9

Cyrus System Administration

In this chapter:

- *Cyrus System Administration*
- *Common Tasks*
- *Batch Account Maintenance*
- *Shared Folders and Bulletin Boards*
- *Mailstore Partitioning*
- *Quota Maintenance*
- *Disaster Recovery*
- *Migration from Berkeley Mailbox Format to Cyrus*
- *Mail Forwarding and Filtering*
- *Usenet Integration*
- *Troubleshooting*
- *Adding SSL Support*

Now that you've installed and configured the Cyrus server, you're faced with maintaining it. This chapter covers the basics of managing a Cyrus system on a day-to-day basis. We will walk through examples of how to create, delete, and list the properties of mailboxes using *cyradm*, the Cyrus administration tool. We will also see examples of how to use *cyradm* to manage existing mailboxes. Cyrus administrators are often faced with the task of creating, deleting, or modifying a batch of accounts. Examples of batch *cyradm* scripts are shown. We will also see examples of how to add and remove partitions to and from the Cyrus mailstore. Shared folders and bulletin boards are valuable features of the Cyrus server. We will see examples of how to set up and manage both.

Cyrus System Administration with cyradm

cyradm is a Tcl-based client for performing system administration on the Cyrus server. *cyradm* can be run in either interactive mode or batch mode. We will look at interactive mode first, and cover batch operations later in the chapter. Note that the information contained in this chapter is not intended to be a comprehensive

account of *cyradm*—there are *cyradm* command options that are rarely used. See the manual page, *cyradm*(1), or Appendix A, *Conversion from Berkeley Mail Format to Cyrus: Tools*, for the nitty-gritty details. The objective here is to cover the most common tasks that Cyrus administrators encounter.

To start *cyradm* in interactive mode, simply enter the command:

```
$ cyradm -user username hostname port
```

`username` is a Cyrus administrative user defined in */etc/imapd.conf*. `hostname` is the hostname of the Cyrus server. `port` defaults to port 143, the standard IMAP port. Here is an example interactive session:

```
$ cyradm -user cyrus localhost
localhost password: XXXXXXXX
localhost>
```

If you need help, type *help* at the prompt. When you want to quit, use one of the commands *quit* or *exit*.

cyradm has a set of commands for performing common tasks on a Cyrus system, such as creating accounts and listing users' quotas. Table 9-1 lists the commands and gives a brief description of the purpose of each. *cyradm* commands can be abbreviated to cut down on keystrokes—the abbreviations are also shown in Table 9-1.

Table 9-1. cyradm Commands

Command	Abbreviation	Purpose
listmailbox	*lm*	Lists the names of all mailboxes that match a given pattern.
createmailbox	*cm*	Creates a new top-level mailbox.
deletemailbox	*dm*	Deletes a mailbox and all mailboxes below it in its hierarchy.
renamemailbox	*renm*	Renames a mailbox.
setaclmailbox	*sam*	Adds an entry to a maibox's ACL.
deleteaclmailbox	*dam*	Deletes an entry from a mailbox's ACL.
listaclmailbox	*lam*	Lists a mailbox's ACL.
setquota	*sq*	Sets a quota limit on a quota root.
listquota	*lq*	Lists the quotas on a quota root.
listquotaroot	*lqr* or *lqm*	Lists the quota roots on a mailbox.

The *renamemailbox* command is more complicated than meets the eye—it renames a single mailbox only and ignores all other mailboxes in the hierarchy. A workaround for renaming complete mailbox hierarchies is provided later in this chapter.

The .cyradmrc File

If the file *.cyradmrc* exists in *user*'s home directory, *cyradm* will evaluate the file as a Tcl script after connecting and authenticating to *server* and just before reading the first command from standard input.

Common Tasks

This section shows examples of common tasks you'll perform every day using *cyradm*: listing, creating, and deleting mailboxes; setting quotas; and setting ACLs.

Listing Mailboxes

The *listmailbox* (or *lm*) command returns a list of mailbox names that match the pattern given as an argument. The pattern can contain one of the wildcard characters asterisk (*) or percent (%). The * wildcard matches zero or more characters. The % wildcard is like the * wildcard, except that it only matches mailboxes at a single level in the mailbox hierarchy.

You can list all the users on the system by listing their top-level mailboxes (remember, a top-level mailbox is essentially the same as a username in the Cyrus namespace):

```
localhost>  listmailbox user.%
```

To list all mailboxes one level below *abt0003*'s top-level mailbox, you would use the % wildcard character to restrict output to include only that level:

```
localhost> listmailbox user.abt0003.%
user.abt0003.drafts     user.abt0003.sent-mail
```

To list all users whose usernames begin with the letters *abt*, you would again use the % wildcard to restrict output to only top-level mailboxes:

```
localhost> listmailbox user.abt%
user.abt0003  user.abt0008
```

The next example shows how the * wildcard character returns mailboxes that match the pattern at all levels of the mailbox hierarchy:

```
localhost> listmailbox user.abt*
user.abt0003            user.abt0003.sent-mail  user.abt0008.drafts
user.abt0003.drafts     user.abt0008
```

Creating a Mailbox or Adding a User

The *createmailbox* (*cm*) command creates a new mailbox, *mailbox*. There is an optional *partition* argument that specifies the name of the partition on which to

create the mailbox. If no partition is specified, the mailbox is created on the partition named *default* (it is defined in */etc/imapd.conf* as the `defaultpartition` option).

On production Cyrus systems, users are usually added to the system in batches by running a script, but on occasion, you might have to add a new user manually. Once a top-level mailbox is created for a user, the user is officially "on the system" and can begin receiving email. To add a new user, *abt0010*, you would issue the command:

```
localhost> createmailbox user.abt0010
localhost> listmailbox user.abt0010
user.abt0010
```

abt0010's top-level mailbox would be created on the *defaultpartition*.

A top-level mailbox is essentially the same as an IMAP account—once a user has a top-level mailbox and some means to authenticate to the server, he or she has an account on the Cyrus server.

If the user does not already have authentication credentials, then you should set them up now—see Chapter 8, *Configuring the Cyrus Server*, for details on setting up authentication.

You may also create mailboxes below a user's top-level mailbox. Many sites create a few default mailboxes for each new user added to the system, such as a *Trash* mailbox (*user.username.Trash*) or a *Drafts* mailbox (*user.username.Drafts*), for the convenience of the user.

Mailbox Access Control

Cyrus has an Internet standards-compliant way of organizing access to each mailbox. That method is known as an access control list (ACL). Simply speaking, an ACL is like a security guard with a clipboard sitting at the entrance to each and every mailbox, checking all who would presume to enter against an administrative list of who's allowed to do what. A more familiar example of an access control system may be Unix file ownership and permissions. We don't want to launch into a full-fledged description of Unix access control—it's been done well in other books. Unix files and directories have an owner, and access to other users can be granted with different combinations of group ownership and permission settings on the file.

A Cyrus mailbox also has an owner, and as with a Unix file, access to the mailbox can be granted to other users. Cyrus access control is more granular, though, than

the Unix access control model. In the Cyrus model, it's possible to grant more than one group of users access to the mailbox without nesting groups, as you would have to do when dealing with Unix groups. Instead of just the read, write, and execute permissions that are granted on Unix files, a Cyrus user can be granted nine different levels of access (see Table 9-2).

When a new mailbox is created in the Cyrus system, it is created with a default set of access rights that are defined in the IMAP configuration file, */etc/imapd.conf*. That default ACL applies only to newly created top-level mailboxes—mailboxes that are created in an existing hierarchy inherit the ACL of their nearest parent mailbox. That's a bit different from what you would expect if you're familiar with Unix permissions. In the Unix system, permissions on subdirectories are not inherited from the parent directory in a filesystem.*

The question is, then, "when are ACLs used?" They are primarily used to allow users other than the mailbox owner to access a mailbox. This might be desirable when:

- You want to allow a colleague to read mail in a mailbox where you store mail related to a project you're collaborating on.
- A group of users, such as a technical support group, need to share a mailbox and keep track of the status of messages in that mailbox.
- You want to make a mailing list archive publicly accessible to the Internet.†

Read on to find out how to set the ACL for each of those three situations.

The setaclmailbox command

setaclmailbox is the *cyradm* command to modify a mailbox's ACL. The usage is:

```
setaclmailbox mailbox identifier rights
```

identifier refers to a user, group (a group is an entity specific to your authentication mechanism; e.g., a group in */etc/group* if you use Unix authentication—refer to Chapter 8 for information on setting up groups), or one of the predefined special identifiers, *anonymous* or *anyone*, which were described in Chapter 6, *Introduction to the Cyrus IMAP Server*. Rights are shown in Table 9-2.

---

* Subdirectories in a Unix filesystem can be forced to inherit the ownership of the parent directory by setting the appropriate "special" bits on the file.

† CMU makes the *info-cyrus* mailing list archive publicly available as a Cyrus shared folder. It's available through a web interface at *http://asg.web.cmu.edu/archive/mailbox.php3?mailbox=archive.info-cyrus*.

Table 9-2. Mailbox Access Rights

Access Right	Purpose
l	Look up the name of the mailbox (but not its contents).
r	Read the contents of the mailbox.
s	Preserve the "seen" and "recent" status of messages across IMAP sessions.
w	Write (change message flags such as "recent," "answered," and "draft").
i	Insert (move or copy) a message into the mailbox.
p	Post a message in the mailbox by sending the message to the mailbox's submission address (for example, post a message in the *cyrushelp* mailbox by sending a message to *sysadmin+cyrushelp@somewhere.net*).
c	Create a new mailbox below the top-level mailbox (ordinary users cannot create top-level mailboxes).
d	Delete a message and/or the mailbox itself.
a	Administer the mailbox (change the mailbox's ACL).

There are abbreviations describing the more common sets of rights that make it easier to set the more common ACLs. The abbreviations are listed in Table 9-3.

Table 9-3. Abbreviations for Common Access Rights

Abbreviation	Access Rights	Result
none	Blank	The user has no rights whatsoever.
read	*lrs*	Allows a user to read the contents of the mailbox.
post	*lrps*	Allows a user to read the mailbox and post to it through the delivery system by sending mail to the mailbox's submission address.
append	*lrsip*	Allows a user to read the mailbox and append messages to it, either via IMAP or through the delivery system.
write	*lrswipcd*	Allows a user to read the maibox, post to it, append messages to it, and delete messages or the mailbox itself. The only right not given is the right to change the mailbox's ACL.
all	*lrswipcda*	The user has all possible rights on the mailbox. This is usually granted to users only on the mailboxes they own.

Cyrus administrators (those users defined as *admins* in */etc/imapd.conf*) have *l* and *a* rights on all mailboxes by default. When a new user is added to the system, the user is first assigned all rights on her top-level mailbox. In all other cases, when a new mailbox is created, the new mailbox inherits the ACL of the closest parent mailbox. Non-user mailboxes (such as those used to export Usenet news groups) with no parent are assigned the ACL defined in the `defaultacl` option in */etc/imapd.conf*.

To compute a user's or group's mailbox access rights, the server takes the union of the user's rights and the rights of all groups the user is a member of. In the following example ACL, user *mary* is a member of group *helpdesk*, so she inherits *p* rights from the *helpdesk* group and, as a result, has *l*, *r*, *s,* and *p* rights:

```
mary lrs
group:helpdesk lrsp
```

It is also possible to assign a user negative rights by prefixing the identifier (not the access right) with a dash (-) character. The result is that the access rights are removed from the mailbox for the user or group that comprise the identifier. For example:

```
anyone read
-anonymous s
```

This ACL allows *anyone l*, *r*, and *s* rights, while *anonymous* is allowed only *l* and *r* rights. After computing a user's access rights, the server computes the user's negative rights by taking the union of all negative rights assigned to the user and all groups the user is a member of, and removes those rights.

Common examples

Earlier, we promised to illustrate how to set up ACLs for three common uses of shared folders. The first example involves sharing a mailbox with one other user. Suppose *johndoe* has a mailbox, *user.johndoe.grant-proposal*, and he wishes to give his colleague, *annsmith*, read-only access to the messages in that mailbox. *johndoe* would set the ACL on the mailbox as follows:

```
localhost> listaclmailbox user.johndoe.grant-proposal
johndoe lrswipcda
localhost> setaclmailbox user.johndoe.grant-proposal annsmith read
localhost> listaclmailbox user.johndoe.grant-proposal
johndoe lrswipcda
annsmith lrs
```

The second example involves sharing a mailbox with both read and write access to a group of users that needs to preserve the state of the mailbox between access by different users. Such a group might be a Helpdesk. In the example that follows, the group *helpdesk* (defined in */etc/group*) is given write access to the mailbox *user.help*. One member of the *helpdesk* group, *boss*, is granted administrative access—somebody has to maintain the mailbox's ACL, and *boss*, the Helpdesk coordinator, seems to be the best candidate:

```
localhost> listaclmailbox user.help
help lrswipcda
localhost> setaclmailbox user.help group:helpdesk write
localhost> setaclmailbox user.help boss all
```

```
localhost> listaclmailbox user.help
help lrswipcda
group:helpdesk lrswipcd
boss lrswipcda
```

In the final example, a site maintains a mailing list archive and wants to make it accessible to anyone on the Internet. The archive is stored in the mailbox *user.lists.security-l.archive*. To open up access to anyone on the Internet, the access rights would be set to allow anyone to read the mailbox:

```
localhost> listaclmailbox user.lists.security-l.archive
lists lrswipcda
localhost> setaclmailbox user.lists.security-l.archive anonymous read
help lrswipcda
anonymous lrs
```

Deleting a Mailbox or Removing a User

The *deletemailbox* (*dm*) command deletes a top-level mailbox, its contents, and all mailboxes below it in the hierarchy, essentially removing the user from the Cyrus system.

Administrators do not have delete rights on mailboxes by default.

Before you attempt to delete a mailbox, be sure to use the *setaclmailbox* command to give yourself explicit *d* (delete) rights before deleting a mailbox, as in the following example:

```
localhost> setaclmailbox user.johndoe cyrusadm d
localhost> deletemailbox user.johndoe
```

Managing Quotas

setquota sets the quota limit on a quota root to a given value. Quotas on the Cyrus system are always expressed in kilobytes. Typically, the quota root is a user's top-level mailbox:

```
localhost> setquota user.johndoe 15000
```

The *listquotaroot* (or *lqr*) command lists the usage and limit on the given quota root. In the following example, the user *johndoe* has a quota limit of 15,000 kilobytes on his top-level mailbox and has used 1,363 kilobytes:

```
localhost> listquotaroot user.johndoe
user.dianna STORAGE 1363/15000 (9%)
```

cyradm does not offer a facility for removing a user's quota—quotas have to be removed manually by deleting the quota file associated with the user, then rebuilding the Cyrus quota database. The details are explained later in this chapter.

Renaming a User's Account

The *renamemailbox* command:

```
renamemailbox mailbox newmailbox partition
```

renames *mailbox* to *newmailbox*. The optional *partition* argument is used if you want to move the *newmailbox* to a different partition.

renamemailbox does exactly what its name implies, and that's all it does—it renames a single mailbox. The command has an important limitation, though: it cannot be used to rename a top-level mailbox. Here is what happens when we attempt to rename a top-level mailbox:

```
localhost> renamemailbox user.diannal user.diannam
command failed: Operation is not supported on mailbox
```

Top-level mailboxes cannot be renamed because of a Cyrus architectural issue. Renaming a top-level mailbox requires changes in other parts of the system, such as the mailboxes file, the quota subsystem, the mailbox subscription database, and the names of all mailboxes below the top level in the hierarchy. This can be a problem—many sites allow users to change their account names at some time after the account is initially created. It is also important to note that *renamemailbox* does not hierarchically rename mailboxes—it only renames the mailbox that it is given as an argument. All other mailboxes below that mailbox in the hierarchy are left untouched. CMU will provide a hierarchical *renamemailbox* command in Cyrus 2.0, but it does not yet appear in any release up through Version 1.6.22. Until *cyradm* supports hierarchical renaming, it will be of limited use on its own. Fortunately, as with many limitations, there is a usually a workaround. The workaround, which is implemented as a Tcl script in Example 9-2 in the next section of this chapter, involves these steps (in the order given):

1. Create a new top-level mailbox named for the new username (for example, *user.newname*).
2. Create new sub-mailboxes for all the mailboxes in the user's old hierarchy (*user.newname.sent-mail*).
3. Replace the new, empty mailbox hierarchy with the old mailbox hierarchy.
4. Delete the old account.

5. Reconstruct the new account using the Cyrus *reconstruct* utility.
6. Set a quota root on the new account.

Note that although you may read about other workarounds that involved direct editing of the mailboxes file, we caution you never to edit the mailboxes file directly! It's an unnecessary risk—the tools exist for working within the system; it's just a matter of stringing them together in the right way to do the job.

Batch Account Maintenance with cyradm

cyradm can be invoked in a script to read and evaluate a series of Tcl commands. In batch mode, *cyradm* command names cannot be abbreviated as they can be in interactive mode (e.g., *setaclmailbox* cannot be invoked as *sam*). When running *cyradm* in non-interactive mode, you will always use one Tcl command that has not been mentioned yet: the command *cyradm connect*. The *cyradm connect* command opens an IMAP connection to the server, and it is always the first command you execute in a batch *cyradm* script.

The usage of the *cyradm connect* command is:

```
cyradm connect connectionname
```

where *connectionname* is an arbitrary handle that denotes the connection to the IMAP server. Once a connection is established, other *cyradm* commands are issued as:

```
connectionname command
```

The command *command* is any one of the *cyradm* commands discussed earlier in the chapter. It may also be one of the following commands that have not yet been introduced:

connectionname servername
: Returns the hostname of the server the connection is connected to.

connectionname authenticate
: Authenticates the connection. *connection authenticate* has two command switches, shown in Table 9-4.

Table 9-4. connection authenticate Command Switches

Switch	Function
`-user` *`username`*	Log in to the Cyrus server as *username*.

`-pwcommand` *`script`*	Perform a plaintext password login. *script* must consist of Tcl commands that return the username and password, for example:
	`cyr_conn authenticate -pwcommand {`
	`set adminid "cyrusadm"`
	`set adminpw "xxxxxxxx"`
	`list $adminid $adminpw`
	`}`

Add New Users

Example 9-1, *addusers*, is a Tcl script that uses *cyradm* to create a batch of new IMAP user accounts and set a quota root on each account. To run the script, type the command:

```
$ addusers filename
```

where *filename* is a plain file containing one username per line.

Example 9-1. The addusers Script

```
#!/usr/local/bin/cyradm -file

# Batch Cyrus user creation script. Usage: addusers filename

set inputfile [lindex $argv 0]  # Name of file containing users
set quotalimit 15360            # Quota limit in Kbytes

eval cyradm connect cyr_conn venus 143
puts stdout "Connected to IMAP server. Authenticating..."

if [catch {eval cyr_conn authenticate -pwcommand {{
    set hostname "localhost"
    set adminid "cyrusadm"
    set adminpw "xxxxxxxx"
    list $adminid $adminpw
}} } result ] {
    puts stderr "$result (cleartext)"
    return -code error $result
} else {
    puts "Authentication successful."
}
```

The script opens a connection to the Cyrus server called *venus* and logs in as user *cyrusadm* with password *xxxxxxxx*. Once the connection is established and authenticated, the input file is opened. The script loops through each line of input, assigns the contents of the line to the variable *user*, and creates a new top-level

mailbox for *user*. After the mailbox is created, the quota defined in the variable *quotalimit* at the start of the script is applied to the new mailbox:

```
if [catch {open $inputfile r} fileId] {
    puts stderr "Error: cannot open $inputfile"
} else {

    while {[gets $fileId user] >= 0} {

        ## Create the INBOX

        if [catch {cyr_conn createmailbox user.$user} result] {
            puts stderr $result
        } else {
            puts "Created mailbox user.$user"
        }

        ## Create the default mailboxes

        if [catch {cyr_conn createmailbox user.$user.drafts} result] {
            puts stderr $result
        } else {
            puts "  Created mailbox user.$user.drafts"
        }

        if [catch {cyr_conn createmailbox user.$user.sent-mail} result] {
            puts stderr $result
        } else {
            puts "  Created mailbox user.$user.sent-mail"
        }

        ## Set the quota

        puts "  Setting quota $quotalimit on user.$user..."
        cyr_conn setquota "user.$user" "storage" "$quotalimit"
    }
}
```

Rename an Account

The next script, *rename*, is a Tcl script that renames a user's mailboxes. Cyradm has a built-in *rename* command, but the command works only on top-level mailboxes—it does not rename mailboxes lower in the hierarchy. The *rename* script in Example 9-2 renames the top-level mailbox and all mailboxes below it in the user's mailbox hierarchy.

You will often find it necessary to rename a user's mailbox if her name is reflected in her username and her full name changes. In the examples provided in this section, user *anndoe* (Ann Doe) married and changed her last name to Smith, and she asked the system administrator to change her Cyrus username to *annsmith*.

First, in *cyradm*, list the user's mailboxes, excluding the top-level mailbox. Save the mailbox names in a text file—you will use it later as input to the script that renames the account:

```
localhost> lm user.anndoe.*
user.anndoe.networker.bootstrap
user.anndoe.networker
user.anndoe.saved-messages
user.anndoe.sent-mail
```

Suppose we saved the *lm* output in a file called *lm.out*. To rename *anndoe*'s account to *annsmith*, you would run the command:

```
$ rename lm.out anndoe annsmith
```

Example 9-2. The rename Script

```
#!/usr/local/bin/cyradm -file
#
#    Usage: rename filename olduser newuser
#
set inputfile [lindex $argv 0]
set oldmb [lindex $argv 1]
set newmb [lindex $argv 2]
set mailstore "/var/spool/imap/user"

eval cyradm connect cyr_conn localhost 143
puts stdout "Connected to IMAP server. Authenticating..."

if [catch {eval cyr_conn authenticate -pwcommand {{
    set hostname "localhost"
    set adminid "cyrus"
    set adminpw "XXXXXXXX"
    list $adminid $adminpw
}} } result ] {
    puts stderr "$result (cleartext)"
    return -code error $result
} else {
    puts "Authentication successful."
}
##
## Open the file containing mailbox names, and create the
## top-level mailbox.
##
if [catch {open $inputfile r} fileId] {
    puts stderr "Error: cannot open $inputfile"
} else {

    ## Create the toplevel mailbox

    if [catch {cyr_conn createmailbox user.$newmb} result] {
        puts stderr $result
    } else {
        puts "Created mailbox user.$newmb"
    }
```

Example 9-2. The rename Script (continued)

```
    while {[gets $fileId line] >= 0} {

        ## Build the new mailbox name from the old one
        set newf [join [lreplace [split $line .] 1 1 $newmb] . ]

        ## Create the mailbox
        if [catch {cyr_conn createmailbox $newf} result] {
            puts stderr $result
        } else {
            puts "Created sub-mailbox $newf"
        }
    }

    file delete -force /var/spool/cyrus/user/$newmb
    file copy $mailstore/$oldmb $mailstore/$newmb
    {exec /usr/bin/chown -R cyrus:mail $mailstore/$newmb}

    ## Delete the old account
    if [catch {cyr_conn setaclmailbox user.$oldmb cyrusadm d} \
        result] {
        puts stderr $result
    } else {
        puts "setaclmailbox user.$oldmb cyrusadm d"
    }

    if [catch {cyr_conn deletemailbox user.$oldmb} result] {
        puts stderr $result
    } else {
        puts "Deleted mailbox user.$oldmb"
    }
}

puts "Please run \'reconstruct user.$newmb\' as cyrus."
```

After renaming the mailboxes, run *reconstruct* as the *cyrus* user, before doing anything else. Until you have run *reconstruct*, the account is not fully active:

```
$ reconstruct -r user.annsmith
```

Shared Folders and Bulletin Boards

The Cyrus IMAP server is unique in its capability to make a mailing list available to many users via the IMAP protocol alone. Cyrus accomplishes that feat with shared folders and bulletin boards.

Shared folders and bulletin boards are ordinary Cyrus mailboxes with ACLs that allow more than one user access to the mailbox. There is really not much difference between shared folders and bulletin boards: they are both Cyrus mailboxes,

and both allow users other than the mailbox owner to access the mailbox with the permissions defined in the mailbox's ACL.

When a Cyrus mailbox is referred to as a *shared folder*, it generally means that it is a mailbox owned by an individual user who wants to allow other users access to the mailbox. An additional feature of a shared folder is that, other than the "read" flag, it does not retain message state information that is unique per user. Message information like *deleted* or *important* is global to all users. That feature could be useful if it is desirable to preserve state information across accesses by different users. If you use a shared folder for a group of users, such as a Helpdesk, chances are you will want to preserve the *seen* and *answered* states across sessions, to provide a sort of work flow for the Helpdesk employees.

A *bulletin board* is a Cyrus mailbox that is owned by the system, rather than by an individual user. Bulletin boards are generally used when it is desirable to maintain a per-user *seen* state. Good uses for bulletin boards are forums such as Usenet groups and Internet mailing lists.

There are no fixed rules, only guidelines, about whether to use a shared folder or bulletin board for your particular application.

Implementing Shared Folders

When creating a shared folder, the first step is to create the mailbox that will be shared. Because clients handle mailbox names differently, make the mailbox name as descriptive as possible. Remember that with Unix authentication, the mailbox name is actually the name of a user in the password file. That limits the mailbox name to eight characters or less. A good guideline to follow when creating the top-level mailbox is to configure for the lowest common denominator. Don't use more than eight characters in the top-level mailbox name—anticipate a migration to another mail system or authentication system someday in the future. There is also the issue of MUAs—some clients can get confused by long usernames. If there are to be sub-folders in the hierarchy, you need not be as strict with naming them.

After creating the mailboxes, set the ACL to allow the appropriate users access. For example, suppose we need a shared mailbox called *announce*, with sub-folders called *events*, *for_sale*, and *official_notices*. The user *announce* has all permissions on all the mailboxes in the hierarchy. We also want the sub-folders to be readable by everyone on the Cyrus system, and we want only two users, *johndoe* and *msmith,* to append messages to the sub-folders. To allow those two users to post messages to the sub-folders, the ACLs on the sub-folders should look like:

```
localhost> lam user.announce.events
announce lrswipcda
msmith p
johndoe p
anyone lr
```

johndoe and *msmith* can append mail directly to any of the mailboxes in the hierarchy via IMAP by using the "+" notation. For example, to post a message directly to the *events* mailbox, *johndoe* or *msmith* would send a message to:

```
announce+events@yourdomain.com
```

The *deliver* program would append the message to the *events* folder in the *announce* mailbox. If any other user attempts to append mail directly to one of the shared folders, it will end up in *announce*'s inbox, where only *announce* can see it.

In order for a user to post mail to a mailbox using the + notation, that user must have *p* access rights on the mailbox.

That implies that the ACL on the top-level mailbox must be set more strictly than the *events*, *for_sale*, and *official_notices*. First, we create the top-level mailboxes and the sub-folders:

```
localhost> createmailbox user.announce
```

By default, the *user.announce* mailbox is created with an ACL that grants all rights to the mailbox owner, *announce*. Because we don't want other users to view the mailbox, no change in the ACL is necessary on the top-level mailbox:

```
localhost> createmailbox user.announce.events
localhost> createmailbox user.announce.for_sale
localhost> createmailbox user.announce.official_notices
```

Finally, set the ACL on each sub-folder to allow everyone read and list access, and *msmith* and *johndoe* permission to post messages to the mailboxes (repeat the last three commands for *msmith*):

```
localhost> setaclmailbox user.announce.events anyone lr
localhost> setaclmailbox user.announce.for_sale anyone lr
localhost> setaclmailbox user.announce.official_notices anyone lr
localhost> setaclmailbox user.announce.events johndoe p
localhost> setaclmailbox user.announce.for_sale johndoe p
localhost> setaclmailbox user.announce.official_notices johndoe p
```

If you use a shared folder to provide a group of staff members with a forum, distribute your workload—delegate someone as the shared folder or bulletin board owner. Give that person *all* access rights, so that person can manage the mailbox's ACL and its permissions as changes in staff occur.

Implementing Bulletin Boards

As with shared folders, the first step in creating a bulletin board is to create a mailbox that will be used as the bulletin board. After creating the mailbox, set the ACL—the ACL should have the *l* and *r* bits set so that the users can see and read the mailbox. If users are allowed to post to the bulletin board, then the *p* bit should also be set. If they are allowed to store messages they have seen in the bulletin board, then the *s* bit should be set.

If your installation's *sendmail* configuration was built using the *cyrus-proto.mc* M4 file that came with the *sendmail* distribution, mail sent to *bb+mailboxname* will be delivered to the bulletin board.

Sites that use some MTA other than *sendmail* should take a look at the *deliver*(8) manpage for details on how to invoke *deliver* to allow postings to the bulletin board.

A Word of Warning . . .

d (delete) rights should be used on bulletin boards and shared folders with caution. While *d* rights may be granted with the intention of allowing a user to delete messages in the folder or bulletin board, *d* rights also enable the user to delete the folder or bulletin board itself. When the folder or bulletin board is deleted, incoming mail bounces. The problem has been known for some time and is fixed in Cyrus Version 2.0, due to be released soon. Although the problem cannot necessarily be considered a bug, a fix exists that can be applied to versions of the source code prior to 2.0 to allow *d* rights yet prevent users from deleting the folder itself.

The fix involves changing code that checks the user's ACL in *imap/mailboxlist.c* in the *mboxlist_deletemailbox* function. The code provided in the Cyrus distribution checks the user's ACL for *d* rights before allowing him to delete a mailbox. By changing the code to test for *a* rights instead of *d* rights, a user can still be allowed to delete messages in the folder, but must have administrative rights added to his ACL before he can delete the folder. There are two ACL tests in *mboxlist_deletemailbox*, shown in Example 9-3.

Example 9-3. The Offending Code from mboxlist.c (Cyrus IMAP Version 1.5.19)

```
mboxlist_deletemailbox(name, isadmin, userid, auth_state, checkacl)
char *name;
int isadmin;
char *userid;
struct auth_state *auth_state;
int checkacl;
{
```

Example 9-3. The Offending Code from mboxlist.c (Cyrus IMAP Version 1.5.19) (continued)

(Lines deleted for brevity...)

```
        /* Check ACL before doing anything stupid
         * We don't have to lie about the error code since we know
         * the user is an admin.
         */
        if (!(acl_myrights(auth_state, acl) & ACL_DELETE)) {
            return IMAP_PERMISSION_DENIED;
        }
```

(More lines of code deleted...)

```
    access = acl_myrights(auth_state, acl);
    if (checkacl && !(access & ACL_DELETE)) {
        mboxlist_unlock();
```

The same code, with the fix applied (in boldface type), is shown in Example 9-4.

Example 9-4. Fixed mailboxlist.c

```
mboxlist_deletemailbox(name, isadmin, userid, auth_state, checkacl)
char *name;
int isadmin;
char *userid;
struct auth_state *auth_state;
int checkacl;
{
```

(Lines deleted for brevity...)

```
        /* Check ACL before doing anything stupid
         * We don't have to lie about the error code since we know
         * the user is an admin.
         */
        if (!(acl_myrights(auth_state, acl) & ACL_ADMIN)) {
            return IMAP_PERMISSION_DENIED;
        }
```

(More lines of code deleted...)

```
    access = acl_myrights(auth_state, acl);
    if (checkacl && !(access & ACL_ADMIN)) {
        mboxlist_unlock();
```

Mailstore Partitioning

Cyrus scales well as your storage requirements grow. Scaling is accomplished by spreading the mailstore across filesystems. New partitions can be added to the

mailstore at any time without requiring downtime,* copying of files, or even the users' knowledge.

The default Cyrus configuration requires two properties related to mailstore partitioning in */etc/imapd.conf*:

```
partition-default: /var/spool/cyrus
defaultpartition: default
```

Under this default configuration, when a new mailbox is created, it inherits the partition of its parent mailbox. If the new mailbox does not have a parent mailbox, then the mailbox is placed on the default partition *partition-default*.

Depending on the specifics of the hardware configuration, the system administrator may want to distribute the mailstore across several disks. Suppose, for example, a system with a single disk for its mailstore reached 90% of its disk capacity, and the administrator added two disks and a new disk controller. In this case, the system administrator would probably decide to keep her mailstore partitioning as simple as possible and add a new partition for each new disk.† After adding the new hardware to the system, formatting the disks, and mounting the new partitions, the administrator would create a *user* subdirectory under each new partition and change */etc/imapd.conf* to look like:

```
partition-default: /var/spool/cyrus
partition-1: /var/spool/cyrus1
partition-2: /var/spool/cyrus2
defaultpartition: default
```

To add new accounts to one of the new partitions, the administrator would need to specify the partition name as an argument to the *createmailbox* command. If she does not specify the partition, the mailbox is created on the default partition. For example, using *cyradm* interactively, give the command:

```
localhost> createmailbox user.marydoe 1
```

The *createmailbox* command creates a new account for *marydoe* and, because `partition-1` is located on the filesystem */var/spool/cyrus1*, places *marydoe*'s mailbox in */var/spool/cyrus1/user/marydoe/*.

To move existing accounts to other partitions, the rename script (Example 9-2) could be used. There is also a very useful public domain IMAP tool, *fast.imap* (see Chapter 18, *IMAP Tools*), which can be used to manually move or automatically balance mailboxes across partitions.

---

* This is assuming the partitions are being added, not moved. Downtime may, of course, be required to add new disk hardware to a system, but that downtime would be required regardless of the mail server software.

† While simple, the configuration takes maximum advantage of the hardware by spreading I/O requests across the two controllers.

A common partitioning scheme that is used at many sites involves dividing the mailstore into 26 partitions, one for each letter in the alphabet. Users are then assigned to the partition that corresponds to the first letter of their last name. Although many sites do this to make the system more scalable or to work around filesystem limits, we don't recommend it. The problem with the alphabet partitioning scheme is that you're likely to get an uneven distribution of mailboxes across partitions; e.g., your "m" partition might fill up before your "z" partition. Additionally, having a large number of small partitions makes the system more complex and difficult to manage. Careful configuration of the system makes this sort of partitioning scheme unnecessary. Keeping a smaller number of partitions keeps the system scalable without making it more complex than it has to be.

Quota Maintenance

On occasion, quotas will get horked. A common corrupted quota scenario goes like this: a user receives alerts from his client that he cannot save mail to his folders.* You use *cyradm* to check his quota, and everything looks fine—he has plenty of space. However, you check a little deeper and you see that his mail is being deferred to the queue and is not being delivered to his mailbox, which usually happens when a user is over quota. A quick way to determine whether the problem is quota-related is to use the *quota* command to fix the user's quota root (run the command as *user cyrus* from the shell prompt):

```
$ quota user.username
```

After running *quota*, run *cyradm* and check the quota again. Chances are that it will report the user as being over quota when you run it this time.

The *quota* command, when run with no command-line arguments, can also be used to report quota limits and usage on your entire user base:

```
$ quota
Quota  % Used    Used Root
15360       0       0 user.aa0002
15360       0       0 user.aa0006
15360       0     137 user.aa0008
15360       0       0 user.aa0009
```

If quotas become corrupted, all that is required to fix the problem is to run the *quota* command with the *–f* switch:

```
$ quota -f
```

* The errors depend on the client and can range from "permission denied" to "cannot save to mailbox."

The command may require several minutes to complete, depending on the number of users your system supports. More information on quota subsystem maintenance is provided in later in the chapter.

Disaster Recovery

Anticipate disaster! The best approach you can take is a proactive approach. Here are some things you can do to be prepared.

Checkpoint Your mailboxes File

Reconstructing the *mailboxes* file is time-consuming, but you can avoid reconstructing that file from scratch. Save a history of past versions of the *mailboxes* file periodically on disk so that you can go back in small increments of time to the last good copy of the *mailboxes* file. True, it may be a few minutes too old to bring your system back to its exact state before the problems started, but in most cases that is better than bringing the system down for several hours to reconstruct the *mailboxes* file. Saving one copy of the file is not useful—suppose the corruption in the *mailboxes* surfaced hours ago, and you've been backing up the corrupt file every five minutes since then? To be on the safe side, save each set of files on a separate physical disk. Even better yet, save *two* copies in two different physical locations. Save as often as possible, as often as every few minutes. A script for rotating the *mailboxes* file is given in Example 9-5 later in this chapter.

Back Up Your Data

Have a good backup strategy. While reliable backups are essential, the ability to recover files on demand is equally important. Are you prepared to restore any part of your system right now? If so, document where the data is kept and how to recover it.

Be Prepared for More than One Disaster

Disasters can happen simultaneously at more than one level in your infrastructure. What would you do if you lost both your tape backup system and your mail server at the same time? Document your infrastructure—what systems and services does Cyrus depend on? Include one or more extra Cyrus servers that can take over while the primary server is being recovered.

Keep Hard Copies of Your Configuration

If a disk containing part of your Cyrus system is lost, you must be prepared to install a new disk, rebuild the filesystem on the new disk, and restore data. To

rebuild the disk, you need to know how the filesystems were laid out on the disk and how much space was allocated to each partition. Keeping hard copies of configuration data, such as your IMAP server configuration file or your MTA configuration, can make recovery of the server much easier if for some reason you must install it from scratch. Useful information to keep in hard copy includes:

- Partition tables from the format command
- Output of the *df* command
- */etc/vfstab* or */etc/fstab*
- */etc/imapd.conf*
- *sendmail.mc*
- RAID configuration files
- Automounter configuration files

Disasters and Recovery Strategies

In this section, we'll look at some common disasters and how to recover from those disasters.

Corruption or inconsistency in the mailboxes file

As we learned in Chapter 7, *Installing the Cyrus IMAP Server*, the */var/imap/mailboxes* file is critical to the operation of the Cyrus system. Fortunately, Cyrus comes with a utility, *reconstruct*, that can be used to rebuild the *mailboxes* file in case of loss or corruption.

The *reconstruct* utility must always run as user *cyrus*! If run as *root* or another user, it will reset the ownership of the mailbox and the mailboxes will no longer be accessible.

Shut down *imapd* before running *reconstruct*. Comment out the *imapd* entry in */etc/inetd.conf*, and send a HUP signal to the *inetd* process to turn off *imapd*. Then, as user *cyrus*, run the *reconstruct* utility with the *–m* switch:

```
$ reconstruct -m
```

The *–m* argument tells *reconstruct* to correct the system-wide *mailboxes* file, if possible. After correcting *mailboxes*, *reconstruct* checks each partition defined in */etc/imapd.conf* for mailboxes, and adds them to *mailboxes* if necessary. When *reconstruct* finishes its work, uncomment *imapd* in */etc/inetd.conf* and send another HUP signal to *inetd* to turn *imapd* back on.

Be aware that on a system with a large number of users, *reconstruct* can take quite some time to run.* It is strongly recommended that on all Cyrus systems, but especially on large systems, you checkpoint your *mailboxes* file to an alternate disk periodically throughout the day and that you keep copies going several hours back. The *mailboxes* file changes when:

- A new user account is created
- A mailbox is created or deleted
- A quota root changes
- A mailbox's ACL changes

If your system has a large number of users, or your users are very active, your *mailboxes* file changes often. For systems with an active *mailboxes* file, checkpointing the *mailboxes* file every 5 or 10 minutes is recommended. Having a good copy of *mailboxes* around will help you avoid taking time to recover the file from tape and will save hours of downtime by making a *reconstruct* of the complete *mailboxes* file unnecessary. Always try replacing the corrupted *mailboxes* file with the backed-up copy before running *reconstruct*. If the file is current enough, chances are that it will suffice and the entire file will not need to be rebuilt. Example 9-5 shows a script for rotating the *mailboxes* file.

Example 9-5. Script to Rotate the mailboxes File

```
#!/usr/local/bin/perl

$file   = "/var/cyrus/mailboxes";
$gzip   = "/usr/local/bin/gzip";
$maxrot = 60;
$suffix = "gz";

if (! -e $file) { print "$file does not exist! exiting.\n"; exit; }

$rot = $maxrot;
while ($rot >= 0) {

    $rotn = $rot + 1;
    if (-e "$file.$rot.$suffix") {
        rename "$file.$rot.$suffix", "$file.$rotn.$suffix";
    }
    $rot = $rot - 1;
}

`cp $file $file.0`;
`$gzip $file.0`;
```

* On a Sun Enterprise 3500 with the *mailboxes* file on local disk, *reconstruct* took 7 hours to rebuild a *mailboxes* file containing 60,000 entries.

On less active systems, backing up *mailboxes* once per hour should be sufficient. If you have a very small number of users, *reconstruct* runs quickly and it's not really necessary to checkpoint your mailboxes file.

reconstruct does not adjust the quota usage recorded in the quota root files. After reconstructing or recovering the *mailboxes* file, it's always necessary to rebuild the quota subsystem using the *quota –f* command.

Corruption in a user's mailbox

On occasion, you will encounter inconsistencies in individual mailbox directories. *reconstruct* can also be used to recover from inconsistencies in mailboxes and mailbox hierarchies. To rebuild a single mailbox without affecting any other mailboxes in the same hierarchy, run *reconstruct* with the mailbox name as an argument. Remember, *reconstruct* must be run as the *cyrus* user:

```
$ reconstruct user.johndoe
user.johndoe
```

To rebuild a mailbox and all mailboxes below it in its local hierarchy, use the *–r* argument to *reconstruct*:

```
$ reconstruct -r user.johndoe
user.johndoe
user.johndoe.Trash
user.johndoe.work
user.johndoe.work.projects
```

reconstruct first checks the mailbox for *cyrus.header* and *cyrus.index* files. If the files exist, then *reconstruct* recovers information from those files that it cannot gather from the message files, such as the date stamp and flag names and states. *reconstruct* recovers information from the message files themselves that it cannot find in the header and cache files.

Inconsistency in quotas

On rare occasions, a mailbox will wind up with the wrong quota root. When this happens, the *cyradm listquota* command will report an incorrect quota usage. A good indication that something in the quota system has gone haywire is when a user's mail is bouncing with "Deferred—quota exceeded" errors, but *listquota* reports that the usage is below the limit. The Cyrus distribution includes a tool, *quota*(8), for maintaining the consistency of the quota subsystem.

You may recall from Chapter 6 that the Cyrus quota subsystem is comprised of a directory in the Cyrus configuration area that contains a set of text files, one file per quota root. The association between a mailbox and a quota root is not made in the quota subsystem—that information is maintained in the *mailboxes* file. Each

quota file, which is named after the quota root, contains the quota root's quota usage (in bytes) and quota limit (in kilobytes). Here is an example of a quota file:

```
$ cd /var/imap/quota
$ cat user.aa0006
4052251
15360
```

The first line of *user.aa0006* is the quota usage, in bytes; the second line is the quota limit, in kilobytes.

If we go to the *mailboxes* file and look for the quota root *user.aa0006*, we can find all mailboxes that it applies to. Here is an excerpt from the *mailboxes* file. The first field in the line is the mailbox name, the second is the partition on which the mailbox resides, the third field is the quota root (sans the "user." prefix), and the last field is the mailbox's ACL:

```
user.aa0006             default aa0006  lrswipcda
user.aa0006.Drafts      default aa0006  lrswipcda
user.aa0006.Trash       default aa0006  lrswipcda
user.aa0006.sent-mail   default aa0006  lrswipcda
```

The *quota* program, when run with the *–f* option, fixes inconsistencies in the quota subsystem by recalculating the quota root of each mailbox and the quota usage of each quota root:

```
$ quota -f
```

The *quota* command first repairs the quota subsystem, then reports its results after all repairs have been made. The results include the quota root, its usage before the consistency check, and its usage after the repairs:

```
user.aa0002: usage was 0, now 307223
user.aa0006: usage was 5181, now 4052251
user.aa0030: usage was 105945, now 5446403
user.aad0006: usage was 0, now 6261
user.aadavis: usage was 0, now 8239
```

The Cyrus distribution does not include a tool for removing a quota root. To remove a quota root without removing the mailbox it is associated with, you must remove the quota root's file from the */var/imap/quota* directory, then run the *quota* command to make the quota subsystem consistent:

```
$ rm /var/imap/quota/user.aa0002
$ quota -f
```

It is advisable to run the *quota* command periodically (e.g., once per week) out of *cron* to keep the quota subsystem in a consistent state.

Loss of a disk

You heeded our advice and saved copies of your disk configurations, so you know how the partitions were laid out on the disk and what each one is used for. Here are the steps to follow:

1. Replace the disk.
2. Boot your system to single-user mode.
3. Rebuild the filesystem on the new disk and determine that you can mount each partition on its usual mount point.
4. Restore data to the new filesystem.
5. Arrange your startup scripts so that *sendmail* does not start up when the system boots, and comment out the entry for *imapd* in */etc/inetd.conf*.
6. If you recovered data that included the *mailboxes* file or any part of a mailbox, then run *reconstruct*.
7. Reboot the system. If the system comes up with all partitions mounted, then start *sendmail* and uncomment the entry for *imapd* in */etc/inetd.conf*.

Migration from Berkeley (Unix) Mailbox Format to Cyrus

Unix systems and the out-of-the-box UW IMAP server both store mail messages in Berkeley mail format (also referred to as *mbox, Unix,* or */var/mail* format) mail folders. Many sites that use Berkeley format move their users onto a Cyrus server to take advantage of the quota and ACL support that Cyrus offers. In this section, we'll walk though the steps involved in such a migration. Source code for the tools we used to accomplish the migration are provided in Appendix A.

How Do I Know My Mail Is Berkeley Format?

If your mail setup matches six out of seven of the criteria below, then you're storing mail in Berkeley format:

- There is no mail server—users log on to a Unix machine and run a mail program to read their mail.
- There *is* a mail server, but it's the UW IMAP server, running out-of-the-box with no special site configuration.
- Each user's incoming mail is stored in a spool directory, such as */var/mail* or */var/spool/mail*.

- Each user's incoming mail is stored within a single file, named after the user.
- Mail folders are stored in the user's home directory.
- Mail reading is done with programs such as Elm, PINE, *mail*(1), *rmail*(1), or *mailx*(1).
- Each mail folder contains a header, blank line, and message body, and delimits its messages with the From header line.*

Issues

When converting a production mail system to Cyrus, several issues need to be taken into consideration to make the conversion go smoothly.

User-driven versus batch conversion

Will you take responsibility for moving your users over to the Cyrus server, or will you put some utility in place to allow them to do it on their own? If moving to Cyrus is optional, it might be easier on you if the users migrate their own mail to the new server. A one-time, all-or-none conversion might be optional in your circumstances. If you don't have another immediate need for the old machine and can keep both the old and new machines running simultaneously, then you could take advantage of a user-driven conversion. The size of your user base plays a role, too—user-driven conversion is much easier to manage in a small user base. In a larger user base, it becomes difficult to get everyone to move their mail before your deadline.

If you are constrained to providing mail service on the new Cyrus server immediately, then you will need to move all your users at one time in batch mode.

Downtime

Downtime is a serious consideration for sites that depend on Cyrus as a production mail server. You may decide to bypass the issue of downtime completely and move one or two users at a time over a period of months until everyone has been moved to the Cyrus server. Some sites do not have the infrastructure or hardware to support that strategy. Those sites will need to move all users from the old system to Cyrus in one fell swoop.

Downtime can be considerable if the migration is not planned carefully. If downtime is a serious issue, then a dry run of your migration plan is advisable.

---

* The From and From: header lines are not the same.

It's all or nothing!

Once you've migrated users to your Cyrus system and start delivering mail to the system, it is possible, but time-consuming, to revert to Berkeley mail. Be prepared to revert to your old system if something goes terribly wrong mid-migration. Tools for backing out of a conversion are provided later in this chapter.

Tools

Unfortunately, there are no generic migration tools good for both large and small user bases. Mark Crispin at the University of Washington developed a set of tools that includes mailbox conversion programs (see Chapter 18). The UW tools can be used to migrate a small number of mailboxes. However, those tools use the IMAP protocol to move mail around and require authentication for each user being converted, making them unsuitable for a mass migration. Those tools are somewhat problematic to use for conversion to Cyrus, and after a few days of working with them unsuccessfully, we found it easier to write our own tools. In this chapter, we provide generic tools that are good for user-driven and both large and small batch conversions.

Backward compatibility

You may be supporting features that could become problematic on a Cyrus server, such as server-side mail filtering or *.forward* files. Keep those features in mind when you're drawing up your migration plans.

User-Driven Conversion

Users can copy their own Berkeley-format incoming mailbox and mail folders into a Cyrus mailbox using the Perl script, *user2cyrus*, shown in Example 9-6. The script makes the transfer using IMAP, so the Berkeley folder and Cyrus server need not reside on the same machine. The script does assume that each user already has an account on the Cyrus server.

If the user runs *user2cyrus* while she has an IMAP client open, she may need to exit and restart the client to see the new mailbox and messages.

To provide *user2cyrus* to your users, you would need to install it somewhere public, like */usr/local/bin*, on the machine where the Berkeley format folders reside. The script automatically skips the first message in the Berkeley folder to avoid

copying the "FOLDER INTERNAL DATA" message. If you want to copy the message, then edit the script as you wish.

user2cyrus requires the NetxAP Perl module, which is freely available from CPAN.

Example 9-6. The user2cyrus Script

```
#!/usr/local/bin/perl
#
# This is a modified version of a public domain script written by
# Steve Snodgrass (ssnodgra@fore.com).
#
# user2cyrus - Dump a user's Unix mail file into a Cyrus mailbox
#
# Usage:    user2cyrus mbox
#
# Input:    Name of an RFC 822 mail folder
#
# Dependency:    NetxAP Perl module from CPAN
#
use File::Basename;
use Net::IMAP;

# Set this to the hostname of your IMAP server
$IMAPSERVER = "europa.acs.unt.edu";

$mbox = "$ARGV[0]";
if (!$mbox) { die "Usage: $0 mbox\n"; }

chop ($whoami = `/usr/ucb/whoami`);
if ($whoami eq "root" || $whoami eq "cyrus") {
    die "This script cannot be run by a privileged user!\n";
}

#
# Main Code
#

# Log in to Cyrus IMAP server
($user, $pass) = GetLogin();
$imap = new Net::IMAP($IMAPSERVER, Synchronous => 1);
$response = $imap->login($user, $pass);
print "Login: ", $response->status, "-",
    $response->status_text, "\n";

# cyrmailbox is the mailbox name on the Cyrus server
$cyrmailbox = "user." . "$user." . basename($mbox);

# Create the new mailbox. If the mailbox already exists, do not
# allow its contents to be overwritten!
$response = $imap->create($cyrmailbox);

print "Create: ", $response->status, "-",
    $response->status_text, "\n";
```

Example 9-6. The user2cyrus Script (continued)

```
if ($response->status eq "NO") {
    print "Mailbox $cyrmailbox already exists on Cyrus server!\n";
    print "Rename your file and try again.\n";
    $response = $imap->logout();
    print "Logout: ", $response->status, "-",
        $response->status_text, "\n";
    exit;
}

# Copy the mbox
if (-s $mbox) {
    TransferMbox($imap, $cyrmailbox, $mbox);
}

# Disconnect from IMAP server
$response = $imap->logout();
print "Logout: ", $response->status, "-",
    $response->status_text, "\n";

#
# Get username and password information
#
sub GetLogin {
    my ($username, $password);

    print "Enter your IMAP username: ";
    chop ($username = <STDIN>);
    system "stty -echo";
    print "Enter your IMAP password: ";
    chop ($password = <STDIN>);
    system "stty echo";
    print "\n";
    return ($username, $password);
}

#
# Dump a Unix-style mbox file into a Cyrus folder
#
sub TransferMbox {
    my ($imap, $mailbox, $mboxfile) = @_;

    my $blank = 1;
    my $count = 0;
    my $message = "";
    my $response;

    print "Transferring $mboxfile...\n";
    open(MBOX, $mboxfile);
    while (<MBOX>) {
    if ($blank && /^From /) {
        if ($message) {
        chop $message; # Remove extra blank line before next From
        $response = $imap->append($mailbox, $message) if $count;
        $count++;
```

Example 9-6. The user2cyrus Script (continued)

```
        }
        $message = "";
    }
    else {
        chop;
        s/$/\r\n/;        # IMAP requires CR/LF on each line
        $message .= $_;
    }
    $blank = /^\r$/ ? 1 : 0;
    }
    $response = $imap->append($mailbox, $message) if $count;
    $count++;
    close(MBOX);
    print "Transferred $count messages from $mboxfile to
        $mailbox.\n";
}
```

Batch Conversion: An Example and Tools

Batch conversion from a Berkeley system to a Cyrus system can be done in two different ways:

Extract and copy
: Create the complete, empty mailbox hierarchy on the Cyrus server, then split the Berkeley-style folders into separate RFC 822 messages and copy them into the appropriate mailbox on the Cyrus system. Once all messages have been copied, reconstruct each mailbox hierarchy.

Extract and deliver
: Similar to extract and copy, but instead of copying the extracted mail messages directly into the Cyrus mailboxes, they are piped into the *deliver* program. *deliver* places the messages in the appropriate mailbox and updates the mailbox header and cache files, making it unnecessary to reconstruct each mailbox hierarchy. There is overhead involved in using *deliver* (the state files must be updated with each message) that makes this method slower than extract and copy.

Batch conversions could be done using a utility similar to the *user2cyrus* utility shown in Example 9-6, but there is a drawback: it's nearly 75% slower in real time than an extract and copy. If you have more than 1,000 users to convert, you should probably rule it out.

The extract and copy method is the most direct and requires the least amount of downtime, making it the best option for large-scale migrations. It saves time by avoiding use of the protocol and a third-party delivery agent and by allowing some of the work (such as creating the mailbox hierarchies) to be done before

taking the system down. Because this method directly manipulates the mailbox database, it is also a good opportunity to learn the mechanics of the Cyrus system. For those reasons, we will examine the extract and copy approach.

Procedure

The starting point for the conversion is a text file listing usernames of the accounts to be converted. The procedure for conversion from a Unix to a Cyrus system consists of the following steps:

1. Shut down *imapd* and *sendmail*.
2. Create an IMAP account for each user on the list.
3. Create an empty Cyrus mailbox on the new system for each mail folder that resides on the old system.
4. Transfer messages from the old incoming mailbox to the Cyrus incoming mailbox.
5. Transfer messages from the old mail folders into the new mailboxes on the Cyrus server.
6. Reconstruct the Cyrus *mailboxes* file.
7. Restart *imapd* and *sendmail*.

In the rest of the section, we will walk through an actual conversion, using the tools provided in Appendix A. As we go along, we'll also list the output and results of each step. For simplicity's sake, our example will move only one user. Note that the procedure is the same whether you have a few accounts or thousands (we have run this identical procedure on 20,000 accounts in one batch).

Step 1: Shut down imapd and sendmail

Chaos can result if mail is delivered to the Cyrus system while the conversion is underway. Be sure to turn off both *imapd* and *sendmail* before proceeding any further.

Step 2: Create the new accounts

Creating IMAP accounts in advance can save up to several hours of downtime.

First, save the list of usernames you're going to convert in a text file. We saved the usernames we're converting in a text file called *users.txt*, and that's the file used throughout the examples in this chapter. Once you have your text file, create a Cyrus IMAP account for each user in *users.txt*. This involves two steps:

- *Set up authentication for your users*. If you use Unix authentication, add the users (or NIS/NIS+ usernames or netgroups) to the local password file. If you

plan to run your Cyrus system as a black box, assign each user a null shell, such as */bin/false*, to prevent them from logging in and snooping around. If your site authenticates using a SASL mechanism, such as Kerberos or CRAM-MD5, it's not necessary to add the users to the password file. If you use an alternative to Unix authentication, then Unix accounts are not required for your users. Chapter 8 has more information on how to set up authentication for your users.

- *Create an IMAP account for each user*. A handy Tcl script for creating new IMAP accounts, *addusers*, is given in Example 9-1. This step can be performed while both the new and the old systems are up and running:

```
# addusers users.txt
```

The result of Step 2 is that each user on the list now has a top-level mailbox (INBOX) with the default ACL or, in other words, an account on the Cyrus system.

Step 3: Create Cyrus mail folders

The next step, creating empty mailboxes that correspond to folders, is a bit more involved. To understand what the goal is here, see Figure 9-1. The figure shows how *johndoe*'s incoming mailbox and mail folders on a Berkeley mail format system map to Cyrus mailboxes.

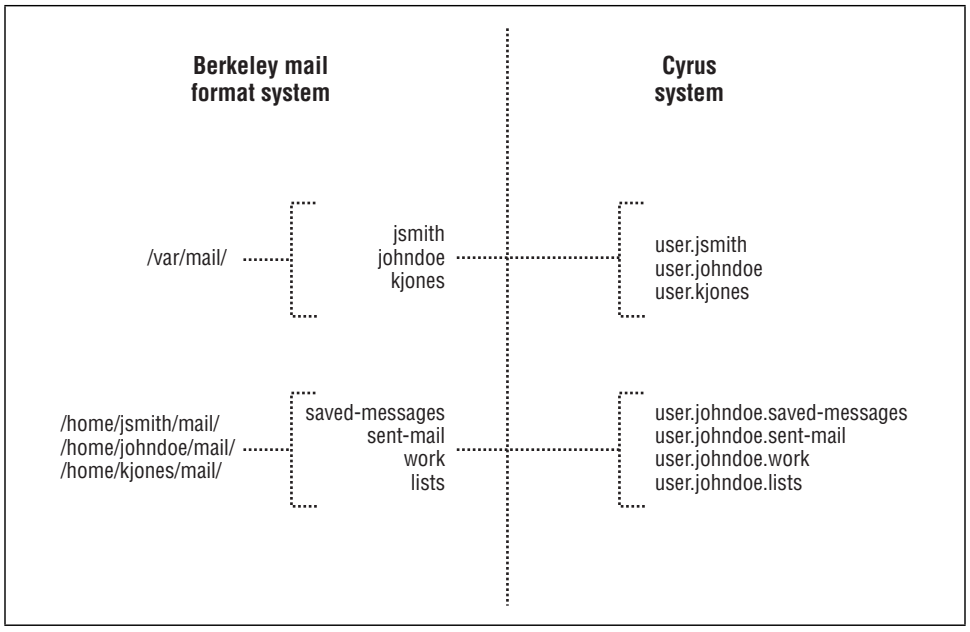

Figure 9-1. Mapping of johndoe's Berkeley mail folders to Cyrus mailboxes

As the figure shows, *johndoe*'s incoming mailbox and each of his folders must be converted into a Cyrus mailbox. The Cyrus mailbox format is different from Berkeley format in several ways. For one, it's hierarchical—the incoming mailbox is the root of the tree, and the folders and sub-folders are branches. The other difference is that Berkeley format mail folders contain many mail messages within a single file, whereas Cyrus folders store each message in a separate file. Another difference is that the Cyrus account holder does not own his own mail folders—they're owned by the Cyrus system. The details of the Cyrus mailbox format were covered in Chapter 6.

Because Cyrus mail format is so different from Berkeley format, some preparation work must be done before you actually create the empty mailboxes. First, you have to find the pathnames of all the Berkeley format folders that belong to the user and map them into mailbox names in the Cyrus namespace. Using *johndoe* as an example, we could look up his home directory in the password file and, knowing that, find all the folders in his *~/Mail* directory, then translate those folder names into Cyrus mailbox names.

The mapping from Berkeley folder names into the Cyrus namespace is accomplished using the Perl script, *bsd2cyrus*. *bsd2cyrus* does nothing more than formulate a list of Cyrus mailbox names—the output of *bsd2cyrus* is to be fed into another script later on to create the actual mailboxes.

To run the script and redirect its output to a file, *mailboxes.txt* (for later use in the other conversion scripts), use the command:

```
# bsd2cyrus users.txt > mailboxes.txt
```

Save the output of *bsd2cyrus* in a text file and keep it for later reference. Other conversion tools will rely on it for input, and it's the only correlation you'll have between Berkeley folder pathnames and Cyrus mailbox names.

There are several points worth noting about the *bsd2cyrus* script:

- The script expects that each user's mail folders are stored in a subdirectory off the user's home directory on the Berkeley system. In the examples in this chapter, that subdirectory is *~/mail*. The script also expects that the user's home directories will be mounted temporarily on the Cyrus machine for the duration of the conversion.
- There is always the possibility that folders on the Berkeley system may have content that is not RFC 822 compliant (e.g., they might be compressed files,

executables, or subdirectories). Such folders are ignored by *bsd2cyrus*. You always have the option of going back and handling the exceptions.

- You may recall from Chapter 6 that Cyrus does not support special characters in mailbox names. *bsd2cyrus* converts all characters except alphabetic characters, digits, underscore, and dash to their ASCII representation preceded by an underscore. For example, user *diannam*'s Berkeley folder name *mail&news* would become *user.diannam.mail_046news* on the Cyrus system (046 is the ASCII representation of the character "&").

The output of the *bsd2cyrus* script will be used as input into two other scripts, which perform the following:

- Create the new, empty folders on the Cyrus system
- Convert the contents of the old BSD mail folders to Cyrus format and insert the mail into the new mailboxes

Let's look at what the *bsd2cyrus* script does. Our user, *diannam*, has several BSD mail folders on the old system:

```
% ls -aF
.               binaryfile*     how spacy       s.me
..              compressed.Z    s!me            same
.hidden         directory/      s&me            saved
Trash           gzip.gz         s*me            sent
```

After running the script, take a look at its output:

```
# cat mailboxes.txt
diannam:user.diannam.Trash:/home/jove/stu/dl0020/mail/Trash
diannam:user.diannam.s_041me:/home/jove/stu/dl0020/mail/s!me
diannam:user.diannam.s_046me:/home/jove/stu/dl0020/mail/s&me
diannam:user.diannam.s_052me:/home/jove/stu/dl0020/mail/s*me
diannam:user.diannam.s_056me:/home/jove/stu/dl0020/mail/s.me
diannam:user.diannam.same:/home/jove/stu/dl0020/mail/same
diannam:user.diannam.saved:/home/jove/stu/dl0020/mail/saved
diannam:user.diannam.sent:/home/jove/stu/dl0020/mail/sent
```

Note that the script is designed to ignore certain files, such as hidden files, directories, binary files, and empty files—that is why some files in the directory are not found in *mailboxes.txt*. As you can see in the output, the script renamed the new folders using the ASCII representation of the special characters.

Next, feed the *bsd2cyrus* output into the *createfolders* script. *createfolders* is a Tcl script that creates empty Cyrus mailboxes. This work must be done with Tcl script, because the command to create a new mailbox on a Cyrus system is a *cyradm* command—you may recall that *cyradm* is an extended Tcl interpreter. The *createfolders* script is shown in Example A-2 in Appendix A.

Here's the output of *createfolders* when we run it on the *mailboxes.txt* file that we saved earlier:

```
# createfolders mailboxes.txt
Connected to IMAP server. Authenticating...
Authentication successful.
Created mailbox user.diannam.sent-mail
Created mailbox user.diannam.saved-messages
Created mailbox user.diannam.same
Created mailbox user.diannam.s_041me
Created mailbox user.diannam.s_052me
Created mailbox user.diannam.Trash
Created mailbox user.diannam.s_046me
Created mailbox user.diannam.s_056me
```

Step 4: Transfer messages from old inbox to new inbox

Now that the framework of mailbox hierarchies is in place, the next step is to populate the top-level mailboxes with messages from the Berkeley inbox on the old system. The *inboxfer* script copies mail from the Berkeley inbox into the Cyrus INBOX. The script is shown in Example A-3. It takes the filename of your list of users (*users.txt* in our examples) as input. The usage is:

```
# inboxfer users.txt
```

Step 5: Transfer messages from old folders to Cyrus folders

Once the Berkeley inboxes have been migrated to the Cyrus system, we turn to the Berkeley-format folders. The script in Example A-5 copies the content of Berkeley-format folders on the old system, converts it to Cyrus-friendly format, and copies it into the appropriate empty Cyrus folder on the Cyrus system. The usage of the *folderxfer* script is:

```
$ folderxfer mailboxes.txt
```

Again, you use the *mailboxes.txt* file that we created with the *bsd2cyrus* script as input.

Step 6: Reconstruct the new mailboxes

The final step once everything has been copied over is to reconstruct each user's mailbox hierarchy. *reconstruct* can only be run as the *cyrus* user, so before running the *reconstruct* command, you must change the ownership of all mailboxes to user *cyrus* and group *mail*:

```
# chown -R cyrus:mail /var/spool/imap/user
```

If you have a large number of users, the *chown* command may take an hour or so to complete. Once the permissions have been changed, run the *batchreconstruct* script shown in Example A-6. The usage is:

```
$ batchreconstruct users.txt
```

Step 7: Restart imapd and sendmail

At this point, all mail has been moved to the Cyrus system. Uncomment the `imapd` entry in */etc/inetd.conf*, restart *inetd*, and restart *sendmail*.

Backing Out

Things don't always go as planned, and so we provide you with a way to back out of Cyrus and go back to Berkeley-format mailboxes.

We once began a conversion from the UW server to a commercial IMAP server based on CMU Cyrus. We were concerned with the amount of time it would take to convert our 20,000-user base to the new server. The company's technical staff assured us that, using their conversion tools, it wouldn't take more than a day to have our users up and running on the new server. We made the very expensive mistake of blindly trusting their assurances and based our entire plan on their prediction of our downtime. We warned our users of the downtime, prepared our data, and on the big day, took our old system offline and started the conversion scripts provided to us by the developers at the company. Twelve hours later, after only 265 users had been converted, we realized we were in deep water and decided to back out. Example 9-7 is the script we threw together on the spot and used to grab the converted mail and put it back where it came from.

Example 9-7. revert

```
#!/usr/local/bin/perl

$LOCK_EX = 2;
$LOCK_UN = 8;
$formail = "/usr/local/bin/formail";
$base    = "/var/spool/imap/user";
$spool   = "/var/mail";

open (USERLIST, "$ARGV[0]") || die "$!";

while (<USERLIST>) {

    chop;
    print "$_\n";

    opendir(DIR,"$base/$user") || die;
    @files = readdir(DIR);
    closedir DIR;

    open (BSDMAILBOX,">>$spool/$user") || next;

    foreach $file (@files) {

        next if $file eq '..';
        next if $file eq '.';
```

Example 9-7. revert (continued)

```
        flock (BSDMAILBOX,$LOCK_EX);
        if ($file =~ /\.$/) {

             open(FORMAIL,"$formail < $base/$user/$file |")
                 || die;
             while (<FORMAIL>) {
                 $_ =~ s/^M//g;
                 print BSDMAILBOX $_;
             }
             close FORMAIL;
        }
        flock (BSDMAILBOX,$LOCK_UN);
    }
    close BSDMAILBOX;
}
close USERLIST;
```

After our horrendous experience, we attempted to develop our own conversion tools and give it another try. However, the server's value-added features and proprietary twists kept our conversion code from working as expected. We gave up and installed the open source Cyrus server instead. The tools we developed worked like a charm with Cyrus and gave us more appreciation than ever for the fruits of the open source community.

Mail Forwarding and Filtering on a Black Box

The Cyrus server is intended to run as a black box. Because users have no home directories on the Cyrus server, they cannot enable mail forwarding by creating a *.forward* file.

Forwarding

How is forwarding done without *.forward* files? You guessed it—user mail forwarding is from within the *sendmail aliases* database on a Cyrus system. To make maintenance easier, you can keep the user aliases and the system aliases in separate files:

- System aliases should be kept in the standard */etc/mail/aliases* file.
- Users' mail forwarding orders should be stored in another file (or files), defined in your local *sendmail* configuration.

Let's suppose that we will store users' forwarding orders in an alias file that's named */etc/mail/forward*. The format of the */etc/mail/forward* file is the same as the *sendmail aliases* file format. To activate the */etc/mail/forward* file, edit your M4

file (*sendmail* ships with an example, *cyrusproto.mc*, or you may have rolled your own) and add the following line:

```
define('ALIAS_FILE', '/etc/mail/aliases,/etc/mail/forward')
```

Finally, build a new *sendmail.cf* file and restart *sendmail*. If you don't use an M4 macro to build your *sendmail.cf* file, then edit your *sendmail.cf* file and look for the lines:

```
# location of alias file
O AliasFile=/etc/mail/aliases
```

Change the definition of `AliasFile` to:

```
O AliasFile=/etc/mail/aliases,/etc/mail/forward
```

Because users cannot log in and directly edit their forwarding orders, they need to be allowed to edit their forwarding orders via a web form or some other indirect method.

Migrating existing .forward files to aliases

.forward files have to be taken into account during a conversion process from a Berkeley-style mail system to Cyrus. The Perl script in Example 9-8, *fwd2alias*, finds *.forward* files for a list of users and converts the contents of the *.forward* files into an *aliases* file. Run the script on your Berkeley or UW mail system. The *aliases* file that the script produces should be moved to your Cyrus server's */etc/mail* directory and incorporated into your sendmail configuration as described earlier in this section.

Example 9-8. fwd2alias

```
#!/usr/local/bin/perl

open (USERS,"users.txt") || die "$!";
open (OUTFILE,">/tmp/forward") || die "$!";

while (<PR>) {

    chop;
    ($name,$pass,$uid,$gid,$quota,$comment,$gcos,$dir,$shell)
         = getpwnam($_);

    open (FW,"$dir/.forward") || warn "can't open $dir/.forward";
    chop ($address = <FW>);
    print OUTFILE "$_:$address\n";
    close FW;
}
close USERS;
close OUTFILE;
```

.forward support

If you simply must allow use of *.forward* files for one reason or another, then you can mount home directories on the Cyrus server and tweak your *sendmail* configuration to enable support for *.forward* files (*sendmail*'s *cyrusproto.mc* file comes with support for *.forward* files disabled).

To enable *.forward* support, add the users to the */etc/passwd* file, if necessary (for example, if you're using an authentication mechanism that does not require that the users have Unix accounts). *.forward* forwarding will not work unless the users have Unix logins.

Edit your *sendmail* M4 file and find the line:

```
MAILER(cyrus)
```

On the line immediately following, add the statement:

```
define('CYRUS_MAILER_FLAGS','A5@W')
```

The *w* flag tells sendmail that users exist in */etc/passwd* and that it should look for a *~/.forward* in the user's home directory.

Server-Side Mail Filtering with procmail

procmail is a popular Unix mail filtering utility used to perform server-side mail filtering. On Cyrus servers, the mail storage format makes it difficult for individual users to use *procmail* to filter and sort incoming messages into mailboxes. Traditionally, the user invokes *procmail* from their *.forward*, which doesn't exist on a black box server.

We've seen that it is indeed possible to allow users to use *.forward* files with the Cyrus IMAP server. If a user were to invoke *procmail* from a *.forward* file, he would have to use the Cyrus *deliver* program in his *procmail* rules file to deposit mail into his mailbox. In order for that to even be possible, the permissions on the *deliver* program would have to be changed to allow any user to run it (normally, it can only be run by user *cyrus*). Changing permissions in that way puts the integrity of the system at risk—we've mentioned before that the Cyrus system is finicky about ownership and permissions. It also opens a gaping security hole, essentially allowing any user to insert mail into any other user's mailbox by filtering the mail either directly or through an intermediate program.

Obviously, invoking *procmail* out of *.forward* is something to be avoided. Instead, users' mail-filtering rules could be stored centrally in a subdirectory under */var/imap* (the examples in this chapter will be stored under */var/imap/procmail*) and owned by the Cyrus system. To edit his rules, the user would log in to an authenticated CGI that runs as user *cyrus*.

The setup that is described in this section has the following essential features:

- sendmail calls *procmail,* not *deliver*, as a delivery agent.
- *procmail* reads a global *procmail* rules file and calls Cyrus *deliver* from that file to deliver mail.
- The global *procmail* rules file looks to see if the user has a personal rules file and if so, reads the rules in that file and applies them.

Figure 9-2 illustrates the three features. In the figure, the MTA passes the mail message to the delivery agent, *procmail*. *procmail* filters the message through the global rules file. After processing the message, the global rules file checks whether the recipient has a personal rules file and, if he does, the message is filtered through that file. Finally, the message is dropped into the user's mailbox.

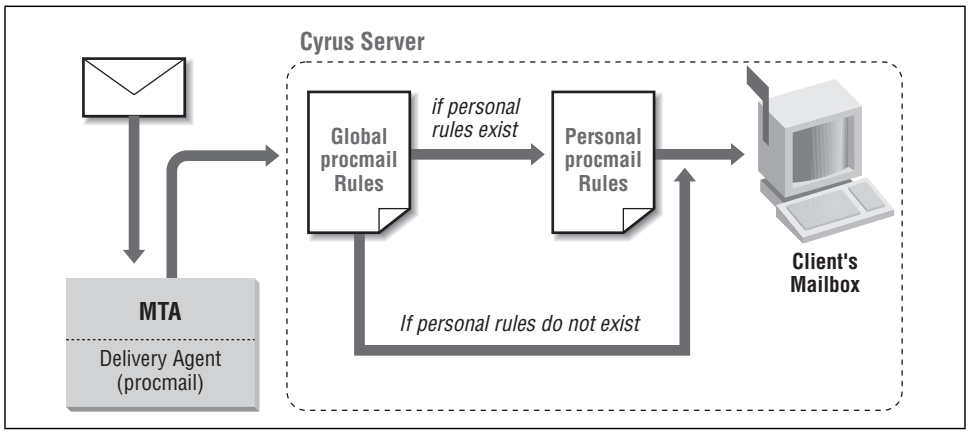

Figure 9-2. Procmail server-side filtering on a Cyrus server

The global procmail rules file

To put *procmail* to work, start by creating a global *procmail* rules file, */var/imap/procmail/procmail.global*. The global rules file contains *procmail* filtering rules that apply to every message that is received by the MTA. The rules in the global file apply to all users. Any mailbox used in a recipe in the global file must already exist—if it does not exist, *deliver* will fail and the message will bounce.

deliver is invoked in the *procmail* rules as:

```
/usr/cyrus/bin/deliver -e -a username -m user.username
```

The *–a* parameter authorizes username (the mailbox owner) to deliver mail to the specified mailbox. Why is the *–a* parameter necessary, you might ask? When *sendmail* invokes *deliver*, it runs the *deliver* process as user *cyrus*. Because *cyrus* owns every mailbox in the Cyrus system, *cyrus* can deliver to any mailbox. In fact, *cyrus* is the only user other than *root* who can deliver to a mailbox. When *deliver* is run

outside of *sendmail*, it runs as the user who invoked it. Because the user is not *cyrus*, she can't deliver into any mailbox. *deliver* has to be told to grant authorization to the user to perform the delivery.

An example global rules file is given in Example 9-9. The first rule in Example 9-9 is used during testing—it saves a backup copy of every incoming message in the user's *backup* mailbox (*user.username.backup*). In order to use this rule, the *backup* mailbox *must* exist for every user on the system. Once testing is complete, the backup rule can be commented out.

The variable CYRUSUSER refers to the username of the mail recipient. CYRUSUSER is set by the MTA before *procmail* is called, so the value of CYRUSUSER is available to the filtering script.

Example 9-9. A Global procmail Rules File

```
# File: procmail.global

PATH=/usr/bin:/usr/local/bin:/usr/cyrus/bin
SHELL=/bin/sh
DELIVER=/usr/cyrus/bin/deliver
SPAM=/dev/null

# Make a backup copy of all incoming mail (comment the next entry
# out once you're finished testing procmail integration)
#:0 ic
#| $DELIVER -e -a $CYRUSUSER -m user.$CYRUSUSER.backup

# Execute CYRUSUSER's personal rules
INCLUDERC=/var/imap/procmail/user/procmail.$CYRUSUSER
```

[The user's procmail rules are included (via the INCLUDERC statement above) and executed here. Rules after this point resume after the user's personal rules have been executed.]

```
# Example recipes

# If the "To:" line doesn't exist, it's SPAM
:0:$CYRUSUSER.lock
* !^To:
| $SPAM

# Get rid of SPAM from a specific email address
:0:$CYRUSUSER.lock
* ^To:.*makemoneyfast@aol.com
| $SPAM

# All the mail that falls through to this point
# will be delivered into the user's INBOX
:0:$CYRUSUSER.lock
| $DELIVER -e -a $CYRUSUSER -m user.$CYRUSUSER
```

Personal procmail rules file

Each user has a personal *procmail* rules file. It's stored under */var/imap/procmail/user/* in the file *procmail.username*. The personal rules file is the file each user edits when he wants to change his filtering rules. If the file exists, the `INCLUDERC` statement in the global *procmail* rules file runs the rules in the personal file. Example 9-10 shows a simple example of a personal *procmail* rules file.

Example 9-10. A Personal procmail Rules File

```
# File: procmail.username

MAILLISTS=user.$CYRUSUSER.Folders.Mailing_Lists
WORK=user.$CYRUSUSER.Folders.Work

# Filter mailing list messages into the appropriate mailbox

:0:$CYRUSUSER.lock
* (^Cc:|^CC:|^To:|^Sender:).*imap@cac.washington.edu
| $DELIVER -e -a $CYRUSUSER -m $MAILLISTS.imap_list

:0:$CYRUSUSER.lock
* (^Cc:|^CC:|^To:|^Sender:).*root@.*
| $DELIVER -e -a $CYRUSUSER -m $WORK
```

Remember that mail cannot be delivered to a mailbox unless the mailbox already exists.

For the rules in the previous example to work, the mailboxes *user.username.Folders.Mailing_Lists.imap_list* and *user.username.Folders.Work* must both exist. *deliver* will fail if it cannot find the mailbox it's told to deliver to.

As with mail forwarding, the personal *procmail* files are private to the Cyrus system. In order to edit the rules, the user needs some sort of indirect method, such as a CGI form, to create or modify his rules.

Setting up the MTA

Putting it all into action involves replacing *deliver* with *procmail* as the local delivery agent. As we saw in the example *procmail* rules files, *deliver* is invoked by *procmail*, instead of directly by the MTA.

Edit */etc/sendmail.cf* and look for the section that defines the Cyrus mailer. It will look something like the lines below. If you've customized your *sendmail* configuration, it might look slightly different, but will begin with the identifier "Mcyrus".

```
Mcyrus,     P=/usr/cyrus/bin/deliver, F=lsDFMnPqA5@, S=10, R=20/40, T=X-Unix,
            U=cyrus:mail,
            A=deliver -e -m $h -- $u
```

Comment out that section and add the following lines below it:

```
Mcyrus,     P=/usr/bin/procmail, F=lsDFMnPqA5@, S=10, R=20/40, T=X-Unix,
            U=cyrus:mail,
            A=procmail -p /var/imap/procmail/procmail.global CYRUSUSER=$u
```

The *–p* argument tells *procmail* to preserve the existing environment (see the *procmail*(1) manual page for details on which environment variables are preserved). The last parameter, */var/imap/procmail/procmail.global*, tells *procmail* to use */var/imap/procmail/procmail.global* to determine its filtering behavior. The last parameter, `CYRUSUSER`, sets the CYRUSUSER variable to *username* and passes the variable to *procmail*. CYRUSUSER is used in the *procmail* rules files, as we saw in Example 9-9 and Example 9-10. Once the *sendmail* configuration changes are in place, restart *sendmail* to make the changes active.

Server-Side Filtering with CMU Sieve

CMU Sieve is a server-side mail filtering language and is described in detail in Chapter 15, *Server-Side Mail Filtering*. Sieve is supported in Cyrus releases newer than 1.6.1b and does not require any special installation. Sieve runs out of *deliver* and requires sendmail 8.9 or higher.

Usenet Integration

Cyrus IMAP supports exporting Usenet news groups as mailboxes. If you run an INN server at your site, the Cyrus distribution provides utilities that allow you to integrate Usenet news into Cyrus. INN is beyond the scope of this book—we assume that, if you attempt to integrate news with Cyrus, you have a working knowledge of the INN server.

Programs for News Integration

Four programs, *collectnews*, *rmnews*, *syncnews*, and *feedcyrus* are provided with the Cyrus distribution for managing and integrating newsgroups with Cyrus. The programs are located in */usr/cyrus/bin*. Each command, except *feedcyrus*, is further documented in section 8 of the online manual pages.

collectnews

: *collectnews* adds a list of news articles to the Cyrus auxiliary databases. When *collectnews* comes across a newsgroup that does not have a corresponding IMAP mailbox, it creates one.

rmnews

rmnews removes a list of canceled, superseded, and expired news articles from the Cyrus auxiliary databases and unlinks the article files.

syncnews

syncnews compares the news active file with the full list of IMAP news mailboxes and removes mailboxes that are not found in the active file. If newsgroups in the active file are found that do not have a corresponding mailbox, then the mailbox is created.

feedcyrus

feedcyrus is a shell script that sends news to the Cyrus IMAP server. The script is created during the installation process if the build was configured to support news.

Configuring News

Integrating Usenet with Cyrus is simply done in two steps:

1. Create a partition for the news spool directory.
2. Set up for maintenance of the Cyrus server's auxiliary databases.

The partition name *news* is reserved specifically for Usenet news. You must name your news partition *news*. Even if you're not integrating news with Cyrus, the partition name *news* cannot be used for any other purpose.

Create the news partition

Select a directory to use as the *news* partition. It should not be the same directory as your INN news spool. In the following example, we selected */var/spool/imap/news* to use as the news partition.

First, create the new partition directory and give ownership to the *cyrus* user:

```
# cd /var/spool/imap
# mkdir news
# chown cyrus imap-news
# chgrp mail imap-news
# chmod 750 imap-news
```

Edit your */etc/imapd.conf* and set the `newsspool` and `partition-news` options as shown in the lines below. `newsspool` is the pathname of your INN news spool directory. `newsprefix` is optional—if you want the name of the news group to

appear with a prefix (e.g., "news.comp.mail" instead of "comp.mail"), then set it to your preferred prefix, followed by a ".".

```
partition-news: /var/spool/imap/news
newsprefix: news.
newsspool: /var/spool/news/articles
```

Set up auxiliary databases

The basic setup for news involves granting write access to *cyrus* on the news spool and setting up *cron* entries to feed news to the Cyrus server and synchronize the newsgroups with Cyrus mailboxes.

The Cyrus utilities run as the *cyrus* user and will write to the news spool directory. To make the news spool directory writable by user *cyrus*, add the *cyrus* user to the *news* group and make the news spool directory group writable. In */etc/group*, the line defining the *news* group should appear as follows:

```
news::13:cyrus
```

Change permissions on the news spool to allow group write access:

```
# chmod -R g+w  /var/spool/news
```

Update the *newsfeeds* file by adding the following line:

```
collectnews!:*:Tf,WO:collectnews
```

Set up *cron* jobs to maintain the auxiliary databases. The jobs should run as *cyrus*, so as the *cyrus* user, add the *cron* jobs to *cyrus*'s *crontab*. *feedcyrus* should be run every ten minutes. *syncnews*, *collectnews*, and *rmnews* should be run once a day. If necessary, replace */var/news/active* with the pathname of your news active file. The crontab entries are as follows:

```
10,20,30,40,50 * * * * /usr/bin/feedcyrus
0 2 * * * /usr/cyrus/bin/syncnews /var/news/active > /dev/null 2>&1
10 * * * * /usr/cyrus/bin/collectnews
20 2 * * * /usr/cyrus/bin/rmnews </home/news/lib/expire-these.files
```

Troubleshooting

This section describes some problems commonly encountered on Cyrus systems, tells you how to diagnose them, and gives you a fix for one.

Testing the Server

From any account, telnet to the IMAP port on your Cyrus server and issue the IMAP NOOP command:

```
% telnet localhost imap
Trying 127.0.0.1...
```

```
Connected to localhost.
Escape character is '^]'.
* OK venus Cyrus IMAP4 v1.5.14 server ready
. noop
. OK Completed
```

If the server returns the message *OK Completed*, then the server is up and responding. If it returns anything other than *OK Completed*, then there is a problem—check the following:

1. Check that */etc/services* contains an entry for *imap*:

   ```
   imap            143/tcp         imap            # IMAP Server
   ```

2. Check that an entry for *imapd* exists in */etc/inetd.conf* and that *imapd* runs as the *cyrus* user:

   ```
   imap stream tcp nowait cyrus /usr/cyrus/bin/imapd imapd
   ```

3. Check the permissions on the *imapd* executable:

   ```
   $ ls -l /usr/cyrus/bin/imapd
   -rwxr-xr-x   1 cyrus   mail  303576 Apr  1 10:07 /usr/cyrus/bin/imapd*
   ```

4. Check permissions on all directories on the path to *imapd*. The *cyrus* user must be able to traverse that path.

User cannot access mailboxes

If the server is up, then the problem is most likely a client configuration. Check to make sure she used the correct syntax to define her mailboxes.

If the syntax is correct, then the user may be trying to access a mailbox for which she has no permissions. In that case, check the ACL on the mailbox.

User stops receiving mail

If the user is able to log in to the server and read his mail, but complains that he has not been receiving mail, then:

1. Make sure *sendmail* is running.
2. Check to see if the user is over quota.
3. Check to see if the user's quota is in a consistent state.

Users are unable to log in

First, determine whether the problem is authentication-related or if the server is not responding. Check the latest entries in */var/log/imapd.log*. If there are more than a few `badlogin` entries, such as this one:

```
May 13 15:21:54 venus imapd[21934]: badlogin: europa [129.120.220.72] plaintext
announce Incorrect password
May 13 15:22:02 venus last message repeated 1 time
```

then the problem is related to authentication. If it does not appear to be an authentication problem, skip this section and read on.

If you use Unix shadow passwords to authenticate users:

1. (Cyrus IMAP Version 1.5.24 and older) Check to see if the *pwcheck* daemon is running. If not, then start it (*etc/init.d/pwcheck start*).
2. Determine whether the user's password has a space character. Space characters are not supported in all IMAP clients.*

If you authenticate users using Kerberos:

1. Check that the */etc/srvtab* file exists.
2. If the file exists, check that the permissions are 0400 (`-r--------`) and that it is owned by user *cyrus* and group *mail*.
3. Run *klist –srvtab*. If either *rcmd* or *imap* are not there, restart the server.

Adding SSL Support to Cyrus

If sending passwords in cleartext across the wire makes you nervous and you don't have the infrastructure to support Kerberos or CRAM, then you can hack SSL support into IMAP as a quick workaround. To take advantage of SSL, you would of course have to use an IMAP client that supports SSL. Currently, your choices are narrow: Netscape Messenger and Outlook Express/Outlook 98. The entire procedure for adding SSL support to IMAP is covered in Appendix B, *Adding SSL Support to IMAP*.

* Not all IMAP clients quote the password like they should.