the message was accepted. If the remote code was 4xx it's a temporary error, if 5xx a permanent error.

Qmail only delivers mail to one recipient at a time, but *qmail-remote* accepts multiple recipient arguments, and tries to deliver to multiple recipients. It returns separate status codes for each RCPT TO and sends the message if any of the recipients were accepted.

11.5 Serialmail

Qmail was designed for an environment with fairly fast network connections, where the roundtrip delay on a connection dominates the data transfer time, so it's faster to have multiple single-recipient deliveries in progress that can share the connection. In environments where this is not true, the *serialmail* add-on package delivers one message at a time. It's also useful to deliver mail to hosts via intermittent dialup connections.

To use *serialmail*, first configure qmail to deliver mail to a Maildir, then run programs from the *serialmail* package to take mail out of the Maildir and send it across the Net. You can run it on a schedule to push out mail to a slow host or on demand to send mail when a dialup host connects.

The *serialmail* package is installed the same way as other DJB software. It depends on the *tcpclient* program that is in the UCSPI package. (That's the same package that contains *tcpserver*, so you should already have it installed.) The most useful programs in the package include:

maildirserial

The driver program that selects messages from a Maildir, calls another program to try to deliver them, and deals with the results

serialsmtp

The actual SMTP client called indirectly from *maildirserial*

maildirsmtp

A shell wrapper that calls *maildirserial* to deliver the files in a directory via SMTP

setlock

Runs a program with a file locked to ensure that multiple copies of the program aren't running simultaneously

To deliver mail to a domain with *serialmail*, first define the domain as a virtual domain and deliver all its mail into a Maildir. If you want to handle the domain bad.example.com, add a line to *virtualdomains* like this:

```
bad.example.com:alias-badex
```

Then create *~alias/.qmail-badex-default*, containing *./bemaildir/*, to deliver all of the mail for that domain into *bemaildir*.

Then, when it's time to deliver the mail, run a command like this:

```
setlock bemaildir.lock \
 maildirsmtp bemaildir alias-badex- 10.2.3.4 my.example.com 2>&1 |
```

Chapter 12. Virtual Domains

In qmail-speak, a virtual domain is one handled locally but with a different set of mailboxes from the home domain. Qmail's virtual domain handling is one of its strongest features, thanks to a simple and clean design.

12.1 How Virtual Domains Work

Qmail turns addresses in a virtual domain into subaddresses of a local user, then handles the subaddressed message like any other local message. The translation from virtual to local addresses is in the control file *virtualdomains*. Assume, for example, that it contained the line:

```
myvirt.com:virtuser
```

Then mail addressed to marvin@myvirt.com is translated to virtuser-marvin, and then delivered normally. If there's a user virtuser, it checks for *~virtuser/.qmail-marvin* or *~virtuser/.qmail-default*. If there's no such user (which is often the case), the address is turned into alias-virtuser-marvin and delivered under the control of *~alias/.qmail-virtuser-marvin* or *~alias/.qmail-virtuser-default* or *~alias/.qmail-default*.

When qmail translates the mailbox part of a virtual domain address, it does not change the domain. That is, in the previous example, marvin@myvirt.com is translated to virtuser-marvin@myvirt.com. This seems like a mistake, because this is not Marvin's address, until you realize that the translated address is used only within qmail. The virtual domain remains with the address throughout the delivery process, so delivery programs can check $HOST or $RECIPIENT to tell whether a message was addressed to a virtual domain or to the (nearly) equivalent local address. Later in this chapter, *fastforward* makes good use of this ability.

Don't Forget the DNS

If you want the outside world to be able to send mail to your virtual domains, they all need MX records in the DNS. If your local tests all work fine, but people elsewhere can't send you mail, DNS misconfiguration is a likely problem. If people can send you mail but your server rejects it, you forgot to put the domains in *rcpthosts* or *morercpthosts*.

12.2 Some Common Virtual Domain Setups

Although qmail's virtual domain mechanism is extremely flexible, most of its applications fall into a few common models.

In each case, you must pick a local user or subaddress to assign the virtual domain to. It can be a real user in */etc/passwd*, or if you use qmail's users mechanism (see Chapter 15), a qmail-only user. If you handle several virtual domains in the same way, all the domains can share a user, with delivery programs distinguishing among them by checking $HOST or $RECIPIENT. If you're only forwarding mail, you can handle virtual domains under *~alias*; otherwise, it's a good idea to set up separate user IDs per virtual domain or per kind of virtual domain so that the programs you run can only smash that user's files when they break. Also, if you want to delegate the management of a virtual domain to someone else, make a separate Unix user account for the domains so the manager can log in as that user and change the domain's mail setup.

If you want people outside your network to be able to send mail to the virtual domain, you must list the domain in *rcpthosts*. If you're using a virtual domain as a service gateway for your own users to a fax server or the like, don't put it in *rcpthosts*.

Finally, each time you change the contents of *virtualdomains* or *locals*, you must tell qmail to reread them by sending a hang-up signal. Assuming you're running qmail under daemontools, use:

```
# svc -h /service/qmail
```

12.2.1 Mapping a Few Addresses with .qmail Files

In the simplest case, you can just create a *.qmail* file per address. Assume you have the domain *myvirt.com*, with addresses *william*, *wilbur*, and *wilfred*, which you want to forward to local addresses *biff*, *buddy*, and *butch*, respectively. This example doesn't do any special processing on the mail, so just set it up as a subuser of alias. Add this to *virtualdomains*:

```
myvirt.com:alias-myvirt
```

(Don't forget to signal qmail to reread the configuration.) Now you need to create only three *.qmail* files in *~alias* and you're done:

```
.qmail-myvirt-william    &biff
.qmail-myvirt-wilbur     &buddy
.qmail-myvirt-wilfred    &butch
```

That's it. Mail to the three addresses is now forwarded to the three mailboxes.

In a realistic example, you'll probably want to define a few more standard addresses such as postmaster and abuse. Either you can create individual control files like *.qmail-myvirt-abuse*, or you can make a catchall file to collect mail to all other addresses in the domain, *.qmail-myvirt-default*. A catchall file catches mail to misspelled versions of the three explicit addresses as well as to other administrative addresses such as webmaster, hostmaster, and support. Unfortunately, the majority of mail to other addresses is likely to be spam. If you have a spam filter, aim the default file at a filtering program and deliver whatever survives the process, or do nothing about the process and bounce the mail. If you have no *.qmail-myvirt-default* in *~abuse*, but there is a global catchall *.qmail-default*, the global catchall will handle all of the misaddressed myvirt mail, which you do not want. To bounce any misaddressed mail, put something like this in *.qmail-myvirt-default*:

```
| bouncesaying "Not a valid address at myvirt.com."
```

12.2.2 Mapping Many Addresses with fastforward

12.3 Some Virtual Domain Details

Finally, here are a few virtual domain odds and ends.

12.3.1 qmail-foo Versus qmail-alias-foo

After qmail rewrites a virtual domain address into a local address, the local address is then handled just like any other address. In particular, if there's no match for the rewritten address, it's handled by *~alias*. This means that if there's no local user *myvirt*, these two lines are equivalent:

```
myvirt.com:myvirt
myvirt.com:alias-myvirt
```

Use the latter version, to make it explicit that you're not expecting the user to exist. That way if someone later creates a user *myvirt*, mail to that virtual domain won't mysteriously start failing.[2]

[2] Guess how I learned about this trick.

12.3.2 Local-Only Domains

If you provide a service gateway, such as mail-to-fax or mail-to-news, you'll probably want to let users on the local network use it, but not outsiders. To ensure that, create a subdomain for the gateway, (e.g., fax.example.com), but don't put the domain in the DNS. (If you have split-horizon DNS, with internal hosts seeing different data than external hosts, it's OK to put the gateway domain in the DNS visible to local hosts.) Also be sure not to put the gateway domain in *rcpthosts*, so that the only people who can send mail to the gateway domain are local users and authorized SMTP users who can send to nonlocal domains. Finally, in the gateway delivery program, check that the mail was sent to the virtual domain, not to the equivalent local address. If you do individual deliveries, that's easily handled in the *.qmail* file:

```
| case "$HOST" in fax|fax.example.com) exit 0 ;; *) bouncesaying "Not authorized." ;;
esac
| gatewayprogram "$DEFAULT"
```

If you do batched delivery via a Maildir, this trick also works, because qmail treats a Maildir delivery as a program delivery using an internal program:

```
| case "$HOST" in fax|fax.example.com) exit 0 ;; *) bouncesaying "Not authorized." ;;
esac ./faxmaildir/
```

In either case, the delivery program can check the domain itself, by checking $HOST in individual deliveries or by checking the domain in the Delivered-To: line in batched gateways, but it's usually easier to check in the *.qmail* file so the gateway doesn't have to be coded to know what domain it's handling.

An alternate approach is to make all addresses virtual. That is, create a virtual domain for all the local mailboxes, and put something like *localdomain* in *locals* but not *rcpthosts* for miscellaneous, locally generated mail. If you have many local users, this approach is painful because you have to map all the users' mail from the virtual domain into their mailboxes, but it's not a bad idea on systems that are supposed to be POP toasters or gateways without local shell users.

Chapter 13. POP and IMAP Servers and POP Toasters

If you want to access your mailbox across a network using mail user agents (MUAs) such as Eudora, Microsoft Exchange, Pegasus, mutt and fetchmail, you must run the qmail POP server. The qmail POP server allows these clients to read and delete mail from their mailbox, but doesn't include a method for sending email; use *qmail-smtpd* or *ofmipd* for that.

Consistent with qmail's component design, the qmail POP server is actually three separate programs that cooperate to create the POP service. (Traditional POP servers such as *qpopper* are typically implemented as a single large program.)

The qmail POP server only handles Maildirs, not mbox mailboxes or anything else. If you are installing qmail on an existing mail system, you must convert any existing mailboxes to Maildir if you want to use the qmail POP service. (If you want to keep using mboxes, you can use the popular *qpopper* POP server, which is not covered here.)

POP Mail Versus Local Mail Clients

If you want to be able to read your mail both with a local mail client running on your mail host and with POP, you have a few options, described in Section 4.8 in Chapter 4. If you're using Maildirs, your best bet is to use either the freeware mutt MUA or Courier IMAP, described later in this chapter, and an IMAP client such as pine. Or you can deliver to mboxes, using *qpopper* for POP and any of the many MUAs that handle mboxes.

13.1 Each Program Does One Thing

The qmail POP server consists of a set of three cooperating programs (or four if you include the copy of *tcpserver* that runs the rest of the server):

qmail-popup

Read the username and password from the network connection

checkpassword

Validate the username and password

qmail-pop3d

Handle requests to read and delete mail from the user's Maildir

13.1.1 The Flow of Control

In detail, a POP session proceeds as follows:

- *tcpserver* listens for network connections on the POP3 port 110 and spawns *qmail-popup*.
- *qmail-popup* inherits the environment variables and the socket created by *tcpserver*. (*qmail-popup* actually has no idea that it is connected to a socket; it merely reads from stdin and writes to stdout—knowing this comes in handy when we want to test the POP Server.) *qmail-popup* performs one very simple function. It understands just enough of the POP protocol to read the username and password sent across the network. Once this data is read, *qmail-popup* spawns *checkpassword*, passing it the username and password. *qmail-popup* has now completed its part in this session.
- *checkpassword* checks the username and password against the password file. (It uses *getpwnam( )* which usually reads */etc/passwd*, but this detail varies considerably from one version of Unix to the next.) If the password is correct, *checkpassword* extracts information about that user from the password file, does enough of a login process to permit *qmail-pop3d* to do its work, and now spawns *qmail-pop3d*.
- *qmail-pop3d* handles the rest of the POP3 session with the client. When *qmail-pop3d* exits, the POP session is completed.

13.1.2 Functional Partitioning

Using four programs to establish each POP session might seem like a lot of needless work. But each program is small

13.2 Starting the Pop Server

Setting up the POP server is similar to setting up the qmail SMTP server.

13.2.1 Prerequisite Packages

The POP server depends on the daemontools and ucspi-tcp packages. If you've set up qmail as described in Chapter 3, these tools are already installed and available. You also need a checkpassword package. If you want to use the same passwords that you use for shell logins, the standard checkpassword package at http://cr.yp.to/checkpwd.html will do the trick. The checkpassword section of www.qmail.org has a long list of other versions to handle virtual domains, retrieve passwords from databases, support multiple mailboxes per user, and other options. The discussion here presumes that you're using the standard version, but the component design of the POP server means that you can substitute your own version without changing the rest of the setup.

13.2.2 Directories

Two directories need to be created: one that contains the scripts and data files used to run the POP server, and the directory that will contain the log files. (You can put these directories anywhere you want, but the following names are chosen to match the names used in the widely used "Life With Qmail" setup.)

As root, create the script and data file directories, and the log directory (see Example 13-1).

Example 13-1. Creating the POP server directories

```
# mkdir /var/qmail/supervise/qmail/pop3d
# mkdir /var/qmail/supervise/qmail/pop3d/log
# chmod u=rwx,go= /var/qmail/supervise/qmail/pop3d

# mkdir /var/qmail/supervise/qmail/pop3d/log
# mkdir /var/qmail/supervise/qmail/pop3d/log/main
# chown qmaill /var/qmail/supervise/qmail/pop3d/log/main
# chmod u=rwx,go= /var/qmail/supervise/qmail/pop3d/log
```

13.2.3 The Listening Script

Example 13-2 has been purposely written to be as flexible as possible and will work for most situations. It goes into */var/qmail/supervise/qmail-pop3d/run*.

Example 13-2. The listening script

```
 1. #!/bin/sh
 2. limit datasize 2m
 3. exec                                          \
 4.  tcpserver                                    \
 5.   -HRv -l pop.example.com                     \
 6.   -x /var/qmail/supervise/qmail-pop3d/rules.cdb \
 7.   0 110                                       \
 8.   /var/qmail/bin/qmail-popup pop.example.com \
 9.   checkpassword                               \
10.   /var/qmail/bin/qmail-pop3d Maildir 2>&1
```

Once created, the script needs to be made executable with:

```
# chmod +x /var/qmail/supervise/qmail/pop3d/run
```

The beginning of this script should be familiar from the SMTP daemon setup, from the exec on line 3 to the rules file

13.3 Testing Your POP Server

The easiest way to test the POP server is to connect to it with your favorite MUA. Can you retrieve mail? If so, congratulations.

If the POP server doesn't work, check the log file */service/qmail-pop3d/log/main/current* if that file exists. If it doesn't exist, *multilog* isn't running, probably due to a protection error or typo in the *log/run* file, so do a *ps* and look for clues in the *readproctitle* line. If the log file exists, it may contain a diagnostic message that identifies the problem. If this doesn't work, check each installation step to diagnose the problem. There are two major categories of problems. Either you can connect to the POP server and then "something" goes wrong or you cannot connect to the POP server at all.

13.3.1 You Cannot Connect to the POP Server

If you cannot connect to the POP server at all but the other parts of qmail are running, it is likely that there's a typo or protection error in the *run* file.

As root run:

```
# svstat /service/qmail-pop3d /service/qmail-pop3d/log
```

You should see something like this:

```
/service/qmail-pop3d: up (pid 37197) 5021 seconds
/service/qmail-pop3d/log: up (pid 37198) 5022 seconds
```

showing "up" as the status for both. If not, check the permissions and contents of the failing *run* file.

13.3.2 You Can Connect, but Then Something Fails

This is actually a good sign as it means that the *supervise* processes are running and the run scripts are at least partially correct. There are two primary reasons for a connection starting and then failing; a good way to find out the precise nature of the problem is to use *telnet* to manually step through the POP session to see exactly what happens.

First connect to the POP server with *telnet* like this:

```
telnet localhost 110
```

(If the POP server is running on a particular IP address or different port, *telnet* to the appropriate place.) After a few seconds you should see a banner from the POP server, something like this:

```
Connected to example.com.
Escape character is '^]'.
+OK <54559.982199402@example.com>
```

If you don't get the "+OK" line, then check the run script for typos. Most likely the *qmail-popup* line is wrong in some way. If that looks right check that the *tcprules* (*/service/popd/rules.cdb*) has been created with the correct entries.

If you receive the +OK line, *tcpserver* has successfully started *qmail-popup*. The next step is to try and log in by entering the login and password like this:

```
USER yourlogin
```

13.4 Building POP Toasters

A POP toaster is a system that provides POP mail service for a potentially large set of mailboxes. Rather than create each mailbox as a Unix user account, a POP toaster generally runs as a single user, puts all of the mailboxes in virtual domains, keeps its own database of usernames, and arranges for mail deliveries and POP/IMAP sessions to use that database for validation.

The widely used vpopmail package (previously called vchkpw) is available from http://www.inter7.com/vpopmail.html . It provides all of the POP toaster functions, along with some nice additions, such as POP-before-SMTP relay validation for roaming users, database interfaces so the user information can be kept in a MySQL, Pgsql, or Oracle database, and a design that makes it straightforward to create clustered mail servers for added performance or reliability. At the time of this writing, the current version of vpopmail is 5.3.16.

13.4.1 Installing Vpopmail

Vpopmail uses the conventional autoconf configuration scheme. Download it from http://www.inter7.com/vpopmail.html, and unpack it into a directory. Don't try to build it yet; you must create the vpopmail user ID first. It depends on ucspi-tcp (the package that contains *tcpserver*) so be sure you've installed that already, as described in Chapter 3.

All of vpopmail's mailboxes and control files belong to the same Unix user. The usual user and group IDs are vpopmail and vcheckpw. If you can, create them with numeric user and group IDs of 89. (Some versions of FreeBSD may already have them defined as 89.) If your vpopmail setup expands to multiple machines, you want to have the same numeric IDs on all of them, and 89 is as good a number as any. Be sure that the disk partition on which you create the vpopmail home directory has sufficient space for all of the mail directories you plan to create. In some cases, you can move directories around later and use symlinks to splice the subtrees together, but you might as well allocate enough space in the first place and avoid the trouble.

If you plan to have many thousands of mailboxes, you should put them on a separate partition. Since Maildirs put each message in a separate file, the average file size on a mail partition is smaller than on general purpose partitions, so you must build the partition with extra inodes. You can estimate that average messages are about 5K, so divide the size of the partition by 5K and allocate that many inodes.

In the following examples, I use /var/vpopmail as the home directory for the mailboxes. It doesn't matter for vpopmail's configuration whether it's a partition mount point or not.

Before you configure vpopmail, you have to make a few decisions:

- Do you want to handle mailboxes not in virtual domains? If your system has shell users that get their mail in Maildirs in their home directories, yes. Otherwise, if your system is just a POP toaster or the shell users don't pick up their mail remotely, no.
- Do you want to allow roaming users to send mail through your SMTP server? Usually yes.
-

13.5 Picking Up Mail with IMAP and Web Mail

Although POP is by far the most common way for users to collect their mail, many mail servers also offer IMAP and web mail. IMAP is conceptually similar to POP except that the client program has a full set of tools to manipulate the mailbox on the server. The advantage of IMAP over POP is that the mailbox remains on the server, so the user can use different mail programs from different locations, seeing consistent mailbox contents at all times. While qmail provides no IMAP server of its own, the IMAP server from the Courier mail package uses Maildirs as its mailbox format and works well with qmail. I describe its setup below.

Web mail provides access to a mailbox using a web browser as the mail client. Many web mail packages are available on the Net that use POP or IMAP to access the user mailboxes. They're not specific to qmail, so I don't describe them here. The Courier package includes a web mail component called SqWebMail that uses Maildirs as its mailbox format. I describe its installation later in this chapter.

Maildirs don't have to be locked while a client reads or updates them, so the POP and IMAP servers and SqWebMail can access the same mailbox simultaneously without trouble. Systems based on mboxes can't do that. I routinely have my mailbox open in pine on a BSD system, Pegasus and Opera 7 on a Windows laptop, and SqWebMail on a web browser, all at the same time without any trouble. You can download the sources for Courier IMAP and SqWebMail by following the links from http://www.courier-mta.org/download.php. As of the time this book was written, the current version of Courier IMAP is 1.7.0 and of SqWebMail is 3.5.0.

Courier IMAP and SqWebMail share the same user validation scheme, an "auth" framework that calls out to a variety of authorization modules to handle everything from passwd files to vpopmail to MySQL (different from the vpopmail flavor) to LDAP. Once you have it set up for one, it's easy to transfer to the other. Courier IMAP includes a POP server that provides no more function than the qmail one but uses the Courier authentication scheme, letting your POP and IMAP login rules be consistent.

Unpacking bz2 Files

The various parts of the Courier package are distributed as tar files compressed with the bzip2 data compressor. While bzip2 is a fine compression package, it's probably not one for which you have the decompressor installed, which is called *bzcat*.

Fortunately, bzip2 is easy to install. At http://sources.redhat.com/bzip2/, you can find executable versions for Linux and a few other Unix variants, and the source code (in *tar.gz* format.) Download it, compile it, and install it. By default it installs itself into */usr/bin*, so if you'd rather have it in */usr/local/bin*, make PREFIX=/usr/local install does the trick.

13.5.1 Courier's Extended Maildir++

All of the pieces of the Courier package support an upwardly compatible extended version of Maildirs known as Maildir++. The extensions allow subfolders within a Maildir and provide a convention for folders that can be shared

Chapter 14. Mailing Lists

The original impetus for writing qmail was to send out list mail faster than existing MTAs, so it's not surprising that qmail has excellent built-in support for mailing lists. The first part of this chapter looks at its list handling support, which is quite adequate for small and medium-sized lists. Then it covers ezmlm, the automated mailing list package designed to work with qmail, and other qmail-compatible list management software.

14.1 Sending Mail to Lists

The easiest and most common way to handle a small list is to put the list in a *.qmail* file. To reiterate an example from Chapter 10, assume a user's name is fred, and the list is about fishing. Then the list goes into *~fred/.qmail-fishing*, one address per line like any other *.qmail* file (see Example 14-1).

Example 14-1. Fred's fishing list

```
fred@example.com
jim@example.org
mary@myvirt.com
&/fn=hunt/ln=dash/@bigcorp.com
```

Note that the third address, an X.509 address that contains slashes, is preceded by an ampersand to keep it from being interpreted as a filename. Also, Fred's address is in the list so he gets copies of messages sent to it. To send mail to this list, one needs only to send a message to *fred-fishing*, and it's redistributed to all of the list members.

14.1.1 Maintaining List Files

Qmail provides a small but useful set of functions to maintain list files. To edit a file safely, set the otherwise unused "sticky" bit in the user's home directory, edit the file, then unset the sticky bit:

```
$ cd
$ chmod +t .
$ emacs .qmail-fishing
$ chmod -r .
```

Should any mail arrive for addresses handled by a *.qmail* file in the directory while the sticky bit is set, *qmail-local* notices the sticky bit and exits with code 111 so the delivery is retried later.

This list file example highlights a possible security hole when an address looks like a filename.[1] There are three ways to solve the problem. The simplest, but most error prone, is to put an ampersand in front of each name, or at least in front of each name that might look like a filename or command. The second is to set the owner execute bit on the file, which tells *qmail-local* that the file should only contain forwarding addresses, so any file or program deliveries fail. The third (undocumented) is to put a line containing +list in the file, which tells *qmail-local* that subsequent lines have to be forward addresses. This permits a few setup lines at the beginning before the addresses. For example, to require that each message's subject line has a keyword, see Example 14-2.

[1] This isn't a new problem; in some ancient versions of Unix you could send mail to /etc/passwd and it'd add your message to the end of the password file.

Example 14-2. Fred's fishing list with subject checking

```
| egrep -qi "^Subject:.*(largemouth|smallmouth|squid)" || bouncesaying "Not fishy
enough."
+list
fred@example.com
jim@example.org
mary@myvirt.com
&/fn=hunt/ln=dash/@bigcorp.com
```

In the examples so far, the list has an address that is a subaddress of a user address. List files can equally well live in *~alias* in which case they have regular addresses; the list file *~alias/.qmail-fishing* has the address *fishing*.

14.1.2 Bounce Handling and VERP

14.2 Using Ezmlm with qmail

The most popular list manager used with qmail is ezmlm-idx, an extended version of Dan Bernstein's ezmlm. The original ezmlm has a very solid core of mailing list functions: subscription and unsubscription, message distribution and bounce management, and simple message archiving and retrieval. Unlike most list managers, ezmlm lets individual users run automatically managed private lists using subaddresses of their user addresses, as well as the more conventional arrangement where the system manager sets up a list with an address of its own. Ezmlm-idx adds more complex features such as digests, moderated lists, remote list management, and distributed lists with sublists. Nearly all ezmlm users use ezmlm-idx, because the basic ezmlm lacks now-essential abilities such as letting only list members post to a list. The following discussion all applies to ezmlm-index.

14.2.1 Installing Ezmlm-idx

Ezmlm-idx is a little bit tricky to install, because you have to combine the original ezmlm with the additions and patches for ezmlm-idx yourself. The easiest place to find the ezmlm and ezmlm-idx tarballs is http://www.ezmlm.org, where you can click the Download link near the top of the page to find a nearby archive with *ezmlm-0.53.tar.gz* and *ezmlm-idx-0.40.tar.gz* (or a newer version if available). While you're there, if you plan to run large lists (tens of thousands of addresses), you might also want to patch qmail, as discussed in Chapter 16, to increase the number of parallel deliveries above 255.

To install ezmlm-idx once you've installed the two archives:

1. Unzip and untar *ezmlm-0.53.tar.gz* into a directory in any convenient place, creating a subdirectory *ezmlm-0.53* containing the ezmlm files.
2. Unzip and untar *ezmlm-idx-0.40.tar.gz* in the same directory, creating an adjacent subdirectory *ezmlm-idx-0.40*.
3. Move all of the ezmlm-idx files, which include both new files and patching instructions for existing files, into the *ezmlm* directory:
   ```
   $ mv ezmlm-idx-0.40/* ezmlm-0.53
   ```
4. Go into the *ezmlm* directory and apply the patches. The *patch* program should report that all of the patches succeeded. (If not, either you have an obsolete patch program and need to install the current GNU version, or the files in the ezmlm-0.53 directory were already modified, so delete the directory and recreate it from the *.gz* file.)
   ```
   $ cd ezmlm-0.53
$ patch <idx.patch
Hmm...  Looks like a unified diff to me...
The text leading up to this was:
--------------------------
|--- ezmlm-warn.1       1998/02/17 00:32:45     1.1
|+++ ezmlm-warn.1       1998/12/21 04:35:16     1.5
--------------------------
Patching file ezmlm-warn.1 using Plan A...
Hunk #1 succeeded at 3.
Hunk #2 succeeded at 21.
 ... more patch reports ...
--------------------------
```

14.3 Using Other List Managers with Qmail

Although ezmlm is the list manager most often used with qmail, any list manager that's written to work with sendmail can easily be adapted to work with qmail. The most popular freeware packages are GNU Mailman, which has qmail config advice in *README.QMAIL*, and the Majordomo2 list manager, which has qmail support for lists in virtual domains built-in.[5]

[5] Majordomo1 is obsolete, and anyone thinking of using it should use majordomo2 instead. The commercial packages such as LISTSERV and Lyris include their own SMTP engine so they can run in parallel with qmail on a different virtual IP address, but they don't connect to qmail, or any other local MTA, directly.

14.3.1 Incoming Mail to List Managers

Mail sent to a list manager includes both the messages for the lists and the administrative mail to *--request* addresses and the like. Systems with a small number of lists usually put sendmail alias entries for all of the entries into */etc/aliases*. That also works with qmail, but can get unwieldy as the number of lists grows and if there are collisions between list names and usernames. Systems with lots of lists usually put the lists into virtual domains. Sendmail handles virtual domains differently from qmail, so the setup for qmail has to be a little different. List manager software is usually set-uid because it would difficult to control the UID for programs run from sendmail's */etc/aliases*. With qmail, the virtual domain(s) for the list manager should belong to the list manager user, removing the need for set-uid except perhaps on CGI scripts for web interfaces. The individual list and administrative addresses can each be a *.qmail* file, or it might be easier to put them all in one file and use *fastforward* as described in Chapter 12.

14.3.2 Outgoing Mail from List Managers

List managers can hand mail to the MTA in two ways, by calling sendmail or SMTP. Using sendmail makes sense for administrative mail sent to a single recipient. It's a problem for list mail because the operating systems set a maximum total argument size in a call to sendmail or any other program, typically 64 K characters, which would limit lists to under 4,000 names. To get around this limit, the list manager can break the list up into sections and call sendmail multiple times or, more often, open an SMTP session to localhost, which permits an unlimited number of RCPT TO recipients. Either of these techniques works with qmail, although of course calling *qmail-queue* directly works better if a list manager has code to support it.

Some list managers can sort recipient addresses by domain and pass all the addresses in a domain together. This speeds up sendmail, which does domain or MX sorting internally, but doesn't help qmail. In fact, it can lead to somewhat unfortunate behavior; if qmail processes a message with a hundred recipients all in the same domain, it will open a large number of SMTP connections to that domain's mail server, which system managers misinterpret as an attempt to overload their system. If you can prevent your list manager from sorting its addresses by domain, do so.

14.4 Sending Bulk Mail That's Not All the Same

Qmail does a magnificent job of sending identical copies of a single message to thousands of recipients.[6] It does a considerably less magnificent job of sending thousands of messages all of which are different each to a single recipient. The overhead of passing a message to *qmail-queue*, storing it in the *queue/todo* directory, then moving it into the delivery queue, is substantial. When large numbers of messages arrive quickly, *qmail-send* can fall behind to the point where it's so busy dealing with injected mail it doesn't schedule deliveries as fast as it should, the so-called "silly qmail syndrome." The big-todo patch discussed in Chapter 16 helps somewhat, but the fastest way to deliver lots of unique messages is to avoid asking qmail to deal with them in the first place. To test this theory out, I wrote a small Perl module Qspam, available as http://www.iecc.com/Qspam.pm, to send lots of unique messages fast.

[6] Identical except for the VERP envelope, of course.

The program sending the mail starts by calling qspam_start(N, &donefunc), where N is the number of deliveries to handle at once (analogous to *concurrencyremote*) followed by a callback routine that's called each time a delivery attempt finishes. To send a message, the program calls qspam_send("to", "from", mfile, code), where to and from are the envelope addresses, mfile is the name of a file containing the entire message to send, headers and body, and code is an optional code string that identifies the message. When the delivery is done, it calls the callback as donefunc(mfile, code, resultflag, resultmsg) where mfile and code are from *qspam_send*, and resultflag is "y" if the message was delivered, "n" if the delivery failed (in which case resultmsg is the error message), or a null string if the delivery was deferred until later. At the end of the program, *qspam_flush( )* waits for all of the delivery attempts to complete.

How does this all work? Qspam_send forks and calls *qmail-remote* to deliver the message. The module keeps a table of all of the deliveries in progress and won't start more than the delivery limit at once. When an instance of *qmail-remote* completes, if it either delivered the mail or got a permanent error, the delivery is done. If there was a temporary error, Qspam forks again to call *qmail-queue* to use the standard qmail delivery scheme, which always succeeds (from Qspam's point of view). Because *qmail-remote* can't deliver local mail, qspam_send checks the delivery address of each message against *locals* and *virtualdomains* to see if an address is local, and if so calls *qmail-queue* immediately. In practice, most remote delivery attempts succeed or fail on the first try, so only a small fraction of the messages need to be queued. Some mail is accepted by the remote MTA only to be bounced back later, and qmail returns its usual bounce messages if a queued delivery eventually fails, so the application needs to use envelope return addresses that can be handled by a companion bounce processor, just like list mail sent directly through qmail.

Although Qspam wasn't written for maximum efficiency (it opens and closes temporary files rather than using pipes), it's pretty fast. On a modestly sized PC sending lightly customized mail to a list of several thousand users selected from a MySQL database, it has no trouble keeping 100 simultaneous deliveries going at once. The entire application is written in Perl, but it spends nearly all of its time waiting for *qmail-remote* processes to finish so there's little reason to rewrite it to be faster. This approach, try one delivery attempt before queueing, has proven to be a simple but effective way to handle customized list mail.

Chapter 15. The Users Database

In Chapter 10, we saw that local deliveries all look up the mailbox in qmail's users database to determine both where to deliver a message, and what user and group ID and home directory to use when making the delivery. Although the most common setup of users is to deliver to the users in */etc/passwd*, the users database is considerably more flexible than a mere mirror of the password file.

The users database maps each local address to a set of user data including:

- Username
- Numeric user ID
- Numeric group ID
- Home directory
- Character to separate parts of a subaddress, usually a dash
- Extension, used to find an appropriate qmail file

The *qmail-lspawn* program changes to the user and group ID and home directory before starting a delivery, then uses the separator character and extension to locate a *.qmail* file to control the delivery, as covered in Chapter 10.

15.1 If There's No Users Database

If you don't create a users database, qmail calls *qmail-getpw*, which implements a default mapping from login users to qmail users for each local delivery. It takes the local part, passed as its argument, and looks that up using the standard *getpwnam( )* routine. If the user exists and meets some safety criteria (discussed in a moment), it returns user information for the user, uid, gid, and home directory from the password file, and null dash and extension. If the name is of the form *user-extension* and the username exists, it returns the user information with the dash being a literal dash and the extension the part of the local part after the dash. If the user doesn't exist, it falls back to the default user *alias* with the dash being a dash and the extension being the entire local part, so in that case the delivery is controlled by *~alias/.qmail- localpart*.[1]

[1] You can replace the dash with another character by adjusting the contents of *conf-break* at the time you build qmail.

To avoid security problems, *qmail-getpw* only returns user information if a user account has a nonzero uid (isn't the super-user), and the account's home directory exists, is readable, and belongs to the user. It also ignores any account with capital letters in the name or with a name more than 32 characters long.

Do You Need a Users Database?

Experienced qmail users have widely varying opinions about whether to create a users database. I've always used one, but my system has only a handful of shell users and (mostly for historical reasons) many mail-only users with addresses in the same domain as the shell users. A more typical system either has a lot of shell users, nearly all of whom receive mail, or runs a system where all the addresses are in virtual domains controlled by a few dedicated user IDs. If the list of users in your passwd file is nearly the same as the list of addresses that should get mail, you may be happier with no users database so users can get mail as soon as they're added to the passwd file.

A setup with a users database is somewhat faster, because a lookup in the users CDB is faster than running *qmail-getpw*, and marginally more secure, because *qmail-getpw* depends on the system *getpwnam( )* library routine, which can be complex and fragile. But unless you're trying to squeeze every bit of speed out of a mail server, the more compelling argument is what you find more convenient.

15.2 Making the Users File

The format of */var/qmail/users/assign* is fairly simple. It's a sequence of lines with two slightly different formats, one for an exact match and one for wildcards. An exact match line starts with an equals sign:

```
=local:user:uid:gid:homedir:dash:ext:
```

This means that mail to address *local* is delivered to *user* with user and group IDs *uid* and *gid* and home directory *homedir*, using a qmail file named *.qmail dashext*. (Usually *dash* and *ext* are null.)

A wildcard line starts with a plus sign:

```
+loc:user:uid:gid:homedir:dash:pre:
```

In this case, any address that starts with *loc* is handled by the given user, with *pre* inserted in front of the rest of the address to determine the name of the qmail file. (In this case *dash* is usually a dash, and *pre* is usually null.)

Here's a snippet from a real *assign* file:

```
+:alias:121:105:/var/qmail/alias:-::
=carol:carol:108:102:/usr/home/carol:::
+carol-:carol:108:102:/usr/home/carol:-::
```

In this case, mail to carol is handled by the second line, and delivered using */usr/home/carol/.qmail*, while mail to carol-ina is handled by the third line and delivered using */usr/home/carol/.qmail-ina*. Any address not starting with carol is handled by the first catchall line so that mail to, say, fred is delivered using */var/qmail/alias/.qmail-fred*. Note the hyphen in the third line in carol-, so that line matches any of carol's subaddresses, but not plain carol.

Usually the list of users in *assign* is more or less the same as the list in */etc/passwd*, so qmail provides the *qmail-pw2u* utility to create your *assign* file. I use this *Makefile* to control the process:

```
cdb:        assign
        ../bin/qmail-newu

assign: /etc/passwd append exclude
        cp assign assign.old
        ../bin/qmail-pw2u < /etc/passwd > assign
```

When creating *assign*, *qmail-pw2*u uses approximately the same rules as *qmail-getpw*, ignoring any users that have a zero uid, don't own their home directory, or contain capital letters. For each user, the output contains two lines, with the username, user and group IDs, and home directory from the password file, as in the "carol" example.

Several command-line flags to *qmail-pw2u* modify the default behavior and are documented in the manpage, but I've never found the flags very useful. The only ones I've ever used are -h, fail if a user's home directory doesn't exist, and -c, change the separator character from a hyphen to something else, usually a plus sign for compatibility with the subaddressing in sendmail and postfix. What is useful is a set of auxiliary files in */var/qmail/users* that modify the generated *assign* file:

exclude

A list of users to omit, either because they shouldn't get mail or because their mail setup isn't the default. It should include accounts such as bin, daemon, and uucp that don't have human readers to read the mail. (You can and should create qmail files in *~alias* to forward mail sent to any of those addresses that are likely to get interesting mail, of

15.3 How Qmail Uses the Users Database

Once you've created the users database, qmail checks it for each local delivery. First it checks for an exact match of the mailbox as a nonwildcard address. If that doesn't work, it tries for the longest match against a wildcard, starting with the full mailbox and shrinking a character at a time until there's a match. (To speed up this process, *qmail-newu* makes a list of the final characters used by all the wildcard entries and stores it in the CDB file. When looking up a mailbox, *qmail-lspawn* only checks substrings where the last character of the substring is one of those final characters.) The wildcard match always succeeds, either against one of the subuser entries, or else against the default wildcard entry created by *qmail-newu,* which looks like this:

```
+:alias:uid:gid:/var/qmail/alias:-::
```

Once *qmail-lspawn* has the user data, either from the database or from *qmail-getpw*, it changes to the user ID, group ID, and home directory, then runs *qmail-local* to read the *.qmail* file and perform the delivery.

15.4 Typical Users Setup

The simplest arrangement makes a qmail user for all of the live users in the passwd file. In that case, in */var/qmail/users* create an *exclude* file that lists all of the passwd entries that don't correspond to people, such as root, bin, daemon, uucp, ftp, and lpd. Then create a *Makefile* as described earlier in this chapter, and as the super-user type *make*. This creates a CDB with an entry for all of the un-excluded users.

Having excluded root, bin, and so forth from your users file, be sure to arrange for mail sent to those addresses to be delivered somewhere, because daemons tend to send reports to those addresses. Either create individual qmail files like *~alias/.qmail-root* or, if you use *fastforward*, put the instructions in */etc/aliases*.

15.5 Adding Entries for Special Purposes

If your system acts as a mail server for more than the people with shell accounts, you'll probably want to add some entries to the users database.

15.5.1 Adding a Few Mail-Only Accounts

In many cases, a host serves a mix of shell and mail-only accounts. If the number of mail-only accounts is small, it's not worth installing an entire virtual domain POP system. To handle my mail-only users, I created a user *maildrop* that owns all of the Maildirs for the mail-only users. Each user has a Maildir, so that if fred is a mail-only user, his Maildir is *~maildrop/fred/* and his mail is delivered via *~maildrop/.qmail-fred*, which contains either just the name of the Maildir, *./fred/*, or more likely a call to procmail to filter out viruses and spam before delivery. Fred is a subuser of maildrop, so his address would be *maildrop-fred* rather than *fred*. To make his plain address work, you can forward his mail via a qmail file *~alias/.qmail-fred* or an entry in */etc/aliases* forwarding to *maildrop-fred*. Or what I do is to use the *subusers* file, with entries like this:

```
fred:maildrop:fred:
```

(Also modify the *Makefile* to add *subusers* to the end of the line starting with assign:, so that it rebuilds the users database if the subusers file changes.) This has exactly the desired result, to treat mail to *fred* as though it were addressed to *maildrop-fred*. It also routes subaddressed mail, so if you want Fred's subaddresses to work, you should create *~maildrop/.qmail-fred-default*, which in a simple case can be a link to *.qmail-fred* to deliver all of fred's subaddressed mail the same as his regular mail.

You must also arrange for the POP server to know about the mail-only users. See Chapter 13 for advice on doing so.

15.5.2 Preparing for the POP Toaster

If you have a more complicated mail setup, you may want to add a few custom lines to the users database by putting them in *append*. If you run a POP toaster, a mail server for POP users with mailboxes in virtual domains, and the user mailboxes belong to user pop, but you want to put the mailboxes in */var/popmail* rather than in *~pop*, just add a line like this to *append*:

```
+popmail-:popmail:111:222:/var/popmail:-::
```

(Use the user and group IDs for *pop* rather than 111 and 222, of course.) Once you've rebuilt the users database, any mail addressed to *popmail-something* will be delivered via */var/popmail/.qmail-something* or */var/popmail/.qmail-default*, running as user pop. I find this a convenient way to work, so I can put files of software and notes to myself in pop's home directory, and keep the mailboxes on a separate large filesystem.

Chapter 16. Logging, Analysis, and Tuning

Although qmail performs well in its standard configuration, it's often possible to tune it to work better, particularly for very large or very small installations.

16.1 What Qmail Logs

Qmail logs quite a lot of information about what it's doing, although it can be daunting to collect it all together. If you're using daemontools, each daemon has its own set of logs, kept in a rotating set of log files maintained by *multilog*, usually with a TAI64N timestamp (see TAI64 Time Stamps). The *qmail-send* process logs each message queued and each delivery attempt. The *qmail-smtpd* process logs each incoming SMTP connection, although it won't describe what happened during the connection. *tcpserver* logs every connection denied due to entries in the connection rules file, and *rblsmtpd* logs every connection it blocked due to a DNSBL entry. If you use QMAILQUEUE to run other programs at SMTP time, anything they send to stderr is logged, and if you've added other patches to *qmail-smtpd*, anything they write to stderr is logged, too.

A system can be set up to do logs analysis on the fly, every time *multilog* switches to a new log file or once a day in a batch. It often makes sense to combine the two, doing some work at switching time and the rest daily. Although it's usually more convenient to keep the logs for each application separate, it's not hard to create combined logs for analysis or just to keep around in case someone needs to look at them later. If a set of logs from different programs all have TAI64N timestamps, merge them using the standard sort program *sort -m*. TAI64N timestamps are fixed-length hex strings, so merging them in alphanumeric order is the same as date order.[1] Once they're merged, *tai64local* can make the timestamps readable by people. So to merge a set of log files, all of which have the standard multilog TAI64N names that start with an at-sign:

[1] Well, unless your system uses EBCDIC rather than ASCII. Unless you're running an obscure mainframe Unix version from the 1970s, it doesn't, so we won't worry about it.

```
sort -m \@* | tai64nlocal > merged-log
```

TAI64 Time Stamps

TAI stands for International Atomic Time, an extremely precise standard maintained by the International Bureau of Weights and Measures (BIPM). The BIPM is in France, so the acronyms are for Temps Atomique Inernational and Bureau International des Poids et Measures. Dan Bernstein noted that Unix has no generally accepted way to store times at a granularity of less than a second, and the standard 32-bit timestamps can't represent times before 1970 or after 2038, so he devised a new set of TAI-based timestamp conventions for his logs.

A TAI64 label is a 16-digit hex number that represents a 64-bit number of seconds. 4000000000000000 is the beginning of 1970, the same time as a zero Unix timestamp. Smaller or larger numbers represent earlier or later times. A TAI64N label is a timestamp in nanoseconds represented as a 12-digit hex number, which is a TAI64N label followed by another four-digit hex number representing the number of nanoseconds within the second. TAI64N labels are conventionally preceded by an @ sign, like @400000003ff4ccf806d0f4fc. The *multilog* program can prefix TAI64N timestamps to each line of the information that it logs, and *tai64nlocal* translates those timestamps to readable dates and times.

See http://cr.yp.to/libtai/tai64.html for more detail.

16.2 Collecting and Analyzing Qmail Logs with Qmailanalog

The qmailanalog package extracts statistics from the logs created by *qmail-send*. It consists of *matchup*, which preprocesses the qmail logs; some scripts such as *zoverall* and *zddist*, which collect and print statistics; a second set of scripts, such as *xsender*, for picking out subsets of messages to analyze; and a few other auxiliary programs and scripts. The only C programs are *matchup* and *columnt*, an auxiliary program that neatens up the columns in the reports. Everything else in the package is short awk or shell scripts that are not hard to edit.

Using qmailanalog is more painful than it should be because it expects its input log files to use an older decimal timestamp format used by the now obsolete *splogger* and *accustamp* rather than the TAI64N format used by *multilog*. I have a patch to *matchup* to translate TAI64N to the older format as the logs are read at http://www.iecc.com/qmailanalog-date-patch. The rest of the discussion here assumes that *matchup* has that patch. To build the qmailanalog package, download the current version (0.70 as of this writing) from http://cr.yp.to, download and apply the patch, do the usual *make*, then become super-user and make setup check. Normally qmailanalog installs itself in */usr/local/qmailanalog*. To change the installation directory, edit *conf-home*. The setup instructions advise against installing the programs in */usr/local/bin* because some of the names may collide with other unrelated programs.

To use qmailanalog, first you pass the raw logs through *matchup* to create a condensed file with one line per message and one line per delivery. Then the analysis scripts read the condensed files and produce reports. *matchup* writes both the condensed file and a second file listing messages that haven't been completely processed. The next time *matchup* runs, it needs that second file to pick up where it left off. The condensed file is written to standard output, and the second file to file descriptor 5.

16.2.1 Log Analysis at Rotation Time

The condensed files produced by *matchup* are about half the size of the raw qmail logs and *matchup* is fairly fast, so it makes sense to call *matchup* from *multilog* to create the condensed logs each time it switches log files, as shown in Example 16-1.

Example 16-1. Qmail log run with analysis

```
 1. #!/bin/sh
 2.   exec setuidgid qmaill \
 3.   multilog t s4000000 \
 4.   !'cat /dev/fd/4 - | /usr/local/qmailanalog/bin/matchup' \
 5.   ./main
```

Line 4 in this modified run file creates the condensed logs, using the short quoted shell script as the log processor. Because *matchup* was written before *multilog*, their file descriptor conventions almost, but not quite, agree with each other. When multilog runs the log processor, it opens the existing log file as standard input and as a file of saved data from the previous run on file descriptor 4. The standard output is saved as the old log file, named with the current TAI64N timestamp, and any output on file descriptor 5 is stored away for the next time the processor runs. Although *matchup* does write information about partially processed messages to file descriptor 5, the next run reads that information from the previous run from the standard input along with the next log file. Hence use *cat /dev/fd/4 -* to read from the two file decriptors and pipe it all to *matchup*. The result of all of this is a set of condensed log files in */service/qmail-send/log/main*.

16.2.2 Log Analysis Once a Day

It's equally possible to do the log analysis once a day from *cron* or */etc/daily*. If the original logs have to be saved,

16.3 Analyzing Other Logs

There's nothing like qmailanalog for the *qmail-smtpd* logs, mostly because the useful information in them varies so much depending on what auxiliary programs and what patches are in use. I've written some Perl scripts that read through the logs and count the rejection messages for each DNSBL in use, but they rarely reveal anything interesting beyond the dismayingly large amount of spam that's showing up at my mail servers.

16.4 Tuning Qmail

More often than not, qmail doesn't need any tuning. It's designed to work well on typical Unix systems. For local deliveries, qmail is usually disk-bound, because it syncs files and directories to disk to avoid losing mail if the system crashes. Although it's possible on some systems to set filesystem parameters to subvert the syncs, that's usually a poor economy. If you want your local mail delivered faster, get a faster disk.[2] If your system has a lot of unusually slow local delivery programs, or it runs really slow spam filters (Spamassassin can fall into that category), it's possible that local deliveries could be CPU-bound. The easiest way to find that out is with a utility like *top* that shows what's running. Much of the slowness in slow spam filters is due to DNSBL lookups, which are in fact network bound. Modern CPUs are so fast that it's a rare mail system that is even occasionally compute-bound.

[2] If you haven't priced 15K RPM SCSI disks or 10K RPM ATA disks on eBay, you may be amazed how cheap they are. Be sure to get a drive cooler, too.

Remote deliveries are invariably network-bound. If the goal is to deliver mail as fast as possible, crank the concurrency up as high as possible. Looking at the *zoverview* results, it completed deliveries of 309400658 to 65158 recipients, for an average of a little under 5 Kbps per message. The average xdelay was 5.8 seconds, so each delivery was sending under 1 Kbps. This system happens to be on a T1 line, which can transmit 192 Kbps (that's 1.5 megabits divided by 8 bits per byte). So if each delivery sends 1 Kbps and the channel is 192 Kbps, it takes about 192 simultaneous deliveries to fill up the T1. Note that the ddelay, the time from when a message enters the queue to when a delivery finishes, is 35 seconds, while the average xdelay, the time from the beginning to end of a delivery, is only 5 seconds, which means messages wait 30 seconds to get a delivery slot. The mail traffic on this system is very bursty; a message comes in for a majordomo list and is queued for delivery to the 900 members of the list. The remote concurrency is 110, so the 110 slots immediately fill up and the other 790 deliveries have to wait for slots to be available as deliveries finish. Increasing the concurrency speeds overall deliveries. (I don't do this, because there are web and other servers on the network, and I don't want to squeeze them out every time there's a mailing list message.)

These numbers are fairly typical; if the channel ran at an Ethernet-like 10 megabits, the useful concurrency would be over 1000. Of course, most networks aren't entirely dedicated to email, but these sorts of estimates remain useful for setting up a system to use as much email bandwidth as the system manager wants to use.

16.4.1 Tuning Small Servers

Usually the only tuning needed on a small server is to adjust *concurrencylocal* and *concurrencyremote*. On very small systems with slow deliveries (Spamassassin run from procmail), it may be useful to decrease *concurrencylocal* to limit the hit on system performance from a lot of incoming mail, at the cost of slower deliveries. Set *concurrencyremote* using 1 K per second per delivery so that, for example, a DSL connection with 256 Kbits/sec of outbound bandwidth is 64 Kbytes/sec, so it would make sense to set *concurrencyremote* to 64 to use all of the bandwidth or to 32 to use up to half of it.

16.4.2 Tuning Large Servers

Large servers can be tuned and patched to increase the concurrency past what's normally possible. All of the necessary patches are at www.qmail.org in the section "Patches for high-volume servers."

For systems with a very large number of injected messages, the big-todo patch improves performance. In qmail's mail queue, most of the queues are divided into 23 subdirectories, with the files distributed pseudo-randomly into the 23 directories, but incoming mail goes into a single *todo* directory. If mail is injected at a high enough rate, the *todo*

16.5 Tuning to Deal with Spam

The vast amount of spam sent from forged return addresses to nonexistent recipients causes correspondingly vast numbers of bounces and doublebounces when qmail bounces the spam and finds that it can't deliver the bounce to the nonexistent return address. Because nearly all doublebounces are now due to spam, there's little point in doing anything with them. To throw them away, change the configuration file *doublebounceto* to *nobody*, and if you haven't already done so, create *~alias/.qmail-nobody* containing a single comment line to throw the mail away. (The file can't be empty, because that's treated as a default delivery, but just # will do.)

This still queues and delivers doublebounces. To throw them away without queueing them, apply the small patch at http://www.qmail.org/doublebounce-trim.patch, which adds a special case to *qmail-send* so that if *doublebounceto* contains a blank line, doublebounces are just discarded.

16.6 Looking at the Mail Queue with qmail-qread

It's not a bad idea to look at the contents of your mail queue every week or two just to see if there's anything strange. The two utility programs to do that are *qmail-qstat* and *qmail-qread*.

For a two-line summary of your queue, run *qmail-qstat* as the super-user:

```
 messages in queue: 21
messages in queue but not yet preprocessed: 0
```

The first line is the number of messages that have been queued but not delivered yet. On most systems the number should be small, less than a hundred. If your system hosts mailing lists, the number of messages can reasonably be larger because each list message stays in the queue until every recipient address is either delivered or bounces, and on any list of significant size, there will be a few addresses that have gone bad but take a long time to bounce.

The number of messages not preprocessed should always be zero or close to it. If you have many messages waiting to be preprocessed, it means that qmail can't deliver the mail as fast as it's arriving. If you have a very large mail system you may need to install one of the big-todo patches discussed earlier in this chapter. If not, you should look at the queue in more detail and see what's clogging it up. There's no convenient tool to look at the waiting messages, but if you simply look at the files in *queue/todo* with *more*, you can easily make out the envelope information for each message. The text of the message is stored in a file in a subdirectory of *mess* with the same filename as the *todo* file. To find the message that goes with *todo/123456*, the easiest approach is *more mess/*/123456*. Don't change or delete files in any of the queue directories while qmail is running, because *qmail-send* does not expect to have files changed or deleted while it's running.

To look at the messages in the queue, run *qmail-qread*, also as super-user. If you don't use mailing lists, its report will probably be quite short, while if you do use lists, it can be enormous. On the host I use for individual user mail, its output is about 50 lines, while on the mailing list host, its output is over 29,000 lines, because the qread output contains a line for every recipient of every message including the ones that have already been delivered, which with mailing lists can add up fast.

```
 30 Dec 2003 20:49:34 GMT  #1222959  2113  <mary@example.com>
  done  remote  aaron@myvirt.com
        remote  zelda@somewhere.aq
4 Jan 2004 04:18:44 GMT  #1223051  11419  <>
        remote  user1@bogus.com
```

In this qread output, the first message from mary@example.com has been delivered to aaron@myvirt.com, but not yet to zelda@somewhere.aq. The second message, which is a bounce because it has a null sender, has not yet been delivered to user1@bogus.com. Deliveries to local recipients say local rather than remote. If some of the deliveries have failed, the report will say bouncing. The number after the # sign in each report is the message number in the queue, so you can find the file for the second message with *more mess/*/1223051*. The number after the message number is the size of the message in bytes.

When looking at the queue content for hosts with mailing lists, it is useful to leave out the addresses that are done:

```
 # /var/qmail/bin/qmail-qread | grep -v 'done'
```

On my list host, that gets the report down from 29,000 lines to 1500.

The results of qread are rarely very interesting, but when they are, if say you see a whole lot of large messages queued to addresses that you don't recognize, they can be the key to tracking down otherwise hard to detect problems.

Chapter 17. Many Qmails Make Light Work

Qmail is well-suited for environments with multiple computers working together, as well as multiple copies of qmail dividing up work in various ways. This chapter starts by looking at the aspects of qmail useful for multiple operation and then explains some common applications.

17.1 Tools for Multiple Computers and Qmail

Here's a quick rundown of the tools in our multisystem toolbox.

17.1.1 Multiple Copies of Qmail

Normally, all of qmail is installed in */var/qmail*. That directory is specified at build time in *conf-qmail*. If you change the contents of *conf-qmail* to, say, */var/qmail2* and rebuild and install qmail, you'll create a complete second copy of qmail along with its queue directories. You can send mail into it using */var/qmail2/bin/qmail-queue* or any of the programs that call it, such as */var/qmail2/bin/forward*, or by using *tcpserver* to run a SMTP service with */var/qmail2/bin/qmail-smtpd*. Outbound mail works normally, although you can control it using the standard mechanisms such as *concurrencyremote* and *smtproutes*.

Remember that qmail's queue cannot be on a shared or remote disk; a single local copy of *qmail-send* has to manage each queue.

To pass mail for particular domains from one copy of qmail to the other, you can use either SMTP or *virtualdomains*. To use SMTP, set up a SMTP daemon for the second copy of qmail on localhost (127.0.0.1), but listening on port 26 or any other unused port. Then in the *control/smtproutes/* in the first copy, route the mail for each domain to that SMTP daemon:

```
bad.example.com:localhost:26
```

To route using virtual domains, add *virtualdomain* entries to assign all the domains to a pseudo-user called qmail2:

```
example.com:qmail2
myvirt.com:qmail2
```

Then in *~alias/.qmail-other-default*, forward the mail to the other copy of qmail:

```
| /var/qmail2/bin/forward "$DEFAULT@HOST"
```

The qmail2 version of *forward* will use the qmail2 version of *qmail-queue* to queue the mail in the second copy of qmail. If you've applied the QMAILQUEUE patch, you can set QMAILQUEUE to */var/qmail2/bin/qmail-queue* in any command that queues mail to force the mail into the second copy of qmail.

17.1.2 mini-qmail

mini-qmail is a stripped-down qmail package. It uses QMQP, a faster and simpler scheme than SMTP, to send all mail to another host running regular qmail. Because *mini-qmail* makes neither local nor remote mail deliveries, and has no mail queue (all mail is sent to the QMQP server immediately), it's useful on client hosts in a mail cluster. The details of setting up *mini-qmail* are discussed later in this chapter.

17.1.3 Shared Mail Folders

Maildir format mailboxes can safely be shared read/write using NFS. Each message is written as a separate file, so the hosts creating the files use their hostnames as part of the files they create to avoid name collisions, and NFS does a reasonably good job of making file rename operations atomic; delivery to and retrieval from remote Maildirs works well. This means that one host can deliver the mail into a mailbox and another can pick it up, such as when one is the SMTP server and the other is the POP server. Or several hosts can use a shared Maildir as a gateway to a single host or service.

17.2 Setting Up mini-qmail

Installing *mini-qmail* requires two steps: installing a QMQP server or two, and then installing the *mini-qmail* QMQP client.

17.2.1 Setting Up a QMQP Server

If you already have an SMTP server running, setting up QMQP is easy, because its configuration is much simpler. The only pitfall is that QMQP has no relay protection at all, so you have to make sure that only your own QMQP clients connect to the servers. QMQP doesn't queue, which means that clients discard mail if they can't deliver it to a server immediately, so you should set up at least two QMQP servers if possible.

First, create the rules file to permit connections only from your network. Create */var/qmail/rules/qmqprules.txt*:

```
# only allow connections from our network
:deny
172.16.42.:allow
```

Replace the 172.16.42. line with your own network range(s), of course. If you created a *Makefile* for your SMTP rules file, add the QMQP rules file to it, too, and then run *make* to create *qmqprules.cdb*:

```
default: smtprules.cdb qmqprules.cdb

smtprules.cdb: smtprules.txt
        cat $> | /usr/local/bin/tcprules $@ smtprules.tmp

qmqprules.cdb: qmqprules.txt
        cat $> | /usr/local/bin/tcprules $@ qmqprules.tmp
```

Now it's time to create the directories for the QMQP service:

```
# mkdir /var/qmail/supervise/qmail-qmqpd
# mkdir /var/qmail/supervise/qmail-qmqpd/log

# mkdir /var/qmail/supervise/qmail-qmqpd/log/main
# chown qmaill /var/qmail/supervise/qmail-qmqpd/log/main
```

And create */var/qmail/supervise/qmail-qmqpd/run*:

```
1. #!/bin/sh
2. limit datasize 3m
3. exec tcpserver \
4.    -u000 -g000 -v -p -R \
5.    -x/var/qmail/rules/qmqprules.cdb 0 628 \
6.      /var/qmail/bin/qmail-qmqpd 2>&1
```

In line 4, use the values on your system for *qmaild*. Note on line 5 that the service is running on port 628. Finally, create */var/qmail/supervise/qmail-qmqpd/log/run*. It's identical to its smtpservice equivalent:

```
#!/bin/sh
exec setuidgid qmaill \
    multilog t s4000000 ./main
```

Once you have all the files created, symlink the *supervise/qmail-qmqpd* directory so *svscan* starts it up:

```
# ln -s /var/qmail/supervise/qmail-qmqpd /service
```

If you look at *log/current* you should see the initial tcpserver status line:

```
tcpserver: status: 0/40
```

Chapter 18. A Compendium of Tips and Tricks

The good thing about qmail is that there are simple ways to perform a wide variety of mail handling tasks, even though qmail doesn't have as many task-specific features as other MTAs. The bad thing is that the simple ways are often a less than obvious combination of more basic qmail features. Here is a list some common problems, and some of those tasks and combinations.

18.1 Qmail Won't Compile

You have unpacked the qmail sources and typed *make*, but it won't compile. If you're receiving error messages about errno, you've run into a compatibility problem between qmail and recent versions of the GNU C library. The fix is very simple. See Building with Recent GLIBC and Fixing the errno Problem in Chapter 3.

(This is the number one question on the qmail mailing list, so frequent that there's an autoresponder that mails back the answer to any message that contains the word "errno".)

18.2 Why Qmail Is Delivering Mail Very Slowly

If qmail seems to wait about half a minute to do anything when you inject mail, the problem is almost certainly that the *lock/trigger* file used to communicate between *qmail-queue* and *qmail-send* is messed up. That file should be a named pipe:

```
# ls -l /var/qmail/queue/lock/trigger
prw--w--w-  1 qmails  qmail  0 Nov  7 03:02 /var/qmail/queue/lock/trigger
```

If it's a regular file or anything other than a pipe, you have a problem. Fortunately, it's a problem that's easy to fix:

```
# svc -td /service/qmail-send   # shut qmail down for a minute
# tail -f /service/qmail-send/log/main/current
# # wait until the log says that it's exited
# rm /var/qmail/queue/lock/trigger  # remove bogus trigger
# cd wherever you built qmail from source
# make setup check   # recreates all the crucial files including trigger
# svc -u /service/qmail-send    # restart qmail
```

This is the second most frequently asked question on the qmail mailing list, and tends to get aggrieved responses pointing out that the answer is in the archives about a hundred times. So don't ask it, because now you know the answer.

18.3 Stuck Daemons and Deliveries

Some of the most frustrating problems are due to background daemons that don't do what they're supposed to do. Fortunately the daemontools package makes daemon debugging relatively straightforward.

18.3.1 Daemons Won't Start, or They Start and Crash Every Few Seconds

Starting a daemon under *svscan* and *supervise* is simple in concept, although the details can bite you. The super-daemon is started at system boot time by running */command/svscanboot*. It runs *svscan* to control daemons and the useful but obscure readproctitle, which takes any error messages from *svscan* and puts them into its command area so that ps will show it.[1]

[1] This odd way of displaying error messages is intended to work even in the presence of serious configuration screwups like disks that should be mounted but aren't and directories that are supposed to be writable but aren't.

Every five seconds *svscan* looks at all of the subdirectories of */service* and starts up a *supervise* process on any that don't have one running. In the usual case that the subdirectory in turn has a subdirectory called *log*, it starts a second *supervise* process in the subdirectory and pipes the output from the first process to the second.

When *supervise* starts up a daemon, it runs the file *run* in the daemon's directory. That file has to be a runnable program that either is or, more commonly, exec's the daemon itself. That means that *run* has to have its execute bits set and, if it's a shell script, start with #!/bin/sh so that it's runnable. If either of those isn't the case, there is a failed attempt to start the daemon every five seconds. A *ps l* that shows readproctitle should reveal the error messages and give hints about what needs to be fixed.

The *run* script generally sets up the program environment and then exec's the actual daemon. If you become super-user and type ./run, the daemon should start. If that works, the daemon still doesn't start, and you don't use full program paths in the *run* file, the problem is most likely that the search path that *supervise* uses isn't the same as the one you're using. Look at */command/svscanboot* to see the search patch that it uses. Most notably, it does not include */var/qmail/bin* unless you edit the file yourself to include it.

18.3.2 Nothing Gets Logged

Sometimes the daemon runs but nothing's going into the log files. This generally is due to either file protection problems or an incorrect set of *multilog* options. The usual way to run *multilog* is to create a subdirectory called *main* in which it rotates log files. It's safer to run daemons as a user other than root, so when possible, use qmaill, the qmail log user. A common error is to forget to change the ownership of the log file directory to qmaill (or whatever the log user is). When *multilog* starts successfully, it creates a *current* log file in the directory, so if there's no *main/current*, the most likely problem is directory ownership or protection.

If *multilog* is running but there's nothing logged, the most likely problems are that the daemon isn't sending anything to log, or that *multilog*'s options are telling it to discard everything. Because the daemon and the logger are connected with a regular Unix pipe, only messages sent to the daemon's standard output go to the logger. In particular, anything sent to standard error shows up in readproctitle, not the log. If, as is usually the case, you want to log the errors a daemon reports, just redirect the error output to the standard output in the *run* script with the standard shell redirect 2>&1. (That redirect is at the end of just about every *run* script example in this book.)

18.4 Mail to Valid Users Is Bouncing or Disappearing

If you use *users/assign* as described in Chapter 15, a common mistake is to add a user to the system without updating the *users* file. Fortunately, this oversight is easily remedied:

```
# cd /var/qmail/users; make
```

18.5 Mail Routing

Qmail lets you build very complex routing strategies on top of its three basic delivery paths: local, virtual, and remote.

18.5.1 Sending All Mail to a Smarthost

If your qmail system has a full-time Internet connection, route all mail to a smarthost with a default entry in *smtproutes*, e.g., :mail.myisp.com. If you have a dialup or other intermittent connection, use a default virtual domain to route all outgoing mail into a Maildir, then when you connect to your ISP, use *maildirsmtp* to take the mail out of the directory and send it to the smarthost. See Chapter 11.

If you have a few locally connected systems to which you can send mail directly, you can also put specific entries for them in *smtproutes*, overriding the smarthost default. If you use *virtualdomain* delivery, you also need not-virtual entries for each of them in *virtualdomains*, e.g., nearby.com:. See Chapter 12.

18.5.2 Treating a Few Remote Addresses as Local

If you have local users who use addresses at another system as their return address on mail, you can "short circuit" mail to them and handle mail to them as local, by creating individual address *virtualdomains* entries for them. See Chapter 12.

18.5.3 Slowing Mail Delivery to Certain Domains

Some mail servers have an unfortunate habit of accepting more incoming SMTP connections than they can handle, and then collapsing. The simplest way to limit the number of connections to a server is to route all the mail destined to it into a Maildir using lines in *virtualdomains*, then run *maildirserial* from *cron* to deliver the mail one at a time. Another approach is to install two copies of qmail, the main one with the usual high concurrencyremote level, and a second one with a very low concurrencyremote level of 5 or so. Then in the main system, for any domains that need to be fed mail slowly, use either virtual domains or *smtproutes* to hand mail for those domains to the secondary copy of qmail. See Chapter 17.

18.6 Local Mail Delivery Tricks

Even though qmail's local mail delivery design is pretty simple, it still has the flexibility to handle all sorts of situations.

18.6.1 Using a Subaddress Separator Character Other than Hyphen

Some people prefer to use a plus sign rather than a hyphen in subaddresses, so they like *carol+prunes* rather than *carol-prunes*. If you can't persuade them that their life will be easier if they use a hyphen like everyone else, it's not hard to arrange if you use *users* for local mail delivery. Create */var/qmail/users/assign* if it doesn't exist yet, and then in the user's wildcard entry, change the first hyphen to a plus:

```
+carol-:carol:108:108:/home/carol:-::

+carol+:carol:108:108:/home/carol:-::
```

Then run *qmail-newu* to rebuild the users database. That's all it takes. The plus sign only affects the separator between the name and extension, not the name of *.qmail* files, which will still be *.qmail-prunes* in this case, nor the character that separates subextensions.

In the usual case that the users file changes from time to time as the password file is updated, put the user's name in *exclude* and put the two lines for that user (the modified line that starts with a plus and the unmodified line that starts with an equals sign) in *append* so they'll be included automatically each time *qmail-pw2u* runs.

18.6.2 Customized Bounce Messages for Virtual Domains

Often a virtual domain belongs (logically at least) to a different organization than the main domain on the mail server. When mail to a bad address at the virtual domain bounces, it is nice to give an error specific to that domain. Say the domain *myvirt.com* is routed to the *myvirt* user. If addresses in that domain are handled by individual *.qmail* files, anything that lands in *.qmail-default* is a bad address, easily handled by *bouncesaying*:

```
| bouncesaying "Not a valid user at myvirt.com.  Call 617-637-VIRT for information."
```

If addresses are handled by an alias file created by *setforward*, set -p to tell *fastforward* not to fail on an unknown address so you can handle it yourself. Put these two lines in *.qmail-default*:

```
| fastforward -p myvirt.cdb
| bouncesaying "Not a valid user at myvirt.com.  Call 617-637-VIRT for information."
```

18.7 Delivering Mail on Intermittent Connections

If your qmail system is a hub host for remote systems that connect intermittently by dialup, it is straightforward but messy to deliver the mail while the remote systems are connected.

One approach is to create a flag file in a known directory when a host connects and delete the file when the host disconnects. Then run a script periodically from *cron* that loops over all of the flag files to push out mail to currently connected hosts.

To flesh out this example, assume there are three dialup hosts called red.example.com, blue.example.com, and green.example.com. Create *virtualdomains* that give them different virtual domain prefixes:

```
red.example.com:alias-dial-red
blue.example.com:alias-dial-blue
green.example.com:alias-dial-green
```

You can put all of the *alias-dial* mail into one Maildir since the Delivered-To: prefixes keep them separate. To put all the mail for the three hosts into *~alias/dialmail/*, create *~alias/.qmail-dial-default* containing the line ./dialmail/.

To track the currently connected hosts, put the flag files into *~alias/dialflags* and have the dialup connection script create a file with the host's simple name (red, blue, or green) in that directory containing the host's current IP address. Then run this script from *cron* to push out the mail to whichever hosts are currently connected:

```
#!/bin/sh
#  run this every 15 minutes from cron to push out the mail

cd /var/qmail/alias/dialflags

for hn in *
do
  ip=$(cat $hn) # IP address in the flag file

  setlock ../$hn.lock \  # lock deliveries to this host
       maildirsmtp /var/qmail/alias/dialmail \
         alias-dial-$hn- $ip my.example.com 2>&1 |
      splogger serial
done
```

If you also want to push out any waiting mail as soon as a host connects, also put a call to *maildirsmtp* into the host's connection script. Be sure to use the same lock file to avoid confusion if the *cron* job happens to run at the same time. If you add another host called purple, you only need to add another line to *virtualdomains*:

```
purple.example.com:alias-dial-purple
```

The remote hosts can use a similar setup to forward their mail to the main host, using a single smarthost entry in *virtualdomains*. See the discussion of serialmail in Chapter 11.

18.8 Limiting Users' Mail Access

Some organizations grant different amounts of access to email to different users. In particular, some are allowed to send mail outside the organization and some can't. There are a lot of different ways to set this up, but one of the simplest to set up is to create two parallel copies of qmail on the same host, one for restricted users and one for general users. Following the instructions in Chapter 17, create two instances of qmail; the regular one for unrestricted users and incoming mail in */var/qmail*, and the restricted one in */var/rqmail*. Create accounts for all of the users so that every user has a mailbox, and set up a POP (and IMAP if you want it) server.

Set up SMTP daemons for both instances on separate IP addresses, and set up the users' PCs so that the restricted users send their outgoing mail to the restricted server and the unrestricted users to the general server. To keep the restricted users from sending any mail through the general server, add their addresses to */var/qmail/control/badmailfrom*. To keep them from sending external mail from the restricted server, put this line to fail all remote deliveries into */var/rqmail/control/smtproutes*:

```
:[127.0.0.0]
```

(This is a deliberately bad address that will refuse all connections.)

Another approach that's a little harder to set up but easier to administer is to use a single copy of qmail but to check the mail as users send it. If you use the old-fashioned fixup scheme described at the beginning of Chapter 7 to handle injected mail, you can check whether a user is allowed to send external mail in the fixup script. Modify *~alias/.qmail-fixup-default* to something like this:

```
| bouncesaying 'Permission denied' [ "@$HOST" != "@fixme" ]
| ./checkrestrict
| qmail-inject -f "$SENDER" -- "$DEFAULT"
```

Example 18-2 checks whether the sender is in a list of authorized users.

Example 18-2. checkrestrict script for .qmail-fixme

```
#!/bin/sh
# inherit $SENDER and $DEFAULT from the .qmail file

  case "$DEFAULT" in
       *@example.com) # our domain, always permitted
            exit 0 ;;
       *@*) # external address
            if egrep -q "^($SENDER)$" authorized-users
            then
                exit 0
            else
                bouncesaying "You cannot send external mail."
            fi ;;
        *) # local mail, always permitted
            exit 0 ;;
  esac
```

This script needs to be ruggedized a little, because mail from user fred might have a sender of *fred* or *fred@example.com* depending on how his mail program is set up, and a local recipient address might be *mary@EXAMPLE.COM* in uppercase, but the checking remains quite simple.

If you use *ofmipd*, you can't easily use the fixup trick, but assuming you've applied the QMAILQUEUE patch, you can run *qmail-qfilter* and use a similar script that checks $QMAILUSER and $QMAILRCPTS and returns an exit code of 31 to reject the mail or 0 to permit it. (Remember that if you accept the mail, you have to copy the message from stdin to stdout, too, or the message you accept will always be empty.) Call the checking program, which can

18.9 Adding a Tag to Each Outgoing Message

Some organizations want to add a footer to every message with text that identifies the company, includes disclaimers, or makes implausible claims about the legal status of messages. This is another problem that's easily solved with *qmail-qfilter*, in this case so easily that it doesn't even need a program of its own, just a two-line script I'll call *addtag*, as shown in Example 18-4.

Example 18-4. addtag script to add a tag to messages

```
#!/bin/sh
exec /var/qmail/bin/qmail-qfilter \
    cat - /etc/mailtag
```

Put the tag in */etc/mailtag*, and set QMAILQUEUE to run the tagging script in *ofmipd* and anywhere else that mail is injected. If local programs inject mail with sendmail, you might want to rename */var/qmail/bin/sendmail* to *realsendmail* and put this in its place:

```
#!/bin/sh
QMAILQUEUE=/var/qmail/bin/addtag exec /var/qmail/bin/realsendmail "$@"
```

If you use the older fixup approach to inject mail, you can add the tag in *.qmail-fixup-default*, as shown in Example 18-5.

Example 18-5. .qmail-fixup that adds a tag

```
 | bouncesaying 'Permission denied' [ "@$HOST" != "@fixme" ]
| cat - /etc/mailtag | qmail-inject -f "$SENDER" -- "$DEFAULT"
```

18.10 Logging All Mail

Some organizations need to log all email passing in or out of their system. An obscure feature called QUEUE_EXTRA makes this quite straightforward. Every time *qmail-queue* enqueues a message, it adds the string QUEUE_EXTRA to the recipient addresses. Normally that string is empty, but you can edit *extra.h* in the qmail source code to be whatever you want. The usual change (recommended in the qmail FAQ) is to make it add a recipient called *log* to each message. Change QUEUE_EXTRA to be the exact string to add to the recipient string including the leading T and trailing null, and set QUEUE_EXTRALEN to be the length of the string. Then rebuild and reinstall qmail. See Example 18-6.

Example 18-6. Code in extra.h to copy everything to log

```
#define QUEUE_EXTRA "Tlog\0"
#define QUEUE_EXTRALEN 5
```

Now every message will be copied to the address *log*, so you can create *~alias/.qmail-log* to save the mail:

```
./logmaildir/
```

The *.qmail* file must save the mail but cannot forward it. Why not? Because forwarding mail invokes *qmail-queue* again, which will redeliver the mail to *log*, creating a nasty mail loop.

18.11 Setting Mail Quotas and Deleting Stale Mail

Because qmail's mailboxes are normally in each user's home directory, any quota scheme that applies to the user's files automatically includes the file(s) in the mailbox. For many purposes, this is all the mail quota that's needed. You may want to apply Jeff Hayward's quota exceeded patch to *qmail-local* that recognizes an over quota error and treats it as a hard error so mail is bounced back to the sender, rather than a soft error so mail stays in the queue.

For POP toasters, the vpopmail package discussed in Chapter 13 includes code to enforce mail quotas. If you build your own simpler POP-only system, use the *mailquotacheck* script in *.qmail* files to check quotas as mail is delivered. (All these have links at www.qmail.org.)

You may also want to set a policy for stale mail, so that mail is deleted from the server after some period of time. If you use Maildirs, this is very easy to implement, because each message is in a separate file with a timestamp. In each Maildir, messages in the *new* subdirectory haven't been read, and messages in *cur* have been read and left on the server. My policy is to delete unread mail after a month, on the theory that if you don't look at your mail once a month, you'll probably never look at it at all, and to delete read mail after three months. This is easily arranged with a couple of shell commands to run every day or every week. While you're at it, you might as well delete mail that's been marked deleted (the T flag in the filename) or moved into the Trash subfolder. If all the user directories are under */home*:

```
cd /home
{
   # unread mail over a month old
   find /home/*/Maildir/new -type f -mtime +30 -p
   # read mail over three months
   find /home/*/Maildir/cur -type f -mtime +90 -p
   # any mail marked deleted
   find */Maildir -type f -name "*:2,*T*" -print
   # any mail in Trash/new or cur
   find */Maildir/.Trash/??? -type f -print
} | xargs -t rm
```

If your Maildirs are somewhere else, modify the find commands appropriately to look where they are. By adding a few more commands, you can add policies like deleting mail from a spam subfolder after a week and other subfolders after some other amount of time. With slightly fancier programming, probably in Perl, it's also straightforward to delete the oldest files from a Maildir until the user is under quota. The elegance of the Maildir design makes this all much easier than with mboxes because nothing has to be locked or rewritten, and the cleanup can proceed safely while mail deliveries are going on.

18.12 Backing Up and Restoring Your Mail Queue

The bad news about backing up and restoring your mail queue in */var/qmail/queue* is that it's nearly impossible. The good news is that it's rarely necessary.

The filenames in qmail's queue directory are numbers that depend on the inode number of the file containing the text of the message. Backup and restore programs don't restore files using the same inodes that the files used when they were backed up, which means that if you back up the queue and then restore it, it won't work.

If you're moving your qmail queue from one disk to another, there are two general strategies. If you can run your system with both disks for a while, rename the old queue to something like */var/old-qmail*, build two copies of qmail as described in Chapter 17 (one for the old queue and one for the new one) start up both copies so that new mail goes into the new queue while mail in the old queue is eventually delivered or bounces, and then delete the old queue and its copy of qmail. The other is just to bite the bullet and move the queue. To do that, first shut down both *qmail-send* and anything that might put mail into the queue, preferably by shutting down the system to single user. Then copy the queue to */var/qmail/queue.old* on the new disk, and use Harald Hanche-Olsen's script at http://www.qmail.org/queue-rename to rename the files to their correct names based on their current inode numbers. You can also use the more complex *queue-fix* program for www.qmail.org, but for this purpose you don't need anything that fancy.

If your disk fails and you restore from backups, it's usually more trouble than it's worth to restore the queue. If your backup is more than a few minutes old, nearly all of the messages in the queue when it was dumped will have been delivered, and the only ones not delivered are likely to bogus addresses that will never be delivered. To clean out the queue, shut down qmail and anything that might try to queue mail, then delete any queued mail with *rm -rf /var/qmail/queue/** (be sure to type that correctly), go to the directory where you built qmail and *make setup check* to recreate an empty queue, and then restart qmail.

Appendix A. A Sample Script

A.1 A Mail-to-News Gateway

This is my batch news gateway, run every five minutes from *cron*. The incoming messages to the gateway are stored in a Maildir *~alias/newsdir*, using a virtual domain setup that sends mail to the pseudo-domains *news* and *news.example.com* to *alias-news*, which is delivered by *~alias/.qmail-news-default*.

My news gateway handles news from multiple hosts on my network by the simple trick of symlinking *newsdir*, which is exported over the LAN by NFS, into *~alias* on each host, so that all the hosts store messages into the same directory. I find this easier and faster than running a copy of the gateway on each host.

The script *run* from *cron* uses *maildirserial* to select mail messages, and *tcpclient* to open an NNTP connection to the local news server, as shown in Example A-1.

Example A-1. Script called from cron to push out news

```
#!/bin/sh

exec setlock newsdir.lock \
    maildirserial -b -t 345600 newsdir alias-news- \
        tcpclient localhost 119 \
                /var/qmail/alias/newsgate alias-news-
```

The actual mail to news script is fairly long, but nearly all of it is devoted to cleaning up headers, as shown in Example A-2.

Example A-2. Mail to news gateway script

```
#!/usr/bin/perl
# -*- perl -*-
# process batched messages from maildirserial into news

use Getopt::Std;
use FileHandle;

# options
# -d  debug, use tty for I/O
# -s  don't use date from incoming messages
#     to avoid complaints about stale news

getopts('ds');

$linelimit = 2000; # truncate long msgs after this many lines

$| = 1;

# get prefix to strip off Delivered-To:
$prefix = shift or die "need prefix";

# read null terminated input for file names
msgloop:
while(!eof STDIN) {
    my ($from, $sender, $replyto);
    {
        local $/ = "\0";
        $fn = <STDIN>;
        chop $fn;
    }

    open(MSG, $fn) or die "cannot open '$fn'\n";
```

Appendix B. Online Qmail Resources

Qmail is well supported by its online community of users. Here are some places to look.

B.1 Web Sites

There are several excellent sources of qmail information online.

http://cr.yp.to

Dan Bernstein's web site, the official source for qmail and all of his ad-on packages.

http://www.qmail.org

Russ Nelson's qmail resource site, intended to have links to all of the other resources on the Web.

http://qmail.gurus.com

The author's companion site for this book, containing scripts, updates and corrections, links to other resources, and ordering info for more copies.

http://www.lifewithqmail.org

Dave Sill's *Life with qmail*, an online guide to setting up and using qmail. It offers specific advice about where to install qmail, and where to put all of the files and directories that qmail needs. This is by far the most widely used setup and the one that qmail experts are the most familiar with, so it's the one you should use. The file and directory locations used in this book are consistent with these.

http://www.lifewithqmail.org/ldap/

Henning Brauer's *Life with qmail-ldap*, a guide to setting up *qmail-ldap*. Indispensable for *qmail-ldap* users.

http://www.ezmlm.org

The home page for the ezmlm-idx mailing list manager, with software and documentation.

http://tinydns.org

Russ Nelson's site for Dan Bernstein's *djbdns*, a DNS package that relates to BIND roughly as qmail relates to sendmail. Not required for qmail, but if you're setting up a DNS server along with your mail server, it's probably the software you want to use.

B.2 Mailing Lists

The qmail community has a variety of mailing lists. While it's possible to get excellent advice on them, the givers of advice can be rather impatient with questions from people who appear not to have checked the list archive to see if their question has been asked and answered a dozen times before, or who ask questions without giving enough detail to provide a useful answer. So be sure to read a list's archives both to look for your question and to get the tone of the list before asking.

Needless to say, all the lists about qmail are maintained in ezmlm or ezmlm-idx so that you subscribe to any of them by writing to the list address with *-subscribe* appended and then respond to the challenge. For anti-spam purposes, Dan Bernstein's lists at list.cr.yp.to also use a program called *qsecretary* that sends a confirmation challenge each time you send something to the list.

The qmail list qmail@list.cr.yp.to

A discussion list about qmail is maintained. Archives are available at http://www.ornl.gov/lists/mailing-lists/qmail.

The qmail announcement list qmailannounce@list.cr.yp.to

Announcements about new versions of qmail. Very low volume.

The ezmlm list ezmlm@list.cr.yp.to

Discussions about ezmlm and ezmlm-idx. Partial archive at http://madhaus.utcs.utoronto.ca/ezmlm/archive/maillist.html
.

Colophon

Our look is the result of reader comments, our own experimentation, and feedback from distribution channels. Distinctive covers complement our distinctive approach to technical topics, breathing personality and life into potentially dry subjects.

The animal on the cover of qmail is a tawny owl. Generally, it's dark brown and streaked with black and buff, but occasionally, it is grey. The tawny owl is the most common owl in Britain, and its distribution extends from Europe to North Africa and eastward to Iran and western Siberia. It is also found in India, southern China, Korea, and Taiwan.

The tawny owl does not built its own nest, rather it nests in natural holes and in the abandoned nests of crows, magpies, and even the nests of buzzards. It remains within its nesting territory all year round and pairbonds last for life. The female tawny owl will stay with her nestlings while the male gathers food. While the male hunts for rabbits, moles, mice, shrews, and other rodents, the female defends her territory passionately with threatening behavior and erratic flying. Occasionally, a human is attacked; in Britain, at least two people are known to have lost an eye, including Eric Hosking, the famous bird photographer.

The tawny owl is best known for its distinctive song. The normal song of the male owl announces territory, courtship, and food. The song begins with a drawn out hooo and then is followed by a pause before the male owl abruptly sings out ha, followed immediately by huhuhuhooo. Occasionally, the female tawny owl makes a similar hooting sound in response to the male's call. However, unlike the clear, resonant sound of the male song, the female's song possesses a wailing quality of wowowhooo. The duet that is performed between the two has led to a myriad of names for the tawny owl, including Billy hooter and Jenny howlet.

Sarah Sherman was the production editor and the copyeditor for qmail. Genevieve d'Entremont was the proofreader. Reg Aubry and Mary Anne Weeks Mayo provided quality control. Tom Dinse wrote the index.

Emma Colby designed the cover of this book, based on a series design by Edie Freedman. The cover image is a 19th-century engraving from the Dover Pictorial Archive. Emma Colby produced the cover layout with QuarkXPress 4.1 using Adobe's ITC Garamond font.

David Futato designed the interior layout. This book was converted by Joe Wizda to FrameMaker 5.5.6 with a format conversion tool created by Erik Ray, Jason McIntosh, Neil Walls, and Mike Sierra that uses Perl and XML technologies. The text font is Linotype Birka; the heading font is Adobe Myriad Condensed; and the code font is LucasFont's TheSans Mono Condensed. The illustrations that appear in the book were produced by Robert Romano and Jessamyn Read using Macromedia FreeHand 9 and Adobe Photoshop 6. This colophon was written by Sarah Sherman.

The online edition of this book was created by the Safari production group (John Chodacki, Becki Maisch, and Madeleine Newell) using a set of Frame-to-XML conversion and cleanup tools written and maintained by Erik Ray, Benn Salter, John Chodacki, and Jeff Liggett.

[A] [B] [C] [D] [E] [F] [G] [H] [I] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [Z]

[Team LiB]

[Team LiB]